shifter Documentation

Release 18.03.01

Shane Canon and Douglas Jacobsen

Aug 29, 2021

Contents

shifter is a purpose-built tool for adapting concepts from Linux containers to extreme scale High Performance Computing resources. It allows a user to create their own software environment, typically with Docker, then run it at a supercomputing facility.

The core goal of shifter is to increase scientific computing productivity. This is achieved by:

1. Increasing scientist productivity by simplifying software deployment and management; allow your code to be portable!
2. Enabling scientists to share HPC software directly using the Docker framework and Dockerhub community of software.
3. Encouraging repeatable and reproducible science with more durable software environments.
4. Providing software solutions to improve system utilization by optimizing common bottlenecks in software delivery and I/O in-general.
5. Empowering the user - deploy your own software environment

Contents:

CHAPTER 1

Shifter Frequently Asked Questions

Note that this document is a work in progress, if you don't see your question or answer, please contact shifter-hpc@googlegroups.com

1.1 Do the Image Manager and Image Worker processes need to run with root privilege?

No. The image manager doesn't really do any real work on its own, and the image worker uses only user-space tools to construct images (in the default configuration). A trusted machine should be used to run the imagegw to ensure that images are securely imported for your site. In particular, the python and squashfs tools installations should ideally be integrated as part of the system or be installed in trusted locations.

1.2 Can Shifter import Docker images with unicode filenames embedded?

Yes. If you are seeing errors similar to the following in the image gateway log then you'll need to include the provide sitecustomize.py in your python setup or just point the gateway's PYTHONPATH to include it:

```
[2016-08-01 09:41:20,664: ERROR/MainProcess] Task shifter_imagegw.imageworker.
↪dopull[XXXXXXX-omitted-XXXXXXX] raised unexpected: UnicodeDecodeError('ascii', '/
↪path/is/omitted/some\xc3\xa9_unicode', 35, 36, 'ordinal not in range(128)')
Traceback (most recent call last):
  File "/usr/lib64/python2.6/site-packages/shifter_imagegw/imageworker.py", line 304,␣
↪in dopull
      if not pull_image(request,updater=us):
        File "/usr/lib64/python2.6/site-packages/shifter_imagegw/imageworker.py",␣
↪line 161, in pull_image
            dh.extractDockerLayers(expandedpath, dh.get_eldest(), cachedir=cdir)
  File "/usr/lib64/python2.6/site-packages/shifter_imagegw/dockerv2.py", line 524, in␣
↪extractDockerLayers
```

(continues on next page)

(continued from previous page)

```
      tfp.extractall(path=basePath,members=members)
  File "/usr/lib64/python2.6/tarfile.py", line 2036, in extractall
      self.extract(tarinfo, path)
  File "/usr/lib64/python2.6/tarfile.py", line 2073, in extract
      self._extract_member(tarinfo, os.path.join(path, tarinfo.name))
  File "/usr/lib64/python2.6/posixpath.py", line 70, in join
      path += '/' + b
      UnicodeDecodeError: 'ascii' codec can't decode byte 0xc3 in position 35:␣
↪ordinal not in range(128)
```

To fix this, either set PYTHONPATH to include

- /usr/libexec/shifter (RedHat variants)
- /usr/lib/shifter (SLES variants)

In order to get the shifter sitecustomize.py into your PYTHONPATH.

If that isn't appropriate for your site, then you can examine the contents of the sitecustomize.py and prepare your own that does similar.

1.3 Adding additional registries

The locations block of the imagemanager.json file can be used to add additional registeries. Here is an example that adds various public registeries.:

```
"Locations": {
    "index.docker.io": {
        "remotetype": "dockerv2",
        "authentication": "http"
    },
    "nvcr.io": {
        "remotetype": "dockerv2",
        "authentication": "http"
    },
    "quay.io": {
        "remotetype": "dockerv2",
        "authentication": "http"
    }
},
```

Pulling the image is done similar to the method used for docker.:

```
shifterimg pull quay.io/biocontainers/barrnap:0.9--3
```

1.4 Can Shifter use a proxy to access registries

Shifter does support using a proxy. This is controlled through the enviornment variables, http_proxy. This should be set to the appropriate proxy (e.g. https://myproxy.me.org). Domains can be excluded by setting no_proxy to a comma separated list of domains to exclude. A SOCKS proxy can be used for all connections by setting all_proxy to the socks proxy URL.

CHAPTER 2

Shifter Recommended Practices

2.1 Recommended Practices for Container Developers

2.2 Recommended Practices for Users

2.3 Recommended Practices for System Administrators

0. Avoid running privileged processes, as much as possible, in the User Defined Image environments. This is because the user-defined image is prepared externally, and may or may not contain security vulnerabilities fixed or otherwise not present in your environment.

1. Avoid race conditions in shifter startup by pre-creating udiMount and loopMount (paths defined in udiRoot.conf). If, /var/udiMount, if that is configured as your udiMount path, does not exist, shifter will attempt to create it. If a user starts a parallel application without WLM support (not recommended), all copies will attempt to create this directory, which could lead to a possible race. Avoid this with:

```
mkdir /var/udiMount
mkdir /var/loopMount
```

   as part of your node bootup.

2. Pre-load all of the shifter-required kernel modules as part of system boot. For legacy reasons, which will be removed in a future release, shifter can be configured to auto-load certain kernel modules. It is recommended to avoid this and simply pre-load all the kernel modules your instance of shifter might need. Which modules this may be entirely depends on your site kernel configuration.

   An example might be:

```
modprobe loop max_loop=128
modprobe squashfs
modprobe xfs ## (optional, for perNodeCache)
```

3. Ensure there are plenty of loop devices available. I recommend at least 2x more than the expected number of independent shifter instances you plan on allowing per node. How this is configured is dependent upon how your kernel is built, whether the loop device is compiled-in or presented as a kernel module. If the loop device is compiled-in, you'll need to set max_loop=<number> as a boot kernel argument. If it is compiled as a kernel module, you'll need to specify max_loop=<number> to insmod or modprobe when it is loaded.

4. Make your parallel/network filesystems available in user containers! This is done by setting the siteFs variable in udiRoot.conf. Note that any path you add will be done _prior_ to adding user content to the image, and it will obscure user content if there is a conflict anywhere along the path.

   e.g., if you have a parallel filesystem mounted on /home, adding:

   ```
   siteFs=/home:/home
   ```

   will mount your /home into all user containers on the /home mountpoint; however no content the user defined image might have in /home will be accessible.

   It is strongly recommended to mount your parallel filesystems on the same path users are accustomed to if at all possible, (e.g. mount /home on /home, /scratch on /scratch, etc). If this might obscure important user content, for example if you have a parallel filesystem mounted under /usr, then you must change the shifter mount points to avoid damaging user content.

5. Use the pre and post mount hooks to customize content to match your site. When shifter creates the siteFs-specified mount points, it will attempt a simple "mkdir", not the equivalent of "mkdir -p", so, if for example, you have a siteFs like:

   ```
   siteFs=/home:/home;\
          /global/project:/global/project;\
          /global/common:/global/common;
          /var/opt/cray/alps:/var/opt/cray/alps
   ```

   Your sitePreMountHook script might look like:

   ```
   #!/bin/bash
   mkdir global
   mkdir -p var/opt/cray
   ```

   This is because shifter can create home trivially, but the paths require some setup. Note that the script is executed in the udiMount cwd, however it is _not_ chroot'd into it, this allows you to use content from the external environment, however, means you should be very careful to only manipulate the udiMount, in this example, that means doing *mkdir global*, not *mkdir /global*

   You can use the sitePostMountHook script to do any final setup before the user content is added to the image. For example, if your site usually symlinks /global/project to /project and you want that available in the shifter containers, your sitePostMountHook script might look like:

   ```
   #!/bin/bash
   ln -s global/project project
   ```

6. Use the optUdiImage path to add content to user images at runtime. The path specified for optUdiImage in udiRoot.conf will be recursively copied into the image on /opt/udiImage. This is a great way to force content into an out-of-the-way path at runtime, however, since it is copied (instead of bind- mounted, like siteFs), it is intended to be used for performance-sensitive executables. The optional sshd is in this path, and if any of the optional site-specific MPI support is desired, this should be copied to this path as well.

CHAPTER 3

Updating Shifter

3.1 Version 18.03.1

The parameter –no-xattrs is used when creating the squashed file. This option may not be supported on older OSs such as RedHat 6 variants. To disable this behavior, add the following section to your imagemnager.json config file.

'ConverterOptions': {'squashfs': []}

3.2 Version 17.04 to 18.03

Release 18.03 dropped the dependency on celery. This removes the need to run Redis and simplifies the overall installation. Deployments that had relied on this to run in a distributed mode where the workers ran on a different set of nodes from the API server will need a deployment change. It is recommended to run the API service on the node used for the worker. Broker configuration parameter is no longer and there is no longer a need to launch separate celery workers. The API service will launch a number of local threads to process pull and remove requests. The number of threads can be controlled with the "WorkerThreads" parameter in imagemanager.json. In local mode, the API service must have direct acccess to the shared file system. In remote mode, the API service will copy the completed images via scp which requires RSA keys to be configured.

3.3 Version 16.08 to 17.04

Configurations should be backwards compatible. There are new optional parameters that have been added.

imagemanager.json

- New optional parameter (Metrics). True/False parameter to enable basic image metrics.

If private images are pulled, the resulting metadata will be incompatible with previous versions of the shifter runtime. Set an environment variable called DISABLE_ACL_METADATA to a value in order to enable backwards compatibility. Note that this means some access checks in the runtime will not be enforced and a determined user could use

this to access a private image they should not have access to. So this option should only be used as a temporary bridge and sites should inform the users of the potential risks.

3.4 Version 15.12.0 to 16.08.1

udiRoot.conf

- siteFs format changed from space separated to semicolon separated. Now requires that both "to" and "from" locations be specified. Recommend using same in most cases:

```
siteFs=/home:/home;\
       /var/opt/cray/spool:/var/opt/cray/spool:rec
```

  siteFs now also supports recursive bind mounts (as well as a few other less common mount options).

