## Contents

1 **Users Guide**  
1.1 Getting Started .......................................................... 3  
1.2 Configuring Sentry with Nginx ........................................... 10  
1.3 Upgrading ................................................................. 12  
1.4 Configuration .............................................................. 12  
1.5 Queuing Work ............................................................ 14  
1.6 Update Buffers ............................................................ 15  
1.7 Node Storage ............................................................... 16  
1.8 Throttles and Rate Limiting .............................................. 18  
1.9 Time-series Storage ...................................................... 19  
1.10 Inbound Mail .............................................................. 20  
1.11 Beacon ...................................................................... 22  
1.12 Performance Tuning ...................................................... 22  
1.13 Command Line Usage ................................................... 25  
1.14 Available Clients ........................................................ 26  
1.15 Plugins .................................................................... 27  
1.16 Frequently Asked Questions .......................................... 28  

2 **Developers**  
2.1 Contributing ............................................................... 31  
2.2 Writing a Client ........................................................... 34  
2.3 Writing a Plugin .......................................................... 43  
2.4 Interfaces ................................................................. 53  

3 **Internals**  
3.1 System Options .......................................................... 57  

4 **Reference**  
4.1 Changelog ................................................................. 59  
4.2 License ................................................................... 87  

5 **Resources** ............................................................... 89  

6 **Screenshots** ............................................................. 91  
6.1 Aggregated Events ....................................................... 92  
6.2 Event Details ............................................................ 92  

7 **Deprecation Notes** ..................................................... 93
Sentry is a modern error logging and aggregation platform.

It’s important to note that Sentry should not be thought of as a log stream, but as an aggregator. It fits somewhere in-between a simple metrics solution (such as Graphite) and a full-on log stream aggregator (like Logstash).
1.1 Getting Started

This guide will step you through setting up a Python-based virtualenv, installing the required packages, and configuring the basic web service.

Disclaimer: While Sentry is seemingly easy to get up and running, it’s a fairly complex system. This documentation does not intend to teach you how to maintain and configure many of the services, and the task should not be taken lightly.

1.1.1 Dependencies

Some basic prerequisites which you’ll need in order to run Sentry:

- A UNIX-based operating system. We test on Ubuntu.
- Python 2.7
- python-setuptools, python-pip, python-dev, libxml2-dev, libz-dev, libffi-dev, libssl-dev
- A real database (PostgreSQL)
- Redis (2.6.12 or newer)
- Nginx (with RealIP, i.e. nginx-full)
- A dedicated domain to host Sentry on (i.e. sentry.yourcompany.com).

1.1.2 Hardware

Sentry provides a number of mechanisms to scale its capacity out horizontally, however there is still a primary SPOF at the database level. In an HA setup, the database is only utilized for event indexing and basic data storage, and becomes much less of a capacity concern (see also Node Storage).

We don’t have any real numbers to tell you what kind of hardware you’re going to need, but we’ll help you make your decision based on existing usage from real customers.

If you’re looking for an HA, and high throughput setup, you’re going to need to setup a fairly complex cluster of machines, and utilize all of Sentry’s advanced configuration options. This means you’ll need Postgres, Riak, Redis, Memcached, and RabbitMQ. It’s very rare you’d need this complex of a cluster, and the primary usecase for this is getsentry.com.
For more typical, but still fairly high throughput setups, you can run off of a single machine as long as it has reasonable IO (ideally SSDs), and a good amount of memory.

The main things you need to consider are:

- TTL on events (how long do you need to keep historical data around)
- Average event throughput
- How many events get grouped together (which means they get sampled)

At a point, getsentry.com was processing approximately 4 million events a day. A majority of this data is stored for 90 days, which accounted for around 1.5TB of SSDs. Web and worker nodes were commodity (8GB-12GB RAM, cheap SATA drives, 8 cores), the only two additional nodes were a dedicated RabbitMQ and Postgres instance (both on SSDs, 12GB-24GB of memory). In theory, given a single high-memory machine, with 16+ cores, and SSDs, you could handle the entirety of the given data set.

### 1.1.3 Setting up an Environment

The first thing you’ll need is the Python `virtualenv` package. You probably already have this, but if not, you can install it with:

```
$ pip install -U virtualenv
```

Once that’s done, choose a location for the environment, and create it with the `virtualenv` command. For our guide, we’re going to choose `/www/sentry/`:

```
$ virtualenv /www/sentry/
```

Finally, activate your virtualenv:

```
$ source /www/sentry/bin/activate
```

**Note:** Activating the environment adjusts your PATH, so that things like pip now install into the virtualenv by default.

### 1.1.4 Install Sentry

Once you’ve got the environment setup, you can install Sentry and all its dependencies with the same command you used to grab virtualenv:

```
$ pip install -U sentry
```

Don’t be worried by the amount of dependencies Sentry has. We have a philosophy of using the right tools for the job, and not reinventing them if they already exist.

Once everything’s installed, you should be able to execute the Sentry CLI, via `sentry`, and get something like the following:

```
$ sentry
usage: 
```

**Using MySQL or Postgres**

We **highly** recommend using PostgreSQL for your database. The default database is sqlite and is only intended for Sentry’s development environment. MySQL will work with various caveats (you should use InnoDB, for one), but is not officially supported by the Sentry team.
These databases require additional packages, but Sentry provides a couple of meta packages to make things easier:

```
# install sentry and its postgresql dependencies
pip install -U sentry[postgres]

# or if you choose, mysql
pip install -U sentry[mysql]
```

**Installing from Source**

If you’re installing the Sentry source (e.g. from git), you’ll also need to install **npm**.

Once your system is prepared, symlink your source into the virtualenv:

```
$ python setup.py develop
```

**Note:** This command will install npm dependencies as well as compile static assets.

### 1.1.5 Initializing the Configuration

Now you’ll need to create the default configuration. To do this, you’ll use the `init` command. You can specify an alternative configuration path as the argument to `init`, otherwise it will use the default of `~/.sentry/sentry.conf.py`.

```
# the path is optional
sentry init /etc/sentry.conf.py
```

The configuration for the server is based on `sentry.conf.server`, which contains a basic Django project configuration, as well as the default Sentry configuration values. It defaults to SQLite, however **SQLite is not a fully supported database and should not be used in production**.

```
# for more information on DATABASES, see the Django configuration at:
# https://docs.djangoproject.com/en/1.6/ref/databases/
DATABASES = {
    'default': {
        # We suggest PostgreSQL for optimal performance
        'ENGINE': 'sentry.db.postgres',

        # Alternatively you can use MySQL
        'ENGINE': 'django.db.backends.mysql',

        'NAME': 'sentry',
        'USER': 'postgres',
        'PASSWORD': '',
        'HOST': '',
        'PORT': '',
    }
}

# No trailing slash!
SENTRY_URL_PREFIX = 'http://sentry.example.com'
SENTRY_WEB_HOST = '0.0.0.0'
SENTRY_WEB_PORT = 9000
```
1.1.6 Configure Redis

Redis is used as the default implementation for various backend services, including the time-series storage, SQL update buffers, and rate limiting.

We recommend running two separate Redis clusters: one for persistent data (TSDB) and one for temporal data (buffers, rate limits). This is because you can configure the nodes in very different ones to enable more aggressive/optimized LRU.

That said, if you’re running a small install you can probably get away with just setting up the defaults:

```python
SENTRY_REDIS_OPTIONS = {
    'hosts': {
        0: {
            'host': '127.0.0.1',
            'port': 6379,
            'timeout': 3,
            #'password': 'redis auth password'
        }
    }
}
```

All built-in Redis implementations (other than the queue) will use these default settings, but each individual service also will allow you to override it’s cluster settings.

See the individual documentation for the queue, update buffers, quotas, and time-series storage for more details.

1.1.7 Configure Outbound Mail

Several settings exist as part of the Django framework which will configure your outbound mail server. For the standard implementation, using a simple SMTP server, you can simply configure the following:

```plaintext
EMAIL_HOST = 'localhost'
EMAIL_HOST_PASSWORD = ''
EMAIL_HOST_USER = ''
EMAIL_PORT = 25
EMAIL_USE_TLS = False
```

Being that Django is a pluggable framework, you also have the ability to specify different mail backends. See the official Django documentation for more information on alternative backends.

1.1.8 Running Migrations

Sentry provides an easy way to run migrations on the database on version upgrades. Before running it for the first time you’ll need to make sure you’ve created the database:

```bash
# If you're using Postgres, and kept the database `$NAME` as `$sentry`
$ createdb -E utf-8 sentry

# alternatively if you're using MySQL, ensure you've created the database:
$ mysql -e 'create database sentry'
```
Once done, you can create the initial schema using the `upgrade` command:

```bash
$ SENTRY_CONF=/etc/sentry.conf.py sentry upgrade
```

Next up you’ll need to create the first user, which will act as a superuser:

```bash
# create a new user
$ SENTRY_CONF=/etc/sentry.conf.py sentry createuser
```

All schema changes and database upgrades are handled via the `upgrade` command, and this is the first thing you’ll want to run when upgrading to future versions of Sentry.

**Note:** Internally, this uses South to manage database migrations.

### 1.1.9 Starting the Web Service

Sentry provides a built-in webserver (powered by gunicorn and eventlet) to get you off the ground quickly, also you can setup Sentry as WSGI application, in that case skip to section Running Sentry as WSGI application.

To start the webserver, you simply use `sentry start`. If you opted to use an alternative configuration path you can pass that via the `SENTRY_CONF` parameter.

```bash
# Sentry's server runs on port 9000 by default. Make sure your client reflects
# the correct host and port!
$ SENTRY_CONF=/etc/sentry.conf.py sentry start
```

You should now be able to test the web service by visiting `http://localhost:9000/`.

**Note:** This doesn’t run any workers in the background, so assuming queueing is enabled (default in 7.0.0+) no asynchronous tasks will be running.

### 1.1.10 Starting the Workers

A large amount of Sentry’s work is typically done via it’s workers. While Sentry will seemingly work without using a queue you will not actually see anything show up in Sentry. Once you’ve configured the queue, you’ll also need to run workers. Generally, this is as simple as running “celery” from the Sentry CLI.

So do not forget to run the workers!

```bash
$ SENTRY_CONF=/etc/sentry.conf.py sentry celery worker -B
```

Technically there is a way to run sentry without the queues by setting `CELERY_ALWAYS_EAGER` to `True` but this is heavily discouraged and not supported.

**Note:** Celery is an open source task framework for Python.

### 1.1.11 Setup a Reverse Proxy

By default, Sentry runs on port 9000. Even if you change this, under normal conditions you won’t be able to bind to port 80. To get around this (and to avoid running Sentry as a privileged user, which you shouldn’t), we recommend you setup a simple web proxy.
Proxying with Apache

Apache requires the use of mod_proxy for forwarding requests:

```
ProxyPass / http://localhost:9000/
ProxyPassReverse / http://localhost:9000/
ProxyPreserveHost On
RequestHeader set X-Forwarded-Proto "https" env=HTTPS
```

You will need to enable `headers`, `proxy`, and `proxy_http` apache modules to use these settings.

Proxying with Nginx

You’ll use the builtin HttpProxyModule within Nginx to handle proxying:

```
location / {
    proxy_pass http://localhost:9000;
    proxy_redirect off;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
}
```

See Configuring Sentry with Nginx for more details on using Nginx.

Enabling SSL

If you are planning to use SSL, you will also need to ensure that you’ve enabled detection within the reverse proxy (see the instructions above), as well as within the Sentry configuration:

```
SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')
```

1.1.12 Running Sentry as a Service

We recommend using whatever software you are most familiar with for managing Sentry processes. For us, that software of choice is Supervisor.

Configure `supervisord`

Configuring Supervisor couldn’t be more simple. Just point it to the `sentry` executable in your virtualenv’s `bin/` folder and you’re good to go.

```
[program:sentry-web]
directory=/www/sentry/
command=/www/sentry/bin/sentry start
autostart=true
autorestart=true
redirect_stderr=true
stdout_logfile=syslog
stderr_logfile=syslog

[program:sentry-worker]
```
directory=/www/sentry/
command=/www/sentry/bin/sentry celery worker -B
autostart=true
autorestart=true
redirect_stderr=true
stdout_logfile=syslog
stderr_logfile=syslog

1.1.13 Removing Old Data

One of the most important things you’re going to need to be aware of is storage costs. You’ll want to setup a cron job that runs to automatically trim stale data. This won’t guarantee space is reclaimed (i.e. by SQL), but it will try to minimize the footprint. This task is designed to run under various environments so it doesn’t delete things in the most optimal way possible, but as long as you run it routinely (i.e. daily) you should be fine.

```
$ crontab -e
0 3 * * * sentry cleanup --days=30
```

1.1.14 Additional Utilities

If you’re familiar with Python you’ll quickly find yourself at home, and even more so if you’ve used Django. The `sentry` command is just a simple wrapper around Django’s `django-admin.py`, which means you get all of the power and flexibility that goes with it.

Some of those which you’ll likely find useful are:

**createuser**

Quick and easy creation of superusers. These users have full access to the entirety of the Sentry server.

**runserver**

Testing Sentry locally? Spin up Django’s built-in `runserver` (or `pip install django-devserver` for something slightly better).

1.1.15 What’s Next?

There are several applications you may want to add to the default Sentry install for various security or other uses. This is a bit outside of the scope of normal (locked down) installs, as typically you’ll host things on your internal network. That said, you’ll first need to understand how you can modify the default settings.

First pop open your `sentry.conf.py`, and add the following to the **very top** of the file:

```
from sentry.conf.server import *
```

Now you’ll have access to all of the default settings (Django and Sentry) to modify at your own will.

We recommend going over all of the defaults in the generated settings file, and familiarizing yourself with how the system is setup.
1.2 Configuring Sentry with Nginx

Nginx provides a very powerful platform for running in front of Sentry as it gives us features like rate limiting.

If you’re on Ubuntu, you can simply install the `nginx-full` package which will include the required RealIP module. Otherwise you’ll need to compile Nginx from source with `--with-http_realip_module`.

Below is a sample production ready configuration for Nginx with Sentry:

```nginx
http {
    # set REMOTE_ADDR from any internal proxies
    # see http://nginx.org/en/docs/http/ngx_http_realip_module.html
    set_real_ip_from 127.0.0.1;
    set_real_ip_from 10.0.0.0/8;
    real_ip_header X-Forwarded-For;
    real_ip_recursive on;

    # SSL configuration -- change these certs to match yours
    ssl_certificate /etc/ssl/sentry.example.com.crt;
    ssl_certificate_key /etc/ssl/sentry.example.com.key;

    # NOTE: These settings may not be the most-current recommended
    # defaults
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_session_cache shared:SSL:128m;
    ssl_session_timeout 10m;

    server {
        listen 80;
        server_name sentry.example.com;

        location / {
            if ($request_method = GET) {
                rewrite ^ https://$host$request_uri? permanent;
            }
            return 405;
        }
    }

    server {
        listen 443 ssl;
        server_name sentry.example.com;

        proxy_set_header Host $http_host;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-For $remote_addr;
        proxy_redirect off;

        # keepalive + raven.js is a disaster
        keepalive_timeout 0;

        # use very aggressive timeouts
        proxy_read_timeout 5s;
        proxy_send_timeout 5s;
        send_timeout 5s;
        resolver_timeout 5s;
        client_body_timeout 5s;
    }
}
```
# buffer larger messages
client_max_body_size 5m;
client_body_buffer_size 100k;

location / {
    proxy_pass http://localhost:9000;
    add_header Strict-Transport-Security "max-age=31536000";
}

## 1.2.1 Proxying uWSGI

While Sentry provides a default webserver, you’ll likely want to move to something a bit more powerful. We suggest using uWSGI to run Sentry.

Install uWSGI into your virtualenv (refer to quickstart if you're confused):

```
pip install uwsgi
```

Create a uWSGI configuration which references the Sentry configuration:

```
[uwsgi]
env = SENTRY_CONF=/etc/sentry.conf.py
module = sentry.wsgi

; spawn the master and 4 processes with 8 threads each
http = 127.0.0.1:9000
master = true
processes = 4
threads = 8

; allow longer headers for raven.js if applicable
; default: 4096
buffer-size = 32768

; allow large file uploads
limit-post = 5242880

; various other explicit defaults
post-buffering = 65536
thunder-lock = true
disable-logging = true
enable-threads = true
single-interpreter = true
lazy-apps = true
log-x-forwarded-for = true
```

Finally, re-configure supervisor to run uwsgi instead of ‘sentry start’:

```
/www/sentry/bin/uwsgi --ini /www/sentry/uwsgi.ini
```
1.3 Upgrading

Always upgrade the Sentry server before upgrading your clients unless the client states otherwise.

Upgrading Sentry simply requires you to run migrations and restart your web services. We recommend you run the migrations from a separate install so that they can be completed before updating the code which runs the webserver.

