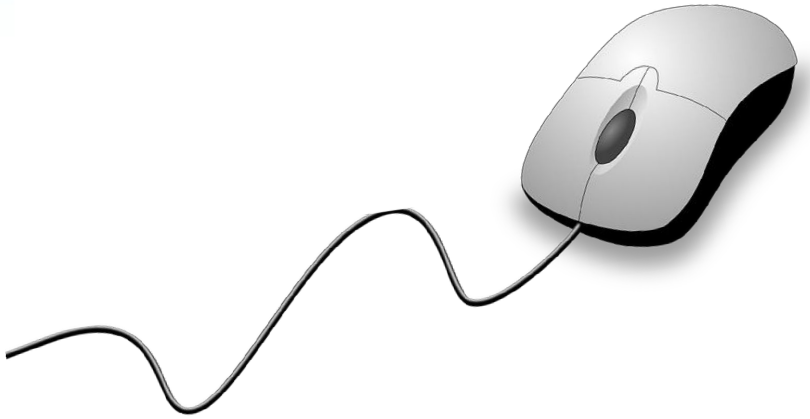
공개SW솔루션 설치 & 활용 가이드

미들웨어 > 분산시스템SW



CELERY 제대로 배워보자

How to Use Open Source Software

Open Source Software Installation & Application Guide

OPENUP

오픈소스 소프트웨어 통합지원센터

Open Source Software Support Center

CONTENTS

1. 개요

CELERY

소개	• 간단하고, 유연하며, 믿을 만한 방대한 양의 메시지를 처리하기 위한 분산 시스템 • 분산 시스템을 유지하기 위한 도구와 함께할 수 있는 운영방식을 제공 • 실시간 처리에 기반을 둔 Task 큐이며, 작업 스케줄링도 지원		
주요기능	• Task 큐 기능 지원		
대분류	• 미들웨어	**소분류**	• 분산시스템SW
라이선스형태	• 3-Clause BSD License	**사전설치 솔루션**	• Python
		버전	• 5.3.0b1(2022-12 최신)
특징	• 유지 보수 및 사용이 간편하며, 특정한 설정파일 불필요 • 워커와 클라이언트가 연결 상실 및 실패시 자동적으로 연결 재시도 • RabbitMQ, py-librabbitmq 및 최적화 된 설정 사용시 단일 셀러리 프로세스가 밀리초 미만의 왕복 지연 시간으로 분당 수백만 개의 작업 처리 가능		
보안취약점	• 취약점 ID : CVE-2021-23727 • 심각도 : Medium • 취약점 설명 : 공격자가 Celery result 백엔드 내의 메타데이터에 대한 액세스 권한을 얻거나 어떻게든 조작할 수 있으면 저장된 명령 주입 취약성을 발생시키고 잠재적으로 시스템에 대한 추가 액세스 권한을 얻을 수 있음 • 대응방안 : Celery를 5.2.2 이상 버전으로 업그레이드 • 참고 경로 : https://security.snyk.io/vuln/SNYK-PYTHON-CELERY-2314953		
개발회사 /커뮤니티	• https://groups.google.com/g/celery-users		
공식 홈페이지	• https://docs.celeryq.dev/en/stable/index.html		

1. 개요

- Celery는 사용하기 간편한 Task 큐임
- 해결해야 할 문제의 전체적인 것이나 복잡한 것을 학습하지 않고 시작할 수 있을 정도로 쉬움
- Celery를 사용한 프로그램이 타 언어로 확장 및 통합 가능하도록 모범 사례를 중심으로 설계됨
- 운영환경에서 이러한 시스템을 실행하는 데 필요한 도구와 지원이 함께 제공됨

2. 기능요약

- 기본적으로 제공되는 도구와 외부 도구를 통해 모니터링 가능
- 초/분/시간당 실행할 수 있는 Task 수 또는 Task 최대 실행가능 시간을 설정 가능
- Task를 실행할 시간을 초 또는 datetime 형식, crontab 표현식으로 사용 가능
- 리소스 릭 방지
- 각 워커의 구성요소 커스터마이징 가능
- 활용가능 분야
  - 시간이 오래 걸리는 작업을 비동기 처리하기 위해 사용가능