- Recommended Setting: defaultImageType=docker This will allow shifter commands (shifter, shifterimg, Slurm integration) to assume, unless otherwise noted, that images requested are docker images. This enables the user to do:

```
shifter --image=ubuntu:14.04
```

  Instead of:

```
shifter --image=docker:ubuntu:14.04
```

sshd

The sshd is no longer started by default by the Slurm integration. Add "enable_sshd=1" to plugstack.conf if you want it.

CCM mode is no longer enabled by default in the Slurm integration. Add "enable_ccm=1" to plugstack.conf if you want it.

If the sshd is configured to run via the WLM, the sshd runs as the user. If you have an existing udiImage directory that you want to keep using, be sure to update the port in sshd_config and ssh_config to use port 1204 (or some other port that makes sense for your site).

3.5 Image Manager API

We recommend that you install and run the image manager via gunicorn. This is much more scalable than the Flask development server started by the older (now considered for debug-only) imagemngr.py.

To start with gunicorn do:

```
/usr/bin/gunicorn -b 0.0.0.0:5000 --backlog 2048 \
    --access-logfile=/var/log/shifter_imagegw/access.log \
    --log-file=/var/log/shifter_imagegw/error.log \
    shifter_imagegw.api:app
```

Handling Unicode

The workers have been updated to do a better job of efficiently converting Docker images into Shifter's preferred squashfs format. Start the gateway with the included sitecustomize.py in your PYTHONPATH. This is typically installed in the libexec directory (in Redhat), or the lib directory (in SuSE).

e.g.,:

```
export PYTHONPATH=/usr/libexec/shifter
```

CHAPTER 4

Installation Guides

4.1 Installing Shifter on a RHEL/Centos/Scientific Linux 6 System

4.1.1 Building RPMs

First, ensure your build system has all necessary packages installed:

```
yum install epel-release
yum install rpm-build gcc glibc-devel munge libcurl-devel json-c \
    json-c-devel pam-devel munge-devel libtool autoconf automake \
    gcc-c++ python-pip xfsprogs squashfs-tools python-devel
```

Next, if not using a prepared source tarball, generate one from the repo:

```
git clone https://github.com/NERSC/shifter.git
# checkout a particular branch or commit you require
VERSION=$(grep Version: shifter/shifter.spec | awk '{print $2}')
cp -rp shifter "shifter-$VERSION"
tar cf "shifter-$VERSION.tar.gz" "shifter-$VERSION"
```

Build the RPMs from your tarball:

```
rpmbuild -tb "shifter-$VERSION.tar.gz"
```

Note: To build with Slurm support, execute:

```
rpmbuild -tb "shifter-$VERSION.tar.gz" --define "with_slurm /usr"
```

Change `/usr` to the base path Slurm is installed in.

Installing the Image Manager

The image manager system can run on a login node or other service node in your cluster so long as it is running munge using the same munge key as all the compute nodes (and login nodes) and all nodes can access the imagegwapi port (typically 5000) on the image manager system.

Install the needed dependencies and shifter RPMs:

```
yum install epel-release
yum install python python-pip munge json-c squashfs-tools
rpm -i /path/to/rpmbuild/RPMS/x86_64/shifter-imagegw-$VERSION.rpm
```

`shifter-runtime` is optional, but is recommended on the image gateway system

```
rpm -i /path/to/rpmbuild/RPMS/x86_64/shifter-runtime-$VERSION.rpm
```

Configuring the Image Manager

Copy `/etc/shifter/imagemanager.json.example` to `/etc/shifter/imagemanager.json`. At minimum you should check that:

- `MongoDBURI` is correct URL to shifter imagegw mongodb server.
- `CacheDirectory` exists, semi-permanent storage for docker layers.
- `ExpandDirectory` exists, temporary storage for converting images.
- Change `mycluster` under `Platforms` to match your system name, should match the `system` configuration in `udiRoot.conf`.
- Ensure the `imageDir` is set correctly for your system.

The `imageDir` should be a network volume accessible on all nodes in the system. The `CacheDirectory` and `ExpandDirectory` need only be visible on the imagegw system.

Note: See TODO:FutureDocument for more information on imagegw configuration.

Installing the Shifter Runtime

The shifter runtime needs to be installed on the login nodes as well as the compute nodes.

Install the needed dependencies and shifter RPMs:

```
yum install epel-release
yum install json-c munge

rpm -i /path/to/rpmbuild/RPMS/x86_64/shifter-runtime-$VERSION.rpm
```

Configuring the Shifter Runtime

Copy `/etc/shifter/udiRoot.conf.example` to `/etc/shifter/udiRoot.conf` At minimum you need to change:

- set the value for `system` to match the platform name from `imagemanager.json`
- set the URL for imageGateway to match your imagegw machine, no trailing slash

Generate a `passwd` and `group` file for all your shifter users and place in:

- `/etc/shifter/etc_files/passwd`
- `/etc/shifter/etc_files/group`

Often, these can be generated as easily as `getent passwd > /etc/shifter/etc_files/passwd`, however you'll need to setup to match your local user management configuration. The path to these share etc files for all shifter containers can be controlled with the `etcPath` configuration in `udiRoot.conf`. It is recommended that it be on a network volume to ease updating the passwd and group files.

Note: See TODO:FutureDocument for more information on `udiRoot.conf` configuration.

Starting the Image Manager

Ensure that `mongod` is running. If configured to be on the same host as the `imagegw`, do something like:

```
yum install mongodb-server
/etc/init.d/mongod start
```

Note: TODO: Put init scripts into RPM distribution

Without init scripts, do something like:

```
/usr/libexec/shifter/imagegwapi.py > /var/log/imagegwapi.log &
```

Ensure that `CLUSTERNAME` matches the values in `udiRoot.conf` (system) and `imagemanger.json` (platform).

4.2 Installing Shifter on a RHEL/Centos/Scientific Linux 7 System

4.2.1 Building RPMs

First, ensure your build system has all necessary packages installed:

```
yum install epel-release
yum install rpm-build gcc glibc-devel munge libcurl-devel json-c \
    json-c-devel pam-devel munge-devel libtool autoconf automake \
    gcc-c++ python-pip xfsprogs squashfs-tools python-devel
```

Next, if not using a prepared source tarball, generate one from the repo:

```
git clone https://github.com/NERSC/shifter.git
[[ perform any needed git operations to get a particular branch or commit
   you require.  We currently recommend the 16.08.3_2 tag. ]]
VERSION=$(grep Version: shifter/shifter.spec | awk '{print $2}')
cp -rp shifter "shifter-$VERSION"
tar cf "shifter-$VERSION.tar.gz" "shifter-$VERSION"
```

Build the RPMs from your tarball:

```
rpmbuild -tb "shifter-$VERSION.tar.gz"
```

Note about Slurm support To build with Slurm support do:

```
rpmbuild -tb "shifter-$VERSION.tar.gz" --define "with_slurm /usr"
```

Change "/usr" to the base path Slurm is installed in.

4.2.2 Installing the Image Manager

Install the Shifter runtime RPM

yum -y install shifter{,-imagegw}-1*rpm

Create a config, create directories, start Mongo.

cp /etc/shifter/imagemanager.json.example /etc/shifter/imagemanager.json mkdir /images mkdir -p /data/db mongod –smallfiles &

Start Munge

echo "abcdefghijklkmnopqrstuvwxyz0123456" > /etc/munge/munge.key chown munge.munge /etc/munge/munge.key chmod 600 /etc/munge/munge.key runuser -u munge munged

Start the image gateway and a worker for "mycluster"

gunicorn -b 0.0.0.0:5000 –backlog 2048 shifter_imagegw.api:app &

4.2.3 Installing the Runtime

yum -y install shifter{,-runtime}-1*rpm cp /etc/shifter/udiRoot.conf.example etc/shifter/udiRoot.conf sed -i 'sletcPath=.*letcPath=/etc/shifter/shifter_etc_files/|' **/etc/shifter/udiRoot.conf sed -i 's|imagegwapi:5000|localhost:5000|'** /etc/shifter/udiRoot.conf echo "abcdefghijklkmnopqrstuvwxyz0123456" > /etc/munge/munge.key chown munge.munge /etc/munge/munge.key chmod 600 /etc/munge/munge.key runuser -u munge munged

4.3 Installing Shifter on a RHEL/Centos/Scientific Linux 7 PPC64LE System

4.3.1 Building RPMs

First, ensure your build system has all necessary packages installed:

```
yum install epel-release
yum install rpm-build gcc glibc-devel munge libcurl-devel json-c \
    json-c-devel pam-devel munge-devel libtool autoconf automake \
    gcc-c++ python-pip xfsprogs squashfs-tools python-devel
```

Next, if not using a prepared source tarball, generate one from the repo:

```
git clone https://github.com/NERSC/shifter.git
[[ perform any needed git operations to get a particular branch or commit
   you require ]]
VERSION=$(grep Version: shifter/shifter.spec | awk '{print $2}')
cp -rp shifter "shifter-$VERSION"
tar cf "shifter-$VERSION.tar.gz" "shifter-$VERSION"
```

Build the RPMs from your tarball. Building the Slurm integration has not been tested. Building the sshd daemon is not working (yet) due to issues with musl:

```
rpmbuild --define 'dist .el7' --define 'acflags --disable-staticsshd' -tb shifter-16.
↪08.3.tar.gz
```

4.3.2 Installing the Image Manager

The image manager system can run on a login node or other service node in your cluster so long as it is running munge using the same munge key as all the compute nodes (and login nodes) and all nodes can access the imagegwapi port (typically 5000) on the image manager system.

Install the needed dependencies and shifter RPMs:

```
yum install epel-release
yum install python python-pip munge json-c squashfs-tools
rpm -i /path/to/rpmbuild/RPMS/x86_64/shifter-imagegw-$VERSION.rpm
## shifter-runtime is optional, but recommended on the image gateway system
rpm -i /path/to/rpmbuild/RPMS/x86_64/shifter-runtime-$VERSION.rpm
```

4.3.3 Configuring the Image Manager

Copy /etc/shifter/imagemanager.json.example to /etc/shifter/imagemanager.json. At minimum you should check that:

- "MongoDBURI" is correct URL to shifter imagegw mongodb server
- "CacheDirectory" exists, semi-permanent storage for docker layers
- "ExpandDirectory" exists, temporary storage for converting images
- Change "mycluster" under "Platforms" to match your system name, should match the "system" configuration in udiRoot.conf
- Ensure the "imageDir" is set correctly for your system

The imageDir should be a network volume accessible on all nodes in the system. The CacheDirectory and ExpandDirectory need only be visible on the imagegw system.

