User Guide
JALoP over HTTP (v2.x)

Draft

08 March 2024

This Page is Intentionally Left Blank

Table of Contents

1 JALoPv2.x 5
2 Download/Clone Source Code 5
2.1 GitHub 5
2.2 JALoP CTC-Internal GitLab Repositories 5
3 Build and Install C Publisher (jald) and C/C++ Subscribers (jal_subscribe, jal_sub_cpp) 6
3.1 System Provisioning 6
3.1.1 RHEL 7 / CentOS 7 6
3.1.2 RHEL 8 / CentOS 8 7
3.2 Build and Install C Publisher (jald) and C/C++ Subscribers (jal_subscribe, jal_sub_cpp) 7
3.3 Configure C Publisher (jald) 8
3.4 Configure JAL-Local-Store (jal-local-store) 10
3.5 Configure C/C++ Subscriber (jal_subscribe, jal_sub_cpp) 12
4 Build and Install Java Subscriber (jnl_test-2.1.x.x.jar) 14
4.1 System Provisioning 14
4.2 Build and Install Java Subscriber 14
4.3 Configure Java Subscriber (jnl_test) 15
5 Run Publisher and Subscriber to Transfer JAL Records 15
5.1 Disable SELinux (Test Environment Only) 15
5.2 Stop Firewall (Test Environment Only) 16
5.3 Start JAL-LOCAL-STORE: 16
5.4 Insert Records into JAL-Local-Store 16
5.4.1 Log Records 16
5.4.2 Audit Records 16
5.4.3 Journal Records 16
5.5 Check for Inserted Records in JAL-Local-Store 17
5.6 Start Publisher 17
5.7 Start Java Subscriber (jjnl_test) 17
5.8 Start C/C++ Subscribers (jal_subscribe, jal_sub_cpp) 17
5.9 Purge Records from Local Store 18
5.10 Run the Publisher (jald) with GDB 18

5.11 Run the Publisher (jald) with Valgrind 18
6 JALoP Data-Taps 18
6.1 jalop-coreutils 19
6.1.1 Build & Install 19
6.1.2 Run tee and tail 19
6.2 jalop-jalauditd 19
6.2.1 Build & Install 19
6.2.2 Configure jalauditd 20
6.2.3 Run jalauditd 20
6.3 jalop-log4cxx 21
6.3.1 Build & Install 21
6.3.2 Create, Build & Run a Test Executable 22
6.4 jalop-rsyslog 23
6.4.1 Build & Install 23
6.4.2 Configure omjal plugin 24
6.4.3 Install omjal plugin 25
6.5 jalop-jaljournald 25
6.5.1 Build & Install 25
6.5.2 Configure jaljournald 26
6.5.3 Run jaljournald 27

This Page is Intentionally Left Blank

1 JALoPv2.x

JALoPv2.x is JALoP over HTTP. It has a C Publisher, Subscribers (available in C, C++, and Java), and several data-taps. The Publisher usually resides on a CDS system to securely and reliably transfer journal, audit, and log records generated by the CDS to one or more remote Subscribers.

2 Download/Clone Source Code

Create a top-level 'jalop' directory to store all JALoP repositories. This will be referred to as `<jalop_root>` throughout this document.

```
% mkdir jalop
% cd jalop/
```

2.1 GitHub

JALoPv2.x Publisher and Subscriber repositories are available on GitHub at this URL – https://github.com/JALoP

Clone "JALoP" and "jjnl" repositories using git –

```
% git clone https://github.com/JALoP/JALoP.git
% cd JALoP/
% git checkout -t origin/2.x.x.x (if not already on 2.x.x.x branch)

% git clone https://github.com/JALoP/jjnl.git
% cd jjnl/
% git checkout -t origin/2.x.x.x (if not already on 2.x.x.x branch)
```

2.2 JALoP CTC-Internal GitLab Repositories

All the JALoP source code repositories are available on the CTC-internal gitlab server on the ISIS network. You must VPN into the ISIS network to access those repositories.

If you have been granted access to the internal gitlab repositories as a developer, you can git clone the repositories as below –

```
% git clone git@gitlab.cdsc.ctc.com:jalop/jalop.git
% cd jalop/
% git checkout -t origin/2.x.x.x

% git clone git@gitlab.cdsc.ctc.com:jalop/jjnl.git
% cd jjnl/
% git checkout -t origin/2.x.x.x
```

3 Build and Install C Publisher (jald) and C/C++ Subscribers (jal_subscribe, jal_sub_cpp)

This is in "jalop" (or "JALoP" if cloned from GitHub) repository. Clone the repository as mentioned above, if not done yet. The C Publisher (jald) is the process that negotiates with the Subscriber(s) and sends JAL records to them. The C/C++ Subscriber(s) provide

3.1 System Provisioning

3.1.1 RHEL 7 / CentOS 7

The following instructions will allow a CentOS 7 minimal install to be provisioned to build and run the JALoP C implementation:

