Asynchronous Tasks With Django and Celery

by Real Python

63 Comments advanced databases django web-dev

Table of Contents

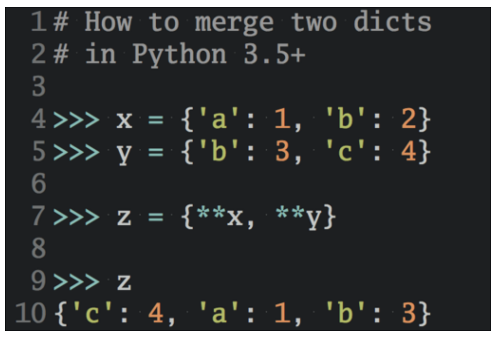
When I was new to Django, one of the most frustrating things I experienced was the need to run a bit of code periodically. I wrote a nice function that performed an action that needed to run daily at 12am. Easy, right? Wrong. This turned out to be a huge problem to me since at the time I was used to "Cpanel-type" web hosting where there was a nice handy GUI for setting up cron jobs for this very purpose.

After much research, I found a nice solution—Celery, a powerful asynchronous job queue used for running tasks in the background. But this led to additional problems, since I couldn't find an easy set of instructions to integrate Celery into a Django Project.

Of course I eventually did manage to figure it—which is what this article will cover: **How to integrate Celery into a Django Project and create Periodic Tasks.**

Free Bonus: Click here to get access to a free Django Learning Resources Guide (PDF) that shows you tips and tricks as well as common pitfalls to avoid when building Python + Django web applications.

This project utilizes Python 3.4, Django 1.8.2, Celery 3.1.18, and Redis 3.0.2.



```
# How to merge two dicts
# in Python 3.5+

>>> x = {'a': 1, 'b': 2}
>>> y = {'b': 3, 'c': 4}

>>> z = {**x, **y}

>>> z
{'c': 4, 'a': 1, 'b': 3}
```

Overview

For your convenience, since this is such a large post, please refer back to this table for brief info on each step and to grab the associated code.

Step	Overview	Git Tag
Boilerplate	Download boilerplate	v1
Setup	Integrate Celery with Django	v2
Celery Tasks	Add basic Celery Task	v3
Periodic Tasks	Add Periodic Task	v4
Running Locally	Run our app locally	v5
Running Remotely	Run our app remotely	v6

What is Celery?

"Celery is an asynchronous task queue/job queue based on distributed message passing. It is focused on real-time operation, but supports scheduling as well." For this post, we will focus on the scheduling feature to periodically run a job/task.

Why is this useful?

- Think of all the times you have had to run a certain task in the future. Perhaps you needed to access an API every hour. Or maybe you needed to send a batch of emails at the end of the day. Large or small, Celery makes scheduling such periodic tasks easy.
- You never want end users to have to wait unnecessarily for pages to load or actions to complete. If a long process is part of your application's workflow, you can use Celery to execute that process in the background, as resources become available, so that your application can continue to respond to client requests. This keeps the task out of the application's context.

Setup

Before diving into Celery, grab the starter project from the Github repo. Make sure to activate a virtualenv, install the requirements, and run the migrations. Then fire up the server and navigate to http://localhost:8000/ in your browser. You should see the familiar “Congratulations on your first Django-powered page” text. When done, kill the server.

Next, let’s install Celery using pip:

Shell

```
$ pip install celery==3.1.18
$ pip freeze > requirements.txt
```

Now we can integrate Celery into our Djang

Step 1: Add *celery.py*

Inside the “picha” directory, create a new f

Python

```
from __future__ import absolute_imp
import os
from celery import Celery
from django.conf import settings

# set the default Django settings module for the 'celery' program.
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'picha.settings')
app = Celery('picha')

# Using a string here means the worker will not have to
# pickle the object when using Windows.
app.config_from_object('django.conf:settings')
app.autodiscover_tasks(lambda: settings.INSTALLED_APPS)


@app.task(bind=True)
def debug_task(self):
    print('Request: {0!r}'.format(self.request))
```

Take note of the comments in the code.