Generally, you’ll start by installing the upgraded Sentry package:

```
easy_install -U sentry
```

Continue by running all required migrations, with the upgrade command:

```
sentry upgrade
```

Finally, restart any Sentry services you had running.

1.3.1 Upgrading to 7.x

An extremely large amount of changes happened between the 6.x and 7.x series. Many of them are backwards incompatible so you should review the setup guide again.

- Redis (at least version 2.4) is now a requirement
- The queue and buffer systems are no longer optional for production systems
- Time series data (graphs) have been moved to a new system (there is no data migration)
- The default sentry.conf.py has greatly changed

Due to the configuration generation being greatly improved, we recommend merging your existing settings with the new defaults. To do that just backup your `sentry.conf.py` and generate a new one using `sentry init`.

See the Changelog for additional backwards incompatible APIs.

1.4 Configuration

This document describes additional configuration options available to the Sentry server. If you are looking for documentation for the client, it is maintained in the Raven project.

**SENTRY_URL_PREFIX**

Absolute URL to the sentry root directory. Should not include a trailing slash.

Defaults to "".

```
SENTRY_URL_PREFIX = 'http://sentry.example.com'
```

1.4.1 Authentication

**SENTRY_FEATURES['auth:register']**

Should Sentry allow users to create new accounts?

Defaults to True (can register).

```
SENTRY_FEATURES['auth:register'] = True
```
**SENTRY_PUBLIC**

Should Sentry make all data publicly accessible? This should **only** be used if you’re installing Sentry behind your company’s firewall.

Users will still need to have an account to view any data.

Defaults to `False`.

```python
SENTRY_PUBLIC = True
```

**SENTRY_ALLOW_PUBLIC_PROJECTS**

Should Sentry allow users without the ‘sentry.change_project’ permission to make projects globally public?

Defaults to `True` (can set public status).

```python
SENTRY_ALLOW_PUBLIC_PROJECTS = False
```

**SENTRY_ALLOW_ORIGIN**

If provided, Sentry will set the Access-Control-Allow-Origin header to this value on /api/store/ responses. In addition, the Access-Control-Allow-Headers header will be set to ‘X-Sentry-Auth’. This allows JavaScript clients to submit cross-domain error reports.

You can read more about these headers in the Mozilla developer docs.

Defaults to `None` (don’t add the Access-Control headers)

```python
SENTRY_ALLOW_ORIGIN = "http://foo.example"
```

### 1.4.2 Services

**Web Server**

The following settings are available for the built-in webserver:

**SENTRY_WEB_HOST**

The hostname which the webserver should bind to.

Defaults to `localhost`.

```python
SENTRY_WEB_HOST = '0.0.0.0'  # bind to all addresses
```

**SENTRY_WEB_PORT**

The port which the webserver should listen on.

Defaults to `9000`.

```python
SENTRY_WEB_PORT = 9000
```

**SENTRY_WEB_OPTIONS**

A dictionary of additional configuration options to pass to gunicorn.

Defaults to `{}`.

```python
SENTRY_WEB_OPTIONS = {
    'workers': 10,
    'worker_class': 'gevent',
}
```

Note: The logging options of gunicorn is overridden by the default logging configuration of Sentry. In order to reuse loggers from gunicorn, put `LOGGING['disable_existing_loggers'] = False` into your configuration file.
SMTP Server

The following settings are available for the built-in SMTP mail server:

**SENTRY_SMTP_HOST**

The hostname which the smtp server should bind to.

Defaults to localhost.

```
SENTRY_SMTP_HOST = '0.0.0.0'  # bind to all addresses
```

**SENTRY_SMTP_PORT**

The port which the smtp server should listen on.

Defaults to 1025.

```
SENTRY_SMTP_PORT = 1025
```

**SENTRY_SMTP_HOSTNAME**

The hostname which matches the server’s MX record.

Defaults to localhost.

```
SENTRY_SMTP_HOSTNAME = 'reply.getsentry.com'
```

1.4.3 Data Sampling

**SENTRY_SAMPLE_DATA**

New in version 1.10.0.

Controls sampling of data.

Defaults to True.

If this is enabled, data will be sampled in a manner similar to the following:

- 50 messages stores ~50 results
- 1000 messages stores ~400 results
- 10000 messages stores ~900 results
- 100000 messages stores ~1800 results
- 1000000 messages stores ~3600 results
- 10000000 messages stores ~4500 results

```
SENTRY_SAMPLE_DATA = False
```

1.5 Queuing Work

Sentry comes with a built-in queue to process tasks in a more asynchronous fashion. For example, with workers enabled, when an event comes in instead of writing it to the database immediately, it sends a job to the queue so that the request can be returned right away, and the background workers handle actually saving that data.

**Note:** As of version 3.3.0 the queue is now powered by Celery.
1.5.1 Running a Worker

Workers can be run by using the Sentry CLI. Specifically, you call out to celery, which is the worker manager process of the Celery library.

```
sentry celery worker -B
```

**Note:** You will need to run both celery workers and celerybeat. In our example, the -B flag runs a beat instance (in addition to the worker), but in production you may want to run them separately.

We again recommend running this as a service. Below is an example configuration with supervisor:

```
[program:sentry-worker]
directory=/www/sentry/
command=/www/sentry/bin/sentry celery worker -B -l WARNING
autostart=true
autorestart=true
redirect_stderr=true
killasgroup=true
```

1.5.2 Enable the Queue

Once you’ve brought up a worker, the next step is to enable the queue. This is done with a simple settings flag:

```
CELERY_ALWAYS_EAGER = False
```

It’s also highly recommended that you switch away from the default queue settings, which rely on the database, and move to something more efficient. These are documented in more details as part of the Celery documentation, but something simple like Redis will do just fine.

An example configuration using a local Redis server might look like this:

```
BROKER_URL = "redis://localhost:6379/0"
```

1.6 Update Buffers

Sentry provides the ability to buffer certain updates to events, such as counts and timestamps. This is extremely helpful if you have high concurrency, especially if they’re frequently the same event.

For example, if you happen to receive 100,000 events/second, and 10% of those are reporting a connection issue to the database (where they’d get grouped together), enabling a buffer backend will change things so that each count update is actually put into a queue, and all updates are performed at the rate of how fast the queue can keep up.

1.6.1 Choosing a Backend

To specify a backend, simply modify the SENTRY_BUFFER and SENTRY_BUFFER_OPTIONS values in your configuration:

```
SENTRY_BUFFER = 'sentry.buffer.base.Buffer'
SENTRY_BUFFER_OPTIONS = {}
```
1.6.2 The Redis Backend

Configuring the Redis backend requires the queue or you won’t see any gains (in fact you’ll just negatively impact your performance).

Configuration is fairly straight forward:

```python
SENTRY_BUFFER = 'sentry.buffer.redis.RedisBuffer'
SENTRY_BUFFER_OPTIONS = {
    'hosts': {
        0: {
            'host': 'localhost',
            'port': 6379
        }
    }
}
```

Because the Redis buffer relies on the Nydus package, this gives you the ability to specify multiple nodes and have keys automatically distributed. It’s unlikely that you’ll need this functionality, but if you do, a simple configuration might look like this:

```python
SENTRY_BUFFER_OPTIONS = {
    'hosts': {
        0: {
            'host': '192.168.1.1'
        },
        1: {
            'host': '192.168.1.2'
        }
    }
}
```

With the default configuration this will distribute keys using a simple partition router (relatively even distribution).

1.7 Node Storage

Sentry provides an abstraction called ‘nodestore’ which is used for storing key/value blobs.

The default backend simply stores them as gzipped blobs in in the ‘nodestore_node’ table of your default database.

1.7.1 Django Backend

The Django backend stores all data in the ‘nodestore_node’ table, using a the gzipped json blob-as-text pattern.

The backend provides no options, so it should simply be set to an empty dict.

```python
SENTRY_NODESTORE = 'sentry.nodestore.django.DjangoNodeStorage'
SENTRY_NODESTORE_OPTIONS = {}  
```

1.7.2 Riak Backend

Ria is the recommended backend for installations which have a large data consumption pattern, and would prefer to scale out, rather than scale up a single SQL node.

Some notes on your Riak installation:
- You will want to use the leveldb backend as blobs are larger, and compression helps greatly.
- Reads explicitly use \( r=1 \).
- We recommend \( n=2 \) for replicas, but if the data isn’t extremely important, \( n=1 \) is fine.

```python
SENTRY_NODESTORE = 'sentry.nodestore.riak.RiakNodeStorage'
SENTRY_NODESTORE_OPTIONS = {
    # specify each of your Riak nodes, or the address of your load balancer
    'nodes': [
        {'host': '127.0.0.1', 'http_port': 8098},
    ],
    # (optional) specify an alternative bucket name
    # 'bucket': 'nodes',
    # (optional) change the default resolver
    # 'resolver': riak.resolver.last_written_resolver
}
```

### 1.7.3 Cassandra Backend

Cassandra is a horizontally scalable datastore in many of the same ways as Riak.

The Sentry Cassandra backend only operates over the native CQL interface, so requires Cassandra 1.2+

```sql
CREATE KEYSPACE sentry WITH replication = {
    'class': 'SimpleStrategy',
    'replication_factor': '2'
};

USE sentry;

CREATE TABLE nodestore (
    key text PRIMARY KEY,
    flags int,
    value blob
) WITH
    compaction={'sstable_size_in_mb': '160', 'class': 'LeveledCompactionStrategy'} AND
    compression={'sstable_compression': 'SnappyCompressor'};

SENTRY_NODESTORE = 'sentry.nodestore.cassandra.CassandraNodeStorage'
SENTRY_NODESTORE_OPTIONS = {
    'servers': ['127.0.0.1:9042'],
    # (optional) specify an alternative keyspace
    # 'keyspace': 'sentry',
    # (optional) specify an alternative columnfamily
    # 'columnfamily': 'nodestore',
}
```

### 1.7. Node Storage
1.7.4 Custom Backends

If you have a favorite data storage solution, it only has to operate under a few rules for it to work w/ Sentry’s blob storage:

- set key to value
- get key
- delete key

For more information on implementing your own backend, take a look at `sentry.nodestore.base.NodeStorage`.

1.8 Throttles and Rate Limiting

With the way Sentry works you may find yourself in a situation where you’ll see too much inbound traffic without a good way to drop excess messages. There’s a few solutions to this, and you’ll likely want to employ them all if you are faced with this problem.

1.8.1 Enabling Quotas

Sentry provides a built-in quota mechanism to limit on per-project and system-wide basis.

As of version 7.0.0, quotas are enabled by default and configured to use the Redis backend. To adjust the settings or the backend, you can use the following settings:

```python
SENTRY_QUOTAS = 'sentry.quotas.redis.RedisQuota'
SENTRY_QUOTA_OPTIONS = {
    'hosts': {
        0: {
            'host': 'localhost',
            'port': 6379
        }
    }
}
```

You can additionally configure system-wide maximums, and a default value for all projects:

```python
SENTRY_DEFAULT_MAX_EVENTS_PER_MINUTE = '90%'
SENTRY_SYSTEM_MAX_EVENTS_PER_MINUTE = 500
```

If you have additional needs, you’re freely available to extend the base Quota class just as the Redis implementation does.

1.8.2 Rate Limiting with IPTables

One of your most effective options is to rate limit with your system’s firewall, in our case, IPTables. If you’re not sure how IPTables works, take a look at Ubuntu’s IPTables How-to.

A sample configuration, which will limit a single IP from bursting more than 5 messages in a 10 second period might look like this:
# create a new chain for rate limiting
-N LIMITED

# rate limit individual ips to prevent stupidity
-I INPUT -p tcp --dport 80 --m state --state NEW --m recent --set
-I INPUT -p tcp --dport 443 --m state --state NEW --m recent --set
-I INPUT -p tcp --dport 80 --m state --state NEW --m recent --update --seconds 10 --hitcount 5 -j LIMITED
-I INPUT -p tcp --dport 443 --m state --state NEW --m recent --update --seconds 10 --hitcount 5 -j LIMITED

# log rejected ips
-A LIMITED -p tcp -m limit --limit 5/min -j LOG --log-prefix "Rejected TCP: " --log-level 7
-A LIMITED -j REJECT

1.8.3 Rate Limiting with Nginx

While IPTables will help prevent DDOS they don’t effectively communicate to the client that it’s being rate limited. This can be important depending on how the client chooses to respond to the situation.

An alternative (or rather, an addition) is to use something like ngx_http_limit_conn_module.

An example configuration looks something like this:

```
limit_req_zone $binary_remote_addr zone=one:100m rate=3r/s;
limit_req_zone $projectid zone=two:100m rate=6r/s;
limit_req_status 429;
limit_req_log_level warn;

server {
  listen 80;

  location / {
    proxy_pass http://internal;
  }

  location ~* /api/(?P<projectid>\d+)/?store/ {
    proxy_pass http://internal;
    limit_req zone=one burst=3 nodelay;
    limit_req zone=two burst=10 nodelay;
  }
}
```

1.8.4 Using Cyclops (Client Proxy)

An additional option for rate limiting is to do it on the client side. Cyclops is a third-party proxy written in Python (using Tornado) which aims to solve this.

It’s not officially supported, however it is used in production by several large users.

1.9 Time-series Storage

Sentry provides a service to store time-series data. Primarily this is used to display aggregate information for events and projects, as well as calculating (in real-time) the rates of events.
### 1.9.1 Choosing a Backend

To specify a backend, simply modify the `SENTRY_TSDB` and `SENTRY_TSDB_OPTIONS` values in your configuration:

```python
SENTRY_TSDB = 'sentry.tsdb.dummy.DummyTSDB'
SENTRY_TSDB_OPTIONS = {}
```

### 1.9.2 The Redis Backend

Configuration is fairly straightforward:

```python
SENTRY_TSDB = 'sentry.tsdb.redis.RedisTSDB'
SENTRY_TSDB_OPTIONS = {
    'hosts': {
        0: {
            'host': 'localhost',
            'port': 6379
        }
    }
}
```

Because the Redis buffer relies on the Nydus package, this gives you the ability to specify multiple nodes and have keys automatically distributed. It’s unlikely that you’ll need this functionality, but if you do, a simple configuration might look like this:

```python
SENTRY_TSDB_OPTIONS = {
    'hosts': {
        0: {
            'host': '192.168.1.1'
        },
        1: {
            'host': '192.168.1.2'
        }
    }
}
```

With the default configuration this will distribute keys using a simple partition router (relatively even distribution).

### 1.10 Inbound Mail

Sentry provides support for handling incoming email in various situations. Currently it only supports processing replies to error and note notifications.

For configuration instructions, choose a backend:

#### 1.10.1 Inbound Email via Mailgun

New in version 7.2.0.

Start by choosing a domain to handle inbound email. We find it easiest if you maintain a separate domain from anything else. In our example, we’re going to choose `inbound.sentry.example.com`. You’ll need to configure your DNS records for the given domain according to the Mailgun documentation.

Create a new route in mailgun:
Configure Sentry with the appropriate settings:

```
# Your Mailgun API key (used to verify incoming webhooks)
MAILGUN_API_KEY = ''

# Set the SMTP hostname to your configured inbound domain
SENTRY_SMTP_HOSTNAME = 'inbound.sentry.example.com'

# Inform Sentry to send the appropriate mail headers to enable incoming replies
SENTRY_ENABLE_EMAIL_REPLIES = True
```

That's it! You'll now be able to respond to activity notifications on errors via your email client.

### 1.10.2 Inbound Email via Nginx

Start by choosing a domain to handle inbound email. We find it easiest if you maintain a separate domain from anything else. In our example, we're going to choose `inbound.sentry.example.com`. You'll need to configure your DNS records appropriately.

Add another supervisor config to run the Sentry `smtp` service:

```
[program:sentry-inbound-mail]
directory=/www/sentry/
command=/www/sentry/bin/sentry start smtp
autostart=true
autorestart=true
stdout_logfile syslog
stderr_logfile syslog
```

Configure an Nginx route as an SMTP mail proxy:

```
http {
    # Bind an http server to localhost only just for the smtp auth server
    server {
        listen 127.0.0.1:80;

        # Return back the address and port for the listening Sentry smtp server. Default is 127.0.0.1:1025.
        location = /smtp {
            add_header Auth-Server 127.0.0.1;
            add_header Auth-Port 1025;
            return 200;
        }
    }

    mail {
        auth_http localhost/smtp;
    }
}
```
server {
    listen 25;
    protocol smtp;
    proxy on;
    smtp_auth none;
    xclient off;
}

And finally, update Sentry with the appropriate settings:

```python
# Set the SMTP hostname to your configured inbound domain
SENDER_SMTP_HOSTNAME = 'inbound.sentry.example.com'

# Inform Sentry to send the appropriate mail headers to enable
# incoming replies
SENDER_ENABLE_EMAIL_REPLIES = True
```

That's it! You'll now be able to respond to activity notifications on errors via your email client.

