
gaffer Documentation

Release

Author

May 27, 2013

CONTENTS

1	Gaffer	1
1.1	Features	1
1.2	Contents:	2
2	Indices and tables	15

GAFFER

Control, Watch and Launch your applications and jobs over HTTP.

Gaffer is a set of Python modules and tools to easily maintain and interact with your applications or jobs launched on different machines over HTTP and websockets.

It promotes distributed and decentralized topologies without single points of failure, enabling fault tolerance and high availability.

1.1 Features

- RESTful HTTP Api
- Websockets and [SOCKJS](#) support to interact with a gaffer node from any browser or SOCKJS client.
- Framework to manage and interact your applications and jobs on different machines
- Server and `command-line` tools to manage and interact with your processes
- manages topology information. Clients query `gaffer_lookupd` to discover gaffer nodes for a specific job or application.
- Possibility to interact with STDIO and PIPES to interact with your applications and processes
- Subscribe to process statistics per process or process templates and get them in quasi RT.
- Procfile applications support (see `gaffer`) but also JSON config support.
- Supervisor-like features.
- Fully evented. Use the libuv event loop using the [pyuv library](#)
- Flapping: handle cases where your processes crash too much
- Easily extensible: add your own endpoint, create your client, embed gaffer in your application, ...
- Compatible with python 2.7x, 3.x

Note: gaffer source code is hosted on [Github](#)

1.2 Contents:

1.2.1 Getting started

This tutorial exposes the usage of gaffer as a tool. For a general overview or how to integrate it in your application you should read the `overview` page.

Introduction

Gaffer allows you to launch OS processes and supervise them. 3 `command line` tools allows you to use it for now:

- `gaffer` is the process supervisor and should be launched first before to use other tools.
- `gaffer` is a Procfile application manager and allows you to load your Procfile applications in `gaffer` and watch their status.
- `gafferctl` is a more generic tool than `gaffer` and is more admin oriented. It allows you to setup any process templates and manage your processes. You can also use it to watch the activity in `gaffer` (process activity or general activity)

A process template is the way you describe the launch of an OS process, how many you want to launch on startup, how many time you want to restart it in case of failures (flapping).... A process template can be loaded using any tool or on `gaffer` startup using its configuration file.

Workflow

To use gaffer tools you need to:

1. First launch `gaffer`
2. use either `gaffer` or `gafferctl` to manage your processes

Launch gaffer

For more informations of `gaffer` go on its `documentation` page .

To launch `gaffer` run the following command line:

```
$ gaffer -c /path/to/gaffer.ini
```

If you want to launch custom plugins with `gaffer` you can also set the path to them:

```
$ gaffer -c /path/to/gaffer.ini -p /path/to/plugin
```

Note: default plugin path is relative to the user launching `gaffer` and is set to `~/.gaffer/plugins`.

Note: To launch it in daemon mode use the `--daemon` option.

Then with the default configuration, you can check if `gaffer` is alive

The configuration file

The configuration file can be used to set the global configuration of gaffer, setup some processes and webhooks.

Note: Since the configuration is passed to the plugin you can also use this configuration file to setup your plugins.

Here is a simple example of a config to launch the dummy process from the example folder:

```
[process:dummy]
cmd = ./dummy.py
numprocesses = 1
redirect_output = stdout, stderr
```

Note: Process can be grouped. You can then start and stop all processes of a group and see if a process is member of a group using the HTTP api. (sadly this is not yet possible to do it using the command line).

For example if you want dummy be part of the group test, then `[process:dummy]` will become `[process:test:dummy]`. A process template as you can see can only be part of one group.

Groups are useful when you want to manage a configuration for one application or processes / users.