Step 2: Import your new Celery app

To ensure that the Celery app is loaded when Django starts, add the following code into the *__init__.py* file that sits next to your *settings.py* file:

Python

```
from __future__ import absolute_import

# This will make sure the app is always imported when
# Django starts so that shared_task will use this app.
from .celery import app as celery_app
```

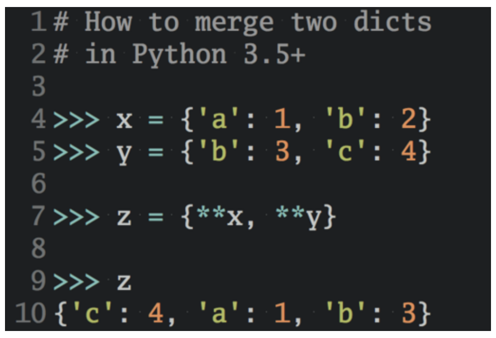
Having done that, your project layout should now look like:

```
├── manage.py
├── picha
│   ├── __init__.py
│   ├── celery.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── requirements.txt
```

Step 3: Install Redis as a Cel

Celery uses “brokers” to pass messages be
Redis as the message broker.

First, install Redis from the official downlo
in a new terminal window, fire up the serve


```
# How to merge two dicts
# in Python 3.5+

>>> x = {'a': 1, 'b': 2}
>>> y = {'b': 3, 'c': 4}

>>> z = {**x, **y}

>>> z
{'c': 4, 'a': 1, 'b': 3}
```

Shell

```
$ redis-server
```

You can test that Redis is working properly by typing this into your terminal:

Shell

```
$ redis-cli ping
```

Redis should reply with `PONG` - try it!

Once Redis is up, add the following code to your settings.py file:

Python

```python
# CELERY STUFF
BROKER_URL = 'redis://localhost:6379'
CELERY_RESULT_BACKEND = 'redis://localhost:6379'
CELERY_ACCEPT_CONTENT = ['application/json']
CELERY_TASK_SERIALIZER = 'json'
CELERY_RESULT_SERIALIZER = 'json'
CELERY_TIMEZONE = 'Africa/Nairobi'
```

You also need to add Redis as a dependency in the Django Project:

Shell

```
$ pip install redis==2.10.3
$ pip freeze > requirements.txt
```

That’s it! You should now be able to use Celery with Django. For more information on setting up Celery with Django, please check out the official Celery documentation.

Before moving on, let’s run a few sanity checks to ensure all is well…

Test that the Celery worker is ready to receive tasks:

Shell

```
$ celery -A picha worker -l info
...
[2015-07-07 14:07:07,398: INFO/MainProcess] Connected to redis://localhost:6379//
[2015-07-07 14:07:07,410: INFO/MainProcess] mingle: searching for neighbors
[2015-07-07 14:07:08,419: INFO/MainProcess] mingle: all alone
```

Kill the process with CTRL-C. Now, test that the Celery task scheduler is ready for action:

Shell
```
$ celery -A picha beat -l info
...
[2015-07-07 14:08:23,054: INFO/MainProcess] beat: Starting...
```

Boom!

Again, kill the process when done.

Celery Tasks

Celery utilizes tasks, which can be thought

For example, let’s turn this basic function i

Python
```python
def add(x, y):
    return x + y
```

First, add a decorator:

Python
```python
from celery.decorators import task

@task(name="sum_two_numbers")
def add(x, y):
    return x + y
```

Then you can run this task asynchronously with Celery like so:

Python
```python
add.delay(7, 8)
```

Simple, right?