### 1.11 Beacon

Sentry will periodically communicate with a remote beacon server. This is utilized for a couple of things, primarily:

- Getting information about the current version of Sentry
- Retrieving important system notices

The remote server is operated by the Sentry team (getsentry.com), and the information reported follows the company's privacy policy.

The following information is reported:

- A unique installation ID
- The version of Sentry
- A technical contact email (`SENDER_ADMIN_EMAIL`)
- General anonymous statistics on the data pattern (such as the number of users)

Note: The contact email is utilized for security announcements, and will never be used outside of such.

The data reported is minimal and it greatly helps the development team behind Sentry. With that said, you can disable the beacon with the following setting:

```python
SENDER_BEACON = False
```

### 1.12 Performance Tuning

This document describes a set of best practices which may help you squeeze more performance out of various Sentry configurations.
1.12.1 Redis

Ensure you’re using at least Redis 2.4

All Redis usage in Sentry is temporal, which means the append-log/fsync models in Redis do not need to apply.

With that in mind, we recommend the following changes to (some) default configurations:

- Disable saving by removing all `save XXXX` lines.
- Set `maxmemory-policy allkeys-lru` to aggressively prune all keys.
- Set `maxmemory 1gb` to a reasonable allowance.

1.12.2 Web Server

Switching off of the default Sentry worker model and to uWSGI + emperor mode can yield very good results.

If you’re using supervisord, you can easily implement emperor mode and uWSGI yourself by doing something along the lines of:

```plaintext
[program:web]
command=newrelic-admin run-program /srv/www/getsentry.com/env/bin/uwsgi -s 127.0.0.1:90%(process_num)02d
process_name=%(program_name)s_%(process_num)02d
numprocs=20
numprocs_start=0
startsecs=5
startretries=3
stopsignal=QUIT
stopwaitsecs=10
stopasgroup=true
killasgroup=true
environment=SENTRY_CONF="/srv/www/getsentry.com/current/getsentry/settings.py"
directory=/srv/www/getsentry.com/current/
stdout_logfile syslog
stderr_logfile syslog
```

Once you’re running multiple processes, you’ll of course need to also configure something like Nginx to load balance to them:

```plaintext
upstream internal {
    least_conn;
    server 127.0.0.1:9000;
    server 127.0.0.1:9001;
    server 127.0.0.1:9002;
    server 127.0.0.1:9003;
    server 127.0.0.1:9004;
    server 127.0.0.1:9005;
    server 127.0.0.1:9006;
    server 127.0.0.1:9007;
    server 127.0.0.1:9008;
    server 127.0.0.1:9009;
    server 127.0.0.1:9010;
    server 127.0.0.1:9011;
    server 127.0.0.1:9012;
    server 127.0.0.1:9013;
    server 127.0.0.1:9014;
    server 127.0.0.1:9015;
    server 127.0.0.1:9016;
    server 127.0.0.1:9017;
}
```
server 127.0.0.1:9018;
server 127.0.0.1:9019;
}

server {
    listen 80;

    server_name sentry.example.com;

    # keepalive + raven.js is a disaster
    keepalive_timeout 0;

    # use very aggressive timeouts
    proxy_read_timeout 5s;
    proxy_send_timeout 5s;
    send_timeout 5s;
    resolver_timeout 5s;
    client_body_timeout 5s;

    # buffer larger messages
    client_max_body_size 150k;
    client_body_buffer_size 150k;

    location / {
        uwsgi_pass internal;

        uwsgi_param Host $host;
        uwsgi_param X-Real-IP $remote_addr;
        uwsgi_param X-Forwarded-For $proxy_add_x_forwarded_for;
        uwsgi_param X-Forwarded-Proto $http_x_forwarded_proto;

        include uwsgi_params;
    }
}

See uWSGI’s official documentation for emperor mode details.

1.12.3 Celery

Celery can be difficult to tune. Your goal is to maximize the CPU usage without running out of memory. If you have JavaScript clients this becomes more difficult, as currently the sourcemap and context scraping can buffer large amounts of memory depending on your configurations and the size of your source files.

On a completely anecdotal note, you can take the same approach that you might take with improving the webserver: spawn more processes. We again look to supervisord for managing this for us:

```
[program:celeryd]
command=/srv/www/getsentry.com/env/bin/sentry celery worker -c 6 -P processes -l WARNING -n worker-%(process_name)02d
process_name=%(program_name)s_%(process_num)02d
numprocs=16
numprocs_start=0
startsecs=1
startretries=3
stopsignal=TERM
stopwaitsecs=10
stopasgroup=false
killasgroup=true
```
1.12.4 Monitoring Memory

There are cases where Sentry currently buffers large amounts of memory. This may depend on the client (javascript vs python) as well as the size of your events. If you repeatedly run into issues where workers or web nodes are using a lot of memory, you’ll want to ensure you have some mechanisms for monitoring and resolving this.

If you’re using supervisord, we recommend taking a look at superlance which aids in this situation:

```bash
[eventlistener:memmon]
command=memmon -a 400MB -m ops@example.com
events=TICK_60
```

1.13 Command Line Usage

Sentry installs a command line script under the name `sentry`. This will allow you to perform most required operations that are unachievable within the web UI.

If you’re using a non-standard configuration location you’ll need to use the SENTRY_CONF environment variable to specify the path:

```bash
SENTRY_CONF=/etc/sentry.conf.py sentry help
```

For a list of commands, you can also use `sentry help`, or `sentry [command] --help` for help on a specific command.

**Note:** The script is powered by a library called Logan and simply acts as a conduit to django-admin.py.

1.13.1 Built-in Commands

`init [config]`

Initializes the configuration file for Sentry.

Defaults to `~/.sentry/sentry.conf.py`

```bash
sentry init /etc/sentry.conf.py
```

**Note:** The init command requires you to pass the configuration value as the parameter whereas other commands require you to use SENTRY_CONF for passing the location of this file.

`start [service]`

Starts a Sentry service. By default this value is ‘http’.

```bash
sentry start
```

`upgrade`

Performs any needed database migrations. This is similar to running `django-admin.py syncdb --migrate`.

`cleanup`

Performs all trim operations based on your configuration.
repair
Performs any needed repair against the Sentry database. This will attempt to correct things like missing teams, project keys, etc.

If you specify --owner it will also update ownerless projects:

```
sentry repair --owner=<username>
```

### 1.14 Available Clients

The following clients are officially recognized as production-ready, generally supported by the Sentry team, and work with the current Sentry protocol:

- Python (raven-python)
- PHP (raven-php)
- Java (raven-java)
- Ruby (raven-ruby)
- Chef (chef-sentry-handler)
- JavaScript (raven-js)
- Node.js (raven-node)
- iOS / Objective-C (raven-objc)
- C# (raven-csharp)

Additionally, the following third party/experimental clients are available:

- Action Script 3 (raven-as3)
- Android (Sentry-Android)
- CFML (raven-cfml)
- Erlang (raven-erlang)
- Go (raven-go)
- Grails (raven-grails)
- PL/SQL (Oracle) (raven-plsql)
- R (logging)
- Server-Side ActionScript (raven-ssas)

### 1.14.1 Client Criteria

If you’re developing a client for your platform, there’s several things we highly encourage:

- It should fully implement the current version of the Sentry protocol.
- It should conform to the standard DSN configuration method.
- It should contain an acceptable level of documentation and tests.
- The client should be properly packaged, and named raven-<platform>. 
1.15 Plugins

There are several interfaces currently available to extend Sentry. These are a work in progress and the API is not frozen.

1.15.1 Bundled Plugins

Sentry includes several plugins by default. Built-in plugins are controlled via the `INSTALLED_APPS` Django setting:

```python
INSTALLED_APPS = [
    ...
    'sentry.plugins.sentry_mail',
    'sentry.plugins.sentry_urls',
    'sentry.plugins.sentry_useragents',
]
```

- `sentry.plugins.sentry_urls`
  Enables auto tagging of URLs based on the HTTP interface contents.

- `sentry.plugins.sentry_mail`
  Enables email notifications when new events or regressions happen.

- `sentry.plugins.sentry_useragents`
  Enables auto tagging of browsers and operating systems based on the 'User-Agent' header in the HTTP interface.

  New in version 4.5.0.

1.15.2 3rd Party Extensions

The following extensions are available and maintained by members of the Sentry community:

- `sentry-bitbucket`
- `sentry-campfire`
- `sentry-flowdock`
- `sentry-fogbugz`
- `sentry-freight`
- `sentry-github`
- `sentry-groveio`
- `sentry-hipchat`
- `sentry-heroku`
- `sentry-irc`
- `sentry-irccat`
- `sentry-jira`
- `sentry-kafka`
- `sentry-ldap-auth`
- `sentry-lighthouse`
- `sentry-notifico`
• sentry-opsgenie
• sentry-phabricator
• sentry-pivotal
• sentry-pushover
• sentry-searchbutton
• sentry-sprintly
• sentry-statsd
• sentry-top
• sentry-trello
• sentry-whatsapp
• sentry-xmpp
• sentry-youtrack
• sentry-slack
• sentry-zabbix
• sentry-taiga

Have an extension that should be listed here? Submit a pull request and we’ll get it added.

1.16 Frequently Asked Questions

1.16.1 Common Problems

My sentry is running at example.com:9000 but whenever I visit it I get redirected to example.com  You likely have not correctly configured SENTRY_URL_PREFIX. See Configuration for more information.

AJAX requests do not seem to work properly It’s likely you have not correctly configured SENTRY_URL_PREFIX, so you’re hitting CORS issues. See Configuration for more information.

The client reports success (200 OK) but I don’t see events Something is misconfigured. A 200 OK from the API means “I have validated and enqueued this event”, so the first thing you should check is your workers.

Counts on events aren’t increasing Counts are incremented in bulk asynchronously utilizing the buffer and queue subsystems. Check your configuration on those.

1.16.2 How do I

... script the Sentry installation to bootstrap things like projects and users? Sentry is a simple Django (Python) application that runs using a utility runner. A script that creates a project and default user might look something like this:

```python
# Bootstrap the Sentry environment
from sentry.utils.runner import configure
configure()

# Do something crazy
from sentry.models import Team, Project, ProjectKey, User, Organization
```
user = User()
user.username = 'admin'
user.email = 'admin@localhost'
user.is_superuser = True
user.set_password('admin')
user.save()

organization = Organization()
organization.name = 'MyOrg'
organization.owner = user
organization.save()

team = Team()
team.name = 'Sentry'
team.organization = organization
team.save()

project = Project()
project.team = team
project.name = 'Default'
project.organization = organization
project.save()

key = ProjectKey.objects.filter(project=project)[0]
print 'SENTRY_DSN = "$s"' % (key.get_dsn(),)
2.1 Contributing

Want to contribute back to Sentry? This page describes the general development flow, our philosophy, the test suite, and issue tracking.

2.1.1 Documentation

If you’re looking to help document Sentry, you can get set up with Sphinx, our documentation tool, but first you will want to make sure you have a few things on your local system:

• python-dev (if you’re on OS X, you already have this)
• pip
• virtualenvwrapper

Once you’ve got all that, the rest is simple:

# If you have a fork, you'll want to clone it instead
git clone git://github.com/getsentry/sentry.git

# Create a python virtualenv
mkvirtualenv sentry

# Make the magic happen
make dev-docs

Running make dev-docs will install the basic requirements to get Sphinx running.

Building Documentation

Inside the docs directory, you can run make to build the documentation. See make help for available options and the Sphinx Documentation for more information.

2.1.2 Localization

If you’re just looking to help translate Sentry, apply for membership via Transifex.
2.1.3 Developing Against HEAD

We try to make it easy to get up and running in a development environment using a git checkout of Sentry. You’ll want to make sure you have a few things on your local system first:

- python-dev (if you’re on OS X, you already have this)
- pip
- virtualenv (ideally virtualenvwrapper)
- node.js (for npm and building css/javascript)
- (Optional) Redis
- (Optional) Potgresql

Once you’ve got all that, the rest is simple:

```bash
# If you have a fork, you'll want to clone it instead
git clone git://github.com/getsentry/sentry.git

# Create a python virtualenv
mkvirtualenv sentry

# Make the magic happen
make
```

Running `make` will do several things, including:

- Setting up any submodules (including Bootstrap)
- Installing Python requirements
- Installing NPM requirements

**Note:** You will want to store your virtualenv out of the `sentry` directory you cloned above, otherwise `make` will fail.

Create a default Sentry configuration just as if this were a production instance:

```
sentry init
```

You’ll likely want to make some changes to the default configuration (we recommend developing against Postgres, for example). Once done, migrate your database using the following command:

```
sentry upgrade
```

**Note:** The `upgrade` shortcut is simply a combination of South’s migrate, and Django’s syncdb commands.

2.1.4 Coding Standards

Sentry follows the guidelines laid out in pep8 with a little bit of flexibility on things like line length. We always give way for the Zen of Python. We also use strict mode for JavaScript, enforced by jshint.

You can run all linters with `make lint`, or respectively `lint-python` or `lint-js`. 
Spacing

Python: 4 Spaces
JavaScript: 2 Spaces
CSS: 2 Spaces
HTML: 2 Spaces

Django Templates

Newer style code prefers a flat hierarchy of templates to ensure simplicity.
For example, the “create team” template lives at sentry/create-team.html, whereas legacy templates have a url-like hierarchy.
All new templates that are not partials or base layouts must follow this hierarchy.

2.1.5 Running the Test Suite

The test suite consists of multiple parts, testing both the Python and JavaScript components in Sentry. If you’ve setup your environment correctly, you can run the entire suite with the following command:

```
make test
```

If you only need to run the Python tests, you can do so with `make test-python`, as well as `test-js` for the JavaScript tests.

You’ll notice that the test suite is structured based on where the code lives, and strongly encourages using the mock library to drive more accurate individual tests.

**Note:** We use py.test for the Python test suite, and a combination of phantomjs and jasmine for the JavaScript tests.

2.1.6 Static Media

Sentry uses a library that compiles its static media assets (LESS and JS files) automatically. If you’re developing using runserver you’ll see changes happen not only in the original files, but also the minified or processed versions of the file.

If you’ve made changes and need to compile them by hand for any reason, you can do so by running:

```
sentry compilestatic
```

The minified and processed files should be committed alongside the unprocessed changes.

2.1.7 Developing with Django

Because Sentry is just Django, you can use all of the standard Django functionality. The only difference is you’ll be accessing commands that would normally go through manage.py using the `sentry` CLI helper instead.

For example, you probably don’t want to use `sentry start` for development, as it doesn’t support anything like automatic reloading on code changes. For that you’d want to use the standard built-in `runserver` command:
2.1.8 DDL (Schema Changes)

Schema changes should always introduce the new schema in a commit, and then introduce code relying on that schema in a followup commit. This also means that new columns must be NULLable.

Removing columns and tables requires a slightly more painful flow, and should resemble the follow multi-commit flow:

- Remove all references to the column or table (but don’t remove the Model itself)
- Remove the model code
- Remove the table or column

2.1.9 Contributing Back Code

All patches should be sent as a pull request on GitHub, include tests, and documentation where needed. If you’re fixing a bug or making a large change the patch must include test coverage.

Uncertain about how to write tests? Take a look at some existing tests that are similar to the code you’re changing, and go from there.

You can see a list of open pull requests (pending changes) by visiting https://github.com/getsentry/sentry/pulls

2.2 Writing a Client

Note: This document describes protocol version 6 which was introduced in Sentry 7.

A client at its core is simply a set of utilities for capturing various logging parameters. Given these parameters, it then builds a JSON payload which it will send to a Sentry server using some sort of authentication method.

The following items are expected of production-ready clients:

- DSN configuration
- Graceful failures (e.g. Sentry server unreachable)
- Scrubbing w/ processors
- Tag support

Feature based support is required for the following:

- If cookie data is available, it’s not sent by default
- If POST data is available, it’s not sent by default

Additionally, the following features are highly encouraged:

- Automated error handling (e.g. default error handlers)
- Logging integration (to whatever standard solution is available)
- Non-blocking event submission
- Basic data sanitization (e.g. filtering out values that look like passwords)
2.2.1 Client Usage (End-user)

Generally, a client consists of three steps to the end user, which should look almost identical no matter the language:

1. Creation of the client (sometimes this is hidden to the user)

   ```javascript
   var myClient = new RavenClient('http://public_key:secret_key@example.com/project-id');
   ```

2. Capturing an event

   ```javascript
   var $resultId = myClient->captureException($myException);
   ```

3. Using the result of an event capture

   ```javascript
   println('Your exception was recorded as %s', $resultId);
   ```

The constructor ideally allows several configuration methods. The first argument should always be the DSN value (if possible), followed by an optional secondary argument which is a map of options:

```javascript
client = new RavenClient('http://public_key:secret_key@example.com/project-id', {
    'tags': {'foo': 'bar'}
})
```

**Note:** If an empty DSN is passed, you should treat it as a valid option which signifies disabling the client.

Which options you support is up to you, but ideally you would provide defaults for generic values that can be passed to the capture methods.

