RODEOS Ingest

Release 0+untagged.52.g0bb945b

Manuel Holtgrewe, Dieter Beule

Jan 20, 2021

MANUAL

The aim of the RODEOS (Raw Omics Data accEss and Organisation System) system is to facilitate the management and access to Omics raw mass data (e.g., genomics or proteomics data). The system itself is based on the iRODS ecosystem:

- iRODS for mass data storage and meta data management,
- Metalnx as a graphical user interface to iRODS, and
- Davrods for WebDAV based access to the data.

This is the documentation for the data ingest (aka **data import**) subsystem. That is, for the import of data into iRODS employing the irods_capability_automated_ingest (ICAI) library.

This documentation provides the following information:

- the implemented logic / workflow,
- the installation of the ingest services (that are based on Celery as ICAI is based on Celery),
- the configuration the module using environment variables, and
- how to properly call the importa via ICAI.

Where necessary, it provides hints on the configuration of external software such as Redis (an indirect dependency through Celery). However, for an overview of the "super" system RODEOS itself and the individual external libraries, please see the corresponding documentation.

CHAPTER

ONE

INSTALLATION

Please also refer to the the documentation of irods_capability_automated_ingest (ICAI).

1.1 Dependencies

1.1.1 Python 3

Python version 3.6 is required. There currently (January 2020) is a limitation in ICAI that prevents it from being run on Python 3.7 and above. Once this limitation has been removed, RODEOS Ingest will be able to run on Python 3.6 and above.

1.1.2 iRODS and iRODS iCommands

- You will need a working iRODS (catalogue/provider and resource) server(s).
- You must install iRODS iCommands on the machine that you want to run RODEOS Ingest on.
- The account that the RODEOS Ingest runs as must be properly setup and authenticated with the iRODS server. In other words, `ils` should work. Note well that PAM accounts will generally be affected by PAM login timeouts so you might want to use native authentication for such service users.

1.1.3 Redis

Redis is a key value store/database. ICAI is based on Celery and both Celery and ICAI directly use Redis for queue management and data caching. The default settings of redis in many systems is to have no limit on its memory which can lead to out of memory situations when ingesting many files. Make sure to adjust the following settings in `redis.conf`, below are sensible defaults.

```
maxmemory 1073741824             # 1GB, can be adjusted
maxmemory-policy allkeys-lru     # enable cache evication based on LRU
```

1.2 Steps

We recommend installing the software in a Python virtualenv.

Perform a checkout

```
# git clone git@github.com:bihealth/rodeos-ingest.git
```

Next install using `pip`.

```
# cd rodeos-ingest
# pip install -e .
```

CHAPTER

TWO

WORKER SERVICE

The next step is setting up the worker (which is based on the Celery work queue). You might want to create an appropriate service user for this on your ingest server (the user must be able to use the irods icommands without having to authenticate again). Below is an example systemd file (that could be placed into `/etc/systemd/system/rodeos-ingest.service`). Please refer to the systemd configuration on details for running this.

```
[Unit]
Description=RODEOS Ingest Celery Queue
AssertPathExists=/home/rodeos-ingest/.irods/irods_environment.json
AssertPathExists=/home/rodeos-ingest/.irods/.irodsA

[Service]
Type=forking
User=rodeos-ingest
Environment=CELERY_APP=irods_capability_automated_ingest.sync_task
Environment=CELERY_BIN=/opt/rodeos-ingest-env/bin/celery
Environment=CELERY_BROKER_URL=redis://127.0.0.1:6379/0
Environment=CELERYD_NODES=worker1
Environment=CELERYD_LOG_FILE="/var/log/rodeos-ingest/rodeos-ingest-%n%I.log"
Environment=CELERYD_PID_FILE="/var/run/rodeos-ingest/%n.pid"
Environment=CELERYD_LOG_LEVEL=INFO
Environment=CELERYD_OPTS=--concurrency=8
RuntimeDirectory=rodeos-ingest
WorkingDirectory=/home/rodeos-ingest
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} \
  ${CELERYD_OPTS}'
ExecStop=/bin/sh -c '${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} \
   ${CELERYD_OPTS}'

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

You could also run this manually using the following command. Make sure that you activate the appropriate virtualenv before.

```
/opt/rodeos-ingest-env/bin/celery \
    multi start \
    worker1 \
    -A irods_capability_automated_ingest.sync_task \
```

(continues on next page)

(continued from previous page)

```
    --logfile=path/to/logs-%n%I.log \
    --pidfile=path/to/pid-%n.pid \
    --loglevel=INFO \
    --concurrency=8
```

2.1 Configuration

Configuration is done using environment variables. As the Celery workers are actually performing the ingest work you have to update the environment variables when starting the `celery` executable.

You can find a documentation of the environment variables in *Settings*.

CHAPTER

THREE

RUNNING

You run the ingest from the directory `$SOURCE` to the destination collection `$DEST` as follows. Use the papropriate path to the handler module that you want to use.

