Django Correlation Id Documentation

Release 2.4.dev0

Contributors of django-cid

Aug 07, 2023

Contents

CHAPTER 1

Django Correlation ID

build unknown

Logging is important. Anyone who has had a call at 3am to say the site is down knows this. Without quality logging it is almost impossible to work out what on earth is happening.

The more you log, the harder it is to track down exactly what the effects of a particular request are. Enter Django Correlation ID. Incoming requests are assigned a unique identifier. This can either happen in your public facing web server (e.g. nginx) or be applied by Django itself.

This correlation id (also known as request id) is then available through the Django request/response cycle and may be automatically included in all log messages. That way, you can easily link all log messages that relate to the same request:

```
2018-10-01T08:18:39.86+00:00 correlation_id=2433d5d4-27a3-4889-b14b-107a131368a3 Call␣
↪to plug from cpoint=1
2018-10-01T08:18:39.90+00:00 correlation_id=72fbd7dd-a0ba-4f92-9ed0-0db358338e86 Call␣
↪to state by cpoint=2 with {'state': {'B': 'idle', 'A': 'on_charge'}}
2018-10-01T08:18:39.92+00:00 correlation_id=2433d5d4-27a3-4889-b14b-107a131368a3␣
↪Ended rental=7 customer="John Smith" on plug
```

In this example, we can see that the first and the third log messages are tied to the same request, while the second message relates to a distinct request.

In addition to these logs, `django-cid` can include the correlation id:

- in all SQL queries (as a comment);
- in rendered templates;
- as a header in the HTTP response generated by Django;
- and possibly anywhere by using the API of `django-cid`, for example as an HTTP header on a request to another internal system of yours, which is especially useful in service-oriented architecture.

Documentation can be found at: https://django-correlation-id.readthedocs.org/

Sources are on GitHub: https://github.com/Polyconseil/django-cid

1.1 Supported versions

We currently support Python >= 3.6 and Django >= 2.2.

Other versions may work but are not supported.

1.2 Topics

1.2.1 Installation and configuration

Installation

At the command line:

```
$ pip install django-cid
```

Configuration

You need to add `cid.apps.CidAppConfig` to your list of installed apps.

```
INSTALLED_APPS = (
    # some apps
    'cid.apps.CidAppConfig',
    # some other apps
)
```

Generation of the correlation id

The correlation id may be generated by `django-cid` itself or come from upstream through an incoming HTTP header.

To let `django-cid` generate an id, set `CID_GENERATE` to true in the settings:

```
CID_GENERATE = True
```

By default, `django-cid` uses `str(uuid.uuid4())` to generate the correlation id but you can customize this generation to suit your needs in the settings:

```
CID_GENERATOR = lambda: f'{time.time()}-{random.random()}'
```

Letting `django-cid` generate a new correlation id is perfectly acceptable but does suffer one drawback. If you host your Django application behind another web server such as nginx, then nginx logs won't contain the correlation id. `django-cid` can handle this by extracting a correlation id created in nginx and passed through as a header in the HTTP request. For this to work, you must enable a middleware in the settings, like this:

```
MIDDLEWARE = (
    'cid.middleware.CidMiddleware',
    # other middlewares
)
```

The middleware takes care of getting the correlation from the HTTP request header. By default it looks for a header named `X_CORRELATION_ID`, but you can change this with the `CID_HEADER` setting:

```
CID_HEADER = 'X_MY_CID_HEADER'
```

Note: Most WSGI implementations sanitize HTTP headers by appending an `HTTP_` prefix and replacing `-` by `_`. For example, an incoming `X-Correlation-Id` header would be available as `HTTP_X_CORRELATION_ID` in Django. When using such a WSGI server in front of Django, the latter, sanitized value should be used in the settings.

If a correlation id is provided upstream (e.g. "1234"), it is possible to concatenate it with a newly generated one. The cid will then look like `1234, 1aa38e4e-89c6-4655-9b8e-38ca349da017`. To do so, use the following settings:

```
CID_GENERATE = True
CID_CONCATENATE_IDS = True
```

This is useful when you use a service-oriented architecture and want to be able to follow a request amongst all systems (by looking at logs that have the first correlation id that was set upstream), and also on a particular system (by looking at logs that have the id added by the system itself).