Each process section should be prefixed by *process:*. Possible parameters are:

- **cmd:** the full command line to launch. eg. `./dummy.p`
- **args:** arguments to pass as a string. eg. `-some value --option=a`
- **cwd:** path to working directory
- **uid:** user name or id used to execute the process
- **gid:** group name or id used to execute the process
- **detach:** if you want to completely detach the process from gaffer (gaffer will still continue to supervise it)
- **shell:** The process is executed in a shell (unix only)
- **flapping:** flapping rule. eg. 2, 1, 7, 5 which means attempts=2, window=1, retry_in=7, max_retry=5
- **redirect_input:** to allow you to interact with stdin
- **redirect_output:** to watch both stdout & stderr. output names can be whatever you want. For example you. eg. `redirect_output = mystdout, mystderr` stdout will be labelled *mystdout* in this case.
- **graceful_timeout:** time to wait before definitely kill a process. By default 30s. When killing a process, gaffer is first sending a SIGTERM signal then after a graceful timeout if the process hasn't stopped by itself send a SIGKILL signal. It allows you to handle the way your process will stop.
- **os_env:** true or false, to pass all operating system variables to the process environment.
- **priority:** Integer. Allows you to fix the order in which gaffer will start the processes. 0 is the highest priority. By default all processes have the same order.

Sometimes you also want to pass a custom environment to your process. This is done by creating a special configuration section named `env:processname`. Each environment sections are prefixed by `env:`. For example to pass a special PORT environment variable to dummy:

```
[env:dummy]
port = 80
```

All environment variables key are passed in uppercase to the process environment.

Manage your Procfile applications

The **gaffer** command line tool is an interface to the `gaffer HTTP api` and include support for loading/unloading Procfile applications, scaling them up and down,

It can also be used as a manager for Procfile-based applications similar to `foreman` but using the `gaffer` framework. It is running your application directly using a Procfile or export it to a `gaffer`d configuration file or simply to a JSON file that you could send to `gaffer`d using the `HTTP api`.

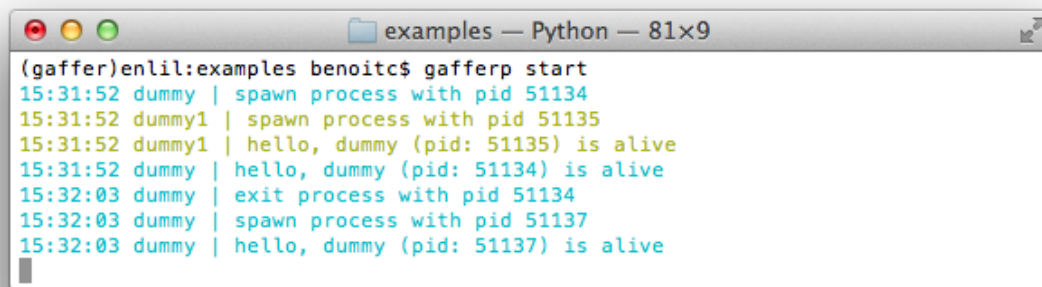
Example of use

For example using the following **Procfile**:

```
dummy: python -u dummy_basic.py
dummy1: python -u dummy_basic.py
```

You can launch all the programs in this procfile using the following command line:

```
$ gaffer start
```



```
(gaffer)enlil:examples benoitc$ gaffer start
15:31:52 dummy | spawn process with pid 51134
15:31:52 dummy1 | spawn process with pid 51135
15:31:52 dummy1 | hello, dummy (pid: 51135) is alive
15:31:52 dummy | hello, dummy (pid: 51134) is alive
15:32:03 dummy | exit process with pid 51134
15:32:03 dummy | spawn process with pid 51137
15:32:03 dummy | hello, dummy (pid: 51137) is alive
```

Or load them on a `gaffer` node:

```
$ gaffer load
```

All processes in the Procfile will be then loaded to `gaffer`d and started.

If you want to start a process with a specific environment file you can create a `.env` in the application folder (or use the command line option to tell to `gaffer` which one to use). Each environment variables are passed by lines. Ex:

```
PORT=80
```

and then scale them up and down:

```
$ gaffer scale dummy=3 dummy1+2
Scaling dummy processes... done, now running 3
Scaling dummy1 processes... done, now running 3
```



```
(gaffer)enlil:examples benoitc$ gaffer ps
== dummy: `python -u dummy_basic.py`
Total CPU: 1.800000 Total MEM: 0.30

dummy.1: up for 0:00.44
dummy.3: up for 0:00.40
dummy.4: up for 0:00.40
== dummy1: `python -u dummy_basic.py`
Total CPU: 1.700000 Total MEM: 0.40

dummy1.12: up for 0:00.38
dummy1.13: up for 0:00.36
dummy1.14: up for 0:00.37
dummy1.15: up for 0:00.36
(gaffer)enlil:examples benoitc$
```

have a look on the `gaffer` page for more informations about the commands.