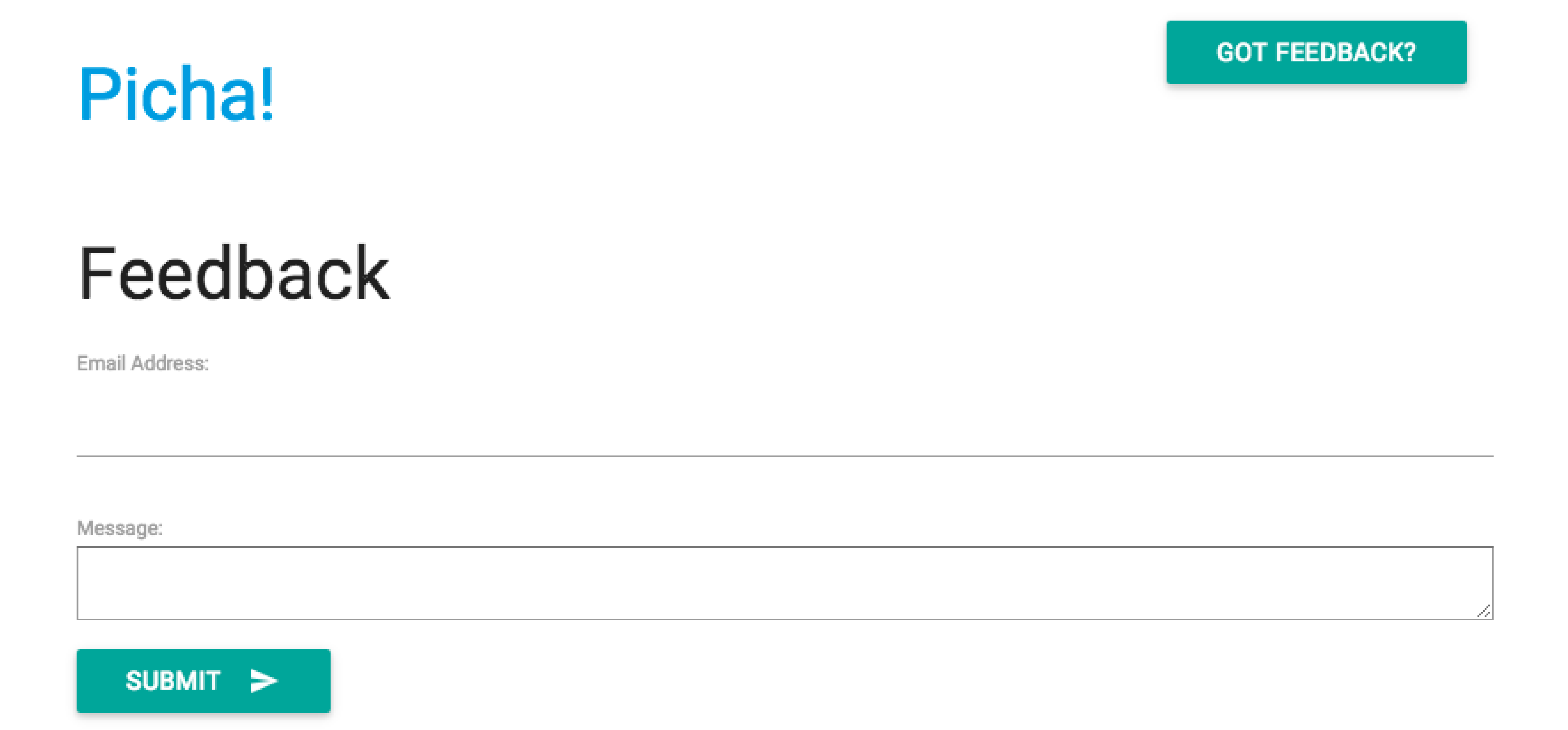
So, these types of tasks are perfect for when you want to load a web page without making the user wait for some background process to complete.

Let’s look at an example…

Going back to the Django Project, grab version three, which includes an app that accepts feedback from users, aptly called `feedback`:

```
├── feedback
│   ├── __init__.py
│   ├── admin.py
│   ├── emails.py
│   ├── forms.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
├── manage.py
├── picha
│   ├── __init__.py
│   ├── celery.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── requirements.txt
└── templates
    ├── base.html
    └── feedback
        ├── contact.html
        └── email
            ├── feedback_email_body
            └── feedback_email_subj
```

Install the new requirements, fire up the app, and navigate to http://localhost:8000/feedback/. You should see:



Let's wire up the Celery task.

Add the Task

Basically, after the user submits the feedback form, we want to immediately let him continue on his merry way while we process the feedback, send an email, etc., all in the background.

