Installation, Configuration and Performance Tuning of Shifter V16 on Blue Waters

HonWai Leong*, Timothy Bouvet†, Brett Bode‡, Jeremy Enos§ and David King¶
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
Illinois, United States of America
*Email: *hwleong@illinois.edu, †tbouvet@illinois.edu, ‡brett@illinois.edu,*
§jenos@illinois.edu, ¶kingda@illinois.edu

***Abstract*—NCSA recently announced the availability of Shifter version 16.08.3 (V16) for production use on Blue Waters. Shifter provides researchers with the capability to execute container-based HPC applications on Blue Waters. In this paper, we present the procedure that we performed to backport Shifter V16 to Blue Waters. We describe the details of the installation of the Shifter software stack, code customization, configuration, and the complex integration efforts to scale Shifter jobs to start in parallel on a few thousands compute nodes. We will discuss in this paper the methods and workarounds that we utilized to address the challenges that we encountered during the deployment, which include security hardening, performance tuning, running GPU workloads and other operation related issues. Today, we have successfully tuned Shifter to the scale that could execute a container-based job on Blue Waters across more than 4000 compute nodes.**

***Keywords*-Shifter, Docker, Blue Waters, Container**

I. Introduction

Shifter [1], [2] is a software solution that enables the execution of container-based applications on HPC systems. The development of Shifter, a joint collaboration effort between NERSC and Cray that started in 2015, was driven by the need to use Docker-like container technology on HPC systems to address the increasing demand of data-intensive workloads [3]. Shifter was initially designed to port over container-based applications to run on NERSC's Edison system, a Cray XC30 supercomputer system. It was later made available as an open source tool for the general HPC community. Like Docker, Shifter allows researchers to reuse a Docker container on any HPC system. Researchers can develop and test their scientific software stack using Docker on their own workstation, then publish the Docker container to a public registry such as Dockerhub, where they can import it using Shifter into an HPC system, and be able to run their simulations there without redeveloping their software stack. Unlike Docker which could only start a container-based application on a single machine locally and requires root access, Shifter is an HPC-centric implementation of container technology. It was designed for researchers (who usually do not have root access to a shared HPC system), to leverage the flexibility of container technology, to seamlessly scale their applications from local workstations or a smaller HPC system to a large number of nodes in a larger HPC system. Adopting container-based technology in software development potentially reduces the development overhead for both researchers and HPC system support staff of porting an application across different systems, and improves reproducibility.

The very first release of Shifter (Version 1.0) is included and is supported as part of the software stack in Cray Linux Environment (CLE) release 5.2.UP04. Blue Waters, a Cray hybrid XE6 and XK7 supercomputer system has had Shifter V1 in production since 2016. The first release of Shifter had a simple architecture where only an image gateway (written in Python) is used to download and convert a Docker image into an user defined image (UDI) format. Upon request by a job, the workload manager invokes a prologue script to mount the requested UDI onto the allocated compute nodes as read-only ext4 file system, giving access for the job to launch applications through the software stack environment provided by the container. In this architecture, the Docker engine is still required to be installed locally on the image manager node to pull Docker containers from public Dockerhub registry. The implementation details of Shifter V1 can be found in [1].

Since the initial release of Shifter, regular updates are provided through CLE patches. A major change in Shifter architecture was introduced in the 2016 version (V16) [2]. This version release (and the following updates) is officially supported and implemented in CLE release 6.0.UP02 and onwards for Cray XC series systems. However, for Cray XE6/XK7 systems such as Blue Waters, CLE 5.2.UP04 is the last CLE major release so we do not get these Cray-provided updates. Though we could continue to maintain the V1 release of Shifter on Blue Waters, a minor security issue has raised concerns and the scalability demand of porting and executing container-based applications on Blue Waters have together driven us to find a way to upgrade and make available the newer release of Shifter (Version 16) on Blue Waters.

In researching a path to install Shifter V16 onto Blue Waters, we referred to the official Cray configuration guide of Shifter [4], [5] and documentation from Shifter's web resource [6]. Cray provides seamless integration of Shifter V16 into the XC series system. It requires installing the

Shifter RPM packages into the compute node boot image, rebuilding the boot image and rebooting the system to the new boot image. As Shifter V16 has not been tested on a Cray XE/XK system before, we foresaw inevitable ongoing changes to the boot image during the integration, and the efforts required in working and testing on the boot image directly would be tedious and time consuming. Also, as a best practice for a four-year old seasoned system like Blue Waters, we would prefer not to modify the boot image whenever possible to reduce risk of breaking the system after installing untested Shifter RPM packages into the boot image directly. Hence, to mitigate the risk in a more controllable fashion, we decided to install Shifter V16 in the `/dsl` layer, Cray's proprietary file system projected through the Data Virtualization Service (DVS). Testing Shifter in `/dsl` allows faster turnaround time, where a fix in the source code or configuration can be retested quickly without the hassle of rebuilding the boot image and rebooting the compute node. We first installed Shifter V16 and carried out most of the integration work on our test and development system (TDS) [7], before replicating the deployment over to Blue Waters. As the TDS does not have the scale to reproduce the performance issue that we encountered on Blue Waters, testing Shifter in `/dsl` gave us the agility to quickly compile an instrumented Shifter binary on Blue Waters to debug the bottleneck root cause.

In this paper, we will present the procedure used to backport Shifter V16 to Blue Waters. The purpose of this paper is to provide comprehensive guidance to other HPC sites that are planning to implement Shifter V16 on older generations of non-XC Cray systems. This paper addresses issues that other sites may also encounter, including those that are running XC systems. Though the procedure presented here focuses on Cray systems, some of the points may also be applicable to other general HPC systems. The following sections will be discussed in this paper:

II. Installation of Shifter software stack;
III. Configuration of Shifter software stack;
IV. Integration with workload manager;
V. Scaling performance;
VI. GPU support on Shifter; and
VII. Other operational issues.

II. Installation of Shifter Software Stack

The Shifter V16 software stack includes the following components (indicated software version is latest at the time of installation):

1. Shifter 16.08.3
2. Squashfs Linux kernel module
3. MongoDB 3.4.7
4. Redis 3.2.8
5. Python 2.7.13
6. Python modules: *Celery*, *PyMongo*, *Flask*, *redis*, *gunicorn*.
7. Munge

We will describe the installation procedure for each of these components. We planned and tested the installation procedure on the TDS before replicating the procedure over to Blue Waters. All installations are done in `/dsl` through `xtopview` utility on the boot node.

```
boot:~ # xtopview
default/:/ #
```

Listing 1. Installation in `/dsl` through `xtopview`.

A. Dependencies

The following dependencies were installed to provide header files and libraries required to build Shifter RPM packages: *fdupes*, *json*, *squashfs*. These packages are only required to build Shifter RPM packages, they are not required to be installed on Blue Waters. We installed the source RPM packages of these dependencies (downloaded from *openSUSE* [8] online repository), built the binary RPM packages and installed them in `/dsl`.

```
default/:/software/rpms # rpm -ivh fdupes-1.61-9.3.1.src.rpm json-c-0.12.1-47.4.src.rpm squashfs-4.3-45.1.x86_64.rpm
default/:/software/rpms # export CC=gcc
default/:/software/rpms # rpmbuild -ba /usr/src/packages/SPECS/fdupes.spec
default/:/software/rpms # rpmbuild -ba /usr/src/packages/SPECS/json-c.spec
default/:/software/rpms # cd /usr/src/packages/RPMS/x86_64
default/:/usr/src/packages/RPMS/x86_64 # rpm -ivh fdupes-1.61-9.3.1.x86_64.rpm libjson-c-devel-0.12.1-47.4.x86_64.rpm libjson-c2-0.12.1-47.4.x86_64.rpm
```

Listing 2. Install dependencies RPM packages

As CLE 5.2.UP04 is based on older SUSE Linux Enterprise Server 11.3 distribution, building of Shifter RPM packages requires newer *Autoconf* and *Automake* tools.

```
default/:/software/autoconf-2.69 # ./configure --prefix=/software/usr CC=gcc
default/:/software/autoconf-2.69 # make
default/:/software/autoconf-2.69 # make install

default/:/software/automake-1.15.1 # export PATH=/software/usr/bin:$PATH
default/:/software/automake-1.15.1 # ./configure --prefix=/software/usr CC=gcc
default/:/software/automake-1.15.1 # make
default/:/software/automake-1.15.1 # make install
```

Listing 3. Install newer version of *Autoconf* and *Automake* tools.

B. Shifter

We cloned the source distribution of Shifter from NERSC's Shifter *github* repository.

```
default/:/software # git clone https://github.com/NERSC/shifter.git shifter-16.08.3
```

Listing 4. Cloning Shifter source distribution from NERSC's *github* repository.

1) Source Code Modification: During our test and development phase on the TDS, we encountered a few issues that required editing the Shifter source code to suit our environment. Listing 5, 6 and 7 list the changes made in `shifter_core.c`, `UdiRootConfig.h` and `UdiRootConfig.c` source files respectively.

1. Shifter is distributed with its own `mount` binary. By default, this binary is installed as `/usr/lib64/shifter/mount`. The location was hard coded by `LIBEXECDIR` definition in `shifter_core.c` source file at the time when Shifter RPM packages were built, thus the installation path of `mount` binary is not relocatable other than `/usr`. We worked around this issue by adding a configurable `mountCmd` definition in `UdiRootConfig.c` and `UdiRootConfig.h` source files, and replaced `LIBEXECDIR` with `udiConfig->mountCmd` in `shifter_core.c` source file. With this change in place, the installation path of Shifter's `mount` binary became relocatable and can be defined by the `MountCmd` parameter in `UdiRoot.conf` configuration file (see section III-B later).