- Install EPEL
  - `$ sudo yum install epel-release`
- Install JALoP dependencies available from repos
  - `$ sudo yum install @development libxml2-devel libconfig-devel libuuid-devel openssl-devel libdb-devel xmlsec1-openssl-devel python2-scons python36-scons lcov libtool-ltdl-devel libcurl-devel doxygen libseccomp-devel systemd-devel libcap-devel libcap-ng-devel libmicrohttpd-devel texinfo gnutls gnutls-devel`
- Download and install test-dept unit test library.
  - `$ git clone https://github.com/norrby/test-dept.git`
  - `$ cd test-dept`
  - `$ ./boostrap`
  - `$ ./configure`
  - `$ sudo make install`
- Download and Install libmicrohttpd library
  - `Note that CentOS/RHEL 7 has an available installation of libmicrohttpd, but it too old for our purposes. It is recommended to uninstall the version installed by yum, if present`
  - `sudo yum remove libmicrohttpd`
  - `git clone https://git.gnunet.org/libmicrohttpd.git`
  - `git checkout v0.9.59`
    - `This is the version used by CentOS/RHEL8, for consistency`
  - `./bootstrap`
  - `./configure --enable-https --with-gnutls`
  - `make`
  - `sudo make install`
    - `See REAMDE for additional configuration options`
- Build and install axl
  - `$ git clone` https://github.com/ASPLes/libaxl.git
  - `$ cd libaxl`
  - `$ ./autogen.sh`
  - `$ sudo make install`
- Enable devtools repository
  - `(RHEL Only) sudo subscription-manager repo --enable rhel-7-server-devtools-rpms`
  - `(CentOS Only) sudo yum install centos-release-scl`
  - `sudo yum install devtoolset-7`
- Ensure /usr/local/lib is in the dynamic linker path

    - $ echo /usr/local/lib | sudo tee /etc/ld.so.conf.d/usr-local-lib.conf
    - $ sudo ldconfig

3.1.2 RHEL 8 / CentOS 8

The following instructions will allow a CentOS 8 Stream minimal install to be provisioned to build and run the JALoP C implementation:

- RHEL 8: Enable CodeReady Builder
    - `$ sudo subscription-manager repos --enable codeready-builder-for-rhel-8-x86_64-rpms`
- CentOS 8: Enable Powertools (unbranded version of RHEL's CodeReady Builder)
    - `$ sudo dnf config-manager --set-enabled powertools`
- Install JALoP dependencies available from repos
    - `$ sudo dnf install @development libxml2-devel libconfig-devel libuuid-devel openssl-devel libdb-devel xmlsec1-openssl-devel libtool-ltdl-devel apr-util-devel libcurl-devel doxygen lcov libseccomp-devel systemd-devel libcap-devel libcap-ng-devel libmicrohttpd-devel`
- `Build and install test-dept`
    - `$ git clone https://github.com/norrby/test-dept.git`
    - `$ cd test-dept`
    - `$ ./bootstrap`
    - `$ ./configure`
    - `$ sudo make install`
- `Build and install axl`
    - `$ git clone https://github.com/ASPLes/libaxl.git`
    - `$ cd libaxl`
    - `$ ./autogen --disable-py-axl`
    - `$ sudo make install`
- `Install python36-scons`
    - `$ sudo yum install python36-scons`
    - `$ sudo pip3 install scons`
- Use python36 as python
    - `$ sudo alternatives --config python`
        - Enter the number for the /usr/bin/python36 selection
- Ensure /usr/local/lib is in the dynamic linker path
    - $ echo /usr/local/lib | sudo tee /etc/ld.so.conf.d/usr-local-lib.conf
    - $ sudo ldconfig

3.2 Build and Install C Publisher (jald) and C/C++ Subscribers (jal_subscribe, jal_sub_cpp)

The "jalop" (or "JALoP" if cloned from GitHub) repository has and builds the following applications –

- "jald" (the Publisher)
- "jal-local-store" (the JAL-Local-Store)
- "jaldb_tail"
- "jal_dump"
- "jal_purge"
- "jalp_test" (a test Producer)
- "jal_subscribe" (C Subscriber)

- “jal_sub_cpp” (C++ Subscriber)

It also builds the following shared libraries –

- “libjal-common.so”
- “libjal-db.so”
- “libjal-network.so”
- “libjal-producer.sp”
- “libjal-utils.so”

Follow the instructions below to build and install the above mentioned components -

- Change to the jalop top directory –
  - `% cd <jalop_root>/jalop/  (or, cd <jalop_root>/JALoP/, if cloned from github).`
- Checkout the v2.x branch of jalop -
  - `% git checkout -t origin/2.x.x.x`
- Clean up and build jalop –
  - `Note: PKG_CONFIG_PATH may need to be set to include /usr/local/lib/pkgconfig until we remove vortex dependency in JALoP v2.x.`
  - `One CentOS/RHEL7, the libmicrohttpd library built from source will also require the PKG_CONFIG_PATH to be set to wherever the .pc file is placed, which is also /usr/local/lib/pkgconfig by default`
  - `(CentOS7/RHEL7 Only) scl enable devtoolset-7 $SHELL`
    - `This enables a fully C++17 compliant version of gcc`
  - `% export PKG_CONFIG_PATH=”/usr/local/lib/pkgconfig”`
  - `% scons -c`
  - `% scons`
- Install the publisher and subscribers -
  - `% sudo PKG_CONFIG_PATH=”/usr/local/lib/pkgconfig” ./install_rhel_x86_64.sh`