To accomplish this, first add a file called *tasks.py* to the "feedback" directory:

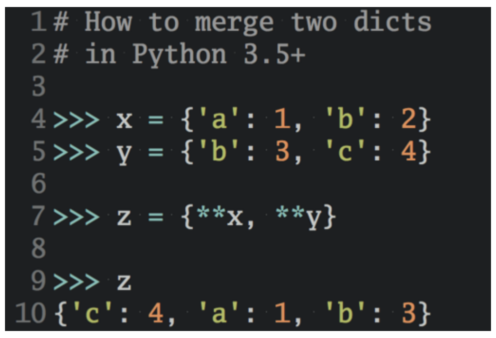
Python

```python
from celery.decorators import task
from celery.utils.log import get_task_logger

from feedback.emails import send_feedback_email

logger = get_task_logger(__name__)


@task(name="send_feedback_email_task")
def send_feedback_email_task(email, message):
    """sends an email when feedback form is filled successfully"""
    logger.info("Sent feedback emai
    return send_feedback_email(emai
```

Then update *forms.py* like so:

Python

```python
from django import forms
from feedback.tasks import send_fee


class FeedbackForm(forms.Form):
    email = forms.EmailField(label=
    message = forms.CharField(
        label="Message", widget=forms.Textarea(attrs={'rows': 5}))
    honeypot = forms.CharField(widget=forms.HiddenInput(), required=False)

    def send_email(self):
        # try to trick spammers by checking whether the honeypot field is
        # filled in; not super complicated/effective but it works
        if self.cleaned_data['honeypot']:
            return False
        send_feedback_email_task.delay(
            self.cleaned_data['email'], self.cleaned_data['message'])
```



```python
# How to merge two dicts
# in Python 3.5+

>>> x = {'a': 1, 'b': 2}
>>> y = {'b': 3, 'c': 4}

>>> z = {**x, **y}

>>> z
{'c': 4, 'a': 1, 'b': 3}
```

Improve Your Python

...with a fresh 🐍 **Python Trick** 💌 code snippet every couple of days:

Email Address

Send Python Tricks »

In essence, the `send_feedback_email_task.delay(email, message)` function processes and sends the feedback email in the background as the user continues to use the site.

> **NOTE**: The `success_url` in *views.py* is set to redirect the user to `/`, which does not exist yet. We'll set this endpoint up in the next section.

Periodic Tasks

Often, you'll need to schedule a task to run at a specific time every so often - i.e., a web scraper may need to run daily, for example. Such tasks, called periodic tasks, are easy to set up with Celery.

Celery uses "celery beat" to schedule periodic tasks. Celery beat runs tasks at regular intervals, which are then executed by celery workers.

For example, the following task is scheduled to run every fifteen minutes:

Python

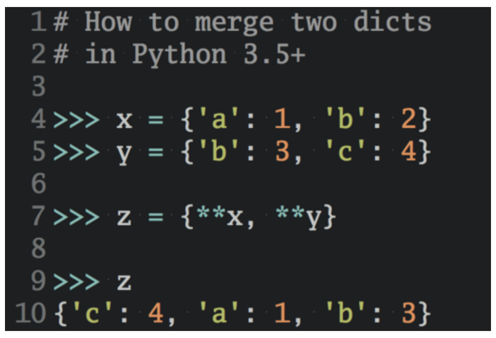
```python
from celery.task.schedules import crontab
from celery.decorators import periodic_task


@periodic_task(run_every=(crontab(minute='*/15')), name="some_task", ignore_result=True)
def some_task():
    # do something
```

Let's look at more robust example by adding this functionality into the Django Project…

Improve Your Python

Back to the Django Project, grab version four, which includes another new app, called `photos`, that uses the Flickr API to get new photos for display on the site:

```
├── feedback
│   ├── __init__.py
│   ├── admin.py
│   ├── emails.py
│   ├── forms.py
│   ├── models.py
│   ├── tasks.py
│   ├── tests.py
│   └── views.py
├── manage.py
├── photos
│   ├── __init__.py
│   ├── admin.py
│   ├── models.py
│   ├── settings.py
│   ├── tests.py
│   ├── utils.py
│   └── views.py
├── picha
│   ├── __init__.py
│   ├── celery.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── requirements.txt
└── templates
    ├── base.html
    ├── feedback
    │   ├── contact.html
    │   └── email
    │       ├── feedback_email_body.txt
    │       └── feedback_email_subject.txt
    └── photos
        └── photo_list.html
```



Install the new requirements, run the migrations, and then fire up the server to make sure all is well. Try testing out the feedback form again. This time it should redirect just fine.