Control gafferd with gafferctl

`gafferctl` can be used to run any command listed below. For example, you can get a list of all processes templates:

```
$ gafferctl processes
```

You can simply add a process using the `load` command:

```
$ gafferctl load_process ../test.json
$ cat ../test.json | gafferctl load_process -
$ gafferctl load_process - < ../test.json
```

`test.json` can be:

```
{
  "name": "somename",
  "cmd": "cmd to execute":
  "args": [],
  "env": {}
  "uid": int or "",
  "gid": int or "",
  "cwd": "working dir",
  "detach": False,
  "shell": False,
  "os_env": False,
  "numprocesses": 1
}
```

You can also add a process using the `add` command:

```
gafferctl add name inc
```

where `name` is the name of the process to create and `inc` the number of new OS processes to start.

To start a process run the following command:

```
$ gafferctl start name
```

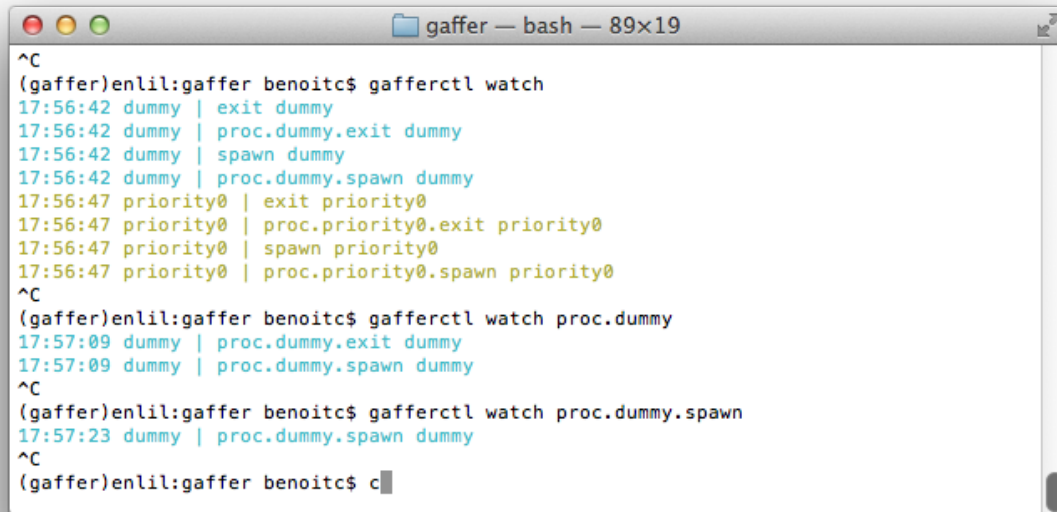
And stop it using the stop command.

To scale up a process use the add command. For example to increase the number of processes from 3:

```
$ gafferctl add name 3
```

To decrease the number of processes use the command stop/

The command watch allows you to watch changes in a local or remote gaffer node.



```
^C
(gaffer)enlil:gaffer benoitc$ gafferctl watch
17:56:42 dummy | exit dummy
17:56:42 dummy | proc.dummy.exit dummy
17:56:42 dummy | spawn dummy
17:56:42 dummy | proc.dummy.spawn dummy
17:56:47 priority0 | exit priority0
17:56:47 priority0 | proc.priority0.exit priority0
17:56:47 priority0 | spawn priority0
17:56:47 priority0 | proc.priority0.spawn priority0
^C
(gaffer)enlil:gaffer benoitc$ gafferctl watch proc.dummy
17:57:09 dummy | proc.dummy.exit dummy
17:57:09 dummy | proc.dummy.spawn dummy
^C
(gaffer)enlil:gaffer benoitc$ gafferctl watch proc.dummy.spawn
17:57:23 dummy | proc.dummy.spawn dummy
^C
(gaffer)enlil:gaffer benoitc$ c
```

For more informations go on the `gafferctl` page.

