sqlbox cvs-2008.11.03 User's Guide

SQL-Based queue engine for Kannel

Renee Kluwen
Sqlbox Author
Chimit

rene.kluwen at chimit dot nl
http://www.chimit.nl/

Martin Conte
Standalone Version and Patches

reflejo at gmail dot com

Alejandro Guerrieri
Maintainer, Documentation and Patches
Magicom

aguerrieri at kannel dot org
http://www.blogalex.com/

sqlbox cvs-2008.11.03 User's Guide : SQL-Based queue engine for Kannel

by Renee Kluwen, Martin Conte, and Alejandro Guerrieri

Abstract

This document describes how to install and use sqlbox, the SQL-Based queue engine for Kannel.

Revision History

Revision cvs- 2008.11.03

Table of Contents

List of Tables

Chapter 1. Introduction

Sqlbox is a special Kannel box that sits between bearerbox and smsbox and uses a database queue to store and forward messages.

Overview

Sqlbox behaves similar to other Kannel boxes and share a compatible configuration file format and command line options.

It works between bearerbox and smsbox, intercept all messages and use a couple of database tables to process messages.

Messages are queued on a configurable table (defaults to `send_sms`) and moved to another table (defaults to `sent_sms`) afterwards.

You can also manually insert messages into the `send_sms` table and they will be sent and moved to the `sent_sms` table as well. This allows for fast and easy injection of large amounts of messages into kannel.

Features

- Modular architecture: Easily integrates into Kannel infrastructure.
- Compatible configuration file format and command line arguments.
- Supports most Kannel features.

Requirements

sqlbox is being developed on Linux and OSX systems, and should be fairly easy to export to other Unix-like systems. However, we don't yet support other platforms, due to lack of time, although it should be working without major problems on Windows (through Cygwin), Mac OSX, Solaris and FreeBSD.

sqlbox requires the following software environment:

- Kannel libraries (gwlib) installed.
- C compiler and libraries for ANSI C, with normal Unix extensions such as BSD sockets and related tools. (GNU's GCC tool-chain is recommended)
- GNU Make.
- An implementation of POSIX threads (`pthread.h`).

- DocBook processing tools: DocBook style-sheets, jade, jadetex, etc; see `README` , section 'Documentation', for more information (pre-formatted versions of the documentation are available, and you can compile Sqlbox itself even without the documentation tools).
- GNU autoconf

Chapter 2. Installing sqlbox

This chapter explains how to build and install sqlbox from source or from a binary package. The goal of this chapter is to get the module compiled and all the files in the correct places; the next chapter will explain how to configure it.

> **Note:** If you are upgrading from a previous version, please look at Appendix A for any important information.

Getting the source code

The source code to Sqlbox is available for download at http://www.kannel.org/download.shtml . It is available in various formats and you can choose to download either the latest release version or the daily snapshot of the development source tree for the next release version, depending on whether you want to use Sqlbox for production use or to participate in the development.

If you're serious about development, you probably want to use CVS, the version control system used by the Kannel project. This allows you to participate in Kannel development much more easily than by downloading the current daily snapshot and integrating any changes you've made every day. CVS does that for you. (See the Kannel web site for more information on how to use CVS.)

Finding the documentation

The documentation for Sqlbox consists of two parts:

1. *User's Guide* , i.e., the one you're reading at the moment.
2. The `README` and various other text files in the source tree.

You can also find general information on Kannel's website (http://www.kannel.org) and information about existing problems at our bugtracker (http://bugs.kannel.org) .

We intend to cover everything you need to install and use Sqlbox is in *User's Guide* , but the guide is still incomplete in this respect. The `README` is not supposed to be very important, nor contain much information. Instead, it will just point at the other documentation.

Compiling sqlbox

If you are using Sqlbox on a supported platform, or one that is similar enough to one, compiling Sqlbox should be trivial. After you have unpacked the source package of your choose, or after you have checked

out the source code from CVS, enter the following commands:

```
./bootstrap ./configure make
```

The `bootstrap` script uses autoconf to generate the files needed to build the module. The `configure` script investigates various things on your computer for the Sqlbox compilation needs, and writes out the `Makefile` used to compile the module. **make** then runs the commands to actually compile it.

