MiniClusterTools Documentation

Repo: https://github.com/coyleej/MiniClusterTools

Author: Eleanor Coyle

Operating system and shell

These scripts were developed to work with both the server and desktop versions of **Ubuntu 18.04** and the `bash` shell. If you are using a different operating system or a different version of Ubuntu, you will likely need to make modifications. My goal is to maintain POSIX compliance in all of the shell scripts, but I make no promises.

Documentation notes

This documentation automatically pulls relevant information from a larger, actively updated document. As a result, some internal references point to information that is not included. A few broken links seemed a reasonable compromise to ensure up-to-date documentation.

Contents

Chapter 1

Setup Scripts

1.1 sshd_setup.sh

This setup is automated in the `sshd_setup.sh` file from MiniClusterTools repository.

1. Install `fail2ban` and `openssh-server` with `apt`. `fail2ban` will ban IPs that exceed 5 failed tries in 10 minutes. (Never modify `/etc/fail2ban/jail.conf`! Copy it into `jail.local` and modify that.)
2. We're required to display a banner message (option A) on the servers prior to login. It will be activated in a few steps. For now, we're just creating it.
   (a) Create the login banner file:
   `sudo touch /etc/ssh/sshd_banner`
   (b) Add the warning found in the MiniClusterTools repo.
3. Open `/etc/ssh/sshd_config` and change some `ssh` settings.
   The convention in this file is to have default settings commented out and to only uncomment something if you change the value! Note: We are making an exception to this convention for values the higher ups have explicitly requested.
   (a) Specify that all `ssh` access to the servers must use protocol 2 by adding the following right below the line `#Port 22`. (Completely unnecessary for `openssh 7.6+`, but including anyway.)
   `Protocol 2`
   (b) Change `LoginGraceTime` to `1m`
   This changes the allowable time between typing `ssh <server>` and entering a password.
   (c) Change `PermitRootLogin` to `no`
   (d) Uncomment `StrictModes` to `yes` (default value)
   (e) Change `MaxAuthTries` to `3`
   This allows three password attempts after typing `ssh <server>` before resetting.
   (f) Restrict who is allowed to remotely access the server. Add these lines below `MaxSessions`:

   ```
   DenyUsers root
   DenyGroups root
   AllowGroups users slurm
   ```

   Only groups `users` and `slurm` are able to `ssh` or `sftp`. All (human) users should be placed in group `users` when creating their accounts! See chapter **??** for managing group membership.
   Technically the `Deny...` statements are redundant. `root` is already forbidden from using `ssh` because its password is disabled and locked.

(g) Change `IgnoreUserKnownHosts` to `yes`

(h) Uncomment `PermitEmptyPasswords` to `no` (default value)

(i) Uncomment `X11Forwarding` to `yes` (default value)

(j) Have it print the last login for `$USER` (NOTE: While the convention would suggest that `yes` is the default, someone typo'd. The default is actually `no`.)

`PrintLastLog yes`

(k) Uncomment `PermitUserEnvironment` to `no` (default value)

(l) Uncomment `Compression` to `delayed` (default value)

(m) Change `ClientAliveInterval` to `600`

(n) Change `ClientAliveCountMax` to `1`

(o) Display the banner text in between typing `ssh <user>@<server>` and password entry

`Banner /etc/ssh/sshd_banner`

(p) `UsePrivilegeSeparation sandbox` *is depricated starting from* `openssh 7.5`*!* Privilege separation is now mandatory. Because it is depricated and results in a warning when checking the configuration, we are not adding this line to the config file.

4. Check that `sshd_config` is valid and error-free:

   ```
   sudo sshd -t
   ```

5. Reload the daemon:

   ```
   sudo systemctl restart sshd
   ```

6. The code will remind you to manually add users to the `ssh`-approved group:

   ```
   sudo usermod -a -G <ssh-users> <username>
   ```

7. Test by logging in again.

1.2 login_banner.sh

You must configure `gdm3` as described below. These changes are automated in the `login_banner.sh` file.

1. NVidia and Wayland will not get along if you modify the default `gdm` settings (read: you can't log in and the NVidia drivers get corrupted). Open `/etc/gdm3/custom.conf` and set:

   ```
   WaylandEnable=false
   ```

2. Create the following files and directories:

   ```
   sudo touch /etc/dconf/profile/gdm
   sudo mkdir /etc/dconf/db/gdm.d
   sudo touch /etc/dconf/db/gdm.d/01-banner-message
   ```

3. Open `/etc/dconf/profile/gdm` and add the following:

   ```
   user-db:user
   system-db:gdm
   file-db:/usr/share/gdm/greeter-dconf-defaults
   ```

4. Open `/etc/dconf/db/gdm.d/01-banner-message` and add the following:

   ```
   [org/gnome/login-screen]
   banner-message-enable=true
   banner-message-text='I have read & consent to terms in IS user agreement.'
   ```

5. Reconfigure `gdm3` and `dconf`.

   ```
   sudo dconf update
   sudo dpkg-reconfigure gdm3
   ```

6. Restart your computer for the changes to take effect.

1.3 unattended_upgrades.sh

Ubuntu is set to automatically download and install security updates, but not security upgrades. We will use `unattended-upgrades` to automate upgrades. I recommend that users automate security upgrades and check for other upgrades manually. Automating everything is fine, but `apt` occasionally attempts to install or remove other packages. If you do automate everything and encounter dependency issues, refer to section **??**. (Security upgrades are far less likely to cause problems.)

Automatic upgrades must be set up manually. While there are several ways to do this, we will use `unnattended-upgrades`. It's preinstalled and can be configured in two different ways. Both are automated in the `set_unattended_upgrades.sh` file:

- Run `sudo dpkg-reconfigure unattended-upgrades` and follow the prompts. The whole process should take about a minute. Automate all upgrades if you want but be aware of the warning above. Detailed documentation can be found here.
- Configure it manually by editing `/etc/apt/apt.conf.d/50unattended-upgrades`. The setup for our servers is described below. Additional details can be found at these locations: 1, 2, 3, and 4.

Manual setup:

1. Install `bsd-mailx`. In the installation process, pick "local only" and use "$HOSTNAME" as the mail server host. (Substitute the name of your machine for "$HOSTNAME".)
2. Navigate to `/etc/apt/apt.conf.d/`.
3. Copy `50unattended-upgrades` to `50unattended-upgrades.backup`.

   Open `50unattended-upgrades` and make the following changes:

   (a) Uncomment `Unattended-Upgrade::Mail` and change the address to `<admin_account>`.

   (b) Set it to send you mail only when there are errors. (The default is sending mail every time it updates.) Messages are placed in `/var/mail`. Uncomment the following:
   `Unattended-Upgrade::MailOnlyOnError "true"`

   (c) All other items are left as the defaults.
4. Copy `20auto-upgrades` to `20auto-upgrades.backup`.

