jobarchitect Documentation

Release 0.6.0

Tjelvar Olsson

Jun 15, 2017

Contents

CHAPTER 1

Content

Architect jobs for running analyses

- Documentation: http://jobarchitect.readthedocs.io
- GitHub: https://github.com/JIC-CSB/jobarchitect
- PyPI: https://pypi.python.org/pypi/jobarchitect
- Free software: MIT License

Overview

This tool is intended to automate generation of scripts to run analysis on data sets. To use it, you will need a data set that has been created (or annotated) with dtool. It aims to help by:

1. Removing the need to know where specific data items are stored in a data set
2. Providing a means to split an analyses into several chunks (file based parallelization)
3. Providing a framework for seamlessly running an analyses inside a container

Design

This project has two main components. The first is a command line tool named `sketchjob` intended to be used by the end user. It is used to generate scripts defining jobs to be run. The second (`_analyse_by_ids`) is a command line tool that is used by the scripts generated by `sketchjob`. The end user is not meant to make use of this second script directly.

Installation

To install the jobarchitect package.

```
$ cd jobarchitect
$ python setup.py install
```

Use

The `jobarchitect` tool only works with “smart” tools. A “smart” tool is a tool that understands dtoolcore datasets, has no positional command line arguments and supports the named arguments `--dataset-path`, `--identifier`, `--output-directory`. The tool should only process the dataset item specified by the identifier and write all output to the specified output directory.

A dtool dataset can be created using dtool. Below is some sample:

```
$ dtool new dataset
project_name [project_name]:
dataset_name [dataset_name]: example_dataset
...

$ echo "My example data" > example_dataset/data/my_file.txt
$ datatool manifest update example_dataset/
```

Create an output directory:

```
$ mkdir output
```

Then you can generate analysis run scripts with:

```
$ sketchjob my_smart_tool.py exmaple_dataset output/
#!/bin/bash

_analyse_by_ids \
  --tool_path=my_smart_tool.py \
  --input_dataset_path=example_dataset/ \
  --output_root=output/ \
  290d3f1a902c452ce1c184ed793b1d6b83b59164
```

Try the script with:

```
$ sketchjob my_smart_tool.py exmaple_dataset output/ > run.sh
$ bash run.sh
$ cat output/first_image.png
290d3f1a902c452ce1c184ed793b1d6b83b59164  /private/var/folders/hn/
↪crprzwh12kj95plc9jjtxmq82nl2v3/T/tmp_pTfc6/stg02d730c7-17a2-4d06-a017-e59e14cb8885/
↪first_image.png
```

Working with Docker

Building a Docker image

For the tests to pass, you will need to build an example Docker image, which you do with the provided script:

```
$ bash build_docker_image.sh
```

Running code with the Docker backend

By inspecting the script and associcated Docker file, you can get an idea of how to build Docker images that can be used with the jobarchitect Docker backend, e.g:

```
$ sketchjob scripts/my_smart_tool.py ~/junk/cotyledon_images ~/junk/output --
↪backend=docker --image-name=jicscicomp/jobarchitect
#!/bin/bash

IMAGE_NAME=jicscicomp/jobarchitect
docker run  \
  --rm  \
  -v /Users/olssont/junk/cotyledon_images:/input_dataset:ro  \
  -v /Users/olssont/junk/output:/output  \
  -v /Users/olssont/sandbox/scripts:/scripts:ro \
  $IMAGE_NAME  \
  _analyse_by_ids  \
    --tool_path=/scripts/my_smart_tool.py \
    --input_dataset_path=/input_dataset  \
    --output_root=/output  \
    290d3f1a902c452ce1c184ed793b1d6b83b59164 09648d19e11f0b20e5473594fc278afbede3c9a4
```

Defining jobs using CWL

CWL tools are defined using a YAML syntax. They specify the way in which a command is run.

When Jobarchitect uses CWL to run its jobs, it makes use of two hardcoded parameters, input_file and output_file. These are then dynamically generated from items in the dataset.

