Linux

Use the following instructions to build endpoint SDKs in Java, C, and C++ for Linux.

> **Verified against:**
>
> **Host OS:** Ubuntu 14.04 LTS Desktop 64-bit.

Java endpoint SDK

To build the Java endpoint SDK, generate the Java endpoint SDK in Admin UI and download the generated .jar file.

C endpoint SDK

Before building the C endpoint SDK, install the following components on your machine:

1. Install compilers:
   a. For automatic installation, execute the following commands:

```
$ sudo apt-get install gcc
```

   b. For manual installation of version 4.8, refer to the following example:

```
$ sudo apt-get install python-software-properties
$ sudo add-apt-repository ppa:ubuntu-toolchain-r/test
$ sudo apt-get update
$ sudo apt-get install gcc-4.8
$ sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.8 50
```

2. Install CMake utility:
   a. For automatic installation, execute the following commands (tested on Ubuntu 14.04):

```
$ sudo apt-get install cmake
```

   b. For manual installation, refer to the following example:

```
$ wget http://www.cmake.org/files/v3.3/cmake-3.3.0-rc2.tar.gz
$ tar -zxf cmake-3.3.0-rc2.tar.gz
$ cd cmake-3.3.0-rc2/
$ ./configure
$ sudo make install
```

3. Install OpenSSL:

```
$ sudo apt-get install libssl-dev
```

4. Install CUnit:

```
$ sudo apt-get install libcunit1-dev
```

To build the C endpoint SDK, do the following:

1. Generate the C endpoint SDK in Admin UI.
2. Download and untar the Kaa C SDK archive.
3. Run the following commands.

```
mkdir build
cd build
```

4. Run the following Linux-specific commands.

```
cmake ..
make
make install
```

Build configuration

To configure the C endpoint SDK build, you can optionally specify the following parameters for the **cmake** command.

Parameter	Description	Values	Usage example
CMAKE_INSTALL_PREFIX	Directory for Kaa to be installed in.	Accepted: '/path/to/some/directory' Default: '/usr/local'	cmake -DCMAKE_INSTALL_PREFIX='/home/username/l
KAA_DEBUG_ENABLED	Build type	Accepted: • 0 - the release type • 1 - the debug type Default: 0	cmake -DKAA_DEBUG_ENABLED=1
KAA_MAX_LOG_LEVEL	Maximum log level used by the SDK	Accepted: • 0 - NONE (no logs) • 1 - FATAL • 2 - ERROR • 3 - WARN • 4 - INFO • 5 - DEBUG • 6 - TRACE Default: • 4 - If KAA_DEBUG_ENABLED=0 • 6 - if KAA_DEBUG_ENABLED=1	cmake -DKAA_MAX_LOG_LEVEL=4

KAA_WITHOUT_<MODULE>	Kaa module to be omitted during the build	Accepted: • EVENTS • LOGGING • CONFIGURATION • NOTIFICATION Default: All modules are present in the build	cmake -DKAA_WITHOUT_EVENTS=1
KAA_PLATFORM	SDK target platform **NOTE:** Before running the **cmake** command with the KAA_PLATFORM parameter for a platform other than supported, do the following: 1. Create a folder in $KAA_HOME/listfiles/platform/ and name it using [a-zA-z_-] symbols. You will be able to use the folder name as a value for the KAA_PLATFORM parameter. 2. Put the CMakeLists.txt file into the created folder. This file may contain specific compilation/linking flags, platform-dependent source files, third-party library dependencies, that is all information necessary for building the C endpoint SDK for this platform. 3. Optionally, specify the following parameters in the CMakeLists.txt file. • KAA_INCLUDE_PATHS - full path(s) to folder(s) containing additional header files • KAA_SOURCE_FILES - full path(s) to additional source files • KAA_THIRDPARTY_LIBRARIES - third-party libraries (the name of the library, for example, ssl, crypto)	Accepted: x86-64 ios esp8266 cc32xx Default: x86-64	cmake -DKAA_PLATFORM=x86-64 In the CMakeLists.txt file: set(KAA_INCLUDE_PATHS ${KAA_INCLUDE_PATHS} path_to_folder_1_with_header_files path_to_folder_2_with_header_files) set(KAA_SOURCE_FILES ${KAA_SOURCE_FILES} path_to_source_file_1 path_to_source_file_2) set(KAA_THIRDPARTY_LIBRARIES ${KAA_THIRDPARTY_LIBRARIES} some_library_1 some_library_2)

The following example illustrates the build procedure for the debug build with the INFO log level and disabled EVENTS feature.

```
mkdir build
cd build
cmake -DKAA_DEBUG_ENABLED=1 -DKAA_MAX_LOG_LEVEL=4 -DKAA_WITHOUT_EVENTS=1 ..
make
make install
```

C++ endpoint SDK

Before building the C++ endpoint SDK, install the following components on your machine:

1. Install compilers:
    a. For automatic installation, execute the following commands:

```
$ sudo apt-get install g++
```

b. For manual installation of version 4.8, refer to the following example:

```
$ sudo apt-get install python-software-properties
$ sudo add-apt-repository ppa:ubuntu-toolchain-r/test
$ sudo apt-get update
$ sudo apt-get install g++-4.8
$ sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-4.8 50
```

2. Install CMake utility:
   a. For automatic installation, execute the following commands (tested on Ubuntu 14.04):

```
$ sudo apt-get install cmake
```

   b. For manual installation, refer to the following example:

```
$ wget http://www.cmake.org/files/v3.3/cmake-3.3.0-rc2.tar.gz
$ tar -zxf cmake-3.3.0-rc2.tar.gz
$ cd cmake-3.3.0-rc2/
$ ./configure
$ sudo make install
```