Demo

1.2.2 Gaffer REST API Resources

Jobs

Jobs are the configurations that can be used to launch the processes in Gaffer.

Processes

API to handle directly launched OS processes.

Auth

Authenticate to gaffer to get an authorization key. See the `../..//authenticate` documentation.

Keys

API to manage authorizations keys in a gaffer Node. The authorizations keys give certain rights to the users in gaffer. You need to be a node admin to access to this api.

Users

Miscellaneous

GET /auth

make an HTTP BASIC Auth request to get an authorization key

Resource URL <http://localhost:5000/auth>

Parameters None

GET <http://localhost:5000/auth>

- **200:** OK
- **401:** Unauthorized

GET /jobs

Get the list of all jobs configurations loaded on this node. A job describe how a process can be launched on the machine. Each jobs are prefixed by the session name: <session>.<jobname>.

Resource URL <http://localhost:5000/jobs>

Parameters None

GET <http://localhost:5000/jobs>

GET /<pid>

Get the information and status of an active process. The name is <sessionid>.<jobname>.

Resource URL <http://localhost:5000/pid>

Parameters None

GET <http://localhost:5000/1>

GET /pids

List all active processes.

Resource URL <http://localhost:5000/pids>

Parameters None

GET <http://localhost:5000/pids>

GET /sessions

Get the list of all resources loaded on this node. A resource is a collection of job configuration. It is generally corresponding to the list of jobs needed for an application.

Resource URL <http://localhost:5000/sessions>

Parameters None

GET <http://localhost:5000/sessions>

GET /jobs/<sessionid>

Get all jobs from a session

Resource URL <http://localhost:5000/jobs/sessionid>

Parameters None

GET <http://localhost:5000/jobs/procfile>

GET /jobs/<sessionid>/<jobname>

Get a job configuration

Resource URL <http://localhost:5000/jobs/sessionid/jobname>

Parameters None

GET <http://localhost:5000/jobs/procfile/dummy>

A process configuration has the following parameters:

- **name**: name of the process
- **cmd**: program command, string)
- **args**: the arguments for the command to run. Can be a list or a string.
- **env**: a mapping containing the environment variables the command will run with. Optional
- **uid**: int or str, user id
- **gid**: int or st, user group id,
- **cwd**: working dir

- **detach**: the process is launched but won't be monitored and won't exit when the manager is stopped.
- **shell**: boolean, run the script in a shell. (UNIX only)
- **redirect_output**: list of io to redirect (max 2) this is a list of custom labels to use for the redirection. Ex: ["a", "b"] will redirect stdout & stderr and stdout events will be labeled "a"
- **redirect_input**: Boolean (False is the default). Set it if you want to be able to write to stdin.
- **custom_streams**: list of additional streams that should be created and passed to process. This is a list of streams labels. They become available through `streams` attribute.
- **custom_channels**: list of additional channels that have been passed to process.

GET /jobs/<sessionid>/<jobname>/numprocesses

Get the number of processes that should be kept alive for a job configuration.

Resource URL <http://localhost:5000/jobs/sessionid/jobname/numprocesses>

Parameters None

GET <http://localhost:5000/jobs/procfile/dummy/numprocesses>

GET /jobs/<sessionid>/<jobname>/pids

Get all processes IDs running for a job configuration.

Resource URL <http://localhost:5000/jobs/sessionid/jobname/pids>

Parameters None

GET <http://localhost:5000/jobs/procfile/dummy/pids>

GET /jobs/<sessionid>/<jobname>/state

Get the status of a job configuration:

- **0**: stopped
- **1**: started

Resource URL <http://localhost:5000/jobs/sessionid/jobname/state>

Parameters None

GET <http://localhost:5000/jobs/procfile/dummy/state>

GET /jobs/<sessionid>/<jobname>/stats

Get statistics of all processes for a job configuration.

Resource URL <http://localhost:5000/jobs/sessionid/jobname>

Parameters None

GET <http://localhost:5000/jobs/procfile/dummy/stats>

GET [/<pid>/stats](#)

Get the current statistics for a process.