   Open `20auto-upgrades` and make sure that it contains the following (time intervals are in days):

   ```
   APT::Periodic::Update-Package-Lists "1";
   APT::Periodic::Download-Upgradeable-Packages "1";
   APT::Periodic::AutocleanInterval "7";
   APT::Periodic::Unattended-Upgrade "1";
   ```

5. Test: `sudo unattended-upgrades --dry-run --debug`
6. If the dry run worked: `sudo rm *.backup`

If things ever go wrong, you may need to check the log files:
`/var/log/unattended-upgrades/unattended-upgrades.log`
`/var/log/apt`

`unattended-upgrades` runs randomly within a twelve hour block to smooth out demand on the mirrors. This is fine for our purposes and does not need modification.

1.4 set_password_policy.sh

1.4.1 Password policy

Passwords must be at least 15 characters long, with at least two upper case letters, two lower case letters, two numbers, and two special characters. They must expire after 60 days and contain at least two characters not in the previous password. Running `set_passwd_policy.sh` will automatically change these settings.

1. Change the password expiration settings. Open `/etc/login.defs` and set these variables:

   ```
   PASS_MAX_DAYS   60
   PASS_WARN_AGE   7
   ```

2. Set the password requirements. Open `/etc/security/pwquality.conf`. Negative values indicate that that number of the thing be present in a new password.

   ```
   difok = 2
   minlen = 15
   dcredit = -2
   ucredit = -2
   lcredit = -2
   ocredit = -2
   minclass = 4
   maxrepeat = 2
   usercheck = 1
   ```

3. Changes to `/etc/login.defs` only affect new users (source). You must also apply these changes to existing users: `sudo chage -M <days> <user>`

   The following code automates these changes (you can confirm changes with `sudo chage -l <user>`):

   ```
   userlist=$(grep "10[0-9][0-9]" /etc/passwd | cut -d ":" -f 1)
   for user in $userlist; do
       sudo chage -M 60 $user
       sudo chage -l $user | grep "Pass.*expire"
   done
   ```

1.4.2 HBSS

We must install HBSS and update the policies. The installation script is called in `set_passwd_policy.sh`. Contact me for the current installation script. If you wish to call it separately, do so with the following:

```
sudo bash <install>.sh -i && /opt/McAfee/cma/bin/cmdagent -c
```

1.5 auto_user_setup.sh

REMOVED in latest commit. (This works on test files. It has not been used for actual setup as of now, so I cannot guarantee full functionality.)

Chapter 2

ML installation scripts

2.1 repo_download_w_some_install.sh

Downloads all of the necessary machine learning repos (pybind11, OpenBLAS, S4, MANTIS, and signac) and installs both MANTIS and Signac. Users must compile the other packages themselves.

2.1.1 MANTIS

Note: MANTIS can (and will if you use the script) be installed prior to S4, but S4 is a required dependency. MANTIS will not work properly unless S4 is installed.

1. Clone MANTIS from github:

   ```
   cd /home/<admin>/Code
   git clone https://github.com/harperes/MANTIS.git
   ```

2. Make sure that the system `pip` is installed. If you skip this bit, then `sudo` won't find `pip3`...

   ```
   sudo apt install python3-pip
   ```

3. Install MANTIS to `/opt`

   ```
   cd MANTIS
   sudo pip3 install .  --target="/opt" --no-deps --no-dependencies
   ```

4. Open `/etc/environment`:

   Prepend `/opt/MANTIS` to the system path.

   Make sure that `/opt` has been added to PYTHONPATH.

5. Create or activate an s4py environment.

6. Test the install. Navigate to the MANTIS `tests` directory and type:

   ```
   python -m unittest
   ```

2.1.2 Signac

Setup is automated in `repo_download_w_some_setup.sh`.

1. Clone Signac from github and check out the develop branch:

   ```
   cd /home/<admin>/Code
   git clone https://bitbucket.org/glotzer/signac.git
   cd signac
   git checkout develop
   ```

2. Make sure that the system `pip` is installed. If you skip this bit, then `sudo` won't find `pip3`...

   ```
   sudo apt install python3-pip
   ```

3. Install MANTIS to `/opt`

   ```
   sudo pip3 install .  --target="/opt" --no-deps --no-dependencies
   ```

4. Open `/etc/environment`:

   Make sure that `/opt` has been added to PYTHONPATH.

2.1.3 S4

`repo_download_w_some_setup.sh` downloads the necessary machine learning repositories (pybind11, OpenBLAS, and S4), but compiling is left to the user.

Please refer to Eric Harper's S4 installation instructions. If you have access to the MANTIS folder, you can direct your browser to `file:///home/<User>/<PathToSyncFolder>/MANTISBIBLE/S4Documentation/html/install.html`. If not, please see the harperes S4 repository on github.

Chapter 3

Monitoring scripts

3.1 check_passwd_expiry.sh

Returns a warning if the user's password expires in 7 or fewer days.

3.2 downtime.py

Used for monitoring downtime. It is a modified version of script created by waleedahmad. The original version can be found on waleedahmad's github page.

3.3 Syncthing

3.3.1 Summarize syncthing usage

The `synthing_usage.sh` script in the MiniClusterTools repository reports the size of all directories in the syncthing folder. It reports on all directories but does *not* message users.

3.3.2 Automated messages about syncthing usage

1. Copy `syncthing-warning` into `/usr/share` on the servers. Add eye-catching ASCII art if desired.
2. Open `/etc/bash.bashrc` and append the following.

```
if [ -f /usr/share/syncthing-warning ]; then
        . /usr/share/syncthing-warning
fi
```

Chapter 4

Files

Descriptions of all files in the `files` directory. Other sections cover the usage of each where relevant.

4.1 banner_text*.txt

The long and short versions of the banner text displayed when logging in. Which message is displayed depends on the login method. See sections 1.1 and 1.2 for details.

4.2 cgroup.conf

A `cgroup.conf` example file in case `install_slurm.sh` fails to properly make it's own.

4.3 s4py.yml

Used to create the `s4py` environment in conda. Contains all the packages required to install our ML tools (chapter 2).

4.4 test_sbatch.sh

Test if slurm is working properly by submitting a simple job with `sbatch`. Contains the `sleep` command to keep it in the queue longer.

Chapter 5

Slurm setup and administration

This setup works on Ubuntu 18.04 and Ubuntu Server 18.04.

SchedMD recommends a separate database server if possible. It may be on the same server as `slurmctld`, but this may impact performance. You should consider optimizing the database performance by mounting the MariaDB or MySQL database directory on a dedicated high-speed file system. Ideally this would be a PCIe SSD disk drive (e.g. Intel SSD P3700 series or Kingston E1000 series), but SSD SAS/SATA will also work. Drives must be qualified for high-volume random small read/write operations, and should be built with the Non-Volatile Memory Express (NVMe) storage interface standard for reliability and performance. A disk size of 200 GB or 400 GB should be sufficient. Consider installing 2 disk drives in a RAID-1 configuration.