Once you accept the options, you should output a logging message describing whether the client has been configured actively (as in, it will send to the remote server), or if it has been disabled. This should be done with whatever standard logging module is available for your platform.

Additionally, you should provide methods (depending on the platform) which allow for capturing of a basic message and an exception-type:

- `RavenClient::captureMessage(string $message)`
- `RavenClient::captureException(exception $exception)`

The above methods should also allow optional arguments (or a map of arguments). For example:

```javascript
client.captureException(myException, {
    'tags': {'foo': 'bar'},
})
```

If your platform supports block statements, it is recommended you provide something like the following:

```javascript
with client.captureExceptions(tags={'foo': 'bar'}):
    # do something that will cause an error
    1 / 0
```

**Note:** In the above example, we're passing any options that would normally be passed to the capture methods along with the block wrapper.

Finally, provide a CLI to test your client's configuration. Python example:

```bash
raven test http://public_key:secret_key@example.com/project-id
```

Ruby example:
2.2.2 Parsing the DSN

Clients are encouraged to allow arbitrary options via the constructor, but must allow the first argument as a DSN string. This string contains the following bits:

'\{PROTOCOL\}:\{PUBLIC_KEY\}:\{SECRET_KEY\}@\{HOST\}/\{PATH\}\{PROJECT_ID\}'

For example, given the following constructor:

new RavenClient('https://public:secret@example.com/sentry/project-id')

You should parse the following settings:

- URI = ‘https://example.com/sentry/’
- Public Key = ‘public’
- Secret Key = ‘secret’
- Project ID = ‘project-id’

If any of these values are not present, the client should notify the user immediately that they’ve misconfigured the client.

The final endpoint you’ll be sending requests to is constructed per the following:

'\{URI\}api/\{PROJECT_ID\}/store/'

So in this case, it would end up as:

'https://example.com/sentry/api/project-id/store/'

2.2.3 Building the JSON Packet

The body of the post is a string representation of a JSON object. It is also preferably gzip encoded, which also means its expected to be base64-encoded.

For example, with an included Exception event, a basic JSON body might resemble the following:

```json
{
    "event_id": "fc6d8c0c43fc4630ad850ee518f1b9d0",
    "culprit": "my.module.function_name",
    "timestamp": "2011-05-02T17:41:36",
    "message": "SyntaxError: Wattttt!",
    "tags": {
        "ios_version": "4.0"
    },
    "exception": [{
        "type": "SyntaxError",
        "value": "Wattttt!",
        "module": "__builtins__"
    }]}
```

The following attributes are required for all events:
**event_id**
Hexadecimal string representing a uuid4 value.
Maximum length is 32 characters.

```
{  
  "event_id": "fc6d8c0c43fc4630ad850ee518f1b9d0"
}
```

**message**
User-readable representation of this event
Maximum length is 1000 characters.

```
{  
  "message": "SyntaxError: Wattttt!"
}
```

**timestamp**
Indicates when the logging record was created (in the Sentry client).
Defaults to `datetime.datetime.utcnow()`
The Sentry server assumes the time is in UTC.
The timestamp should be in ISO 8601 format, without a timezone.

```
{  
  "timestamp": "2011-05-02T17:41:36"
}
```

**level**
The record severity.
Defaults to `error`.
The value can either be the integer value or the string label as specified in `SENTRY_LOG_LEVELS`.

```
{  
  "level": "warning"
}
```

Acceptable values are:
• fatal
• error
• warning
• info
• debug

**logger**
The name of the logger which created the record.
If missing, defaults to the string `root`.

```
{  
  "logger": "my.logger.name"
}
```

Additionally, there are several optional values which Sentry recognizes and are highly encouraged:
platform
A string representing the platform the client is submitting from. This will be used by the Sentry interface to customize various components in the interface.

```json
{
    "platform": "python"
}
```

culprit
Function call which was the primary perpetrator of this event.

```json
{
    "culprit": "my.module.function_name"
}
```

server_name
Identifies the host client from which the event was recorded.

```json
{
    "server_name": "foo.example.com"
}
```

release
The release version of the application. This value will generally be something along the lines of the git SHA for the given project.

```json
{
    "release": "721e41770371db95eee98ca2707686226b993eda"
}
```

tags
A map or list of tags for this event.

```json
{

    "tags": {
        "ios_version": "4.0",
        "context": "production"
    }

}
```

```json
{
    "tags": ["ios_version", "4.0"],
    ["context", "production"]
}
```

modules
A list of relevant modules and their versions.

```json
{
    "modules": {
        "my.module.name": "1.0"
    }
}
```

extra
An arbitrary mapping of additional metadata to store with the event.
Any additional value is assumed to be a data interface, where the key is the Python path to the interface class name, and the value is the data expected by the interface. Interfaces are used in a variety of ways including storing stacktraces, HTTP request information, and other metadata.

See Interfaces for information on Sentry’s builtin interfaces and how to create your own.

### 2.2.4 Authentication

An authentication header is expected to be sent along with the message body, which acts as an ownership identifier:

```
X-Sentry-Auth: Sentry sentry_version=5, 
sentry_client=<client version, arbitrary>, 
sentry_timestamp=<current timestamp>, 
sentry_key=<public api key>, 
sentry_secret=<secret api key>
```

**Note:** You should include the client version string in the User-Agent portion of the header, and it will be used if sentry_client is not sent in the auth header.

- **sentry_version**
  The protocol version. This should be sent as the value ‘5’.

- **sentry_client**
  An arbitrary string which identifies your client, including its version.
  
  The typical pattern for this is `client_name/client_version`.
  
  For example, the Python client might send this as ‘raven-python/1.0’.

- **sentry_timestamp**
  The unix timestamp representing the time at which this event was generated.

- **sentry_key**
  The public key which should be provided as part of the client configuration.

- **sentry_secret**
  The secret key which should be provided as part of the client configuration.

  **Note:** You should only pass the secret key if you’re communicating via secure communication to the server. Client-side behavior (such as JavaScript) should use CORS, and only pass the public key.

**crossdomain.xml**

Cross domain requests from flash are supported within the API by specifying sub-policy, which is located at `/api/<project id>/crossdomain.xml`. 

2.2. Writing a Client
2.2.5 A Working Example

When all is said and done, you should be sending an HTTP POST request to a Sentry webserver, where the path is the BASE_URI/api/PROJECT_ID/store/. So given the following DSN:

https://b70a31b3510c4cf793964a185cf1fd0:b7d80b520139450f903720eb7991bf3d@example.com/1

The request body should then somewhat resemble the following:

```plaintext
POST /api/project-id/store/
User-Agent: raven-python/1.0
X-Sentry-Auth: Sentry sentry_version=5, sentry_timestamp=1329096377,
    sentry_key=b70a31b3510c4cf793964a185cf1fd0, sentry_client=raven-python/1.0,
    sentry_secret=b7d80b520139450f903720eb7991bf3d

{
    "event_id": "fc6d8c0c43fc4630ad850ee518f1b9d0",
    "culprit": "my.module.function_name",
    "timestamp": "2011-05-02T17:41:36",
    "message": "SyntaxError: Wattttt!",
    "exception": [
        {
            "type": "SyntaxError",
            "value": "Wattttt!",
            "module": "__builtins__"
        }
    ]
}
```

2.2.6 Reading the Response

If you're using HTTP, you'll receive a response from the server. The response looks something like this:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": "fc6d8c0c43fc4630ad850ee518f1b9d0"
}
```

One thing to take note of is the response status code. Sentry uses this in a variety of ways. You'll **always** want to check for a 200 response if you want to ensure that the message was delivered, as a small level of validation happens immediately that may result in a different response code (and message).

For example, you might get something like this:

```
HTTP/1.1 400 Bad Request
X-Sentry-Error: Client request error: Missing client version identifier

Client request error: Missing client version identifier
```

**Note:** The X-Sentry-Error header will always be present with the precise error message and it is the preferred way to identify the root cause.

If it's not available, it's likely the request was not handled by the API server, or a critical system failure has occurred.
2.2.7 Handling Failures

It is **highly encouraged** that your client handles failures from the Sentry server gracefully. This means taking care of several key things:

- Soft failures when the Sentry server fails to respond in a reasonable amount of time (e.g. 3s)
- Exponential backoff when Sentry fails (don’t continue trying if the server is offline)
- Failover to a standard logging module on errors.

For example, the Python client will log any failed requests to the Sentry server to a named logger, `sentry.errors`. It will also only retry every few seconds, based on how many consecutive failures its seen. The code for this is simple:

```python
def should_try(self):
    if self.status == self.ONLINE:
        return True

    interval = min(self.retry_number, 6) ** 2
    if time.time() - self.last_check > interval:
        return True

    return False
```

2.2.8 Scrubbing Data

Clients should provide some mechanism for scrubbing data. Ideally through an extensible interface that the user can customize the behavior of.

This is generally done as part of the client configuration:

```python
client = Client(..., {
    'processors': ['processor.className'],
})
```

Each processor listed would be some sort of extensible class or a function callback. It would have a single designated method that is passed the data (after it’s been populated), and would then return the data fully intact, or modified with various bits filtered out.

For example, if you simply supported callbacks for processors, it might look like this:

```php
function my_processor($data) {
    foreach ($data['extra'] as $key => $value) {
        if (strpos($value, 'password')) {
            $data[$key] = '********';
        }
    }
}
```

We recommend scrubbing the following values:

- Values where the keyname matches 'password', 'passw', or 'secret'.
- Values that match the regular expression of `\`r\'(?::\d[-\d]*?)(13,16)$` (credit card-like).
- Session cookies.
- The Authentication header (HTTP).

Keep in mind, that if your client is passing extra interface data (e.g. HTTP POST variables) you will also want to scrub those interfaces. Given that, it is a good idea to simply recursively scrub most variables other than predefined things (like HTTP headers).
2.2.9 Tags

Tags are key/value pairs that describe an event. They should be configurable in the following contexts:

- Environment (client-level)
- Thread (block-level)
- Event (as part of capture)

Each of these should inherit its parent. So for example, if you configure your client as so:

```python
client = Client(..., {
    'tags': {'foo': 'bar'},
})
```

And then you capture an event:

```python
client.captureMessage('test', {
    'tags': {'foo': 'baz'},
})
```

The client should send the following upstream for `tags`:

```json
{
    "tags": [
        ["foo", "bar"],
        ["foo", "baz"],
    ],
}
```

You should also provide relevant contextual interfaces. These should last for the lifecycle of a request, and the general interface is “bind some kind of context”, and then at the end of a request lifecycle, clear any present context.

This interface consists of `_context` methods, as well as a “clear context” method. The following is an example API which is implemented in most clients:

```python
# Bind sentry.interfaces.User
client.user_context({
    'email': 'foo@example.com',
})

# Merge in additional tag context
client.tags_context({
    'key': 'value',
})

# Merge in additional extra context
client.extra_context({
    'key': 'value',
})

# Clear context
client.context.clear()
```

Some additional examples of context helpers which might be relevant:

- http_context(data) – bind sentry.interfaces.Http
- wsgi_context(env) – bind http_context based on a wsgi environment
2.2.10 Variable Size

Most arbitrary values in Sentry have their size restricted. This means any values that are sent as metadata (such as variables in a stacktrace) as well as things like extra data, or tags.

- Mappings of values (such as HTTP data, extra data, etc) are limited to 50 item pairs.
- Event IDs are limited to 32 characters.
- Tag keys are limited to 32 characters.
- Tag values are limited to 200 characters.
- Culprits are limited to 200 characters.
- Most contextual variables are limited to 512 characters.
- Extra contextual data is limited to 4096 characters.
- Messages are limited to ~10kb.
- Http data (the body) is limited to 2048 characters.
- Stacktrace’s are limited to 50 frames. If more are sent, data will be removed from the middle of the stack.

2.3 Writing a Plugin

The plugin interface is a work in progress.

Several interfaces exist for extending Sentry:

- Event Filters (sentry.filters)
- Data Interfaces (sentry.interfaces)
- Plugins (sentry.plugins)

Until we get sample docs up, it’s suggested that you review the builtin plugins and the base classes to understand how the system works.

One thing to note, is that all extended methods (outside of data interfaces) should accept **kwargs to handle future changes.

2.3.1 Structure

A plugins layout generally looks like the following:

```
sentry_pluginname/
sentry_pluginname/__init__.py
sentry_pluginname/plugin.py
```

The __init__.py file should contain no plugin logic, and at most, a VERSION = 'x.x.x' line. For example, if you want to pull the version using pkg_resources (which is what we recommend), your file might contain:

```python
try:
    VERSION = __import__('pkg_resources') \
        .get_distribution(__name__).version
except Exception, e:
    VERSION = 'unknown'
```
Inside of `plugin.py`, you’ll declare your Plugin class:

```python
import sentry_pluginname
from sentry.plugins import Plugin

class PluginName(Plugin):
    title = 'Plugin Name'
    slug = 'pluginname'
    description = 'My awesome plugin!'
    version = sentry_pluginname.VERSION

    author = 'Your Name'
    author_url = 'https://github.com/yourname/sentry_pluginname'

    def widget(self, request, group, **kwargs):
        return "<p>Absolutely useless widget</p>"
```

And you’ll register it via `entry_points` in your `setup.py`:

```python
setup(
    # ...
    entry_points={
        'sentry.plugins': [
            'pluginname = sentry_pluginname.plugin:PluginName'
        ],
    },
)
```

If you’re using models or templates, you’ll also want to include the `sentry.apps` entry point to ensure full registration of your app:

```python
setup(
    # ...
    entry_points={
        'sentry.apps': [
            'pluginname = sentry_pluginname'
        ],
    },
)
```

That’s it! Users will be able to install your plugin via `pip install <package name>` and configure it via the web interface based on the hooks you enable.

### 2.3.2 Next Steps

Dig into the rest of the plugin documentation, and take a look at existing plugins for ideas/best practices.

### Plugin Interface (v1)

This document describes version 1 of Plugin API Interface. If you’re working on a new plugin you should transition to version 2.

```python
class sentry.plugins.base.v1.IPlugin
    Plugin interface. Should not be inherited from directly.
```

A plugin should be treated as if it were a singleton. The owner does not control when or how the plugin gets instantiated, nor is it guaranteed that it will happen, or happen more than once.
>>> from sentry.plugins import Plugin
>>> class MyPlugin(Plugin):
>>>     title = 'My Plugin'
>>>     def widget(self, request, group, **kwargs):
>>>         return self.render('myplugin/widget.html')

All children should allow **kwargs on all inherited methods.

**actions** *(request, group, action_list, **kwargs)*

Modifies the action list for a grouped message.

An action is a tuple containing two elements:

('Action Label', 'uri/to/action/')

This must return action_list.

```python
>>> def actions(self, request, group, action_list, **kwargs):
...     action_list.append(('Google', 'http://google.com'))
...     return action_list
```

**before_events** *(request, group_list, **kwargs)*

Allows preprocessing of groups in the list view.

This is generally useful if you need to cache lookups for something like tags which would otherwise do multiple queries.

If you use this at all you should ensure it’s already reset on each execution.

As an example, here’s how we might get a reference to ticket ids we were storing per event, in an efficient O(1) manner.

```python
>>> def before_events(self, request, event_list, **kwargs):
...     prefix = self.get_conf_key()
...     GroupMeta.objects.get_value_bulk(event_list, '%s:tid' % prefix)
```

**can_enable_for_projects** *

Returns a boolean describing whether this plugin can be enabled on a per project basis

**get_conf_key** *

Returns a string representing the configuration keyspace prefix for this plugin.

**get_conf_title** *

Returns a string representing the title to be shown on the configuration page.

**get_description** *

Returns the description for this plugin. This is shown on the plugin configuration page.

```python
>>> plugin.get_description()
```

**get_notification_forms** (**kwargs**)

Provides additional UserOption forms for the Notification Settings page.

Must return an iterable.

```python
>>> def get_notification_forms(self, **kwargs):
...     return [MySettingsForm]
```

**get_option** *(key, project=None, user=None)*

Returns the value of an option in your plugins keyspace, or None if one is not present.

If project is passed, it will limit the scope to that project’s keyspace.
```python
>>> value = plugin.get_option('my_option')
```

### get_resource_links()
Returns a list of tuples pointing to various resources for this plugin.

```python
>>> def get_resource_links(self):
...     return [
...         ('Documentation', 'http://sentry.readthedocs.org'),
...         ('Bug Tracker', 'https://github.com/getsentry/sentry/issues'),
...         ('Source', 'https://github.com/getsentry/sentry'),
...     ]
```

### get_tags(event, **kwargs)
Return additional tags to add to this instance.

Tags should be a list of tuples.