Resource URL <http://localhost:5000/pid/stats>

Parameters None

GET <http://localhost:5000/1/stats>

POST [/jobs/<sessionid>](#)

Load a job configuration.

Resource URL <http://localhost:5000/jobs/session>

Parameters None

POST <http://localhost:5000/jobs/test>

A process configuration has the following parameters:

- **name**: name of the process
- **cmd**: program command, string)
- **args**: the arguments for the command to run. Can be a list or a string.
- **env**: a mapping containing the environment variables the command will run with. Optional
- **uid**: int or str, user id
- **gid**: int or st, user group id,
- **cwd**: working dir
- **detach**: the process is launched but won't be monitored and won't exit when the manager is stopped.
- **shell**: boolean, run the script in a shell. (UNIX only)
- **redirect_output**: list of io to redict (max 2) this is a list of custom labels to use for the redirection. Ex: ["a", "b"] will redirect stdoutt & stderr and stdout events will be labeled "a"
- **redirect_input**: Boolean (False is the default). Set it if you want to be able to write to stdin.
- **custom_streams**: list of additional streams that should be created and passed to process. This is a list of streams labels. They become available through `streams` attribute.
- **custom_channels**: list of additional channels that have been passed to process.

POST /jobs/<sessionid>/<jobname>/commit

Send a one-off command to gaffer using a job configuration. Committed jobs aren't supervised, they will run until they die but won't be restarted. You can pass to the command a different environment and the graceful timeout. It return the ID of the process launched.

Resource URL <http://localhost:5000/jobs/session/job/commit>

Parameters None

POST `http://localhost:5000/jobs/test/dummy/commit`

POST /jobs/<sessionid>/<jobname>/numprocesses

Scale the number of processes launched for a job configuration. Values can be:

- **+N**: increase the number of processes from N
- **-N**: decrease the number of processes from N
- **=N**: set the number of processes to N

Resource URL <http://localhost:5000/jobs/session?job/numprocesses>

Parameters None

POST `http://localhost:5000/jobs/test/dummy/numprocesses`

POST /jobs/<sessionid>/<jobname>/signal

Send a signal to all processes related to this job configuration.

POSIX signal number (man signal or kill for more information):

No	Name	Default Action	Description
1	SIGHUP	terminate process	terminal line hangup
2	SIGINT	terminate process	interrupt program
3	SIGQUIT	create core image	quit program
4	SIGILL	create core image	illegal instruction
5	SIGTRAP	create core image	trace trap
6	SIGABRT	create core image	abort program (formerly SIGIOT)
7	SIGEMT	create core image	emulate instruction executed
8	SIGFPE	create core image	floating-point exception
9	SIGKILL	terminate process	kill program
10	SIGBUS	create core image	bus error
11	SIGSEGV	create core image	segmentation violation
12	SIGSYS	create core image	non-existent system call invoked
13	SIGPIPE	terminate process	write on a pipe with no reader
14	SIGALRM	terminate process	real-time timer expired
15	SIGTERM	terminate process	software termination signal
16	SIGURG	discard signal	urgent condition present on socket
17	SIGSTOP	stop process	stop (cannot be caught or ignored)
18	SIGTSTP	stop process	stop signal generated from keyboard
19	SIGCONT	discard signal	continue after stop

20	SIGCHLD	discard signal	child status has changed
21	SIGTTIN	stop process	background read attempted from control terminal
22	SIGTTOU	stop process	background write attempted to control terminal
23	SIGIO	discard signal	I/O is possible on a descriptor (see fcntl(2))
24	SIGXCPU	terminate process	cpu time limit exceeded (see setrlimit(2))
25	SIGXFSZ	terminate process	file size limit exceeded (see setrlimit(2))
26	SIGVTALRM	terminate process	virtual time alarm (see setitimer(2))
27	SIGPROF	terminate process	profiling timer alarm (see setitimer(2))
28	SIGWINCH	discard signal	Window size change
29	SIGINFO	discard signal	status request from keyboard
30	SIGUSR1	terminate process	User defined signal 1
31	SIGUSR2	terminate process	User defined signal 2

Resource URL <http://localhost:5000/jobs/session/job/signal>

Parameters None

POST <http://localhost:5000/jobs/test/dummy/signal>

POST [/jobs/<sessionid>/<jobname>/state](#)

Start, Stop or Reload all processes in a job configuration

- **0**: stop
- **1**: start
- **2**: reload

Resource URL <http://localhost:5000/jobs/session/job/state>

Parameters None

POST <http://localhost:5000/jobs/test/dummy/state>

POST [/<pid>/signal](#)

Send a signal to a process.