The following will be installed in this setup guide:

- MPI : OpenMPI version 2
- Slurm 17.11.2-1
- Authentication and digital signatures: MUNGE
- Database : MariaDB

5.1 Reference materials

This guide was constructed from the following references and my own experiences:
Slurm admin quick-start
Slurm official documentation
`man slurm-wlm-doc` for Slurm 17.11.2-1build1
Slurm man pages and configuration file index
Slurm multi-core support
Slurm download and addons list
Slurm configuration (Niflheim)
Slurm database (Niflheim)
Slurm-gpu github

5.2 Basic Slurm set up

1. If you intend to set up a database on its own high speed drive, mount the drive now.
2. Make sure that OpenMPI is installed. If not, install it with
   `sudo apt install libopenmpi2 libopenmpi-dev openmpi-common openmpi-doc`
3. Download the MiniClusterTools repo if you haven't already. It contains a slurm installation script.
   `git clone https://github.com/coyleej/MiniClusterTools.git`

4. Run `install_slurm.sh`. It automates much of the setup. The following explains what it does.

   ```
   bash install_slurm.sh
   ```

   (a) Create Munge user with `uid` and `gid` of 399. (Can be any *unused* value between SYS_UID_MIN and SYS_UID_MAX, which are defined in `/etc/login.defs`).

   ```
   mungeUID=399
   sudo groupadd -g $mungeUID munge
   sudo useradd -r -u $mungeUID -g $mungeUID munge
   sudo usermod -d /nonexistent munge
   ```

   (b) Make sure the system clock is set to the proper timezone and that your system clock is correct:

   ```
   sudo timedatectl set-timezone America/New_York
   timedatectl
   ```

   (c) Check that `nvidia-driver-430` or newer is installed so that slurm can find the GPUs.

   i. Check that we're using the Ubuntu `graphics-drivers` PPA. If we aren't:
   ```
   sudo add-apt-repository ppa:graphics-drivers/ppa
   sudo apt update
   ```
   ii. Use `apt` to purge anything older than `nvidia-driver-430`.

   iii. Use `apt` to install `nvidia-driver-430` if you purged an older driver.

   (d) Install OpenMPI if it is not presently installed:

   ```
   sudo apt install libopenmpi2 libopenmpi-dev openmpi-common openmpi-doc
   ```

   (e) Install MUNGE, SLURM, MySQL, MariaDB, and cgroup-tools:

   ```
   apt install munge libmunge-dev libpam-slurm slurmd slurmdbd slurm-wlm-doc
   cgroup-tools mariadb-common mariadb-server
   ```

   (f) If the node in question is the control node or the backup control node:

   ```
   sudo apt install slurmctld slurm-wlm slurmdbd
   ```

   Otherwise:

   ```
   sudo apt install slurm-client
   ```

   (g) User prompts will gather some information on GPUs.

   (h) Configure the control node, if applicable.

   i. Make sure that `/var/spool/slurmctld/` and `/var/log/slurm-llnl/` exist. If not, create them with `mkdir`.

   ii. Make sure that slurm is the owner of these directories. If not, use `chown slurm: <dirname>`.

   iii. Make sure that the permissions on these directories are set to `755`. If not, use `chmod`.

   iv. Check that `/var/log/slurm-llnl/slurmctld.log` exists and is owned by slurm. Otherwise, create it using `touch` and `chown`.

   v. Create the Linux default accounting file.
   ```
   sudo touch /var/log/slurm-llnl/slurm_jobacct.log
   sudo chown slurm:  /var/log/slurm-llnl/slurm_jobacct.log
   sudo touch /var/log/slurm-llnl/slurm_jobcomp.log
   sudo chown slurm:  /var/log/slurm-llnl/slurm_jobcomp.log
   ```

   (i) Configure the compute nodes. See this site for further details.

   i. Create the `slurmd` spool directory with the correct ownership.
   ```
   mkdir /var/spool/slurmd
   chown slurm:  /var/spool/slurmd
   chmod 755 /var/spool/slurmd
   ```
   ii. Create the log files:
   ```
   touch /var/log/slurmd.log
   chown slurm:  /var/log/slurmd.log
   ```

iii. Create the pid files (only need `slurmctld.pid` on the control node):
`touch /var/log/slurm-llnl/slurmd.pid /var/log/slurm-llnl/slurmctld.pid`
`chown slurm: /var/log/slurm-llnl/slurmd.pid /var/log/slurm-llnl/slurmctld.pid`

iv. View the physical configuration (sockets, cores, real memory, etc.) of each of the compute nodes with the command `slurmd -C`, and update this information in `slurm.conf`.

v. Set the `State` of the node as UNKNOWN (slurm assigns BUSY or IDLE) or FUTURE.

vi. It may be a good idea to assign weights to the compute nodes. All things being equal, jobs will be allocated the nodes with the lowest weight. The enables prioritization based upon hardware parameters such as GPUs, RAM, CPU clock speed, CPU core number, CPU generation. (more info)

vii. It may be a good idea in the future to uncomment `TmpFS=` in `slurm.conf`. (`/tmp` is the default; can change to e.g. `/scratch`.) You can add `TmpDisk=xxxxx` to each compute node line, where `xxxxx` is the size of the temporary file system.

(j) Create spool directories:

```
mkdir -p /var/spool/slurm/d
mkdir /var/spool/slurm/ctld
chown slurm: /var/spool/slurm /var/spool/slurm/d /var/spool/slurm/ctld
```

(k) Create a `gres.conf` file.
Inside this file, add a line for each GPU available on that node as follows: `Name=gpu Type=<type> File=/dev/nvidia#`. (Confirm numbers with `ls -l /dev/nvidia*`.) See the documentation for more options.

(l) Copy `cgroup.conf.example` into `cgroup.conf` and make the following changes:

i. `ConstrainCores=no`

ii. `ConstrainRAMSpace=yes` (change from no)

iii. You may also want to include `MemSpecLimit` and `ContrainKmemSpace`. (reference material)

(m) Adjust the grub configuration. Open `/etc/default/grub`
Add `cgroup_enable=memory swapaccount=1` to `GRUB_CMDLINE_LINUX` line.
Run `update-grub`.

(n) Check the node configuration as detected by slurm by typing `slurmd -C` into the command line. Adjust the appropriate line of the COMPUTE NODES section of the `slurm.conf` file to match.