2. Shifter has the capability to load additional Linux kernel modules required at runtime (e.g. `loop.ko` and `squashfs.ko`), given `kmodBasePath` parameter is defined in `udiRoot.conf` file. The source distribution of Shifter provided an RPM Spec file to build the required Linux kernel module files in an RPM package. Files from the RPM package were to be installed in `$PREFIX/modules/‘uname -r‘/kernel` directory. We noticed a flaw in the original Shifter runtime binary, where it constructs the kernel module lookup path as `kmodBasePath + ‘uname -r‘`, neglecting the `kernel` subdirectory. This caused a failure in the Shifter runtime as it tried to find and load the required kernel module from an invalid path. We fixed this in the `shifter_core.c` source file by including the `kernel` subdirectory in the construct path.

```
diff --git a/src/shifter_core.c b/src/shifter_core.c
index a1ad10c..b4b50fd 100644
--- a/src/shifter_core.c
+++ b/src/shifter_core.c
@@ -1229,7 +1229,7 @@ int loopMount(const char *imagePath, const char *loopMountPath, ImageFormat form
     goto _loopMount_unclean; \
 }

-    snprintf(mountExec, PATH_MAX, "%s/mount", LIBEXECDIR);
+    snprintf(mountExec, PATH_MAX, "%s", udiConfig->mountCmd);

     if (stat(mountExec, &statData) != 0) {
         fprintf(stderr, "udiRoot mount executable missing: %s\n", mountExec);
@@ -2762,7 +2762,7 @@ int loadKernelModule(const char *name, const char *path, UdiRootConfig *udiConfi
     }

     /* construct path to kernel modulefile */
-    snprintf(kmodPath, PATH_MAX, "%s/%s", udiConfig->kmodPath, path);
+    snprintf(kmodPath, PATH_MAX, "%s/kernel/%s", udiConfig->kmodPath, path);
     kmodPath[PATH_MAX-1] = 0;

     if (stat(kmodPath, &statData) == 0) {
```

Listing 5. Source code modification in `shifter_core.c`.

```
diff --git a/src/UdiRootConfig.h b/src/UdiRootConfig.h
index 69fe480..b783423 100644
--- a/src/UdiRootConfig.h
+++ b/src/UdiRootConfig.h
@@ -108,6 +108,7 @@ typedef struct _UdiRootConfig {
     size_t maxGroupCount;
     size_t gatewayTimeout;
     size_t mountPropagationStyle;
+    char *mountCmd;

     char *modprobePath;
     char *insmodPath;
```

Listing 6. Source code modification in `UdiRootConfig.h`.

```
diff --git a/src/UdiRootConfig.c b/src/UdiRootConfig.c
index 4a2e26f..24cd0b1 100644
--- a/src/UdiRootConfig.c
+++ b/src/UdiRootConfig.c
@@ -156,6 +156,10 @@ void free_UdiRootConfig(UdiRootConfig *config, int freeStruct) {
         free(config->mvPath);
         config->mvPath = NULL;
     }
+    if (config->mountCmd != NULL) {
+        free(config->mountCmd);
+        config->mountCmd = NULL;
+    }
     if (config->chmodPath != NULL) {
         free(config->chmodPath);
         config->chmodPath = NULL;
@@ -291,6 +295,8 @@ size_t fprint_UdiRootConfig(FILE *fp, UdiRootConfig *config) {
         (config->cpPath != NULL ? config->cpPath : ""));
     written += fprintf(fp, "mvPath = %s\n",
         (config->mvPath != NULL ? config->mvPath : ""));
+    written += fprintf(fp, "mountCmd = %s\n",
+        (config->mountCmd != NULL ? config->mountCmd : ""));
     written += fprintf(fp, "chmodPath = %s\n",
         (config->chmodPath != NULL ? config->chmodPath : ""));
     written += fprintf(fp, "ddPath = %s\n",
@@ -360,6 +366,9 @@ int validate_UdiRootConfig(UdiRootConfig *config, int validateFlags) {
         if (config->mvPath == NULL || strlen(config->mvPath) == 0) {
             VAL_ERROR("\"mvPath\" is not defined", UDIROOT_VAL_PARSE);
         }
+        if (config->mountCmd == NULL || strlen(config->mountCmd) == 0) {
+            VAL_ERROR("\"mountCmd\" is not defined", UDIROOT_VAL_PARSE);
+        }
         if (config->chmodPath == NULL || strlen(config->chmodPath) == 0) {
             VAL_ERROR("\"chmodPath\" is not defined", UDIROOT_VAL_PARSE);
         }
@@ -392,6 +401,11 @@ int validate_UdiRootConfig(UdiRootConfig *config, int validateFlags) {
         } else if (!(statData.st_mode & S_IXUSR)) {
             VAL_ERROR("Specified \"mvPath\" is not executable.", UDIROOT_VAL_FILEVAL);
         }
```

```
+      if (stat(config->mountCmd, &statData) != 0) {
+          VAL_ERROR("Specified \"mountCmd\" doesn't appear to exist.", UDIROOT_VAL_FILEVAL);
+      } else if (!(statData.st_mode & S_IXUSR)) {
+          VAL_ERROR("Specified \"mountCmd\" is not executable.", UDIROOT_VAL_FILEVAL);
+      }
       if (stat(config->chmodPath, &statData) != 0) {
           VAL_ERROR("Specified \"chmodPath\" doesn't appear to exist.", UDIROOT_VAL_FILEVAL);
       } else if (!(statData.st_mode & S_IXUSR)) {
@@ -494,6 +508,8 @@ static int _assign(const char *key, const char *value, void *t_config) {
       config->cpPath = strdup(value);
   } else if (strcmp(key, "mvPath") == 0) {
       config->mvPath = strdup(value);
+  } else if (strcmp(key, "mountCmd") == 0) {
+      config->mountCmd = strdup(value);
   } else if (strcmp(key, "chmodPath") == 0) {
       config->chmodPath = strdup(value);
   } else if (strcmp(key, "ddPath") == 0) {
```

Listing 7. Source code modification in `UdiRootConfig.c`.

2) RPM Spec file customization: The Shifter source distribution provided two RPM Spec files that build binary RPM packages for Shifter (`shifter.spec`) and an RPM package for Linux kernel modules (`shifter_cle6_kmod_deps.spec.cray`). The original Spec files were written to build RPM packages that install non-relocatable files. The default installation path of the RPM packages is `/usr`, but we wanted to install them to `/opt`, as a best practice to follow our existing third-party software directory structure. Hence, we made these RPM files relocatable by adding `Prefix` key to the Spec files. With guidance from Cray, we also made some changes to the Spec file to suit CLE5 build environment (see listing 8).

```
diff --git a/shifter.spec b/shifter.spec
index edc0a5d..3e7a418 100644
--- a/shifter.spec
+++ b/shifter.spec
@@ -25,6 +25,7 @@ URL: https://github.com/NERSC/shifter
 Packager: Douglas Jacobsen <dmjacobsen@lbl.gov>
 Source0: %{name}-%{version}.tar.gz
 BuildRoot: %{_tmppath}/%{name}-%{version}-%{release}-root
+Prefix: /usr

 %description
 Shifter enables container images for HPC. In a nutshell, Shifter
@@ -55,6 +56,8 @@ BuildRequires: json-c json-c-devel
 BuildRequires: pam-devel
 BuildRequires: libcap-devel
 BuildRequires: python
+Prefix: /usr
+Prefix: /etc
 %endif

 %description runtime
@@ -75,7 +78,11 @@ Shifter.

 %package imagegw
 Summary: Image Manager/Gateway for Shifter
-Requires(pre): shadow-utils
+Requires(pre): shadow
+Group: System Environment/Base
+Prefix: /usr
+Prefix: /etc
+Prefix: /var
 %if 0%{!?_without_systemd:1}
 %{systemd_requires}
 %endif
@@ -104,6 +111,7 @@ use with Shifter.
 Summary: SLURM Spank Module for Shifter
 BuildRequires: slurm-devel
 BuildRequires: xfsprogs
+Prefix: /usr

 %description slurm
 Shifter enables container images for HPC. In a nutshell, Shifter
@@ -199,8 +207,8 @@ pip install celery
 %defattr(-, root, root)
 %doc AUTHORS LICENSE NEWS README* udiRoot.conf.example
 %attr(4755, root, root) %{_bindir}/shifter
-%config(noreplace missingok) %verify(not filedigest mtime size) %{_sysconfdir}/shifter_etc_files/passwd
-%config(noreplace missingok) %verify(not filedigest mtime size) %{_sysconfdir}/shifter_etc_files/group
+%config(noreplace missingok) %verify(not mtime size) %{_sysconfdir}/shifter_etc_files/passwd
+%config(noreplace missingok) %verify(not mtime size) %{_sysconfdir}/shifter_etc_files/group
 %config(noreplace) %{_sysconfdir}/shifter_etc_files/nsswitch.conf
 %{_bindir}/shifterimg
 %{_bindir}/activate_gpu_support.sh
```

Listing 8. Customization of `shifter.spec` file.

