Lovebeat Documentation

Release 1.0

Victor Boivie

Nov 14, 2017

Contents

Have you ever had a nightly backup job fail, and it took you weeks until you noticed it? Say hi to lovebeat, a zero-configuration heartbeat monitor.

Other use-cases include creating triggers for manual tasks that you haven't automated yet (like moving your backup tapes to an offsite location, watering your plants or changing sheets in your bed). It can also be used for the opposite - for finding out when things start to happen that shouldn't, like your frontend calling deprecated methods in the backend.

Lovebeat provides a lot of different APIs. We recommend the "statsd"-compatible UDP protocol when adding triggers to your software for minimum impact on your performance (when you're running Lovebeat on the same machine or trust the network to deliver your UDP packets). A TCP protocol is available if you don't trust UDP and HTTP protocol for curl. And don't forget the web UI.

CHAPTER 1

Installation

1.1 Prebuilt executable

Our releases are built and uploaded to github. You can download a binary matching your architecture and OS at:

https://github.com/boivie/lovebeat/releases

1.2 Using docker

We keep up-to-date docker images.

To get started, simply:

```
$ docker run -it -p 8127:8127/udp -p 8127:8127/tcp -p 8080:8080 boivie/lovebeat
```

You may want to run with other options to specify volumes for the data and configuration.

1.3 Building from source

You will need to have a go toolchain installed as well as npm which will be used for downloading all other dependencies for the frontend development.

After that, simply:

```
$ mkdir go
$ cd go
$ export GOPATH=`pwd`
$ go get github.com/boivie/lovebeat
$ cd src/github.com/boivie/lovebeat
$ make
$ ./lovebeat
```

CHAPTER 2

Getting started

2.1 Concepts

It's really helpful if you understand the different concepts in lovebeat.

Services send "heartbeats" with regular intervals to lovebeat. If they for some reason stop sending these heartbeats, lovebeat will react to this and update the service state, which in may trigger and alarm, which in turn will trigger alerts, such as sending e-mails.

And you can monitor it all in the Lovebeat web UI.

So let's break it down a bit:

2.2 Services

This is the name of the "thing" you want to monitor. You can choose anything as the name, and it typically looks like "myapp.mailers.invoice" with periods as the delimiter.

As you grow and have a lot of services to monitor, it's good to have some sort of hierarchy. It's up to you to choose one.

2.3 States

A service can be in different states. **OK** is the state you want to keep your services in. You can set a timeout so that the service will change state into **ERROR** if the service hasn't issued a beat within that period of time.

A service can also be muted. This will move it into the **MUTED** state, and then it will not trigger any alerts or cause alarms to be in **ERROR**.

2.4 Alarms

An alarm contains a filtered subset of your services. You specify a matching pattern and all services whose identifiers match this pattern will be part of the alarm.

This is an example of an alarm called “backup-jobs” that match all services starting with “backup.”

```
[[alarms]]
name = "backup-jobs"
pattern = "backup.*"
```

Alarms also have states. If all services within the alarm are **OK**, the alarm will be **OK**. But if any service is in **ERROR** state, the alarm will transition into the **ERROR** state.

Alarms can be automatically created based on the service names, which is a powerful feature when your service names have a structure.

Say that you have an application running on three servers (alpha, beta and delta), and the application provides two heartbeats, ”.healthcheck” and ”.background-job-1”. The complete list of services will thus be:

- application-name.alpha.healthcheck
- application-name.alpha.background-job-1
- application-name.beta.healthcheck
- application-name.beta.background-job-1
- application-name.delta.healthcheck
- application-name.delta.background-job-1

By having an alarm configuration such as:

```
[[alarms]]
name = "server-$name"
pattern = "application-name.$name.*"
```

You will then end up with three alarms, “server-alpha” including the services “application-name.alpha.healthcheck” and “application-name.alpha.background-job-1” and similar for “server-beta” and “server-delta”.

For more advanced pattern matching, use `includes` and `excludes` to specify a list of patterns. If any pattern in `includes` match, and no pattern in `excludes` match, the service will be part of the alarm. Example as below:

```
[[alarms]]
name = "source-$name"
includes = ["source.$name.*", "old-source.$name.*"]
excludes = ["source.deprecated.*"]
```

2.5 Alerts

When an alarm changes state (to *ERROR* or *OK*), it can trigger alerts that e.g. sends and e-mail (through SMTP or Mailgun), posts a Slack message to your team’s channel, sends an outgoing webhook or runs a shell script.