(o) Retrieve the configuration files:

i. Determine your version of slurm by typing `dpkg -l | grep slurm`. It should report version 17.11.2-1build1.

ii. Obtain the code directly from the command line with:
`wget https://github.com/SchedMD/slurm/archive/slurm-17-11-2-1.tar.gz`

iii. Extract the files. The example configuration files are in `<unzipped-slurm>/etc/`. Copy all example config files into `/etc/slurm-llnl/`

(p) Copy `slurm.conf.example` to `slurm.conf` and make the following changes (e.g. on thanos):

i. `ClusterName=Marvel`

ii. `ControlMachine=magneto`

iii. `ProctrackType=proctrack/cgroup`

iv. `TaskPlugin=task/cgroup`

v. `InactiveLimit=600`

vi. `NodeName=thanos`

vii. `Nodes=thanos`

viii. `PartitionName=CEM`

ix. Remove `Procs=1` and replace it with `CPUs=128`. (On a multi-core/hyperthreaded system, slurm uses the number of threads as the number of CPUs)

x. Add a RESOURCES section just above COMPUTE NODES with the following: `GresTypes=gpu`.

xi. Also under the RESOURCES section, add `LaunchParameters=send_gids`. This has `slurmctld` look up the user name and group ids instead of the individual nodes and prevents the "couldn't chdir" error. This is the default setting in newer versions of slurm.

xii. In the COMPUTE NODES, add the following to each node containing one or more GPUs. # is the number of available GPUs on that node: `Gres=gpu:#`. Insert this just before `State=UNKNOWN`.

xiii. In the SCHEDULING section, set the default memory per node at 1000 MB. (Slurm's default is ALL, which will not allow multiple jobs simultaneously.)
DefMemPerNode=1000

xiv. Change the location of the slurm PID files to the following:
`SlurmctldPidFile=/var/run/slurm-llnl/slurmctld.pid`
`SlurmdPidFile=/var/run/slurm-llnl/slurmd.pid`

xv. Modify `slurm.conf` so that the nodes can be rebooted while slurm is running. Change the reboot program to `RebootProgram="/sbin/reboot"`.

xvi. Change when a DOWN node will be returned to service. The default (`0`) is that nodes will remain down until the admin manually changes the state. We will change this to `1`, meaning that the nodes will be restored to service if it is reponding, has a valid configuration, and was not manually set as DOWN.
`ReturnToService=1`

xvii. Check that `StateSaveLocation=/var/spool/slurm/ctld`. This directory should already exist, but doublecheck to make sure.

xviii. Check that `FastSchedule=1` and `SchedulerType=sched/backfill` (default settings).

xix. Set the consumable resources (1 and 2): `SelectType=select/cons_res`

xx. You must also select what is allowed as consumable resources. In `slurm.conf`, set `SelectTypeParameters=CR_CPU_Memory`.
NOTE: If you use memory as a consumable resource, you *must* set the `RealMemory` parameter.
NOTE: If CPUs are a consumable resource, Slurm has no notion of sockets, cores, or threads. On single- and multi-core systems, CPU refers to cores. On a multi-core/hyperthread system CPU refers to threads.

xxi. Because both CPUs and Memory are consumable resources, you *must* set `OverSubscribe=NO` to prevent jobs from conflicting with one another. Strange behavior will occur if `OverSubscribe=YES`, as jobs will conflict with one another.

xxii. Configure the partitions in `slurm.conf`, for example:
`PartitionName=xeon8 Nodes=a[070-080] Default=YES DefaultTime=50:00:00 MaxTime=168:00:00 State=UNKNOWN`
In the SCHEDULING section of the `slurm.conf` file, set `EnforcePartLimits=YES`. This will reject jobs that exceed a partition's size and/or time limits when they're submitted.
Things to keep in mind for the future (***not setting these up***):
- Partitions may overlap so that some nodes belong to several partitions.
- Access to partitions is configured in `slurm.conf` using AllowAccounts, AllowGroups, or AllowQos.
- If some partitions (e.g. big memory nodes) should have a higher priority, set this in `slurm.conf` using the multifactor plugin: `PartitionName ... PriorityJobFactor=10 PriorityWeightPartition=1000`

xxiii. By default, slurm propagates all user limits from the submitting node (see `ulimit -a` to the batch jobs. Configure `slurm.conf` so that the locked memory limit isn't propagated by uncommenting and setting as follows:
`PropagateResourceLimitsExcept=MEMLOCK`
(We haven't done the following, but if you have imposed any non-default limits on the login nodes in `/etc/security/limits.conf` or `/etc/security/limits.d/*.conf`, you probably want to prohibit these by setting: `PropagateResourceLimitsExcept=ALL`

See the slurm documentation for available options.)

xxiv. Do NOT modify `#PluginDir`! Doing so causes slurm to crash. Slurm defaults to: `usr/lib/x86_64-linux-gnu/slurm-wlm`

(q) Start `slurmd` and, if applicable, `slurmctld`.

```
sudo systemctl start slurmd
sudo systemctl start slurmctld   # if applicable
```

You will get a warning or error if `slurmd -C` failed and the code autofilled the laptop values.

(r) Removes the extracted folder. The downloaded compressed folder is left untouched.

(s) End of installation script.

5. Check that the `NodeName` line matches the output of `slurmd -C`. If `slurmd -C` fails to execute properly, `install_slurm.sh` autofills with the values for an Oryx Pro.

6. Resolve any errors that popped up when running the installation script.

   (a) If the daemon(s) failed to start, type `systemctl status <daemon>`. If slurm can't find nodes or a machine name, fix the `slurm.conf` and try again.

   (b) If slurm complains that it doesn't have permissions to access a directory, you probably forgot `sudo` when starting slurm.

   (c) If slurm isn't starting because it is missing directories, manually create those directories, set `slurm` as the owner, and try again.

   (d) If slurm claims to be missing any configuration files (`*.conf`), see if it exists in `/etc/slurm-llnl` as `*.conf.example`. If it does, copy it, modify it, and try again. If it doesn't exist, refer to the source code on github for `your version of slurm` and copy it where it needs to go.

   (e) If slurm can't find the GPUs, make sure that the system can see the GPUs and that you have an appropriate Nvidia driver.

   (f) If it's still not working, start slurm manually (section **??**) to see more detailed error messages.

7. *At present the script only handles local setup.*

   (a) `slurm.conf` – Nodes and partitions on remote machines must be added manually. The rest of the file is the same, so all that will be required is copy/pasting the node and partition information. Add `NodeAddr=<IP>` just after `NodeName=<name>` to all of the compute nodes.

   (b) *Copy the proper munge key* into `/etc/munge`, then restart the `munge` and `slurmd` daemons.

8. If you installed slurm with `install_slurm.sh`, `cgroup.conf` will be the same on all nodes and all the `gres.conf` files will be setup appropriately. If you did not use the script, make sure that `cgroup.conf` is the same on all compute nodes and add `gres.conf` files as necessary.

9. Restart the node.

10. Check that munge is setup properly.

    (a) If munge is already running, stop it with `systemctl stop munge`.

    (b) Check that the following files/directories are owned by `munge` instead of `root`: `/etc/munge`, `/usr/bin/munge`, `/usr/sbin/munged`, `/var/lib/munge`, `/var/log/munge`, `/var/run/munge`

    (c) Create a munge key on the control node with `sudo /usr/sbin/create-munge-key`. (Ubuntu may have already done this for you.)