An example wrapping the `cp` command looks like this:

```
cwlVersion: v1.0
class: CommandLineTool
baseCommand: cp
inputs:
  input_file:
    type: File
    inputBinding:
      position: 1
  output_file:
    type: string
    inputBinding:
      position: 2
outputs:
   an_output_file:
     type: File
     outputBinding:
       glob: $(inputs.output_file)
```

This can then be combined with the job description file below.

```
input_file:
  class: File
  path: /Users/olssont/sandbox/cwl_v1/dummy.txt
output_file: dummycopy.txt
```

The job description files are generated dynamically by Jobarchitect’s agent (_analyse_by_ids) at runtime.

A different example using redirection (such that output is captured from stdout) is illustrated below:

```
cwlVersion: v1.0
class: CommandLineTool
inputs:
  input_file:
    type: File
    inputBinding: { position: 1 }
baseCommand: shasum
outputs:
  shasum: stdout
stdout: $(inputs.output_file)
```

Design

Background

The first implementations of jobarchitect used a basic templating language for defining jobs, with two reserved keywords input_file and output_file. An example input job.tmpl file would look like the below.

```
shasum {{ input_file }} > {{ output_file }}
```

A later revision made use of the Common Workflow Language, still making use of the reserved keywords input_file and output_file. The same example as a shasum.cwl file would look like the below.

```
cwlVersion: v1.0
class: CommandLineTool
inputs:
  input_file:
    type: File
    inputBinding: { position: 1 }
baseCommand: shasum
outputs:
  shasum: stdout
stdout: $(inputs.output_file)
```

The jobarchitect tool always had an explicit knowledge of dtoolcore datasets. The sketchjob utility uses this knowledge to split the dataset into batches.

When dtoolcore datasets’ gained the ability to store and provide access to file level metadata our tools started making use of this. However, sketchjob or more precisely _analyse_by_id, assumed that the tools it ran would not have an understanding or need to access this dataset file level metadata.

The section below outlines our thinking with regards to overcoming this problem.

Solution

Now that our scripts work on datasets, rather than individual files, they work at a similar level to _analyse_by_id.

One solution would be to make _analyse_by_id more complex allowing it to know when to work on files in a dataset and when to work on datasets and associated identifiers. The latter is what would be required for our new scripts to work.

Another solution would be to by-pass `_analyse_by_id` completely with our script of interest (that works on datasets and identifiers). The `_analyse_by_id` script could remain accessible, via a `--use-cwl` option which would invoke the existing behaviour.

Another solution would be to add another layer of abstraction, for example a script named `agent.py` that could call either `_analyse_by_id` or the provided script. In this scenario `_analyse_by_id` and user provided scripts would become alternative backends to the `agent.py` script. As such it would make sense to rename `_analyse_by_id` to `_cwl_backend`.

We prefer, the latter of these options. The job written out by `sketchjob` would then take the form of:

```
sketchjob --cwl-backend shasum.cwl exmaple_dataset output/
#!/bin/bash

_jobarchitect_agent \
  --cwl-backend
  --tool_path=shasum.cwl \
  --input_dataset_path=example_dataset/ \
  --output_root=output/ \
  290d3f1a902c452ce1c184ed793b1d6b83b59164
```

Or in the case of a custom script:

```
sketchjob scripts/analysis.py exmaple_dataset output/
#!/bin/bash

_jobarchitect_agent \
  --tool_path=scripts/analysis.py
  --input_dataset_path=example_dataset/ \
  --output_root=output/ \
  290d3f1a902c452ce1c184ed793b1d6b83b59164
```

Now that our tools make use of datasets we can write "smart" tools. The "smart" tools will work on datasets in a standardised fashion, i.e.

```
python scripts/analysis.py  \
  --dataset-path=path/to/dataset  \
  --identifier=identifier_hash  \
  --output-path=output_root
```

This removes the need for CWL. We can therefore take the pragmatic decision to trade the flexibility offered by CWL for simplicity. If we need CWL in the future we can work off the groundwork put into the 0.4.0 release.