3.3 Configure C Publisher (jald)

Make a copy of `<jalop_root>/jalop/test-input/jald.cfg` and adjust according to your Subscriber’s IP, port, etc. This is to avoid changing the sample configuration file in the git source tree. Here is an example of `jald.cfg` file –

```
# the path to the private key, used for TLS negotiation
#private_key = "./test-
input/TLS_CA_Signed/client/jal_publisher_v2_client.key.pem";
private_key = "./test-input/TLS_Self_Signed/pub_key.pem";

# the path to the public cert, used for TLS negotiation
#public_cert = "./test-
input/TLS_CA_Signed/client/jal_publisher_v2_client.cert.pem";
public_cert = "./test-input/TLS_Self_Signed/pub_cert.pem";

# UUID used to identify this publisher
publisher_id = "cc0191c2-97e8-4cbf-af13-920d268d68ec";
```

```
# time in seconds between checks for new records when none are
available
poll_time = 1L;

# time in seconds between attempts to reconnect to peers where the
connection has closed
# the special value of -1 can be used to indicated that jald should
not attempt reconnects
retry_interval = 30L;

# Network timeout for each session, in minutes. Upon failure to send
or receive
# data in this time, a network outage is assumed and the session
closes.
# The special value of 0 implies not network timeout is enforced.
network_timeout = 60L;

# path to the root of the database (optional)
db_root = "./testdb";

# path to a directory containing the JALoP schemas (optional)
schemas_root = "./schemas/";

# file storing PID of jald when daemonized.
#pid_file = "/var/log/jalop/jald-pid.txt";

# Log directory of jald when daemonized.
log_dir = "./";

# A list of supported digest algorithms. These algorithms should be
ordered by preference
# in a single double-quoted string with a space separating the
algorithms.
# Valid values are "sha256", "sha384", and "sha512"
digest_algorithms = "sha256";

# List of subscriber configurations.
peers = ( {
      # the hostname or IP address of the subscriber
      host = "192.168.137.130";
      # the port to connect to
      port = 8444L;
      # the mode of JALoP operation
      # To run JALoP in Archive mode, use "archive" or "archival"
      # To run JALoP in Live mode, use "live"
      mode = "archive";
      # array of digest challenge configuration settings ordered by
descending priority
      # Valid values are "on" and "off"
      digest_challenge = ["on"];
      # array of record types to be sent to the subscriber
      # Valid values are "audit", "log", and "journal"
      record_types = ["audit", "log", "journal"];
      # directory containing the CA certificate(s) to use for TLS
negotiation
      #cert_dir = "./test-
input/TLS_CA_Signed/client/trust_store_dir";
      cert_dir = "./test-input/TLS_Self_Signed";
      } );

# seccomp will restrict the process to the defined system calls.
# When the process is in the setup phase, at startup, it will be
restricted to the
# initial_seccomp_rules, both_seccomp_rules and final_seccomp_rules
system call sets. After the setup phase and before the process
# is doing its routine work, it will be further restricted to only
the both_seccomp_rules and final_seccomp_rules system call set.
#
enable_seccomp = false;
seccomp_debug = false;
initial_seccomp_rules =
["prctl","access","arch_prctl","execve","getcwd","getrlimit","ioctl",
"lstat","set_tid_address","seccomp","statfs"]
both_seccomp_rules =
["brk","close","fstat","lseek","mmap","mprotect","munmap","open","rea
d","rt_sigaction","rt_sigprocmask","set_robust_list","stat","write"]
final_seccomp_rules =
["flock","setsockopt","getpid","clone","connect","exit","exit_group",
"fcntl","fdatasync","ftruncate","futex","getdents","getpeername","get
sockname","getsockopt","gettid","madvise","nanosleep","openat","poll"
,"pread64","pwrite64","recvfrom","rename","rt_sigreturn","sched_yield
","sendto","socket"]
```

3.4 Configure JAL-Local-Store (jal-local-store)

The JAL-Local-Store (`jal-local-store`) is a process that receives and stores JAL Data sent from the JAL Producer applications. It has a Berkeley Database (BDB) to store and process the JAL records. The `jal-local-store` process must be started before the Publisher process (`jald`).

Make a copy of `<jalop_root>/jalop/test-input/local_store.cfg` and update accordingly. This is to avoid changing the sample configuration file in the git source tree. Here is an example local store configuration file –

```
private_key_file = "./test-input/rsa_key";
public_cert_file = "./test-input/cert";
system_uuid = "34c90268-57ba-4d4c-a602-bdb30251ec77";
hostname = "test.jalop.com";
db_root = "./testdb";
schemas_root = "./schemas/";

# The jal-local-store process, at startup, will check if it is
# running under systemd along with a systemd socket unit file
configuration.
# If so, there is no need to define any socket parameters below, they
will be ignored.
# Systemd will create the socket file for jal-local-store.
#
# If there is no systemd socket, jal-local-store will attempt to
create it.
# Enter the file system path where the socket file will be created
# socket_owner and socket_group will default to the user and group
# the jal-local-store process is running as and socket mode will
default to 0666.
socket = "./jal.sock";
#
# Uncomment to define a socket_owner other than the default (the user
the process is running as)
# The username used must exist on the system.
#socket_owner = "jalls";
#
# Uncomment to define a socket_group other than the default (the
group the process user belongs to).
# The groupname used must exist on the system.
#socket_group = "jalproducer";
#
# Uncomment to define socket_mode other than the default (0666).
# This must be a string representing exactly 4 digits.
# Each digit must be in the range of 0-7.
#socket_mode = "0420";
#
# example socket file listing after being created
# sr--w----.  1 jalls jalproducer  0 Jan 23 13:15
/var/run/jalop/jal.sock

# Process will cd to / (root directory),fork, and will run as a
daemon.
# When running the process as daemon, and even though the jal-local-
store
# will resolve relative paths for you, it is always safer to use
# absolute paths for configurations in this file that require file
system paths.
daemon = true;

sign_sys_meta = false;
manifest_sys_meta = false;

# Flow control functionality turned off if accept_delay_thread_count
set to zero.
# Below are the default values if not set.
#accept_delay_thread_count = 10;
#accept_delay_increment = 100;
#accept_delay_max = 10000000;
```

```
# File storing PID of jal-local-store when daemonized.
#pid_file = "/var/log/jalop/jls-pid.txt";

# Log directory of jal-local-store when daemonized.
log_dir = ".";

# Digest algorithm to use to generate digests in system metadata
sys_meta_dgst_alg = “sha256”;

# seccomp will restrict the jal-local-store process to the defined
system calls.
# When the process is in the setup phase, at startup, it will be
restricted to the
# initial_seccomp_rules and final_seccomp_rules system call sets.
After the setup phase and before the process
# is doing its routine work, it will be further restricted to only
the final_seccomp_rules system call set.
#
enable_seccomp = true;
# this rule will restrict the process from setting flags on a file
restrict_seccomp_F_SETFL = true;
initial_seccomp_rules =
["geteuid","getgid","capget","capset","chmod","chown","arch_prctl","b
ind","brk","chdir","dup2","execve","flock","getcwd","getdents","getde
nts64","getrlimit","ioctl","listen","lstat","prctl","prlimit64","rena
me","rt_sigaction","rt_sigprocmask","seccomp","select","set_tid_addre
ss","setsid","statfs","sysinfo"];
final_seccomp_rules =
["sched_yield","accept","access","brk","clone","close","connect","exi
t","exit_group","fcntl","fdatasync","fstat","futex","getpid","getppid
","getrandom","getsockopt","gettid","getuid","lseek","madvise","mkdir
","mmap","mprotect","munmap","open","openat","poll","pread64","pwrite
64","read","recvmsg","rt_sigreturn","set_robust_list","socket","stat"
,"unlink","write"];
```