If either command writes out an error message and stops before it finishes its job, you have a problem, and you either need to fix it yourself, if you can, or report the problem to the Kannel project. See Chapter 4 for details.

For detailed instruction on using the configuration script, see file `INSTALL` . That file is a generic documentation for **configure** . Sqlbox defines a few additional options:

- `--with-kannel-dir=` *`DIR`* Where to look for Kannel Gateway libs and header files *`DIR`* points to the Kannel installation directory. Defaults to `/usr/local`
- `--disable-docs (default is --enable-docs)` Use this option if you don't have DocBook installed and/or you want to save some time and CPU cycles. Pre-generated documentation is available on Kannel's site. Default behavior is to build documentation, b.e., converting the User Guide from the DocBook markup language to PostScript and HTML if DocBook is available.
- `--enable-drafts (default is --disable-drafts)` When building documentation, include the sections marked as `draft` .
- `--with-ctlib=`*`DIR`* Include Ct-Lib support. *`DIR`* is the Ct-Lib install directory, defaults to `/opt/sybase`.
- `--with-freetds=`*`DIR`* Include FreeTDS Ct-Lib support. *`DIR`* is the FreeTDS install directory, defaults to `/usr/local`.

You may need to add compilations flags to configure:

```
CFLAGS='-pthread' ./configure
```

The above, for instance, seems to be required on FreeBSD. If you want to develop Sqlbox, you probably want to add CFLAGS that make your compiler use warning messages. For example, for GCC:

```
CFLAGS='-Wall -O2 -g' ./configure
```

(You may, at your preference, use even stricter checking options.)

Installing Sqlbox

After you have compiled Kannel, you need to install the sqlbox binary in a suitable place. This is most easily done by using **make** again:

```
make bindir=/path/to/directory install
```

Replace */path/to/directory* with the pathname of the actual directory where the programs should be installed. This install the sqlbox binary:

```
gw/sqlbox
```

Using pre-compiled binary packages

To be done

Installing Sqlbox from RPM packages

To be done

Installing Sqlbox from DEB packages

To be done

Chapter 3. Using sqlbox

This chapter explains how to configure and run Sqlbox and also how to tell if it's running from Kannel's HTTP interface.

There is only one configuration file for Sqlbox, and that file commands all aspects of its execution.

Configuring Sqlbox

The configuration file can be divided into two parts: sqlbox configuration and database connection.

Details of each part are in appropriate sections later on this documentation.

Configuration file syntax

The syntax used for the configuration file is the same used in Kannel. Skip this section if you are already familiar with it. Otherwise, keep on reading:

A configuration file consists of groups of configuration variables. Groups are separated by empty lines, and each variable is defined on its own line. Each group in Sqlbox configuration is distinguished with a group variable. Comments are lines that begin with a number sign (`#`) and are ignored (they don't, for example, separate groups of variables).

A variable definition line has the name of the variable, and equals sign (`=`) and the value of the variable. The name of the variable can contain any characters except whitespace and equals. The value of the variable is a string, with or without quotation marks ( ) around it. Quotation marks are needed if the variable needs to begin or end with whitespace or contain special characters. Normal C escape character syntax works inside quotation marks.

Perhaps an example will make things easier to comprehend:

```
01 # Sqlbox configuration
02 group = sqlbox
03 id = "my-sqlbox"
04 smsbox-id = "sqlbox"
...
11 log-level = 0
12 log-file = "/var/log/kannel/kannel-sqlbox.log"
13
14 #MySQL Connection
15 group = mysql-connection
16 id = "my-sqlbox"
17 host = localhost
...
```

The above snippet defines an sqlbox instance with id `my-sqlbox` that identifies with `bearerbox` as `sqlbox` and also sets the log-level and file location. It also defines a MySQL connection to localhost.

Lines 1 and 14 are comment lines. Line 13 separates the two groups. The remaining lines define variables. The group type is defined by the group variable value.

The various variables that are understood in each type of configuration group are explained below.