```python
>>> def get_tags(self, event, **kwargs):
...     return [('tag-name', 'tag-value')]
```

### get_title()
Returns the general title for this plugin.

```python
>>> plugin.get_title()
```

### get_url(group)
Returns the absolute URL to this plugin's group action handler.

```python
>>> plugin.get_url(group)
```

### has_perm(user, perm, *objects, **kwargs)
Given a user, a permission name, and an optional list of objects within context, returns an override value for a permission.

**Parameters**

- **user** – either an instance of `AnonymousUser` or `User`.
- **perm** – a string, such as “edit_project”
- **objects** – an optional list of objects

If your plugin does not modify this permission, simply return `None`.

For example, has_perm might be called like so:

```python
>>> has_perm(user, 'add_project')
```

It also might be called with more context:

```python
>>> has_perm(user, 'edit_project', project)
```

Or with even more context:

```python
>>> has_perm(user, 'configure_project_plugin', project, plugin)
```

### is_enabled(project=None)
Returns a boolean representing if this plugin is enabled.

If `project` is passed, it will limit the scope to that project.

```python
>>> plugin.is_enabled()
```
**is_regression** *(group, event, **kwargs)*
Called on new events when the group’s status is resolved. Return True if this event is a regression, False if it is not, None to defer to other plugins.

**Parameters**
- **group** – an instance of `Group`
- **event** – an instance of `Event`

```python
>>> def is_regression(self, group, event, **kwargs):
...     # regression if 'version' tag has a value we haven't seen before
...     seen_versions = set(t[0] for t in group.get_unique_tags("version"))
...     event_version = dict(event.get_tags()).get("version")
...     return event_version not in seen_versions
```

**is_testable** *(**kwargs)*
Returns True if this plugin is able to be tested.

**missing_perm_response** *(request, perm, *args, **objects)*
Given a user, a permission name, and an optional mapping of objects within a context, returns a custom response.

**Parameters**
- **user** – either an instance of `AnonymousUser` or `User`
- **perm** – a string, such as “edit_project”
- **objects** – an optional mapping of objects

If your plugin does not need to override this response, simply return `None`.

**panels** *(request, group, panel_list, **kwargs)*
Modifies the panel list for a grouped message.

A panel is a tuple containing two elements:
- (‘Panel Label’, ‘uri/to/panel/’)

This must return `panel_list`.

```python
>>> def panels(self, request, group, action_list, **kwargs):
...     panel_list.append((self.get_title(), self.get_url(group)))
...     return panel_list
```

**post_process** *(group, event, is_new, is_sample, **kwargs)*
Post processes an event after it has been saved.

**Parameters**
- **group** – an instance of `Group`
- **event** – an instance of `Event`
- **is_new** – a boolean describing if this group is new, or has changed state
- **is_sample** – a boolean describing if this event was stored, or sampled

```python
>>> def post_process(self, event, **kwargs):
...     print 'New event created:', event.id
...     print group.get_absolute_url()
```

**set_option** *(key, value, project=None, user=None)*
Updates the value of an option in your plugins keyspace.

2.3. Writing a Plugin
If `project` is passed, it will limit the scope to that project’s keyspace.

```python
>>> plugin.set_option('my_option', 'http://example.com')
```

**tags** *(request, group, tag_list, **kwargs)*

Modifies the tag list for a grouped message.

A tag is a string, already marked safe or later escaped, that is shown inline with the event.

This must return `tag_list`.

```python
>>> def tags(self, request, group, tag_list, **kwargs):
...    tag_list.append(':(')
...    return tag_list
```

**unset_option** *(key, project=None, user=None)*

Removes an option in your plugins keyspace.

If `project` is passed, it will limit the scope to that project’s keyspace.

```python
>>> plugin.unset_option('my_option')
```

**view** *(request, group, **kwargs)*

Handles the view logic. If no response is given, we continue to the next action provider.

```python
>>> def view(self, request, group, **kwargs):
...    return self.render('myplugin/about.html')
```

**widget** *(request, group, **kwargs)*

Renders as a widget in the group details sidebar.

```python
>>> def widget(self, request, group, **kwargs):
...    return self.render('myplugin/widget.html')
```

### Plugin Interface (v2)

This document describes version 2 of Plugin API Interface.

New in version 7.1.0.

**Note:** Version 2 of the plugin interface is under active development. If there’s a feature you’d like to see consider opening a ticket or submitting a pull request.

```python
from sentry.plugins import Plugin2

>>> class MyPlugin(Plugin2):
...    def get_title(self):
...        return 'My Plugin'
```

As a general rule all inherited methods should allow `**kwargs` to ensure ease of future compatibility.

**can_enable_for_projects()**

Returns a boolean describing whether this plugin can be enabled on a per project basis.
**get_actions** *(request, group, **kwargs)*  
Return a list of available actions to append this aggregate.

Examples of built-in actions are “Mute Event” and “Remove Data”.

An action is a tuple containing two elements:

`('Action Label', '/uri/to/action/')`

```python
>>> def get_actions(self, request, group, **kwargs):
...     return [('Google', 'http://google.com')]
```

**get_annotations** *(request, group, **kwargs)*  
Return a list of annotations to append to this aggregate.

An example of an annotation might be “Needs Fix” or “Task #123”.

The properties of each tag must match the constructor for `sentry.plugins.Annotation`

```python
>>> def get_annotations(self, request, group, **kwargs):
...     task_id = GroupMeta.objects.get_value(group, 'myplugin:tid')
...     if not task_id:
...         return []
...     return [{'label': '#%s' % task_id}]
```

**get_conf_key** ()  
Returns a string representing the configuration keyspace prefix for this plugin.

**get_conf_title** ()  
Returns a string representing the title to be shown on the configuration page.

**get_description** ()  
Returns the description for this plugin. This is shown on the plugin configuration page.

```python
>>> plugin.get_description()
```

**get_event_preprocessors** (**kwargs)**  
Return a list of preprocessors to apply to the given event.

A preprocessor is a function that takes the normalized data blob as an input and returns modified data as output. If no changes to the data are made it is safe to return `None`.

```python
>>> def get_event_preprocessors(self, **kwargs):
...     return [lambda x: x]
```

**get_feature_hooks** (**kwargs)**  
Return a list of callables to check for feature status.

```python
>>> from sentry.features import FeatureHandler
>>> class NoRegistration(FeatureHandler):
...     features = set(['auth:register'])
...     def has(self, feature, actor):
...         return False

>>> def get_feature_hooks(self, **kwargs):
...     return [NoRegistration()]
```

**get_notifiers** (**kwargs)**  
Return a list of notifiers to append to the registry.

Notifiers must extend `sentry.plugins.Notifier`. 

2.3. Writing a Plugin
get_notifiers(self, **kwargs):
    return [MyNotifier]

get_option(key, project=None, user=None)
    Returns the value of an option in your plugins keyspace, or None if one is not present.
    If project is passed, it will limit the scope to that project’s keyspace.
    >>> value = plugin.get_option('my_option')

get_release_hook(**kwargs)
    Return an implementation of ReleaseHook.
    >>> from sentry.plugins import ReleaseHook
    >>>
    >>> class MyReleaseHook(ReleaseHook):
    >>>     def handle(self, request):
    >>>         self.finish_release(version=request.POST['version'])
    >>>
    >>> def get_release_hook(self, **kwargs):
    >>>     return MyReleaseHook

generate_resource_links()
    Returns a list of tuples pointing to various resources for this plugin.
    >>> def generate_resource_links(self):
    >>>     return [('Documentation', 'http://sentry.readthedocs.org'),
    >>>              ('Bug Tracker', 'https://github.com/getsentry/sentry/issues'),
    >>>              ('Source', 'https://github.com/getsentry/sentry'),
    >>>              ]

generate_rules(**kwargs)
    Return a list of Rule classes to add to the registry.
    >>> def generate_rules(self, **kwargs):
    >>>     return [MyCustomRule]

generate_tags(event, **kwargs)
    Return a list of additional tags to add to this instance.
    A tag is a tuple containing two elements:
    ('tag-key', 'tag-value')
    >>> def generate_tags(self, event, **kwargs):
    >>>     return [('tag-key', 'tag-value')]

generate_title()
    Returns the general title for this plugin.
    >>> plugin.generate_title()
render \texttt{(template, context=\texttt{None})}
Given a template name, and an optional context (dictionary), returns a ready-to-render response.
Default context includes the plugin instance.

\begin{verbatim}
>>> plugin.render('template.html', {'hello': 'world'})
\end{verbatim}

set_option \texttt{(key, value, project=\texttt{None}, user=\texttt{None})}
Updates the value of an option in your plugins keyspace.
If \texttt{project} is passed, it will limit the scope to that project’s keyspace.

\begin{verbatim}
>>> plugin.set_option('my_option', 'http://example.com')
\end{verbatim}

unset_option \texttt{(key, project=\texttt{None}, user=\texttt{None})}
Removes an option in your plugins keyspace.
If \texttt{project} is passed, it will limit the scope to that project’s keyspace.

\begin{verbatim}
>>> plugin.unset_option('my_option')
\end{verbatim}

Permissions

As described in the plugin interface, Sentry provides a large suite of permissions which all fire through the \texttt{has_perm} extension point.

In most cases, a superuser (that is, if User.is_superuser is \texttt{True}), will be granted implicit permissions on everything. Additionally, several cases provide overrides via Django’s standard permission system. All permissions are also bound to some level of inherent permission logic, such as projects only being editable by someone who has some level of control on that project.

This page attempts to describe those permissions, and the contextual objects along with them.

add_project
Controls whether a user can create a new project.

\begin{verbatim}
>>> has_perm('add_project', user)
\end{verbatim}

edit_project
Controls whether a user can edit an existing project.

\begin{verbatim}
>>> has_perm('edit_project', user, project)
\end{verbatim}

remove_project
Controls whether a user can remove an existing project.

\begin{verbatim}
>>> has_perm('remove_project', user, project)
\end{verbatim}

add_project_member
Controls whether a user can add a new member to a project.

\begin{verbatim}
>>> has_perm('add_project_member', user, project)
\end{verbatim}

edit_project_member
Controls whether a user can edit an existing member on a project.

\begin{verbatim}
>>> has_perm('edit_project_member', user, member)
\end{verbatim}

remove_project_member
Controls whether a user can remove an existing member on a project.
>>> has_perm('remove_project_member', user, member)

**create_event**

Controls whether a user can create an event on a project (via the API).

>>> has_perm('create_event', user, project)

### Testing

Sentry provides a basic py.test-based testing framework for extensions.

New in version 7.2.0.

In a simple project, you’ll need to do a few things to get it working:

**setup.py**

Augment your setup.py to ensure at least the following:

```python
setup(
    # ...
    install_requires=[
        'sentry>=7.2.0',
    ]
)
```

**conftest.py**

The `conftest.py` file is our main entry-point for py.test. We need to configure it to load the Sentry pytest configuration:

```python
from __future__ import absolute_import

pytest_plugins = [
    'sentry.utils.pytest'
]
```

### Test Cases

You can now inherit from Sentry’s core test classes. These are Django-based and ensure the database and other basic utilities are in a clean state:

```python
# test_myextension.py
from __future__ import absolute_import

from sentry.testutil import TestCase

class MyExtensionTest(TestCase):
    def test_simple(self):
        assert 1 != 2
```
Running Tests

Running tests follows the py.test standard. As long as your test files and methods are named appropriately (test_filename.py and test_function()) you can simply call out to py.test:

```bash
$ py.test -v
```

```
============================== test session starts ==============================  
platform darwin -- Python 2.7.9 -- py-1.4.26 -- pytest-2.6.4/python2.7  
plugins: django  
collected 1 items  

tests/test_myextension.py::MyExtensionTest::test_simple PASSED  
```

More and better docs coming soon..

## 2.4 Interfaces

Sentry implements data interfaces for storing structured data. At its core, an interface describes what it’s storing, and optionally how its data should be rendered.

Within the client, interfaces are referenced by their full Python module path. For example, if you were sending data for the `sentry.interfaces.Message` class, it would look like this in your JSON packet:

```json
{
    // etc.
    "message": "Hello world"
    "sentry.interfaces.Message": {
        "message": "Hello world"
    }
}
```

### 2.4.1 Aliases

As of protocol version 4, most built-in interface types are aliases for easier reference. For example, instead of using the key `sentry.interfaces.Exception`, you can send the key `exception`.

The mapping is as follows:

```python
* 'exception' => 'sentry.interfaces.Exception'
* 'request' => 'sentry.interfaces.Http'
* 'user' => 'sentry.interfaces.User'
* 'stacktrace' => 'sentry.interfaces.Stacktrace'
* 'template' => 'sentry.interfaces.Template'
```

For the canonical names, please see the `get_path` method on each individual Interface.

### 2.4.2 Provided Interfaces

```python
class sentry.interfaces.message.Message(**data)
    A standard message consisting of a message arg, and an optional params arg for formatting.
    
    If your message cannot be parameterized, then the message interface will serve no benefit.
```
• Message must be no more than 1000 characters in length.

```python
>>> {  
>>>     "message": "My raw message with interpreted strings like %s",
>>>     "params": ["this"]
>>> }
```

class sentry.interfaces.exception.Exception(**data)

An exception consists of a list of values. In most cases, this list contains a single exception, with an optional stacktrace interface.

Each exception has a mandatory `value` argument and optional `type` and `module` arguments describing the exception class type and module namespace.

You can also optionally bind a stacktrace interface to an exception. The spec is identical to sentry.interfaces.Stacktrace.

```python
>>> {  
>>>     "values": [{  
>>>         "type": "ValueError",
>>>         "value": "My exception value",
>>>         "module": "__builtins__"
>>>     },  
>>>     # see sentry.interfaces.Stacktrace
>>> }
```

Values should be sent oldest to newest, this includes both the stacktrace and the exception itself.

**Note:** This interface can be passed as the ‘exception’ key in addition to the full interface path.

class sentry.interfaces.stacktrace.Stacktrace(**data)

A stacktrace contains a list of frames, each with various bits (most optional) describing the context of that frame. Frames should be sorted from oldest to newest.

The stacktrace contains an element, `frames`, which is a list of hashes. Each hash must contain at least the `filename` attribute. The rest of the values are optional, but recommended.

Additionally, if the list of frames is large, you can explicitly tell the system that you’ve omitted a range of frames. The `frames_omitted` must be a single tuple two values: start and end. For example, if you only removed the 8th frame, the value would be (8, 9), meaning it started at the 8th frame, and went until the 9th (the number of frames omitted is end-start). The values should be based on a one-index.

The list of frames should be ordered by the oldest call first.

Each frame must contain the following attributes:

- **filename** The relative filepath to the call

  OR

- **function** The name of the function being called

  OR

- **module** Platform-specific module path (e.g. sentry.interfaces.Stacktrace)

The following additional attributes are supported:

- **lineno** The line number of the call

- **colno** The column number of the call
**abs_path** The absolute path to filename

**context_line** Source code in filename at lineno

**pre_context** A list of source code lines before context_line (in order) – usually [lineno - 5:lineno]

**post_context** A list of source code lines after context_line (in order) – usually [lineno + 1:lineno + 5]

**in_app** Signifies whether this frame is related to the execution of the relevant code in this stacktrace. For example, the frames that might power the framework’s webserver of your app are probably not relevant, however calls to the framework’s library once you start handling code likely are.

**vars** A mapping of variables which were available within this frame (usually context-locals).

```python
>>> {
>>>   "frames": [{
>>>     "abs_path": "/real/file/name.py",
>>>     "filename": "file/name.py",
>>>     "function": "myfunction",
>>>     "vars": {
>>>       "key": "value"
>>>     },
>>>     "pre_context": [
>>>       "line1",
>>>       "line2"
>>>     ],
>>>     "context_line": "line3",
>>>     "lineno": 3,
>>>     "in_app": true,
>>>     "post_context": [
>>>       "line4",
>>>       "line5"
>>>     ],
>>>   }],
>>>   "frames_omitted": [13, 56]
>>> }
```

**Note:** This interface can be passed as the ‘stacktrace’ key in addition to the full interface path.

**class** `sentry.interfaces.template.Template(**data)`

A rendered template (generally used like a single frame in a stacktrace).

The attributes `filename`, `context_line`, and `lineno` are required.

```python
>>> {
>>>   "abs_path": "/real/file/name.html",
>>>   "filename": "file/name.html",
>>>   "pre_context": [
>>>     "line1",
>>>     "line2"
>>>   ],
>>>   "context_line": "line3",
>>>   "lineno": 3,
>>>   "post_context": [
>>>     "line4",
>>>     "line5"
>>>   ],
>>> }
```

**Note:** This interface can be passed as the ‘template’ key in addition to the full interface path.
class sentry.interfaces.http.Http(**data)

The Request information is stored in the Http interface. Two arguments are required: url and method.

The env variable is a compounded dictionary of HTTP headers as well as environment information passed from the webserver. Sentry will explicitly look for REMOTE_ADDR in env for things which require an IP address.

The data variable should only contain the request body (not the query string). It can either be a dictionary (for standard HTTP requests) or a raw request body.

