Development environment setup

This guide will help you set up the development environment necessary for installing Kaa and programming Kaa applications.

Third-party components

You can find the full list of third-party components used in Kaa at Third-party components.

Installing system components

Depending on what operating system you use, proceed as follows:

Installing components for C++ client

To use C++ clients in your Kaa instance, you need to install the components described in this section.

> **NOTE:** Currently C++ clients can be built only on Unix-type systems.

C++ system components

Ensure that the following C++ components are installed.

	Requirements	Recommended version
Compiler	C++11 standard support	g++ 4.7
Build system	CMake	cmake 2.8.8+

Third party components

The following third party libraries are required for Kaa installation.

Library	Link	Required version
Boost	http://www.boost.org	1.55.0
AvroC++	http://avro.apache.org	1.7.6
Botan	http://botan.randombit.net	1.10

To install these components, proceed as follows.

1. Install the **libbz2-dev** and **zlib1g-dev** libraries by executing the following command.

```
sudo apt-get install maven cmake libbz2-dev zlib1g-dev
```

2. Install the **Boost** libraries (Kaa client requires **log, system**, and **unit_test_framework** libraries) either automatically or manually.

   NOTE
   If Boost-1.55.0 version is not available in your OS package repository, you need to install it manually.

   a. For automatic installation, execute the following commands.

   ```
   sudo apt-get install libboost-log-dev
   sudo apt-get install libboost-system-dev
   sudo apt-get install libboost-unit_test_framework-dev
   ```

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

   ```
   wget -O boost.tar.gz <boost_download_link>
   tar -xvf boost.tar.gz
   # Go to unpacked directory
   ./bootstrap.sh --without-libraries=coroutine,chrono,graph,graph_parallel,math,python,wave
   ./b2
   sudo ./b2 install
   ```

3. Install the **AvroC++** library manually as illustrated in the following example.

   ```
   wget -O boost.tar.gz <boost_download_link>
   tar -xvf boost.tar.gz
   # Go to unpacked directory
   ./bootstrap.sh --without-libraries=coroutine,chrono,graph,graph_parallel,math,python,wave
   ./b2
   sudo ./b2 install
   ```

4. Install the **Botan** library by executing the following command.

   ```
   sudo apt-get install libbotan1.10-dev
   ```

Fetching source code

It is allowed to use any Git client to fetch the source code from the repository.

Set up your Git configuration (at least the username and email) and then proceed as follows:

1. Check your current settings.

   ```
   git config --get user.name
   git config --get user.email
   ```

   Skip the next step if these parameters are correct.
2. Reset parameters.

   ```
   git config --unset user.name
   git config --set user.name <Your Name>
   git config --unset user.email
   git config --set user.email <Your E-mail>
   ```

3. Fetch the source code.

   ```
   cd <local_path>
   git clone <path_to_the_repository>
   ```

   where <path_to_the_repository> is TODO

Kaa project structure looks as follows:

```
kaa/                -- Project root
  avrogenc/         -- Kaa Avro C Generator
  client/            -- Endpoints sources
    client-multi/
      client-c/
      client-cpp/
      client-java-android/
      client-java-core/
      client-java-desktop/
      kaatcp-c/
  common/            -- Shared components
    core/
    endpoint-shared/
    examples/         Demo applications
      robotrun/
      smarthousedemo/
  it/                 -- Integration tests
  sandbox/            -- Kaa sandbox
    builder/
    demo/
    web/
  server/             -- CRL, BSS, OPS
    admin/            -- Web UI
    appenders/        -- Log appenders implementation
    bootstrap/        -- BSS sources
    common/           -- Server shared components
      admin-rest-client/
      avro-ui/
      dao/
      dto/
      log-shared/
      netty-server/
      nosql/
      server-shared/
      thrift/
      thrift-cli-client/
      thrift-cli-server/
      utils/
      zk/
    control/            -- CRL sources
    flume/
    operations/         -- OPS sources
```

Building project

Assuming that you are in the root directory of the project, run the **mvn clean install** from the command line.

Alternatively, each part of the project can be built separately invoking maven build in the corresponding directory.

> **NOTE**
> Use **-Dmaven.test.skip=true** trigger to disable unit and integration tests.
> The C++ client build is held in a separate maven profile named compile-client-cpp (Maven command: **mvn clean install -P compile-client-cpp**).

