dataduct Documentation

Release 0.5.0

Coursera

April 02, 2016

Contents

Dataduct - DataPipeline for humans

Dataduct is a wrapper built on top of AWS Datapipeline which makes it easy to create ETL jobs. All jobs can be specified as a series of steps in a YAML file and would automatically be translated into datapipeline with appropriate pipeline objects.

Features include:

- Visualizing pipeline activities
- Extracting data from different sources such as RDS, S3, local files
- Transforming data using EC2 and EMR
- Loading data into redshift
- Transforming data inside redshift
- QA data between the source system and warehouse

It is easy to create custom steps to augment the DSL as per the requirements. As well as running a backfill with the command line interface.

Contents:

CHAPTER 1

Introduction

Dataduct is a wrapper built on top of AWS Datapipeline which makes it easy to create ETL jobs. All jobs can be specified as a series of steps in a YAML file and would automatically be translated into datapipeline with appropriate pipeline objects.

Features include:

- Visualizing pipeline activities
- Extracting data from different sources such as RDS, S3, local files
- Transforming data using EC2 and EMR
- Loading data into redshift
- Transforming data inside redshift
- QA data between the source system and warehouse

It is easy to create custom steps to augment the DSL as per the requirements. As well as running a backfill with the command line interface.

An example ETL from RDS would look like:

```
name: example_upsert
frequency: daily
load_time: 01:00  # Hour:Min in UTC

steps:
-   step_type: extract-rds
    host_name: test_host
    database: test_database
    sql: |
        SELECT *
        FROM test_table;

-   step_type: create-load-redshift
    table_definition: tables/dev.test_table.sql

-   step_type: upsert
    source: tables/dev.test_table.sql
    destination: tables/dev.test_table_2.sql
```

This would first perform an extraction from the RDS database with the `extract-rds` step using the `COPY ACTIVITY`. Then load the data into the `dev.test_table` in redshift with the `create-load-redshift`. Then perform an `upsert` with the data into the `test_table_2`.

CHAPTER 2

Installation

2.1 Installation using pip

Dataduct can easily be installed using pip with the following commands.

```
pip install dataduct
```

The major dependencies of dataduct are:

- `boto` greater than version 2.34, older versions are missing some of the functionality provided by EMR
- `PyYAML`
- `pandas`
- `psycopg2`
- `pytimeparse`
- `MySQL-python`
- `pyparsing`
- `testfixtures`

Ensure that a boto config file containing proper AWS credentials is present.

The visualizations are created using:

- `graphviz`
- `pygraphviz`

Autocomplete for the CLI is supported using:

- `argcomplete`

The documentation is created using:

- `sphinx`
- `sphinx-napolean`
- `sphinx_rtd_theme`

2.2 Installing in the developer environment

2.2.1 1. Clone the Repo

```
git clone https://github.com/coursera/dataduct.git
```

2.2.2 2. Update PATH and PYTHONPATH

Add these lines into your `.bash_profile` or `.zshrc` etc based on your shell type.

```
export PYTHONPATH=~/dataduct:$PYTHONPATH
export PATH=~/dataduct/bin:$PATH
```

2.2.3 3. Config

Create a config file. Instructions for this are provided in the config section.

2.3 Setup Autocomplete

Install argcomplete with `pip install argcomplete`.

If you're using `bash` then add the following to your `.bash_profile`:

```
eval "$(register-python-argcomplete dataduct)"
```

if you're using `zsh` then add the following line to your `.zshrc`:

```
autoload bashcompinit
bashcompinit
eval "$(register-python-argcomplete dataduct)"
```

CHAPTER 3

Commands

3.1 Pipeline Commands

Commands used to manipulate pipelines.

Usage:

```
dataduct pipeline [-h] {create,validate,activate,visualize}
```

Arguments:

- `-h, --help`: Show help message and exit.

3.1.1 Create, Validate, Activate

Usage:

```
dataduct pipeline {create,validate,activate}
    [-h] [-m MODE] [-f] [-t TIME_DELTA] [-b] [--frequency FREQUENCY]
    pipeline_definitions [pipeline_definitions ...]
```

- `create`: Creates a pipeline locally.
- `validate`: Creates a pipeline on Amazon DataPipeline and validates the pipeline.
- `activate`: Creates and validates the pipeline, then activates the pipeline.

Arguments:

- `-h, --help`: Show help message and exit.
- `-m MODE, --mode MODE`: Mode or config variables to use. e.g. `-m production`
- `-f, --force`: Destroy previous version of this pipeline, if they exist.
- `-t TIME_DELTA, --time_delta TIME_DELTA`: Timedelta the pipeline by x time difference. e.g. `-t "1 day"`
- `-b, --backfill`: Indicates that the timedelta supplied is for a backfill.
- `-frequency FREQUENCY`: Frequency override for the pipeline.
- `pipeline_definitions`: The YAML defintions of the pipeline.