See TODO:FutureDocument for more information on imagegw configuration.

4.3.4 Installing the Shifter Runtime

The shifter runtime needs to be installed on the login nodes as well as the compute nodes.

Install the needed dependencies and shifter RPMs:

```
yum install epel-release
yum install json-c munge

rpm -i /path/to/rpmbuild/RPMS/x86_64/shifter-runtime-$VERSION.rpm
```

4.3.5 Configuring the Shifter Runtime

Copy /etc/shifter/udiRoot.conf.example to /etc/shifter/udiRoot.conf At minimum you need to change:

- set the value for "system" to match the platform name from imagemanager.json

- set the URL for imageGateway to match your imagegw machine, no trailing slash

Generate a passwd and group file for all your shifter users and place in:

- `/etc/shifter/etc_files/passwd`
- `/etc/shifter/etc_files/group`

Often, these can be generated as easily as `getent passwd > /etc/shifter/etc_files/passwd`, however you'll need to setup to match your local user management configuration. The path to these share etc files for all shifter containers can be controlled with the *etcPath* configuration in udiRoot.conf. It is recommended that it be on a network volume to ease updating the passwd and group files.

See TODO:FutureDocument for more information on udiRoot.conf configuration.

4.3.6 Starting the Image Manager

Ensure that mongod is running, if configured to be on the same host as the imagegw, do something like:

1. `yum install mongodb-server`
2. `/etc/init.d/mongod start`

TODO: put init scripts into RPM distribution Without init scripts, do something like:

`/usr/libexec/shifter/imagegwapi.py > /var/log/imagegwapi.log &`

- Ensure that CLUSTERNAME matches the values in udiRoot.conf (system) and imagemanger.json (platform)

4.4 Installing Shifter Runtime in Cray's CLE 6.0UP01

4.4.1 Setting up Compute node kernel support

1. Build compute node image.
2. Boot compute node.
3. Install appropriate `kernel-source` and `kernel-syms`. For example:

```
zypper install --oldpackage kernel-source-3.12.51-52.31.1_1.0600.9146.67.1
zypper install --oldpackage kernel-syms.12.51-52.31.1_1.0600.9146.67.1
```

4. `rpmbuild -bb /path/to/shifter_cle6_kmod_deps.spec`
5. Put resulting RPM in a repo, pkgcoll, and update compute image.

4.4.2 Configuring udiRoot with custom ansible play

1. Refer to the sample ansible play/role in the shifter distribution under `extra/cle6/ansible`.
2. Make modifications to each of the following files to meet your needs
    - `vars/main.yaml`
    - `templates/udiRoot.conf.j2`
    - `files/premount.sh`
    - `files/postmount.sh`

- vars/cluster.yaml

3. Ensure the tasks/main.yaml is appropriate for your site

Configuration considerations for DataWarp

For DataWarp mount points to be available in shifter containers, you'll need to ensure that all mounts under /var/opt/cray/dws are imported into the container. Unfortunately, these mounts can sometimes come in after the container is setup. Also, it is hard to predict the exact mount point, thus, we use two mount flags:

- rec, enables a recursive mount of the path to pull in all mounts below it
- **slave, propagates any changes from the external environment into the container** slaved bind mounts are not functional in CLE5.2, but work in CLE6

Add the following to siteFs in your udiRoot.conf:

```
/var/opt/cray/dws:/var/opt/cray/dws:rec:slave
```

Or, to the "compute" section of the shifterSiteFsByType variable in shifter ansible role.

Configuration considerations for MPI

To enable Cray-native MPI in shifter containers, there are a few needed changes to the container environment.

1. Need to patch /var/opt/cray/alps and /etc/opt/cray/wlm_detect into the container environment. /etc/opt/cray/wlm_detect is a NEW requirement in CLE6
2. To enable override of libmpi.so for client containers, run extra/prep_cray_mpi_libs.py from the shifter distribution (also installed in the %libexec% path if shifter-runtime RPM is used).

   ```
   mkdir -p /tmp/cray
   cd /path/to/shifter/extra
   python ./prep_cray_mpi_libs.py /tmp/cray
   ```

   You can then either copy /tmp/cray/mpich-<version> to /usr/lib/shifter/opt/mpich-<version> or integrate the RPM in /tmp/cray/RPMS/x86_64/shifter_cray_mpich_<version>....rpm into your compute and login images.
3. Setup a shifter module to setup cray mpich by adding the following to your udiRoot.conf:

   ```
   module_mpich_copyPath = /usr/lib/shifter/opt/mpich-<version>
   module_mpich_siteEnvPrepend = LD_LIBRARY_PATH=/opt/udiImage/modules/mpich/lib64
   module_mpich_siteEnv = SHIFTER_MODULE_MPICH=1
   ```

4. If you wish the mpich module to load by default on every shifter invocation, then add the following to your udiRoot.conf:

   ```
   defaultModules = mpich
   ```

prep_cray_mpi_libs.py copies the shared libraries from the CrayPE cray-mpich and cray-mpich-abi modules (for both PrgEnv-gnu and PrgEnv-intel), into the target path. It then recursively identifies dependencies for those shared libraries and copies those to target/dep. Finally it rewrites the RPATH entry for all the shared libraries to /opt/udiImage/cray/lib64/dep; this allows the target libraries to exist in /opt/udiImage/cray/lib64, and ONLY have that path in LD_LIBRARY_PATH, minimizing search time. Also, since none of the dependency libraries are copies to /opt/udiImage/cray/lib64, but to /opt/udiImage/cray/lib64/dep, and those are accessed via the modified RPATH, there is minimal bleedthrough of the dependency libraries into the container environment. prep_cray_mpi_libs.py requires patchelf.

4.5 Manual installation of Shifter with GPU support

4.5.1 Introduction: GPU support / Native performance for container images

Containers allow applications to be portable accross platforms by packing the application in a complete filesystem that has everything needed for execution: code, libraries, system tools, environment variables, etc. Therefore, a container can ensure that an application always run under the same software environment.

To achieve a degree of portability, shifter (as the container runtime) has to provide the same interfaces and behavior independently of the hardware and target platform the container is executing on. This ensures that containerized CPU-based applications can be seamlessly deployed across platforms. However, this often means that hardware accelerated features that require specialized system drivers that are optimized for specific target system cannot be natively supported without breaking portability.

In order to mantain the portability that images provide while supporting site-optimized system features we implement a solution that relies on drivers that provide ABI compatibility and mount system specific features at the container's startup. In other words, the driver that is used by a container to allow the application to run on a laptop can be swapped at startup by the shifter runtime and instead an ABI compatible version is loaded that is optimized for the infrastructure of the supercomputer, therefore allowing the same container to achieve native performance.

We use this approach and solve the practical details for allowing container portability and native performance on a variety of target systems through command line options that indicate if, for instance, gpu support should be enabled and what gpu devices should be made available to the container image at startup.

4.5.2 Installation

Shifter Runtime

Most often, Shifter's runtime should be installed on the frontend node as well as on the compute nodes. However, it is also possible to install shifter solely on the compute nodes and use shifter on the frontend node through Slurm.

Dependencies

Make sure you have all the required packages installed. For RHEL/CentOS/Scientific Linux systems:

```
yum install epel-release
yum install gcc glibc-devel munge libcurl-devel json-c \
    json-c-devel pam-devel munge-devel libtool autoconf automake \
    gcc-c++ xfsprogs python-devel libcap-devel
```

For Debian/Ubuntu systems:

```
sudo apt-get install unzip libjson-c2 libjson-c-dev libmunge2 libmunge-dev \
                     libcurl4-openssl-dev autoconf automake libtool curl \
                     make xfsprogs python-dev libcap-dev wget
```

Download, configure, build and install

Clone the github repository to obtain the source:

```
git clone https://github.com/NERSC/shifter.git
cd shifter
```

The following environment variables indicate the directories where Shifter's configuration files and images are located:

```
export SHIFTER_SYSCONFDIR=/etc/shifter
export UDIROOT_PREFIX=/opt/shifter/udiRoot
```

Configure and build the runtime:

```
./autogen.sh
./configure --prefix=$UDIROOT_PREFIX          \
            --sysconfdir=$SHIFTER_SYSCONFDIR \
            --with-json-c                     \
            --with-libcurl                    \
            --with-munge                      \
            --with-slurm=/path/to/your/slurm/installation
make -j8
sudo make install
```

Create links to system directories and additional required directories:

```
sudo ln -s $UDIROOT_PREFIX/bin/shifter /usr/bin/shifter
sudo ln -s $UDIROOT_PREFIX/bin/shifterimg /usr/bin/shifterimg
sudo mkdir -p /usr/libexec/shifter
sudo ln -s /opt/shifter/udiRoot/libexec/shifter/mount /usr/libexec/shifter/mount
sudo mkdir -p $SHIFTER_SYSCONFDIR
```

Shifter's runtime configuration parameters

At run time, Shifter takes its configuration options from a file named *udiRoot.conf*. This file must be placed in the directory specified with *–sysconfdir* when running shifter's configure script. For reference, a template with a base configuration named *udiroot.conf.example* can be found inside the sources directory.

To illustrate the configuration process, consider the following parameters that were modified from the template configuration (*udiroot.conf.example*) to support the install on our local cluster named *Greina*:

- **imagePath=/scratch/shifter/images** Absolute path to shifter's images. This path must be readable by root and available from all nodes in the cluster.
- **etcPath=/etc/shifter/shifter_etc_files** Absolute path to the files to be copied into /etc on the containers at startup.
- **allowLocalChroot=1**
- **autoLoadKernelModule=0** Flag to determine if kernel modules will be loaded by Shifter if required. This is limited to loop, squashfs, ext4 (and dependencies). *Recommend value 0* if kernel modules (loop, squashfs, and ext4) are already loaded as part of the node boot process, otherwise use *value 1* to let Shifter load the kernel modules.
- **system=greina** The name of the computer cluster where shifter is deployed. It is **important for this to match the platform name in the json configuration file** for the Image Manager.
- **imageGateway=http://greina9:5000** Space separated URLs for where the Image Gateway can be reached.
- **siteResources=/opt/shifter/site-resources** Absolute path to where site-specific resources will be bind-mounted inside the container to enable features like native MPI or GPU support. This configuration only affects the container. The specified path will be automatically created inside the container. The specified path doesn't need to exist on the host.