The provided Linux kernel module RPM Spec file was written to build Linux kernel modules for CLE6 compute kernel. As Blue Waters operates with a CLE5 kernel, We rewrote the Spec file to build Linux kernel modules for the CLE5 compute kernel. (See listing 9).

```
default/:/software/shifter-16.08.3/extra # cat shifter_cle5_gem-c_kmod.spec
Name:       shifter_cle5_kmod_deps-%(uname -r | sed 's/gem_s/gem_c/g')
Version:    1.0
Release:    3
License:    GPL
BuildRequires: kernel-source kernel-syms
BuildRoot: %{_tmppath}/%{name}-%{version}-build
Summary:   kernel mod deps for cle5
Group:     System Environment/Base
Prefix: /lib
%description
xfs, ext4 and deps
%prep
%build
%define KVER %(uname -r | sed 's/-cray.*//g')
rsync -raql /usr/src/linux-%{KVER} %{buildroot}
cd %{buildroot}/linux-%{KVER}
if [ -e "arch/x86/configs/cray_gem_c_defconfig" ]; then
  cp arch/x86/configs/cray_gem_c_defconfig .config
else
  cp /proc/config.gz ./
  gunzip config.gz
  mv config .config
fi
echo "CONFIG_BLK_DEV_LOOP=m" >> .config
echo "CONFIG_EXT4_FS=m" >> .config
echo "CONFIG_CRAMFS=m" >> .config
echo "CONFIG_SQUASHFS=m" >> .config
echo "CONFIG_JBD2=m" >> .config
echo "CONFIG_FS_MBCACHE=m" >> .config
echo "CONFIG_XFS_FS=m" >> .config
echo "CONFIG_XFS_QUOTA=y" >> .config
echo "CONFIG_XFS_DMAPI=m" >> .config
```

```
echo "CONFIG_XFS_POSIX_ACL=y" >> .config
echo "CONFIG_XFS_RT=y" >> .config
yes "" | make oldconfig
make modules_prepare
make modules
%install
%define KVER %(uname -r | sed 's/-cray.*//g')
cd %{buildroot}/linux-%{KVER}
make modules_install INSTALL_MOD_PATH=%{buildroot}
%post
depmod -a
%postun
depmod -a
%files
%define _unpackaged_files_terminate_build 0
%defattr(-,root,root)
/lib/modules/%{KVER}*/kernel
```

Listing 9. `shifter_cle5_gem-c_kmod.spec`

3) Building RPM Packages: The build procedure of Shifter RPM packages is given in listing 10.

```
default/:/software/shifter-16.08.3 # ./autogen.sh
default/:/software/shifter-16.08.3 # cd ..
default/:/software # tar czvf shifter-16.08.3.tar.gz
     shifter-16.08.3
default/:/software # cp shifter-16.08.3.tar.gz
     /usr/src/packages/SOURCES
default/:/software # cd shifter-16.08.3
default/:/software/shifter-16.08.3 # rpmbuild -ba
     shifter.spec
default/:/software/shifter-16.08.3 # cd extra
default/:/software/shifter-16.08.3/extra # rpmbuild -
     bb shifter_cle5_gem-c_kmod.spec
```

Listing 10. Build procedure of Shifter RPM packages.

4) Installing RPM Packages: The install procedure of Shifter RPM packages is given in listing 11. With addition of the `Prefix` key in the RPM Spec files, we were able to install the files at the desired location. These RPM packages were copied from the TDS over to Blue Waters for installation on Blue Waters' `/dsl`.

```
default/:/usr/src/packages/RPMS/x86_64 # rpm -ivh --
     prefix=/opt/cray/shifter/16.08.3
     shifter_cle5_kmod_deps-3.0.101-0.46.1_1
     .0502.8871-cray_gem_c-1.0-3.x86_64.rpm
default/:/usr/src/packages/RPMS/x86_64 # rpm -ivh --
     prefix=/opt/cray/shifter/16.08.3 shifter
     -16.08.3-1.nersc.x86_64.rpm
shifter/:/usr/src/packages/RPMS/x86_64 # rpm -ivh --
     relocate /usr=/opt/cray/shifter/16.08.3 shifter-
     imagegw-16.08.3-1.nersc.x86_64.rpm
shifter/:/usr/src/packages/RPMS/x86_64 # rpm -ivh --
     prefix=/opt/cray/shifter/16.08.3 shifter-runtime
     -16.08.3-1.nersc.x86_64.rpm
```

Listing 11. InstallSing hifter RPM packages.

Installation of Shifter RPM packages create a new user 'shifter' and group 'shifter' if they are not already present in the install host's `/etc/passwd` and `/etc/group` files. We use this user to run the Shifter image manager gateway service and the UDI files stored by the image manager in the Lustre shared file system would be owned by this user. In order to be granted write access permission to the file system, the same credential has to be recognized on the Lustre file system server nodes. We use a centralized LDAP service to achieve consistency of user identity across systems. The 'shifter' user and group were created in LDAP before installing the Shifter RPM packages to avoid re-creation of the same credential locally in the install host's `/etc/passwd` and `/etc/group` files.

C. MongoDB

MongoDB [9] is an open-source distributed database designed to scale horizontally across multiple servers and to provide high availability. Shifter uses MongoDB to store the metadata of available container images and their operational state: whether the image is in download state, in conversion state or ready to use. We obtained MongoDB RPM packages from *https://repo.mongodb.org*. The procedure to install MongoDB RPM packages is given in listing 12.

```
default/:/software/mongodb # wget --no-check-
     certificate https://www.mongodb.org/static/pgp/
     server-3.4.asc
default/:/software/mongodb # rpm --import server-3.4.
     asc
default/:/software/mongodb # rpm -ivh --prefix=/opt/
     mongodb/3.4.7 mongodb-org-3.4.7-1.suse11.x86_64.
     rpm mongodb-org-server-3.4.7-1.suse11.x86_64.rpm
     mongodb-org-shell-3.4.7-1.suse11.x86_64.rpm
     mongodb-org-mongos-3.4.7-1.suse11.x86_64.rpm
     mongodb-org-tools-3.4.7-1.suse11.x86_64.rpm
```

Listing 12. Install procedure of MongoDB

D. Redis

Redis [10] is an open source in-memory key-value data structure store used as a database, cache and message broker. Shifter uses Redis as the message broker for the *Celery* queue system. The install procedure of Redis is given in listing 13.

```
default/:/software # wget http://download.redis.io/
     releases/redis-3.2.8.tar.gz
default/:/software # tar xvf redis-3.2.8.tar.gz
default/:/software # cd redis-3.2.18
default/:/software/redis-3.2.8 # export CC=gcc
default/:/software/redis-3.2.8 # make distclean
default/:/software/redis-3.2.8 # make PREFIX=/opt/
     redis/3.2.8 install
```

Listing 13. Install procedure of Redis

E. Python

The core image manager gateway component of Shifter is a RESTful service written in the Python [11] language. The Shifter installation provided a list of Python modules required to support the functionality of the image manager.

```
default/:/opt/shifter/16.08.3/share/shifter # cat
     requirements
celery
pymongo
flask
redis
gunicorn
pylint
```

Listing 14. Python modules required by Shifter image manager gateway.

Celery [12] - Asynchronous and distributed task queue system to service user requests. *Celery* provides better scalability to multiple requests through queue and dispatch to a distributed pool of workers.

Flask [13] - Web framework that provides RESTful API as the interfacing layer between user requests and underlying image manager. Use of RESTful API replaces a locally installed Docker engine (needed in Shifter V1) to interact with Docker registry.

pymongo [14] - Python API to interface with MongoDB.

redis [15] - Python API to interface with Redis.

gunicorn [16] - Web server gateway interface to work with *Flask*.

We used the *pip* [17] tool to download these Python modules from *PyPI* [18] and installed them under the `/opt/cray/shifter/16.08.3/imagegw_venv` directory using *virtualenv* [19] to automatically resolve the dependencies, creating an isolated Python environment to avoid overwriting existing Python modules installed on the system. As *PyPI* now enforces client connection with SSL enabled, a newer version of Python (2.7.13) was installed to obtain SSL supported *pip*.

```
default/:/software # tar xvf Python-2.7.13.tgz; cd Python-2.7.13
default/:/software/Python-2.7.13 # ./configure --prefix=/opt/python/2.7.13
default/:/software/Python-2.7.13 # make
default/:/software/Python-2.7.13 # make install

default/:/software # tar xvf virtualenv-15.1.0.tar.gz
default/:/software # cd /opt/cray/shifter/16.08.3
default/:/opt/cray/shifter/16.08.3 # /software/virtualenv-15.1.0/virtualenv.py imagegw_venv --python=/opt/python/2.7.13/bin/python

default/:/ # source /opt/cray/shifter/16.08.3/imagegw_venv/bin/activate
(imagegw_venv) default/:/ # pip install -r /opt/cray/shifter/16.08.3/share/shifter/requirements
(imagegw_venv) default/:/ # deactivate
```

Listing 15. Install procedure of Python modules.

F. Munge

Munge [20] service provides the authentication mechanism for communication between compute nodes invoking Shifter and the service node hosting the Shifter image manager gateway service. The `munge` daemon is provided by the *cray-munge* RPM package, which comes with a `/etc/munge.key` file. All nodes including the image manager node must use the same key for authentication. *cray-munge* is included as part of the CLE5 software stack installation.

