How to Build ClusterStor SDA Client Packages

Nikitas Angelinas

Software Engineer IV
Cray Inc.

Date: 2018-05-22

Version: 1.0

Introduction

This document provides instructions on building Lustre client packages for the ClusterStor Secure Data Appliance (SDA), using the SDA Lustre source code repository. Note the following regarding Lustre client package builds with SDA Lustre source:

- Not all combinations of Linux kernel version, Linux distribution, third-party OFED stack release (optional) and, perhaps, additional components of the operating environment are supported by the SDA Lustre source repository. For example, SDA Lustre client builds using the sda4 branch are supported for RHEL 6.5-derived distributions, but not for RHEL 6.9-derived or RHEL 7.x-derived distributions. Some branch names have been selected to convey information regarding the OS distribution version that they support; e.g. branch sda3_rhel7.1_client supports the build of SDA Lustre clients for RHEL 7.1-derived distributions. Branches with this style of purposeful name should support earlier OS distribution versions and may support some (but not all) later OS distribution versions. Support across SLES-derived distributions and third-party OFED stack versions may also be conveyed via the use of purposeful branch names, with version support varying in similar ways.

- Using the SDA Lustre repository as a base, and with some additional effort, it may be possible to successfully build SDA Lustre clients for unsupported configurations.

- This document does not include instructions to build SDA Lustre server packages, although they should be supported by some branches of the SDA Lustre source repository.

Building Client Packages

Use the following procedures to build Lustre client packages for client hosts that need to mount a ClusterStor SDA filesystem.

Obtain the SDA Lustre Source Code

Obtain a copy of the SDA Lustre source code git repository, from Cray's lustre-sda page on Github (https://github.com/Cray/lustre-sda), by issuing the following command:

$ git clone https://github.com/Cray/lustre-sda

Obtain and Configure Supporting Packages

Confirm that the following prerequisites, regarding the host machine that will be used to carry out the build (the build host), have been met:

- Ensure that all necessary build tools are available on the build host. If using a RHEL-derived Linux distribution, these tools can be installed by issuing the following command:

  $ yum install gcc automake autoconf libtool rpm-build kmod libselinux-devel libyaml-devel zlib-devel keyutils keyutils-libs keyutils-libs-devel

  This package list may not be exhaustive; ensure that the full list of build dependencies for a given environment is installed on the build host.

- Lustre client modules make use of Linux kernel APIs. Ensure that kernel headers for the target Linux kernel version on which the SDA Lustre client hosts will run are installed on the build host. If a RHEL-derived Linux distribution is used, and assuming that the target client kernel is running on the build host, you can accomplish this by issuing the following command:

  $ yum install kernel-devel

  If the target client kernel is not running on the build host, a list of available kernel versions can be obtained by issuing the following command:

  $ yum search kernel-devel –showduplicates

  The required kernel headers can then be installed by issuing the following command:

  $ yum install kernel-devel-<version>

Prepare the SDA Lustre Source

1. Change directory into the location containing the SDA Lustre source code repository.

2. Determine the appropriate SDA source branch to use for a particular client build; there are several branches in the source repository. To obtain the full list of remote branches, issue the following command from within the source code directory:

   $ git branch -r

3. Log into an SDA server management node (nodes 00 and 01) and issue the following command:

   $ rpm -qa lustre*

   The command output should indicate the version of the Lustre server module packages installed on the SDA cluster. The package names should contain an s* version substring which, at the time of this writing, should be either s1.6, s2.0 or s2.1.

   Refer to the list of SDA server versions shown in Table 1 (which may not be exhaustive) to determine the corresponding SDA client branch that will be used for a given Lustre client build. Once retrieved from the source repository, the client branch can be used to build a matching set of Lustre client RPMs for a given SDA installation.

SDA Server Version	SDA Client Branch
s1.6	sda2_25, sda2_25_fips, sda2_25_fixed_label_write
s2.0	sda3, sda3_rhel7.1_client, sda3_rhel7.3_client sda3_fips_client, sda3_fixed_label_client, sda3_fixed_label_client_rhel7.3_client
s2.1	sda4, sda4_rhel6.9_port, sda4_fixed_label_client, sda4_fixed_label_client_bns

Table 1

A set of existing branches can yield a client that is compatible with a given server version. Branches within the set that are named sda[2_25|3|4] offer basic functionality, while others offer additional features and/or support for different kernel versions. Additional features, FIPS support and fixed-label client support are described later in this document.

Branch names are intended to be mostly self-explanatory. The sda4_fixed_label_client_bns branch may be an exception. It implements an alternative method of administering fixed-label client functionality that uses mount options to specify the fixed-label client SELinux labels instead of using kernel module parameters, as is done in other fixed-label client branches; this branch also offers support for SLES 12 SP2 kernel versions.

