talon Documentation

Release 0.0.0

Chris Withers

Feb 09, 2018

Contents

CHAPTER 1

Concepts

1.1 What is it?

Talon is a system for periodically running jobs that have constraints.

A *job* will be *run* for a *period* when *triggered*, but will block until all its *constraints* have been met and it can been allocated suitable *resources*.

The *schedule* contains details of jobs to be run and any context needed to run them. *Periods* and the *runs* and *artifacts* that occur within them are stored in the *state store* along with details of the *resources* currently available to the system.

Configuration specifies what implementations to use for for the schedule, state and scheduler along with the types of job, run, period, trigger, constraint and resource available to the schedule.

1.2 Anti-goals

Talon does not aim to do any of the following:

- Get software, including itself, to where executors need it.
- Get data used by jobs run by executors to those executors.
- Implementation of executors is not the focus, managing constraints and state is the focus.

1.3 Job

Attributes

- name
- poll frequency - how long to wait between checking if constraints have been met (default to once a minute)

1.4 Run

- job
- period
- executor
- started
- *Progress*
- finished

The executor indicates who 'owns' the job.

Each run may produce zero or more ref:*artifacts* <*artifact*>.

1.5 Artifacts

- artifacts are identified by a url giving their location
- each artifact is associated with a period and a run
- could be a file, or some data

1.6 Progress

Run progress has a state that is one of the following:

- blocked - triggered, but one or more constraints have not been met
- ready - constraints have been met
- running - an executor is executing this run
- succeeded
- failed
- skipped - cancelled by a constraint

When running, progress may be expressed as:

- percentage-based
- duration-based
- item-based

When item-based, each item may also have a state.

1.7 Period

Attributes:

- type
- start

- environment?

Types:

- daily
- weekly
- monthly
- quarterly
- one shot?

1.8 Constraint

Types:

- start by
- once other specified job has completed
- not more than one successful run in a period
- not more than one instance can be running at once
- not more than one instance can be blocked at once

When checked, contraints will return one of the following actions to take:

- block - this constraint has not been met
- fail - this constraint can never be met and the run should be failed
- pass - this constraint has been met.
- cancel - this constraint has indicated that the current run should be skipped

1.9 Resources

- host of a specific type
- a particular environment?
- actively generated by a job? (or should those be artifacts?)
- resource within a period?
- resource within an environment?
- resources are namespaced, think of them as a nest dictionary hierarchy

1.10 Trigger

These create the *run*.

Examples:

- extended cron-style, eg: “every 15 minutes between 7-10am, hourly until 4pm, every 5 minutes between 4-5pm”
- external

- prior job run completed

1.11 Executor

- SGE
- SSH
- in-process
- Remote node.

1.12 Schedule

- jobs
- parameters for jobs: - simple data - complex data? - artifact specifications?

1.13 State

Stores information about the runs of jobs.

1.14 Config

- config for state store
- config for config2 store
- types of resource, constraint, etc
- auth data sources

CHAPTER 2

Development

This package is developed using continuous integration which can be found here:

https://travis-ci.org/cjw296/talon

The latest development version of the documentation can be found here:

http://talon.readthedocs.org/en/latest/

If you wish to contribute to this project, then you should fork the repository found here:

https://github.com/cjw296/talon/

Once that has been done and you have a checkout, you can follow these instructions to perform various development tasks:

2.1 Setting up a virtualenv

The recommended way to set up a development environment is to turn your checkout into a virtualenv and then install the package in editable form as follows:

```
$ virtualenv .
$ bin/pip install -U -e .[test,build]
```

2.2 Running the tests

Once you've set up a virtualenv, the tests can be run as follows:

```
$ bin/nosetests
```

2.3 Building the documentation

The Sphinx documentation is built by doing the following from the directory containing setup.py:

```
$ source bin/activate
$ cd docs
$ make html
```

To check that the description that will be used on PyPI renders properly, do the following:

```
$ python setup.py --long-description | rst2html.py > desc.html
```

The resulting `desc.html` should be checked by opening in a browser.

To check that the README that will be used on GitHub renders properly, do the following:

```
$ cat README.rst | rst2html.py > readme.html
```

The resulting `readme.html` should be checked by opening in a browser.

2.4 Making a release

To make a release, just update the version in `setup.py`, update the change log, tag it and push to https://github.com/cjw296/talon and Travis CI should take care of the rest.

Once Travis CI is done, make sure to go to https://readthedocs.org/projects/talon/versions/ and make sure the new release is marked as an Active Version.

CHAPTER 3

Changes

3.1 A.B.C (XX YYY ZZZZ)

- Sketching

CHAPTER 4

License

CHAPTER 5

Indices and tables

- genindex
- modindex
- search