G. Post Installation

The installation of Shifter software stack in `xtopview` installed some files under `/var` space in `/dsl`. After exiting from `xtopview`, we copied the Shifter's files from `/dsl`'s `/var` space into the persistent `/var` space of the service node where the Shifter services would be running.

```
default/:/ # exit
boot:~ # cd /rr/current/var/lib
boot:/rr/current/var/lib # cp -Rp mongo /snv/<nid_id>/var/lib
boot:/rr/current/var/lib # cd ../log
boot:/rr/current/var/log # cp -Rp mongodb shifter_imagegw* /snv/<nid_id>/var/log
boot:/rr/current/var/log # cd ../run
boot:/rr/current/var/run # cp -Rp mongodb /snv/<nid_id>/var/run
```

Listing 16. Copying files from `/dsl` `/var` space into service node's persistent `/var` space

III. Configuration of Shifter Software Stack

The following section describes the configuration of the Shifter V16 software stack on Blue Waters.

A. Shifter Image Manager Gateway

The `imagemanager.json` file, written in *JSON* format, configures how the Shifter image manager connects to MongoDB, Redis, Munge, the Docker registry, and provides locations to store original Docker files, temporary files, and the final UDI files. We specialized this file to the utility class in `/dsl`, as only the service nodes from the utility class eligible to host Shifter services would use this file.

```
class/utility:/ # xtspec -c utility /etc/shifter/imagemanager.json
class/utility:/ # cat /etc/shifter/imagemanager.json
{
   "WorkerThreads":8,
   "DefaultLustreReplication": 1,
   "DefaultOstCount": 16,
   "DefaultImageLocation": "registry-1.docker.io",
   "DefaultImageFormat": "squashfs",
   "PullUpdateTimeout": 300,
   "ImageExpirationTimeout": "90:00:00:00",
   "MongoDBURI":"mongodb://shifteradmin:<P@55w0rd>@localhost/Shifter?authMechanism=SCRAM-SHA-1",
   "MongoDB":"Shifter",
   "Broker":"redis://:<P@55w0rd>@localhost/",x`'
   "CacheDirectory": "/mnt/c/scratch/system/shifter/images/cache/",
   "ExpandDirectory": "/mnt/c/scratch/system/shifter/images/expand/",
   "Locations": {
      "registry-1.docker.io": {
         "remotetype": "dockerv2",
         "authentication": "http"
      }
   },
   "Platforms": {
      "bluewaters": {
         "mungeSocketPath": "/var/run/munge/munge.socket.2",
         "accesstype": "local",
         "admins": ["root"],
         "usergroupService": "local",
         "local": {
            "imageDir": "/mnt/c/scratch/system/shifter/images"
         }
      }
   }
}
```

Listing 17. `/etc/shifter/imagemanager.json`

B. *Shifter Runtime*

The `udiRoot.conf` file defines the runtime environment of Shifter. Listing 18 lists the parameters that we explicitly configured on Blue Waters, leaving other parameters as default. This file is read by all compute nodes when Shifter is invoked, it is placed in the default class of `/dsl`.

```
default/:/ # cat /etc/shifter/udiRoot.conf | grep -v ^# | grep -v ^$
udiMount=/var/udiMount
loopMount=/var/udiLoopMount
imagePath=/mnt/abc/scratch/system/shifter/images
udiRootPath=/opt/cray/shifter/16.08.3
sitePreMountHook=/opt/cray/shifter/16.08.3/sbin/premount.sh
optUdiImage=/opt/cray/shifter/16.08.3/lib64/shifter/opt/udiImage
etcPath=/etc/shifter/shifter_etc_files
autoLoadKernelModule=1
mountUdiRootWritable=1
maxGroupCount=31
mountCmd=/opt/cray/shifter/16.08.3/lib64/shifter/mount
modprobePath=/sbin/modprobe
insmodPath=/sbin/insmod
cpPath=/bin/cp
mvPath=/bin/mv
chmodPath=/bin/chmod
ddPath=/bin/dd
rootfsType=ramfs
kmodBasePath=/opt/cray/shifter/16.08.3/modules
siteFs=/home:/home
siteEnv=SHIFTER_RUNTIME=1
siteEnvAppend=PATH=/opt/udiImage/bin
imageGateway=http://shifter:5000 http://shifter:5001 http://shifter:5002
system=bluewaters
defaultImageType=docker
siteResources=/opt/shifter/site-resources
allowLibcPwdCalls=1
```

Listing 18. `/etc/shifter/udiRoot.conf`

The `sitePreMountHook` parameter in `udiRoot.conf` file points to a script to define customized mount points that can be setup in the Shifter runtime container. Using this script, we are able to bind volumes from the compute host into the container, e.g. the user home directory and mount points from Lustre file systems, and `/opt`.

```
default/:/ # cat /opt/cray/shifter/16.08.03/sbin/premount.sh
#!/bin/sh
set -e
mkdir -p mnt/c
mount --bind /mnt/c mnt/c
mkdir -p mnt/a
mount --bind /mnt/a mnt/a
mkdir -p mnt/b
mount --bind /mnt/b mnt/b
mkdir -p ufs
mkdir -p var/opt/cray/alps
mount --bind /ufs ufs
mount --bind /var/opt/cray/alps var/opt/cray/alps
ln -s mnt/c/scratch scratch
ln -s mnt/a/u   u
ln -s mnt/a/sw  sw
ln -s sw/cm      cm
ln -s mnt/b/projects projects
mkdir -p dsl/opt
mount --bind /dsl/opt dsl/opt
exit 0
```

Listing 19. `/opt/cray/shifter/16.08.3/sbin/premount.sh`

C. *Redis Configuration*

The Redis source package provided a sample `redis.conf` file to configure Redis. We copied this file into the `/etc/shifter` directory, and modified only the parameters listed in listing 20.

```
default/:/ # grep -E "^dir|^requirepass" /etc/shifter/redis.conf
dir /var/lib/redis
requirepass P@55w0rd
```

Listing 20. Parameters modified in `/etc/shifter/redis.conf` file as required for Blue Waters' operation environment.

D. *Service Startup Scripts*

Shifter requires multiple services to be running in order to be functional: Shifter image manager gateway, MongoDB, Redis and Munge. We grouped all unused service nodes on Blue Waters into an "utility" class, serving as a resource pool to host software services like Shifter. We placed all services required by Shifter to start on one of the utility class service nodes. All these services use *System V* style init scripts located in the `/etc/init.d` directory: `munge`, `mongod`, `redisd` and `shifter-imagegw`. These services can be started using the trivial way in the order listed in listing 21.

```
shifter:~ # service munge start
shifter:~ # service mongod start
shifter:~ # service redisd start
shifter:~ # service shifter-imagegw start
```

Listing 21. Intializing Shifter image manager gateway and the dependent peripheral services.

When started, the `shifter-imagegw` init script triggers a Python script (See listing 22) to launch a pool of *Celery* workers. These *Celery* workers are standby service threads ready to handle Shifter requests from compute nodes.

```
default/:/ # cat /opt/cray/shifter/16.08.3/sbin/shifter-imagegw
#!/bin/bash
if [ -z ${ROOT_TREE} ]; then
  ROOT_TREE='/opt/cray/shifter/16.08.3'
fi
if [ -z ${PYTHON_VENV} ]; then
  PYTHON_VENV='imagegw_venv'
fi
if [ -z ${SHIFTER_SYSTEM_NAME} ]; then
  SHIFTER_SYSTEM_NAME='bluewaters'
fi
QA="${SHIFTER_SYSTEM_NAME}"
cd ${ROOT_TREE}
source ${PYTHON_VENV}/bin/activate
echo "Starting Celery Queue $QA"
celery -A shifter_imagegw.imageworker worker -Q $QA --loglevel=WARNING -n worker.queue.$QA -E --concurrency=24 &
echo "Starting imagegw API"
python lib64/shifter/imagegwapi.py &
python lib64/shifter/imagegwapi1.py &
python lib64/shifter/imagegwapi2.py &
```

```
wait
```

Listing 22. `shifter-imagegw` init script.

E. Security

We secured the authentication to MongoDB and Redis databases by setting up a password, to prevent access to the databases by unprivileged users.

1) Securing Redis: A password can be set for Redis database using the `requirepass` parameter in `redis.conf` file (see listing 20). The `redisd` init script is configured to read from the `redis.conf` file where the password is stored. The permission of the `redis.conf` file was changed to be accessible by root only.

```
default/:/ # chown root: /etc/shifter/redis.conf
default/:/ # chmod 640 /etc/shifter/redis.conf
default/:/ # chown root: /etc/init.d/redisd
default/:/ # chmod 750 /etc/shifter/redisd
```

Listing 23. Securing Redis

2) Securing MongoDB: By default, MongoDB is configured to listen to connections from localhost only. To better secure MongoDB, we created a *mongodbadmin* user and set a password in the admin database.

```
user@shifter:~> /opt/mongodb/3.4.7/bin/mongo
> use admin
> > db.createUser(
... {
... user: "mongodbadmin",
... pwd: "<P@55w0rd>",
... roles: [ { role: "root", db: "admin" } ]
... }
... )
```

Listing 24. Creating *mongodbadmin* user and set a password in MongoDB's admin database.