Some variable values are marked as `'bool'`. The value for variable can be like true, false, yes, no, on, off, 0 or 1. Other values are treated as 'true' while if the variable is not present at all, it is treated as being 'false'.

Inclusion of configuration files

A configuration file may contain a special directive called `include` to include other file or a directory with files to the configuration processing.

This allows to segment the specific configuration groups required for several services and boxes to different files and hence to have more control in larger setups.

Here is an example that illustrates the `include` statement :

```
group = sqlbox
id = my-sqlbox
smsbox-id = sqlbox
...
log-file = "/var/log/kannel/kannel-sqlbox.log"
log-level = 0
include = "dbconn.conf"
```

Above is the main `sqlbox.conf` configuration file that includes the following `dbconn.conf` file with all required directives for the database connection.

```
group = mysql-connection
id = my-sqlbox
host = localhost
username = myuser
password = mypass
database = kannel
```

The above `include` statement may be defined at any point in the configuration file and at any inclusion depth. Hence you can cascade numerous inclusions if necessary.

At process start time inclusion of configuration files breaks if either the included file can not be opened and processed or the included file has been processed already in the stack and a recursive cycling has been detected.

Sqlbox configuration

The configuration file *MUST* always include an 'sqlbox' group for general configuration. This group should be the first group in the configuration file.

As its simplest form, 'sqlbox' group looks like this:

```
group = sqlbox
id = sqlbox
bearerbox-port = 13001
```

Naturally this is usually not sufficient for any real use. Thus, one or more of the optional configuration variables are used. In following list (as in any other similar lists), all mandatory variables are marked with `(m)`, while conditionally mandatory (variables which must be set in certain cases) are marked with `(c)` .

Table 3-1. Sqlbox Group Variables

Variable	Value	Description
`group (m)`	`sqlbox`	This is a mandatory variable
`smsbox-id (m)`	string	This is the box id.
`global-sender`	number	If no explicit number is given, this number is used when sending messages.
`bearerbox-host (m)`	host-name	This is the host where bearerbox is running.
`bearerbox-port (m)`	port-number	This is the port number used to connect to bearerbox.
`smsbox-port (c)`	port-number	This is the port number to which the smsboxes, if any, connect. This can be anything you want. Must be set if you want to handle any SMS traffic.
`smsbox-port-ssl (o)`	bool	If set to true, the smsbox connection module will be SSL-enabled. Your smsboxes will have to connect using SSL to sqlbox then. This is used to secure communication between sqlbox and smsboxes in case they are in separate networks operated and the TCP communication is not secured on a lower network layer. Defaults to "no".
`sql-log-table`	table-name	Indicates the table where messages are copied after being sent. Defaults to `sent_sms`.
`sql-insert-table`	table-name	Indicates the table where messages should be inserted to sent. Defaults to `send_sms`.
`log-file`	filename	A file in which to write a log. This in addition to `stdout` and any log file defined in command line.

Variable	Value	Description
log-level	number 0..5	Minimum level of log-file events logged. 0 is for 'debug', 1 'info', 2 'warning, 3 'error' and 4 'panic' (see Command Line Options)

A sample more complex 'sqlbox' group could be something like this:

```
group = sqlbox
id = sqlbox-db
smsbox-id = sqlbox
#global-sender = ""
bearerbox-host = localhost
bearerbox-port = 13001
smsbox-port = 13005
smsbox-port-ssl = false
sql-log-table = sent_sms
sql-insert-table = send_sms
log-file = "/var/log/kannel/kannel-sqlbox.log"
log-level = 0
```

The DB Connection

`sqlbox` needs a connection to a supported DB engine to operate. This connection is established at startup time and kept open until the box stops.

At the moment, `sqlbox` only supports MySQL and PostgreSQL, with support for other engines being planned.

The process of configuring a DB connection is simple: You need to create a `[engine]-connection` section (where [engine] is the DB engine name, either mysql or pgsql) and indicate a few parameters needed to establish the DB connection.