POSIX signal number (man signal or kill for more information):

No	Name	Default Action	Description
1	SIGHUP	terminate process	terminal line hangup
2	SIGINT	terminate process	interrupt program
3	SIGQUIT	create core image	quit program
4	SIGILL	create core image	illegal instruction
5	SIGTRAP	create core image	trace trap
6	SIGABRT	create core image	abort program (formerly SIGIOT)
7	SIGEMT	create core image	emulate instruction executed
8	SIGFPE	create core image	floating-point exception
9	SIGKILL	terminate process	kill program
10	SIGBUS	create core image	bus error
11	SIGSEGV	create core image	segmentation violation
12	SIGSYS	create core image	non-existent system call invoked

13	SIGPIPE	terminate process	write on a pipe with no reader
14	SIGALRM	terminate process	real-time timer expired
15	SIGTERM	terminate process	software termination signal
16	SIGURG	discard signal	urgent condition present on socket
17	SIGSTOP	stop process	stop (cannot be caught or ignored)
18	SIGTSTP	stop process	stop signal generated from keyboard
19	SIGCONT	discard signal	continue after stop
20	SIGCHLD	discard signal	child status has changed
21	SIGTTIN	stop process	background read attempted from control terminal
22	SIGTTOU	stop process	background write attempted to control terminal
23	SIGIO	discard signal	I/O is possible on a descriptor (see fcntl(2))
24	SIGXCPU	terminate process	cpu time limit exceeded (see setrlimit(2))
25	SIGXFSZ	terminate process	file size limit exceeded (see setrlimit(2))
26	SIGVTALRM	terminate process	virtual time alarm (see setitimer(2))
27	SIGPROF	terminate process	profiling timer alarm (see setitimer(2))
28	SIGWINCH	discard signal	Window size change
29	SIGINFO	discard signal	status request from keyboard
30	SIGUSR1	terminate process	User defined signal 1
31	SIGUSR2	terminate process	User defined signal 2

Resource URL <http://localhost:5000/pid/signal>

Parameters None

POST <http://localhost:5000/1/signal>

PUT [/jobs/<sessionid>/<jobname>](#)

Update a job configuration and relaunch processes

Resource URL <http://localhost:5000/jobs/session/job>

Parameters None

PUT <http://localhost:5000/jobs/test/dummy>

A process configuration has the following parameters:

- **name**: name of the process
- **cmd**: program command, string)
- **args**: the arguments for the command to run. Can be a list or a string.
- **env**: a mapping containing the environment variables the command will run with. Optional
- **uid**: int or str, user id
- **gid**: int or st, user group id,
- **cwd**: working dir
- **detach**: the process is launched but won't be monitored and won't exit when the manager is stopped.
- **shell**: boolean, run the script in a shell. (UNIX only)
- **redirect_output**: list of io to redict (max 2) this is a list of custom labels to use for the redirection. Ex: ["a", "b"] will redirect stdoutt & stderr and stdout events will be labeled "a"

- **redirect_input**: Boolean (False is the default). Set it if you want to be able to write to stdin.
- **custom_streams**: list of additional streams that should be created and passed to process. This is a list of streams labels. They become available through `streams` attribute.
- **custom_channels**: list of additional channels that have been passed to process.

DELETE `/<pid>`

Stop a process.

Resource URL `http://localhost:5000/pid`

Parameters None

DELETE `http://localhost:5000/1`

DELETE `/jobs/<sessionid>/<jobname>`

Unload a job configuration and stop all related processes

Resource URL `http://localhost:5000/jobs/session/job`

Parameters None

DELETE `http://localhost:5000/jobs/test/dummy`

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*