Setting up Kaa infrastructure

1. Start MongoDB.

   ```
   sudo service mongodb start
   ```

2. Start ZooKeeper.

   ```
   sudo service zookeeper start
   ```

3. Install Kaa Control, Bootstrap and Operations servers.

```
sudo dpkg -i server/control/target/kaa-control*.deb
sudo dpkg -i server/bootstrap/target/kaa-bootstrap*.deb
sudo dpkg -i server/operations/target/kaa-operations*.deb
```

4. Clear logs.

```
sudo rm -rf /var/log/kaa/*
```

5. Start services.

```
sudo service kaa-control start
sudo service kaa-bootstrap start
sudo service kaa-operations start
```

6. Load test data.

```
cd /usr/lib/kaa-control/test
mongo kaa --eval "db.dropDatabase()"
sudo ./cleanup.sh
sudo ./init-configuration.sh
```

7. Generate Java SDK.

```
sudo ./gen-java-sdk.sh
```

8. Generate the client's SDK using PRS CLI.

```
cd /usr/lib/kaa-control/bin
./api
connect localhost:9090
generateSdk -a <application_id> -csv <conf_schema_version> -nsv <notification_schema_version> -psv
<profile_schema_version> -sdk <sdk_type> [-out <path_to_sdk>]

where:
  application_id        -- Application Id generated using ./init_configuration script.
                   Can be found in Mongo DB:
                   mongo kaa
                   db.application.find();
                   ..."_id": ObjectId(<here is needed value>)...
  conf_schema_version        -- Configuration schema version
  notification_schema_version    -- Notification schema version
  profile_schema_version    -- Profile schema version
  sdk_type            -- Sdk type ("java" or "cpp")
  path_to_sdk            -- Optional. Path where sdk should be stored.
```

Kaa property files

After Kaa installation on Ubuntu/Debian OS (deb packages), configuration files for each Kaa component will be extracted into the

/usr/lib/kaa-{component-name}/conf or /etc/kaa-{component-name}/conf directories.

Database properties

The default MongoDb properties file conf/mongo.properties looks as follows:

```
# Database name
db_name=kaa

# NoSQL database provider name
nosql_db_provider_name=mongodb

# Specific configurations for DAO layer
# Max wait time in seconds for history dao class. Custom property for Kaa History Service.
dao_max_wait_time=5

# specify hibernate sql dialect
hibernate_dialect=org.hibernate.dialect.PostgreSQL82Dialect

# specify if hibernate will format sql request
hibernate_format_sql=false

# specify if show hibernate sql request
hibernate_show_sql=false

# specify hibernate hbm2ddl strategy
hibernate_hbm2ddl_auto=update

# specify jdbc driver class
jdbc_driver_className=org.postgresql.Driver

# specify jdbc database user name
jdbc_username=postgres

# specify jdbc database password
jdbc_password=admin

# specify jdbc database host
jdbc_host=localhost

# specify jdbc database post
jdbc_port=5432
```

Control server

The default Control server properties file conf/control-server.properties looks as follows:

```
# Thrift configurations (more information about thrift look at http://thrift.apache.org/)
# The Control Server notifies every Operations/Bootstrap Server on most data updates via a Thrift-based
protocol.

# Thrift control server host
thrift_host=localhost

# Thrift control server port
thrift_port=9090

# Zookeeper service configuration
# Each Kaa cluster node (Kaa server) reports its state to Apache Zookeeper.
# Every node in the deployment can always obtain location of the active Control Server
# and the list of active Bootstrap and Operations Servers.

# Specifies if need to use zookeeper service. This is property have to be always "true".
# It is possible to change it for development or debug process.
zk_enabled=true

# Zookeeper service url list.
zk_host_port_list=localhost:2181

# The max retry time in milliseconds.
zk_max_retry_time=3000

# Time to sleep in milliseconds between searches for work.
zk_sleep_time=1000

# Specifies if need to throw runtime exception during registration control zookeeper node.
zk_ignore_errors=true

# Default Rebalance Class.
# Calculate average load for all Operations servers. This is help to make correct client load balancing.
dynamic_mgmt_class=org.kaaproject.kaa.server.control.service.loadmgmt.dynamicmgmt.DefaultRebalancer

# Recalculate period in seconds for Operations server load process.
recalculation_period=120

# Default TTL in seconds for historical information about Operations server load.
ops_server_history_ttl=3600
```