Database Connection Configuration

Table 3-2. Sqlbox Database connection configuration variables

Variable	Value	Description
group	mysql-connection	This is a mandatory variable if we're connecting to a MySQL database.
group	pgsql-connection	This is a mandatory variable if we're connecting to a PostgreSQL database.

Variable	Value	Description
id (m)	string	An id to identify which external connection should be used for Sqlbox storage. Any string is acceptable, but semicolon ';' may cause problems, so avoid it and any other special non-alphabet characters.
host (m)	string	The hostname where the DB engine is running.
username (m)	string	The username used to connect to the DB engine.
password (m)	string	The password used to connect to the DB engine.
database (m)	string	The database name to use to store the data.
max-connections	number	Create a pool with this number of connections open.

MySQL Storage

Uses a MySQL database to store the data. You need to specify the `mysql-connection` group.

Here is an example configuration:

```
group = mysql-connection
id = my-sqlbox
host = localhost
username = foo
password = bar
database = kannel
max-connections = 1
```

PostgreSQL Storage

Uses a PostgreSQL database to store the data. You need to specify the `pgsql-connection` group.

Here is an example configuration:

```
group = pgsql-connection
id = pg-sqlbox
host = localhost
username = foo
password = bar
database = kannel
max-connections = 1
```

Running Sqlbox

You need to start `sqlbox` after starting the `bearerbox`, otherwise it won't have a port open to connect to. The preferred way to do this is to include sqlbox into your Kannel's startup script.

Starting the box

If you want to start it from command line (for testing, for example), give the following command:

```
/path/to/sqlbox -v 1 [config-file]
```

The `-v 1` sets the logging level to `INFO`. This way, you won't see a large amount of debugging output (the default is `DEBUG`). Full explanation of Sqlbox command line arguments is below.

[config-file] is the name of the configuration file you are using with Sqlbox. The basic distribution packet comes with a sample configuration file you can use with some minor tweakings (check on the `/examples` folder. Feel free to edit the file to suit your needs.

Of course you need to have the `bearerbox` running before starting the box. Without the bearer box, sqlbox won't even start.

Command line options

Sqlbox accept certain command line options and arguments when they are launched. These arguments are:

Table 3-3. Sqlbox Command Line Options

`-v <level>` `--verbosity <level>`	Set verbosity level for stdout (screen) logging. Default is 0, which means 'debug'. 1 is 'info, 2 'warning', 3 'error' and 4 'panic'
`-D <places>` `--debug <places>`	Set debug-places for 'debug' level output.
`-F <file-name>` `--logfile <file-name>`	Log to file named file-name, too. Does not overrun or affect any log-file defined in configuration file.
`-V <level>` `--fileverbosity <level>`	Set verbosity level for that extra log-file (default 0, which means 'debug'). Does not affect verbosity level of the log-file defined in configuration file, not verbosity level of the `stdout` output.
`-H`	Only try to open HTTP sendsms interface; if it fails, only warn about that, do not exit. (smsbox only)

`--tryhttp`	
`-g` `--generate`	Dump all known config groups and config keys to stdout and exit.
`-u <username>` `--user <username>`	Change process user-id to the given.
`-p <filename>` `--pid-file <filename>`	Write process PID to the given file.
`-d` `--daemonize`	Start process as daemon (detached from a current shell session). Note: Process will change CWD (Current working directory) to `/`, therefore you should ensure that all paths to binary/config/config-includes are absolute instead of relative.
`-P` `--parachute`	Start watcher process. This process watch a child process and if child process crashed will restart them automatically.
`-X <scriptname>` `--panic-script <scriptname>`	Execute a given shell script or binary when child process crash detected. This option is usable only with `--parachute/-P`. Script will be executed with 2 arguments: scriptname 'processname' 'respawn-count'.

Database Tables

Sqlbox creates it's DB tables on the fly if the tables are not present at that moment. If you're upgrading from a previous version, or happen to have tables with the same names as the ones Sqlbox uses, but having a different structure, this will probably cause problems and there's a good chance the process will panic and stop. In that case, rename/drop the offending tables or change the names Sqlbox uses by using the `sql-log-table` and `sql-insert-table` variables.