Shifter Startup

As mentioned earlier, the Shifter runtime requires the loop, squashfs, ext4 kernel modules loaded. If these modules are not loaded automatically by shifter, they can be loaded manually with:

```
sudo modprobe ext4
sudo modprobe loop
sudo modprobe squashfs
```

4.5.3 Image Gateway

The Image Gateway can run on any node in your cluster. The requirement for the Image Gateway are:

- munge must be running.
- its using the same munge key as all login and compute nodes.
- all nodes can access the imagegwapi address and port as indicated in Shifter's configuration file.

The Image Gateway depends on *MongoDB* server, *squashfs-tools*, virtualenv (to further install all other python dependencies on a virtual environment), and python2.7. It is recommended to also install the dependencies needed by the shifter runtime, as of this time we have not verified which of Shifter's dependencies can be omitted as they are not needed by the image gateway.

For RHEL/CentOS/Scientific Linux systems:

```
yum install mongodb-server squashfs-tools
```

For Debian/Ubuntu systems:

```
sudo apt-get install mongodb squashfs-tools
```

Install *virtualenv* through *pip* for Python:

```
wget https://bootstrap.pypa.io/get-pip.py
sudo python get-pip.py
sudo pip install virtualenv
```

We need to create three directories:

1. Where to install the Image Gateway
2. Where the Image Gateway will cache images
3. Where the Image Gateway will expand images. **Note: For performance reasons this should be located in a local file system** (we experienced a **40x** slowdown of pulling and expanding images when the images were expanded on a Lustre parallel file system!).

```
export IMAGEGW_PATH=/opt/shifter/imagegw
export IMAGES_CACHE_PATH=/scratch/shifter/images/cache
export IMAGES_EXPAND_PATH=/var/shifter/expand
mkdir -p $IMAGEGW_PATH
mkdir -p $IMAGES_CACHE_PATH
mkdir -p $IMAGES_EXPAND_PATH
```

Copy the contents of *shifter-master/imagegw* subdirectory to *$IMAGEGW_PATH*:

```
cp -r imagegw/* $IMAGEGW_PATH
```

Next step is to prepare a python virtualenv in the Image Gateway installation directory. If this directory is owned by root, the virtualenv and the python requirements need to be also installed as root.

Note

- Installing packages in the virtualenv as a regular user using *sudo pip install* will override the virtualenv settings and install the packages into the system's Python environment.
- Creating the virtualenv in a different folder (e.g. your */home* directory), installing the packages and copying the virtualenv folder to the Image Gateway path will make the virtualenv refer to the directory where you created it, causing errors with the workers and configuration parameters.

```
cd $IMAGEGW_PATH
# Install the virtualenv and all python dependencies as root
sudo -i
# Set the interpreter for the virtualenv to be python2.7
virtualenv python-virtualenv --python=/usr/bin/python2.7
source python-virtualenv/bin/activate
# The requirement file should already be here if the imagegw folder has been copied
# from the Shifter sources
pip install -r requirements.txt
deactivate
# If you switched to root, return to your user
exit
```

Clone and extract the rukkal/virtual-cluster repository from Github:

```
wget https://github.com/rukkal/virtual-cluster/archive/master.zip
mv master.zip virtual-cluster-master.zip
unzip virtual-cluster-master.zip
```

Copy the following files from the virtual-cluster installation resources:

```
cd virtual-cluster-master/shared-folder/installation
sudo cp start-imagegw.sh ${IMAGEGW_PATH}/start-imagegw.sh
sudo cp init.d.shifter-imagegw /etc/init.d/shifter-imagegw
```

For configuration parameters, the Image Gateway uses a file named *imagemanager.json*. The configuration file must be located in the directory that was specified in Shifter's *$SHIFTER_SYSCONFDIR* (*–sysconfdir* when running shifter's *configure* script). A base template file named *imagemanager.json.example* can be found inside the sources directory.

As a reference of configuration parameters consider the following entries as they were used when installing in our local cluster (Greina):

- **"CacheDirectory": "/scratch/shifter/images/cache/"**: Absolute path to the images cache. The same you chose when defining **$IMAGES_CACHE_PATH**
- **"ExpandDirectory": "/var/shifter/expand/"**: Absolute path to the images expand directory. The same you chose when defining **$IMAGES_EXPAND_PATH**.
- Under **"Platforms"** entry change **"mycluster"** to the name of your system. This should be the same name you set for system in *udiRoot.conf*.
- **"imageDir": "/scratch/shifter/images"**: This is the last among the fields defined for your platform. It is the absolute path to where shifter can find images. Should be the same as *imagePath* in *udiRoot.conf*.

Save the file to a local copy (e.g. *imagemanager.json.local*, just to have a backup ready for your system) and copy it to the configuration directory:

```
sudo cp imagemanager.json.local $SHIFTER_SYSCONFDIR/imagemanager.json
```

Lastly, open *$IMAGEGW_PATH/start-imagegw.sh* and enter the name of your system in the line.

```
SYSTEMS="mycluster"
```

Start the service MongoDB:

```
sudo systemctl start mongod
```

Start the Shifter Image Gateway:

```
sudo /etc/init.d/shifter-imagegw start
```

4.6 Image Gateway Deployment Options

The Image Gateway is responsible for importing images from a registry or other sources and translating those into a format that can be used by the Shifter Runtime layer. The Image Gateway supports several deployment models. The Image Gateway can run in a local or remote mode. In addition, the worker component which is handles the actual conversion can be run in a distributed mode. We will briefly describe some of the options. These options can be mixed or matched when a single image gateway is being used to support multiple systems.

4.6.1 Local Mode

This is the simplest deployment method and works best when the image gateway can run directly on the target system. The image gateway should be run on a service node, login node, or other system that isn't used to run compute jobs. It must have write access to a file system that is visible to all nodes on the compute cluster. Here is a sample configuration. Notice that Mongo would need to run on the same service node.

```
{
    "DefaultImageLocation": "registry-1.docker.io",
    "DefaultImageFormat": "squashfs",
    "PullUpdateTimeout": 300,
    "ImageExpirationTimeout": "90:00:00:00",
    "MongoDBURI":"mongodb://localhost/",
    "MongoDB":"Shifter",
    "CacheDirectory": "/images/cache/",
    "ExpandDirectory": "/images/expand/",
    "Locations": {
        "registry-1.docker.io": {
            "remotetype": "dockerv2",
            "authentication": "http"
        }
    },
    "Platforms": {
        "mycluster": {
            "mungeSocketPath": "/var/run/munge/munge.socket.2",
            "accesstype": "local",
            "admins": ["root"],
            "local": {
                "imageDir": "/images"
            }
        }
```

(continues on next page)

(continued from previous page)

```
    }
}
```

4.6.2 Remote Mode

In this model, the Image Gateway and worker runs on a system external to the target system. After an image has been converted, it is copied using scp to the target system. This model may be used when multiple systems are being supported. This model may also be used if the entire compute system is inside a firewall since the Image Gateway could run on a system entirely outside the cluster. The deployment must be configured with ssh keys between that can be used between the Image Gateway and a login or service node on the target system. We recommend that the Image Gateway be run as a dedicated non-privileged user (e.g. shifter). A similar account should exist on the target system and an RSA-based key-pair should be configured on the target system with a copy of the private key available to the special account on the Image Gateway. Here is a sample configuration for this deployment model.

```
{
    "DefaultImageLocation": "registry-1.docker.io",
    "DefaultImageFormat": "squashfs",
    "PullUpdateTimeout": 300,
    "ImageExpirationTimeout": "90:00:00:00",
    "MongoDBURI":"mongodb://localhost/",
    "MongoDB":"Shifter",
    "CacheDirectory": "/images/cache/",
    "ExpandDirectory": "/images/expand/",
    "Locations": {
        "registry-1.docker.io": {
            "remotetype": "dockerv2",
            "authentication": "http"
        }
    },
    "Platforms": {
        "mycluster": {
            "mungeSocketPath": "/var/run/munge/munge.socket.2",
            "accesstype": "remote",
            "admins": ["root"],
            "host": [
                "mycluster01"
            ],
            "ssh": {
                "username": "shifter",
                "key": "/home/shifter/.ssh/ssh.key",
                "imageDir": "/images"
            }
        }
    }
}
```

CHAPTER 5

References

5.1 `shifter` command

5.1.1 Synopsis

shifter [options] *command* [command options]

5.1.2 Description

`shifter` command generates or attaches to an existing Shifter container environment and launches a process within that container environment. This is done with minimal overhead to ensure that container creation and process execution are done as quickly as possible in support of High Performance Computing needs.

5.1.3 Options

`-i | --image` Image selection specification

`-V | --volume` Volume bind mount

`-h | --help` This help text

`-v | --verbose` Increased logging output

5.1.4 Image Selection

Shifter identifies the desired image by examining its environment and command line options. In order of precedence, shifter selects image by looking at the following sources:

- `SHIFTER` environment variable containing both image type and image speicifier
- `SHIFTER_IMAGE` and `SHIFTER_IMAGETYPE` environment variables

- `SLURM_SPANK_SHIFTER_IMAGE` and `SLURM_SPANK_SHIFTER_IMAGETYPE` environment variables
- `--image` command line option

Thus, the batch system can set effective defaults for image selection by manipulating the job environemnt, however, the user can always override by specifying the `--image` command line argument.

The format of `--image` or the `SHIFTER` environment variable are the same:

```
imageType:imageSpecifier
```

where `imageType` is typically `docker` but could be other, site-defined types. `imageSpecifier` is somewhat dependent on the `imageType`, however, for `docker`, the image gateway typically assigns the sha256 hash of the image manifest to be the specifier.

Shifter will attempt to see if the global environment already has a Shifter image configured matching the users arguments. If a compatible image is already setup on the system the existing environment will be used to launch the requested process. If not, Shifter will generate a new mount namespace, and setup a new shifter environment. This ensures that multiple Shifter instances can be used simultaneously on the same node. Note that each Shifter instance will consume at least one loop device, thus it is recommended that sites allow for at least two available loop devices per Shifter instance that might be reasonably started on a compute node. At NERSC, we allow up to 128 loop devices per compute node.

