Using the Agent SDK *Beta*

Overview

This tutorial is an overview of how to use the Agent SDK to build your own New Relic agent or to instrument your application. You'll learn how to use the Agent SDK to instrument and measure the performance of your application's web transactions and database operations.

The purpose of the SDK is to enable you to use New Relic for applications that were not written in one of our supported languages. The instructions in this tutorial assume your application is written in C or C++. If you are not using C or C++, then you will need to create a wrapper around the APIs. The means of creating a wrapper will vary from language to language. For example, with perl you can use SWIG or XS. Other languages have direct support for calling native functions.

Operating system and compiler support

The libraries in the SDK were built using glibc 4.12. If you need a release built using an older version of glibc, create a request via the standard support channels.

The SDK only supports Linux at this time.

Prerequisites

The SDK uses boost, which dynamically links to librt and pthreads. These libraries will need to be installed on your system. Additionally, the SDK uses libcurl for http requests, which will also need to be installed.

Choosing a usage mode

In order to maintain communication with New Relic, the Agent SDK requires a thread to be started that harvests data once per minute. This means that your app will need to be able to start a thread, which will run continuously as long as transactions are being executed. For some languages, this is simple to do. For others, there is no mechanism to start a thread within the same process running your web transactions.

For example, if you are running perl apps through Apache, then each perl script is a short-lived process that exits upon completion of the script. There is no ability to startup a thread that lives beyond the life of the script.

To solve this problem, we designed the SDK so that your agent can run in two different modes: daemon-mode or embedded-mode. In daemon-mode, you'll run the `newrelic-collector-client-daemon` that came with the Agent SDK as a stand-alone process. The daemon will collect and send data to New Relic. In embedded-mode your application will start a thread, which will have the same responsibilities as the daemon but will be running within the same process as your transactions.

If you choose to run in daemon-mode read “How to run in Daemon-mode.” Otherwise read “How to run in Embedded-mode.” After that, you should read “How to Instrument Your Code.”

Environment variable	Format/ Example	Default(s)	Required for Daemon-mode
NEWRELIC_LICENSE_KEY	<your license key>	–	Yes
NEWRELIC_APP_NAME	Hello World	–	Yes
NEWRELIC_APP_LANGUAGE	Perl	–	Yes
NEWRELIC_APP_LANGUAGE_VERSION	5.18.2	–	Yes

NEWRELIC_ENABLE_SSL	0 or 1	1 (enabled)	No
NEWRELIC_ENABLE_HIGH_SECURITY	0 or 1	0 (disabled)	No
NEWRELIC_LOG_PROPERTIES_FILE	<path/to/log4cplus.properties>	(1) ~/.newrelic (2) <current working dir>	No
NEWRELIC_APP_SERVER_APDEX_T	0.25	0.5	No
http_proxy	http://<username>:<password>@<proxy_server>:<port>	–	No

Name	Format/ Example	Default(s)	Required for Daemon-mode
NEWRELIC_LICENSE_KEY	<your license key>	–	Yes
NEWRELIC_APP_NAME	"Hello World"	–	Yes
NEWRELIC_APP_LANGUAGE	"Perl"	–	Yes
NEWRELIC_APP_LANGUAGE_VERSION	5.18.2	–	Yes
NEWRELIC_ENABLE_SSL	0 or 1	1 (enabled)	No
NEWRELIC_ENABLE_HIGH_SECURITY	0 or 1	0 (disabled)	No
NEWRELIC_LOG_PROPERTIES_FILE	<path/to/log4cplus.properties>	(1) ~/.newrelic (2) <current working dir>	No
NEWRELIC_APP_SERVER_APDEX_T	0.25	0.5	No
http_proxy	"http://<name>:<passwd>@<proxy>:<port>"	–	No

How to run in Daemon-mode

Note: The SDK daemon will run with the privileges of the application starting it, which may be elevated. For example, if the daemon is started by Apache, it may run as root. You should run the daemon by a non-root user if possible.

1. Set up your environment

Set the required environment variables before launching the daemon (see table above).

2. Start the daemon process

Execute the following command in a terminal window:

```
./newrelic-collector-client-daemon
```

How to run in Embedded-mode

To run in embedded-mode, you'll need to use `libnewrelic-collector-client` in your web server process.

1. Embed `libnewrelic-collector-client` in your application

Follow these instructions (in this order) to embed the collector client in your app:

1. Include the `newrelic_collector_client.h` and `newrelic_common.h` header files. Link your application to `libnewrelic-common` and `libnewrelic-collector-client`.