What's next?

Well, since we would need to call the Flickr API periodically to add more photos to our site, we can add a Celery task.

Add the Task

Add a *tasks.py* to the `photos` app:

Python

```python
from celery.task.schedules import crontab
from celery.decorators import periodic_task
from celery.utils.log import get_task_logger

from photos.utils import save_latest_flickr_image

logger = get_task_logger(__name__)


@periodic_task(
    run_every=(crontab(minute='*/15')),
    name="task_save_latest_flickr_i
    ignore_result=True
)
def task_save_latest_flickr_image()
    """
    Saves latest image from Flickr
    """
    save_latest_flickr_image()
    logger.info("Saved image from F
```

Here, we run the `save_latest_flickr_ima` `task`. The `@periodic_task` decorator abstr easy to read!

Running Locally

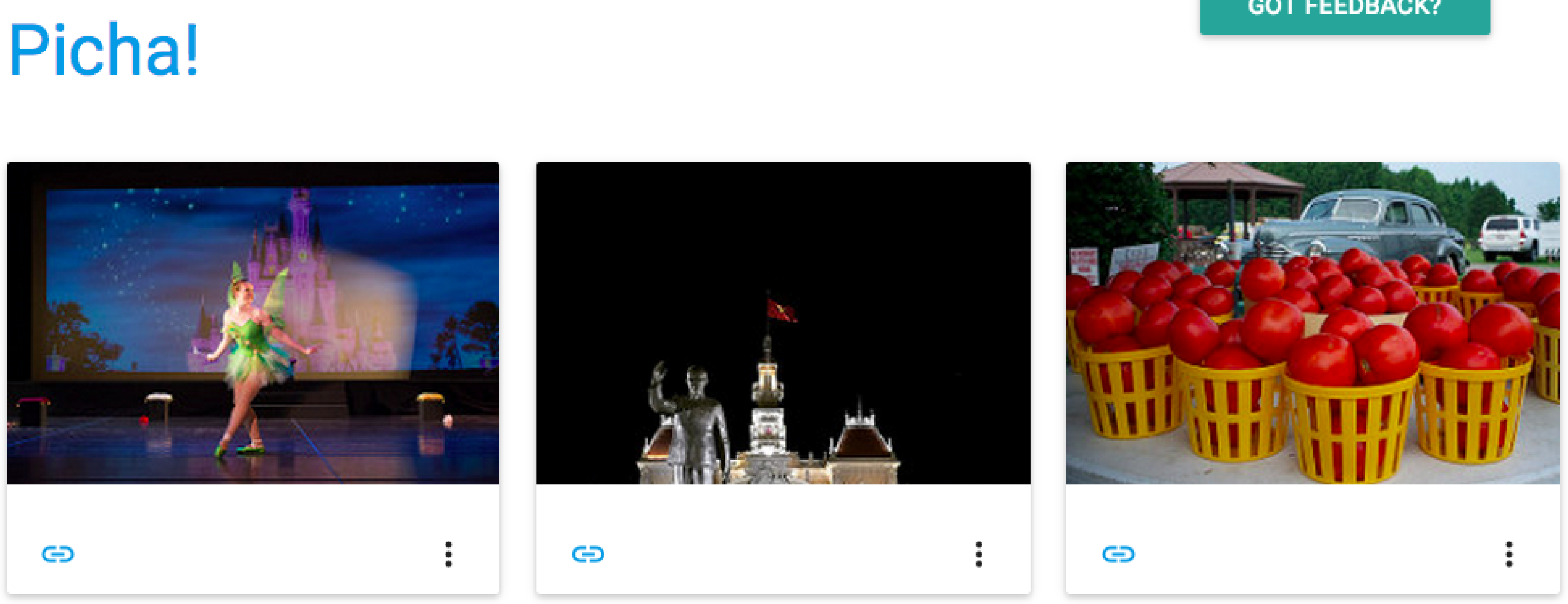
Ready to run this thing?

