motion-pipeline Documentation

Release 0.1.0

Jason Antman

Jun 13, 2018

Contents

Frontend and recording management pipeline for the Motion video motion detection project.

Docs: https://motion-pipeline.readthedocs.io/en/latest/

CHAPTER 1

Status

This project is abandoned. I got it working in a very basic sense - it's able to handle events from `motion` (motion detection and new movies), display the movies in the very simplistic web UI, extract the first frame, and send a summary notification (with the first frame image) via Pushover. It also has support for archiving/deleting events and some basic classification of them.

Unfortunately I moved last week, and the motion detection logic in `motion` can't handle all of the moving shadows from trees on my property. ZoneMinder, which takes color information into account, can handle it much better. So I've abandoned this in favor of ZoneMinder. I'm leaving the code here in case it's of use to anyone.

CHAPTER 2

Introduction

I have two IP cameras at home that I'd like to use for security, specifically motion-activated recording and notification of events when I'm away. The cameras (see my detailed unboxing/review of them on my blog) are 960P and 1080P, respectively. The current options for Free/Open Source software to do this aren't adequate for me; ZoneMinder, the de-facto standard, doesn't meet my resource constraints of being able to run on (or partially run on) a RaspberryPi 3 B+ and the other options I could find aren't mature or lack features I need.

As a result, I'm building this. It's a project composed of multiple services and intended to handle processing data from Motion (recordings and the metadata associated with them), storing it, and providing a viewing interface and notifications/alerts. The project is intended to be modular, utilizing a storage service (S3 or the local S3-compatible minio), a queue (Redis) and a database (MySQL) to connect a handler that runs on the same device as `motion` (this could be anything from a RaspberryPi to a server), an asynchronous task worker for ingesting new data from motion, triggering notifications, and generating thumbnails, and a web frontend.

CHAPTER 3

Documentation

Full documentation for this project is hosted on ReadTheDocs at https://motion-pipeline.readthedocs.io/en/latest/.

CHAPTER 4

Bugs and Feature Requests

Bug reports and feature requests are happily accepted via the GitHub Issue Tracker. Pull requests are welcome. Issues that don't have an accompanying pull request will be worked on as my time and priority allows.

CHAPTER 5

Contents

5.1 Overview

I have two IP cameras at home that I'd like to use for security, specifically motion-activated recording and notification of events when I'm away. The cameras (see my detailed unboxing/review of them on my blog) are 960P and 1080P, respectively. The current options for Free/Open Source software to do this aren't adequate for me; ZoneMinder, the de-facto standard, doesn't meet my resource constraints of being able to run on (or partially run on) a RaspberryPi 3 B+ and the other options I could find aren't mature or lack features I need.

As a result, I'm building this. It's a project composed of multiple services and intended to handle processing data from Motion (recordings and the metadata associated with them), storing it, and providing a viewing interface and notifications/alerts. The project is intended to be modular, utilizing a storage service (S3 or the local S3-compatible minio), a queue (Redis) and a database (MySQL) to connect a handler that runs on the same device as `motion` (this could be anything from a RaspberryPi to a server), an asynchronous task worker for ingesting new data from motion, triggering notifications, and generating thumbnails, and a web frontend.

The architecture is intended to be decoupled, equally happy on a single fanless computer or in the cloud, and to allow separating realtime tasks (`motion` for motion detection and recording, and ingesting its recordings and metadata into the system) from less-time-sensitive tasks (generating thumbnails for videos, etc.) and the user interface.

5.1.1 Architecture

The main architecture includes three pieces: `motion_handler.py` that is triggered by `motion`, the asynchronous task workers, and the web frontend. It relies on Redis, MySQL, and Amazon S3 or a local S3-compatible storage service (minio) as dependencies.

