shifter Documentation

Release 16.05.01

Shane Canon and Douglas Jacobsen

Feb 15, 2020

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 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

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.

3.1 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 your tag appropriately, of course)

3.2 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:~>
```

3.3 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

3.3.1 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 (https://nixos.org/patchelf.html)

Next copy /tmp/craylibs to your shifter module path (see Modules) under mpich/lib64, e.g., `/usr/lib/shifter/modules/mpich/lib64`.

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/modules/mpich"
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/

CHAPTER 4

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

4.1 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:

```
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 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 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.

4.4 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
```

4.5 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.

4.6 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)

CHAPTER 5

Installing Shifter Runtime in Cray's CLE 6.0UP01

5.1 Setting up Compute node kernel support

1. Build compute node image
2. Boot compute node

3. Install appropriate kernel-sources and kernel-syms e.g.,

    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 6. Put resulting RPM in a repo, pkgcoll, and update compute image

5.2 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 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

5.2.1 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 the udiRoot.conf siteFs: /var/opt/cray/dws:/var/opt/cray/dws:rec:slave

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

5.2.2 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/lib64 python /path/to/extra/prep_cray_mpi_libs.py /tmp/cray/lib64 ""

   Copy /tmp/cray to a path within the udiImage directory (by default /usr/lib/shifter/opt/udiImage)
3. **To automate override of libmpi.so modify siteEnvAppend to add:** LD_LIBRARY_PATH=/opt/udiImage/cray/lib64

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 (https://nixos.org/patchelf.html).

CHAPTER 6

Manual installation of Shifter with GPU support

6.1 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.

CHAPTER 7

Installation

7.1 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.

7.1.1 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
```

7.1.2 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
```

7.1.3 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.

7.1.4 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
```

7.2 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.

7.2.1 Software dependencies for the Image Gateway

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
```

7.2.2 Installation of the Image Gateway

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
```

7.2.3 Configure

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"
```

7.2.4 Image Gateway Startup

Start the service MongoDB:

```
sudo systemctl start mongod
```

Start the Shifter Image Gateway:

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

CHAPTER 8

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.

8.1 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.

8.2 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.

8.3 Using Shifter with Slurm

8.3.1 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.

8.3.2 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
```

8.3.3 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.

8.3.4 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
```

8.4 Frequently Asked Questions

8.4.1 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.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search