Previous topic

Workers Guide

Next topic

Periodic Tasks

This Page

Show Source

Quick search

Go

This document describes the current stable version of Celery (4.2). For development docs, go here.

Daemonization

Generic init-scripts

See the extra/generic-init.d/ directory Celery distribution.

This directory contains generic bash init-scripts for the **celery worker** program, these should run on Linux, FreeBSD, OpenBSD, and other Unix-like platforms.

Init-script: **celeryd**

Usage: */etc/init.d/celeryd {start|stop|restart|status}*
Configuration file:
`/etc/default/celeryd`

To configure this script to run the worker properly you probably need to at least tell it where to change directory to when it starts (to find the module containing your app, or your configuration module).

The daemonization script is configured by the file `/etc/default/celeryd`. This is a shell (**sh**) script where you can add environment variables like the configuration options below. To add real environment variables affecting the worker you must also export them (e.g., **export DISPLAY=":0"**)

Superuser privileges required:

The init-scripts can only be used by root, and the shell configuration file must also be owned by root.

Unprivileged users don't need to use the init-script, instead they can use the **celery multi**

utility (or **celery worker --detach**):

```
$ celery multi start worker1 \
    -A proj \
    --pidfile="$HOME/run/celery/%n.pid" \
    --logfile="$HOME/log/celery/%n%I.log"

$ celery multi restart worker1 \
    -A proj \
    --logfile="$HOME/log/celery/%n%I.log" \
    --pidfile="$HOME/run/celery/%n.pid

$ celery multi stopwait worker1 --pidfile="$HOME/run/celery/%n.pid"
```

Example configuration

This is an example configuration for a Python project.

`/etc/default/celeryd`:

```
# Names of nodes to start
#   most people will only start one node:
CELERYD_NODES="worker1"
#   but you can also start multiple and configure settings
#   for each in CELERYD_OPTS
#CELERYD_NODES="worker1 worker2 worker3"
#   alternatively, you can specify the number of nodes to start:
#CELERYD_NODES=10

# Absolute or relative path to the 'celery' command:
CELERY_BIN="/usr/local/bin/celery"
#CELERY_BIN="/virtualenvs/def/bin/celery"

# App instance to use
# comment out this line if you don't use an app
CELERY_APP="proj"
# or fully qualified:
#CELERY_APP="proj.tasks:app"

# Where to chdir at start.
CELERYD_CHDIR="/opt/Myproject/"

# Extra command-line arguments to the worker
CELERYD_OPTS="--time-limit=300 --concurrency=8"
# Configure node-specific settings by appending node name to arguments:
#CELERYD_OPTS="--time-limit=300 -c 8 -c:worker2 4 -c:worker3 2 -Ofair:wo

# Set logging level to DEBUG
#CELERYD_LOG_LEVEL="DEBUG"

# %n will be replaced with the first part of the nodename.
CELERYD_LOG_FILE="/var/log/celery/%n%I.log"
CELERYD_PID_FILE="/var/run/celery/%n.pid"

# Workers should run as an unprivileged user.
#   You need to create this user manually (or you can choose
#   a user/group combination that already exists (e.g., nobody).
CELERYD_USER="celery"
CELERYD_GROUP="celery"

# If enabled pid and log directories will be created if missing,
# and owned by the userid/group configured.
CELERY_CREATE_DIRS=1
```

Using a login shell

You can inherit the environment of the `CELERYD_USER` by using a login shell:

```
CELERYD_SU_ARGS="-l"
```

Note that this isn't recommended, and that you should only use this option when absolutely necessary.

Example Django configuration

Django users now uses the exact same template as above, but make sure that the module that defines your Celery app instance also sets a default value for **`DJANGO_SETTINGS_MODULE`** as shown in the example Django project in First steps with Django.

Available options

- `CELERY_APP`

  App instance to use (value for **`--app`** argument).

- `CELERY_BIN`

  Absolute or relative path to the **celery** program. Examples:

  - `celery`
  - `/usr/local/bin/celery`
  - `/virtualenvs/proj/bin/celery`
  - `/virtualenvs/proj/bin/python -m celery`

- `CELERYD_NODES`

  List of node names to start (separated by space).