Inclusion of the correlation id in the response

By default `django-cid` sets an HTTP header in the HTTP response with the same name as configured in `CID_HEADER`. You may customize it with `CID_RESPONSE_HEADER` in the settings:

```
CID_RESPONSE_HEADER = 'X-Something-Completely-Different'
```

Note: As indicated in the note above, if Django is behind a WSGI server that sanitizes HTTP headers, you need to customize `CID_RESPONSE_HEADER` if you want the same header name in the response as in the request.

```
# Nginx sets ``X-Correlation-Id`` but it is sanitized by the WSGI server.
CID_HEADER = 'HTTP_X_CORRELATION_ID'
# Don't use the default value (equal to CID_HEADER) for the response header.
CID_RESPONSE_HEADER = 'X-Correlation-Id'
```

If you don't want the header to appear in the HTTP response, you must explicitly set `CID_RESPONSE_HEADER` to `None`.

```
# Don't include the header in the HTTP response.
CID_RESPONSE_HEADER = None
```

Inclusion of the correlation id in logs

The most useful feature of `django-cid` is to include the correlation id in logs. For this you need to add the `cid.log.CidContextFilter` log filter in your log settings, apply it to each logger, and customize the formatter(s) to include the `cid` variable.

Here is what it looks like on the the default logging configuration provided by Django's `startproject`. Changed lines are highlighted.

```
LOGGING = {
    'version': 1,
```

(continues on next page)

(continued from previous page)

```
    'formatters': {
        'verbose': {
            'format': '[cid: %(cid)s] %(levelname)s %(asctime)s %(module)s %(message)s
↪'
        },
        'simple': {
            'format': '[cid: %(cid)s] %(levelname)s %(message)s'
        },
    },
    'handlers': {
        'console': {
            'level': 'INFO',
            'class': 'logging.StreamHandler',
            'formatter': 'verbose',
            'filters': ['correlation'],
        },
    },
    'filters': {
        'correlation': {
            '()': 'cid.log.CidContextFilter'
        },
    },
    'loggers': {
        'testapp': {
            'handlers': ['console'],
            'filters': ['correlation'],
            'propagate': True,
        },
    },
}
```

You can then use your loggers as you normally do, safe in the knowledge that you can tie them all back to the correlation id.

If you want to include the correlation id in all logs, you need to tweak the “root” key like this:

```
LOGGING = {
    # ...
    'root': {
        'level': 'INFO',
        'handlers': ['console'],
        'filters': ['correlation'],
    },
    # ...
}
```

Inclusion of the correlation id in SQL queries

`django-cid` can add the correlation id as a comment before the SQL query so that the correlation id appears in your database logs like this:

```
/* cid: 1234567-68e8-45fc-85c1-e025e5dffd1e */
SELECT col FROM table
```

For this you need to change your database backend to one that is provided by `django-cid`. For example, for sqlite3 you need to use the following:

```
DATABASES = {
    'default': {
        'ENGINE': 'cid.backends.sqlite3',
        'NAME': location('db.sqlite3'),
    }
}
```

`django-cid` has a wrapper for all backends that are currently supported by Django. Here is the full list:

mysql cid.backends.mysql

oracle cid.backends.oracle

postgis cid.backends.postgis

postgresql cid.backends.postgresql

sqlite3 cid.backends.sqlite3

By default, the correlation id appears as shown in the example above. You may change that by defining a `CID_SQL_COMMENT_TEMPLATE` that is a string with a `cid` format parameter:

```
CID_SQL_COMMENT_TEMPLATE = 'correlation={cid}'
```

Inclusion of the correlation id in templates

`django-cid` provides a template context processor that adds the correlation id to the template context if it is available. To enable it, you need to add it in the list of `TEMPLATE_CONTEXT_PROCESSORS` in the settings:

```
TEMPLATE_CONTEXT_PROCESSORS = (
    # other template processors
    'cid.context_processors.cid_context_processor',
)
```

It will add a context variable `correlation_id` if a correlation id is available. You may include it in your template with the follwing snippet:

```
{% if correlation_id %}
    <meta name="correlation_id" content="{{ correlation_id }}">
{% endif %}
```

1.2.2 API

`cid.locals.`**`generate_new_cid`**`(`*upstream_cid=None*`)`
    Generate a new correlation id, possibly based on the given one.

`cid.locals.`**`set_cid`**`(`*cid*`)`
    Set the correlation id on the context.

`cid.locals.`**`get_cid`**`()`
    Return the currently set correlation id (if any).

1.2.3 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of contributions

Report bugs

Report bugs at https://github.com/Polyconseil/django-cid/issues.

If you are reporting a bug, please include:

- the versions of `django-cid`, Django and Python;
- any details about your local setup that might be helpful in troubleshooting;
- detailed steps to reproduce the bug.

Write documentation

`django-cid` could always use more documentation. Don't hesitate to report typos or grammar correction.