3.1.2 Visualize

Visualizes the pipeline into a PNG file.

Usage:

```
dataduct pipeline visualize [-h] [-m MODE] [--activities_only]
                            filename pipeline_definitions
                            [pipeline_definitions ...]
```

Arguments:

- `-h --help`: Show help message and exit.
- `-m MODE, --mode MODE`: Mode or config variables to use. e.g. `-m production`
- `--activities_only`: Visualize pipeline activities only.
- `filename`: The filename for saving the visualization.
- `pipeline_definitions`: The YAML defintions of the pipeline.

3.2 Database Commands

Commands used to generate SQL for various actions.

3.2.1 Create, Drop, Grant, Recreate

- `create`: Generates SQL to create relations.
- `drop`: Generates SQL to drop relations.
- `grant`: Generates SQL to grant permissions.
- `recreate`: Generates SQL to recreate relations.

Usage:

```
dataduct database {create,drop,grant,recreate}
    [-h] [-m MODE] table_definitions [table_definitions ...]
```

Arguments:

- `-h, --help`: Show help message and exit.
- `-m MODE, --mode MODE`: Mode or config variables to use. e.g. `-m production`
- `table_definitions`: The SQL definitions of the relations.

3.2.2 Visualize

Creates an entity relationship diagram of the tables as a PNG file.

Usage:

```
dataduct database visualize [-h] [-m MODE]
                            filename table_definitions
                            [table_definitions ...]
```

Arguments:

- `-h, --help`: Show help message and exit.
- `-m MODE, --mode MODE`: Mode or config variables to use. e.g. `-m production`
- `filename`: The filename for saving the visualization.
- `table_definitions`: The SQL definitions of the tables.

3.3 Configuration Commands

Commands used to synchronize the config file from/to Amazon S3.

3.3.1 Sync To S3

Uploads the local config file to S3. Will automatically detect the location of the config file.See the config documentation for more information.

Usage:

```
dataduct config sync_to_s3 [-h] [-m MODE]
```

Arguments:

- `-h, --help`: Show help message and exit.
- `-m MODE, --mode MODE`: Mode or config variables to use. e.g. `-m production`

3.3.2 Sync From S3

Downloads the local config file from S3 and saves it to a file.

Usage:

```
dataduct config sync_from_s3 [-h] [-m MODE] filename
```

Arguments:

- `-h, --help`: Show help message and exit.
- `-m MODE, --mode MODE`: Mode or config variables to use. e.g. `-m production`
- `filename`: The filename for saving the config file.

3.4 SQL Shell Commands

Commands used to connect to either MySQL or Redshift via the terminal.

3.4.1 MySQL

Connects to a MySQL database using a host alias.

Usage:

```
dataduct sql_shell mysql [-h] [-m MODE] host_alias
```

Arguments:

- `-h, --help`: Show help message and exit.
- `-m MODE, --mode MODE`: Mode or config variables to use. e.g. `-m production`
- `host_alias`: The host alias of the database to connect to.

3.4.2 Redshift

Connects to the Redshift database specified in Dataduct configs.

Usage:

```
dataduct sql_shell redshift [-h] [-m MODE]
```

Arguments:

- `-h, --help`: Show help message and exit.
- `-m MODE, --mode MODE`: Mode or config variables to use. e.g. `-m production`

CHAPTER 4

Config

All the dataduct setting are controlled from a single config file that stores the credentials as well as different settings.

The config file is read from the following places in the specified order of priority.

1. `/etc/dataduct.cfg`
2. `~/.dataduct/dataduct.cfg`
3. `DATADUCT_CONFIG_PATH` environment variable

Minimum example config:

```
ec2:
    INSTANCE_TYPE: m1.large
    ETL_AMI: ami-05355a6c # Default AMI used by data pipeline - Python 2.6
    SECURITY_GROUP: FILL_ME_IN

emr:
    MASTER_INSTANCE_TYPE: m1.large
    NUM_CORE_INSTANCES: 1
    CORE_INSTANCE_TYPE: m1.large
    CLUSTER_AMI: 3.1.0

etl:
    S3_ETL_BUCKET: FILL_ME_IN
    ROLE: FILL_ME_IN
    RESOURCE_ROLE: FILL_ME_IN
```

4.1 Config Parameters

4.1.1 Bootstrap

```
bootstrap:
    ec2:
    -   step_type: transform
        command: echo "Welcome to dataduct"
        no_output: true
    emr:
    -   step_type: transform
        command: echo "Welcome to dataduct"
        no_output: true
```

Bootstrap steps are a chain of steps that should be executed before any other step in the datapipeline. This can be used to copy files from S3 or install libraries on the resource. At Coursera we use this to download some binaries from S3 that are required for some of the transformations.