4. Switch to the corresponding local branch, by issuing the following command:

   $ git checkout <branch name>

   For example, if sda4 is identified as the matching SDA client branch in Step 2, issue the following command:

   $ git checkout sda4

5. Prepare the source tree for the client package build by issuing the following command:

   $ sh autogen.sh

Build Packages Using a Third-Party OFED Stack

For client hosts that use a third-party OFED stack (such as the ones provided by Mellanox) SDA Lustre client packages must be built against the same OFED stack. In such cases, verify that the development headers for the OFED stack that matches the target Linux kernel version and Linux distribution are installed on the build host. For Mellanox-provided OFED stacks, the development headers will likely be located under the 'RPMS' directory of the corresponding ISO file and are typically named mlnx-ofa_kernel-devel-*. The procedure to obtain and install the development headers for third-party OFED stacks is beyond the scope of this document.

Build Packages Using FIPS Support

Loading SDA Lustre client modules on kernels that have booted with FIPS support, i.e. by specifying **'fips=1'** on the kernel boot line, normally causes a kernel panic on RHEL 6.7 and later kernels. Corresponding kernel versions for non-RHEL-derived Linux distributions will (likely) vary, although this behavior has not been verified.

Branches with names containing the **'*_fips'** substring enable build products to be loaded in kernels of the aforementioned versions (if booted in FIPS mode), without causing a kernel panic. The git commits that fix this issue are b56feb093 for the sda3_fips_client branch and 2c17b751d for the sda2_25_fips_client branch. If FIPS support is required for a given build, select the corresponding branch in Step 2 of Prepare the SDA Lustre Source, above. Currently, there is no FIPS support for the sda4 branch. If this configuration is required, cherry-pick or otherwise apply one of the aforementioned git commits to a branch forked from the sda4 branch to yield a sda4-compliant and FIPS-compatible source tree. Using commit b56feb093 for this purpose might be preferential, as the sda4 branch should be more similar to the sda3 branch than the sda2_25 branch; the b56feb093 commit might be easier to apply than the 2c17b751d commit. In any case, both versions of the FIPS-related fix should be quite similar.

Using the FIPS-related fix will disable a number of Lustre checksumming algorithms in the resulting Lustre client modules, i.e. adler32, CRC32, and CRC32C. There may be other methods that allow the SDA Lustre client modules to be loaded in RHEL 6.7-derived or later version kernels that have been booted with FIPS support, while retaining support for the checksumming algorithms, but they may involve more complicated implementations. One method to allow such kernel versions to be booted (with FIPS mode enabled and without using the FIPS-related fix), might be to sign the Lustre client modules using the RHEL signature key or another signature key that may be accepted by the kernel module signature checking procedure.

Build a Fixed-Label Client

Building from branches named **'*_fixed_*'** yields what is known as a fixed-label or a single-level client. These clients allow the administrator to specify a fixed set of SELinux security labels that the client uses in its interactions with the Lustre filesystem. Such a client might be useful for tasks such as data migration and/or to run on hosts for which SELinux is not available or should be avoided for some reason, or for other types of work. Such clients are intended to run on SELinux-disabled (or not supported) hosts, although they may be able to run on SELinux-enabled hosts. In the latter case, they will likely ignore the security labels generated by SELinux and use the fixed security labels specified by the administrator in their interactions with the Lustre filesystem. Note that the use of a fixed-label client in SELinux-enabled hosts has not been validated through testing.

None of the client branches support both fixed-label client and FIPS-compatible functionality. If such a configuration is needed, try to create a new branch and combine the interesting patches into a single branch to yield a compatible source tree.

Build the Client Packages

After preparing the source tree for building, issue a **configure** command to run the autotools configure scripts and further tailor the source tree to the build environment; for RHEL-derived distributions, the base format of this command is as follows:

$ ./configure --disable-server --with-linux=/usr/src/kernels/<kernel version>

SLES-derived build target Linux distributions will need to modify the "**--with-linux**" option to **--with-linux=/usr/src/<kernel version>** and also require a **'--with-linux-obj=/usr/src/<kernel version-obj/<architecture>/<flavor>/'** parameter to be specified, which enables the build process to identify all necessary kernel-related files.

When building against a third-party OFED stack, the aforementioned command must include the **'--with-o2ib=<OFED development headers directory>'** parameter. In builds for which GSS/Kerberos functionality is not required, the **'--disable-gss'** parameter can be specified to exclude relevant code paths from the build process. This exclusion may be useful when building with branches that have not been tested for GSS/Kerberos functionality support, which can cause compile-time issues when relevant code paths are built. Additional options are available to further customize the build configuration. To display a list of documented options, issue the following command:

$ ./configure --help

However, as some options may not be documented, it may be necessary to examine the autotools scripts, and in particular the **'*.m4'** files, to view the complete set of options.

