GottWall Scalable Realtime Statistics Aggregator Documentation

Release 0.2.10

GottWall team

October 10, 2015

Users Guide:

CHAPTER 1

Quickstart

Some basic prerequisites which you'll need in order to run GottWall:

- Python 2.6, or 2.7
- python-setuptools, python-dev
- Likely a UNIX-based operating system

1.1 Environment configuration

We recomended to install GottWall to separated 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:

```
easy_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/gottwall/`:

```
virtualenv /www/gottwall
```

Finally, activate your virtualenv:

```
source /www/gottwall/bin/activate
```

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

1.2 Installation

After environment activation install GottWall package to your env via `easy_install`:

```
easy_install -U gottwall
```

or `pip`:

```
pip install gottwall
```

After installation you can execute command in console `gottwall -h`, it's show gottwall manager documentation.

1.2.1 Installation storage backend

You can use different storages to save you data. We recommend to use `gottwall-storage-redis`:

```
easy_install -U gottwall-storage-redis
```

or via `pip`:

```
pip install gottwall-storage-redis
```

1.3 Configuration

Now you'll need to create the default configuration.

Execute `gottwall init config.py ~/.gottwall/gottwall.conf.py` or `examples/config.py` to your location (as example ~/.gottwall/gottwall.conf.py.)

```
STORAGE = 'gw_storage_redis.storage.RedisStorage'

BACKENDS = {
 }

TEMPLATE_DEBUG = True

STORAGE_SETTINGS = dict(
   HOST = 'localhost',
   PORT = 6379,
   PASSWORD = None,
   DB = 2
)

REDIS = {"CHANNEL": "gottwall"}


USERS = ["you@email.com"]

SECRET_KEY = "very secret key"



PROJECTS = {"test_project": "my_public_key",
            "another_project": "public_key2"}

cookie_secret="fkerwerwerwerw"

TEMPLATE_DEBUG = True

PREFIX = ""
```

1.4 Startings services

GottWall have 2 independent parts. Web interface application and aggregator application (application that process data).

To run web application execute command: `gottwall --config="examples/config.py" server start`

To run aggregator application execute command: `gottwall --config="examples/config.py" aggregator start`

Available Storages

Storage is a component of system that store calculated metrics data and performs calculation operations.

GottWall supports several storages in core package.

The following storages are supported current GottWall server:

- `gottwall.storages.memory.Memory` - stored metrics in memory
- `gottwall.storages.memory.Redis` - stored metrics in redis database

To use specified store need to setup `STORAGE` variable in GottWall config.

2.1 Storage development

Also you can develop custom storage for your own server. You need make package that included backend class inherited from `gottwall.storages.base.BaseBackend`.

Custom storage must override methods:

class `CustomStorage` (*gottwall.storages.base.BaseBackend*)

> **`incr(project, name, timestamp, value=1, filters={}, **kwargs):`**
> Add count for metric *name* and *filters*
>
> **`decr`**()
> Sub value from metric *name* in *project*
>
> **`slise_data`**()
> Get data by range and filters
>
> **`metrics`**()
> Get metrics list

2.2 Third party storages

CHAPTER 3

Available Clients

The following clients are officially recognized as production-ready, and support the current Sentry protocol:

- stati-redis (stati-redis-python) with redis transport.

3.1 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 GottWall 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 stati-<lang>-<transport-name>.

Developers:

CHAPTER 4

Contibuting

1. Check for open issues or open a fresh issue to start a discussion around a feature idea or a bug. There is a Contributor Friendly tag for issues that should be ideal for people who are not very familiar with the codebase yet.
2. Fork the repository on Github to start making your changes to the **develop** branch (or branch off of it).
3. Write a test which shows that the bug was fixed or that the feature works as expected.
4. Send a pull request and bug the maintainer until it gets merged and published.

4.1 Environment

We created environment vagrant kit for contributors.

It's named gottwall vagrant dev kit. You need to clone this repository to local system, initialize submodules and execute `vagrant up` in repository directory. This cookbooks configure virtual box node, installed needed services: postgresql, redis, rabbitmq.

4.2 Profiling

Stability and performance is a main priotitets. We working on its every day.

We use next utils to profile aplication:

cProfile

Most power tool to profile python applications.

1. To start profiling application need run next command:

```
    python -m cProfile -o profiling/gottwall_aggregator.prof gottwall/runner.py --config=examples/co
```

2. After you need to send data for aggregation via clients.
3. Next step need to analyze profiling results via pstats:

```
    python -m pstats profiling/gottwall_aggregator.prof
```

Also many helpful to use results map image.

To convert cProfile result to img need execute:

```
python tools/gprof2dot.py -f pstats profiling/gottwall_aggregator.prof | dot -Tpng -o profiling/aggre
```

plop

Another tool to profile python application.

To profile an entire Python script, run:

```
python -m plop.collector gottwall/runner.py --config=examples/config.py server start -h 0.0.0.0 --rel
```

This will write the profile to /tmp/plop.out

To use the viewer, run:

```
cp /tmp/plop.out ./profiles/*

python -m plop.viewer --datadir=profiles
```

and go to http://localhost:8888

CHAPTER 7

Writing a Client

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 GottWall server using some sort of authentication method.

The following items are expected of production-ready clients:

- DSN configuration
- Graceful failures

Additionally, the following features are highly encouraged:

- Non-blocking event submission

7.1 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)

```
var my_client = new RedisClient('http://public_key:secret_key@example.com/default');
```

or

::

var my_client = new RedisClient(private_key="private_key", public_key="public_key",
project="project_name", host="host")

2. Send data

```
my_client.incr(name="metric_name", value=2, timestamp=ts, filters={"status": "New", "user":"regi
```

CHAPTER 8

Indices and tables

- genindex
- modindex
- search