With your Django App and Redis running, open two new terminal windows/tabs. In each new window, navigate to your project directory, activate your virtualenv, and then run the following commands (one in each window):

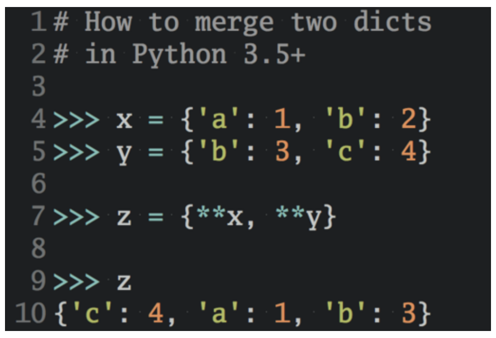
Shell

```shell
$ celery -A picha worker -l info
$ celery -A picha beat -l info
```

When you visit the site on http://127.0.0.1:8000/ you should now see one image. Our app gets one image from Flickr every 15 minutes:



Take a look at `photos/tasks.py` to see the code. Clicking on the “Feedback” button allows you to… send some feedback:



```
# How to merge two dicts
# in Python 3.5+

>>> x = {'a': 1, 'b': 2}
>>> y = {'b': 3, 'c': 4}

>>> z = {**x, **y}

>>> z
{'c': 4, 'a': 1, 'b': 3}
```

This works via a celery task. Have a look at `feedback/tasks.py` for more.

That’s it, you have the Picha project up and running!

This is good for testing while developing your Django Project locally, but does not work so well when you need to deploy to production - like on DigitalOcean, perhaps. For that, it is recommended that you run the Celery worker and scheduler in the background as a daemon with Supervisor.

Running Remotely

Installation is simple. Grab version five fror

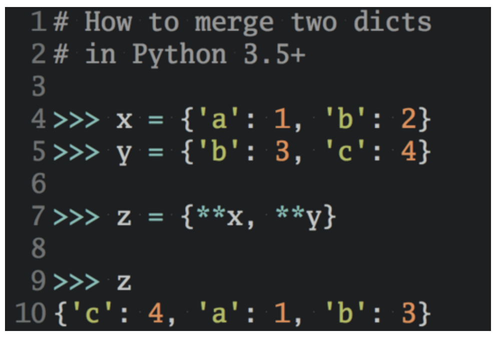
and run:

Shell
```
$ sudo apt-get install supervisor
```

We then need to tell Supervisor about our

“/etc/supervisor/conf.d/” directory on the r

the Celery worker and one for the Celery sc



```
# How to merge two dicts
# in Python 3.5+

>>> x = {'a': 1, 'b': 2}
>>> y = {'b': 3, 'c': 4}

>>> z = {**x, **y}

>>> z
{'c': 4, 'a': 1, 'b': 3}
```

Locally, create a folder called “supervisor”

Celery Worker: ***picha_celery.conf***

Text

```
; ==================================
;  celery worker supervisor example
; ==================================

; the name of your supervisord program
[program:pichacelery]

; Set full path to celery program if using virtualenv
command=/home/mosh/.virtualenvs/picha/bin/celery worker -A picha --loglevel=INFO

; The directory to your Django project
directory=/home/mosh/sites/picha

; If supervisord is run as the root
; before doing any processing.
user=mosh

; Supervisor will start as many ins
numprocs=1

; Put process stdout output in this
stdout_logfile=/var/log/celery/pich

; Put process stderr output in this
stderr_logfile=/var/log/celery/pich

; If true, this program will start automatically when supervisord is started
autostart=true

; May be one of false, unexpected, or true. If false, the process will never
; be autorestarted. If unexpected, the process will be restart when the program
; exits with an exit code that is not one of the exit codes associated with this
; process' configuration (see exitcodes). If true, the process will be
; unconditionally restarted when it exits, without regard to its exit code.
autorestart=true

; The total number of seconds which the program needs to stay running after
; a startup to consider the start successful.
startsecs=10

; Need to wait for currently executing tasks to finish at shutdown.
; Increase this if you have very long running tasks.
stopwaitsecs = 600

; When resorting to send SIGKILL to the program to terminate it
; send SIGKILL to its whole process group instead,
; taking care of its children as well.
killasgroup=true

; if your broker is supervised, set its priority higher
; so it starts first
priority=998
```