After restarting the MongoDB service with the `--auth` argument, we login again to the MongoDB's admin database using the *mongodbadmin* credential to create a *shifteradmin* credential for the *Shifter* database.

```
user@shifter:~> /opt/mongodb/3.4.7/bin/mongo
> use admin
switched to db admin
> db.auth("mongodbadmin", "<P@55w0rd>")
1
> use Shifter
switched to db Shifter
> db.createUser(
... {
... user: "shifteradmin",
... pwd: "<P@55w0rd>",
... roles: [ { role: "dbOwner", db: "Shifter" } ]
... }
... )
> exit
```

Listing 25. Creating *shifteradmin* credential for the *Shifter* database in MongoDB.

3) Shifter Image Manager: With password authentication configured in Redis and MongoDB, the `MongoDBURI` and `Broker` parameters in the `imagemanager.json` configuration file were updated to use the secured credentials for connections to Redis and MongoDB. In addition, we also noted that it is not recommended to launch *Celery* worker threads as root, hence the `shifter-imagegw` init script was edited to start the image manager gateway service as the 'shifter' user, and it requires read access to `imagemanager.json` file. As passwords are passed as clear text in the `imagemanager.json` file, the ownership and permission of the `imagemanager.json` file was changed to be accessible by root and 'shifter' user only.

```
default/:/ # chown shifter:root /etc/shifter/imagemanager.json
default/:/ # chmod 660 /etc/shifter/imagemanager.json
```

Listing 26. Securing Shifter image manager gateway.

IV. Integration with Workload Manager

Figure 1 illustrates the integration of Shifter jobs with Blue Waters' workload manager. The Shifter source distribution provided the integration scripts to work with Torque Resource Manager. These scripts were written for Shifter V1, but not updated to work in Shifter V16. We modified these scripts to integrate Shifter V16 with Torque Resource Manager on Blue Waters. In the general HPC cluster use case (including on a Cray system), a Shifter container-based application can be launched on compute nodes by invoking the `shifter` command line interface. For Cray systems that use ALPS, Shifter can be configured to invoke `setupRoot` through job prologue scripts to setup container environment on all compute nodes before beginning execution of the job script, such that usual ALPS syntax can be used to launch container based applications onto the compute nodes, without causing additional overhead induced by `shifter` CLI. Both methods require Munge service to be running on the compute nodes in order to be able to communicate with the Shifter image manager gateway. A generic resource `shifter16` was setup in the resource and workload manager, in which upon request at job submission, the Cray login node (a.k.a. mom node) allocated to the job will invoke a series of prologue scripts (see listing 28) to start Munge service on all compute nodes allocated to the job.

```
user@h2ologin:~> qsub -l nodes=1:ppn=16,gres=shifter16 jobscript.sh
```

Listing 27. Requesting for `gres=shifter16` generic resource in job submission.

```
mom:/var/spool/torque/mom_priv # awk "/Shifter ver. 16/,/END Shifter/" prologue
# Request for Shifter ver. 16.08.3
if [ $(/opt/torque/default/bin/qstat -f ${BATCH_JOB_ID} | grep Resource_List.gres | grep -c '\bshifter16\b') -gt 0 ];then
   echo "In Torque Shifter prologue batchID: ${BATCH_JOB_ID}"
```

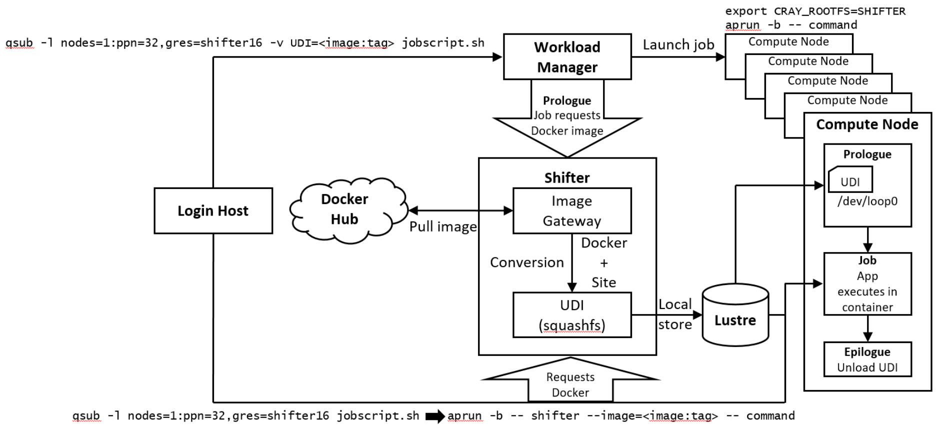

Figure 1. Architecture of Shifer implementation on Blue Waters

```
    echo "In Torque Shifter prologue batchID: ${BATCH_JOB_ID}" >> /scratch/system/shifter/shifter16.log
    shifter_prologue=/opt/cray/shifter/16.08.3/wlm/torque/cray-shifter-prologue
    if [[ -x $shifter_prologue ]]; then
       $shifter_prologue ${BATCH_JOB_ID} $2 $3 ${RESV_ID} ${NIDS}
    fi
fi
# END Shifter

mom:/opt/cray/shifter/16.08.3/wlm/torque # cat cray-shifter-prologue
#!/bin/bash
SHIFTER_JOBID=$1
SHIFTER_USER=$2
SHIFTER_GROUP=$3
SHIFTER_RESVID=$4
SHIFTER_NIDS=$5
ROOT=/opt/cray/shifter/16.08.3
PROLOGUE=$ROOT/wlm/udiRoot-prologue
QGETENV=$ROOT/wlm/torque/qgetenv
PCMDCMD="/opt/cray/nodehealth/default/bin/pcmd"
id $SHIFTER_USER
cp /var/run/nscd/passwd /scratch/system/shifter/jobs/passwd.${SHIFTER_JOBID}
cp /var/run/nscd/group /scratch/system/shifter/jobs/group.${SHIFTER_JOBID}
$PCMDCMD -r -q -n ${SHIFTER_NIDS} "/dsl/usr/bin/chroot /dsl sh /opt/cray/shifter/16.08.3/wlm/torque/cray-shifter-extra-service start ${SHIFTER_JOBID}"
if [[ ! -x $PROLOGUE ]]; then
      # shifter/udiRoot is not installed. Nothing to do.
      exit 0
fi
SHIFTER_ENV_VAR=$( $QGETENV $SHIFTER_JOBID UDI )
if [[ -z $SHIFTER_ENV_VAR ]]; then
      # Job did not ask for shifter resources.
      exit 0
fi
$PROLOGUE $SHIFTER_JOBID $SHIFTER_USER $SHIFTER_GROUP $SHIFTER_RESVID DOCKER $SHIFTER_ENV_VAR
RET=$?
exit $RET

mom:/opt/cray/shifter/16.08.3/wlm/torque # cat cray-shifter-extra-service
#!/bin/bash

opts=$1
jobid=$2
kmodpath=/opt/cray/shifter/16.08.3/modules/kernel
case $opts in
   start)
      echo Starting MUNGE service.
      /etc/init.d/munge start
      cp /scratch/system/shifter/jobs/passwd.$jobid /var/run/nscd/passwd
      cp /scratch/system/shifter/jobs/group.$jobid /var/run/nscd/group
      echo Starting NSCD service.
      /etc/init.d/nscd start
      /sbin/insmod $kmodpath/drivers/block/loop.ko max_loop=128
      /sbin/insmod $kmodpath/fs/squashfs/squashfs.ko
      ;;
   stop)
      echo Stopping NSCD service.
      /etc/init.d/nscd stop
      echo Stopping MUNGE service.
      /etc/init.d/munge stop
      rm /var/run/nscd/passwd /var/run/nscd/group /var/run/nscd/services
      ;;
   *)
      echo Unknown option
esac
```

Listing 28. Prologue scripts triggered by `gres=shifter16`.

A. `shifter` *Command Line Interface*

If using the `shifter` CLI to launch Shifter tasks in a job script, the job submission requires only the `gres=shifter16` argument to be passed to the job submission `qsub` command. Like a regular job script, `aprun` command is used to invoke the `shifter` CLI, together with the `--image=<image:tag>` argument to specify which Docker image to use, followed by the application command to be executed in the container compute environment. For example (see listing 29), specifying `--image=centos:latest` would trigger Shifter image manager to download the *centos:latest* docker image from the Docker registry, convert it into UDI, and mount it on the compute node, then proceed to execute the following command in the container environment.

```
aprun -b -- shifter --image=<image:tag> -- command
```

Listing 29. `shifter` CLI

B. *Using `setupRoot` in Prologue*

Alternatively, if using `setupRoot` to setup the container environment on compute nodes before executing the job script, the `-v UDI=<image:tag>` argument is required to be passed to `qsub` command in addition to `gres=shifter16`.

```
user@h2ologin:~> qsub -l nodes=1:ppn=16,gres=shifter16 -v UDI=<image:tag> myscript.sh
```

Listing 30. Requesting for `shifter16` generic resource and `UDI` in job submission.

