django_distributed_task Documentation

Release 1.0.0

Marc Riegel

August 25, 2014

Contents:

CHAPTER 1

Getting started

Step by step introduction.

1.1 Overview

In between of *celery <https://github.com/celery/celery>*, distributed_task is extremely lightweight. We'd decided to keep it simple with less of flexibility but straight at the needs. Just define for each method a *task*-method using the decorator and delay it at run time.

1.1.1 Use case

The goal is to prevent "heavy" tasks to be executed between a web request and it's response.

Examples for those tasks are:

- Sending e-mails.
- Generation of pdf/csv/... files.
- Rendering of images, videos.

1.2 Installation

To use distributed_task, a *Django <https://www.djangoproject.com/>* installation is required.

1.2.1 Requirements

It's well tested with following versions:

Version	Python	Django
1.0	2.7, 3.3, 3.4	1.5, 1.6, 1.7

1.2.2 Get the code

django_distributed_task package is available on `pip`:

```
pip install django_distributed_task
```

1.2.3 Register app in your Django settings.py

After install, register `distributed_task` to your `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    ...
    "distributed_task",
)
```

And finally `sync` your database:

```
./manage.py syncdb distributed_task
```

1.3 Settings

1.3.1 Message Broker

The default message broker `distributed_task.broker.backends.dummy.DummyMessageBroker` does not provide any functionality.

The only message broker which is tested and available is *RabbitMQ <http://www.rabbitmq.com/>*.

Sample setup for RabbitMQ:

```
DISTRIBUTED_TASK_BROKER = {
    'BROKER': 'distributed_task.broker.backends.amqp.AMQPMessageBroker',
    'OPTIONS': {
        # Your connection data for RabbitMQ
        'HOST': 'localhost',
        'USERNAME': 'guest',
        'PASSWORD': 'guest',
        'PORT': 5672,

        # Your desired queue name (need to change it for multiple installations)
        'QUEUE': 'distributed_task_queue',
    }
}
```

1.4 Usage

The default call scheme is: Task.delay -> Broker -> Worker -> Execution

1.4.1 Tasks

distributed_task will check in every installed app (`INSTALLED_APPS`) for a `tasks.py` file.

1.4.2 Define your first task

Create a `tasks.py` file in your desired app of choice:

```python
from distributed_task import register_task

@register_task
def my_heavy_task_method():
    pass
```

1.4.3 Call your task

The decorator adds a `delay` method to your task. You can decide in runtime if you'd like to execute the task delayed or immediately.

Execute delayed in a worker process:

```python
my_heavy_task_method.delay(*args, **kwargs)
```

Default method execution (bypasses task distribution):

```python
my_heavy_task_method(*args, **kwargs)
```

1.4.4 Arguments

You can pass all args/kwargs to the `my_heavy_task_method.delay` method as you would call it normally. The serializer is also able to handle Django model instances but not QuerySets.

This works fine:

```python
instance = User.objects.first()

my_heavy_task_method.delay('arg 1', user=instance, some_other_arg=False, some_float=12.5212)
```

1.4.5 Response / Return values

Method return values are not available. Maybe in a further version.

1.4.6 Start the worker

Finally, you need to start the worker process:: python manage.py run_worker

CHAPTER 2

Getting Started

- **Installation:** *Install*

CHAPTER 3

Indices and tables

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