Note: In the example above we do not use positional arguments. This is a design decision to make it easier to extend the tool in the future whilst remaining backwards compatible. Although this makes the tool a bit more difficult to run, we are not expecting to run this directly, it will be run programatically.

Removing CWL backend also means that we do not yet need to implement the second layer of abstraction (`agent.py`).

API documentation

jobarchitect

jobarchitect package.

jobarchitect.agent

Jobarchitect agent.

class `jobarchitect.agent.`**`Agent`**(*tool_path*, *dataset_path*, *output_root='/tmp'*)

Class to create commands to analyse data.

`run_tool_on_identifier`(*identifier*)

Run the tool on an item in the dataset.

`jobarchitect.agent.`**`analyse_by_identifiers`**(*tool_path*, *dataset_path*, *output_root*, *identifiers*)

Run analysis on identifiers.

Parameters

- **`tool_path`** – path to tool
- **`dataset_path`** – path to input dataset
- **`output_root`** – path to output root

Identifiers list of identifiers

`jobarchitect.agent.`**`cli`**()

Command line interface for _analyse_by_ids

jobarchitect.backends

Job output backends.

class `jobarchitect.backends.`**`JobSpec`**(*tool_path*, *dataset_path*, *output_root*, *hash_ids*, *image_name=None*)

Job specification class.

`dataset_path`

Return the dataset path.

`hash_ids`

Return the hash identifiers as a string.

`image_name`

Return the container image name.

`output_root`

Return the output root path.

`tool_path`

Return the path to the tool.

`jobarchitect.backends.`**`generate_bash_job`**(*jobspec*)

Return bash job script job as a string.

The script contains code to run all analysis on all data in one chunk from a split dataset.

Parameters **`jobspec`** – job specification as a `jobarchitect.JobSpec`

Returns bash job script as a string

`jobarchitect.backends.`**`generate_docker_job`**(*jobspec*)

Return docker job script as a string.

The script contains code to run a docker container to analyse data.

Parameters **`jobspec`** – job specification as a `jobarchitect.JobSpec`

Returns docker job script as a string

`jobarchitect.backends.`**`generate_singularity_job`**(*jobspec*)

Return singularity job script as a string.

The script contains code to run a docker container to analyse data.

Parameters **`jobspec`** – job specification as a `jobarchitect.JobSpec`

Returns docker job script as a string

`jobarchitect.backends.`**`render_script`**(*template_name*, *variables*)

Return script as a string.

jobarchitect.sketchjob

Tool to create jobs to carry out analyses on datasets.

class `jobarchitect.sketchjob.`**`JobSketcher`**(*tool_path*, *dataset_path*, *output_root*, *image_name=None*)

Class to build up jobs to analyse a dataset.

`sketch`(*backend*, *nchunks*)

Return generator yielding instances of `jobarchitect.JobSec`.

Parameters

- **`backend`** – backend function for generating job scripts
- **`nchunks`** – number of chunks the job should be split into

Returns generator yielding jobs as strings

`jobarchitect.sketchjob.`**`generate_jobspecs`**(*tool_path*, *dataset_path*, *output_root*, *nchunks*, *image_name=None*)

Return generator yielding instances of `jobarchitect.JobSec`.

Parameters

- **`tool_path`** – path to tool
- **`dataset_path`** – path to input dataset
- **`output_root`** – path to output root
- **`nchunks`** – number of chunks the job should be split into
- **`image_name`** – container image name

Returns generator yielding instances of `jobarchitect.JobSec`

`jobarchitect.sketchjob.`**`sketchjob`**(*tool_path*, *dataset_path*, *output_root*, *backend*, *nchunks*, *image_name=None*)

Return list of jobs as strings.

Parameters

- **`tool_path`** – path to tool
- **`dataset_path`** – path to input dataset
- **`output_root`** – path to output root
- **`backend`** – backend function for generating job scripts

- **nchunks** – number of chunks the job should be split into

Returns generator yielding jobs as strings

jobarchitect.utils

Utilities for jobarchitect.

`jobarchitect.utils.`**`mkdir_parents`**`(`*path*`)`

Create the given directory path.