3. Install the **Boost** libraries:
   a. For automatic installation, execute the following commands:

```
$ sudo apt-get install libboost1.55-all-dev
```

   b. For manual installation, refer to the following example:

```
$ sudo apt-get install libbz2-dev libbz2-1.0 zlib1g zlib1g-dev
$ wget
http://sourceforge.net/projects/boost/files/boost/1.58.0/boost_1_58_0.tar
.gz
$ tar -zxf boost_1_58_0.tar.gz
$ cd boost_1_58_0/
$ ./bootstrap.sh
$ sudo ./b2 install
```

4. Install the **AvroC++** library manually:

```
$ wget
http://archive.apache.org/dist/avro/avro-1.7.5/cpp/avro-cpp-1.7.5.tar.gz
$ tar -zxf avro-cpp-1.7.5.tar.gz
$ cd avro-cpp-1.7.5/
$ cmake -G "Unix Makefiles"
$ sudo make install
```

5. Install the **Botan** library by executing the following command:

```
$ wget https://github.com/randombit/botan/archive/1.11.28.tar.gz
$ tar -zxf 1.11.28.tar.gz
$ cd botan-1.11.28/
$ ./configure.py
$ sudo make install
$ sudo ln -s /usr/local/include/botan-1.11/botan /usr/local/include/botan
```

6. Install the **SQLite** library by executing the following command:
   a. For automatic installation, execute the following commands (tested on Ubuntu 14.04):

```
$ sudo apt-get install libsqlite3-0 libsqlite3-dev
```

   b. For manual installation, refer to the following example:

```
$ wget https://www.sqlite.org/2015/sqlite-autoconf-3081002.tar.gz
$ tar -zxf sqlite-autoconf-3081002.tar.gz
$ cd sqlite-autoconf-3081002/
$ ./configure
$ sudo make install
```

NOTE: Instead of manually installing all required components and libraries, you can follow the quick way to build C/C++ endpoint SDK. (only applicable for x86_64 platform build)

To build the C++ endpoint SDK, do the following:

1. Generate the C++ endpoint SDK in Admin UI.
2. Download and untar the Kaa C++ SDK archive.
3. Run the following commands.

```
mkdir build
cd build
cmake ..
make
make install
```

Build configuration

To configure the C++ endpoint SDK build, you can optionally specify the following parameters for the **cmake** command.

Parameter	Description	Values	Usage example
CMAKE_INSTALL_PREFIX	Directory for Kaa to be installed in	Accepted: '/path/to/some/directory' Default: '/usr/local'	cmake -DCMAKE_INSTALL_PREFIX= '/home/username/kaa'
KAA_DEBUG_ENABLED	Build type	Accepted: • 0 - the release type • 1 - the debug type Default: 0	cmake -DKAA_DEBUG_ENABLED=1

KAA_MAX_LOG_LEVEL	Maximum log level used by the SDK	Accepted: • 0 - NONE (no logs) • 1 - FATAL • 2 - ERROR • 3 - WARN • 4 - INFO • 5 - DEBUG • 6 - TRACE Default: • 4 - If KAA_DEBUG_ENABLED=0 • 6 - if KAA_DEBUG_ENABLED=1	cmake -DKAA_MAX_LOG_LEVEL=4
KAA_WITHOUT_<MODULE>	Kaa module to be omitted during the build	Accepted: • EVENTS • LOGGING • CONFIGURATION • NOTIFICATIONS • OPERATION_TCP_CHANNEL • OPERATION_LONG_POLL_CHANNEL • OPERATION_HTTP_CHANNEL • BOOTSTRAP_HTTP_CHANNEL • CONNECTIVITY_CHECKER Default: All modules are present in the build	cmake -DKAA_WITHOUT_EVENTS=1
KAA_WITH_SQLITE_LOG_STORAGE	Point the Kaa LOGGING module to use the SQLite persistent log storage. **NOTE:** Works only if KAA_WITHOUT_LOGGING=0.	Accepted: • 0 - disable SQLite log storage • 1 - enable SQLite log storage Default: 0	cmake -DKAA_WITH_SQLITE_LOG_STORAGE=1

The following example illustrates the build procedure for the debug build, with the INFO log level and disabled EVENTS feature and specified path to the folder Kaa will be installed in:

```
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX='/home/username/kaa' -DKAA_DEBUG_ENABLED=1
-DKAA_MAX_LOG_LEVEL=4 -DKAA_WITHOUT_EVENTS=1 ..
make
make install
```

Quick way to build C/C++ endpoint SDK

If you want to quickly build the endpoint SDK or build and run Kaa C/C++ demo applications, you can use a docker container with all necessary environment preinstalled.
NOTE: docker natively supports only amd64 architecture.

1. Follow docker installation guide depends on your OS.
2. Download the docker container.

```
docker pull kaaproject/demo_c_cpp_environment
```

3. Get inside container and compile what you need: SDK, demo applications, etc.

```
docker run -it kaaproject/demo_c_cpp_environment bash
```

NOTE:
To mount a host directory to the container's filesystem, add the following flag to the previous command: -v FOLDER_WITH_DEMO:FOLDER_INSIDE_CONTAINER
For example, the following command will build a demo project and direct you to the container's shell, where you can test immediately:

```
docker run -v FOLDER_WITH_DEMO:/opt/demo
 -it kaaproject/demo_c_cpp_environment bash -c 'cd /opt/demo/ &&
 chmod +x build.sh && ./build.sh clean build && bash'
```

4. After the compilation, launch the demo binary located at /opt/demo/build/ in the container's filesystem.
   NOTE:
   If you would like to run a compiled binary on some other host, you should have all third-party libraries like boost, etc. preinstalled.