2. (Optional) Create a function to receive status change notifications:

```
void newrelic_status_update(int status) {
	if (status == NEWRELIC_STATUS_CODE_SHUTDOWN) {
		// do something when the SDK shuts down
	}
}

newrelic_register_status_callback(newrelic_status_update);
```

3. Register required callback that will send data to New Relic when transactions are completed. You will need to register the default callback already defined in libnewrelic-collector-client (`newrelic_message_handler`).

```
newrelic_register_message_handler(newrelic_message_handler);
```

4. Initialize the library, for example:

```
newrelic_init("my_license_key", "My Application", "perl", "5.5");
```

5. (Optional) Shutdown the connection to New Relic

```
newrelic_request_shutdown("insert your reason here");
```

How to record a custom metric

In this section, you will learn how to report a custom metric. Custom metrics give you a way to record arbitrary metrics about your application. You can also instrument your code, which will report performance metrics automatically whenever that code is executed. With a custom metric, you provide the value to be recorded for a specified metric name, like this:

```
newrelic_record_metric("ActiveUsers", 25);
```

To view custom metrics, create a custom dashboard from the metrics you collect.

How to instrument your code

Follow these instructions (in this order) to measure transactions and database operations in your application:

1. Include the `newrelic_transaction.h` and `newrelic_common.h` header files. Link your application to `libnewrelic-common` and `libnewrelic-transaction`.

2. Create a web transaction and configure its settings.

```
/* Start a transaction */
long transaction_id = newrelic_transaction_begin();
newrelic_transaction_set_name
(
       transaction_id,
       "create_account"
);
```

3. Measure segments:

```
/* Start a generic segment */
long segment_id = newrelic_segment_generic_begin
(
       transaction_id,
       parent_id,
       "check_if_account_exists"
);

<execute some code>

newrelic_segment_end
(
       transaction_id,
       segment_id
);

/* Start a datastore segment */
long datastore_segment_id = newrelic_segment_datastore_begin
(
       transaction_id,
       parent_id,
       "customers",
       NEWRELIC_DATASTORE_SELECT,
       "SELECT account_id FROM accounts WHERE username = 'joe'"
       "get_acccount_id",
       newrelic_basic_literal_replacement_obfuscator
```

```
);

<execute a database query>

newrelic_segment_end
(
        transaction_id,
        datastore_segment_id
);
```

4. Set transaction type if it is a not a web transaction.

```
/* Set transaction type to Other (Non-Web) */
newrelic_transaction_set_type_other
(
        transaction_id
);
```

5. End your transaction.

```
/* End transaction */
int error_code = newrelic_transaction_end
(
        transaction_id
);
```

That is all it takes to get up and running with the Agent SDK! Fire up your application and you will start seeing data in New Relic.

Transaction Trace

Before learning how to construct a Transaction Trace, it is important to understand the components that make up a trace.

Component	Definition
Segment	A node of a transaction trace representing a timed operation. e.g.: function calls, database operations
External Segment	A timed segment representing an external service call e.g.: making an external web request
Datastore Segment	A timed segment representing a database operation e.g.: a select statement

Generic Segment	A timed segment representing a generic operation. e.g.: function/method calls, complex chunks of code, etc
Transaction	A timed operation consisting of multiple segments.
Transaction Trace	A collection of nodes representing all the segments of a transaction

Transaction traces are automatically generated when transactions exceed the transaction trace threshold (4 x ApdexT). ApdexT has a default value of 0.5 seconds, so the default trace threshold for a transaction is 2 seconds.

The process to create a Transaction Trace is simple:

- Start and name a transaction
- For each important section of code you'd like to measure:
    - Start a segment and relate it to the transaction and its parent segment
    - <your code here>
    - End the segment
- End the transaction
    - If the transaction duration is longer than the transaction trace threshold, then a trace will be collected.
    - Note: a maximum of two traces will be sent to New Relic per minute.

SQL Trace

An SQL trace is automatically generated for each Datastore segment. Every minute the agent sends several of these traces up to New Relic. The traces selected will represent the worst performing SQL statements during that minute.

SQL Obfuscation

Your Database code may contain sensitive information that you don't want (or need) to send to New Relic. By default, the Agent SDK obfuscates your SQL strings. It uses a basic literal replacement obfuscator that strips the SQL string literals and numeric sequences, replacing them with the ? character. For example,

This SQL:

```
SELECT * FROM table WHERE ssn='000-00-0000'
```

obfuscates to:

```
SELECT * FROM table WHERE ssn=?
```

Because our default obfuscator just replaces literals, there could be cases that it does not handle well. For instance, it will not strip out comments from your SQL string, it will not handle certain database-specific language features, and it could fail for other complex cases.

If this level of obfuscation is not sufficient, you can supply your own custom obfuscator.

Apdex T

Apdex T is the response time threshold used to measure customer satisfaction. The transaction trace threshold is calculated as four times the Apdex T value. The default Apex T value is 0.5 seconds.

You can change the Apdex T used across all transactions of your application using the environment variable `NEWRELIC_APP_SERVER_APDEX_T`. If you don't set this environment variable, the Apdex T will default to 0.5 seconds.

If you want to change the Apdex T for a specific transaction, go to New Relic and make it a Key Transaction. This will give you the option of changing its Apdex T. Once it's been saved, your agent will be notified the next time it's ready to send data up to New Relic.