3.5 Configure C/C++ Subscriber (jal_subscribe, jal_sub_cpp)

Make a copy of `<jalop_root>/jalop/test-input/jal_subscribe.cfg` and update accordingly. This is to avoid changing the sample configuration file in the git source tree. Here is an example C/C++ subscriber configuration file –

An example of jal_subscribe.cfg given below –

```
# If true, utilize tls encryption and authentication
enable_tls = true

# The path to the private key, used for TLS.
private_key = "<repo_root>/test-
input/TLS_CA_Signed/server/jal_subscriber_v2_server.key.pem";
```

```
# The path to the public cert, used for TLS.
public_cert = "<repo_root>/test-
input/TLS_CA_Signed/server/jal_subscriber_v2_server.cert.pem";

# File containing all certificates for remote peers concatenated
together.
trust_store = "<repo_root>/test-
input/TLS_CA_Signed/server/trust_store_dir/jal_subscriber_v2_server.t
rusted_certs";

# JalSubscribe v2, Subject to Change
# The path to the root of the database.
db_root = "./testdb";

# The port to listen on for incoming connections
port = 1234;

# The IPv4 hostname or address to bind the http server to for
incoming connections.
address = "127.0.0.1";

# The supported record types.
record_type = [ "audit", "log", "journal" ];

# Allowed mode (archive/archival or live)
mode = "archive";

# Max number of sessions before the least-recently-active session is
removed
session_limit = 100;

# The timeout to set before an http transaction times out  0 means no
timeout
# This timeout only affects a single connection - e.g. time between
POST packets when
# transferring a large journal payload. If reached, the session
associated with that
# exchange will be closed
network_timeout = 0;

# The maximum size in bytes of the RAM buffer used to process
incoming records.
# If a record payload exceeds this size, it will be dumped to the
filesystem during receipt
# and processed in bufferSize chunks.
# A high value here puts more demand on system memory, but can
improve performance considerably by
# eliminating use of disk.
# A low value will result in more record payloads being written to
disk.
```

```
# Each session (probably 3 per producer in most cases) may
simultaneously consume up to bufferSize memory
# for processing record data.
# NOTE: This applies only to generic message handling. A specific db
interface
# (BerkeleyDB for audit and log types for instance) may not respect
this setting
buffer_size = 4096;

# The maximum number of active sessions allowed at once
session_limit = 100;

# A list of supported digest algorithms. These algorithms should be
in a single double-quoted string with a space separating the
algorithms.
# Valid values are "sha256", "sha384", and "sha512"
digest_algorithms = "sha256";

# The database format with which to store received records
# Valid values are "bdb" and "fs"
database_type = "bdb";

# For http server implementations which support thread pools (such as
libmicrohttpd)
# Set the size of the thread pool
# 9 is sufficient for 3 fully connected publishers to be handled
concurrently, and is a safe starting point
http_server_thread_pool_size = 9;
```

4 Build and Install Java Subscriber (jnl_test-2.1.x.x.jar)

This is the “jjnl” repository. Clone the repository as below (if not done yet) –

```
% cd <jalop_root>/

% git clone https://github.com/JALoP/jjnl.git
Or,
% git clone git@gitlab.cdsc.ctc.com:jalop/jjnl.git


% cd jjnl
% git checkout -t origin/2.x.x.x (if not already in that branch)
```

4.1 System Provisioning

Install the following packages –

```
% sudo yum install java-11-openjdk-devel ant maven
```

4.2 Build and Install Java Subscriber

- Build the “master” branch for JALoPv2.x -

- `% cd <jalop_root>/jjnl/jnl_parent/`
    - `% mvn clean`
    - `% mvn -Djava.version=1.8 -U clean package (for Java 8 build), or`
    - `% mvn -Djava.version=11 -U clean package (for Java 11 build)`

- Install (optional) -
    - `% cd <jalop_root>/jjnl/jnl_lib/`
    - `% mvn install`

4.3 Configure Java Subscriber (jnl_test)

```
% cd <jalop_root>/jjnl/jnl_test/
```

Copy `./jnl_test/target/test-classes/sampleHttpSubscriber.json` here and update accordingly. This is to avoid changing the sample config file in git source tree.

