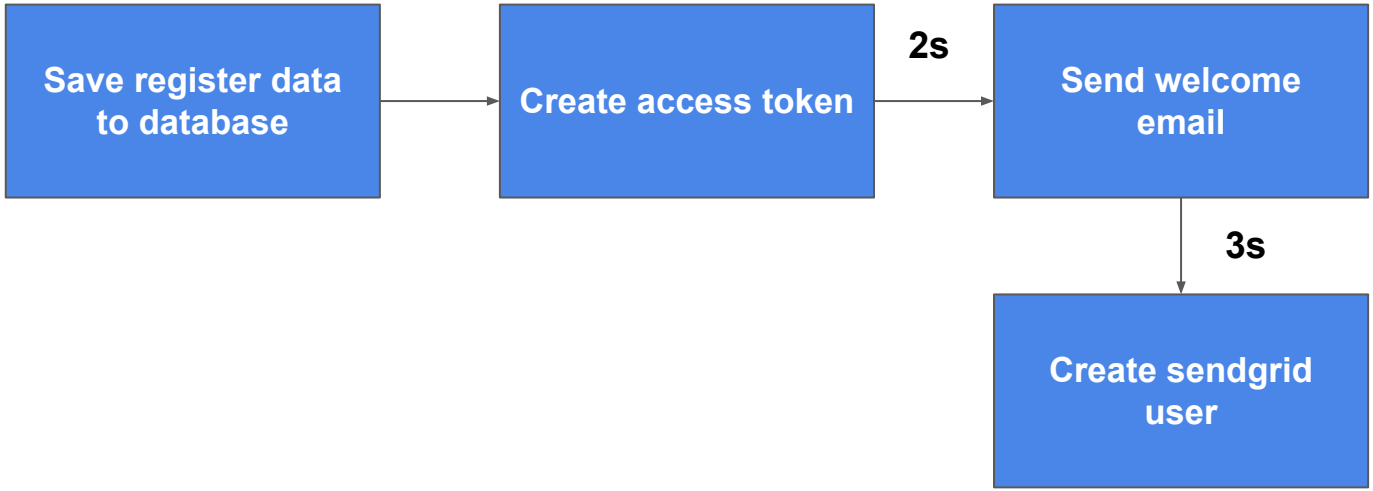
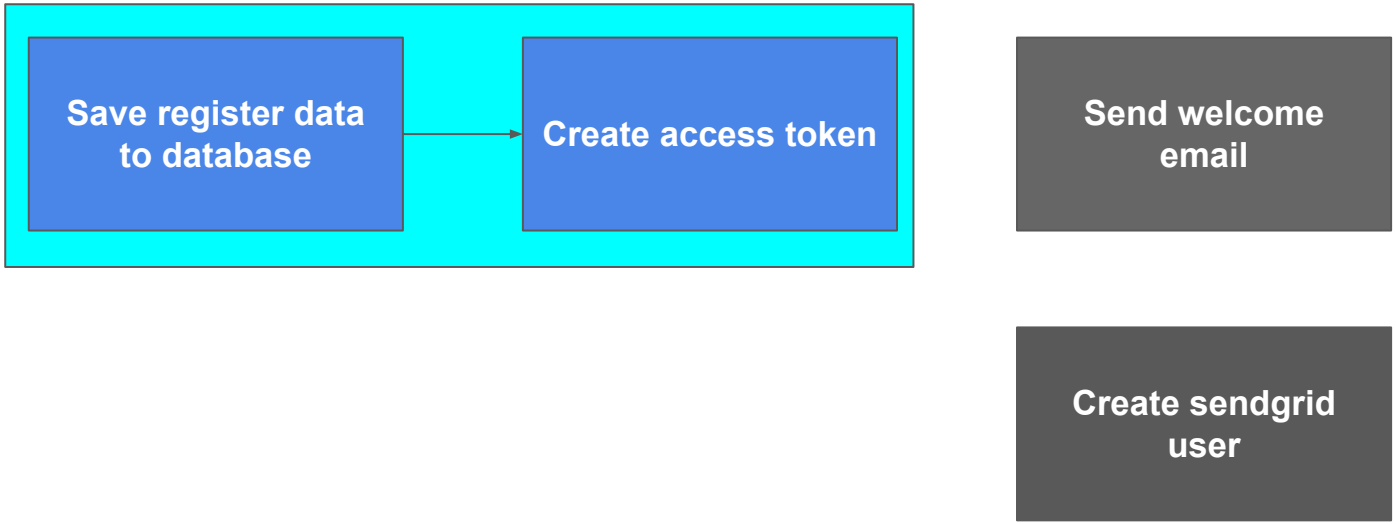
CELERY