    (d) On the controller, make sure the munge key (`munge.key`) is in `/etc/munge/munge.key` and change the owner to munge.

(e) Copy the key from the control node to all existing compute nodes:
`sudo scp /etc/munge/munge.key admin@compute-node:/home/<admin>/`

(f) On the compute nodes, move the `munge.key` into `/etc/munge`. Make sure that it is owned by `munge` with file permissions `400`.

(g) Make sure that munge is enabled and (re)start it on all machines:
`sudo systemctl start munge`

(h) Check if munge is running by typing `systemctl status munge`.

(i) Test munge:
Generate a credential on stdout:
`munge -n`
Check if a credential can be locally decoded:
`munge -n | unmunge`
Check if a credential can be remotely decoded:
`munge -n | ssh <admin>@<node> unmunge`
Run a quick benchmark:
`remunge`

11. Start slurm. Don't worry about enabling the daemons just yet; that will happen later.

```
sudo systemctl start slurmctld   # Control node
sudo systemctl start slurmd   # Compute nodes
```

12. Test that the job submission is working. The submission command is `sbatch <script-name>`. To check the status of the job, type `squeue`. Output will be written in the same folder as the script. Refer to section **??** for an explanation of the SBATCH directives.

A very basic test script:

```
#!/bin/bash
#SBATCH --job-name="testjob"
#SBATCH --time=2:00:00
echo "running job"
sleep 120
echo "All done! :)"
```

A slightly less basic test script (%x is the job name and %j is the job number):

```
#!/bin/bash
#SBATCH --job-name=example
#SBATCH --cpus-per-task=1
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=10:00
#SBATCH --mem=1000
#SBATCH --partition=CEM
#SBATCH --output=%x.o%j

python temp.py
sleep 120
```

Note: Both of these test scripts contain the `sleep` *command specifically to keep the job "running" for a longer time; it is not necessary for actual jobs.*

13. Stop the slurm daemons:

`systemd: sudo systemctl stop <daemon>`
Manual start: `Ctrl-C`

14. We are using the default Prolog and Epilog scripts. Refer to the documentation if this changes.
15. Restart the node.
16. Start slurm and test the queue to confirm that it can run multiple jobs simultaneously.
17. Enable slurm.

```
sudo systemctl enable slurmctld   # Control node(s)
sudo systemctl enable slurmd   # Compute nodes
```

5.3 Hyperthreading

Check for hyperthreading on your system with `lscpu` or `lscpu | grep "per core"`.

From the slurm documentation on hyperthreading:

> If your nodes are configured with hyperthreading, then a CPU is equivalent to a hyperthread. Otherwise a CPU is equivalent to a core. You can determine if your nodes have more than one thread per core using the command "scontrol show node" and looking at the values of "ThreadsPerCore".
>
> Note that even on systems with hyperthreading enabled, the resources will generally be allocated to jobs at the level of a core (see NOTE below). Two different jobs will not share a core except through the use of a partition OverSubscribe configuration parameter. For example, a job requesting resources for three tasks on a node with ThreadsPerCore=2 will be allocated two full cores. Note that Slurm commands contain a multitude of options to control resource allocation with respect to base boards, sockets, cores and threads.
>
> (NOTE: An exception to this would be if the system administrator configured SelectTypeParameters=CR_CPU and each node's CPU count without its socket/core/thread specification. In that case, each thread would be independently scheduled as a CPU. This is not a typical configuration.)

If `SelectTypeParameters` is set to `CR_CPU` or `CP_CPU_Memory`, slurm will treat each thread as a CPU and completely disregard which core a thread is on. If it is set to `CR_Core` or `CR_Core_Memory`, slurm can assign multiple threads to a core but will not assign multiple jobs to the same core. If it is set to `CR_ONE_TASK_PER_CORE`, then slurm will only assign one task per core regardless of the number of threads available.

You can also disable hyperthreading in the kernel using `grub`:

1. Create a backup copy of `/etc/default/grub`.
2. Open `/etc/default/grub` and set the maximum number of CPUs the the number of physical cores:

   Find the line that begins with `GRUB_CMDLINE_LINUX_DEFAULT=`. Append `maxcpus=N` after other options like `quiet splash`
3. Save the file, then run `sudo update-grub`
4. Reboot, then check for hyperthreading with `lscpu | grep "per core"`.

We need to BENCHMARK to decide the best option. Benchmarking tests:

1. Use `CR_CPU_Memory` with hyperthreading enabled.
2. Use `CR_Core_Memory` with hyperthreading enabled.
3. Use `CR_ONE_TASK_PER_CORE` with hyperthreading enabled.
4. Use `CR_CPU_Memory` with hyperthreading disabled in the kernel.

5.4 Sharing StateSaveLocation

If you have two controllers (primary and backup) and the system was set up with NFS instead of Lustre with each controller as an NFS host, some `rsync` and `ssh` key trickery is required to make both controllers share the state save information. Each machine must check the other should one of them go offline.

(The choice of NFS was a good idea with the initial cluster. We did not originally expect to have two controllers. Not switching to Lustre as long as this kluge works.)

1. Use `sudo -i` to run as `root`.

2. Open `root`'s crontab on both machines with `crontab -e`.

   Add the following lines to copy to the other machine once an hour (for testing, ultimately more frequently). (Slurm by default writes this state every 5 seconds.) Note: you must use the admin account for `rsync`. Running `rsync` from the root crontab with `sudo [-i] -u <admin> rsync` will be rejected by the remote servers.

   ```
   0    * * * * rsync -rlptDu --chmod="o+rw" /var/spool/slurm/ctld/ \
            /home/<localadmin>/slurm_state/
   14   * * * * chown -R slurm: /home/<localadmin>/slurm_state
   15   * * * * rsync -rlptDu --chmod="o-rw" /home/<localadmin>/slurm_state/* \
            /var/spool/slurm/ctld
   15   * * * * chown -R nebula: /home/<localadmin>/slurm_state
   ```

3. Type `exit` to return to the `admin` account.

4. If `ssh` keys are already set up, skip to step 7. Otherwise, make ssh keys on each machine in the admin account. Modify the key name/location if desired, but *you must leave the passphrase empty.*

   ```
   ssh-keygen [-f ~/.ssh/<custom_name>]
   ```

5. Copy the keys to the admin account on the other controller, then test the key.

   ```
   ssh-copy-id <admin>@<remote>
   ssh -i ~/.ssh/<local_id_rsa> <admin>@<remote>
   ```

6. Test that `rsync` works over `ssh` in the admin account.

   ```
   rsync -a -e "ssh -i ~/.ssh/<local_id_rsa>" "<randomfile>" "<admin>@<remote>:~/"
   ```

7. Modify the admin crontab on each machine to copy to the remote machine once an hour.

   ```
   20 * * * *  rsync -au --fake-super -e "ssh -i ~/.ssh/<local_key>" \
          "/home/<local-admin>/slurm_state" "<admin>@<remoteIP>:/home/<admin>/"
   ```

8. Check `/var/log/syslog` or the log mailed by `rsync` to confirm that it works.

9. Each node needs to copy the state files from the other system *before* starting slurm. I could make this into a daemon that is required before `slurmctld` can start, but I'm going with the following for now.