3. 실행 환경

- 메시지 전송
  - RabbitMQ, Redis, AmazonSQS
- 동시성
  - prefork, Eventlet, gevent, single threaded (solo)
- 결과 저장
  - AMQP, Redis
  - memcached
  - SQLAlchemy, Django ORM
  - Apache Cassandra, IronCache, Elasticsearch
- 직렬화
  - pickle, json, yaml, msgpack
  - zlib, bzip2 compression
  - Cryptographic message signing
- Microsoft windows를 지원하지 않음

4. 설치 및 실행

세부 목차

1. Broker 선택
2. Celery 설치
3. Celery 실행

4. 설치 및 실행

4.1 Broker 선택

- Celery는 메시지를 보내고 받기 위한 솔루션을 필요로 하는데, 이때 메시지 Broker라 불리는 분리된 서비스를 사용함

- RabbitMQ
  - 안정적이고 설치하기 쉬우며 프로덕션 환경을 위한 메시지 Broker
  - 도커를 이용하여 설치 및 실행

```
$ docker run -d -p 5672:5672 rabbitmq
Unable to find image 'rabbitmq:latest' locally
latest: Pulling from library/rabbitmq
```

- Redis
  - 전반적으로 속도가 빠르나 데이터 손실에 취약한 메시지 Broker
  - 도커를 이용하여 설치 및 실행

```
$ docker run -d -p 6379:6379 redis
Unable to find image 'redis:latest' locally
latest: Pulling from library/redis
```

4. 설치 및 실행

4.2 Celery 설치

- Celery 설치

```
$ pip install celery
Collecting celery
  Downloading celery-5.2.7-py3-none-any.whl (405 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 405.6/405.6 K
B 5.1 MB/s eta 0:00:00
```

- Redis를 Celery의 메시지 Broker로 사용하려면 추가적인 의존성 설치 필요

```
$ pip install -U "celery[redis]"
```

- tasks.py 파일에 코드 작성

```
from celery import Celery

app = Celery('tasks', broker='redis://localhost:6379/0')

@app.task
def add(x, y):
    return x + y
```

4. 설치 및 실행

4.3 Celery 실행

- Celery 실행 명령어

```
$ celery -A tasks worker --loglevel=INFO
```

- Celery 실행 결과

```
 -------------- celery@         -Latitude-3520 v5.2.7 (dawn-chorus)
--- ***** -----
-- ******* ---- Linux-5.15.0-56-generic-x86_64-with-glibc2.35 2022-12-21 10:03:04
- *** --- * ---
- ** ---------- [config]
- ** ---------- .> app:         tasks:0x7f43a3ee9030
- ** ---------- .> transport:   redis://localhost:6379/0
- ** ---------- .> results:     disabled://
- *** --- * --- .> concurrency: 8 (prefork)
-- ******* ---- .> task events: OFF (enable -E to monitor tasks in this worker)
--- ***** -----
 -------------- [queues]
                .> celery           exchange=celery(direct) key=celery


[tasks]
  . tasks.add

[2022-12-21 10:03:05,084: INFO/MainProcess] Connected to redis://localhost:6379/0
[2022-12-21 10:03:05,086: INFO/MainProcess] mingle: searching for neighbors
[2022-12-21 10:03:06,092: INFO/MainProcess] mingle: all alone
[2022-12-21 10:03:06,103: INFO/MainProcess] celery@tanks2438-Latitude-3520 ready.
```

5. 기능소개

세부 목차

1. Task 요청하기
2. Task의 상태 추적하기
3. Celery 설정

5. 기능소개

5.1 Task 요청하기

- delay라는 함수를 통해 Celery에 Task 실행 요청
- delay 함수는 apply_async 함수를 간편하게 사용할 수 있도록 만들어진 함수임

```
>>> from tasks import add
>>> add.delay(4, 4)
```

- Celery에 Task 실행을 요청하면 AsyncResult 인스턴스를 반환함
- 이 인스턴스는 Task의 상태, 리턴 값 등을 받는 것에 사용할 수 있음

```
<AsyncResult: 547a19d3-5aa1-4bcb-bf0c-6752b6241d59>
```