2.6 Web UI

Just point your browser to http://localhost:8080/ to see the current status of all your services and alarms.

CHAPTER 3

Alerters

You can setup lovebeat to send mails or issue outgoing webhooks (HTTP POST) to your web service whenever an alarm changes state. This is done on an alarm by modifying the configuration file.

3.1 Send Mail

The first step is to specify the SMTP server address and the e-mail address that will be used when sending the e-mails. It doesn't currently support SMTP authentication, so you might want to run a local SMTP server to proxy the sent e-mails.

If you have an account at Mailgun, you can specify your domain and API key to let Lovebeat send mails using Mailgun's API. By doing this, the SMTP settings will not be used.

The configuration file should look as following for SMTP:

```
[mail]
server = "localhost:25"
from = "lovebeat@example.com"
```

If you're using Mailgun for sending your mails, your configuration will look as follows:

```
[mailgun]
domain = "example.com"
from = "lovebeat@example.com"
api_key = "key-5ap419x2asxge9a6xaqq0ztagv-a4axj"
```

Example of specifying an alarm that sends mails:

```
[[alarms]]
name = "example"
pattern = "test.*"
alerts = ["mail-alert"]
```

```
[alerts.mail-alert]
mail = "administrator@example.com"
```

3.2 Outgoing Webhooks

When an alarm changes state, a POST will be sent to the URL(s) specified in the configuration. The JSON data that is sent follows:

```
POST /your/url/endpoint HTTP/1.1
Content-Type: application/json
Accept: application/json
User-Agent: Lovebeat
X-Lovebeat: 1

{
  "name": "alarm.name.here",
  "from_state": "ok",
  "to_state": "error",
  "incident_number": 4
}
```

Example of the configuration file:

```
[[alarms]]
name = "example"
pattern = "test.*"
alerts = ["to-requestbin"]

[alerts.to-requestbin]
webhook = "http://requestb.in/191w85o1"
```

3.3 Slack

Lovebeat can post messages to a slack channel whenever an alarm changes state. First of all, setup an incoming webhook to get a Webhook URL that you will enter in the lovebeat configuration file.

A working example would look like:

```
[[alarms]]
name = "example"
pattern = "test.*"
alerts = ["message-to-ops"]

[alerts.message-to-ops]
slack_channel = "#ops"

[slack]
webhook_url = "https://hooks.slack.com/services/T12345678/B12345678/abrakadabra"
```

3.4 Script

Lovebeat can run arbitrary scripts (or other executable files) whenever an alarm changes state. The details of the alert will be posted as environment variables:

- LOVEBEAT_ALARM=<name of the alarm>
- LOVEBEAT_STATE=<the current state>
- LOVEBEAT_PREVIOUS_STATE=<the previous state>
- LOVEBEAT_INCIDENT=<incident number>

The script will also inherit any environment variables that Lovebeat was started with.

The script's stdout and stderr will be printed, and the script will be invoked with no arguments. If a script doesn't finish within 10 seconds, it will be terminated. Remember to make your script executable using `chmod a+x script.sh`.

Example of the configuration file:

```
[[alarms]]
name = "example"
pattern = "test.*"
alerts = ["test-alert"]

[alerts.test-alert]
script = "/path/to/script.sh"
```

The script (/path/to/script.sh) could look like:

```
#!/bin/bash

echo "Hello World"
env
```

The output would then be (among other environment variables):

```
2016/01/26 18:10:56 INFO ALARM 'example', 11: state ok -> error
2016/01/26 18:10:56 INFO Running alert script /path/to/script.sh
Hello World
LOVEBEAT_ALARM=example
LOVEBEAT_STATE=ERROR
LOVEBEAT_PREVIOUS_STATE=OK
LOVEBEAT_INCIDENT=11
```

CHAPTER 4

Advanced Topics

4.1 Automatic Setting of Timeouts

While you can set the timeout manually, they can also be automatically calculated based on the frequency and regularity of the heartbeats.

A regular heartbeat results in a low threshold (compared to the median frequency of the heartbeats) and an irregular heartbeat sets the threshold higher so that it doesn't expire during normal operations.

The algorithm is rather well performing in theory and modeled (and tested) using the bundled Jupyter Notebook.

4.2 Monitoring

Lovebeat is designed to be resistant to environmental disturbances but it can still fail if e.g. the machine it's running on is degraded or if the network is experiencing problems. It's a very good idea to monitor Lovebeat so that you are confident that it's monitoring your services correctly.