Celery Scheduler: ***picha_celerybeat.conf***

Text

```
; ================================
;  celery beat supervisor example
; ================================

; the name of your supervisord program
[program:pichacelerybeat]

; Set full path to celery program if using virtualenv
command=/home/mosh/.virtualenvs/picha/bin/celerybeat -A picha --loglevel=INFO

; The directory to your Django project
directory=/home/mosh/sites/picha

; If supervisord is run as the root
; before doing any processing.
user=mosh

; Supervisor will start as many ins
numprocs=1

; Put process stdout output in this
stdout_logfile=/var/log/celery/pich

; Put process stderr output in this
stderr_logfile=/var/log/celery/pich

; If true, this program will start automatically when supervisord is started
autostart=true

; May be one of false, unexpected, or true. If false, the process will never
; be autorestarted. If unexpected, the process will be restart when the program
; exits with an exit code that is not one of the exit codes associated with this
; process’ configuration (see exitcodes). If true, the process will be
; unconditionally restarted when it exits, without regard to its exit code.
autorestart=true

; The total number of seconds which the program needs to stay running after
; a startup to consider the start successful.
startsecs=10

; if your broker is supervised, set its priority higher
; so it starts first
priority=999
```

Make sure to update the paths in these files to match the remote server’s filesystem.

Basically, these supervisor configuration files tell supervisord how to run and manage our ‘programs’ (as they are called by supervisord).

In the examples above, we have created two supervisord programs named “pichacelery” and “pichacelerybeat”.

Now just copy these files to the remote server in the “/etc/supervisor/conf.d/” directory.

We also need to create the log files that are mentioned in the above scripts on the remote server:

Shell

```
$ touch /var/log/celery/picha_worker.log
$ touch /var/log/celery/picha_beat.log
```

Finally, run the following commands to make Supervisor aware of the programs - e.g., `pichacelery` and `pichacelerybeat`:

Shell

```
$ sudo supervisorctl reread
$ sudo supervisorctl update
```

Run the following commands to stop, start, and/or check the

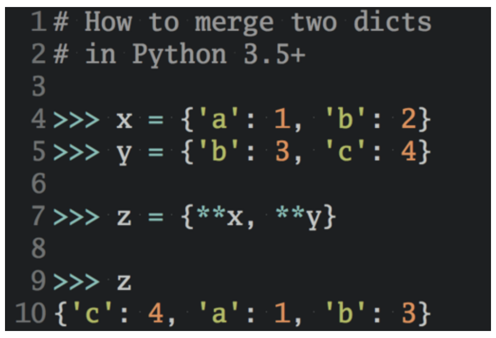
Shell

```
$ sudo supervisorctl stop pichacelery
$ sudo supervisorctl start pichacelery
$ sudo supervisorctl status pichacelery
```

You can read more about Supervisor from the official documentation.

Final Tips

1. *Do not pass Django model objects to C*
   before it is passed to a Celery task, pa
   use the primary key to get the object
2. *The default Celery scheduler creates s*
   schedule.db" and "celerybeat.pid". If
   good idea to ignore this files and not



```
# How to merge two dicts
# in Python 3.5+

>>> x = {'a': 1, 'b': 2}
>>> y = {'b': 3, 'c': 4}

>>> z = {**x, **y}

>>> z
{'c': 4, 'a': 1, 'b': 3}
```