Inserting MT messages by SQL

One of the nice features Sqlbox provides is the ability to insert MT messages into Kannel's queue by inserting rows into the `send_sms` table. Keep in mind that both tables have the same schema, but you

only need to care about `send_sms`. Sqlbox will move messages to the `sent_sms` table autmatically after processing it.

Database Structure

The tables structure is as follows:

Table 3-4. Sqlbox Database structure

Value	Type	Description	sendsms equivale
`sql_id`	`BIGINT(20)`	This is the auto-incremented PRIMARY KEY and should be always left alone. Set it to NULL or do not include it in your INSERT query.	-
`momt`	`ENUM('MO', 'MT')`	Specifies if the message is either MO or MT. You should always use "MT" here.	-
`sender`	`VARCHAR(20)`	Phone number of the sender. If this variable is not set, sqlbox `global-sender` is used.	from
`receiver`	`VARCHAR(20)`	Phone number of the receiver.	to
`msgdata`	`TEXT`	Contents of the message, URL encoded as necessary. The content can be more than 160 characters, but then Kannel's `sendsms-user` group must have `max-messages` set more than 1.	text
`udhdata`	`BLOB`	Optional User Data Header (UDH) part of the message. Must be URL encoded.	udh

`time`	`BIGINT(20)`	An integer timestamp. You can uses UNIX_TIMESTAMP() on MySQL or any similar function here. You can also leave the field empty/alone if you don't care about having a timestamp on your messages.	-
`smsc_id`	`VARCHAR(255)`	Optional virtual smsc-id from which the message is supposed to have arrived. This is used for routing purposes, if any denied or preferred SMS centers are set up in SMS center configuration. This variable can be overridden on Kannel with a `forced-smsc` configuration variable. Likewise, the `default-smsc` variable can be used to set the SMSC if it is not set otherwise.	smsc
`service`	`VARCHAR(255)`	Optional. Service name from which the message is supposed to have arrived. This field is logged as SVC in the log file so it allows you to do some accounting on it if your front end uses the same username for all services but wants to distinguish them in the log.	smsc

`account`	`VARCHAR(255)`	Optional. Account name or number to carry forward for billing purposes. This field is logged as ACT in the log file so it allows you to do some accounting on it if your front end uses the same username for all services but wants to distinguish them in the log. In the case of a HTTP SMSC type the account name is prepended with the service-name (username) and a colon (:) and forwarded to the next instance of Kannel. This allows hierarchical accounting.	`account`
`id`	`BIGINT(20)`	Kannel's internal message identifier. This have no meaning when you're inserting your own messages, since Kannel doesn't have an identifier on your message yet. Leave it alone.	–
`sms_type`	`BIGINT(20)`	A numeric value indicating if it's an MO, MT or DLR message. ALWAYS INSERT A "2" HERE (Meaning: MT), OTHERWISE KANNEL'S QUEUE WILL GET CORRUPTED IF YOU RESTART IT AND YOU HAVE PENDING MESSAGES.	–

`mclass`	`BIGINT(20)`	Optional. Sets the Message Class in DCS field. Accepts values between 0 and 3, for Message Class 0 to 3, A value of 0 sends the message directly to display, 1 sends to mobile, 2 to SIM and 3 to SIM toolkit.	`mclass`
`mwi`	`BIGINT(20)`	Optional. Sets Message Waiting Indicator bits in DCS field. If given, the message will be encoded as a Message Waiting Indicator. The accepted values are 0,1,2 and 3 for activating the voice, fax, email and other indicator, or 4,5,6,7 for deactivating, respectively. [a]	`mwi`
`coding`	`BIGINT(20)`	Optional. Sets the coding scheme bits in DCS field. Accepts values 0 to 2, for 7bit, 8bit or UCS-2. If unset, defaults to 7 bits unless a udh is defined, which sets coding to 8bits.	`coding`
`compress`	`BIGINT(20)`	Optional. Sets the Compression bit in DCS Field.	`compress`