```
/opt/rodeos-ingest-env/bin/python \
    -m irods_capability_automated_ingest.irods_sync \
    start \
        --event_handler rodeos_ingest.genomics.illumina.bcl \
        --job_name rodeos-ingest-$(date +%Y-%m-%d_%H-%M-%S) \
         \
        --exclude_file_name ".*_MANIFEST_*.txt"  \
        -- \
        $SOURCE \
        $DEST
```

Please refer to the ICAI documentation (or the output when including `--help`) for more information.

The following handler modules are provided.

module	data type
`rodeos_ingest.genomics.illumina.bcl`	Illumina run folders
`rodeos_ingest.genomics.illumina.fastq`	FASTQ demultiplexing results

CHAPTER

FOUR

COMMON WORKFLOW

Generally, ingest works by scanning a **source** directory (`${SOURCE}`) and transferring it into an iRODS **destination** (`${DEST}`) collection. Each entry of the source directory corresponds to a data set (e.g., a directory or files).

Ingest can be run in a one-time or a repeated/periodic fashion. Each time that the ingest work is done is called a **job**.

4.1 Common Multi-File Workflow

The following workflow has been implemented in a generic fashion and is reused in the case of multi-file data sets. Such data sets consist of one directory that contains (meta) data files and also marker files that indicate the status of the data by their presence or content.

before each

job for each directory `${ENTRY}` in `${SOURCE}`:

- a corresponding collection `${DEST}/${ENTRY}` is created in iRODS if it does not exist
- the meta data `rodeos::ingest::first_seen` is set to the current date and time

for each file in the source directory

- the file is created/updated by the `irods_capability_automated_ingest` functionality
- the collection `${DEST}/${ENTRY}` gets its meta data `rodeos::ingest::last_update` set to the current date and time
- the checksum of the file is registered in iRODS using `ichksum`

after each job

for each directory `${ENTRY}` in `${SOURCE}`:

- if `${DEST}/${ENTRY}` is checked whether it exists in iRODS and is considered as done and skipped if not so; the detection of whether `${ENTRY}` is done is functionality implemented in the specialization of the common workflow
- if `${DEST}/${ENTRY}` has its last update after a certain period of time and is considered at rest; if not then it it is skipped
- a call to `ichksum -r` ensures that all files in `${DEST}/${ENTRY}` have checksums
- a manifest file (listing all files below `${SOURCE}/${ENTRY}` with their size in bytes and checksum; excluding the manifest file of course) is created for the local directory using the `hashdeep` tool
- a corresponding manifest file is created using the files in the iRODS catalogue and the checksum known to iRODS

- the local and iRODS manifest files are compared (semantically, their content will not be byte identically) and the process is stopped if they are not equal
- both files are uploaded into iRODS (and get their checksum computed)
- **the folder `${SOURCE}/${ENTRY}` is moved to `${SOURCE}-INGESTED/${ENTRY}`**
  - this explicitely and verbosely marks the process as done to the user
  - the data generation instrument can be given access only to `${SOURCE}` such that it only has access to the data during generation but not afterwards; thus access to the instrument only grants access to the currently created data set but not the backcatalogue

CHAPTER

FIVE

BCL WORKFLOW

This workflow implements ingest for directories as created by Illumina sequencing machines with BCL (base call) files. The overall pattern follows the one described in the section *Common Multi-File Workflow*. The specialization are:

for each file in the source directory

- in addition to the steps described in the common workflow,
- if the file is the run info or run parameters XML file then meta data is appropriately extracted from the XML file and applied to `${DEST}/${ENTRY}`
- if the file is a netcopy complete file then the timestamp is extracted and applied to `${DEST}/${ENTRY}`

done detection

- is implemented by the presence of the appropriate marker files depending on the Illumina device type and version
- the device type and version is detected by logic analyzing the run info and run parameters XML files

CHAPTER

SIX

FASTQ WORKFLOW

This workflow is used for ingesting the result of the base call to sequence conversion step (sometimes also referred to as “demultiplexing” of raw Illumina base call data). The overall pattern follows the one described in the section *Common Multi-File Workflow*. The specialization are:

done file detection

- is performed by the presence of a marker file configured by `RODEOS_ILLUMINA_FASTQ_DONE_MARKER_FILE`

CHAPTER

SEVEN

DEVELOPER OVERVIEW

Contributions are welcome via GitHub pull requests. Continuous integration (CI) has been setup and it is expected that patches are accompanied with appropriate tests. Further, static analysis and coverage analysis via Codacy is integrated via CI and it is expected that code does not strongly decrease code coverage or introduce true issues detected by static analysis.

7.1 Commit Messages

Prefix your commit messages with 3 letter emojis as documented here.

- https://robinpokorny.github.io/git3moji/

7.2 Development Setup

The following currently is a sketch only.

- install redis or run via docker
- add `redis` to `/etc/hosts` with appropriate IP (e.g., localhost)
- install irods or run via docker
- add `irods` to `/etc/hosts` with appropriate IP (e.g., localhost)
- setup `.irods/irods_environment.json` file, e.g.