An example of `sampleHttpSubscriber.json` given below –

```
{
  "address": "127.0.0.1",
  "port": 8444,
  "subscriber": {
    "maxSessionLimit": 100,
    "recordType": [ "audit", "log", "journal" ],
    "configureDigest": [ "on", "off"],
    "digestAlgorithms": [ "SHA256", "SHA384", "SHA512" ],
    "configureTls": "off",
    "output": "./output",
    "mode": "archive",
    "createConfirmedFile" : "on",
  }
  "ssl": {
    "Key Store Passphrase": "changeit",
    "Key Store": "./certs/trust_store/server.jks",

    "Trust Store Passphrase": "changeit",
    "Trust Store": "./certs/trust_store/remotes.jks",
  }
}
```

Check section “Run the Publisher and Subscriber to Transfer JAL Records” below.

5 Run Publisher and Subscriber to Transfer JAL Records

Follow the steps below to run the C Publisher and Java Subscriber to transfer JAL records.

5.1 Disable SELinux (Test Environment Only)

```
==========================================
Permanent: have "SELINUX=disabled" in "/etc/selinux/config" file,
restart VM.
Temporary: Enter the command "/usr/sbin/setenforce 0"
```

5.2 Stop Firewall (Test Environment Only)

```
==========================================
RHEL/CentOS 6.x: % sudo service iptables stop
        Disable: % sudo chkconfig iptables off


RHEL/CentOS 7.x: % sudo service firewalld stop
           Or,   % sudo systemctl stop firewalld
        Disable: % sudo systemctl disable firewalld
```

5.3 Start JAL-LOCAL-STORE:

```
========================================
# May need to clean up the first time. Make sure no jal-local-store is
running.
$ cd <jalop_root>/jalop/
$ pkill jal-local-store
$ sudo rm -rf ./jal.sock
$ sudo rm -rf /root/testdb
$ sudo mkdir /root/testdb

Make a copy of ./test-input/local_store.cfg here and update
accordingly. This is to avoid changing the sample configuration file
in the git cource tree.

$ sudo ./release/bin/jal-local-store --debug --no-daemon -c
./local_store.cfg &

For 2.0.0.2-beta and older:
$ sudo ./release/bin/jal-local-store --debug ./local_store.cfg &
```

5.4 Insert Records into JAL-Local-Store

5.4.1 Log Records

```
========================================
$ sudo ./release/bin/jalp_test -j ./jal.sock -a ~/jalop/jalop-test-
data/input/app_meta/log4cxx-warn.cfg -p ~/jalop/jalop-test-
data/input/journal/big_payload.txt -n 100 –t l
```

5.4.2 Audit Records

```
========================================
$ sudo ./release/bin/jalp_test -j ./jal.sock -a ~/jalop/jalop-test-
data/input/app_meta/log4cxx-warn.cfg -p ~/jalop/jalop-test-
data/input/journal/big_payload.txt -n 100 -t a
```

5.4.3 Journal Records

```
========================================
$ sudo ./release/bin/jalp_test -j ./jal.sock -a ~/jalop/jalop-test-
data/input/app_meta/log4cxx-warn.cfg -p ~/jalop/jalop-test-
data/input/journal/big_payload.txt -n 100 -t j
```

5.5 Check for Inserted Records in JAL-Local-Store

```
==============================================
$ sudo ./release/bin/jaldb_tail -n 1000000 -h /root/testdb/ -t l | wc
-l
$ sudo ./release/bin/jaldb_tail -n 1000000 -h /root/testdb/ -t a | wc
-l
$ sudo ./release/bin/jaldb_tail -n 1000000 -h /root/testdb/ -t j | wc
-l

Or,

$ sudo ./release/bin/jal_purge -h /root/testdb -b 2033-11-11T11:11:11
-x -t l | wc -l
$ sudo ./release/bin/jal_purge -h /root/testdb -b 2033-11-11T11:11:11
-x -t a | wc -l
$ sudo ./release/bin/jal_purge -h /root/testdb -b 2033-11-11T11:11:11
-x -t j | wc -l
```

5.6 Start Publisher

```
========================================
$ cd <jalop_root>/jalop/

Make a copy of ./test-input/jald.cfg here and update accordingly. This
is to avoid changing the sample configuration file in the git source
tree.

$ sudo ./release/bin/jald -d -c ./jald.cfg --no-daemon -s 2>&1 | tee
publisher.log
```

5.7 Start Java Subscriber (jjnl_test)

```
$ cd <jalop_root>/jjnl/jnl_test/

Make a copy of ./jnl_test/target/test-
classes/sampleHttpSubscriber.json here and update accordingly. This is
to avoid changing the sample configuration file in the git source
tree.

$ java -jar target/jnl_test-2.0.0.0.jar ./sampleHttpSubscriber.json
2>&1 | tee subscriber.log
```