   NOTE: I've confirmed that this script works in isolation but have yet to properly test takeover on the controllers!

   (a) Open the `~/.bashrc` file on the admin account.
   Append the following text:

   ```
   # Slurmctld start if daemon test condition failed
   # Daemon:  /etc/systemd/system/multi-user.target.wants/slurmctld.service
   if (! systemctl status slurmctld | grep "[Aa]ctive.*[Rr]unning" > /dev/null); then
       deltat=$(( $(date "+%s") - $(date "+%s" -r /var/spool/slurm/ctld/job_state) ))
   ```

```
        if (test $deltat -ge 30); then
            rsync -au --fake-super -e "ssh -i ~/.ssh/<local_key>" \
                  "/home/<local-admin>/slurm_state" "<admin>@<remoteIP>:/home/<admin>/"
            sudo chmod -R 660 /var/spool/slurm/ctld/*
            sudo rsync -rlptDu "~/slurm_state/*" "/var/spool/slurm/ctld/"
            sudo chown -R slurm: /var/spool/slurm/ctld
            sudo chmod -R 600 /var/spool/slurm/ctld/*
        else
            echo "WARNING: Slurmctld not running for unknown reasons!"
        fi
        echo "Starting slurmctld..."
        sudo systemctl start slurmctld
    fi
```

(b) Stop `slurmctld` for testing purposes.

(c) Source `~/.bashrc`.

(d) If you see `Starting slurmctld...` and no error messages, then the script works properly. Confirm that the daemon is running with `systemctl status slurmctld`.

(e) Disable `slurmctld`. The daemon will only start when the admin logs in for the first time, but the slurm state will always be up to date when the system resumes.

10. Open the admin `.bashrc` on both controllers and alias a command that transfers the slurm state immediately prior to issuing the takeover command.

    (a) Create a file called `transfer_slurm_state.sh`.

    Guides to crazy ssh options: 1

    Place the following text into the file:

```
rsync -au --fake-super -e "ssh -i ~/.ssh/<local_key>"
      "/home/<local-admin>/slurm_state" "<admin>@<remoteIP>:/home/<admin>/"

#ssh -i ~/.ssh/id_rsa_magneto -t nebula@nebula 'sudo cp -v thanos_grep_nss.txt /home/coyleej/'

ssh -i ~/.ssh/id_rsa_nebula magneto@magneto 'bash -s' < test.sh

## PUT THINGS HERE!!
'"
```

    (b) Alias the following in `.bash_aliases` (preferred) or `.bashrc`, then source `.bashrc`.

```
alias state_takeover='bash  /Code/MiniClusterTools/transfer_slurm_state.sh; sudo scontrol
takeover'
```

11. Switch from using `cron` to the `systemd` timers to run more frequently.

5.5 Database setup

1. If you followed the basic slurm install instructions in section 5.2, you should have downloaded the MiniClusterTools git repo. If not, do it now.

```
git clone https://github.com/coyleej/MiniClusterTools.git
```

2. Run `slurmdb_initial_setup.sh`. It automates much of the setup:

```
bash slurmdb_initial_setup.sh
```

   Here's what the script does, with some explanation:

(a) Create the log file:
```
touch /var/log/slurmdbd.log
chown slurm:  /var/log/slurmdbd.log
```

(b) Create the pid file:
```
touch /var/run/slurm-llnl/slurmdbd.pid
chown slurm:  /var/run/slurm-llnl/slurmdbd.pid
```

(c) In `slurm.conf`, make the following changes:

i. Uncomment:
```
JobAcctGatherType=jobacct_gather/linux
JobAcctGatherFrequency=30
AccountingStorageType=accounting_storage/slurmdbd
```

ii. Modify:
```
AccountingStorageHost=<IP or domain name>
AccountingStorageLoc=/var/lib/mysql
AccountingStoragePass=/var/run/munge/munge.socket.2   # munge daemon port
AccountingStoragePort=3306
AccountingStorageUser=slurm
```

iii. Add:
```
AccountingStoreJobComment=YES
AccountingStorageEnforce=associations
AccountingStorageTRES=gres/gpu,gres/gpu:gtx1080ti   # by default billing, CPU, energy, and node are tracked
```

(d) Restart `slurmctld`, as required by some of these changes:
```
systemctl restart slurmctld
```

(e) Copy `slurmdbd.conf.example` to `slurmdbd.conf`.

(f) Open `slurmdbd.conf`

i. Change the following lines to the following:
```
DbdAddr=<magnetoIP>
DbdHost=magneto
PidFile=/var/run/slurm-llnl/slurmdbd.pid
```

ii. Modify the following:
```
StorageHost=magneto
StoragePort=3306   # the mysql default port
StoragePass=<password>   # slurm's password in MariaDB StorageLoc=slurm_acct_db
```

iii. Add the following:
```
PurgeEventAfter=12months
PurgeJobAfter=12months
PurgeResvAfter=2months
PurgeStepAfter=2months
PurgeSuspendAfter=1month
PurgeTXNAfter=12months
PurgeUsageAfter=12months
```

(g) Re-read the config files: `scontrol reconfigure`

(h) We need to enable remote access to mariadb. Open `/etc/mysql/my.cnf` (it's symlinked to `/etc/mysql/mariadb.cnf`), and append the following to the end of the file:
```
[mysqld]
skip-networking=0
skip-bind-address
```

(i) Start MariaDB: `systemctl start mariadb`

3. Verify the setup with

   ```
   scontrol show config | grep AccountingStorageHost
   ```

4. Troubleshoot the MariaDB daemon if it didn't start automatically in the script. Follow whatever error messages it gives, then restart the node and try again.

   ```
   sudo systemctl start mariadb
   ```

   If there have been multiple failed connection attempts, you may need to use the following to unblock the host IP:

   ```
   sudo mysqladmin flush-hosts
   ```

5. Set up MariaDB:

   (a) `sudo mysql_secure_installation`

   (b) Set up the MariaDB root user password: Y

   (c) Create root password: [redacted]

   (d) Remove the anonymous user: Y

   (e) Restrict root user access to the local machine: Y

   (f) Remove the test database: Y

   (g) Reload privilege tables: Y

6. Log in to the MariaDB server as the root user and add a slurm user. (MariaDB doesn't actually require the capitalization, but I'm including it to match their documentation.