5.1.1.1 Motion Ingest

```
    +---+     +-----------------+
    |cam+----->RaspberryPi, etc.|    ... N instances of
    +---+     |motion_handler.py|    cam -> motion_handler
```

(continues on next page)

(continued from previous page)

```
               +---+------------++
                   |            |
                   |            |
+------------------v----+      +v-----------------+
|redis "motion-incoming"|      |S3 or minio bucket|
|queue                  |      +-+--------^-------+
+-------------+---------+        |        |
              |                  |        |
              |                  |        |
  +-----------v------+    +------v--------+-----+
  |data ingest worker+---->Video Thumbnail      |
  |Worker            |    |Worker               |
  +------------------+    +-----+-----------+---+
                     |          |           |
                     |          |     +-----v-----------+
                    +v----------v--+  |user notification|
                    |MySQL Database|  |worker           |
                    +--------------+  +-----------------+


(generated by asciiflow Infinity - http://asciiflow.com/)
```

5.1.1.2 User Interface and Notifications

TBD.

5.2 Installation

motion-pipeline is made up of three overall components that can be installed separately: `motion_handler.py` (the script used by `motion` itself in the `on_*` settings), the task worker, and the web frontend.

5.2.1 Requirements

- Python 3.4+ (currently tested with 3.4, 3.5, 3.6) and `pip`
- Python VirtualEnv (recommended installation method; your OS/distribution should have packages for these)
- motion on the system that will run the motion-handler
- Redis and MySQL on *some* system (can be colocated with any of these components, or completely separate system(s))

For installations other than on RaspberryPi or similar dedicated systems, it's recommended that you install into a virtual environment (virtualenv / venv). See the virtualenv usage documentation for information on how to create a venv.

To install the requirements for `motion-handler` on a RaspberryPi running Raspbian Stretch (`2018-04-18-raspbian-stretch-lite` image) I did the following:

1. `wget https://github.com/Motion-Project/motion/releases/download/release-4.1.1/pi_stretch_motion_4.1.1-1_armhf.deb && sudo apt install ./pi_stretch_motion_4.1.1-1_armhf.deb` - this installs the `motion` [4.1.1 package from GitHub](https://github.com/Motion-Project/motion/releases/tag/release-4.1.1), which includes some important fixes over the 4.0-1 package in the Raspbian Stretch repositories.

2. `sudo apt-get install python3-pip git python3-dev`

If you don't have them installed already, you may need to install the common build tools on your OS.

5.2.2 Installation of Dependencies

- To install motion-pipeline and only the dependencies for `motion_handler.py`: `pip install motion-pipeline`
- To install motion-pipeline and the dependencies for the task worker: `pip install motion-pipeline[worker]`
- To install motion-pipeline and the dependencies for the web frontend: `pip install motion-pipeline[web]`
- To install all of the above: `pip install motion-pipeline[all]`

5.2.3 Installation of motion_handler.py

I haven't had time to write real installation tooling for this yet. In the mean time, on the computer where you'll be running `motion`:

1. `pip install motion-pipeline`
2. Find and record the absolute path to the `motion-handler` entrypoint: `which motion-handler`
3. Copy your *motion-pipeline settings file or module* somewhere on that machine (as an example, we put it at `/etc/motion-pipeline_settings.py`).
4. Set up your `motion.conf` file for motion. Specify the various `on_*` configuration settings to point to the path to the motion-handler entrypoint and set the `--config` option to the path to your settings file/module. See *Example motion.conf* for an example.

5.3 Configuration

5.3.1 motion-pipeline settings file or module

`motion-pipeline` is configured via a Python settings file. This file is imported by `settings.py` and can be specified as either a dot-delimited importable python module or the absolute path to a Python source file on disk (if using a source file, it must be specified as an absolute path). The configuration file can be set either via the `MOTION_SETTINGS_PATH` environment variable or via the `-c` / `--config` options to most of the entrypoint scripts.

Note that this file is imported whenever `motion` sends an event to `motion-handler`. While it *can* include any Python code, it should be as simple and fast-loading as possible.

An example file is in the motion-pipeline source as `settings_example.py`; you should copy that file as an example and edit as necessary.

5.3.2 Configuration of Motion Itself

Configuration of `motion` itself involves setting the correct event handler commands to point to `motion_handler.py` and pass the required arguments. See the example below.