5.8 Start C/C++ Subscribers (jal_subscribe, jal_sub_cpp)

```
=========================================
$ cd <jalop_root>/jalop/


Make a copy of ./test-input/jal_subscribe.cfg here and update
accordingly. This is to avoid changing the sample configuration file
in the git source tree.
```

```
To run the installed binary against installed libraries, use the
following:

$ [jal_subscribe|jal_sub_cpp] -c jal_subscribe.cfg 2>&1 | tee
subscriber.log

To instead run a locally built binary against locally built libraries
(not suitable for deployment to real hardware):

$ export LD_LIBRARY_PATH=<repo_root>/[debug|release]/lib/
$ ./[debug|release]/bin/[jal_subscribe|jal_sub_cpp] -c
jal_subscribe.cfg 2>&1 | tee subscriber.log
```

5.9 Purge Records from Local Store

```
========================================
$ sudo ./release/bin/jal_purge -h ./testdb/ -d -f -c -b 2033-11-
11T11:11:11 -t l
$ sudo ./release/bin/jal_purge -h ./testdb/ -d -f -c -b 2033-11-
11T11:11:11 -t a
$ sudo ./release/bin/jal_purge -h ./testdb/ -d -f -c -b 2033-11-
11T11:11:11 -t j

OR,

$ now=$(date -u "+%Y-%m-%dT%H:%M:%S.%N")
$ sudo ./release/bin/jal_purge -h ./testdb/ -d -f -c -b "$now" -t l
$ sudo ./release/bin/jal_purge -h ./testdb/ -d -f -c -b "$now" -t a
$ sudo ./release/bin/jal_purge -h ./testdb/ -d -f -c -b "$now" -t j
```

5.10 Run the Publisher (jald) with GDB

```
========================================
$ export LD_LIBRARY_PATH="/home/marefin/jalop/jalop/debug/lib/"
$ gdb -ex=r --args ./debug/bin/jald -d -c ./jald.cfg --no-daemon -s
```

5.11 Run the Publisher (jald) with Valgrind

```
=======================================
$ valgrind --tool=memcheck --leak-check=full --verbose --track-
origins=yes --log-file=valgrind_out.txt ./debug/bin/jald -d -c
./jald.cfg --no-daemon -s
```

6 JALoP Data-Taps

The JALoP data-taps capture and send JAL records to the JALoP Local Store (jal-local-store). Each data-tap depends on JALoP libraries, so JALoP v1 or v2 libraries must be build and installed prior to building a data-tap repository. Generally, each of the data-taps requires the JALoP `socket` and `db_root` addresses that `jal-local-store` (JLS) uses. Note that `jal-local-store` must be already running for any data-tap to send JAL records to it via the `socket`.

6.1 jalop-coreutils

Available on InteLink and JALoP development Git Server.

This has the JALoP version of GNU `tail` and GNU `tee` commands.

6.1.1 Build & Install

```
% cd <jalop_root>/jalop-coreutils/
% autoreconf -fiv  # for RHEL7 only
% ./bootstrap --skip-po  # for RHEL8 only
% ./configure --disable-gcc-warnings
% make -j
% make check
% [[ -e /tmp/install ]] || mkdir /tmp/install
% make install DESTDIR=/tmp/install
% cp /tmp/install/usr/local/bin/{tee,tail} /usr/local/bin
```

6.1.2 Run tee and tail

Note that the `jal-local-store` (JLS) process must be running already to receive and insert records.

```
# Based on <db_root> location, user may need 'sudo' in the
following instructions.

% echo "This is a test message to tee into JLS" | tee -j --
path=<path to JALoP socket> --schema-root=<path to JALoP schema
directory>
% echo -e "Test message1\nTest message2" > file_to_tail
$ tail -j --path=<path to JALoP socket> --schema-root=<path to
JALoP schema directory> ./file_to_tail
```

Use `jaldb_tail` to check for newly inserted records -

```
% jaldb_tail -t l -h <db_root>
```

Use `jal_dump` to verify that the log records came from `auditd` –

```
% jal_dump -u <UUID found using jaldb_tail> -h <db_root)-t l -d
z
```

6.2 jalop-jalauditd

Available on the NCDSMO InteLink and JALoP development Git Server. This is also available at GitHub.com/JALoP/JALoP-Auditd-Plugin.

6.2.1 Build & Install

```
% cd <jalop_root>/jalop-jalauditd (or JALoP-Auditd-Plugin)
% make clean
% make
% sudo make install
```

This shall install the followings –

- The binary `/sbin/jalauditd` (or `/usr/sbin/jalauditd`)

- The configuration files for RHEL7 and RHEL8 go into different directories. make install will install the configuration files in the following locations and will need to be copied to appropriate directory:
    - /etc/jalauditd/jalauditd.conf
    - /etc/audit/plugins.d/audisp-jalauditd.conf
- For RHEL7 the configuration files should be:
    - /etc/audisp/jalauditd.conf
    - /etc/audisp/plugins.d/audisp-jalauditd.conf
- For RHEL8 the configuration files should be:
    - /etc/jalauditd/jalauditd.conf
    - /etc/audit/plugins.d/audisp-jalauditd.conf
- The `jalauditd.conf` file is initially empty; you will need to edit this file, see below.

6.2.2 Configure jalauditd

The configuration file `jalauditd.conf` is initially empty. There can be 4 settings in this file as shown below.

```
socket = “/path/to/jalop/socket”;
schemas = “/path/to/schemas/root”;
keypath = “/path/to/key”;
certpath = “/path/to/cert”;
```

If the `socket` and `schemas` locations are not specified above, default locations specified by the JAL Producer Library (JPL) will be used. Below are the default socket and schemas locations –

```
socket = “/var/run/jalop/jalop.sock”
schemas = “/usr/share/jalop/schemas”
```

If `keypath` or `certpath` are not specified, no key or cert will be used for signing.