5.1.5 User-Specified Volume Mounts

Todo: Add documendation for user-specified volume mounts.

5.2 udiRoot.conf reference

udiRoot.conf is read by shifter and most other related shifter utilities within the udiRoot component. Unless udiRoot is built enabling particular options udiRoot.conf must be owned by root, but readable by all users, or at least all users you want accessing shifter.

5.2.1 Configuration File Format

The file configuration format is a basic key=value, however space seperated strings can be used for multiple options. Multiple lines can be used if the final character on the line is ‘’. Items cannot be quoted to allow spaces within the configuration option.

5.2.2 Configuration File Options

udiMount (required)

Absolute path to where shifter should generate a mount point for its own use. This path to this should be writable by root and should not be in use for other purposes.

Recommended value: /var/udiMount

loopMount (required)

Absolute path to where shifter should mount loop device filesystems. This path should be writable by root and should not be in use for other purposes.

Recommended value: /var/udiLoopMount

imagePath (required)

Absolute path to where shifter can find images. This path should be readable by root. This path should be visible by all nodes in the system. It may be possible to use some kind of rsyncing method to have this path be local on all systems, but that may prove problematic if a user attempts to use an image while it is being rsynced. Recommend using GPFS or lustre or similar.

udiRootPath (required)

Absolute path (can be a symlink) to where current version of udiRoot is installed. This path is used to find all needed shifter-specific utilities (shifter, shifterimg, setupRoot, unsetupRoot, mount, etc).

Recommended value: /opt/shifter/default

sitePreMountHook

Script to be run before bind-mounting the siteFs filesystems. This script needs to be root owned and executable by root. It should create any directories on the path to the mount point, but not the mount point itself (e.g.,

> *mkdir -p global* but not *mkdir -p global/u1* if your siteFs path is /global/u1)

Note that the script is executed within your udiMount directory and so all your paths within the script should be relative to that.

Recommended value: /etc/opt/nersc/udiRoot/premount.sh

sitePostMountHook

Script to be run after bind-mounting the siteFs filesystems. This script need to be root owned and executable by root. It should do any work required after performing the mounts, e.g., generating a symlink.

Note that the script is executed within your udiMount directory and so all your paths within the script should be relative to that.

Recommended value: /etc/opt/nersc/udiRoot/postmount.sh

optUdiImage

Absolute path to the udiImage directory to be bind-mounted onto /opt/udiImage. This is typically pre-built with shifter to include an sshd, but you could add other things if you so desire.

Recommended value: /opt/shifter/udiRoot/default/deps/udiImage

etcPath

Absolute path to the files you want copied into /etc for every container. This path must be root owned (including the files within), and it must contain, at minimum, nsswitch.conf, passwd, group.

Note that any files you put in this path will override whatever the user included in their image.

Recommended value: /opt/shifter/default/etc_files

allowLocalChroot (0 or 1)

shifter can be used to construct a "container" out a of local path instead of a loop device filesystem. This can be useful if you have an unpacked layer you want to examine, or to enable shifter services within an existing path. Setting to 1 will allow this path-specified shifting, 0 will not.

This must be enabled if the "ccm" emulation mode is desired. (ccm emulation is effectively done with *shifter –image=local:/* within the Slurm integration.

autoLoadKernelModule (0 or 1)

Flag to determine if kernel modules can be automatically loaded by shifter if required. This is typically limited to loop, squashfs, ext4 (and its dependencies)

Recommend 0 if you already load loop, squashfs, and ext4 as part of node bootup process.

Recommend 1 if you want to let shifter load them for you.

mountUdiRootWritable (required)

Flag to remount the udiMount VFS read-only after setup. This is typically only needed for debugging, and should usually be set to 1.

Recommended value: 1

maxGroupCount (required)

Maximum number of groups to allow. If the embedded sshd is being used, then this should be set to 31. This is used when preparing the /etc/group file, which is a filtered version of the group file you provide to shifter. The filtering is done because the libc utilities for parsing an /etc/group file are typically more limited than the LDAP counterparts. Since LDAP is not usable within shifter, a filtered group file is used.

Recommended value: 31

modprobePath (required)

Absolute path to known-good modprobe

insmodPath (required)

Absolute path to known-good insmod

cpPath (required)

Absolute path to known-good cp

mvPath (required)

Absolute path to known-good mv

chmodPath (required)

Absolute path to known-good chmod

mkfsXfsPath

Absolute path to known-good mkfs.xfs. This is required for the perNodeCache feature to work.

rootfsType (required)

The filesystem type to use for setting up the shifter VFS layer. This is typically just tmpfs. On cray compute nodes (CLE 5.2 and 6.0), tmpfs will not work, instead use ramfs.

Recommended value: tmpfs

gatewayTimeout (optional)

Time in seconds to wait for the imagegw to respond before failing over to next (or failing).

siteFs

Space seperated list of paths to be automatically bind-mounted into the container. This is typically used to make network filesystems accessible within the container, but could be used to allow certain other facilities, like /var/run or /var/spool/alps to be accessible within the image (depending on your needs).

Do not attempt to bind things under /usr or other common critical paths within containers.

It is OK to perform this under /var or /opt or a novel path that your site maintains (e.g., for NERSC, /global).

siteEnv

Space seperated list of environment variables to automatically set (or add, or replace) when a shifter container is setup.

Example:: siteEnv=SHIFTER_RUNTIME=1

This can be useful if network home directories are mounted into the container and you users want a way to prevent their localized dotfiles from running. (e.g., do not execute if SHIFTER_RUNTIME is set).

siteEnvAppend

Space seperated list of environment variables to automatically append (or add) when a shifter container is setup. This only makes sense for colon seperated environment variables, .e.g, PATH.

Example:: siteEnvAppend=PATH=/opt/udiImage/bin

This can be used if your site patches in a path that you want to appear in the path. Recommend that all binaries are compatible with all containers, i.e., are statically linked, to ensure they work.

siteEnvPrepend

Space seperated list of environment variables to automatically prepend (or add) when a shifter container is setup. This only makes sense for colon seperated environment variables, e.g., PATH.

Example:: siteEnvPrepend=PATH=/opt/udiImage/bin

This can be used if your site patches in a path that you want to appear in the path. Recommend that all binaries are compatible with all containers, i.e., are statically linked, to ensure they work.

siteEnvUnset

Space separated list of environment variables to be unset when a shifter container is setup. This only makes sense for bare environmental variable names.

Example:: siteEnvUnset=LOADEDMODULES _LMFILES_

imageGateway

Space seperated URLs for your imagegw. Used by shifterimg and Slurm batch integration to communicate with the imagegw.

batchType (optional)

Used by batch integration code to pick useful settings. May be deprecated in the future as it is not necessary at this point.

system (required)

Name of your system, e.g., edison or cori. This name must match a configured system in the imagegw. This is primarily used by shifterimg to self-identify which system it is representing.

5.2.3 Shifter Module Options in udiRoot.conf

For each module a site wants to configure, at least one of the *module_** configuration parameters needs to include. Only specify the options that are needed for your module.

Note that modules will be loaded in the order the user specifies. By specifying a custom set of modules, the user will disable whatever modules were specified by the administrator in the defaultModules configuration parameter.

The siteEnv* parameters are evaluated after all modules have been evaluated when performing environmental setup.

defaultModules

comma-separated list of modules that should be loaded by default for every container invocation. The user can override this and provide their own list of modules that are appropriate for their need.

module_<name>_siteEnv

Like siteEnv, allows the site to define the value an environment variable should take when setting up the container environment, but only if the target module is activated. This value will replace anything set up previously either in the external environment, user-specified option, or in the container definition. The global siteEnv can override this.

module_<name>_siteEnvPrepend

Space seperated list of environment variables to automatically append (or add) when a shifter container is setup. This only makes sense for colon seperated environment variables, .e.g, PATH.

Example:: module_<name>_siteEnvAppend=PATH=/opt/udiImage/modules/<name>/bin

This can be used if your site patches in a path that you want to appear in the path. Recommend that all binaries are compatible with all containers, i.e., are statically linked, to ensure they work. The global siteEnvPrepend is applied after this.

module_<name>_siteEnvAppend

Space seperated list of environment variables to automatically append (or add) when a shifter container is setup. This only makes sense for colon seperated environment variables, .e.g, PATH.

Example:: module_<name>_siteEnvAppend=PATH=/opt/udiImage/modules/<name>/bin

This can be used if your site patches in a path that you want to appear in the path. Recommend that all binaries are compatible with all containers, i.e., are statically linked, to ensure they work. The global siteEnvAppend is applied after this.

module_<name>_siteEnvUnset

Space separated list of environment variables to be unset when a shifter container is setup. This only makes sense for bare environmental variable names.

Example:: module_<name>_siteEnvUnset=MPICH_...

module_<name>_conflict

Space separated list of other modules that cannot be concurrently loaded with this module. Attempt to specify multiple modules that conflict will result in shifter termination.

module_<name>_siteFs

Module-specific external paths to be bind mounted into the container. This will allow additional external content, from the external OS or a shared filesystem to be included, optionally, as part of the module. All the warnings and conditions applied to siteFs apply here as well.

module_<name>_copyPath

The directory in the external environment that is to be copied to /opt/udiImage/modules/<name>/

This can include libraries, scripts or other content that needs to be accessed locally in the container.

module_<name>_enabled

By default a module is enabled. Setting enabled = 0 will prevent any container from specifying or using it. This parameter does not need to be specified in most cases.

module_<name>_roothook

Executes the configured script with /bin/sh as root, in the external environment following most of the site container customization, including siteFs setup, copyPath setup, and etc configuration, but before user content is introduced to the container (in the non-overlay version of shifter). This can be used to perform some transformations of the container environment. The current working directory of the executed script is the container environment (be careful to use relative paths from the CWD). Non-zero exit status of the roothook terminates container construction.

module_<name>_userhook

Executes the configured script with /bin/sh as the user after container construction, but immediately before a process is launched in the container environment. This may be useful to perform lightweight evaluations to determine if the module is compatible with the user's container. Non-zero exit status of the userhook will terminate shifter execution before processes are launched into the container. The userhook can write to stdout or stderr to communicate with the user. Note: the userhook is run after the container has already lost access to the external environment. Recommend setting userhook to use a path in /opt/udiImage/modules/<name> if module_<name>_copyPath is used, or use a siteFs path.