As seen in listing 28, when `UDI` argument is passed to `qsub` command, `setupRoot` is triggered by the `udiRoot-prologue` script (see listing 31) to download the Docker image indicated by the `<image:tag>` label from Docker registry onto the mom node, convert it into UDI and mount it on all compute nodes allocated to the job. After completion of all prologue scripts, the mom node then proceeds to begin execution of the job script. In the job script, by setting `CRAY_ROOTFS=SHIFTER` environment variable, regular `aprun` syntax can be used to launch container-based applications onto the compute nodes' container environment. This method of job submission preserves the same working shell environment from the mom node onto the compute node. When using `aprun` with `shifter` CLI, `PATH` and `LD_LIBRARY_PATH` environment variables from the mom node are not passed to the `shifter` tasks executing in the container.

```
#!/bin/bash
jobId="$1"
user="$2"
group="$3"
resId="$4"
udiRootType="$5"
udiRootValue="$6"
shift 6
PATH=${PATH}:/opt/cray/alps/default/bin
nodeContext=""
udiRootPath=/opt/cray/shifter/16.08.3
mode="alps"
nodelist=""
tasksPerNode=1
volumes=()
while getopts ":m:n:N:v:" opt; do
   case "${opt}" in
      m)
         mode="${OPTARG}"
         if [[ -n "$mode" && "$mode" == "local" ]]; then
            nodeContext="";
         fi
         ;;
      n)
         nodelist="${OPTARG}"
         ;;
      N)
         tasksPerNode="${OPTARG}"
         ;;
      v)
         volumes+=(${OPTARG})
         ;;
      \?)
         echo "Invalid option: -${OPTARG}" >&2
         exit 1
         ;;
      :)
         echo "Option -${OPTARG} requires an argument" >&2
         exit 1
         ;;
   esac
done
die() {
   local msg
   msg="$1"
   echo "$msg" 1>&2
   exit 1
}
[[ -n "$jobId" ]] || die "Job ID is undefined"
[[ -n "$user" ]] || die "user is undefined"
[[ -n "$group" ]] || die "group is undefined"
[[ -n "$udiRootType" ]] || die "udi image type is undefined"
[[ -n "$udiRootValue" ]] || die "udi image value is undefined"
userUid=$( id -u "$user" )
groupGid=$( getent group "$group" | awk -F ’:’ ’{print $3}’ )
[[ -n "$userUid" ]] || die "user Uid is unknown"
[[ -n "$groupGid" ]] || die "group Gid is unknown"
jobEnv=()
entrypoint=""
udiRootId=""
echo "Initializing udiRoot, please wait."
if [[ "$udiRootType" == "DOCKER" ]]; then
   echo "Retrieving Docker Image"
   status=$(su - $user "$udiRootPath/bin/shifterimg pull $udiRootValue" | awk {’print $NF’})
   if [[ "$status" == "READY" ]]; then
      data=$(su - $user "$udiRootPath/bin/shifterimg lookup $udiRootValue")
      ret=$?
   else
      echo "Failed to download docker image: $udiRootValue" 2>&1
      exit 1
   fi
   for item in $data; do
      if [[ "$item" == "ENV:"* ]]; then
         envItem=$(echo "$item" | cut -c 5-)
         jobEnv+=($envItem)
      elif [[ "$item" == "ENTRY:"* ]]; then
         entrypoint=$(echo "$item" | cut -c 7-)
      else
         udiRootId=$item
      fi
   done
   if [[ -z "$udiRootId" || $ret -ne 0 ]]; then
      echo "Failed to get udi image: $udiRootValue" 1>&2
      exit 1
   fi
else
   echo "Unknown image type: $udiRootType" 1>&2
   exit 1
fi
umask 066
datadir="/var/run/shifter/jobs/$user/$jobId"
mkdir -p "$datadir"
umask 022
homeDir=$( eval "echo ~$user" )
pubKey="$homeDir/.shifter/id_rsa.pub"
if [[ -r $pubKey ]]; then
   sshPubKey=$( cat $pubKey )
else
   ssh-keygen -t rsa -f "$datadir/id_rsa" -N ’’ >/dev/null 2>&1
   chown "$user" "$datadir/id_rsa" "$datadir/id_rsa.pub"
```

```
    chmod 600 "$datadir/id_rsa" "$datadir/id_rsa.pub"
    sshPubKey=$( cat "$datadir/id_rsa.pub" )
fi
envFile="$datadir/env";
for envItem in "${jobEnv[@]}"; do
    echo "$envItem" >> "$envFile"
done
if [[ -n "$entrypoint" ]]; then
    echo "$entrypoint" > "$datadir/entrypoint"
fi
reservation=""
if [[ "$mode" == "local" ]]; then
    reservation="local";
elif [[ -n "$BASIL_RESERVATION_ID" ]]; then
    reservation="$BASIL_RESERVATION_ID"
else
    reservation="$resId"
fi
[[ -z "$reservation" ]] && die "Failed to identify job reservation"
job_nodelist="$datadir/nodelist"
if [[ "$reservation" == "local" ]]; then
    hostname > "$job_nodelist"
else
    apstat -rvvv -R "$reservation" | awk '/^[ ]*PE / { printf "nid%05d\n", $6 }' | sort > "$job_nodelist"
fi
xtxqtcmd_log="$datadir/log_start"
xtxqtcmd="/opt/cray/nodehealth/default/bin/xtxqtcmd"
[[ -x "$xtxqtcmd" ]] || die "Could not find xtxqtcmd. Exiting"
## get list of unique nodes to run setupRoot on
unique_nodes="$datadir/unique_nodes"
cat "$job_nodelist" | sort -u > "$unique_nodes"
## minimize nodelist for putting hosts file on the compute node
if [[ "$mode" == "local" ]]; then
    minNodes=$( /opt/slurm/default/bin/scontrol show hostnames "$nodelist" | awk -v taskCount="$tasksPerNode" '{ print $1 "/" taskCount }' | xargs )
else
    minNodes=$( cat "$job_nodelist" | sort | uniq -c | awk '{ print $2 "/" $1 }' | xargs )
fi
echo $minNodes >> $xtxqtcmd_log
cmdStr="/dsl/usr/bin/chroot /dsl ${udiRootPath}/sbin/setupRoot \"$udiRootType\" \"$udiRootId\" -s \"$sshPubKey\" -u \"$user\" -U \"$userUid\" -G \"$groupGid\" -N \"$minNodes\" -V"
for volume in "${volumes[@]}"; do
    cmdStr="$cmdStr -v \"$volume\""
done
ok=0
expected=0
if [[ "$mode" == "local" ]]; then
    echo $cmdStr >> $xtxqtcmd_log
    /bin/sh -c "$cmdStr"
    [[ $? -eq 0 ]] && ok=1
    expected=1
else
    echo "$xtxqtcmd $unique_nodes $cmdStr" >> $xtxqtcmd_log
    "$xtxqtcmd" "$unique_nodes" "$cmdStr" >> $xtxqtcmd_log 2>&1
    ok=$( grep "Reply (complete) from .* exit code: 0" $xtxqtcmd_log | wc -l )
    expected=$( cat "$unique_nodes" | wc -l )
fi
ret=0
if [[ "$ok" -eq "$expected" ]]; then
    echo "udiRoot Start successful"
else
    echo "udiRoot Start FAILURE, $ok of $expected responses"
    ret=1
fi
exit $ret
```

Listing 31. `udiRoot-prologue`

By default on Blue Waters, when `aprun` is invoked in a job script, applications are placed to start in an environment where `/dsl` is set as the relative root. `/dsl` is set as the default root in the CLE compute node root runtime environment (CNRTE) configuration file (`roots.conf`). Alternative root can be defined in `roots.conf` file using `<ROOT_NAME=/absolute/path/to/root>` format. If `CRAY_ROOTFS` environment variable is defined in a job script and matches one of the available `<ROOT_NAME>` in `roots.conf` file, `aprun` will launch applications on the compute node with root set at the path defined by the corresponding `<ROOT_NAME>` in `roots.conf` file. Since Shifter was installed at `/dsl` root on Blue Waters, the `udiMount=/var/udiMount` parameter defined in `udiRoot.conf` file instructs `setupRoot` to mount Docker image at `/dsl/var/udiMount` on the compute node. Hence, an alternative root `SHIFTER=/dsl/var/udiMount` was added to the `roots.conf` file.

```
default/:/ # grep SHIFTER /etc/opt/cray/cnrte/roots.conf
SHIFTER=/dsl/var/udiMount
```

Listing 32. `/etc/opt/cray/cnrte/roots.conf`

With this configuration in place, and `CRAY_ROOTFS=SHIFTER` environment variable defined in job script, ALPS tasks would be executed at `/dsl/var/udiMount` root on the compute node where the Docker image is mounted (see listing 33). This condition is not required for jobs that use the `shifter` CLI.

```
export CRAY_ROOTFS=SHIFTER
aprun -b -- command
```

Listing 33. Launching ALPS task in Shifter container environment, when passing `gres=shifter16` and `-v UDI=<image:tag>` to `qsub`.