```python
>>> {
    "url": "http://absolute.uri/foo",
    "method": "POST",
    "data": {
        "foo": "bar"
    },
    "query_string": "hello=world",
    "cookies": "foo=bar",
    "headers": {
        "Content-Type": "text/html"
    },
    "env": {
        "REMOTE_ADDR": "192.168.0.1"
    }
}
```

Note: This interface can be passed as the ‘request’ key in addition to the full interface path.

class sentry.interfaces.user.User(**data)

An interface which describes the authenticated User for a request.

You should provide at least either an id (a unique identifier for an authenticated user) or ip_address (their IP address).

All other attributes are optional.

```python
>>> {
    "id": "unique_id",
    "username": "my_user",
    "email": "foo@example.com",
    "ip_address": "127.0.0.1",
    "optional": "value"
}
```

class sentry.interfaces.query.Query(**data)

A SQL query with an optional string describing the SQL driver, engine.

```python
>>> {
    "query": "SELECT 1",
    "engine": "psycopg2"
}
```
3.1 System Options

System configuration is handled at two stages:

- **In the `sentry_options` table.**
- **In Sentry configuration (Django settings via `SENTRY_OPTIONS`).**

The options package should be the primary point of configuration outside of critical systems (such as database connection information).

### 3.1.1 `sentry.options`

The options manager is exported as `sentry.options.default_manager` and the public API is also exposed at options module level:

```python
from sentry import options

if options.get('foo') == 'bar':
    print('foo is set to bar')
```

**class** `sentry.options.manager.OptionsManager (cache=``None`` , ttl=``None`` , logger=``None``)`

A backend for storing generic configuration within Sentry.

Legacy Django configuration should be deprioritized in favor of more dynamic configuration through the options backend, which is backed by a cache and a database.

You *always* will receive a response to `get()`. The response is eventually consistent with the accuracy window depending on the queue workload and you should treat all values as temporary as given a dual connection failure on both the cache and the database the system will fall back to hardcoded defaults.

Overall this is a very loose consistency model which is designed to give simple dynamic configuration with maximum uptime, where defaults are always taken from constants in the global configuration.

- Values must be strings.
- Empty values are identical to null values which are represented by `''`.

**delete (``key``)**

Permanently remove the value of an option.

This will also clear the value within the cache, which means a following `get()` will result in a miss.
get (key)
Get the value of an option prioritizing the cache, then the database, and finally the local configuration.
If no value is present for the key, an empty value ("") is returned.

```
>>> from sentry import options
>>> options.get('option')
```

set (key, value)
Set the value for an option. If the cache is unavailable the action will still succeed.