Challenges

There are a few challenges you might face when creating transactions and transaction traces for your application or language:

1. How to keep track of shared data (e.g. parent id's)
2. How to name transactions
3. How to deal with threading considerations

Keeping track of shared data (e.g. parent id's)

In order to build a transaction trace, you need to provide information to the SDK about the relationships between segments. You do this by passing parent id's when starting new segments. Depending on your language, you might be able to take advantage of the auto-scoping feature of the SDK.

Auto-scoping can be used to automatically identify the transaction id and parent id of a newly started segment. It works by keeping track of the last segment started within the current thread (using thread local storage). It can be used in cases where a transaction runs uninterrupted from beginning to end within the same thread. Note: see the threading section below for details about various threading considerations within the SDK.

If auto-scoping does not work, you will need to find a way to keep track of the current call stack. A simple way to do this is to keep a stack of segment ids in memory. When a new segment

starts, add it to the stack. When a segment ends, pop it from the stack. The challenge then becomes one of identifying where to store the stack. Here are some options:

1. You can use a global web context object if your language supports it
2. You can use thread local storage
3. You can create a global map of running transactions

Note: For segments at the root of the transaction, the parent_segment_id should be set to `NEWRELIC_ROOT_SEGMENT`.

Naming transactions

Transactions should be given names based on the purpose of the transaction.

Use caution when naming transactions. Issues come about when the granularity of transaction names is too fine, resulting in hundreds if not thousands of different transactions. If you name transactions poorly, your application may get blacklisted due to too many metrics being sent to New Relic.

A basic rule is to name transactions based on the controller action rather than the URL. To learn more, read https://docs.newrelic.com/docs/features/metric-grouping-issues

Threading considerations

Different languages support different threading models. Depending on the characteristics of your threading model, you will need to consider how your threads will interact with the SDK.

Simple case

In some languages, all segments of a transaction occur within the same thread. Additionally, a transaction will run from start to finish on the thread before another transaction begins on the same thread. This threading model is fully supported by the SDK.

Multiplexing

Some languages are single threaded, but can support multiple “simultaneous” transactions. In these languages, when a routine is blocked on I/O, it can be switched out in favor of a different routine on the same thread. The SDK should work in this scenario if you keep track of transaction IDs and parent IDs.

Multi-threading within a single transaction

This is thread-safe in the Agent SDK as of version 0.14 Beta.

How to configure logging

The Agent SDK uses log4cplus for logging and gives you flexibility into log levels, log file locations, etc.

The SDK searches for a log4cplus.properties file in the following locations (in order):

1. Location specified by NEWRELIC_LOG_PROPERTIES_FILE environment variable.

2. In your $HOME/.newrelic directory
3. In the current working directory

To configure logging:

- Copy the config/log4cplus.properties file to one of the locations specified above
- If desired, modify the log4cplus.properties file per these instructions: http://log4cplus.sourceforge.net

Understanding errors and error IDs

All functions in the SDK return integers or longs. Any positive number indicates success and any negative number indicates a failure.

See the New Relic header files for a list of all error codes that might be returned.

Disabling the SDK and setting limits

The SDK was designed to have a minimal impact to your system out-of-the-box. However, in addition to the out-of-the-box settings, we have provided some controls to dial back or disable the collection of data.

The following table describes the controls available to you:

Control	How to use it
Disable the collection of data during a transaction	newrelic_enable_instrumentation(false); **Note**: if you are running a web server that spawns off new processes per transaction, you may need to call this for every transaction.
Shut down the agent	If running in embedded mode: newrelic_request_shutdown("reason for shutting down…"); If running in daemon mode: Kill the newrelic-collector-client-daemon process.
Configuring transaction trace thresholds	newrelic_transaction_set_threshold(transaction_id, 5000); **Note**: if you are running a web server that spawns off new processes per transaction, you may need to call this for every transaction.

Configuring the number of trace segments collected in a transaction trace	// Only collect up to 50 trace segments
	newrelic_transaction_set_max_trace_segments(transaction_id, 50);
	
	Note: if you are running a web server that spawns off new processes per transaction, you may need to call this for every transaction.

Security Notes

This is prerelease software and should be treated as such. We are giving you early access to the beta, so please use it as an opportunity to provide feedback and guide us as we add more features and work towards a production release of the SDK.

By default, the Agent SDK enables SSL for communication between the SDK and New Relic. However, developers are responsible for obfuscating or hiding any sensitive data that should not appear in the New Relic UI. For instance, you may want to hide certain parameters inside a request url or sql query.

Use caution when naming custom metrics, transactions, and segments. Issues come about when the granularity of names is too fine, resulting in metric explosion which can impact the performance of both your application and New Relic. Your application may also get blacklisted. To learn more about this issue and best practices, read https://docs.newrelic.com/docs/features/metric-grouping-issues