- worker의 콘솔 출력에서 Task가 실행되고 있는 것을 확인할 수 있음

```
[2022-12-21 10:31:09,282: INFO/MainProcess] Task tasks.add[547a19d3-5aa1-4bcb-bf0c-6752b6241d59] received
[2022-12-21 10:31:09,283: INFO/ForkPoolWorker-8] Task tasks.add[547a19d3-5aa1-4bcb-bf0c-6752b6241d59] succeeded in 0.00015120700118131936s: 8
```

5. 기능소개

5.2 Task의 상태 추적하기

CELERY

- Celery는 result 백엔드를 사용하여 Task의 상태 추적이 가능
- Celery는 여러 종류의 result 백엔드를 지원(예: Django ORM, MongoDB, Redis 등)
- Celery에 result 백엔드를 설정하려면 전달인자에 backend를 주거나 Celery 설정 모듈에서 result_backend 세팅 사용

```
app = Celery('tasks', backend='redis://', broker='redis://localhost:6379/0')
```

- result 백엔드를 설정해 놓으면 Celery에 Task 실행을 요청했을 때 리턴 값으로 받는 AsyncResult 인스턴스를 사용하여 Task의 상태 추적이 가능
  - ready 함수를 사용하면 Task의 실행이 종료 되었는지 확인가능
  - Task의 실행이 종료 되었으면 True 리턴

```
>>> a.ready()
True
```

  - AsyncResult.traceback을 사용하면 task에서 에러가 발생했을 시, 에러로그 확인가능

```
>>> a.traceback
'Traceback (most recent call last):\n
nv/lib/python3.10/site-packages/celery/
```

5. 기능소개

5.3 Celery 설정

- Celery는 여러 방법을 사용하여 기본 설정을 변경하는 것이 가능

  - 설정 1개를 변경하고 싶을 때

```
app.conf.task_serializer = 'json'
```

  - 설정 여러개를 변경하고 싶을 때

```
app.conf.update(
    task_serializer='json',
    accept_content=['json'],
    result_serializer='json',
    timezone='Europe/Oslo',
    enable_utc=True,
)
```

- 외부 설정 모듈을 사용하여 기본 설정을 변경하는 것도 가능
  - app.config_from_object 함수를 사용하여 설정 모듈 적용 가능

```
app.config_from_object('celeryconfig')
```

  - 설정 모듈 검증도 가능

```
$ python -m celeryconfig
```

```
broker_url = 'pyamqp://'
result_backend = 'rpc://'

task_serializer = 'json'
result_serializer = 'json'
accept_content = ['json']
timezone = 'Europe/Oslo'
enable_utc = True
```

<celeryconfig.py>
Celery 설정 파일 예

5. 기능소개

5.3.1 Celery 설정의 여러 세팅들

- broker_url
  - 기본값: “amqp://”
  - 형식: “transport://userid:password @hostname:port/virtual_host”
  - 설명: 기본 메시지 Broker를 설정하기 위한 세팅
- result_backend
  - 기본값: 없음
  - 형식: 기본적으로 문자열이나 어떤 서비스를 백엔드로 쓰느냐에 따라 형식이 다름
  - 설명: 기본 result 백엔드를 설정하기 위한 세팅
- task_serializer
  - 기본값: “json” (4.0 버전부터, 이전 버전은 “pickle”)
  - 형식: 사용할 직렬화 방법을 나타내는 문자열
  - 설명: JSON, pickle, YAML, msgpack, kombu.serialization.registry에 등록된 커스텀 serializer 중 하나를 직렬화 방법으로 설정하도록 만드는 세팅

5. 기능소개

5.3.1 Celery 설정의 여러 세팅들

- accept_content
  - 기본값: {'json'}
  - 형식: (set, list, or tuple)
  - 설명: 허용할 콘텐츠 타입들이나 serializers의 화이트 리스트를 설정하는 세팅
- timezone (2.5 버전 부터)
  - 기본값: "UTC"
  - 형식: pytz 라이브러리에서 지원하는 타임존의 문자열
  - 설명: Celery가 커스텀 타임존을 사용할 수 있도록 설정하는 세팅
- enable_utc (2.5 버전 부터)
  - 기본값: True(enabled) (버전 3.0 부터는 기본적으로 True(enabled))
  - 형식: True or False(Boolean)
  - 설명: 메시지에 날짜와 시간이 가능하게 되어 있다면 UTC 타임존을 사용하여 변경하는 것을 설정하는 세팅

