django-envy Documentation

Release 0.1.1

Michael Pedersen

Nov 05, 2017

Contents

Django-envy is an opinionated environment variable parser, with a focus on strictness, and doing a single thing well. Can be used standalone or with Django.

CHAPTER 1

Motivation

With the 12-factor app gaining popularity, and an increased focus on devops, there's already quite a few packages out there which can read and parse environment variables. However, django-envy makes a series of design decisions that I feel are important.

In the following, I will go over each decision in turn, and lay out the reasoning for it.

1.1 `.env` Files Are Not Environment Variables

The 12-factor app "manifesto" is very clear on favoring environment variables over language or OS specific config files for configuration, for a number of very good reasons.

While it's possible to interpret these reasons and conclusions in different ways, django-envy takes a very opinionated stand: **The application should know absolutely nothing about where environment variables come from.**

There are dozens of ways to set an environment variable so that it will be available to your application:

- Globally on startup for the entire instance
- For a specific user
- In your process manager, whether that be `startup`, `runit`, `supervisor`, `system.d`, `god`, `monit`, or something entirely different
- Using a wrapper script
- Built into your docker image
- When starting your docker image
- From a series of files using `envdir`
- Sourcing a bash script
- Etc.

All of these are acceptable options, and each have their pros and cons. With so many ways to set environment variables, there's no reason for the application to be able to set them too.

If you need to set environment variables in development, consider something like foreman or honcho, or the built in facilities for doing so in docker, if that's what you use.

1.2 Don't Duplicate Packages

There are a number of well-maintained packages to take care of Django-specific settings, such as database, cache or email settings:

- dj-database-url
- dj-email-url
- django-cache-url

However much the naming inconsistency annoys me, I don't see a good reason to duplicate the functionality of these packages, for the sake of fewer dependencies.

For an example of how to use any of these packages with django-envy, please see ...

1.3 Strictness And Correctnes Above Convenience

Configuration is important, and getting a vital piece of information wrong can be devastating. That's why django-envy takes a firm stance on what values are allowed when casting between types, and will always prefer raising an error to trying to be clever.

Examples:

- There is only two acceptable values when casting to boolean: `"true"` and `"false"` (they are case insensitive though)
- Floats must always be specified using a period (`.`) as the decimal separator. There is no logic for "guessing" the thousand separator. (Though it is possible to use _ for readability as in Python 3.6)
- If a cast does not seem to make sense, django-envy will throw an error. This includes trying to cast to nested collections.

1.4 Alternatives

If these design decisions aren't to your liking, there are other packages out there, which have chosen a different set of tradeoffs:

- django-environ
- envparse
- python-decouple
- django12factor
- django-confy
- json_environ

CHAPTER 2

Usage

2.1 Creating an Environment

2.2 Reading Environment Variables

2.3 Providing Defaults

2.4 Casting

2.4.1 Simple Types

2.4.2 Collections

2.4.3 Custom Types

2.4.4 Casting of the Default Value

2.5 Convenience Methods

2.6 Using With Other Django Settings Packages

CHAPTER 3

API

class envy.**Environment**(*environ*)

Class for reading and casting environment variables

This class presents the main interface for interacting with the environment. Once instantiated, it can either be called as a function, or any of the convenience methods can be used.

Parameters **environ** (*dict*) – Environment to read variables from

__call__(*var*, *default=<class 'envy.NoValue'>*, *cast=None*, *force=True*)

Function interface

Once the environment has been initialised, it can be called as a function. This is necessary to provide custom casting, or it can sometimes be preferred for consistency.

Examples

Casting an environment variable:

```
>>> env = Environment({'MY_VAR': '1'})
>>> env('MY_VAR', cast=int)
1
```

Providing a default:

```
>>> env = Environment({})
>>> env('ANOTHER_VAR', default='value')
"value"
```

Parameters

- **var** (*str*) – The name of the environment variable
- **default** – The value to return if the environment variable does not exist
- **cast** – type or function for casting environment variable. See casting

- **force** (*bool*) – Whether to force casting of the default value

Returns The environment variable if it exists, otherwise default

Raises `ImproperlyConfigured`

__contains__(*var*)
Test if an environment variable exists

Allows using the `in` operator to test if an environment variable exists.

Examples

```
>>> env = Environment({'MY_VAR': '1'})
>>> 'MY_VAR' in env
True
>>> 'ANOTHER_VAR' in env
False
```

bool(*var*, *default=<class 'envy.NoValue'>*, *force=True*)
Convenience method for casting to a bool

decimal(*var*, *default=<class 'envy.NoValue'>*, *force=True*)
Convenience method for casting to a decimal.Decimal

Note: Casting

---

dict(*var*, *default=<class 'envy.NoValue'>*, *cast=None*, *force=True*)
Convenience method for casting to a dict

Note: Casting

---

float(*var*, *default=<class 'envy.NoValue'>*, *force=True*)
Convenience method for casting to a float

int(*var*, *default=<class 'envy.NoValue'>*, *force=True*)
Convenience method for casting to an int

json(*var*, *default=<class 'envy.NoValue'>*, *force=True*)
Get environment variable, parsed as a json string

list(*var*, *default=<class 'envy.NoValue'>*, *cast=None*, *force=True*)
Convenience method for casting to a list

Note: Casting

---

set(*var*, *default=<class 'envy.NoValue'>*, *cast=None*, *force=True*)
Convenience method for casting to a set

Note: Casting

---

str(*var*, *default=<class 'envy.NoValue'>*, *force=True*)
Convenience method for casting to a str

tuple(*var*, *default=<class 'envy.NoValue'>*, *cast=None*, *force=True*)
Convenience method for casting to a tuple

Note: Casting

---

url(*var*, *default=<class 'envy.NoValue'>*, *force=True*)
Get environment variable, parsed with urlparse/urllib.parse

3.1 Exceptions

class envy.**ImproperlyConfigured**
Configuration Exception

Imported from Django if available, otherwise defined as a simple subclass of Exception

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index

Symbols

B

D

E

F

I

J

L

S

T

U