Feature register



Time consumed: 5.5s

Feature register

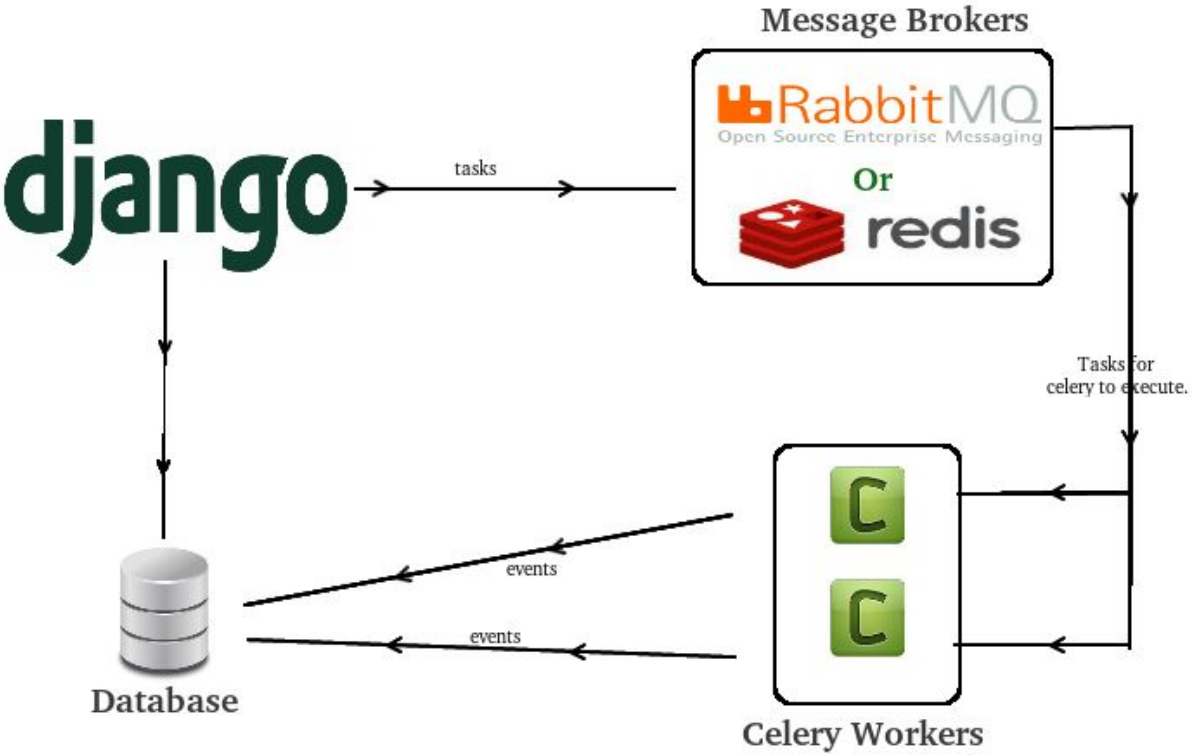


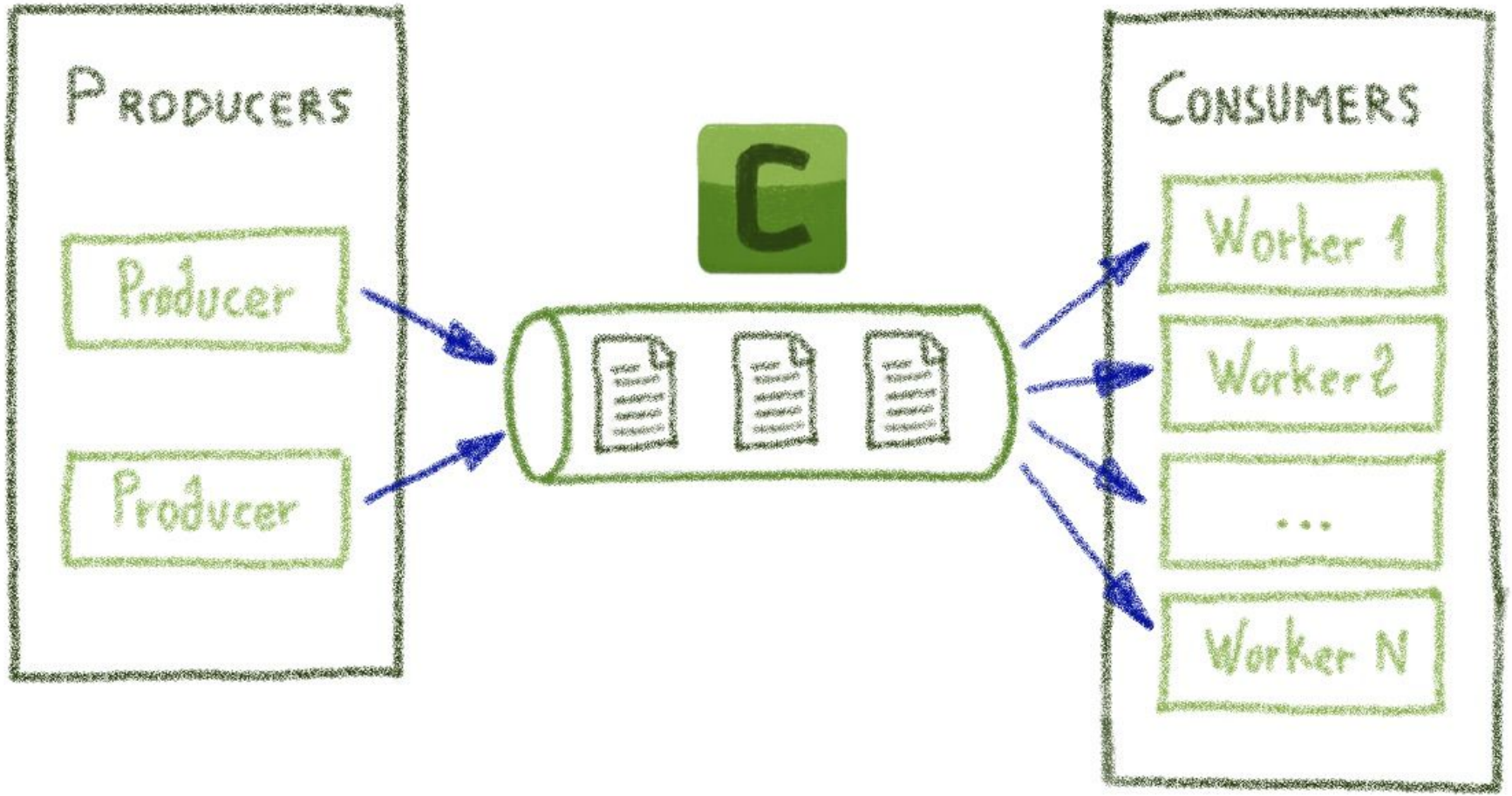
Time consumed: 0.5s

Celery - Distributed Task Queue

Celery is a simple, flexible, and reliable distributed system to process vast amounts of messages, while providing operations with the tools required to maintain such a system.

It's a task queue with focus on real-time processing, while also supporting task scheduling.


Message Brokers
RabbitMQ
Open Source Enterprise Messaging
Or
redis
tasks
Tasks for
celery to execute.
events
events
Database
Celery Workers


Producers
Producer
Producer
C
Consumers
Worker 1
Worker 2
...
Worker N

- Choosing a Broker
  - RabbitMQ
  - Redis
  - Other brokers
- Installing Celery
- Application
- Running the Celery worker server
- Calling the task
- Keeping Results
- Configuration
- Where to go from here
- Troubleshooting
  - Worker doesn't start: Permission Error
  - Result backend doesn't work or tasks are always in `PENDING` state

Choosing a Broker

Celery requires a solution to send and receive messages; usually this comes in the form of a separate service called a *message broker*.