5.3.2.1 Example motion.conf

The only parts of `motion.conf` that are specific to motion-pipeline are the various `on_*` options that tell motion to execute our handler when various events occur:

```

# Number of steps to make (stepper motor option) (default: 40)
track_stepsize 40


############################################################
# External Commands, Warnings and Logging:
# You can use conversion specifiers for the on_xxxx commands
# %Y = year, %m = month, %d = date,
# %H = hour, %M = minute, %S = second,
# %v = event, %q = frame number, %t = camera id number,
# %D = changed pixels, %N = noise level,
# %i and %J = width and height of motion area,
# %K and %L = X and Y coordinates of motion center
# %C = value defined by text_event
# %f = filename with full path
# %n = number indicating filetype
# Both %f and %n are only defined for on_picture_save,
# on_movie_start and on_movie_end
# Quotation marks round string are allowed.
############################################################

# Do not sound beeps when detecting motion (default: on)
# Note: Motion never beeps when running in daemon mode.
quiet on

# Command to be executed when an event starts. (default: none)
# An event starts at first motion detected after a period of no motion defined by␣
↪event_gap
on_event_start /usr/local/share/motion-pipeline/motion_pipeline/motion_handler.py --
↪config=/etc/motion/mysettings.py --action=event_start --date="%Y-%m-%d %H:%M:%S" --
↪event_id=%v --frame_num=%q --cam=%t --changed_px=%D --noise=%N --text_event="%C" --
↪motion_width=%i --motion_height=%J --motion_center_x=%K --motion_center_y=%L --cam-
↪name="%$" --threshold=%o --labels=%Q --fps=%{fps} --host=%{host}

# Command to be executed when an event ends after a period of no motion
# (default: none). The period of no motion is defined by option event_gap.
on_event_end /usr/local/share/motion-pipeline/motion_pipeline/motion_handler.py --
↪config=/etc/motion/mysettings.py --action=event_end --date="%Y-%m-%d %H:%M:%S" --
↪event_id=%v --frame_num=%q --cam=%t --changed_px=%D --noise=%N --text_event="%C" --
↪motion_width=%i --motion_height=%J --motion_center_x=%K --motion_center_y=%L --cam-
↪name="%$" --threshold=%o --labels=%Q --fps=%{fps} --host=%{host}

# Command to be executed when a picture (.ppm|.jpg) is saved (default: none)
# To give the filename as an argument to a command append it with %f
on_picture_save /usr/local/share/motion-pipeline/motion_pipeline/motion_handler.py --
↪config=/etc/motion/mysettings.py --action=picture_save --date="%Y-%m-%d %H:%M:%S" --
↪event_id=%v --cam=%t --text_event="%C" --filename="%f" --filetype=%n --cam-name="%$
↪" --threshold=%o --labels=%Q --fps=%{fps} --host=%{host}

# Command to be executed when a motion frame is detected (default: none)
; on_motion_detected value

```

(continues on next page)

(continued from previous page)

```
# Command to be executed when motion in a predefined area is detected
# Check option 'area_detect'.   (default: none)
; on_area_detected value

# Command to be executed when a movie file (.mpg|.avi) is created. (default: none)
# To give the filename as an argument to a command append it with %f
; on_movie_start value

# Command to be executed when a movie file (.mpg|.avi) is closed. (default: none)
# To give the filename as an argument to a command append it with %f
on_movie_end /usr/local/share/motion-pipeline/motion_pipeline/motion_handler.py --
↪config=/etc/motion/mysettings.py --action=movie_end --date="%Y-%m-%d %H:%M:%S" --
↪event_id=%v --cam=%t --text_event="%C" --filename="%f" --filetype=%n --cam-name="%$
↪" --threshold=%o --labels=%Q --fps=%{fps} --host=%{host}

# Command to be executed when a camera can't be opened or if it is lost
# NOTE: There is situations when motion don't detect a lost camera!
# It depends on the driver, some drivers dosn't detect a lost camera at all
# Some hangs the motion thread. Some even hangs the PC! (default: none)
on_camera_lost /usr/local/share/motion-pipeline/motion_pipeline/motion_handler.py --
↪config=/etc/motion/mysettings.py --action=cam_lost --date="%Y-%m-%d %H:%M:%S" --cam=
↪%t --cam-name="%$" --host=%{host}
```

5.4 Running

Here's how to run it.