4.2.1 External Monitoring

It is easy to have an external monitoring system find out if lovebeat and the services reporting to it are healthy. There are two API endpoint, `/api/status` and `/api/alarms/<alarm_name>/status` for that purpose.

Calling it will result in the following response:

```
$ curl http://localhost:8080/api/status
num_ok 4
num_error 2
has_error true
good false
```

If you call it with the `Accept` HTTP header set to `application/json`, the following will be the response instead:

```
$ curl -H "Accept: application/json" http://localhost:8080/api/status
{
  "num_ok": 4,
  "num_error": 2,
  "has_error": true,
  "good": false
}
```

`good` will be **true** only if there are no services in **ERROR** state.

You can let e.g. nagios monitor it. There is a provided nagios plugin in the contrib/ directory.

4.2.2 Lovebeat Monitoring

For more detailed monitoring, you can have two (or more) instances of Lovebeat monitor each other. By having one or several `notify` sections in the configuration file, you can specify a URL to which Lovebeat will post its heartbeats.

```
[[notify]]
lovebeat = "http://some-other-host:8080"
```

4.3 Logging

Lovebeat prints its logs to stderr. If you want the logs to be sent to the local syslog service, add the command line switch `-syslog`.

You can also increase the verbosity of the logs by adding `-debug`.

4.4 Metrics reporting

Lovebeat can send metrics to a statsd proxy using the UDP protocol, to allow them to be shown in e.g. graphite, influxdb or similar.

You will get some health information about Lovebeat itself, such as the time it takes to save its database, and also status information (as gauges) of all services and alarms. This allows you to correlate service status with other metrics you collect.

Simply specify a server and the prefix that Lovebeat will use for all metrics in the lovebeat configuration file:

```
[metrics]
server = "localhost:8125"
prefix = "lovebeat"
```

4.5 Behind a reverse proxy

Lovebeat can be located behind a reverse proxy and properly handle that it's served from a different path than the root path. Please keep in mind that the websocket functionality requires a proxy server with proper support for them.

In nginx, this would be a working configuration:

```
location /monitoring/lovebeat/ {
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_pass http://localhost:8080/;
}
```

4.6 Database on S3

When designing Lovebeat, a key decision was to build a solution with as few dependencies to other systems as possible since those systems can fail as well. Having the database on a separate SQL server is then something we have opted out from, but instead having a local file based database.

That works well as long as the host Lovebeat is running on is healthy and the disk where the database is located on isn't corrupted or disappears. When deploying Lovebeat on a transient host, such as on an auto-scaling instance on Amazon Web Services, this will cause problems as the disk isn't persistent if the service is restarted.

To support this common use-case, Lovebeat supports downloading and uploading its database to an Amazon S3 bucket. On startup, Lovebeat will download the file from the S3 storage, and every time the database is saved (defaults to once per minute and when Lovebeat exits), the database will be uploaded to the same S3 bucket.

To enable this, configure the database as follows:

```
[database]
filename = "lovebeat.db"
interval = 60
remote_s3_url = "s3://bucket-name/path/to/lovebeat.db"
remote_s3_region = "eu-west-1"
```

CHAPTER 5

API

5.1 statsd API

Using UDP or TCP.

We use the graphite protocol to trigger heartbeats and to set timeouts. UDP is great for generating heartbeats from within your application since the performance cost is very small and your application will not be affected if the lovebeat server isn't running for any reason.

- To trigger a heartbeat, send a counter value >= 0 to "<service>.beat"
- To set a timeout (in seconds), set the gauge value of "<service>.timeout"
- To clear a value, set the timeout to `-1`.
- To set the timeout to be automatically calculated, set the timeout to `-2`.
- A shortcut for setting the timeout to `-2` and issuing a beat (since this is a fairly common pattern), send a counter to `<service>.autobeat`.

Examples:

```
# UDP
$ echo "invoice.mailer.beat:1|c" | nc -4u -w0 localhost 8127

# TCP
$ echo "invoice.mailer.timeout:3600|g" | nc -c localhost 8127

# TCP, setting timeout to 'auto'
$ echo -e "invoice.mailer.timeout:-2|g\ninvoice.mailer.beat:1|c" | nc -c localhost␣
↪8127

# UDP, same as above
$ echo "invoice.mailer.autobeat:1|c" | nc -4u -w0 localhost 8127
```

You can even put a statsd proxy in front of lovebeat if you don't want to send UDP packets outside your localhost.