RabbitMQ

RabbitMQ is feature-complete, stable, durable and easy to install. It's an excellent choice for a production environment. Detailed information about using RabbitMQ with Celery:

> Using RabbitMQ

If you're using Ubuntu or Debian install RabbitMQ by executing this command:

```
$ sudo apt-get install rabbitmq-server
```

Or, if you want to run it on Docker execute this:

```
$ docker run -d -p 5672:5672 rabbitmq
```

When the command completes, the broker will already be running in the background, ready to move messages for you: `Starting rabbitmq-server: SUCCESS`.

Redis

Redis is also feature-complete, but is more susceptible to data loss in the event of abrupt termination or power failures. Detailed information about using Redis:

Using Redis

If you want to run it on Docker execute this:

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

What we need

- Pipenv
- Application -> Django
- Messaging Queue
- Celery

Install & Run RabbitMQ

```
docker run -p 5672:5672 --name "celery-rabbit-mq" rabbitmq
```

docker run -p 5672:5672 --name "celery-rabbit-mq" rabbitmq

Create Django Project

```
mkdir celery-workshop
pipenv shell
pip install django
pip install celery
django-admin startproject calculator .
touch calculator/celery.py
```

mkdir celery-workshop
pipenv shell
pip install django
pip install celery
django-admin startproject calculator .
touch calculator/celery.py

Setup Celery Instance

calculator/celery.py

```python
import os

from celery import Celery

# Set the default Django settings module for the 'celery' program.
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'calculator.settings')

app = Celery('calculator')

# Using a string here means the worker doesn't have to serialize
# the configuration object to child processes.
# - namespace='CELERY' means all celery-related configuration keys
#   should have a `CELERY_` prefix.
app.config_from_object('django.conf:settings', namespace='CELERY')

# Load task modules from all registered Django apps.
app.autodiscover_tasks()


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

Always run celery when application start

calculator/__init__.py

```
from .celery import app as celery_app

__all__ = ('celery_app',)
```

Create New App

```
python manage.py startapp calculator_app
```

Create Task

calculator/tasks.py

```python
from celery import shared_task


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


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

Calling a Task

- Basics
- Linking (callbacks/errbacks)
- On message
- ETA and Countdown
- Expiration
- Message Sending Retry
- Connection Error Handling
- Serializers
- Compression
- Connections
- Routing options
- Results options

Calling Tasks — Celery 5.1.0 documentation (celeryproject.org)

Quick Cheat Sheet

- `T.delay(arg, kwarg=value)`

  Star arguments shortcut to `.apply_async`. (`.delay(*args, **kwargs)` calls `.apply_async(args, kwargs)`).
- `T.apply_async((arg,), {'kwarg': value})`
- `T.apply_async(countdown=10)`

  executes in 10 seconds from now.
- `T.apply_async(eta=now + timedelta(seconds=10))`

  executes in 10 seconds from now, specified using `eta`
- `T.apply_async(countdown=60, expires=120)`

  executes in one minute from now, but expires after 2 minutes.
- `T.apply_async(expires=now + timedelta(days=2))`

  expires in 2 days, set using **`datetime`**.

Run command

- celery -A calculator worker -l INFO
- python manage.py runserver

Monitoring

Install Flower

```
pip install flower
```

Monitoring and Management Guide — Celery 5.1.0 documentation (celeryproject.org)

Start flower

```
flower -A calculator
or
celery flower -A calculator
```

Installation — Flower 1.0.0 documentation

Let try

- Start flower
- Run any task
- See the result
- Stop flower
- Start flower

persistent

Enable persistent mode. If the persistent mode is enabled Flower saves the current state and reloads on restart (by default, *persistent=False*)

Start flower with persistent flag

```
flower -A calculator --persistent=True
```

Let try

- Start flower
- Run any task
- See the result
- Stop flower
- Start flower

First steps with Django — Celery 5.1.0 documentation (celeryproject.org)
Introducing Director – a tool to build your Celery workflows | OVHcloud Blog
celery/examples/django at master · celery/celery (github.com)
rabbitmq - celery flower does not show previously run tasks after restart - Stack Overflow