Submit feedback

The best way to send feedback is to file an issue at https://github.com/Polyconseil/django-cid/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome. :)

Get started!

Ready to contribute? Here's how to set up `django-cid` for local development.

1. Fork the `django-cid` repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/django-cid.git
   ```

3. Set up a virtual environment and install the dependencies:

   ```
   $ pip install -e .
   $ pip install -r requirements/tests.txt
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

Now you can make your changes locally.

5. Test your changes locally by running `make test`.
5. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

Sandbox project

The repository has a `sandbox` directory that contains a Django project that showcases features and may help in testing and debugging. It does not replace automated tests, though.

Install `django-cid` and you can run the server:

```
$ cd sandbox
$ ./manage.py runserver
[...]
Starting development server at http://127.0.0.1:8000/
```

The home page at http://127.0.0.1:8000/ is self-documented.

Pull request guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated.
3. The pull request should work for all supported versions of Python and Django. Check https://travis-ci.org/Polyconseil/django-cid/pull_requests and make sure that the tests pass for all supported Python versions.

1.2.4 Releasing a new version

We use the excellent zest.releaser tool to make new releases. There is a Makefile rule that does a bit more cleaning beforehand. Just type:

```
make release
```

And then follow the instructions.

We try to use semantic versioning, i.e. use MAJOR.MINOR.PATCH version numbers with:

- MAJOR version when we make incompatible API changes;
- MINOR version when we add functionality in a backwards-compatible manner;
- PATCH version when we make backwards-compatible bug fixes.

Although the distinction between MINOR and PATCH has not always been followed, the changelog should be clear enough.

1.2.5 Credits

Original author: Jonathan Moss <jonathan.moss@snowballone.com.au>.

Current maintainers: the (mostly) nice people at Polyconseil.

Contributors:

- Francis Reyes <francis.reyes@snowballone.com.au>

1.2.6 History

2.4 (unreleased)

- Nothing changed yet.

2.3 (2022-06-13)

- Under Python 3.7 and later, use context variables (with the contextvars module) instead of a thread-local variable to avoid state bleeding in concurrent code.

2.2 (2021-03-15)

- Add support of Django 3.1.
- Remove support of Python 3.5.
- Under Python 3.7 and later, use context variables (with the contextvars module) instead of a thread-local variable to avoid state bleeding in concurrent code. Version 2.2 had a bug that caused context variables to never be used. Thread-local variables were always used.

2.1 (2020-06-22)

- Add support of Django 3.0
- Drop support of Django 2.1.

2.0 (2019-09-27)

- Drop support of Python 3.4.
- Drop support of Django 1.11 and Django 2.0.
- Add `CID_GENERATOR` setting to allow the customization of the correlation id.

1.3 (2018-10-09)

- **bugfix**: Fix packaging bug (introduced in version 1.2) that caused two extra packages `tests` and `sandbox` to be installed.

1.2 (2018-10-08)

- **bugfix:** Fix bug (introduced in version 1.0) that caused the correlation id to be reused across all requests that were processed by the same thread.

1.1 (2018-10-01)

- Allow to concatenate an upstream correlation id with a locally-generated one, with a new `CID_CONCATENATE_IDS` setting.

1.0 (2018-10-01)

Warning: this release includes changes that are not backward compatible. Be sure to read the details below to know if and how you can migrate.

- Drop support of Django 1.10 and earlier.
- Drop support of Python 2.
- Add support of Django 2. Version 0.x could already be used with Django 2 but tests were not run against it. They now are.
- Generate cid outside of the middleware when `GENERATE_CID` is enabled, so that it's available even if the middleware is not used.
- Fix support of Django 1.11 in database backends.
- Add PostGIS database backend.
- Add `CID_SQL_COMMENT_TEMPLATE` to customize how the cid is included as comments in SQL queries.
- Change the app name to be used in INSTALLED_APPS.

  Migration from version 0.x: if you had `cid` in `INSTALLED_APPS`, replace it by `cid.apps.CidAppConfig`. If you did not, add the latter.
- Drop compatibility with `MIDDLEWARE_CLASSES`. You should use the `MIDDLEWARE` setting. If you already did, no change is necessary.

  If you really must use the old `MIDDLEWARE_CLASSES` setting, include `CidOldStyleMiddleware` instead of `CidMiddleware`.

0.2.0 (2016-12-06)

- Added support for Django 1.10 middleware (thanks @qbey)

0.1.2 (2016-12-01)

- Made CID repsonse header configurable, and optional (thanks @dbaty)

0.1.0 (2014-08-05)

- First release on PyPI.

Python Module Index

Index