Next steps

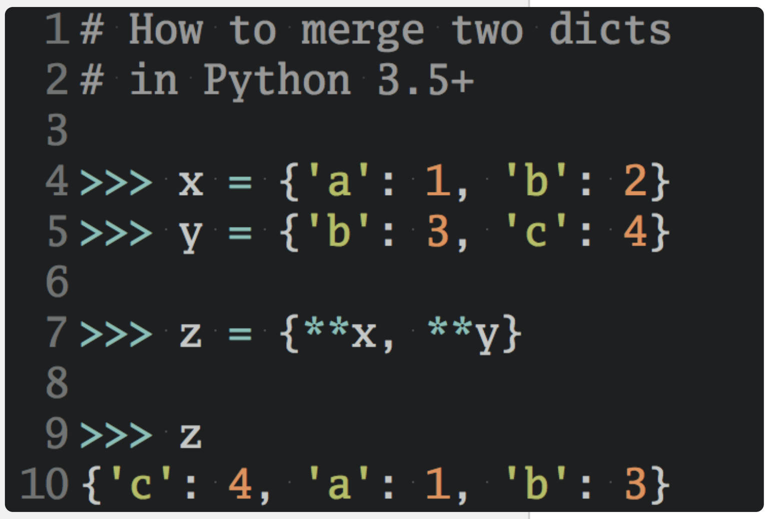
Well, that's it for the basic introduction to i

Want more?

1. Dive into the official Celery User Guide to learn more.
2. Create a Fabfile to setup Supervisor and the configuration files. Make sure to add the commands to `reread` and `update` Supervisor.
3. Fork the Project from the repo and open a Pull Request to add a new Celery task.

Free Bonus: Click here to get access to a free Django Learning Resources Guide (PDF) that shows you tips and tricks as well as common pitfalls to avoid when building Python + Django web applications.

Happy coding!



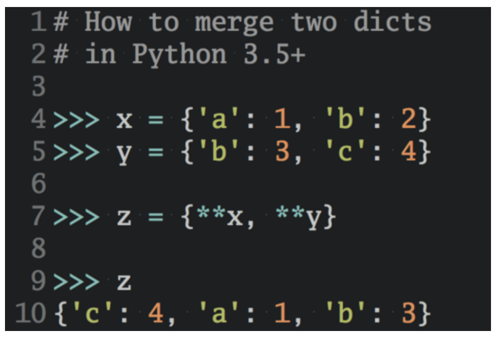
```
# How to merge two dicts
# in Python 3.5+

>>> x = {'a': 1, 'b': 2}
>>> y = {'b': 3, 'c': 4}

>>> z = {**x, **y}

>>> z
{'c': 4, 'a': 1, 'b': 3}
```

What Do You Think?

Tweet Share Email

Real Python Comment Policy: The most useful comments are those written with the goal of learning
from or helping out other readers—a
Complaints and insults generally wo

What's your #1 takeaway or favorite thi
Leave a comment below and let us kno



```
# How to merge two dicts
# in Python 3.5+

>>> x = {'a': 1, 'b': 2}
>>> y = {'b': 3, 'c': 4}

>>> z = {**x, **y}

>>> z
{'c': 4, 'a': 1, 'b': 3}
```

Improve Your Python

...with a fresh 🐍 Python Trick 💌
code snippet every couple of days:

Email Address

Send Python Tricks »

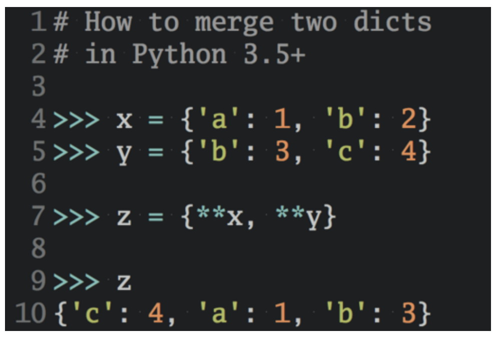
Keep Learning

Related Tutorial Categories: advanced databases django web-dev

All Tutorial Topics

advanced api basics best-practices community databases data-science devops django docker flask front-end intermediate machine-learning python testing tools web-dev web-scraping

Improve Your Python



```
# How to merge two dicts
# in Python 3.5+

>>> x = {'a': 1, 'b': 2}
>>> y = {'b': 3, 'c': 4}

>>> z = {**x, **y}

>>> z
{'c': 4, 'a': 1, 'b': 3}
```

Improve Your Python

...with a fresh 🐍 **Python Trick** 💌 code snippet every couple of days:

Email Address

Send Python Tricks »

Table of Conte

Tweet Share Email

CUSTOM-MADE MUGS & ACCESSORIES FOR PYTHON DEVELOPERS ❤
IN Guido WE TRUST
Python It's good for you.
pythonista

Improve Your Python