```
>>> from sentry import options
>>> options.set('option', 'value')
```
4.1 Changelog

4.1.1 Version 7.6.0

This releases entirely removes Access Groups. If you’re upgrading form an installation that had yet to migrate away from this system you should first upgrade to a previous version, run the migration wizard, and then continue the upgrade.

- The project-wide Alert system has been removed (to be re-implemented in the future).
- Access groups have been permanently removed.
- Added `access_token` to data blacklist.
- The legacy (unused) search tables have been removed.
- Upgrades must now be applied manually via `sentry upgrade` or with `sentry start --upgrade`. (Don’t forget to use `--noinput` if you’re doing this via automated tooling!)
- `Event.checksum` and `Group.checksum` have been removed.
- The cleanup task has been removed (the command is still available).
- Various optimizations to `cleanup` for Postgres users.
- Within single organization mode users will automatically be added to the default organization.
- Added `Organization.merge_to()` helper to assist with merging organizations. (i.e. in an on-premise install which wants to convert to a single organization)

4.1.2 Version 7.5.4

- Yet another case where valid team membership was being excluded.

4.1.3 Version 7.5.3

- Fix another case where valid team membership was being excluded
4.1.4 Version 7.5.2

• Correctly support SENTRY_PROJECT.

4.1.5 Version 7.5.1

• Fix case where certain pages were not correctly including valid team membership
• Fix default user creation (regain automated signal)
• Fix sampling of internal metrics

4.1.6 Version 7.5.0

This release removes the ability to login or create accounts using a social auth backend.

If your install was based purely on social accounts you’ll need to use the standard reset password flows to recover accounts.

Redis must be at least version 2.6.12.

• Interface.compute_hashes() now receives the platform of the event.
• Server-side data scrubbers were incorrectly filtering invalid interface aliases.
• The sensitive_fields option is now exposed in project settings.
• The default logger name is now an empty value.
• Celery has been upgraded to 3.1 and is now available at ‘sentry.celery’.
• Facebook, Google, and Twitter identities are no longer available.
• Plugin’s inheriting from TagPlugin are now based on v2 of the API.
• Metrics (counters) are now collected both in sentry.tsdb and an optionally configured statsd instance.
• Organizations can now toggle open membership which allows members to freely join/leave any team.
• ProjectKey.user has been removed.
• Organization API keys are now exposed in the UI.
• Team.owner has been removed.
• TeamMember has been removed.
• PendingTeamMember has been removed.
• Added OrganizationMember.counter.
• Added ‘sentry.db.postgres’ optimized Postgres backend.
• Added ReleaseTrackingPlugin and various release-focused endpoints.
• Dedicated configuration pages for Release Tracking and Issue Tracking integrations now exist.
• Several additions to the Release schema.
• Notification integrations are now present within project’s notification settings.
• Fixed an issue with tag key deletion not following explicit constraints.
4.1.7 Version 7.4.3

- Corrected various issues involving sampled data. Things should now sample according to actual MATH.

4.1.8 Version 7.4.2

- Corrected invalid reference to SENTRY_ALLOW_REGISTRATION.

4.1.9 Version 7.4.1

- Correct an issue with AuthProvider’s migrations on MySQL.

4.1.10 Version 7.4.0

- A new features subsystem was added, and many optional features are now run through it.
- The ‘add_organization’ permission is no longer used.
- The ‘add_team’ permission is no longer used.
- SENTRY_ALLOW_REGISTRATION is deprecated in favor of SENTRY_FEATURES[‘auth:register’].
- SOCIAL_AUTH_CREATE_USERS is deprecated in favor of SENTRY_FEATURES[‘social-auth:register’].
- SENTRY_SERVER_EMAIL is no longer used.
- Added first pass API at storing javascript artifacts via release APIs.
- Improved error reporting for JavaScript source/sourcemap errors.
- Improved configuration handling on sentry.cache.
- Added various release API endpoints.
- Added various organization API endpoints.
- Added various tag API endpoints.
- Sourcemaps will now always treat sourceRoot as a path.
- Changed many permissions to use new sentry.access abstractions which are now scoped based, shared with the API scopes.
- Initial first pass at the SSO subsystem.
- The Sentry internal client will now aggressively prevent recursive errors, but allow more internal errors to be recorded to itself.
- An XSS vulnerability with tag values not being escaped (on the group details page) has been resolved.

4.1.11 Version 7.3.2

- Fixed missing beacon task import.
4.1.12 Version 7.3.1

- Updated Gunicorn version and default configuration.
- Fixed periodic task for beacon.

4.1.13 Version 7.3.0

- The SENTRY_ADMIN_EMAIL setting now exists, and should be the technical contact for the install.
- Fixed an issue that would cause job fanout in deletions.
- Notifications are now sent when someone assigns you to an event.
- Release objects are now created automatically.
- Changed team-based API endpoints to use slugs.
- New API endpoints: - Help page list - Help page details - Group tag values - Project tag values
- Moved JavaScript sourcemap processing into language extension. - Sourcemap processing errors are now annotated into the frame.
- Refactored API to be scope-based permissions.
- Added backend support for organization API keys.
- Moved sentry_webhooks into builtins.
- Added reporting Beacon (see docs).

4.1.14 Version 7.2.0

- A py.test plugin now exists to make extension testing easier.
- A Mailgun webhook endpoint is now available for incoming email.
- Added security token for JS source expansion.
- Duct tape workaround for event navigation buttons repeating the same event.
- Origins now support custom protocols as well as relaxed support on other components.
- Minor optimizations for deletion tasks and endpoints.
- Minor optimizations for Redis buffer.
- Added the ability to enable/disable project keys.
- Added audit log entries for project keys.
- Added UI, API, and async task for deleting project tag keys.
- Various fixes/improvements to regression windows.
- Rules will no longer execute duplicate actions.
- EventFrequencyCondition will now fire a maximum of once every 30 minutes.
4.1.15 Version 7.1.4

• Fixed an issue with hourly rollups not expiring accurately in TSDB

If you’ve been running a 7.x series release for a while you may considering running a cleanup script to ensure erroneous values don’t still exist in Redis:

https://gist.github.com/dcramer/55a44904be883f8d03e1

4.1.16 Version 7.1.3

• Resolved an issue with migrations we created in 7.1.2.

4.1.17 Version 7.1.2

• Fix migrations to ensure org/teams aren’t created with empty slugs.

4.1.18 Version 7.1.1

• Suggest realip module over X-Forwarded-For

4.1.19 Version 7.1.0

Plugin v2

The beginnings of version 2 of the plugin interface have landed. This will expand over time but currently includes the following hooks:

• get_actions
• get_annotations
• get_event_preprocessors
• get_notifiers
• get_rules
• get_tags

Other Changes

• crossdomain.xml no longer supports projects-by-slug.
• A basic file storage abstraction is now available internally.
• The project group list API endpoint now has pagination.
• Several API endpoints have been added to public docs.
• X-Forwarded-For is now handled automatically (and we do not support non-proxy installs).
• Various tweaks to GroupMeta caching.
• The raven-python Sentry server-specific client code is now part of Sentry.
• Various improvements to tag rendering.
Various improvements to bulk deletion strategies.

Added NodeStore.multi_delete.

### 4.1.20 Version 7.0.0

Between version 6.4.x and 7.0 a significant number of changes have landed (approximately a years worth of commits). There are a large number of overall architecture changes, as well as various API compatibility changes. This document does not attempt to cover them all.

As many things have changed, it’s strongly recommended that you regenerate your `sentry.conf.py` (using `sentry init`) and merge in your custom settings.

#### Backwards Incompatible Changes

- Django has been upgraded to 1.6
- The buffer API has been rewritten and the Redis buffer has had its performance greatly improved.
- The UDP server has been removed. Threaded/async models or a buffer proxy are the preferred replacement.
- The `is_rate_limited` plugin hook has been removed in favor of singular quota managers.
- The trends feature has been removed until it can be reimplemented in a more scalable way.
- Filters have been removed. Integrations should use the tagging infrastructure instead.
- `NodeStore.generate_id()` now returns a base64-encoded UUID.
- The API for interfaces has been rewritten.
- `GroupMeta.objects.get_value` no longer errors when a value is missing.
- `sentry.plugins.sentry_sites` has been removed.
- The Search API has been rewritten and now powers the entire stream.
- Removed `Event.{logger,site,logger,level,culprit}` references/columns. Legacy attributes for transitional behavior are available.
- Removed `Project.owner`.
- Removed `Team.owner`.
- `is_new` in `Plugin.post_process` will only be set if the event is brand new.
- Project-based URLs are changed to be organization scoped instead of team scoped.
- Static assets are no longer bundled as part of the repository and are instead compiled during the sdist/installation phases (as needed). See installation notes for further information.
- Graphs have been transitioning to `sentry.tsdb` (historical data is not preserved)
- Workers must now also run `celerybeat`. This can be done either via passing `-B` to worker, or by running another process: `sentry celerybeat`.
- Access Groups are now deprecated and will be removed in a version version.
Organizations

Organizations are the new top-level item of your Sentry install. All membership and team management has been moved into the new organization hub.

Generally for self-hosted installations you’ll only have a single organization.

Migrations will happen automatically for this, and while these were able to applied cleanly and automatically on getsentry.com, we suggest considering snapshotting your database before running them. Additionally consider doing this at a period of the day where users generally won’t be tweaking options to team membership.

TSDB

A new time-series backend is now used (removing the legacy SQL-based graphing solutions). The only currently supported/bundled backend requires Redis. More information can be found in sentry.tsdb.

Rules

A new system for basic controls around notifications (called Rules) has been added. These are per-project hooks for doing dynamic customization of “what happens when a new event is seen?”.

Web API

The first offering a public API has landed. You’ll find it with the new embedded docs (at /docs/).

Protocol Version 6

Version 6 of the protocol has been introduced. The only change is the addition of the release attribute.

BIGINT

All integer fields have been replaced with bigint’s. This migration is not automated (due to it causing locking) and if you need it locally you will need to sort out the migration on your own, as well as set SENTRY_USE_BIG_INTS = True in your configuration.

Help Pages

The beginnings of bundled documentation now exists. The defaults are generated from disk (see sentry/data/help_pages/).

4.1.21 Version 6.4.0

Some major backend changes are introduced in Sentry 6.4.0.

django.contrib.auth

The builtin Django authentication module is no more. The Group and Permission models are no longer used, and the User model has been replaced with sentry.User.
Search

Search has been abstracted from the simple search manager, and now lives within `sentry.search`. Currently only the Django backend is supported, but some sample code for Solr Cloud integration is also available.

Event Storage

A new application called `nodestore` now manages the event blob data. By default it uses a Django-based backend (storing each node as a row in a table), however a Riak backend is included and fully supported.

Additional Changes

- Some initial support for time zones (user setting).
- You can now choose which tag annotations are applied to the event stream via Project Settings -> Tags.
- A new subsystem for handling email is available in `sentry.utils.email`.
- You can now receive email notifications for notes.
- Charts now show tooltips describing the datapoint.
- JavaScript sourcemaps now support embedded sources.
- Stream annotations can now be customized to show any tags (not just number of users).
- Stacktrace frames now get truncated down to a maximum length of 50.

Protocol Version 5

- `sentry_version` should be sent as ‘5’.
- The stacktrace interface now accepts a ‘frames_omitted’ tuple.

4.1.22 Version 6.3.0

- Most of the db utilities were refactored into `sentry.db`.
- The user interface now accepts an `ip_address` attribute.
- User tracking will now use the `ip_address` attribute if available.
- `time_spent` is now expected to be sent as an integer in milliseconds.
- A new Notes feature is available for leaving comments on an event.

4.1.23 Version 6.2.0

- Added tracking to which users have seen which groups
- The stream now reflects if you’ve viewed an event since it was created or regressed.
- The group details page shows other users who have glanced at an event.
- Streams which have recorded user data will now show the number of unique users an event has happened to.
- Various stream polling fixes and improvements.
4.1.24 Version 6.1.1 (Security Release)

Sentry 6.1.1 is a security release which patches a remote code execution exploit.
This only affects servers hosted public clients (e.g. raven.js).
The following commit demonstrates the exploit, as well as the patch:
https://github.com/getsentry/sentry/commit/5793c6cac19aeb7d2e19f9a09f4421b771af4306

4.1.25 Version 6.1.0

- Added Rate Limit controls.

4.1.26 Version 6.0.0

Say hello to our new logomark! Version 6 of Sentry includes many changes within the core designed to improve future compatibility for various platforms.

It also includes quite a number of incompatible changes, so it’s recommended that plugin authors read through the notes carefully, and run some simply sanity checks. It also introduces version 4 of the protocol.

Incompatible Changes

- Django has been upgraded to 1.5.
- django.contrib.auth.models.User has been replaced with sentry.models.User.
- sentry_servers no longer exists, and the behavior is now always present. If you had it manually listed in INSTALLED_APPS, simply remove it.
- Filters have been deprecated, and most functionality removed. See notes below.
- TrackedUser, AffectedByUser, and Group.users_seen have been removed in favor of reusing the tagging architecture for user tracking.
- Suspension of team members is no longer available. The feature was rarely used, and removing it greatly simplifies the auth logic in the API.
- The mail plugin can no longer be disabled, and all mail-specific options have been removed.
- Nearly all data within an event now has a fixed max size. See client developer documentation for details.
- Project keys are no longer created for individuals by default
- sentry.conf.settings has been removed
- LOG_LEVELS, DEFAULT_LOG_LEVEL, and DEFAULT_LOGGER_NAME are no longer configurable.
- DEFAULT_ALERT_PROJECT_THRESHOLD and DEFAULT_ALERT_GROUP_THRESHOLD are no longer configurable.
- SENTRY_EMAIL_SUBJECT_PREFIX and SENTRY_SERVER_EMAIL are no longer used, and default to the appropriate Django options.
- SENTRY_CACHE_BACKEND is no longer configurable.
- SENTRY_AUTH_PROVIDERS is now AUTH_PROVIDERS.
- Existing account recovery tokens are no longer valid.
• sentry.utils.router has been removed.

Protocol Version 4

• sentry_version should be sent as ‘4’.
• Aliases are now recommended instead of full interface names.
• The exception interface now supports a bound stacktrace, and all events (in all protocols) which contain both an exception and a stacktrace will be rolled up. This should be sent as the `stacktrace` attribute.
• The exception interface now supports chained exceptions, and should be sent as a list of exception values. Check the updated documentation for ordering details.

Alerts

A basic alert system has been added.
• Configured at the project level.
• Signaled via Plugin.on_alert.

Filters

The outdated filter system has been mostly removed. You should rely on tagged data for filter generation.
If you were previously defining `SENTRY_FILTERS` you should remove it from your settings as it will be entirely obsolete in a future version.

Other Changes

• Activity feeds will now attempt to filter out some duplicates.
• Tags now show on the event details page.
• Corrected some invalid behavior when storing tags with an individual event.
• Plugin.get_tag_values was added.
• A new team select dashboard exists if you are a member of multiple teams.
• A new Sentry logo mark has replaced the simple text header.
• C# has been added to experimental clients.

4.1.27 Version 5.4.0

SENTRY_PUBLIC now dictates whether or not a Sentry install should be considered accessible by all users or not. This should better solve the use-case of companies hosting a Sentry instance internally and not necessarily needing the permissions that teams give you.

If enabled, all teams and all projects will be accessible all members (for any endpoint which does not require a certain level of access). Project.is_public now dictates the implied state of all events, but will not allow a non-member to browse events.

Additionally, this includes the following other items:
• [Important!] The URLs for social authentication have been moved.
• Improved rendering of data values in all interfaces.
• django-compressor was replaced with django-static-compiler.
• A better defined public view of events (which removes several items from being visible).
• Improved SourceMap discovery.
• Most events will no longer cause a hard error when validation fails. Instead we attempt to drop any non-required
data so that at least a partial event is stored.
• MessageCountByMinute was renamed to GroupCountByMinute.
• MessageFilterValue was renamed to GroupTag.
• Syntax highlighting was disabled (pending performance solutions).
• Added Team.date_added column.
• Lots of various design changes.
• Trending SQL queries can now be disabled by setting SENTRY_USE_TRENDING = False

4.1.28 Version 5.3.0

A brand new Sentry design has landed.

Some things of note:
• An improved dashboard.
• Improved activity feeds on aggregate details.
• Similar event navigation links on aggregate details.
• Redesigned team management flow.

Additionally:
• Aggregates now happen on (project, checksum), which means events that have different levels or culprits can
 now be grouped together.
• Sentry now requires authentication for all pages.
• SENTRY_PUBLIC behavior has changed to signify the default state of projects.
• Project slug’s are now only unique within a team.

4.1.29 Version 5.2.2

• [New] The dashboard will now stream updates to the new and trending event components.

4.1.30 Version 5.2.1

• [Fix] Trends sorting options on the Stream page work correctly again.
4.1.31 Version 5.2.0

- [New] Syntax highlighting now exists on all context frames.
- [New] Support for JavaScript Sourcemaps now exists.
- [New] The server will now fetch remote source files for JavaScript events.
- [New] Sentry will now ask for your project’s platform.
- [Fix] Resolved -> Regressed state change is now atomic.
- [Fix] cleanup now runs with lower resource overhead.
- [Fix] Cookies will now be coerced to dicts if possible.
- More (storage only still) work on user tracking.
- Several indexes were added to speed up various queries.
- Removed savepoint use in plugin hooks (99% of the time there were no queries).

Additional, the following client protocol changes are part of this release:

- Cookies should be not be sent by default.
- POST data should not be sent by default.
- Recommended values regexp for sanitizing credit cards was updated.
- colno was added to the Stacktrace spec.
- Timestamps that are more than one minute in the future are now discarded.
- (Undocumented) Client-side support now exists for GET + Referrer store requests.

We’ve also reduced the test suite time down to 25% of what it originally was (thanks, Alex!)

4.1.32 Version 5.1.3

A new user’s affected-tracking mechanism is present (storage only). This will become available in the UX in a future version, and relies on the existing HTTP and User interface datas.

- [Fix] Correct a bug with search queries.
- [New] Group.users_seen will now track unique users when possible.
- [New] Team and project owners can now change ownership (non-superuser).
- [Fix] Counts are now formatted correctly when number of visible digits > 3.

4.1.33 Version 5.1.2

- [Fix] Option schema (key length = 64).
- [Fix] Template interface now renders correctly.
- [Fix] Update design on admin status pages.
- [New] iOS client documentation.
- Improved client documentation visuals.
4.1.34 Version 5.1.1

Several schema changes are made in this upgrade:

- A new model: LostPasswordHash
- Two new fields on ProjectKey: date_added and user_added
- A new field on Event: platform
- A new field on Group: platform

The following changes are also part of this release:

- [New] A new plugin, sentry-interface-types will now automatically tag the available interface types in an event.
- [New] The platform value sent with an event is now recorded in the database.
- [New] A recover account flow has been added.
- [New] There is now a sticky nav on the event details pages.
- [New] getting started page now shows if there’s no data for a project.
- [New] An API key management page now exists (under project settings).
- [Fix] Line numbers now show correctly in collapsed source context.
- [Fix] Pending members now show correctly on the team management page (under project settings).
- [Fix] The time since value now updates correctly when events change.
- Various changes to how event details components render.
- Various fixes for breadcrumbs and header styles.
- Most plugins will no longer default to enabled on new projects.

4.1.35 Version 5.1.0

Minor point releases now signify major changes (5.1.0 is a major release). Bugfix and smaller releases will continue as normal.

This release includes a very large set of changes from 5.0.0, including a new client protocol and an overhaul to the frontend code (specifically the JavaScript).

If you’re a contributor, take note that there is now a JavaScript test suite. You can run all test suites with the `make test` command.

Protocol Version 3

Sentry 5.1 removes support for version 1.0 of the protocol, and maintains a compatibility layer for version 2.0.

Additionally, the following changes apply to the new protocol:

- `sentry_version` should be sent as ‘3’ (not 3.0).
- Signed messages are no longer supported (signatures are not calculated). - `sentry_signature` is no longer used. - `sentry_timestamp` is no longer used.
- Clients must pass `sentry_secret` for server-side requests as part of the auth header. This check runs in the event that there is no Origin header sent. - version 2.0 supports validation of this, version 3 requires it.
• The `project` attribute in the JSON packet is no longer required.
• The `platform` attribute is now recommended.
• The `tags` attribute is now recommended.

Other Changes

• Aggregation ‘Views’ have been removed.
• All streaming components are now powered by Backbone.js.
• Frontend has been updated to Bootstrap 2.1.
• The event stream now includes sparklines representing the last 24 hours of data for each event.
• Trends have greatly improved.
• Grouping events that have identical stacktraces other than the function name (e.g. dynamically generated functions) is not possible.
• SiteFilter has been removed, as has the sites plugin.
• If frames are included in a stacktrace that have the attribute `in_app: false`, they will be hidden by default in the details view.
• crossdomain.xml support is now available (see documentation).
• The search feature now uses buffers to better handle write concurrency.
• Early support for WSGI (should be functional) exists as `sentry.wsgi`.
• Many fixes around cache usage.

4.1.36 Version 5.0.0

• Variable versions of Django are no longer supported. Django 1.4.x must now be used.
• Public projects are restricted to viewing without being authenticated.
• The default behavior of Sentry is to now use timezone-aware datetimes everywhere.
• Permissions have been refactored to be more precise.

4.1.37 Version 4.10.0

• A new IssuePlugin base is available.
• Charts have been refactored to show 7 days worth of data.
• `django.contrib.staticfiles` is now supported.
• `django.contrib.messages` is now supported.

4.1.38 Version 4.9.0

Social authentication is now supported!

By configuring several options (documented in the getting started guide), you can allow users to signup and login with accounts from several social services. Additionally users can associate their existing accounts (via a new identities panel in their account settings) with any number of these services.
Currently, the following services are supported:

- GitHub
- Twitter
- Facebook
- Google

### 4.1.39 Version 4.8.0

Tags are now able to be created dynamically! See the documentation for more details on implementing it inside of your client.

You can also configure which tags will show up as filters in the sidebar via your project’s settings page. By default all tags will show up.

### 4.1.40 Version 4.7.0

User options are now available to plugins. The builtin mail plugin also now takes advantage of them.

- `sentry_mail` will now use the `alert_email` option over your account email if specified.
- `sentry_mail` now allows you to choose which projects to receive alerts for.
- Numeric slugs will no longer return 404s.
- Corrected the repair command with the `--owner` argument.

### 4.1.41 Version 4.6.0

Improvements to the dashboard are trickling in, with this update including:

- Changing “Top Events” to “Trends” (if supported by your RDBMS)
- Time interval selections for both widgets

Plugins now have a new interface for registering themselves. Please see the developer documentation for notes on how to use `entry_points`.