IMPORTANT: These settings must be consistent with the `jal-local-store` configuration file.

6.2.3 Run jalauditd

Note that the `jal-local-store` (JLS) process must run to receive and insert records sent by `jalauditd` via the `socket`.

Restarting `auditd` automatically runs the `jalauditd` child process.

```
% service auditd restart (or, systemctl restart auditd)
```

Use `jaldb_tail` to check for newly inserted records -

```
% sudo jaldb_tail –f –t l –h <db_root>
```

Use `jal_dump` to verify that the log records came from `auditd` –

```
      % sudo jal_dump -u <UUID found using jaldb_tail> -h <db_root> -t
l -d z
```

See `<jalop_root>/jalop-jalauditd/README` file for more details.

6.3 jalop-log4cxx

Available on InteLink and JALoP development Git Server.

6.3.1 Build & Install

Note that JALoP must be built and installed before attempting to build the jalop-log4cxx library.

```
      $ sudo yum install apr-util-devel
```

On RHEL 7 / CentOS 7:

```
            $ sudo yum install cmake3
            $ sudo yum install centos-release-scl
            $ sudo yum install devtoolset-8-gcc-c++ --
enablerepo='centos-sclo-rh'
            $ sudo scl enable devtoolset-8 'bash'
```

On RHEL 8 / CentOS 8:

```
            $ sudo yum install cmake
```

Build the library:

On RHEL 7 / CentOS 7:

```
            $ cd jalop-log4cxx
            $ cmake3 -D CMAKE_CXX_COMPILER=/opt/rh/devtoolset-
8/root/usr/bin/g++ -D CMAKE_INSTALL_PREFIX=/usr CMakeLists.txt
            $ make
```

On RHEL 8 / CentOS 8:

```
            $ cd jalop-log4cxx
            $ cmake -D CMAKE_INSTALL_PREFIX=/usr CMakeLists.txt
            $ make
```

Install the library:

```
      $ sudo make install
```

To uninstall the library:

```
      $ sudo make uninstall
```

6.3.2 Create, Build & Run a Test Executable

An example test executable is created during the CMake build at /path/to/jalop-log4cxx/src/examples/cpp/jalp-appender. To test using this executable, skip ahead to the instructions on starting the jal_local_store and running the executable.

To create a new test executable, do the following:

Create a separate folder for the test executable. This can be separate from the jalop-log4cxx folder:

```
$ mkdir ~/log4cxx-test
$ cd ~/log4cxx-test
```

Copy required files from jalop-log4cxx:

```
$ cp /path/to/jalop-log4cxx/src/main/cpp/jalpappender.cpp .
```

Create a main.cpp for the text executable. Use the following as an example:

```
#include <log4cxx/logger.h>
#include <log4cxx/propertyconfigurator.h>
#include <log4cxx/jalpappender.h>

using namespace log4cxx;

LoggerPtr logger(Logger::getLogger("Test"));

int main(int argc, char **argv)
{
  PropertyConfigurator::configure(argv[1]);
  LOG4CXX_INFO(logger, "qwerty")
  return 0;
}
```

Build the test executable:

```
     $ g++ main.cpp jalpappender.cpp -llog4cxx -ljal-producer -o
log4cxx-test
```

Create a configuration file for the test executable. Use the following as a starting point:

```
log4j.rootLogger=DEBUG,A1

log4j.appender.A1=org.apache.log4j.JalpAppender

log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%-4r [%t] %-5p %c %x -
%m%n
```

The following are optional parameters than can be added to the log4cxx config file:

After starting the ***jal-local-store***, run the test executable:

```
$ ./log4cxx-test path/to/test.cfg
```

This will send one record to jal-local-store before closing the connection. Verify the record with the following:

```
$ jaldb_tail -h /path/to/testdb -t l -d a
```

```
log4j.appender.A1.PATH – The path to jalop.sock
log4j.appender.A1.HOSTNAME – The hostname of the machine that the
JALoP local store is on
log4j.appender.A1.APPNAME – The name of the process that is logging
data
log4j.appender.A1.SCHEMA_ROOT – The path to the JALoP schemas
log4j.appender.A1.KEY_PATH – The path to the RSA key used for signing
log4j.appender.A1.CERT_PATH – The path to the Certificate used for
signing
```

This will print the app metadata for the records. The record inserted by the test executable does not create a payload from the string in LOG4CXX_INFO. It ends up in the message tag in the app metadata.

6.4 jalop-rsyslog

Available on InteLink and JALoP development Git Server.

6.4.1 Build & Install

Note that JALoP must be built and installed before attempting to build the rsyslog plugin.

```
$ sudo yum install git automake libtool libestr-devel
```

If you have access to a libfastjson-devel package, you can install that and skip to Build omjal plugin:

```
$ sudo yum install libfastjson-devel
```

6.4.1.1 Libfastjson headers

If you cannot install libfastjson-devel, the libfastjson headers must be made available to the rsyslog plugin in another way.

```
      $ git clone https://github.com/rsyslog/libfastjson.git
      $ cd libfastjson
      $ sudo mkdir /usr/include/libfastjson
      $ sudo cp ./*.h /usr/include/libfastjson
      $ sudo ln -s /usr/lib64/libfastjson.so.4
/usr/lib64/libfastjson.so
```

On RHEL 7 / CentOS 7:

```
$ export JSON_C_CFLAGS="-g"
$ export JSON_C_LIBS="/usr/lib64/libfastjson.so"
```