validity	BIGINT(20)	Optional. If given, Kannel will inform SMS Center that it should only try to send the message for this many minutes. If the destination mobile is off other situation that it cannot receive the sms, the smsc discards the message. Note: you must have your Kannel box time synchronized with the SMS Center.	validity
deferred	BIGINT(20)	Optional. If given, the SMS center will postpone the message to be delivered at now plus this many minutes. Note: you must have your Kannel box time synchronized with the SMS Center.	deferred
dlr-mask	BIGINT(20)	Optional. Request for delivery reports with the state of the sent message. The value is a bit mask composed of: 1: Delivered to phone, 2: Non-Delivered to Phone, 4: Queued on SMSC, 8: Delivered to SMSC, 16: Non-Delivered to SMSC. Must set dlr-url on sendsms-user group or use the sendsms dlr-url variable or Sqlbox column.	dlr-mask
dlr-url	VARCHAR(255)	Optional. If dlr-mask is given, this is the url to be fetched. (Must be url-encoded)	dlr-url

`pid`	`BIGINT(20)`	Optional. Sets the PID value. (See ETSI Documentation). Ex: SIM Toolkit messages would use something like `pid=127, coding=1, alt-dcs=1, mclass=3`	`pid`
`alt-dcs`	`BIGINT(20)`	Optional. If unset, Kannel uses the alt-dcs defined on smsc configuration, or 0X per default. If equals to 1, uses FX. If equals to 0, force 0X.	`alt-dcs`
`rpi`	`BIGINT(20)`	Optional. Sets the Return Path Indicator (RPI) value. (See ETSI Documentation).	`rpi`
`charset`	`VARCHAR(255)`	Charset of text message. Used to convert to a format suitable for 7 bits or to UCS-2. Defaults to WINDOWS-1252 if coding is 7bits and UTF-16BE if coding is UCS-2.	`charset`
`boxc_id`	`VARCHAR(255)`	The bearerbox ID that should handle this message. You can usually leave this one alone.	`charset`

`binfo`	`VARCHAR(255)`	Optional. Billing identifier/information proxy field used to pass arbitrary billing transaction IDs or information to the specific SMSC modules. For EMI2 this is encapsulated into the XSer 0c field, for SMPP this is encapsulated into the service_type of the submit_sm PDU.	`binfo`

Notes:

a. To set number of messages, use `mwi=[0-3]&coding=0&udh=%04%01%02%<XX>%<YY>`, where YY are the number of messages, in HEX, and XX are `mwi` plus 0xC0 if `text` field is not empty.

Example

As when you're using the `sendsms` interface, you don't need to specify all the columns in order to succesfully enqueue a message.

Here's an example query you can use to send a simple message using Sqlbox:

```
INSERT INTO send_sms (
  momt, sender, receiver, msgdata, sms_type
) VALUES (
  'MT', '1234', '1234567890', 'Hello world', 2
);
```

The former example would send a message with text "Hello world" to number "1234567890". If possible, the sender would be set to "1234".

You can add other parameters to specify routing, charset encoding and any other settings your setup may require. Just remember, try to keep it simple whenever possible

Chapter 4. Getting help and reporting bugs

This chapter explains where to find help with problems related to the gateway, and the preferred procedure for reporting bugs and sending corrections to them.

The Kannel development mailing list is devel@kannel.org. To subscribe, send mail to devel-subscribe@kannel.org (mailto:devel-subscribe@kannel.org). This is currently the best location for asking help and reporting bugs. Please include configuration file and version number.

Appendix A. Upgrading notes

This appendix includes pertinent information about required changes on upgrades.

As a general rule, always check the `ChangeLog` file before upgrading, because it may contain important information worth knowing before making any changes.

Upgrading from different sqlbox versions

Sqlbox is a simple module that usually upgrades easily and without requiring any other changes.

In some cases, a change on the DB structure takes place and this requires changes on the DB schemas as well. Since `sqlbox` automatically generates its tables, the best approach for this kind of upgrades is to make sure that there's no messages pending, backup the tables contents (if there's no messages pending only the `sent_sms` table will have records), drop the tables and let `sqlbox` create the tables again. Alternatively you can check what changes are necessary and ALTER the tables yourself.