Additionally:

- Signed messages are now deprecated. Save yourself some CPU cycles, and rely on SSL.
- A new plugin which tags user’s email addresses.
- Several UI cleanups in various areas, such as improvements to the account dropdown and lists of projects/teams.
- Long awaited cleanup/fixes for applying the initial database migrations (sorry MySQL users).
- Initial support for arbitrary event tagging (coming soon to a client near you).

### 4.1.42 Version 4.5.0

Built-in plugins have been refactored to be more concise. This includes changes to the servers, urls, and sites plugins.

There is also a new built-in plugin:

`sentry.plugins.sentry_useragents`
With these changes, it now makes it even easier to create a basic plugin that just handles extra “tag”-like data. Take a look at one of the aforementioned plugins for an example.

Additionally:

- The mail plugin now sends additional Sentry-specific headers.
- Signatures are deprecated, and no longer required.
- Several fixes regarding CORS support.

### 4.1.43 Version 4.4.0

Two new columns were added to the Group model:

- **resolved_at**: The datetime at which this event was marked as resolved.
- **active_at**: The datetime at which this event was marked as open.

The dashboard’s “New Events” will now use the active_at date rather than the original first seen date on an event.

### 4.1.44 Version 4.3.0

A new global dashboard now exists. The dashboard contains an event graph for all projects which you have access to, as well as a list of the top events, and new events.

Additionally:

- API endpoints now properly send no-cache headers.
- Added a countdown to update buffers.

### 4.1.45 Version 4.2.0

Sentry now has support for buffering a majority of its counters via Redis. For more information, check the buffer section of the configuration documentation.

- Nearly all race conditions are now handled correctly using distributed cache locks.

### 4.1.46 Version 4.1.0

Sentry now maintains full support for CORS requests from external hosts. This means that clients like raven-js will now able to securely send messages to Sentry, without compromising their secret key.

In addition:

- Trusted domains were removed. These were not fully implemented.
- A new datepicker for selecting ranges of events.
- raven-javascript and raven-ruby are now officially supported clients.
- Added sentry repair.
4.1.47 Version 4.0.0

Introducing Teams!

Projects are now assigned to a single team, and that team may consist of many members. Additionally each team may own multiple projects. This makes it much easier to handle permissions across a single organization that has many projects.

This update also includes initial changes to how filters work. ServerName Filter and SiteFilter are no longer specified in the FILTERS setting, but instead are included automatically if you load the respective sentry_servers and sentry_sites plugins.

Additional changes:
- Legacy data format is no longer supported (pre-Raven 1.x)
- API will now validate data before sending it into the queue, enabling better debugging for clients.

4.1.48 Version 3.8.0

Plugins are now configurable per-project. This makes it easy to install a plugin globally and allow it to run only on selective projects.

4.1.49 Version 3.7.0

Several minor fixes, as well as a backwards incompatible change with filters.

All filters must now accept the project argument in the initializer. The signature is now __init__(request, project).

4.1.50 Version 3.6.0

This version focuses on improving membership control. It includes the ability to revoke project members access temporarily (by suspending them), as well as revoking users (outside of Sentry) via the user.is_active flag.

Additionally, it brings the beginnings of an “invite user” flow, allowing you to invite users who may or may not already have accounts in the system. We plan to improve this flow in an upcoming release to allow invitees easy registration within the system if they don’t already have an account.

4.1.51 Version 3.5.0

Several improvements and additions have been made around the administration, including performance improvements.

If you’re utilizing the queue, the installation guide now recommends you switch off the database backend, and move to something more efficient (such as Redis).

4.1.52 Version 3.4.0

Eventlet is no longer used as the default worker for gunicorn, and thus is no longer installed by default. If you’re using the udp worker, or send_fake_data you MUST install eventlet yourself. The recommended worker class for gunicorn is now gevent as it corrects some issues with async queries in psycopg2.
4.1.53 Version 3.3.0

The queue has been changed from direct Kombu, to utilizing Celery entirely. If you were already using the queue, the upgrade simply requires you to change “sentry start worker” to “sentry celeryd”.

4.1.54 Version 3.0.0

Version 3 of Sentry is centered around a restructure of the internal services and the architecture for running those. It also includes improvements to the plugin architecture.

The upgrade process should be trivial, just be aware that if you were using --config before you must change the way you pass it so that it’s before the subcommand. For example if you were doing `sentry start --config...` the command would now be changed to `sentry --config... start`.

- Sentry no longer provides the ability to daemonize processes. This should now be done at the system level.
- All of Sentry’s CLI is now handled through Logan (https://github.com/dcramer/logan), which simply pipes commands to Django’s internal system utilizing custom configuration.
- Plugins now have a hook for managing some level of permissions (beyond the required defaults).
- The plugin interface (IPlugin) is now documented.
- It is now recommended that clients allow the Sentry server to compute checksums.

4.1.55 Version 2.9.0

- Plugins must now behave as singletons and be registered with a newly provided `sentry.plugins.@register` decorator.
- The Bugzilla and Redmine extensions are no longer part of Sentry core.
- Added a global overview of projects to the administration.

4.1.56 Version 2.8.0

- Added an Account Settings panel which allows users to change their name, email, and password.
- The default Sentry server now correctly wraps itself in its own middleware.
- Improved Real-time JavaScript.

4.1.57 Version 2.7.0

- Added `first_seen` and `last_seen` to all message filter values.
- Added a new “since” option to the dashboard with a default value of 3 days.

4.1.58 Version 2.6.0

- The built-in webserver is now powered by gunicorn.
- Cleaned up several admin pages and split them into sub-pages.
4.1.59 Version 2.5.0

- Corrected some queue behavior.
- Resolve Feed now only resolves items active within current filters.
- Handle unicode characters in POST body for replay request.
- Ensure client side requests run checks on HTTP_REFERER
- Adjust documentation for service settings.

4.1.60 Version 2.4.5

- Corrected indexing behavior to handle non strings.
- If queuing is enabled the indexer will now queue it’s jobs.
- Moved group creation into a transaction.

4.1.61 Version 2.4.4

- Mail configuration value of send_to now correctly uses ‘,’ as a separator (rather than ‘;’).
- Changed rendering of sidebar widgets for all builtin extensions.
- Added an event details slot to the sidebar pane for individual events.

4.1.62 Version 2.4.3

- Correct some behavior with MySQL on the groups pane.
- Correct generated configuration behavior to use absolute paths for run and log folders.

4.1.63 Version 2.4.2

- Include DSN and member type on projects list grid.

4.1.64 Version 2.4.1

- Change appearance of resolve states.
- Adjust user management to link username (which is required) and not optional fields.

4.1.65 Version 2.4.0

- Added user management for admins.
- Resolved events should now appear differently.
- Default membership access is now configurable.
4.1.66 Version 2.3.2

• Maintaining some level of support for SQLite.

4.1.67 Version 2.3.1

• Correct rendering of sites, urls, and servers on details panes.

4.1.68 Version 2.3.0

• The polling API and JavaScript have been refactored.
  Events now stream in (they generally do not update if they already exist), and will maintain correct ordering in the feed.
  This API is available for most sort options, excluding trends.
• The builtin plugin's widgets have been fixed.
• Sampling rates are now configurable.
• Some minor design tweaks.

4.1.69 Version 2.2.5

• The |date filter now forces things to UTC (it assumes local time).
• Event templates have been updated to resemble groups.

4.1.70 Version 2.2.4

• Improve error logging in API.
• Update Celery client code to use new send_encoded interfaces.
• Change JS datetimes to use UTC.
• Force clients which specify version 2.0 or newer to pass identification.
• Better default logging configuration.
• Adjust eventlet to monkey patch the world before any imports happen.
• Adjust default configuration to specify LOG and RUN directories.
• upgrade now correctly handles the delete ghosts argument.

4.1.71 Version 2.2.3

• Lower font size of counts on event list.
• Align actions to right side (vertical).
• Fix issue with long filters not transforming to selects.
4.1.72 Version 2.2.2

- Corrected event_id key to contain (project_id, event_id).
- Adjusted project form to default the user to the current user when accessed by an admin.
- Change sentry admin to be /manage/ to avoid any unintentional conflict with the Django admin.

4.1.73 Version 2.2.1

- Cleaned up several pages.

4.1.74 Version 2.2.0

- Sentry has a brand new design utilizing Bootstrap 2.
- Superusers can now create projects for users.

4.1.75 Version 2.1.3

- Ensure we truncate tokens to 128 characters for SearchDocument.
- Gracefully handle errors with indexing.
- Gracefully handle errors with post_process.
- Gracefully handle errors with regression_signal.
- Fixed priority sort option not activating.

4.1.76 Version 2.1.2

- Fixed an issue that was causing signals to not be registered.
- Made date the default sort order for aggregate stream.

4.1.77 Version 2.1.1

- Fixed an issue with indexing pre and post context on templates.

4.1.78 Version 2.1.0

- pytz is now a requirement.
- Changed default TIME_ZONE to be “UTC”.
- Corrected some issues around how dates were localized.
- Initial implementation of full-text search.
- Fixed sending of regression_signal so it only happens if event has been marked as new.
- Changed ProjectMember.get_dsn() to use request.get_host() which corrects a bug in some webservers.
4.1.79 Version 2.0.2

- `start` no longer performs `upgrade` as its problematic.
- Initial queue usage (optional).
- Fix reference to bookmark_querystring.
- Added DSN to project member details page.

4.1.80 Version 2.0.1

- Skip logging of south in default server configuration as it proves to be problematic.
- Remove use of deprecated logging handler in default server configuration.
- Run `upgrade` as part of `start`.
- Fix GroupBookmark related_name to be sentry namespaced.
- Correctly handle before_events() hook in polling responses.
- Removed integrated install documentation as it is no longer officially supported.
- Gracefully handle rendering errors with interfaces.
- Correct a bug with default email options.

4.1.81 Version 2.0

Sentry 2.0 is a major release which contains many new features as well as some large rearchitecting of the codebase. If you were previously extending Sentry, it would be wise to test your extensions before upgrading.

- Added project scoping to all data in sentry.
- Added permissions to projects.
- The authorization header is now X-Sentry-Auth to avoid certain default behaviors such as mod_wsgi’s “don’t pass HTTP Authorization header”.
- `sentry.client` has been removed
- The default key is now base64-encoded.
- `sentry.interfaces` are now used for coercing and rendering structured data.
- The store endpoint has a new API.
- The from_kwargs method has a new API.
- The class_name, traceback, an url fields have been removed.
- GroupedMessage was renamed to Group.
- Message was renamed to Event.
- Switched to Bootstrap framework.
- Added “Replay Request” action for events.
- Graphs can now be generated for Projects.
- There is now a trends sort option for aggregated events.
- Added `sentry manage` command.
Sentry now has its own isolated queues (using Kombu).

4.1.82 Version 1.13.5

- Level filters are now precise (they no longer show their level + messages from higher levels).

4.1.83 Version 1.13.4

- Updated message details page to resemble look and feel of group details.

4.1.84 Version 1.13.3

- Added back in the “raw traceback” view.

4.1.85 Version 1.13.2

- Counts will now render differently with large values (e.g. 13000 will now be 13k)

4.1.86 Version 1.13.1

- Search by message_id will now display a list of results if there are multiple matches.

4.1.87 Version 1.13.0

- Deprecated the Sentry client, and added Raven to the as the default builtin.
- Removed Highstock and replaced it with an awesome OPEN SOURCE alternative, jQuery Flot.
- Default Sentry server options should now work out of the box.
- Packaged Google Web Font as part of Sentry.

4.1.88 Version 1.12.2

- Fixed infinite loop.

4.1.89 Version 1.12.1

- Stabilize migration schema (solves problem with index creation fail introduced in 1.12.0).

4.1.90 Version 1.12.0

- Adjusted message_id to include the checksum to avoid situations were the message_id that was generated did not exist due to sampling.
4.1.91 Version 1.11.4

• Several design improvements for group/message details panes.

4.1.92 Version 1.11.3

• Fixed a bug that was causing exception summary to not show.

4.1.93 Version 1.11.2

• Cleaned up message and group details pages (adding back some missing information).
• Cleaned up some design around odd margins/padding.

4.1.94 Version 1.11.1

• Fixed a bug that would cause the admin (and potentially other modules) to get loaded multiple times.

4.1.95 Version 1.11.0

• Added license headers.
• Removed `sentry.helpers` and `sentry.routers`.
• Global module versions are now cached to avoid continuous path walking and import overhead.

4.1.96 Version 1.10.1

• Fixed an issue that was causing servers to show logger names.

4.1.97 Version 1.10.0

• You can now pass `extra={'stack': True}` to logging methods to capture the current frames and their locals.
• Code refactoring in various places related to stack extraction.
• Denormalized graph data in MessageCountByMinute (stores at 5m intervals).
• Denormalized filter counts into MessageFilterValue
• Added message sampling (adapted from Yuri Baburov’s patch).
• Added SENTRY_MAIL_LEVEL setting.
• Added SENTRY_MAIL_INCLUDE_LOGGERS setting.
• Added SENTRY_MAIL_EXCLUDE_LOGGERS setting.
• Added the `level` argument to the cleanup command.
• The thrashed key is now set correctly in request.sentry.
• Added user information to all messages that have `request`.
• Changed the hashing function for messages that include stacktraces to ignore the `message` and line numbers.
• Much improved test coverage.

### 4.1.98 Version 1.9.0

• Load the Frequency chart asynchronously.
• Frequency chart no longer shows for SQLite.
• Switch graphing library to Highstock from Highcharts.

### 4.1.99 Version 1.8.10

• Ensure dictionary keys are coerced to strings.
• Fixed path to missing_permissions.html.

### 4.1.100 Version 1.8.9

• Safely handle unpickling objects that may not have been stored in a valid format.

### 4.1.101 Version 1.8.8

• Changed `sentry cleanup` to use a range query.

### 4.1.102 Version 1.8.7

• Added “Clear Feed” option.
• Version information will be read from pkg_resources if possible.
• Cleared up documentation on configuration settings.
• Performance improvements to `Client.send()`.
• Added default 404/500 pages.
• Added support for Django’s LOGIN_URL setting.
• Fixed a memory leak in the client (thanks to Ben Bangert).

### 4.1.103 Version 1.8.6.2

• Reverted change which required distribute.
• Cleaned up configuration defaults.

### 4.1.104 Version 1.8.6.1

• Include distribute_setup.py in the MANIFEST to ship it in sdists.
4.1.105 Version 1.8.6

- Corrected an issue which was causing certain settings (like WEB_HOST) to not take affect in custom configuration files.
- The Sentry server will now pull in default server settings, as well as ~/.sentry/sentry.conf.py if --config is not passed to it.

4.1.106 Version 1.8.5.1

- Fixed a bug which caused thrashing prevention to not function correctly.
- Corrected an error in transform()'s recursion safety.
- Changed packaging to use Distribute.

4.1.107 Version 1.8.5

- Pulled test suite out of the Sentry namespace to avoid conflicts in projects.

4.1.108 Version 1.8.4.2

- Added missing invalid_message_id template.

4.1.109 Version 1.8.4.1

- Fixed an error that was causing LOG_LEVELS to not display correctly.

4.1.110 Version 1.8.4

- The Sentry base client will now pass along the timestamp from when the message was generated.

4.1.111 Version 1.8.3.1

- Fixed a case where the client may hit a transaction aborted error when transforming variables.

4.1.112 Version 1.8.3

- Added several settings which were Django specific so that they can be configured isolated to the Sentry instance.
- Graceful failover for when a cache backend isn’t working properly and throttling is enabled.
- Better rendering of non-dict variables when passed within extra data.
- Graceful failover for searching on message_id’s that aren’t found.

4.1.113 Version 1.8.0

- Refactored Sentry server to run standalone (sentry --help).
4.1.114 Version 1.7.5

• Implemented new client/server storage API and signing methods.
• Fixed a bug where accessing __sentry__ would sometimes cause errors on certain code paths.

4.1.115 Version 1.7.4

• Fixed a bug with potential recursion issues.
• Fixed a bug with the storage API and unicode keys.

4.1.116 Version 1.7.3

• Storage API has better responses when data fails to decode, or you send a bad request.
• Documentation improvements for JSON storage API.

4.1.117 Version 1.7.2

• All strings, lists, tuples, and sets are now shortened before sending to the server. Iterable data structures are truncated to the first 50 items, and strings are truncated to the first 200 characters.
  Both shorteners have configurable values in the settings.

4.1.118 Version 1.7.1

• Fixed a bug that slipped through with blocktrans usage.

4.1.119 Version 1.7.0

• Added score to GroupedMessage (schema change).
• Added MessageIndex (schema change).
• Added Async client (thanks to Yuri Baburov).
• Added support for raw_post_data (thanks to Matthew Schinckel).
• django-paging and django-indexer no longer need to be in INSTALLED_APPS.
• Added an index for GroupedMessages.times_seen.
• The score column will update atomically in PostgreSQL and MySQL.
• Added the frequency sort option.
• Better internationalization support.
• Fixed a bug with Oracle’s date truncation support (changed to hh24).
• Respect TIME_ZONE by using auto_now_* on DateTimeField’s.
• Tests required Haystack and Celery are now skipped if module is not found.
4.1.120 Version 1.6.10

- Added JSON support to storage API.
- Changed default client to use JSON format.

4.1.121 Version 1.6.9.1

- Fixed an issue with encoding to UTF-8 for Haystack.

4.1.122 Version 1.6.9

- Added URLs to default search parameters.
- Fixed a bug to ensure template information is only added if its the correct loader.

4.1.123 Version 1.6.8.1

- Search will now show when unauthenticated (e.g. when SENTRY_PUBLIC is enabled).

4.1.124 Version 1.6.8

- Search no longer allows filters.
- Search no longer allows arbitrary queries without Haystack.
- Added logger, level, site, server, and url to search index.
- Haystack has been updated for further flexibility.
- SearchFilter is now unused (you should update your configs).

4.1.125 Version 1.6.7

- Moved static media into /static/.
- Added serve_static view to handle static media solely within Sentry.
- Added SENTRY_STATIC_URL_PREFIX setting.

4.1.126 Version 1.6.6

- setup.py install will no longer install example_project.
- Fixed an issue where __sentry__ would be called even if it wasn’t a callable.
- Fixed an issue where transactions would attempt a rollback when not managed while creating the sort index.

4.1.127 Version 1.8.8

- Set a last_message_id so when thrashing is hit there is still a point of reference for tracing.
- Check correct permissions for Sentry.
4.1.128 Version 1.6.4

• Fixed a memory leak due to TextNode’s being created from leading whitespace in realtime packets.

4.1.129 Version 1.6.3

• Fixed a critical bug in the Sentry JS namespace preventing it from loading.

4.1.130 Version 1.6.2

• LogHandler will attempt to pick up the request automatically using SentryLogMiddleware.
• Updated AJAX CSRF support for Django 1.2.5.
• request.sentry is now set in any event which has request as part of the parameters.

4.1.131 Version 1.6.0

• Added message references (uuid’s) as message_id in Message
• Fixed css compatibility issues with TextWidget
• SearchFilter now allows searching by message reference id
• Added Sentry404CatchMiddleware
• Added SentryResponseErrorIdMiddleware
• The request argument can now be passed into any create_from_ method.

(History beyond 1.6.0 is not present)

4.2 License

Sentry is licensed under a three clause BSD License.
The full license text can be found below (Sentry License).

4.2.1 Authors

Sentry was originally written and is maintained by David Cramer.
A list of additional contributors can be seen on GitHub.

4.2.2 Sentry License

Copyright (c) 2014 David Cramer and individual contributors. All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the follow-
ing disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the Sentry nor the names of its contributors may be used to endorse or promote products
derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY
EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, IN-
CIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSI-
NESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CON-
TRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAM-
AGE.
CHAPTER 5

Resources

• Transifex (Translate Sentry!)
• Bug Tracker
• Code
• Mailing List
• IRC (irc.freenode.net, #sentry)
CHAPTER 6

Screenshots

6.1 Aggregated Events

6.2 Event Details
Deprecation Notes

Milestones releases are 1.3 or 1.4, and our deprecation policy is to a two version step. For example, a feature will be deprecated in 1.3, and completely removed in 1.4.
A
actions() (sentry.plugins.base.v1.IPlugin method), 45
add_project (built-in variable), 51
add_project_member (built-in variable), 51
B
before_events() (sentry.plugins.base.v1.IPlugin method), 45
C
can_enable_for_projects() (sentry.plugins.base.v1.IPlugin method), 45
can_enable_for_projects() (sentry.plugins.base.v2.IPlugin2 method), 48
cleanup (built-in variable), 25
create_event (built-in variable), 52
culprit (built-in variable), 38
D
delete() (sentry.options.manager.OptionsManager method), 57
E
edit_project (built-in variable), 51
edit_project_member (built-in variable), 51
event_id (built-in variable), 36
Exception (class in sentry.interfaces.exception), 54
extra (built-in variable), 38
G
get() (sentry.options.manager.OptionsManager method), 58
get_actions() (sentry.plugins.base.v2.IPlugin2 method), 48
get_annotations() (sentry.plugins.base.v2.IPlugin2 method), 49
get_conf_key() (sentry.plugins.base.v1.IPlugin method), 45
get_conf_key() (sentry.plugins.base.v2.IPlugin2 method), 49
get_conf_title() (sentry.plugins.base.v1.IPlugin method), 45
get_conf_title() (sentry.plugins.base.v2.IPlugin2 method), 49
get_description() (sentry.plugins.base.v1.IPlugin method), 45
get_description() (sentry.plugins.base.v2.IPlugin2 method), 49
get_event_preprocessors() (sentry.plugins.base.v2.IPlugin2 method), 49
get_feature_hooks() (sentry.plugins.base.v2.IPlugin2 method), 49
get_notification_forms() (sentry.plugins.base.v2.IPlugin2 method), 49
get_notifiers() (sentry.plugins.base.v2.IPlugin2 method), 49
get_option() (sentry.plugins.base.v1.IPlugin method), 45
get_option() (sentry.plugins.base.v2.IPlugin2 method), 50
get_release_hook() (sentry.plugins.base.v2.IPlugin2 method), 50
get_resource_links() (sentry.plugins.base.v1.IPlugin method), 46
get_resource_links() (sentry.plugins.base.v2.IPlugin2 method), 50
get_rules() (sentry.plugins.base.v2.IPlugin2 method), 50
get_tags() (sentry.plugins.base.v1.IPlugin method), 46
get_tags() (sentry.plugins.base.v2.IPlugin2 method), 50
get_title() (sentry.plugins.base.v1.IPlugin method), 46
get_title() (sentry.plugins.base.v2.IPlugin2 method), 50
get_url() (sentry.plugins.base.v1.IPlugin method), 46
H
has_perm() (sentry.plugins.base.v1.IPlugin method), 44
Http (class in sentry.interfaces.http), 56
I
IPlugin (class in sentry.plugins.base.v1), 44
IPlugin2 (class in sentry.plugins.base.v2), 48
is_enabled() (sentry.plugins.base.v1.IPlugin method), 46
is_enabled() (sentry.plugins.base.v2.IPlugin2 method), 50
is_regression() (sentry.plugins.base.v1.IPlugin method), 46
is_testable() (sentry.plugins.base.v1.IPlugin method), 47
level (built-in variable), 37
logger (built-in variable), 37
message (built-in variable), 37
Message (class in sentry.interfaces.message), 53
missing_perm_response() (sentry.plugins.base.v1.IPlugin method), 47
modules (built-in variable), 38

OptionsManager (class in sentry.options.manager), 57

panels() (sentry.plugins.base.v1.IPlugin method), 47
platform (built-in variable), 37
post_process() (sentry.plugins.base.v1.IPlugin method), 47

Query (class in sentry.interfaces.query), 56

redirect() (sentry.plugins.base.v2.IPlugin2 method), 50
release (built-in variable), 38
remove_project (built-in variable), 51
remove_project_member (built-in variable), 51
render() (sentry.plugins.base.v2.IPlugin2 method), 51
repair (built-in variable), 25

sentry_client (built-in variable), 39
sentry_key (built-in variable), 39
sentry_secret (built-in variable), 39
sentry_timestamp (built-in variable), 39
sentry_version (built-in variable), 39
server_name (built-in variable), 38
set() (sentry.options.manager.OptionsManager method), 58
set_option() (sentry.plugins.base.v1.IPlugin method), 47
set_option() (sentry.plugins.base.v2.IPlugin2 method), 51
Stacktrace (class in sentry.interfaces.stacktrace), 54

tags (built-in variable), 38
tags() (sentry.plugins.base.v1.IPlugin method), 48
Template (class in sentry.interfaces.template), 55