6. 활용예제

6.1 Django에서 Celery를 사용해보기

6. 활용예제

6.1 Django에서 Celery를 사용해보기

- pip install django 명령을 통해 Django 설치

```
$ pip install django
```

- Django 설치 후 Project 및 앱 생성

```
/celery_ppt$ django-admin startproject mysite
```

```
/celery_ppt/mysite$ python manage.py startapp polls
```

- tree 명령어를 sudo apt install tree 명령어로 설치한 후 생성된 project 폴더(mysite) 에서 tree 명령어를 실행해 Django mysite 디렉토리 구조 확인 가능

```
$ tree
```

```
.
├── manage.py
├── mysite
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-310.pyc
│   │   └── settings.cpython-310.pyc
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── polls
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── migrations
    │   └── __init__.py
    ├── models.py
    ├── tests.py
    └── views.py

4 directories, 15 files
```

6. 활용예제

6.1 Django에서 Celery를 사용해보기

- polls 앱을 mysite/mysite/settings.py의 INSTALLED_APPS에 등록

```
INSTALLED_APPS = [
    'polls',
```

- pip install –U celery 명령을 통해 Celery 설치

```
$ pip install -U celery
```

- pip install “celery[redis]” 명령을 통해 redis를 메시지 Broker나 result 백엔드로 사용할 수 있게 해주는 번들 설치

```
$ pip install "celery[redis]"
```

- redis를 컨테이너로 실행한 다음 docker ps 명령어로 잘 실행되었는지 확인

```
$ docker run -d -p 6379:6379 redis
```

```
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS              PORTS                                       NAMES
6f366721eb3f   redis     "docker-entrypoint.s…"   About a minute ago   Up About a minute   0.0.0.0:6379->6379/tcp, :::6379->6379/tcp   nifty_zhukovsky
```

6. 활용예제

6.1 Django에서 Celery를 사용해보기

- mysite/mysite/celery.py 파일을 생성하고 아래의 내용을 작성

```
import os
from celery import Celery

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'mysite.settings')

app = Celery('mysite')
app.config_from_object('django.conf:settings', namespace='CELERY')
app.autodiscover_tasks()


@app.task(bind=True)
def debug_task(self):
    print(f'Request: {self.request!r}')
```

- os.environ… 라인은 Celery 명령줄 프로그램을 위해 DJANGO_SETTINGS_MODULE 환경변수를 설정
- app = Celery… 라인은 Celery 라이브러리의 인스턴스를 생성
- app.config_from… 라인은 Celery의 설정을 위해 Django 세팅 모듈을 사용
- app.autodiscover… 라인은 생성한 모든 앱에 있는 tasks.py 파일에서 task를 찾음

6. 활용예제

6.1 Django에서 Celery를 사용해보기

- Django를 실행할 때 Celery 라이브러리의 인스턴스(app)을 항상 import 하기위해 mysite/mysite/init.py 파일에 아래의 내용을 작성

```
from .celery import app as celery_app

__all__ = ('celery_app',)
```

- mysite/mysite/settings.py에 아래의 내용을 작성하여 result 백엔드와 메시지 Broker 설정

```
# celery
CELERY_RESULT_BACKEND = 'redis://localhost:6379/0'
CELERY_BROKER_URL = 'redis://localhost:6379/0'
```

- mysite/polls/tasks.py를 생성하고 아래 내용을 작성하여 Task를 만듦
  이때 @shared_task를 사용하여 이전에 생성한 Celery 라이브러리의 인스턴스를 사용하지 않고 Task를 만들 수 있음

```
from celery import shared_task


@shared_task
def add(x, y):
    return x + y
```

```
@shared_task
def mul(x, y):
    return x * y


@shared_task
def xsum(numbers):
    return sum(numbers)
```