```
{
    "irods_host": "irods",
    "irods_port": 1247,
    "irods_authentication_scheme": "NATIVE",
    "irods_default_hash_scheme": "MD5",
    "irods_zone_name": "tempZone",
    "irods_user_name": "rods",
    "irods_password": "rods"
}
```

CHAPTER

EIGHT

SETTINGS

The text in this seciton doubles as API documentation for the `rodeos_ingest.settings` module and the environment variable to configure RODEOS Ingest.

Configuration settings.

The value of the configuration environment variables have to be parseable to the types given for the variables below. In the case of `bool` variables, any string whose lower case value equals `1`, `true`, or `yes` will be converted into `True`, any other value will be convered into `False`.

The values shown below are the defaults if the environment variable remains unset.

`rodeos_ingest.settings.`**`RODEOS_DELAY_UNTIL_AT_REST_SECONDS: int = 300`**
Set how many seconds should pass before data is detected to be at rest.

`rodeos_ingest.settings.`**`RODEOS_HASHDEEP_ALGO: str = 'md5'`**
Algorithm to use for hashing in `hashdeep``.

`rodeos_ingest.settings.`**`RODEOS_HASHDEEP_THREADS: int = 8`**
Number of threads to use in `hashdeep``.

`rodeos_ingest.settings.`**`RODEOS_ILLUMINA_FASTQ_DONE_MARKER_FILE: str = 'DIGESTIFLOW_DEMUX_DO`**
Name of the "done" marker file for Illumina demultiplexing ingest.

`rodeos_ingest.settings.`**`RODEOS_LOOK_FOR_EXECUTABLES: bool = False`**
Whether or not to look for external dependency.

`rodeos_ingest.settings.`**`RODEOS_MANIFEST_IRODS: str = '_MANIFEST_IRODS.txt'`**
File name for iRODS manifest file.

`rodeos_ingest.settings.`**`RODEOS_MANIFEST_LOCAL: str = '_MANIFEST_LOCAL.txt'`**
File name for local manifest file.

`rodeos_ingest.settings.`**`RODEOS_MOVE_AFTER_INGEST: bool = True`**
Whether or not to move directories in ingest after completing item.

CHAPTER

NINE

COMMON

This section contains the API documentation fo the `rodeos_ingest.common` module and sub modules. The intended reader are developers interested in the source code of RODEOS Ingest. Readers that only want to install and/or use the software are probably not interested in it.

Common code for the omics ingest.

`rodeos_ingest.common.`**`KEY_FIRST_SEEN = 'rodeos::ingest::first_seen'`**
AVU key to use for `first_seen` attribute.

`rodeos_ingest.common.`**`KEY_LAST_UPDATE = 'rodeos::ingest::last_update'`**
AVU key to use for `last_update` attribute.

`rodeos_ingest.common.`**`KEY_MANIFEST_MESSAGE = 'rodeos::ingest::manifest_message'`**
AVU key with manifest detailed message

`rodeos_ingest.common.`**`KEY_MANIFEST_STATUS = 'rodeos::ingest::manifest_status'`**
AVU key for manifest status

`rodeos_ingest.common.`**`KEY_STATUS = 'rodeos::ingest::status'`**
AVU key to use destionation run folder ingestion status.

`rodeos_ingest.common.`**`compute_irods_manifest`**(*dst_collection*, *logger*, *src_folder*)
Compute manifest from irods checksums.

`rodeos_ingest.common.`**`compute_local_manifest`**(*logger*, *src_folder*)
Compute local hashdeep manifest.

`rodeos_ingest.common.`**`post_job`**(*hdlr_mod*, *logger*, *meta*, *is_folder_done: Callable[[Union[pathlib.Path, str]], bool]*, *delay_until_at_rest*)
Move completed run folders into the "ingested" area.

`rodeos_ingest.common.`**`pre_job`**(*hdlr_mod*, *logger*, *meta*)
Set the `first_seen` meta data value.

`rodeos_ingest.common.`**`refresh_last_update_metadata`**(*logger*, *session*, *meta*)
Update the `last_update` and `status` meta data value.

`rodeos_ingest.common.`**`run_ichksum`**(*irods_path: str*, *recurse: bool = False*) → None
Run `ichksum $irods_path`.

`rodeos_ingest.common.`**`to_ingested_path`**(*orig_path: Union[str, pathlib.Path]*) → pathlib.Path
Convert a run folder path to an "ingested" path.

CHAPTER

TEN

GENOMICS INGEST

This section contains the API documentation fo the `rodeos_ingest.genomics` module and sub modules. The intended reader are developers interested in the source code of RODEOS Ingest. Readers that only want to install and/or use the software are probably not interested in it.

10.1 Illumina BCL Ingest

10.2 Illumina FASTQ Ingest

10.3 Illumina Run Folder Parsing

PYTHON MODULE INDEX

INDEX