Bootstrap server

The default Bootstrap server properties file conf/bootstrap-server.properties looks as follows:

```
# Thrift configurations (more information about thrift look at http://thrift.apache.org/)
# The Bootstrap Server takes updates from Control server about operations server list via a Thrift-based
protocol.

# Thrift bootstrap server host
thrift_host=localhost

# Thrift bootstrap server port
thrift_port=9094

# Zookeeper service configuration

# Specifies if need to use zookeeper service. This is property have to be always "true".
# It is possible to change it for development or debug process.
zk_enabled=true

# Zookeeper service url
zk_host_port_list=localhost:2181

# The max retry time in milliseconds
zk_max_retry_time=3000

# Time to sleep in milliseconds between searches for work
zk_sleep_time=1000

# Specifies if need to throw runtime exception during registration control zookeeper node.
zk_ignore_errors=true

# Netty framework configurations. Bootstrap HTTP server implementation based on Netty framework.

# Netty bootstrap server host (HTTP protocol)
netty_host=localhost

# Netty bootstrap server port
netty_port=9889

# Netty bootstrap server host (KaaTcp protocol)
channel_kaa_tcp_host=localhost

# Netty bootstrap server port (KaaTcp protocol)
channel_kaa_tcp_port=9888

# Netty max POST request content size in bytes. See io.netty.handler.codec.http.
HttpObjectAggregator#maxContentLength
http_maxContentLength=8192

# Netty thread pool executor size
executorThreadSize=8

# Bootstrap server keys configurations.
# Each client have to know bootstrap public key to make successful connection to bootstrap server.

# Path Bootstrap  to private key
keys_private_key_location=keys/private.key

# Path to Bootstrap public key
keys_public_key_location=keys/public.key

# Number of statistics update during collect window
statistics_calculation_window=300

# Statistics collect window in seconds
statistics_update_times=60
```

Operations server

The default Operations server properties file conf/operations-server.properties looks as follows:

```
# Thrift configurations (more information about thrift look at http://thrift.apache.org/)
# The Operations Server take most data updates via a Thrift-based protocol.

# Thrift operations server host
thrift_host=localhost

# Thrift operations server port
thrift_port=9093

# Zookeeper service configuration
# Each Kaa cluster node (Kaa server) reports its state to Apache Zookeeper.
# Every node in the deployment can always obtain location of the active Control Server
# and the list of active Bootstrap and Operations Servers

# Specifies if need to use zookeeper service. This is property have to be always "true".
# It is possible to change it for development or debug process.
zk_enabled=true

# Zookeeper service url list
zk_host_port_list=localhost:2181

# The max retry time in milliseconds
zk_max_retry_time=3000

# Time to sleep in milliseconds between searches for work
zk_sleep_time=1000

# Specifies if need to throw runtime exception during registration of zookeeper node.
zk_ignore_errors=true

# Operations channels framework configurations. Operations HTTP server implementation based on Netty
framework.

#  server host (HTTP transport type)
channel_http_host=localhost

# server port (HTTP transport type)
channel_http_port=9999

#  server host (HTTP_LP transport type)
channel_http_lp_host=localhost

# server port (HTTP_LP transport type)
channel_http_lp_port=9998

#  server host (KAA_TCP transport type)
channel_kaa_tcp_host=localhost

#  server port (KAA_TCP transport type)
channel_kaa_tcp_port=9997

# Netty max POST request content size in bytes. See io.netty.handler.codec.http.
HttpObjectAggregator#maxContentLength
http_maxContentLength=524288

# Size of thread pool to process requests. Since request processing is delegated to Akka system, this size
should be small.
executorThreadSize=1

# Number of statistics update during collect window
statistics_calculation_window=300

# Statistics collect window in seconds
statistics_update_times=60

# Metrics collect enabled
metrics_enabled=true

# Operations server keys configurations.
# Each client have to know public key to make successful connection to operations server.
```

```
# Path to Operations server private key
keys_private_key_location=keys/private.key

# Path to Operations server public key
keys_public_key_location=keys/public.key

# Path to logs root directory
logs_root_dir=/kaa_log_uploads

# Date pattern for file log appender
date_pattern='.'yyyy-MM-dd-HH-mm

# Layout pattern for file log appender
layout_pattern=%m%n

# Path to tmp keys directory
tmp_keys=/home/kaa/tmp_keys

# Specify if support unencrypted connection
support_unencrypted_connection=true

# Specify the max number of neighbor connections
max_number_neighbor_connections=200
```

Web UI

The default web UI properties file conf/admin-server.properties looks as follows:

```
# Thrift configurations (more information about thrift look at http://thrift.apache.org/)
# The Web admin takes updates from Control server via a Thrift-based protocol.

# Thrift Web admin server host
control_thrift_host=localhost

# Thrift Web admin server port
control_thrift_port=9090
```

Further reading

Use the following guides and references to make the most of Kaa.

Guide	What it is for
Sandbox	Use this guide to try out Kaa in a private environment with demo applications.
Installation guide	Use this guide to install and configure Kaa on a single node or as a cluster.
Design reference	Use this reference to learn about features and capabilities of Kaa.
Programming guide	Use this guide to create your own Kaa applications.