To launch the build process, issue the following command:

$ make -j <number of jobs>

If you need to generate RPM packages of the SDA Lustre client, issue the following command:

$ make rpms

The commands in this section, along with any required configuration options, can be specified in a single command line invocation. As an example, to build a fully-populated configuration for a RHEL-derived distribution against a Mellanox-supplied, third-party OFED stack, command series could be specified as:

$ ./configure --disable-server --disable-gss --with-linux=/usr/src/kernels/<kernel version> --with-o2ib=/usr/src/ofa_kernel/default --with-downstream-version=stable2.4 --with-downstream-release=fips_1.0 && make -j 16 && make rpms

After the build process is complete, kernel module files for the SDA Lustre client should reside in the respective source directories. RPM packages should be placed either in the rpmbuild directory for the current user or at the top-level directory of the source repository, likely depending on the Linux distribution of the build host and/or of the target host. Generated RPM packages can be installed with the tool or the preferred package manager for a specific distribution. For example, if you are using a RHEL-derived distribution, issue the following command, assuming that the relevant package directory only contains packages generated from the SDA Lustre client build:

$ yum localinstall $HOME/rpmbuild/RPMS/x86_64/*.rpm

If you prefer not to use the generated packages, it is possible to install client build products on the build host by using the following command from within the source repository:

$ make install

If you need to uninstall build products from the build host, issue the following command:

$ make uninstall

If you want to remove build products from the source directory, issue one of the following commands:

$ make clean

– or –

$ make distclean

Addressing Build Breakage

Most branches within the source tree should allow SDA Lustre clients to be built successfully against kernel versions of RHEL 6.x-derived distributions and some branches should allow for successful builds against kernel versions of early RHEL 7.x-derived distributions. Building against third-party OFED stacks should be supported to an extent, but results will vary depending on the newness of the OFED stack release. Building SDA clients from any branch might lead to compilation issues during the build process if more recent kernel and/or third-party OFED stack releases are used. If build breakage occurs, it is possible to yield a supporting source tree through a process of porting against the unsupported kernel and/or third-party OFED versions. The effort required to complete this porting can vary, depending on the branch used as the base for the porting effort and how current the kernel version and/or third-party OFED version are, compared to the supported versions.

Build breakage occurs because the Lustre source makes use of the Linux kernel and (optionally) OFED stack APIs, which tend to change as these products evolve. If the SDA Lustre source branches remain stale, building from them against later kernel and third-party OFED stack versions may fail. Based on our experience, most failures are related to the libcfs and LNET layers in Lustre, although failures can be traced to code in different parts of the Lustre source tree. Among other causes, libcfs should offer a layer of abstraction over kernel APIs, while LNET makes use of kernel networking and OFED APIs; both of these layers have changed over time as related projects evolve.

Unless a Lustre client build is attempted against a very recent kernel and/or third-party OFED stack version, it is likely that support for the selected kernel and/or third-party OFED stack (especially if supplied by a major vendor) will be present in the upstream Lustre source tree. Hence, compilation issues that arise during the build process can likely be addressed by cherry-picking and/or applying commits (or series of commits) from the upstream Lustre source tree to a branch within the SDA source tree. During the porting process, it may be useful to obtain a clone of the upstream Lustre source git repository (currently hosted at https://git.hpdd.intel.com/fs/lustre-release.git). It may also be useful to determine which SDA source branch could enable more straightforward porting for a particular client host configuration, a process that might involve some trial-and-error. Depending on the target server release version, one of the sda[2_25|3|4] repositories might be preferable to use, although other branches may support later kernels and/or third-party OFED stacks, and possibly also provide fixed-label client support or support kernels that have booted in FIPS mode.

As previously described, during the porting process, one recommended strategy is to issue a series of **'git blame'**, **'git show'**, and other related git commands, to determine which commit or series of commits in the upstream Lustre tree addresses a specific compilation issue seen during the build process. If LNET-related issues become too complicated to address with reasonable effort, it may be possible to substitute the entire 'lnet' directory at the top level of the Lustre SDA directory with a recent copy of the lnet directory from the upstream Lustre repository, as LNET may not be tightly coupled with other Lustre components. Although directory substitution may enable the installer to avoid addressing most/all LNET-related issues encountered during the build process, this method could, potentially, cause unexpected issues at runtime.

We have not yet verified whether the strategy of replacing an entire directory with a recent copy from the upstream repository could be used for the 'libcfs' directory (at the top level of the Lustre source tree) to avoid problems that may arise during the build process. However, as the libcfs subsystem may be more tightly coupled with other parts of the Lustre source (as compared to LNET), it is less likely that this would be an effective option to avoid compilation issues.