5.4.1 motion_handler.py

`motion_handler.py` is run automatically by `motion` when relevant events are fired. All that needs to be done is *install it*, *configure it and motion itself*, and then run `motion` however is appropriate for your OS.

5.4.2 Celery Task Workers

1. If you installed in a virtualenv, source it. Ensure you've run `python setup.py develop` or `pip install motion-pipeline[worker]`
2. Ensure Redis is running and the `REDIS_BROKER_URL` setting is correct.
3. Ensure the path to your settings file is exported as `MOTION_SETTINGS_PATH`
4. `celery -A motion_pipeline.celerytasks.tasks worker --loglevel=info -Ofair -c 3`

5.4.3 Web Frontend

1. If you installed in a virtualenv, source it. Ensure you've run `python setup.py develop` or `pip install motion-pipeline[web]`

2. Ensure the path to your settings file is exported as `MOTION_SETTINGS_PATH` 4. `FLASK_APP=motion_pipeline.web.app flask run`

5.5 Development

To install for development:

1. Fork the python-package-skeleton repository on GitHub
2. Create a new branch off of master in your fork.

```
$ virtualenv python-package-skeleton
$ cd python-package-skeleton && source bin/activate
$ pip install -e git+git@github.com:YOURNAME/python-package-skeleton.git@BRANCHNAME
↪#egg=python-package-skeleton
$ cd src/python-package-skeleton
```

The git clone you're now in will probably be checked out to a specific commit, so you may want to `git checkout BRANCHNAME`.

5.5.1 Guidelines

- pep8 compliant with some exceptions (see pytest.ini)
- 100% test coverage with pytest (with valid tests)

5.5.2 Testing

Testing is done via pytest, driven by tox.

- testing is as simple as:
  - `pip install tox`
  - `tox`
- If you want to pass additional arguments to pytest, add them to the tox command line after "–". i.e., for verbose pytext output on py27 tests: `tox -e py27 -- -v`

5.5.3 Alembic Database Migrations

This project uses Alembic for DB migrations:

- To generate migrations, run `alembic -c motion_pipeline/alembic/alembic.ini revision --autogenerate -m "message"` and examine/edit then commit the resulting file(s). This must be run *before* the model changes are applied to the DB. If adding new models, make sure to import the model class in `models/__init__.py`.
- To apply migrations, run `alembic -c motion_pipeline/alembic/alembic.ini upgrade head`.
- To see the current DB version, run `alembic -c motion_pipeline/alembic/alembic.ini current`.
- To see migration history, run `alembic -c motion_pipeline/alembic/alembic.ini history`.

5.5.4 Release Checklist

1. Open an issue for the release; cut a branch off master for that issue.
2. Confirm that there are CHANGES.rst entries for all major changes.
3. Ensure that Travis tests passing in all environments.
4. Ensure that test coverage is no less than the last release (ideally, 100%).
5. Increment the version number in python-package-skeleton/version.py and add version and release date to CHANGES.rst, then push to GitHub.
6. Confirm that README.rst renders correctly on GitHub.
7. Upload package to testpypi:
   - Make sure your ~/.pypirc file is correct (a repo called `test` for https://testpypi.python.org/pypi)
   - `rm -Rf dist`
   - `python setup.py register -r https://testpypi.python.org/pypi`
   - `python setup.py sdist bdist_wheel`
   - `twine upload -r test dist/*`
   - Check that the README renders at https://testpypi.python.org/pypi/python-package-skeleton
8. Create a pull request for the release to be merged into master. Upon successful Travis build, merge it.
9. Tag the release in Git, push tag to GitHub:
   - tag the release. for now the message is quite simple: `git tag -s -a X.Y.Z -m 'X.Y.Z released YYYY-MM-DD'`
   - push the tag to GitHub: `git push origin X.Y.Z`

11. Upload package to live pypi:
    - `twine upload dist/*`

10. make sure any GH issues fixed in the release were closed.

5.6 motion-pipeline

5.7 Changelog

5.7.1 x.y.z (YYYY-MM-DD)

- something

CHAPTER 6

Indices and tables

- genindex
- modindex
- search