On RHEL 8 / CentOS 8:

```
$ export LIBFASTJSON_CFLAGS="-g"
$ export LIBFASTJSON_LIBS="/usr/lib64/libfastjson.so"
```

6.4.1.2 Build omjal plugin

Build the omjal plugin:

```
$ cd jalop-rsyslogd
$ export CPPFLAGS="-I/usr/include/libfastjson"
$ sh autogen.sh
$ make
```

6.4.2 Configure omjal plugin

Add the following configuration section to /etc/rsyslog.conf:

```
$ModLoad omjal   # calls JAL for rsyslog pass through to JALoP
$ActionOmjalSocket /var/run/jalop/jalop.sock
$ActionOmjalSchemas /share/jalop/schemas
$ActionOmjalKey <path_to_key>    # Optional
$ActionOmjalCert <path_to_key>   # Optional
*.* :omjal:
```

$ModLoad omjal

States that the omjal plugin shall be loaded

$ActionOmjalSocket /var/run/jalop/jalop.sock

Specifies the socket to be used to forward the messages to the JALoP local store. If this option is omitted from the file, it will use the default value specified in the JALoP library.

$ActionOmjalSchemas /share/jalop/schemas

Specifies the directory where the schema information is located. If this option is omitted from the file, it will use the default value specified in the JALoP library.

$ActionOmjalKey <path_to_key>

Specifies the path to the key to be used for signing. If this option is omitted from the file, then no key will be used.

$ActionOmjalCert <path_to_key>

Specifies the path to they certificate to be used for signing. If this option is omitted from the file, then no certificate will be used.

. :omjal:

<msg_filter_expression>:<output_module>:<parameters>

. specifies that all messages will be sent to this module. If, for example, you wanted to handle mail messages, mail.* would handle that.

See https://www.rsyslog.com/doc/v8-stable/configuration/filters.html for more information.

6.4.3 Install omjal plugin

Install the omjal plugin:

```
$ cd jalop-rsyslogd
$ make install
```

The rsyslog package installs libraries to /usr/lib64/rsyslog. If your environment uses a different location, either manually copy the library from .libs/omjal.so or change the libdir flag in autogen.sh

After installation, and after starting the ***jal-local-store***, restart the rsyslog service:

```
$ sudo systemctl restart rsyslog
```

To ensure that rsyslogd is connecting to jal-local-store, you can run jal-local-store with the debug (-d) flag. When jal-local-store is ready, the message `Thread_count: 1` will appear. After rsyslogd is connected, the message `Thread_count: 2` will appear.

To create messages in rsyslogd, the logger command can be used. For example:

```
logger "This is your message content."
```

Use `jaldb_tail` to check for newly inserted records -

```
% sudo jaldb_tail -f -t l -h <db_root>
```

Use `jal_dump` to verify that the log records came from `auditd -`

```
% sudo jal_dump -u <UUID found using jaldb_tail> -h <db_root> -t
l -d z
```

6.5 jalop-jaljournald

Available on InteLink and JALoP development Git Server. This is also available at GitHub/JALoP/JALoP-Journald.

6.5.1 Build & Install

Dependencies:

/usr/lib64/<jalop shared object files>

/usr/include/<jalop headers>

systemd-devel rpm

```
% cd <jalop_root>/jalop-jaljournald
% make clean
% make
% sudo make install
```

This shall install the followings –

- The binary `/bin/jaljournald`
- The `jaljournald` configuration file `/etc/jalop/jaljournald.cfg`. This file initially has some default values; you will need to edit this file, see below.

6.5.2 Configure jaljournald

The `jalauditd` configuration file `/etc/jalauditd/jalauditd.conf` iinitially has some default values; The settings in this file as shown below.

```
# verbose logging
debug = false;

# Start collecting logs from head, boot, or tail. Default: tail
# head: will get all journal logs available from multiple boots
#      and then continuously poll to get all logs as they arrive
# boot: will get all logs from current boot and then continuously
           poll to get all logs as they arrive.
# tail: will continuously poll to get all logs as they arrive.
init_point = "tail";

# collect all journal fields. Default: MESSAGE only
metadata = false;

# socket path that jal-local-store will be listening.
socket = "/path/to/jalop/socket";

# jalop schema path
schemas = "/path/to/schemas/root";

# hostname to record in log
hostname = "localhost";

# appname to record in log
appname = "jalop-journald";

# poll delay between checking for new journal logs
# in milliseconds 1000 = 1 second
delay = 1000;
```

If the `socket` and `schema_path` locations are not specified above, default locations specified by the JAL Producer Library (JPL) will be used. Below are the default socket and schema_path locations –

```
socket = "/var/run/jalop/jalop.sock"
schema_path = "/etc/jalop/schemas"
```

IMPORTANT: These settings must be consistent with the `jal-local-store` configuration file.

6.5.3 Run jaljournald

Note that the `jal-local-store` (JLS) process must run to receive and insert records sent by `jaljournald` via the `socket`. Also, make sure the jal-local-store socket path matches the jaljournald socket path and you have write permissions on that socket path.

Example command-line
```
      % jaljournald -c <path-to-config> --debug --metadata
```

Use `jaldb_tail` to check for newly inserted records -
```
      % sudo jaldb_tail -f -t l -h <db_root>
```

Use `jal_dump` to verify that the log records came from journal`d` –
```
      % sudo jal_dump -u <UUID found using jaldb_tail> -h <db_root)-t l
-d z
```