fabric8-analytics-rudra Documentation

Release 0.1

Ravindra Singh Ratnawat

May 12, 2020

CONTENTS:

CHAPTER

ONE

INDICES AND TABLES

- genindex
- modindex
- search

Data Store and Retrieval from various Storage.

Basic interface to the Amazon S3.

class rudra.data_store.aws.**AmazonEmr**(**args*, ***kwargs*)

Bases: *rudra.data_store.aws.AmazonS3*

Basic interface to the Amazon EMR.

connect()

Connect to the emr instance.

disconnect()

Close the connection to S3 database.

get_status(*cluster_id*)

Get the status of EMR Instance.

is_connected()

Check if the connection to database has been established.

run_flow(*configs*)

Run emr job flow.

terminate_jobs(*jobs*)

Terminate emr job.

class rudra.data_store.aws.**AmazonS3**(*aws_access_key_id=None*, *aws_secret_access_key=None*, *bucket_name=None*, *region_name=None*, *use_ssl=False*, *encryption=None*, *versioned=None*, *local_dev=False*, *endpoint_url=None*)

Bases: rudra.data_store.abstract_data_store.AbstractDataStore

Basic interface to the Amazon S3.

connect()

Connect to the S3 database.

disconnect()

Close the connection to S3 database.

get_name()

Get name of this object's bucket.

is_connected()
Check if the connection to database has been established.

list_bucket_keys()
List all the keys in bucket.

list_bucket_objects(*prefix=None*)
List all the objects in bucket.

load_matlab_multi_matrix(*s3_path*)
Load a '.mat'file & return a dict representation.

S3_path The path of the object in the S3 bucket.

Returns A dict containing numpy matrices against the keys of the multi-matrix.

object_exists(*object_key*)
Check if the there is an object with the given key in bucket, does only HEAD request.

read_generic_file(*filename*)
Retrieve remote object content.

read_json_file(*filename*)
Read JSON file from the S3 bucket.

read_pickle_file(*filename*)
Read Pickle file from the S3 bucket.

read_yaml_file(*filename*)
Read Yaml file from the S3 bucket.

s3_clean_bucket()
Clean the bucket.

s3_delete_object(*object_key*)
Delete a object in bucket.

s3_delete_objects(*object_keys*)
Delete a object in bucket.

s3_upload_folder(*folder_path*, *prefix=''*)
Upload(Sync) a folder to S3.

Folder_path The local path of the folder to upload to s3

Prefix The prefix to attach to the folder path in the S3 bucket

store_blob(*blob*, *object_key*)
Store blob onto S3.

upload_file(*src*, *target*)
Upload file into S3 Bucket.

write_json_file(*filename*, *contents*)
Write JSON file into S3 bucket.

write_pickle_file(*filename*, *contents*)
Write Pickle file into S3 bucket.

exception rudra.data_store.aws.**NotFoundAccessKeySecret**
Bases: Exception

Exception for invalid AWS secret/key.

Local data_store interface.

class `rudra.data_store.local_data_store.`**`LocalDataStore`**(*src_dir*)
Bases: `rudra.data_store.abstract_data_store.AbstractDataStore`

Wrapper on local filesystem, API similar to s3DataStore.

`get_name`()
Return name of local filesystem root dir.

`load_matlab_multi_matrix`(*local_filename*)
Load a '.mat'file & return a dict representation.

Local_filename The path of the object.

Returns A dict containing numpy matrices against the keys of the multi-matrix.

`read_generic_file`(*filename*)
Read a file and return its contents.

`read_json_file`(*filename*)
Read JSON file from the data_input source.

`read_pickle_file`(*filename*)
Read Pickle file from the data_input source.

`read_yaml_file`(*filename*)
Read Yaml file from the data_input source.

`upload_file`()
Upload file to a data store.

`write_json_file`()
Write json file to data store.

Google Bigquery data collection implementation.

Implementation Bigquery builder base.

class `rudra.data_store.bigquery.base.`**`BigqueryBuilder`**(*query_job_config=None*)
Bases: `object`

BigqueryBuilder class Implementation.

`get_result`(*job_id=None*, *job_query_obj=None*)
Get the result of the job.

`get_status`(*job_id*)
Get the job status of async query.

`run_query_async`()
Run the bigquery asynchronously.

`run_query_sync`()
Run the bigquery synchronously.

class `rudra.data_store.bigquery.base.`**`DataProcessing`**(*s3_client=None*)
Bases: `object`

Process the Bigquery Data.

`update_s3_bucket`(*data*, *bucket_name*, *filename='collated.json'*)
Upload s3 bucket.

Maven bigquery implementation.

class rudra.data_store.bigquery.maven_bigquery.**MavenBQDataProcessing**(*big_query_instance=None, s3_client=None, file_name='collated.json'*)

Bases: *rudra.data_store.bigquery.base.DataProcessing*

Implementation data processing for maven bigquery.

construct_packages(*content*)
Construct package list.

process()
Process Maven Bigquery response data.

class rudra.data_store.bigquery.maven_bigquery.**MavenBigQuery**(**args, **kwargs*)
Bases: *rudra.data_store.bigquery.base.BigqueryBuilder*

MavenBigQuery Implementation.