C. *Epilogue*

At the end of a Shifter job, epilogue scripts are executed from the mom node to clean up the container environment on the compute nodes (see listing 34). `unsetupRoot` is triggered by the `udiRoot-epilgoue` script to unmount the container image from compute nodes (if `-v UDI=<image:tag>` argument is passed to `qsub`). The epilogue scripts shutdown the Munge service on the compute nodes.

```
mom:/var/spool/torque/mom_priv # cat epilogue
...
if [ $(/opt/torque/default/bin/qstat -f ${BATCH_JOB_ID} | grep Resource_List.gres | grep -c '\bshifter16\b') -gt 0 ];then
```

```
   shifter_epilogue=/opt/cray/shifter/16.08.3/wlm/torque/cray-shifter-epilogue
   if [[ -x $shifter_epilogue ]]; then
      $shifter_epilogue ${BATCH_JOB_ID} $2 $3 ${RESV_ID} ${NIDS}
   fi
fi
...

mom:/opt/cray/shifter/16.08.3/wlm/torque # cat cray-shifter-epilogue
#!/bin/bash
SHIFTER_JOBID=$1
SHIFTER_USER=$2
SHIFTER_GROUP=$3
ROOT=/opt/cray/shifter/16.08.3
EPILOGUE=$ROOT/wlm/udiRoot-epilogue
QGETENV=$ROOT/wlm/torque/qgetenv
SHIFTER_RESVID=$4
SHIFTER_NIDS=$5
if [[ ! -x $EPILOGUE ]]; then
      exit 0
fi
SHIFTER_ENV_VAR=$( $QGETENV $SHIFTER_JOBID UDI )
if [[ -n $SHIFTER_ENV_VAR ]]; then
      $EPILOGUE $SHIFTER_JOBID $SHIFTER_USER $SHIFTER_GROUP
      RET=$?
else
      RET=0
fi
PCMDCMD="/opt/cray/nodehealth/default/bin/pcmd"
$PCMDCMD -r -q -n ${SHIFTER_NIDS} "/dsl/usr/bin/chroot /dsl sh /opt/cray/shifter/16.08.3/wlm/torque/cray-shifter-extra-service stop"
rm /scratch/system/shifter/jobs/passwd.${SHIFTER_JOBID} /scratch/system/shifter/jobs/group.${SHIFTER_JOBID}
if [[ -n $SHIFTER_ENV_VAR ]]; then
   ssh_proc_cleanup=/opt/cray/shifter/16.08.3/wlm/torque/cray-shifter-ssh-cleanup
   $PCMDCMD -r -n ${SHIFTER_NIDS} "/dsl/usr/bin/chroot /dsl sh $ssh_proc_cleanup $SHIFTER_USER $SHIFTER_JOBID"
fi
exit $RET
```

Listing 34. Epilogue scripts to clean up container environment on compute nodes.

```
#!/bin/bash
jobId="$1"
user="$2"
group="$3"
shift 3
nodeContext=""
udiRootPath=/opt/cray/shifter/16.08.3
mode="alps"
while getopts ":m:" opt; do
   case "${opt}" in
      m)
         mode="${OPTARG}"
         if [[ -n "$mode" && "$mode" == "local" ]]; then
            nodeContext="";
         fi
         ;;
      \?)
         echo "Invalid option: -${OPTARG}" >&2
         exit 1
         ;;
      :)
         echo "Option -${OPTARG} requires an argument" >&2
         exit 1
         ;;
   esac
done
die() {
   local msg
   msg="$1"
   echo "$msg" 1>&2
   exit 1
}
[[ -n "$user" ]] || die "user is undefined"
[[ -n "$group" ]] || die "group is undefined"
[[ -n "$jobId" ]] || die "Job ID is undefined"
datadir="/var/run/shifter/jobs/$user/$jobId"
[[ -d "$datadir" ]] || exit 0
job_nodelist="$datadir/nodelist"
[[ -e "$job_nodelist" ]] || exit 0
xtxqtcmd_log="$datadir/log_end"
xtxqtcmd="/opt/cray/nodehealth/default/bin/xtxqtcmd"
[[ -x "$xtxqtcmd" ]] || die "Could not find xtxqtcmd. Exiting"
unique_nodes="$datadir/unique_nodes"
[[ -e "$unique_nodes" ]] || exit 0
cmdStr="/dsl/usr/bin/chroot /dsl ${udiRootPath}/sbin/unsetupRoot"
ok=0
expected=0
if [[ "$mode" == "local" ]]; then
   echo $cmdStr >> $xtxqtcmd_log
   /bin/sh -c "$cmdStr"
   [[ $? -eq 0 ]] && ok=1
   expected=1
else
   echo "$xtxqtcmd $unique_nodes $cmdStr" >> $xtxqtcmd_log
   "$xtxqtcmd" "$unique_nodes" "$cmdStr" >> $xtxqtcmd_log 2>&1
   ok=$( grep "Reply (complete) from .* exit code: 0" $xtxqtcmd_log | wc -l )
   expected=$( cat "$unique_nodes" | wc -l )
fi
ret=0
if [[ "$ok" -eq "$expected" ]]; then
   echo "udiRoot Cleanup successful"
else
   echo "udiRoot Cleanup FAILURE, $ok of $expected responses"
   ret=1
fi
if [[ $ret -eq 0 ]]; then
   rm -r "$datadir"
fi
exit $ret
```

Listing 35. `udiRoot-epilogue`

V. Scaling Performance

During initial testing on Blue Waters, a Shifter job could only successfully launch tasks on about 2000 nodes from a single `aprun` when using `shifter` CLI and about 700 nodes when using `setupRoot` to setup containers on compute nodes before starting the job script. In the `shifter` CLI test case, many failed tasks were seen throwing the error message: “`FAILED to lookup docker image <image:tag>`”. Occasionally, the tasks were also seen throwing a different error message: “`Failed to lookup username or attempted to run as root.`”. In `setupRoot` test case, many failed tasks were seen throwing the error messages: “`FAILED to get groups correctly`” and “`FAILED to lookup auxiliary gids. Exiting`”.

By analyzing the source code, we were able to identify the username and groups related error messages occurred due to `getgrouplist()` and `getgid()` calls not returning valid results. These two functions are invoked during the Shifter setup process to query for group IDs belonging to the execution user. Blue Waters uses LDAP as the directory service. When large number of concurrent query requests are sent from the the compute nodes to the LDAP server, the LDAP server reaches its maximum number of connections threshold and failed to respond to all query requests, thus leading to failure in Shifter setup.

To workaround this issue, we included a trigger into the `cray-shifter-extra-service` script (see listing 28) to start the Name Service Cache Daemon (`nscd`) service on each compute node. `nscd` service caches the user and group directory from LDAP. Using `nscd` service, when `shifter` CLI or `setupRoot` is invoked, local `nscd` service returns response to the `getgrouplist()` and `getgid()` queries, instead of sending the query to the busy LDAP server. We wrote the `cray-shifter-prologue` script to execute a "`id $user`" command on the mom node, then copy the local `/var/run/nscd/passwd` and `/var/run/nscd/group` files from the mom node into a shared directory accessible by all compute nodes (See line 25 to 27 of listing 28). These files are labeled with the corresponding job ID. When `cray-shifter-extra-service` is invoked later, the compute nodes copy the shared `nscd` passwd and group files (with reference to the job ID) into their respective local `/var/run/nscd` directory before starting the `nscd` service (see line 50 to 53 of listing 28). These steps ensure the right user and groups information required by the job are cached by `nscd` on all allocated compute nodes. Using this setup, job prologue script was able to initialize `setupRoot` on 1024 compute nodes without failure. As the time required to setup Shifter containers through prologue increases with the number of compute nodes, a prologue timeout of 300 seconds set in the resource manager limited the scaling of Shifter job up to 2048 compute nodes without exceeding the prologue timeout. On the other hand, when using `shifter` CLI to launch applications, this `nscd` setup provides a consistent scaling to 4000 nodes consistently. Without this `nscd` setup, some nodes occasionally failed to execute the `shifter` process successfully due to LDAP not returning valid response. At the end of a Shifter job, we wrote epilogue scripts to remove the local cache files in `/var/run/nscd` directory of all allocated compute nodes and the shared `nscd` files.

For the other issue where Shifter fails to look up docker image, we found the scalability was limited by a single instance of `imagegwapi.py` (listening to port 5000) launched from the `shifter-imagegw` script. To improve the scalability, we duplicated the Python script into `imagegwapi1.py` and `imagegwapi2.py`, modified them to listen to different ports (5001 and 5002 respectively) and added them into the `shifter-imagegw` script (see listing 22). With three instances of `imagegwapi.py` running, we successfully launched `shifter` CLI tasks on 4096 compute nodes from a single `aprun` call.

```
root@shifter:/opt/cray/shifter/16.08.3/lib64/shifter> grep "LISTEN_PORT =" imagegwapi*.py
imagegwapi.py:LISTEN_PORT = 5000
imagegwapi1.py:LISTEN_PORT = 5001
imagegwapi2.py:LISTEN_PORT = 5002
```

Listing 36. `imagegwapi*.py` scripts listening to different ports.

In addition, when `shifter` CLI loads the required loop device kernel module (`loop.ko`), the module parameter `max_loop` is set to 0 by default. Online documentation of Shifter recommends to set `max_loop=128`. This avoids race condition with loading the kernel module when multiple instances of Shifter processes are launched concurrently on the same compute node (e.g. executing "`aprun -n 32 -N 32 -b shifter`"). We verified that the squashfs kernel module (`squashfs.ko`) is required to be preloaded together with the loop device kernel module in order for this type of job launch to be successful. We included these additional prologue routines into the `cray-shifter-extra-service` script (see line 55 and 56 of listing 28.