   (a) Open the database: `sudo mysql`

   (b) Create the database:

   ```
   MariaDB [(none)]> CREATE DATABASE slurm_acct_db;
   ```

   Confirm with:

   ```
   MariaDB [(none)]> SHOW DATABASES;
   ```

   (c) Create a slurm user and grant database access (replace '<pass>' with the value in `slurmdbd.conf`):

   ```
   GRANT ALL ON slurm_acct_db.* TO 'slurm'@'%' IDENTIFIED BY '<pass>' with grant option;
   ```

   Confirm with:

   ```
   MariaDB [(none)]> SELECT user, host, plugin FROM mysql.user;
   MariaDB [(none)]> SHOW GRANTS for slurm@localhost;
   ```

   (d) Review the current setting for MySQL's `innodb_buffer_pool_size` before running the slurmdbd for the first time.

   ```
   MariaDB [(none)]> SHOW VARIABLES LIKE innodb_buffer_pool_size;
   ```

   (e) Consider setting this value large enough to handle the size of the database. This helps when converting large tables over to the new database schema and when purging old records. Setting `innodb_lock_wait_timeout` and `innodb_log_file_size` to larger values than the default is also recommended. Note: The default buffer size is 128M.

   These variables can be changed in one of the following files (not sure which one, but I suspect it's the first one):

   ```
   /etc/mysql/conf.d/mysql.cnf
   /etc/mysql/mariadb.cnf
   /etc/mysql/mariadb.conf.d/*.cnf
   ```

   ```
   [mysqld]
   innodb_buffer_pool_size=256M
   innodb_log_file_size=256M
   innodb_lock_wait_timeout=1800
   ```

To implement this change you must shut down the database and move/remove the log files:

```
sudo systemctl stop mariadb
sudo rm /var/lib/mysql/ib_logfile?
sudo systemctl start mariadb
```

Verify the new buffer setting using the following command in the MariaDB shell:

```
MariaDB [(none)]> SHOW VARIABLES LIKE innodb_buffer_pool_size;
```

This has been left as the default for now (obviously).

(f) Exit MariaDB:

```
MariaDB [(none)]> QUIT;
```

7. Start `slurmdbd`, acting on any issues that may appear:

   ```
   sudo systemctl start slurmdbd
   ```

   One issue I encountered was fixed by manually changing the owner of the database directory, and reinstall mariadb:

   ```
   sudo chown mysql:  /var/lib/mysql
   sudo apt install --reinstall mariadb-common mariadb-client mariadb-server
   ```

   Try setting up the database again.

   If it's still grumpy, install `mysql-server-5.7` with `apt`, then try setting up the database again.

8. Enable `mariadb` and `slurmdbd`.

9. For job accounting to work, the database and accounting tools must be configured as explained in the official documentation. Use `sacctmgr` to create and manage these records.

   Accounting records are maintained based on "associations" consisting of four elements: cluster, account, user names and an optional partition name. All accounting things are lower case. *You must define clusters before you add accounts and you must add accounts before you add users.*

   (a) Add the cluster to the database:

   ```
   sacctmgr add cluster <clustername>
   ```

   (b) Add accounts:

   ```
   sacctmgr add account <account> [Cluster=<clustername>] [parent=<parent>] \
   Description="<description>" Organization=<organization>
   ```

   Omitting `Cluster` will add the account to all clusters. `parent` is only required if the new account is a sub-account of another account.

   (c) Add users:

   ```
   sacctmgr add user <username> [Account=<accounts>] [DefaultAccount=<account>]
   ```

   `Account` can take a single account or a comma separated list. Not specifying `Account` will give the user access to all accounts on the cluster. `DefaultAccount` will set the default account for a user. At least one of the two options is required.

   (d) Commands to view accounting information:

   ```
   sacctmgr list cluster
   sacctmgr list configuration
   sacctmgr list stats
   ```

10. If other nodes than the `slurmdbd` node must be able to connect to the `slurmdbd` service, you must open the firewall to specific hosts. Please see the Slurm_configuration page under the firewall section.

11. Make the following changes in `slurmdbd.conf`:

    May want to set `PrivateData`

12. Currently have no need to set up WCkeys. (Workload characterization keys are an orthogonal way to do accounting against possibly unrelated accounts. This can be useful where users from different accounts are all working on the same project.)

13. QOS includes multifactor job priority and job preemption. View with `sacctmgr`. By default everything is assigned normal. Can create something with higher priority.

14. Job completion logging is redundant if using the accounting infrastructure.

15. Don't set up PAM with the configuration we currently have! As long as users must submit from the node they want to run on, this is counterproductive!! For future use, see this guide.

16. Enable and start all daemons: `mariadb`, `slurmdbd`, `slurmctld`, `slurmd`

17. If you wish to customize `squeue` output, refer to section **??**

5.6 Adjusting configuration files

When changing configuration files (e.g. `slurm.conf`, `cgroup.conf`), the change must be distributed to all nodes before applying the changes. You may also have to restart the slurm daemons.

1. If you modify settings like Epilog, Prolog, SlurmctldLogFile, SlurmdLogFile, etc.), all you need to do is run `scontrol reconfigure` on the control node to force all slurm daemons to re-read the configuration files. The slurm controller (`slurmctld`) forwards the request to all other daemons (e.g. `slurmd`). Running jobs will continue execution.

2. Slurm daemons *must be restarted* if any of these parameters are changed: AccountingStorageEnforce, AuthType, BackupAddr, BackupController, ControlAddr, ControlMach, PluginDir, StateSaveLocation, SlurmctldPort, SlurmdPort.

3. Slurm daemons *must be restarted* if nodes are added to or removed from the cluster.

`ControlMachine` and `ControlAddr` are defunct in newer versions; use `SlurmctldHost` instead.

5.6.1 Adding/removing nodes

When adding/removing nodes, do the following:

1. Stop `slurmctld`

2. Add/remove nodes in `slurm.conf`

3. Restart `slurmd` on all nodes

4. Start `slurmctld`

It is also possible to add nodes to `slurm.conf` with `state=FUTURE`. The nodes will not be seen by slurm commands in this state. Make them available by changing their state in the `slurm.conf` file and update the node state using `scontrol` rather than restarting the `slurmctld` daemon.

5.7 Slurm plugins

Do not change the default Slurm plugin location in `slurm.conf`!
Default: `/usr/lib/x86_64-linux-gnu/slurm-wlm`

5.8 Slurm admin commands

These are the most common admin-specific commands *in addition to the ones listed in section* **??**. (Those commands can be run with `sudo` to affect *any* job.) For details, refer to `man <command>`.