5.3 Authentication to the image gateway

The image gateway requires authentication for all interactions. The purpose of the authentication in most cases to ensure the request came from the target platform (e.g., mycluster), and from a known user id on that system. To that end, the HTTP header must include an "authentication" field.

The value of that field must be a munge-encoded message signed with the same munge credential as is configured for the target platform on the imagegw server system. For basic requests, it may be sufficient to simply munge encode an empty message (e.g., "").

The image gateway may, from time to time perform privileged operations as a result of the API calls (e.g., pull a privileged image from DockerHub). In that case the originating request must include a more extensive "authentication" header which includes the munge encrypted credentials (username and password) for the target platform (at least).

5.3.1 Format of the "authentication" HTTP header

- _Non-privileged requests:_ munge encoded empty string
- _Privileged requests:_ munge encoded JSON document including one or more credentials for the remote image resources

JSON document format:

```
{
    "authorized_locations": {
        "default":"username:password",
        "otherloc":"username:password"
    }
}
```

5.3.2 Specifications for authorized locations

The credentials provided to the image gateway are for the remote locations. The identifier in the "authorized_locations" hash must match the locations in the image_manager.json configuration. The only exception to this is that it may be difficult for the client to determine which location a particular request may be routed to, owing to that, and the fact that the client can not know what the "default" remote location for the image manager is, a special "default" location may be specified which the image manager can use for interacting with the default location (if the default loation is selected) so long as a credential for a more specific name for the default loation is not provided.

e.g., if the "default" location for the imagegw is registry-1.docker.io, and a credential is provided by the client for "registry-1.docker.io", then that credential must be used by the imagegw when interacting with registry-1.docker.io; if, however, only a "default" credential is provided, and "registry-1.docker.io" happens to be the default location, then the "default" credential may be used when interacting with registry-1.docker.io.

The client may provide just the needed credential or many different credentials and the image manager is responsible to parse the data structure and determine which credential is most appropriate given the request. This is allowed because it may be challenging for the client to determine which location will be selected by the imagegw, since the configuration of the imagegw may not be accessible by the client.

CHAPTER 6

Advanced Topics

6.1 MPI Support in Shifter: MPICH ABI

MPICH and its many variants agreed in 2014 to retain ABI compatibility to help improve development practices. However, this ABI compatibility also provides a clear path for almost transparently supporting MPI within shifter environment containers.

The basic idea is that the container developer will use a fairly vanilla version of MPICH and dynamically link their application against that. The Shifter-hosting site then configures Shifter to inject their site-specific version of MPICH (perhaps a Cray, Intel, or IBM variant) linked to the interconnect and workload manager driver libraries. The site-specific version of `libmpi.so` then overrides the version in the container, and the application automatically uses it instead of the generic version originally included in the container.

6.1.1 Container Developer Instructions

Here is an example Dockerfile:

```
FROM ubuntu:14.04
RUN apt-get update && apt-get install -y autoconf automake gcc g++ make gfortran
ADD http://www.mpich.org/static/downloads/3.2/mpich-3.2.tar.gz /usr/local/src/
RUN cd /usr/local/src/ && \
    tar xf mpich-3.2.tar.gz && \
    cd mpich-3.2 && \
    ./configure && \
    make && make install && \
    cd /usr/local/src && \
    rm -rf mpich-3.2

ADD helloworld.c /
RUN mkdir /app && mpicc helloworld.c -o /app/hello

ENV PATH=/usr/bin:/bin:/app
```

Going through the above:

1. base from a common distribution, *e.g.*, `ubuntu:14.04`,
2. install compiler tools to get a minimal dev environment.
3. get and install MPICH 3.2
4. add and compile your application
5. setup the environment to easily access your application

To construct the above container, one would do something like:

```
docker build -t dmjacobsen/mpitest:latest .
```

(setting the tag appropriately, of course).

6.1.2 Slurm User Instructions

If the MPICH ABI environment is configured correctly (see below), it should be very easy to run the application. Building from the example above:

```
dmj@cori11:~> shifterimg pull dmjacobsen/mpitest:latest
2016-08-05T01:14:59 Pulling Image: docker:dmjacobsen/mpitest:latest, status: READY

dmj@cori11:~> salloc --image=dmjacobsen/mpitest:latest -N 4 --exclusive
salloc: Granted job allocation 2813140
salloc: Waiting for resource configuration
salloc: Nodes nid0[2256-2259] are ready for job

dmj@nid02256:~> srun shifter hello
hello from 2 of 4 on nid02258
hello from 0 of 4 on nid02256
hello from 1 of 4 on nid02257
hello from 3 of 4 on nid02259

dmj@nid02256:~> srun -n 128 shifter hello
hello from 32 of 128 on nid02257
hello from 46 of 128 on nid02257
hello from 48 of 128 on nid02257
hello from 55 of 128 on nid02257
hello from 57 of 128 on nid02257
...
...
hello from 26 of 128 on nid02256
hello from 27 of 128 on nid02256
hello from 28 of 128 on nid02256
hello from 29 of 128 on nid02256
hello from 30 of 128 on nid02256
hello from 31 of 128 on nid02256

dmj@nid02256:~> exit
salloc: Relinquishing job allocation 2813140
salloc: Job allocation 2813140 has been revoked.

dmj@cori11:~>
```

System Administrator Instructions: Configuring Shifter

The basic plan is to gather the `libmpi.so*` libraries and symlinks and copy them into the container at runtime. This may require some dependencies to also be copied, but hopefully only the most limited set possible. The current recommendation is to copy these libraries into `/opt/udiImage/<type>/lib64`, and all the dependencies to `/opt/udiImage/<type>/lib64/dep`

We then use patchelf to rewrite the rpath of all copied libraries to point to `/opt/udiImage/<type>/lib64/dep`

The source libraries must be prepared ahead of time using one of the helper scripts provided in the extras directory, or a variant of same. As we get access to different types of systems, we will post more helper scripts and system-type-specific instructions.

Finally, we need to force `LD_LIBRARY_PATH` in the container to include `/opt/udiImage/<type>/lib64`

Cray

Run the `prep_cray_mpi_libs.py` script to prepare the libraries:

```
login$ python /path/to/shifterSource/extra/prep_cray_mpi_libs.py /tmp/craylibs
```

Note: In CLE5.2 this should be done on an internal login node; in CLE6 an internal or external login node should work. You'll need to install PatchELF into your `PATH` prior to running.

Next copy `/tmp/craylibs/mpich-<version>` to your Shifter module path (see *Shifter Modules*): *e.g.*, `/usr/lib/shifter/opt/mpich-<version>`.

Finally, a few modifications need to be made to `udiRoot.conf`:

1. add `module_mpich_siteEnvPrepend = LD_LIBRARY_PATH=/opt/udiImage/modules/mpich/lib64`
2. add `module_mpich_copyPath = /usr/lib/shifter/opt/mpich-<version>`
3. add `/var/opt/cray/alps:/var/opt/cray/alps:rec` to `siteFs`
4. if CLE6, add `/etc/opt/cray/wlm_detect:/etc/opt/cray/wlm_detect` to `siteFs`
5. add `defaultModules = mpich` to load `cray-mpich` support by default in all containers

Note: You may need to modify your `sitePreMountHook` script to create `/var/opt/cray` and `/etc/opt/cray` prior the mounts.

Instead of setting up the `module_mpich_copyPath`, you could use `siteFs` to bind-mount the content into the container instead, which may have performance benefits in some environments, *e.g.* set `module_mpich_siteFs = /usr/lib/shifter/modules/mpich:/shifter/mpich`. In that case you'll need to adjust the `module_mpich_siteEnvPrepend` paths, and pre-create the `/shifter` directory using the `sitePreMountHook`.

Other MPICH variants/vendors coming soon. If you have something not listed here, please contact shifter-hpc@googlegroups.com!

[1] https://www.mpich.org/abi/

6.2 Shifter Integration with Slurm

Shifter offers tight integration with Slurm by way of the SPANK plugin system. Slurm 15.08.2 or better is recommended for basic functionality, and 16.05.3 or better with the extern step integration.

Enabling Slurm integration has the following benefits:

- simple user interface for specifying images and volume maps
- scalable startup of parallel applications
- optional sshd started in node-exclusive allocations to enable complex workflows (processes attached to custom memory cgroup or extern step job container)
- optional Cray-CCM-emulation mode to enable ssh support in the CLE 5.2 DSL environment under "native" Slurm
- optional extern step post-prolog image configuration (useful for tie-breaking parallel prolog operations)

Integration with Slurm causes the `setupRoot` executable to be run in a per-node prolog at job allocation time. Conversely, at job-end a per-node epilog runs the `unsetupRoot` executable to deconstruct the UDI environment. setupRoot generates a container image environment in the same Linux namespace as the slurmd, meaning that the same environment can be re-used over-and-over again by multiple sruns, as well as allowing multiple tasks to be simultaneously launched within that container environment. Use of the setupRoot environment also restricts the quantity of loop devices consumed by a job to just those needed to setup the environment (typically one for a basic environment).

Without `setupRoot` to prepare the environment, the shifter executable can do this, but these are all done in separate, private namespaces which increases setup time and consumes more loop devices. If the user specifies `--image` or `--volume` options for the shifter executable that differ from the job-submitted values, a new shifter instance will be instantiated. This enables the capability of running multiple containers within a single job, at the cost of increased startup time and perhaps a small amount of overhead.

Integration with Slurm also increases scalability, as well as consistency, by caching the specific identity of the image to be used at job submission time. This increases scalability by transmitting the image ID to all nodes, thus limiting interactions between shifter and the shifter image gateway to when the job is initially submitted, and allowing all job setup actions to occur without further interation with the image gateway. Caching image identity increases job consistency by ensuring that the version of an image that was on the system at the time the job was submitted will be used when then job is allocated an run, even if newer versions are pulled later.