- `CELERYD_OPTS`

  Additional command-line arguments for the worker, see *celery worker –help* for a list. This also supports the extended syntax used by *multi* to configure settings for individual nodes. See *celery multi –help* for some multi-node configuration examples.

- `CELERYD_CHDIR`

  Path to change directory to at start. Default is to stay in the current directory.

- `CELERYD_PID_FILE`

  Full path to the PID file. Default is /var/run/celery/%n.pid

- `CELERYD_LOG_FILE`

  Full path to the worker log file. Default is /var/log/celery/%n%I.log **Note**: Using *%I* is important when using the prefork pool as having multiple processes share the same log file will lead to race conditions.

- `CELERYD_LOG_LEVEL`

  Worker log level. Default is INFO.

- `CELERYD_USER`

  User to run the worker as. Default is current user.

- `CELERYD_GROUP`

Group to run worker as. Default is current user.

- `CELERY_CREATE_DIRS`

  Always create directories (log directory and pid file directory). Default is to only create directories when no custom logfile/pidfile set.

- `CELERY_CREATE_RUNDIR`

  Always create pidfile directory. By default only enabled when no custom pidfile location set.

- `CELERY_CREATE_LOGDIR`

  Always create logfile directory. By default only enable when no custom logfile location set.

Init-script: `celerybeat`

Usage: */etc/init.d/celerybeat {start|stop|restart}*
Configuration file:
`/etc/default/celerybeat` or `/etc/default/celeryd`.

Example configuration

This is an example configuration for a Python project:

/etc/default/celerybeat:

```
# Absolute or relative path to the 'celery' command:
CELERY_BIN="/usr/local/bin/celery"
#CELERY_BIN="/virtualenvs/def/bin/celery"

# App instance to use
# comment out this line if you don't use an app
CELERY_APP="proj"
# or fully qualified:
#CELERY_APP="proj.tasks:app"

# Where to chdir at start.
CELERYBEAT_CHDIR="/opt/Myproject/"

# Extra arguments to celerybeat
CELERYBEAT_OPTS="--schedule=/var/run/celery/celerybeat-schedule"
```

Example Django configuration

You should use the same template as above, but make sure the `DJANGO_SETTINGS_MODULE` variable is set (and exported), and that `CELERYD_CHDIR` is set to the projects directory:

```
export DJANGO_SETTINGS_MODULE="settings"

CELERYD_CHDIR="/opt/MyProject"
```

Available options

- `CELERY_APP`

  App instance to use (value for `--app` argument).

- `CELERYBEAT_OPTS`

Additional arguments to **celery beat**, see **celery beat --help** for a list of available options.

- `CELERYBEAT_PID_FILE`

  Full path to the PID file. Default is `/var/run/celeryd.pid`.

- `CELERYBEAT_LOG_FILE`

  Full path to the log file. Default is `/var/log/celeryd.log`.

- `CELERYBEAT_LOG_LEVEL`

  Log level to use. Default is `INFO`.

- `CELERYBEAT_USER`

  User to run beat as. Default is the current user.

- `CELERYBEAT_GROUP`

  Group to run beat as. Default is the current user.

- `CELERY_CREATE_DIRS`

  Always create directories (log directory and pid file directory). Default is to only create directories when no custom logfile/pidfile set.

- `CELERY_CREATE_RUNDIR`

  Always create pidfile directory. By default only enabled when no custom pidfile location set.

- `CELERY_CREATE_LOGDIR`

  Always create logfile directory. By default only enable when no custom logfile location set.

Troubleshooting

If you can't get the init-scripts to work, you should try running them in *verbose mode*:

```
# sh -x /etc/init.d/celeryd start
```

This can reveal hints as to why the service won't start.

If the worker starts with *"OK"* but exits almost immediately afterwards and there's no evidence in the log file, then there's probably an error but as the daemons standard outputs are already closed you'll not be able to see them anywhere. For this situation you can use the **`C_FAKEFORK`** environment variable to skip the daemonization step:

```
# C_FAKEFORK=1 sh -x /etc/init.d/celeryd start
```

and now you should be able to see the errors.