6. 활용예제

6.1 Django에서 Celery를 사용해보기

- celery –A mysite worker –l INFO 명령어를 mysite/ 에서 실행

```
$ celery -A mysite worker -l INFO
```

username

```
 -------------- celery@          -Latitude-3520 v5.2.7 (dawn-chorus)
--- ***** -----
-- ******* ---- Linux-5.15.0-56-generic-x86_64-with-glibc2.35 2022-12-21 05:22:26
- *** --- * ---
- ** ---------- [config]
- ** ---------- .> app:          mysite:0x7fb15aaa73a0
- ** ---------- .> transport:    redis://localhost:6379/0
- ** ---------- .> results:      redis://localhost:6379/0
- *** --- * --- .> concurrency: 8 (prefork)
-- ******* ---- .> task events: OFF (enable -E to monitor tasks in this worker)
--- ***** -----
 -------------- [queues]
                .> celery           exchange=celery(direct) key=celery


[tasks]
  . mysite.celery.debug_task
  . polls.tasks.add
  . polls.tasks.mul
  . polls.tasks.xsum
```

6. 활용예제

6.1 Django에서 Celery를 사용해보기

- 새로운 shell을 열고 python ./manage.py shell 명령어를 mysite/ 에서 실행

```
/mysite$ python ./manage.py shell
```

- Celery에 delay 함수로 Task 실행 요청 및 결과 확인

```
>>> from polls.tasks import add, mul, xsum
>>> res = add.delay(2,3)
>>> res.result
5
```

- Worker의 콘솔 출력 확인

```
[2022-12-21 05:26:38,972: INFO/MainProcess] Task polls.tasks.add[89710013-1326-4c06-9bd7-b9793abebf63] received
```

```
[2022-12-21 05:26:38,973: INFO/ForkPoolWorker-8] Task polls.tasks.add[89710013-1326-4c06-9bd7-b9793abebf63] succeeded in 0.0006346670015773275: 5
```

Q **Celery는 어떤 때에 주로 사용하는 것이 좋을까요?**

A Celery는 시간이 오래 걸리며 비동기적으로 처리해도 되는 작업을 수행해야할 때 사용하는 것이 좋습니다.

Q **Celery를 위한 메시지 Broker로 무엇을 사용하면 좋을까요?**

A 주로 작은 메시지를 빠르게 전송해야 한다면 Redis를, 큰 메시지를 전송해야한다면 RabbitMQ를 사용하는 것이 좋으나 본인의 환경에 따라 어떤 Broker가 좋은지는 달라질 수 있으므로 공식 문서의 관련 내용을 보며 본인의 환경에 가장 잘 맞을 것 같은 Broker를 사용하시는 것이 좋습니다.

Q 대기 중인 모든 Task를 정리하려면 어떻게 해야 하나요?

A celery purge 명령을 사용하여 설정된 모든 작업 대기열을 정리할 수 있습니다.

```
$ celery -A proj purge
```

이 작업은 프로그래밍적인 방법을 통해서도 수행이 가능합니다.

```
>>> from proj.celery import app
>>> app.control.purge()
```

Q 메시지를 암호화 할 수 있나요?

A 일부 AMQP Broker는 SSL의 사용(RabbitMQ 포함)을 지원합니다.
broker_use_ssl 설정을 사용하여 이 기능을 사용하도록 설정할 수 있습니다.
또한 메시지에 암호화 및 보안을 추가할 수도 있습니다.
이를 위해서는 Celery의 메일링 리스트를 통해 연락을 하는 것이 필요합니다.

8. 용어정리

용어	설 명
Task	Celery application의 블록으로 호출가능한 함수등의 객체를 통해 만들어질수 있는 클래스
Worker	요청받은 Task를 실행하는 일종의 프로세스
테스크 큐	스레드 또는 시스템 간에 작업을 분배하는 메커니즘으로 테스크 큐의 입력은 Task라고 부르는 작업의 단위이며 전용 Worker는 수행할 새 작업이 있는지 대기열을 지속적으로 모니터링
AMQP	시스템 연결 및 비즈니스 프로세스에 필요한 정보를 제공하고, 목표를 달성하는 지침을 안정적으로 전달할 수 있으며, 응용 프로그램 또는 조직간에 비즈니스 메시지를 전달하기 위한 개방형 표준 응용 계층 프로토콜
SSL	보안 소켓 계층(Secure Socket Layer)라고도 불리며, 웹 서버와 브라우저 사이나 두 서버사이에 전송되는 데이터를 암호화하여 컴퓨터 네트워크에 통신 보안을 제공하기 위해 설계된 암호 규약

Open Source Software Installation & Application Guide