- Setup and rebooting commands

`scontrol takedown`	Orders switch to backup controller
`scontrol reboot [ASAP] [Nodelist]`	Reboots nodes, see documentation
`scontrol shutdown`	Shuts down the slurm daemons
`sacctmgr shutdown`	Shuts down the cluster
`slurmd -C`	Displays the physical configuration of a node when run on that specific node
`scontrol reconfigure`	Makes running daemons re-read configuration files

- Selected management and accounting commands

`sacct [options]`	Display accounting information for slurm jobs
`sacctmgr`	View and modify slurm account info
`sacctmgr add <entity> <specs>`	Add cluster, accounts, users; identical to `create`
`sacctmgr list <entity> [specs]`	Displays information about the specified entity
`sdiag`	Scheduling diagnostic tool
`smd`	Failure management suport tool
`sreport [options] [command]`	Generates reports of job usage and cluster utilization
`sstat`	Display various status information
`sview`	Graphical user interface to view and modify slurm

- Daemon commands: `slurmctld`, `slurmd`, and `slurmdbd` are the master/control, compute, and database daemons, respectively. They may need to be restarted if configuration files are modified (section 5.6).

`systemctld enable <daemon>`	Enable daemons to start on boot; will not start a stopped daemon
`systemctld disable <daemon>`	Disable daemon so that it will not start; will not stop a running daemon
`systemctld start <daemon>`	Starts daemon manually, does not enable the daemon
`systemctld stop <daemon>`	Stops daemon manually, does not disable the daemon
`systemctld status <daemon>`	Reports status of daemon
`<daemon> -Dvvvv`	Manually starts the daemon; "D" runs in the foreground and "v"s (can have 0 to 7 "v"s) indicates desired verbosity

5.9 Reboot and shutdown nodes

5.9.1 Reboot

Nodes may need to be rebooted after firmware or kernel upgrades. Use the `RebootProgram` in `slurm.conf` to reboot nodes as they become idle. Be mindful of slurm downtime behavior (section 5.9.3).

If you are rebooting the primary controller, the switch should be automatic, but you can also instruct the backup controller to take over. The backup controller requests control from the primary and wait for its termination, then switches to controller mode if primary controller cannot be contacted. This speeds up the slurm controller fail-over mechanism and minimizes disruption when the primary control node is down. Note: the primary controller will take the control back at startup. To reboot nodes

```
# Force switch to the backup controller (primary controller only)
scontrol takeover
# All nodes
scontrol reboot [ASAP] [NodeList]                                          # slurm 17.11.2
scontrol reboot [ASAP] [nextstate=<RESUME|DOWN>] [reason=<reason>] [NodeList]  # newer
```

Explanation: `ASAP` prevents initiation of new jobs. Otherwise the system waits until it is idle to reboot and job scheduling is still allowed. The node state will be DRAIN (17.11.2) or REBOOT (newer) until rebooted or the reboot is cancelled. Newer versions of slurm also include `nextstate`, which specifies the state of the node after reboot, and `reason`, which shows users the reason the node is unavailable.

To cancel a reboot, use one of the following

```
scontrol update NodeName=<nodename> State=RESUME   # slurm 17.11.2
scontrol cancel_reboot <nodelist>                  # newer versions, e.g. 18.08
```

5.9.2 Shutdown

Shut down slurm daemons with `scontrol shutdown [options]`, and servers with `sacctmgr shutdown`.

5.9.3 Slurm downtime behavior

Be mindful of your configured SlurmdTimeout and SlurmctldTimeout values. If the Slurm daemons are down for longer than the specified timeout (currently 5 minutes), nodes will be marked DOWN and their jobs killed. Either increase the timeout values during an upgrade or ensure that the compute node slurmd are not down for longer than SlurmdTimeout.

5.10 Backup and restore database

In order to backup the entire database to a different location (for disaster recovery or migration), the following files must be backed up. (source) Make a database mysqldump using this script /root/mysqlbackup (insert the correct root database password for PWD).

```
#!/bin/sh
# MySQL Backup Script for All Databases
HOST=localhost
BACKUPFILE=/root/mysql_dump
USER=root
PWD='**********'
DUMP_ARGS="--opt --flush-logs --quote-names"
DATABASES="--all-databases"
/usr/bin/mysqldump --host=$HOST --user=$USER --password=$PWD $DUMP_ARGS \
     --result-file=$BACKUPFILE $DATABASES
```

Write permission to $BACKUPFILE is required.

Make regular database dumps, for example by a crontab job: `30 7 * * * /root/mysqlbackup`

Restore of a database backup: The database contents must be loaded from the backup. To restore a MySQL database see for example How do I restore a MySQL .dump file?. As user root input the above created backup file:

`mysql -u root -p < /root/mysql_dump`

5.11 Upgrading slurm

Almost every new major release of Slurm (e.g. 16.05.x to 17.02.x) involves changes to the state files with new data structures, new options, etc. Slurm permits upgrades between any two versions whose major release numbers differ by two or less (e.g. 16.05.x or 17.02.x to 17.11.x) without loss of jobs or other state information. State information from older versions will not be recognized and will be discarded, resulting in loss of all running and pending jobs. State files are not recognized when downgrading and will be discarded. Create backup copies of state files before proceeding to later recover the jobs.

`slurmdbd` must be the same or higher major release as `slurmctld`. When changing the version to a higher release number (e.g. from 16.05.x to 17.02.x) *always* upgrade `slurmdbd` first. Database table changes

may be required for the upgrade. If the database contains a large number of entries, `slurmdbd` may require an hour or two to update the database and will be unresponsive during this time.

`slurmctld` must be upgraded before or at the same time as `slurmd` on the compute nodes. It is recommended to update all daemons at the same time.

The libslurm.so version is increased every major release. Packages with slurm integration (e.g. MPI libraries) should be recompiled. Sometimes symlinking old `.so` name(s) to the new one(s) may work, but this is not guaranteed.

If you built your own version of Slurm plugins, they will likely need modification to support a new version of Slurm. It is common for plugins to add new functions and function arguments during major updates. See the RELEASE_NOTES file for details.

The recommended upgrade order is as follows:

1. Shutdown the slurmdbd daemon
2. Dump the Slurm database using `mysqldump` in case of possible failure
3. Increase `innodb_buffer_size` in `my.cnf` to 128M
4. Upgrade the slurmdbd daemon
5. Restart the slurmdbd daemon
6. Increase SlurmdTimeout and SlurmctldTimeout values and `scontrol reconfigure` to take effect
7. Shutdown the slurmctld daemon(s)
8. Shutdown the slurmd daemons on the compute nodes
9. Copy the contents of the configured StateSaveLocation directory in case of possible failure
10. Upgrade the slurmctld and slurmd daemons
11. Restart the slurmd daemons on the compute nodes
12. Restart the slurmctld daemon(s)
13. Validate proper operation
14. Restore original SlurmdTimeout and SlurmctldTimeout, and then `scontrol reconfigure`
15. Destroy backup copies of database and/or state files

Note: It is possible to update the slurmd daemons on a node-by-node basis after the slurmctld daemon(s) are upgraded, but make sure their down time is below the SlurmdTimeout value.