6.2.1 Configuring Slurm Integration

1. Build shifter with Slurm support `./configure --with-slurm=/slurm/prefix` or `rpmbuild -tb shifter-$VERSION.tar.gz --define "with_slurm /slurm/prefix"`
2. Locate shifter_slurm.so, will be in the shifter prefix/lib/shifter/ (or within lib64) directory.
3. Add `required /path/to/shifter_slurm.so` to your slurm plugstack.conf
4. Modify plugstack.conf with other options as desired (see documentation)
5. Ensure slurm.conf has `PrologFlags=alloc` or `PrologFlags=alloc,contain` The `alloc` flag ensures the user-requested image will be configured on all nodes at job start (allocation) time. The `contain` flag is used to setup the "extern" step job container.

6.2.2 Shifter/Slurm configuration options

Additional configuration options can be added in plugstack.conf after the shifter_slurm.so path. These options will control the behavior of the plugin.

- *shifter_config* - by default the configured location for udiRoot.conf wil be used, however, if there is a specific version that should be used by the Slurm plugin, adding `shifter_config=/path/to/custom/udiRoot.conf` will achieve that.
- *extern_setup* - optionally specify a script/executable to run in the setup of the extern step, which occurs after the prolog runs. This can be used to force specific modifications to the WLM-created container image. Any operation performed in this script must be very quick as ordering of `prolog -> extern_setup -> job start` is only guaranteed on the batch script node; all others may possibly have a possible race with the first srun if the batch script node runs first (thus this option is not recommended). e.g., `extern_setup=/path/to/script`
- *extern_cgroup* - Flag 0/1 (default 0 == off) to put the optional sshd and, therefore all of its eventual children, into the extern cgroup. This is recommended if using the sshd and slurm 16.05.3 or newer., e.g., `extern_cgroup=1`
- *memory_cgroup* - Path to where the memory cgroup is mounted. This is a recommended setting in all cases. The plugin will create a new cgroup tree under the memory cgroup called "shifter", and within that will follow the Slurm standard (uid_<uid>/job_<jobid>). e.g., `memory_cgroup=/sys/fs/cgroup/memory`
- *enable_ccm* - Enable the CCM emulation mode for Cray systems. Requires enable_sshd as well. e.g., `enable_ccm=1`
- *enable_sshd* - Enable the optional sshd that is run within the container. This sshd is the statically linked sshd that is built with shifter. If enabled it is run as the user within the container. See the shifter sshd documentation for more information.

6.2.3 Using Shifter with Slurm

Basic Job Submission and Usage

The shifter_slurm.so plugin will add `--image`, `--volume`, and, if enabled, `--ccm` options to sbatch, salloc, and srun. Typical usage is that a user would submit a job like:

```
cluster$ sbatch script.sh
```

Where script.sh might look like:

```
#!/bin/bash
#SBATCH --image=docker:yourRepo/yourImage:latest

srun shifter mpiExecutable.sh
```

See the MPI-with-shifter documentation for more information on how to configure shifter to make MPI "just work" in shifter.

Remember ideal performance is realized if the `--image` and ALL `--volume` options are supplied in the batch submission script. If these options are supplied to the shifter command it will cause new shifter instances to be generated at runtime, instead of using the prepared `setupRoot` environment.

Non-MPI and serial applications

To start a single instance of your threaded or serial application, there are two formats for the script you can use depending on your needs. In this case we don't need access to srun, thus it is possible to directly execute the script within shifter if that is desirable.

Note that all paths mentioned in `--volume` arguments need to exist prior to job submission. The container image will be setup, including the volume mounts prior to execution of the batch script.

Option 1: Explicitly launch applications in the image environment while keeping logic flow in the external (cluster) environment:

```
#!/bin/bash
#SBATCH --image=docker:yourRepo/yourImage:latest
#SBATCH --volume=/scratch/sd/you/exp1/data:/input
#SBATCH --volume=/scratch/sd/you/exp1/results:/output
#SBATCH -c 64

## -c 64 in this example, assuming system has 64 hyperthreads (haswell),
## because we want the batch script, and thus all the commands it runs to
## get access to all the hardware

cp /scratch/sd/you/baseData /scratch/sd/you/exp1/data
export OMP_NUM_THREADS=32
shifter threadedExecutable /input /output

## do other work with /scratch/sd/you/exp1/results, post-processing
```

Option 2: Execute script in shifter container with no direct access to the external environment. Easier to write more complex workflows, but the container must have everything needed:

```
#!/usr/bin/shifter /bin/bash
#SBATCH --image=docker:yourRepo/yourImage:latest
#SBATCH --volume=/scratch/sd/you/exp1/data:/input
#SBATCH --volume=/scratch/sd/you/exp1/results:/output
#SBATCH -c 64

export OMP_NUM_THREADS=32
threadedExecutable /input /output

python myComplexPostProcessingScript.py /output
```

Complex Workflows with Multiple Nodes and No MPI, or non-site integrated MPI

You can enable the sshd capability by adding the `enable_sshd=1` option in plugstack.conf on the shifter_slurm.so line. This will start a specially constructed sshd on port 204 on each node. This sshd will only all the user to login, and only using an ssh key constructed (automatically) for the explicit use of shifter. All the manipulations to change the default ssh port from 22 to 1204 as well as provide the key are automatically injected into the image container's /etc/ssh/ssh_config file to ease using the sshd.

Once in the container environment the script can discover the other nodes in the allocation by examining the contents of `/var/hostslist`, which is in a PBS_NODES-style format.

This could allow an mpirun/mpiexec built into the image to be used as well by using the `/var/nodeslist` and an ssh-based launcher.

If the user can access the external environment sshd, one could avoid turning on the shifter sshd, and just use the standard `scontrol show hostname $SLURM_NODELIST` to discover the nodes, then do something like: `ssh <hostname> shifter yourExecutable` to launch the remote process.

Note that the shifter sshd is only enabled if the job allocation has exclusive access to the nodes. Shared allocations will not run `setupRoot`, and therefore not start the sshd.

Using Shifter to emulate the Cray Cluster Compatibility Mode (CCM) in native slurm

The CCM (`--ccm`) capability is a special use-case of shifter to automatically start and allow the user access to the sshd that shifter can start. This mode is distinct because it can automatically put the user script/session into the shifter environment prior to task start. This is typically avoided to prevent Slurm from operating with privilege in the user defined environment. However, it is permissible in the unique case of CCM, because CCM targets _only_ the already existing external environment, not a user-specified one. I.e., CCM mode makes a shifter container out of the /dsl environment, starts an sshd in it, then launches the job in that containerized revision of the DSL environment.

To enable `--ccm`, you'll need both `enable_ccm=1` and `enable_sshd=1` in plugstack.conf. In addition you'll need to set `allowLocalChroot=1` in udiRoot.conf. This is because CCM effectively works by doing:

```
shifter --image=local:/  # but with setupRoot so the sshd can be setup
```

6.2.4 Frequently Asked Questions

Why not just start the job in the container environment?

This is technically feasible, however we do not enable it by default for a number of reasons; though there has been much discussion of it in the past and may be more in the future. For example `--ccm` does this for the special case of a locally constructed image `/`.

Why not do it?:

1. We would need to chroot into the container in the task_init_privileged hook which carries a great deal of privilege and is executed far too early in the job setup process. A number of privileged operations would happen in the user specified environment, and we felt the risk was too high.
2. It is highly useful to have access to the external environment. This allows you to perform sruns to start parallel applications, move data the site may not have necessarily exported into the shifter environment, access commands or other resources not trivially imported into a generic UDI.
3. We did try to force slurm into `/opt/slurm` of the container to allow srun and job submission to work within the container environment, but owing to the way Slurm interacts with so many of the local system libraries via dynamic linking, there were too many edge cases where direct interaction with Slurm from within a generic UDI was not working quite right. Also there may be some security concerns with such an approach.

6.3 Shifter Modules

To adapt containers to a particular system and meet specific user needs, Shifter provides sites the ability to configure "modules" which more flexibly implement the the globally applied modifications that Shifter can perform for all container invocations.

A Shifter module allows specific content to be injected into the container, environment variables to be manipulated, and hooks both as root during container instantiation (not _within_ the container), and as the user (in the container) immediately before process start. The capabilities allow a site to customize all user containers and provide local support for a variety of system capabilities, and then give the users the option to use the local support (enable the module) or disable it if their container has other needs.

6.4 Example Modules:

6.4.1 CVMFS at NERSC:

NERSC makes CVMFS available on the /cvmfs_nfs mount point, where cvmfs is made available via NFS and aggressively cached in a number of ways.

If a user wants to use it in their container, a simple module could be:

udiRoot.conf: module_cvmfs_siteFs = /cvmfs_nfs:/cvmfs module_cvmfs_siteEnv = SHIFTER_MODULE_CVMFS=1

Then the shifter invocation would be:

shifter –image=<image> –module=cvmfs ...

This can also be achieved with volume mounts, but the module system allows a user to _avoid_ getting cvmfs, unless they want it.

6.5 Cray mpich Support

Cray mpich support makes use of the ABI compatibility interface. If a user has an application linked against vanilla mpich, then the Cray version can be interposed via linking operations.

Thus if the site prepares a copy of the CrayPE environment for use by shifter (using the prep_cray_mpi_libs.py script in the "extra" directory of the shifter distribution), then uses the copyPath (or siteFs) mechanism to inject those libraries into the container, and finally uses LD_LIBRARY_PATH to force the application to see the new version, then the user can use the same container on a variety of systems.

udiRoot.conf: module_mpich_copyPath = /shared/shifter/modules/mpich module_mpich_siteEnvPrepend = LD_LIBRARY_PATH=/opt/udiImage/modules/mpich/lib64 module_mpich_userhook = /opt/udiImage/modules/mpich/bin/init.sh

If the user prefers their own mpich version, and wants to make use of the shifter ssh interface, then they can avoid using the mpich module. If the site makes the mpich module default, the user can still disable it with by specifying "–module=none", which turns off all modules.

The userhook script above is executed immediately prior to the user process being executed and can be used to validate that the application in the container is compatible with the MPI libraries being injected.

6.6 The shifter sshd

The Workload Manager integration can cause an sshd to be started within each instance of the container. This feature is useful for allowing complex workflows to operate within the User Defined environment, and operate as if they were running on a cluster running the User Defined image.