5.2 HTTP API

The HTTP API is the easy way to send heartbeats from e.g. curl.

This API is also used by the web UI and is fairly complete.

5.2.1 POST /api/services/<service_id>

Generates a heartbeat.

- To set a timeout, add the form field "timeout" and specify the time in seconds or "auto" to calculate one.
- You can also set the timeout value using a query parameter, e.g. `?timeout=3600`.
- Last, but not least, you can also post a JSON payload to this endpoint and let the JSON object's timeout field be set to the timeout value. Note that the `Content-Type` must be set to `application/json`.

This endpoint returns an empty JSON object as response.

Examples:

```
# Only trigger a beat - don't set any value
$ curl -X POST http://localhost:8080/api/services/invoice.mailer

# Set the timeout using a form field value
$ curl -d timeout=3600 http://localhost:8080/api/services/invoice.mailer

# Set the timeout using a query parameter
$ curl -X POST http://localhost:8080/api/services/invoice.mailer?timeout=3600

# Setting the timeout as a JSON object.
$ curl -H "Content-Type: application/json" -d '{"timeout":3600}' http://
↪localhost:8080/api/services/invoice.mailer
```

5.2.2 GET /api/services

Returns the list of services.

5.2.3 GET /api/services/<service_name>

Returns information about a specific service.

- By setting the `?details=1` query parameter, additional information may be returned.

5.2.4 POST /api/services/<service_id>/mute

Mutes the service and puts it into **MUTED** state.

5.2.5 POST /api/services/<service_id>/unmute

Unmutes the service and puts it into **OK** or **ERROR** depending on when it received its last heartbeat.

5.2.6 DELETE /api/services/<service_name>

Deletes a service.

5.2.7 GET /api/alarms

Returns a list of alarms.

5.2.8 GET /api/alarms/<alarm_name>

Returns details of a specific alarm and the services included in it.

5.2.9 DELETE /api/alarms/<alarm_name>

Removes the alarm. The alarm must be empty and it will appear again when a service is created that match this alarm config's patterns.

CHAPTER 6

Configuration

You don't need to write a configuration file to get started (just launch the executable), but some settings need to be specified if you want to configure alarms and use advanced features such as SMTP mail notifications.

Note that lovebeat by default reads `/etc/lovebeat.cfg` but you can override this by specifying the `-config <file>` argument when starting lovebeat. If no configuration file is specified, sensible defaults are used.

Please see the provided `lovebeat.cfg` file where all the settings are documented as well.

```
## Every section and key is documented, and the default values are
## provided here (commented out).

##
## General settings
##
## By specifying a 'public_url', which should be the full URL to
## reach lovebeat, we can insert full links in mail and slack alerts,
## for example.
#
#public_url = "http://lovebeat.example.com/"

##
## The database stores information about all services and alarms. It's
## in one single file and it's safely rewritten on save, which it does
## when exiting the program as well as every minute while running. This
## can be changed by the 'interval' setting.
##
## You can specify an Amazon S3 URL as 'remote_s3_url' from where it
## should download the database on start, and upload when it's saved.
#
#[database]
#filename = "lovebeat.db"
#interval = 60
#remote_s3_url = ""
#remote_s3_region = ""
```

```
##
## UDP listener, in statsd format
#
#[udp]
#listen = ":8127"

##
## TCP listener, in statsd format
#
#[tcp]
#listen = ":8127"

##
## TCP listener, for the dashboard and the HTTP API
#
#[http]
#listen = ":8080"

##
## SMTP settings, for the mail alerter.
#
#[mail]
#server = "localhost:25"
#from = "lovebeat@example.com"

##
## Mailgun settings, which takes priority over the SMTP settings
## if specified
##
## The API Key can be found in Mailgun's Account Settings.
#
#[mailgun]
#domain = ""
#from = ""
#api_key = ""

##
## Metrics reporting to a statsd proxy, using the UDP protocol.
## Note that this one is by default disabled, but can be enabled
## by specifying a server address and port, e.g. "localhost:8125"
#
#[metrics]
#server = ""
#prefix = "lovebeat"

##
## Configuration of the logfile where events are logged. An empty
## or unset path disables the logging.
#
#[eventlog]
#path = "/var/log/lovebeat/events.json"
#mode = 644
```

CHAPTER 7

License