VI. GPU Support on Shifter

The Shifter source distribution branch which we pulled from NERSC's *github* repository came with GPU support. To use this feature, we configured `siteResources=/opt/shifter/site-resources` parameter in `udiRoot.conf` file. A script `activate_gpu_support.sh` is provided in the Shifter distribution to setup the GPU driver in the container. We modified `PATH` in the script to point to the location on the compute host where *nVidia* driver is installed (see listing 37).

```
default/:/ # grep "export PATH" /opt/cray/shifter/16.08.3/bin/activate_gpu_support.sh
export PATH=/opt/cray/nvidia/default/bin:/usr/local/bin:/usr/bin:/bin:/sbin
```

Listing 37. Setting path to *nVidia* driver in `activate_gpu_support.sh`

This `activate_gpu_support.sh` script currently only works with `shifter` CLI. The `CUDA_VISIBLE_DEVICES=0` environment variable is required in the job script to execute GPU application through `shifter` CLI (see listing 38). This setting configures Shifter runtime to bind the GPU driver from the compute host (found from `PATH` set in `activate_gpu_support.sh` script) to the `/opt/shifter/site-resources` directory in the container.

```
export CUDA_VISIBLE_DEVICES=0
aprun -b -- shifter --image=<image:tag> nvidia-smi
```

Listing 38. Using `shifter` CLI to execute GPU application.

VII. Other Operational Issues

A. Encoding and Decoding Issue

During initial tests, Shifter encountered an encoding issue when pulling and converting Docker image that contains files with special characters. Shifter documentation provided a workaround to this issue by setting default encoding to use `utf8` in `sitecustomize.py` file (see listing 39).

```
default/:/opt/cray/shifter/16.08.3/lib64/python2.6/site-packages # cat sitecustomize.py
import sys
reload(sys)
sys.setdefaultencoding('utf8')
```

Listing 39. Changing default encoding to `utf8`.

However, this change introduced another decoding issue. To resolve this, the `dockerv2.py` file was edited to use `utf8` decoding.

```
default/:/opt/cray/shifter/16.08.3/lib64/python26/site-packages/shifter_imagegw # diff dockerv2.py.org dockerv2.py -Nu
--- dockerv2.py.org 2017-08-29 09:08:08.000000000 -0500
+++ dockerv2.py 2017-08-29 11:33:28.000000000 -0500
@@ -625,6 +625,9 @@
          tfp = tar_file_refs[layer_idx]
          members = layer_paths[layer_idx]
+
+         # Change encoding to 'utf8' to take care of unicode character in file paths.
+         base_path = base_path.encode('utf8')
          tfp.extractall(path=base_path, members=members)
          # We need to make sure everything is writeable by the user so
```

Listing 40. Changing decoding to `utf8`.

B. Untracked process in SSH session

When using prologue to setup Docker image on compute nodes, the `setupRoot` process starts an `sshd` daemon in the container allowing the user to remote login via `ssh` from the mom node into the compute node's container environment through port 1204.

```
user@mom:~> cat .shifter/config
Host *
Port 1204
IdentityFile ~/.shifter/id_rsa
StrictHostKeyChecking no
UserKnownHostsFile /dev/null
LogLevel error

user@mom:~> ssh -F .shifter/config nidxxxxx
-bash-4.2$ hostname
```

Listing 41. Direct remote login via `ssh` from mom node to compute node's container environment.

We noticed that any process started on the compute node through this direct login is not tracked by ALPS, thus any background or daemon process would be left running on the compute node even after the job has ended. A `shifter-cray-ssh-cleanup` script was added to the epilogue to ensure a thorough cleanup of the stray processes (if any) on the compute nodes. This script is called from the `cray-shifter-epilogue` script (See listing 34).

```
default/:/opt/cray/shifter/16.08.3/wlm/torque # cat cray-shifter-ssh-cleanup
#!/bin/bash
USER=$1
JOBID=$2
SHIFTERLOG=/scratch/system/shifter/shifter16.log
echo Checking for stray process\(es\) launched by $USER through direct SSH to Shifter container.
ps -u $USER
if [ $? == 0 ]
then
   echo Found stray process\(es\) by $USER, killing these process\(es\)...
   pkill -u $USER
   echo Stray process cleanup completed.
   echo `date` Found stray process\(es\) on `hostname` allocated to job $JOBID left over by user $USER through direct SSH to Shifter container. Cleanup completed. >> $SHIFTERLOG
fi
```

Listing 42. `shifter-cray-ssh-cleanup`

C. User and Group Identity in Container

By design, Shifter copies files from the host's `/etc/shifter/shifter_etc_files` directory into the container's `/etc` directory so that certain site configuration can be preserved in the container environment. Some applications like *Apache Spark* [21] check for valid identity of the execution user. Thus updated `passwd` and `group` files are required to be parsed from the host's `/etc/shifter/shifter_etc_files` directory into the container. A *cron* script was configured on the System Management Workstation (SMW) to update these files (stored on the boot node) in a weekly basis.

```
SMW:~ # crontab -l|tail -2
# Weekly update of /etc/shifter/shifter_etc_files/<passwd/group> files.
@weekly ssh root@boot "sh /opt/localadm/shifter-update-passwd-group.sh"

boot:~ # cat /opt/localadm/shifter-update-passwd-group.sh
#!/bin/bash
if [ -f /rr/current/.shared/.session-lock ]
then
   ps -p `cat /rr/current/.shared/.session-lock` >/dev/null
   if [ $? == 0 ]
   then
      echo xtopview is currently locked, cannot perform update of Shifter passwd/group files.
      exit 1
   fi
fi
xtopview -r /rr/current -x /etc/opt/cray/sdb/node_classes -e "getent passwd > /etc/shifter/shifter_etc_files/passwd" > /dev/null 2>&1
xtopview -r /rr/current -x /etc/opt/cray/sdb/node_classes -e "getent group > /etc/shifter/shifter_etc_files/group" > /dev/null 2>&1
```

Listing 43. *cron* script to update `passwd` and `group` files in `/etc/shifter/shifter_etc_files` directory.

VIII. Conclusion

After five years into production, we continue to implement advanced software tools and capabilities on Blue Waters. We find that our efforts in getting advanced software stacks like Shifter V16 to work on Blue Waters is a valuable experience worth sharing with the community. Though we currently have a working model of Shifter V16, we have yet to explore the high availability features, such as using multiple MongoDB servers for database redundancy and using multiple service nodes to host the Shifter image manager gateway service for better load balancing (comparing to launching multiple `imagegwapi.py` instances on the same service node). We will continue to work on this area to improve the robustness of Shifter software infrastructure. With increased demand of portability and reproducibility in scientific computing, containerization technology is gaining traction in software development. The availability of Shifter V16 on Blue Waters will now provide an opportunity for researchers to develop, test, and use their container-based applications on Blue Waters, paving an early preparation path for portability to the next generation of HPC systems.

Acknowledgment

This work is part of the Blue Waters sustained-petascale computing project, which is supported by the US National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications.

We thank Mr. Mark Dalton of Cray Inc. for his consultation in completing this work and the Shifter open source community for the sharing of valuable online resources.

References

[1] D. M. Jacobsen and R. S. Canon, "Contain This, Unleashing Docker for HPC," in *CUG2015 Proceedings*, 2015.

[2] R. S. Canon and D. Jacobsen, "Shifter: Containers for HPC," in *CUG2016 Proceedings*, 2016.

[3] K. Kincade. (2015) NERSCs Shifter Makes Container-based HPC a Breeze. [Online]. Available: https://www.hpcwire.com/2015/08/07/nerscs-shifter-makes-container-based-hpc-a-breeze/

[4] *Shifter Configuration Guide 1.0*, Cray Inc., 901 Fifth Avenue, Suite 1000, Seattle, 2015. [Online]. Available: http://docs.cray.com/pdf/shifter_configuration_guide.pdf

[5] *XC Series Shifter Configuration Guide (CLE 6.0.UP06) S-2572*, Cray Inc., 901 Fifth Avenue, Suite 1000, Seattle, 2016. [Online]. Available: http://docs.cray.com/PDF/XC_Series_Shifter_Configuration_Guide_CLE60UP06_S-2572.pdf

[6] SHIFTER. [Online]. Available: https://github.com/NERSC

[7] J. Muggli, B. Bode, T. Hoefler, W. Kramer, and C. L. Mendes, "Blue Waters Testing Environment," in *CUG2012 Proceedings*, 2012.

[8] openSUSE download server. [Online]. Available: http://download.opensuse.org

[9] MongoDB for GIANT Ideas. [Online]. Available: https://www.mongodb.com

[10] Redis. [Online]. Available: https://www.redis.io

[11] Python Programming Language. [Online]. Available: https://www.python.org

[12] Celery: Distributed Task Queue. [Online]. Available: http://www.celeryproject.org

[13] Flask (A Python Microframework). [Online]. Available: http://flask.pocoo.org

[14] PyMongo - MongoDB API. [Online]. Available: https://api.mongodb.com/python/current/

[15] Python client for Redis key-value store. [Online]. Available: https://pypi.python.org/pypi/redis

[16] Gunicorn - Python WSGI HTTP Server for UNIX. [Online]. Available: http://www.gunicorn.org

[17] The PyPA recommended tool for installing Python packages. [Online]. Available: https://pypi.python.org/pypi/pip

[18] PyPI - the Python Package Index. [Online]. Available: https://pypi.python.org/pypi

[19] Virtualenv. [Online]. Available: https://virtualenv.pypa.io

[20] MUNGE Uid 'N' Gid Emporium. [Online]. Available: https://dun.github.io/munge/

[21] Apache Spark - Unified Analytics Engine for Big Data. [Online]. Available: https://spark.apache.org