Commonly such errors are caused by insufficient permissions to read from, or write to a file, and also by syntax errors in configuration modules, user modules, third-party libraries, or even from Celery itself (if you've found a bug you should report it).

Usage `systemd`

- extra/systemd/

Usage: *systemctl {start|stop|restart|status} celery.service*
Configuration file:
/etc/conf.d/celery

Service file: celery.service

This is an example systemd file:

`/etc/systemd/system/celery.service`:

```
[Unit]
Description=Celery Service
After=network.target

[Service]
Type=forking
User=celery
Group=celery
EnvironmentFile=/etc/conf.d/celery
WorkingDirectory=/opt/celery
ExecStart=/bin/sh -c '${CELERY_BIN} multi start ${CELERYD_NODES} \
  -A ${CELERY_APP} --pidfile=${CELERYD_PID_FILE} \
  --logfile=${CELERYD_LOG_FILE} --loglevel=${CELERYD_LOG_LEVEL} ${CELERY
ExecStop=/bin/sh -c '${CELERY_BIN} multi stopwait ${CELERYD_NODES} \
  --pidfile=${CELERYD_PID_FILE}'
ExecReload=/bin/sh -c '${CELERY_BIN} multi restart ${CELERYD_NODES} \
  -A ${CELERY_APP} --pidfile=${CELERYD_PID_FILE} \
  --logfile=${CELERYD_LOG_FILE} --loglevel=${CELERYD_LOG_LEVEL} ${CELERY

[Install]
WantedBy=multi-user.target
```

Once you've put that file in `/etc/systemd/system`, you should run **systemctl daemon-reload** in order that Systemd acknowledges that file. You should also run that command each time you modify it.

To configure user, group, **chdir** change settings: `User`, `Group`, and `WorkingDirectory` defined in `/etc/systemd/system/celery.service`.

You can also use systemd-tmpfiles in order to create working directories (for logs and pid).

file: */etc/tmpfiles.d/celery.conf*

```
d /var/run/celery 0755 celery celery -
d /var/log/celery 0755 celery celery -
```

Example configuration

This is an example configuration for a Python project:

`/etc/conf.d/celery`:

```
# Name of nodes to start
# here we have a single node
CELERYD_NODES="w1"
# or we could have three nodes:
#CELERYD_NODES="w1 w2 w3"

# Absolute or relative path to the 'celery' command:
CELERY_BIN="/usr/local/bin/celery"
#CELERY_BIN="/virtualenvs/def/bin/celery"

# App instance to use
# comment out this line if you don't use an app
CELERY_APP="proj"
# or fully qualified:
#CELERY_APP="proj.tasks:app"

# How to call manage.py
CELERYD_MULTI="multi"

# Extra command-line arguments to the worker
CELERYD_OPTS="--time-limit=300 --concurrency=8"

# - %n will be replaced with the first part of the nodename.
# - %I will be replaced with the current child process index
#   and is important when using the prefork pool to avoid race condition
CELERYD_PID_FILE="/var/run/celery/%n.pid"
CELERYD_LOG_FILE="/var/log/celery/%n%I.log"
CELERYD_LOG_LEVEL="INFO"
```

Running the worker with superuser privileges (root)

Running the worker with superuser privileges is a very dangerous practice. There should always be a workaround to avoid running as root. Celery may run arbitrary code in messages serialized with pickle - this is dangerous, especially when run as root.

By default Celery won't run workers as root. The associated error message may not be visible in the logs but may be seen if **`C_FAKEFORK`** is used.

To force Celery to run workers as root use **`C_FORCE_ROOT`**.

When running as root without **`C_FORCE_ROOT`** the worker will appear to start with *"OK"* but exit immediately after with no apparent errors. This problem may appear when running the project in a new development or production environment (inadvertently) as root.

supervisor

- extra/supervisord/

`launchd` (macOS)

- extra/macOS