Npm bigquery implementation.

class rudra.data_store.bigquery.npm_bigquery.**NpmBQDataProcessing**(*big_query_instance=None, s3_client=None, file_name='collated.json'*)

Bases: *rudra.data_store.bigquery.base.DataProcessing*

Implementation data processing for npm bigquery.

construct_packages(*content*)
Construct package from content.

static handle_corrupt_packagejson(*content*)
Find dependencies from corrupted/invalid package.json.

process()
Process Npm Bigquery response data.

class rudra.data_store.bigquery.npm_bigquery.**NpmBigQuery**(**args, **kwargs*)
Bases: *rudra.data_store.bigquery.base.BigqueryBuilder*

NpmBigQuery Implementation.

Deployments scripts.

EMR Deployments.

class rudra.deployments.emr_scripts.**MavenEMR**
Bases: *rudra.deployments.emr_scripts.emr_script_builder.EMRScriptBuilder*

Maven Emr script implementation.

ecosystem = 'maven'

run_job(*input_dict*)
Run the emr job.

class rudra.deployments.emr_scripts.**NpmEMR**
Bases: *rudra.deployments.emr_scripts.emr_script_builder.EMRScriptBuilder*

NPM Emr script implementation.

ecosystem = 'npm'

run_job(*input_dict*)
Run the emr job.

class `rudra.deployments.emr_scripts.`**`PyPiEMR`**

Bases: *`rudra.deployments.emr_scripts.emr_script_builder.EMRScriptBuilder`*

PyPi Emr script implementation.

`ecosystem = 'pypi'`

`run_job`(*input_dict*)

Run the emr job.

Configurations for EMR instance.

class `rudra.deployments.emr_scripts.emr_config.`**`EMRConfig`**(*name, log_uri, ecosystem, s3_bootstrap_uri, training_repo_url, training_file_name='training/train.py', release_label='emr-5.10.0', instance_count=1, instance_type='m3.xlarge', applications=[{'Name': 'MXNet'}], visible_to_all_users=True, job_flow_role='EMR_EC2_DefaultRole', service_role='EMR_DefaultRole', properties={}, hyper_params='{}'*)

Bases: `object`

Config class for EMR.

`get_config`()

Get the config object.

`home_dir = '/home/hadoop'`

EMR script builder implementation.

class `rudra.deployments.emr_scripts.emr_script_builder.`**`EMRScriptBuilder`**

Bases: `rudra.deployments.emr_scripts.abstract_emr.AbstractEMR`

EMR Script implementation.

`construct_job`(*input_dict*)

Submit emr job.

`run_job`(*input_dict*)

Run the emr job.

EMR script implementation for the Maven service.

class `rudra.deployments.emr_scripts.maven_emr.`**`MavenEMR`**

Bases: *`rudra.deployments.emr_scripts.emr_script_builder.EMRScriptBuilder`*

Maven Emr script implementation.

`ecosystem = 'maven'`

`run_job`(*input_dict*)

Run the emr job.

EMR script implementation for the NPM service.

class rudra.deployments.emr_scripts.npm_emr.**NpmEMR**
Bases: *rudra.deployments.emr_scripts.emr_script_builder.EMRScriptBuilder*

NPM Emr script implementation.

ecosystem = 'npm'

run_job(*input_dict*)
Run the emr job.

EMR script implementation for the PYPI service.

class rudra.deployments.emr_scripts.pypi_emr.**PyPiEMR**
Bases: *rudra.deployments.emr_scripts.emr_script_builder.EMRScriptBuilder*

PyPi Emr script implementation.

ecosystem = 'pypi'

run_job(*input_dict*)
Run the emr job.

Package for various utils function.

Validation Utility module.

class rudra.utils.validation.**BQValidation**
Bases: object

Add validation for ecosystems.

validate_pypi(*content*)
Validate python packages.

Attributes:

content (str or [str] or {str}): list/set of packages or package str

Returns: [str]: list of valid packages.

Raises: ValueError: if content is not a type of str or list

rudra.utils.validation.**check_field_exists**(*input_data, fields*)
Check field exist in the input data.

rudra.utils.validation.**check_url_alive**(*url, accept_codes=[401]*)
Validate github repo exist or not.

rudra.utils.validation.**nn**(*name*)
Return a normalized name.

Utility helper functions.

class rudra.utils.helper.**CacheDict**(*max_len=1024*)
Bases: object

CacheDict implementation with max size limit.

rudra.utils.helper.**get_github_repo_info**(*repo_url*)
Get the github repository information.

rudra.utils.helper.**get_training_file_url**(*user, repo, branch='master', training_file_path='training/train.py'*)
Get the training file from the github repo.

rudra.utils.helper.**load_hyper_params**()
Load the hyper parameter from the command line args.

Mercator: implementation of dependencies finder.

class `rudra.utils.mercator.`**`SimpleMercator`**(*content*)

Bases: `object`

SimpleMercator Implementation.

class Dependency(*dep*)

Bases: `object`

Dependency class Implementation.

get_dependencies()

Get the list dependencies.

static handle_corrupt_pom(*content*)

Try to find the dependencies in corrupt/invalid pom.

PYTHON MODULE INDEX

INDEX