This includes all necessary parent directories. Does not raise an error if the directory already exists.

Parameters **path** – path to create

`jobarchitect.utils.`**`output_path_from_hash`**`(`*dataset_path*, *hash_str*, *output_root*`)`

Return absolute output path for a dataset item.

A.k.a. the absolute path to which output data should be written for the datum specified by the given hash.

This function is not responsible for creating the directory.

Parameters

- **dataset_path** – path to input dataset
- **hash_str** – dataset item identifier as a hash string
- **output_root** – path to output root

Raises KeyError if hash string identifier is not in the dataset

Returns absolute output path for a dataset item specified by the identifier

`jobarchitect.utils.`**`path_from_hash`**`(`*dataset_path*, *hash_str*`)`

Return absolute path from a dataset given a hash.

Parameters

- **dataset_path** – path to input dataset
- **hash_str** – dataset item identifier as a hash string

Returns absolute path to dataset item

`jobarchitect.utils.`**`split_dataset`**`(`*dataset_path*, *nchunks*`)`

Return generator yielding lists of file entries.

Parameters

- **dataset_path** – path to input dataset
- **nchunks** – number of chunks the dataset items should be split into

Returns generator yielding lists of file entries

CHANGELOG

This project uses semantic versioning. This change log uses principles from keep a changelog.

[Unreleased]

Added

Changed

Deprecated

Removed

Fixed

Security

[0.6.0] - 2017-06-15

Added

- Agent now creates the output directory that it will pass to the "smart" tool via the `--output-directory`

Changed

- "Smart" tool interface parameter name changed from `--output-path` to `--output-directory`
- Singularity template

Fixed

- Naming of `sketchjob` input argument

[0.5.0] - 2017-06-15

Added

- Ability to access file level metadata from dataset items when running a Python analysis script

Changed

- Tools now need to comply with a specific command line interface to be able to run using scripts produced by `sketchjob`. The command line interface of such "smart" scripts should have no positional arguments and has three required named arguments: `--dataset-path`, `--identifier`, and `--output-path`. Furthermore, such "smart" analysis scripts should only work on one item in a dataset as it is the responsibility of `sketchjob` and the jobarchitect `agent` to split the dataset into individual jobs.
- For jobs that do not adhere to the command line interface above one will have to write thin Python wrappers, for examples have a look at the scripts in `tests/sample_smart_tools/`
- Requiring the use of "smart" tools removes the need to make use of CWL, so its support has been removed
- See the *Design* document for more details about the thought process that led to this redesign

Removed

- CWL support

[0.4.0] - 2017-06-08

Changed

- Now using CWL (Common Workflow Language) to specify program to be run on dataset.

[0.3.0] - 2017-02-07

Changed

- Move `JobSpec` from `jobarchitect` to `jobarchitect.backends` module
- `JobSpec` class now stores absolute paths
- Update Dockerfile to set entrypoint to sketchjob

[0.2.0] - 2017-02-06

Added

- singularity job command backend
- sketchjob CLI options for selecting wrapper script
- Templates for both job command and wrapper script creation
- `jobarchitect.backends.render_script` function
- Hosting of docs on readthedocs
- API documentation to sphinx generated docs
- Change log to sphinx generated docs
- Better docstrings

[0.1.0] - 2017-02-03

Added

- `sketchjob` CLI
- `_analyse_by_ids` CLI
- Support script and files to build docker image
- `jobarchitect.JobSpec` class
- `jobarchitect.agent.analyse_by_identifiers` function
- `jobarchitect.agent.Agent` class
- `jobarchitect.backends.generate_docker_job` backend

- `jobarchitect.sketchjob.generate_jobspecs` function
- `jobarchitect.sketchjob.sketch` function
- `jobarchitect.sketchjob.JobSketcher` class
- `jobarchitect.utils.mkdir_parents` function
- `jobarchitect.utils.output_path_from_hash` function
- `jobarchitect.utils.split_dataset` function
- `jobarchitect.utils.path_from_hash` function

Changed

Deprecated

Removed

Fixed

Security

Python Module Index

Index