6.6.1 How to use it: User Perspective

To use the shifter ssh access, the "basic" way is to start a batch job specifying a shifter image. So long as your job has exclusive access to the compute nodes in its allocation, the shifter container should be setup, and be pre-loaded with the running sshd. You are the only user that can login via this sshd.

Once your job is running, enter the shifter environment with the shifter command, e.g., `shifter /bin/bash`

Once in the shifter environment you should be able to simply *ssh* to any other node in your allocation, and remain within the container environment.

Note, this works because of a special ssh configuration loaded into your container environment at runtime. If your home directory has a `~/.ssh/config` it may override some of the necessary shifter sshd settings, in particular if you override IdentityFile for all hosts.

With a complete implementation of the batch system integration, you should be able to get a complete listing of all the other hosts in your allocation by examining the contents of `/var/hostsfile` within the shifter environment.

TODO: To be continued...

6.6.2 How to configure it: Administrator Perspective

If you didn't disable the build of the sshd when shifter was compiled, it would be installed in the udiImage directory (`%(libexec)/shifter/opt/udiImage`). The udiImage directory can be moved onto your parallel/network storage to ease maintenance. Just make sure to update the udiRoot.conf `udiImagePath` to reflect the change. Under `udiImage/etc` you'll find the default sshd_config and ssh_config files. The can be modified to suit your needs, however they have been pre-configured to attempt to meet most ssh needs of the shifter containers, namely:

1. Disable privileged login
2. Only allow the user to login
3. Only permit the user's udiRoot ssh key to login to the sshd
4. Intentionally omits an ssh host key as these are dynamically generated and accessed within the container environment

When the container is setup, the `udiImagePath` will be copied to `/opt/udiImage` within the container. The ssh installation is based using this path.

TODO: To be continued...

6.6.3 How to configure it: WLM Integrator Perspective

TODO: Not started yet.

6.6.4 Nitty Gritty Details

TODO: Not started yet.

Can I use the user sshd in Cray CLE5.2UP03 or UP04?

Yes, however the default environment does not (or did not) support it out-of-the-box. You'll need a few things:

1. shifter 16.08.3 or newer
2. 5.2UP04 or 5.2UP03 fully patched for the glibc issues earlier in 2016
3. A modified compute node /init

The compute node /init needs to be updated to mount /dev/pts with proper options to allow pty allocation in the absence of pt_chown, e.g., replace:

mkdir -p "$udev_root/pts" mount -t devpts devpts "$udev_root/pts"

with:

mkdir -p “$udev_root/pts” mount -t devpts -o gid=5,mode=620 devpts “$udev_root/pts”

6.7 Importing Images (beta)

While Shifter is primarily designed to run images built using Docker, it also supports directly importing pre-prepared images. Since some sites may not wish to support this or limit its usage, it requires extra configuration.

6.7.1 Security Considerations

Before enabling this feature, the administrator should understand the increased risks of allowing direct imports of images perpared outside of Shifter. The Shifter gateway should not be capable of creating images that contain security attributes. This helps minimize the risk of images introducing additional security risks. If you enable the importing of images, you should take extra precautions in how images are prepared and who you trust to import them. Images should ideally be prepared as a unprivileged user and the -no-xattrs flag should be passed to the mksquashfs command to mitigate the risks of security attributes being included in the image. When using the workload manager integration (e.g. Slurm plugin), it is especially critical that direct import users block xattrs since the mounts may be visible to processes running outside the runtime.

6.7.2 Approach

Importing images works by copying a pre-prepared image into the shared location, generating a pseudo-hash for the image, and adding an entry in the Shifter image list.

6.7.3 Enabling support

To enable support, add an “ImportUsers” parameters into the top section of the imagemanager.json file. This can be a list of user names (in JSON list format) or the keyword “all”. For example:

```
{
   "ExpandDirectory": "/tmp/images/expand/",
   "ImportUsers":"all"
   "Locations": {
   }
}
```

The fasthash script needs to be installed as *fasthash* in a location on the search path for the image gateway for local mode or in the search path on the remote system for remote mode. This script generates a pseudo-hash for the image based on the contents of the image.

6.7.4 Usage

The user issuing the import most have the squashed image in a location that is accessible by the shifter user (e.g. the account used to run the gateway).

The command line tools do not currently support import. So a curl command must be issued. Here is an example of an import command to import the squashfs image located at /tmp/image.squashfs and call it load_test:v1 in Shifter.

```
curl -d '{"filepath":"/tmp/myimage.squashfs","format":"squashfs"}' \
  -H "authentication: $(munge -n)" \
  http://<imagegw>:5000/api/doimport/mycluster/custom/load_test:v1/
```

Once an image is imported, it is run with Shifter like other images. The only difference is the type is "custom" instead of the default "docker".

```
shifter --image=custom:load_test:v1 /bin/app
```

CHAPTER 7

Security Considerations with Shifter

WARNING: Shifter use a great deal of root privilege to setup the container environment. The "shifter" executable is setuid-root, and when run with batch integration the setupRoot/unsetupRoot utilities must be run as root. We are working to reduce the privilege profile of starting Shifter containers to reduce risk as much as possible.

Once a process is launched into the container, processes are stripped of all privilege, and should not be able to re-escalate afterwards.

Shifter enables User Defined Image environment containers. To do this while optimizing I/O on the compute nodes it does require performing several privileged operations on the execution node, both privilege to mount filesystems and rewrite the user space, and privilege to manipulate devices on the system.

Furthermore, because the environment is _user **defined**_, it is possible that a user could include software which could generate security vulnerabilities if a privileged process accesses such software or resources.

Therefore, to mitigate potential risks, we recommend the following:

1. Avoid running privileged processes in containers. This means both explicitly chroot'ing into a container as root, or joining the namespace of an existing container, as well as preventing setuid root applications to operate *at all* within the container.

   On more recent systems, Shifter will attempt to permanently drop privileges using the "no_new_privs" process control setting, see: https://www.kernel.org/doc/Documentation/prctl/no_new_privs.txt

   See the *The shifter sshd* document for more information on the Shifter-included sshd and recommendations around running it as root (don't unless you must).

2. Related to point one, preventing setuid-root applications from operating with privilege is mostly achieved through mounting as much as possible within the Shifter environment "nosuid", meaning the setuid bits on file permissions are ignored. In addition, processes started within shifter and their children are prevented from ever gaining additional privileges by restricting the set of capabilities they can acquire to the null set.

   One exception to the nosuid ban is if the ":rec" or ":shared" siteFs mount flags are used. The recursive bind mount operation will copy the mount flags from the base system, and will not follow Shifter standards. Similarly, the "shared" mount propagation strategy will remove the mounts from Shifter's strict control. The privilege capability restrictions should prevent processes from escalating privilege even without the nosuid restriction.

3. Use the most recent version of Shifter.

7.1 Notes on Security Related Options and Future Directions for udiRoot

Shifter attempts to provide an HPC Environment Container solution for a variety of HPC platforms. This means a wide variety of Linux kernels must be supported.

A note on User Namespaces: We have considered the use of user namespaces, and may make more use of them in the future. At the present, however, too few of the target Linux distributions support them or support them well to make the investment worthwhile. Also there have been a number of security issues surrounding uid and gid mapping so as not to make it a safe and reliable solution until quite recent linux kernels. The promise of user/group mapping will help to ensure the safety of running software from potentially hostile/insecure environments, in particular helping to relax the constraints around the prohibition of setuid files.

A note on loop devices: Shifter makes heavy use of loop devices owing to the great performance of benefits optimizing the balance of local node memory and network bandwidth consumption, while avoiding some of the performance issues caused by distributed locking in traditional network filesystems. The use of these loop devices, however, means that filesystems are being managed and accessed directly by the kernel, with privileged access. This means that the filesystem files *must never be writable by users directly*, and should be produced by toolsets trusted by the site operating Shifter. Directly importing squashfs filesytems should be avoided unless you trust the individual that produced the content with root privileges. (You wouldn't pick up a USB drive off the street and put it into your computer, would you?)

7.2 Notes on Security attributes

Linux supports the ability for file systems to have security attributes attached to individual files. This allows, for example, ping to be non-setuid yet still create raw packets. This can introduce additional risk with running containers. Shifter takes a number of precautions to mitigate these risks. The Image gateway uses a python tar library that doesn't currently support extracting files with the security attribute. In addition, we recommend running the gateway as a non-root user, e.g. shifter. This adds additional security because an unprivileged user cannot add security attributes to a file. Shifter passes flags to mksquashfs to further prevent these attributes from being included in the image. Finally, the runtime uses a number of mechanisms to ensure these attributes are ignored. These combination of features greatly reduce the risk of a unprivileged user from using specially crafted images to gain privileges.

Users making use of the custom image import option (see import.md) should take additional precautions since this mechanism effectively bypasses the image preparation steps in the gateway and the images could include security attributes. It is recommended that sites build these images using a unprivileged account and pass the -no-xattrs flag to mksquashfs to mitigate this risk. The runtime should still prevent these images from conferring any additional privileges to the user process, but dropping the attributes during preparation is a good precaution. In addition, sites should limit the users allowed to perform imports.

7.3 Image Manager Considerations

In general, the image gateway does not require any privileges to performs its functions. The gateway does control what images are made available to the system, but importing these images can be done without special privileges.

7.3.1 User Authentication

Access to the image gateway API service requires authentication using Munge. Munge is a light-weight, shared key authentication system that allows a privileged daemon process to authenticate the user who is calling the process.

User authentication is used in several way. It is used to verify the user before performing some restricted administrative functions (e.g. image expiration) and to prevent external clients from triggering operations that could impact response times. In addition, with the introduction of image ACLs, user authentication is used to limit access to restricted images based on the desired access controls of the image owner(s).

7.3.2 Securing imagegw api

1. run as a non-root user using gunicorn as the python flask host
2. Use a firewall to restrict traffic just to the networks that need to access the imagegw
3. install the mksquashfs, ext3/4 utilities and xfs progs in a trusted way (e.g., package manager of your distribution)

Running the imagegw as a non-root user is particularly important to ensure images generated do not have an Linux security capabilities embedded in the image. This is a non-obvious way that a program may attempt to escalate privilege. On more recent Linux systems (Linux kernel >= 3.5), this risk is somewhat mitigated so long as the shifter executable is rebuilt for those systems.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search