Note that the EMR bootstrap is only executed on the master node. If you want to install something on the task nodes then you should use the bootstrap parameter in the `emr_cluster_config` in your datapipeline.

4.1.2 Custom Steps

```
custom_steps:
-   class_name: CustomExtractLocalStep
    file_path: custom_extract_local.py
    step_type: custom-extract-local
```

Custom steps are steps that are not part of dataduct but are created to augment the functionality provided by dataduct. At Coursera these are often Steps that Inherit from the current object but abstract out some of the functionality so that multiple pipelines don't have to write the same thing twice.

The file_path can be an absolute path or a relative path with respect to the `CUSTOM_STEPS_PATH` path defined in the ETL parameter field. The Step classes are dynamically imported based on the config and `step-type` field is the one that is matched when parsing the pipeline definition.

4.1.3 Database

```
database:
    permissions:
    -   user: admin
        permission: all
    -   group: consumer_group
        permission: select
```

Some steps such as `upsert` or `create-load-redshift` create tables and grant them appropriate permissions so that one does not have to create tables prior to running the ETL. The permission is the `permission` being granted on the table or view to the `user` or `group`. If both are specified then both the grant statements are executed.

4.1.4 EC2

Either Datapipeline can be used for instance management, or you can use an existing Worker Group. Worker groups have priority over Datapipeline instance management.

Using Datapipeline for instance management:

```
ec2:
    INSTANCE_TYPE: m1.small
    ETL_AMI: ami-05355a6c # Default AMI used by data pipeline - Python 2.6
    SECURITY_GROUP: FILL_ME_IN
```

The ec2 config controls the configuration for the ec2-resource started by the datapipeline. You can override these with `ec2_resouce_config` in your pipeline definition for specific pipelines.

Using Worker Groups:

```
ec2:
    WORKER_GROUP: MY_EC2_WORKER_GROUP_NAME
```

4.1.5 EMR

Either Datapipeline can be used for cluster management, or you can use an existing Worker Group. Worker groups have priority over Datapipeline cluster management.

Using Datapipeline for cluster management:

```
emr:
    CLUSTER_AMI: 3.1.0
    CLUSTER_TIMEOUT: 6 Hours
    CORE_INSTANCE_TYPE: m1.large
    NUM_CORE_INSTANCES: 1
    HADOOP_VERSION: 2.4.0
    HIVE_VERSION: null
    MASTER_INSTANCE_TYPE: m3.xlarge
    PIG_VERSION: null
    TASK_INSTANCE_BID_PRICE: null
    TASK_INSTANCE_TYPE: m1.large
```

The emr config controls the configuration for the emr-resource started by the datapipeline.

Using Worker Groups:

```
emr:
    WORKER_GROUP: MY_EMR_WORKER_GROUP_NAME
```

4.1.6 ETL

```
etl:
    CONNECTION_RETRIES: 2
    CUSTOM_STEPS_PATH: ~/dataduct/examples/steps
    DAILY_LOAD_TIME: 1
    KEY_PAIR: FILL_ME_IN
    MAX_RETRIES: 2
    NAME_PREFIX: dev
    QA_LOG_PATH: qa
    DP_INSTANCE_LOG_PATH: dp_instances
    DP_PIPELINE_LOG_PATH: dp_pipelines
    DP_QA_TESTS_LOG_PATH: dba_table_qa_tests
    RESOURCE_BASE_PATH: ~/dataduct/examples/resources
    RESOURCE_ROLE: FILL_ME_IN
    RETRY_DELAY: 10 Minutes
    REGION: us-east-1
    ROLE: FILL_ME_IN
    S3_BASE_PATH: dev
    S3_ETL_BUCKET: FILL_ME_IN
    SNS_TOPIC_ARN_FAILURE: null
    SNS_TOPIC_ARN_WARNING: null
    FREQUENCY_OVERRIDE: one-time
    DEPENDENCY_OVERRIDE: false
    HOOKS_BASE_PATH: ~/dataduct/examples/hooks
    TAGS:
        env:
            string: dev
        Name:
            variable: name
```

This is the core parameter object which controls the ETL at the high level. The parameters are explained below:

- CONNECTION_RETRIES: Number of retries for the database connections. This is used to eliminate some of the transient errors that might occur.
- CUSTOM_STEPS_PATH: Path to the directory to be used for custom steps that are specified using a relative path.
- DAILY_LOAD_TIME: Default time to be used for running pipelines
- KEY_PAIR: SSH key pair to be used in both the ec2 and the emr resource.
- MAX_RETRIES: Number of retries for the pipeline activities
- NAME_PREFIX: Prefix all the pipeline names with this string
- QA_LOG_PATH: Path prefix for all the QA steps when logging output to S3
- DP_INSTANCE_LOG_PATH: Path prefix for DP instances to be logged before destroying
- DP_PIPELINE_LOG_PATH: Path prefix for DP pipelines to be logged
- DP_QA_TESTS_LOG_PATH: Path prefix for QA tests to be logged
- RESOURCE_BASE_PATH: Path to the directory used to relative resource paths
- RESOURCE_ROLE: Resource role needed for DP
- RETRY_DELAY: Delay between each of activity retires
- REGION: Region to run the datapipeline from
- ROLE: Role needed for DP
- S3_BASE_PATH: Prefix to be used for all S3 paths that are created anywhere. This is used for splitting logs across multiple developer or across production and dev
- S3_ETL_BUCKET: S3 bucket to use for DP data, logs, source code etc.
- SNS_TOPIC_ARN_FAILURE: SNS to trigger for failed steps or pipelines
- SNS_TOPIC_ARN_WARNING: SNS to trigger for failed QA checks
- FREQUENCY_OVERRIDE: Override every frequency given in a pipeline with this unless overridden by CLI
- DEPENDENCY_OVERRIDE: Will ignore the dependency step if set to true.
- HOOKS_BASE_PATH: Path prefix for the hooks directory. For more information, see Hooks.
- Tags: Tags to be added to the pipeline. The first key is the Tag to be used, the second key is the type. If the type is string the value is passed directly. If the type is variable then it looks up the pipeline object for that variable.

4.1.7 Logging

```
logging:
    CONSOLE_DEBUG_LEVEL: INFO
    FILE_DEBUG_LEVEL: DEBUG
    LOG_DIR: ~/.dataduct
    LOG_FILE: dataduct.log
```

Settings for specifying where the logs should be outputted and debug levels that should be used in the library code execution.

4.1.8 MySQL

```
mysql:
    host_alias_1:
        HOST: FILL_ME_IN
        PASSWORD: FILL_ME_IN
        USERNAME: FILL_ME_IN
    host_alias_2:
        HOST: FILL_ME_IN
        PASSWORD: FILL_ME_IN
        USERNAME: FILL_ME_IN
```

Rds (MySQL) database connections are stored in this parameter. The pipeline definitions can refer to the host with the host_alias. `HOST` refers to the full db hostname inside AWS.

4.1.9 Redshift

```
redshift:
    CLUSTER_ID: FILL_ME_IN
    DATABASE_NAME: FILL_ME_IN
    HOST: FILL_ME_IN
    PASSWORD: FILL_ME_IN
    USERNAME: FILL_ME_IN
    PORT: FILL_ME_IN
```

Redshift database credentials that are used in all the steps that interact with a warehouse. `CLUSTER_ID` is the first word of the `HOST` as this is used by `RedshiftNode` at a few places to identify the cluster.

4.1.10 Modes

```
production:
    etl:
        S3_BASE_PATH: prod
```

Modes define override settings for running a pipeline. As config is a singleton we can declare the overrides once and that should update the config settings across all use cases.

In the example we have a mode called `production` in which the `S3_BASE_PATH` is overridden to `prod` instead of whatever value was specified in the defaults.

At coursera one of the uses for modes is to change between the dev redshift cluster to the production one when we deploy a new ETL.

CHAPTER 5

Creating an ETL

Dataduct makes it extremely easy to write ETL in Data Pipeline. All the details and logic can be abstracted in the YAML files which will be automatically translated into Data Pipeline with appropriate pipeline objects and other configurations.

5.1 Writing a Dataduct YAML File

To learn about general YAML syntax, please see YAML syntax. The structure of a Dataduct YAML file can be broken down into 3 parts:

- Header information
- Description
- Pipeline steps

Example:

```
# HEADER INFORMATION
name : example_emr_streaming
frequency : one-time
load_time: 01:00  # Hour:Min in UTC
topic_arn: 'arn:aws:sns:example_arn'
emr_cluster_config:
    num_instances: 1
    instance_size: m1.xlarge
    bootstrap:
        string: "s3://elasticmapreduce/bootstrap-actions/configure-hadoop,--yarn-key-value, yarn.sche

# DESCRIPTION
description : Example for the emr_streaming step

# PIPELINE STEPS
steps:
-   step_type: extract-local
    path: data/word_data.txt

-   step_type: emr-streaming
    mapper: scripts/word_mapper.py
    reducer: scripts/word_reducer.py

-   step_type: transform
    script: scripts/s3_profiler.py
```

```
    script_arguments:
    -   --input=INPUT1_STAGING_DIR
    -   --output=OUTPUT1_STAGING_DIR
    -   -f
```

5.2 Header Information

The header includes configuration information for Data Pipeline and the Elastic MapReduce resource.

The name field sets the overall pipeline name:

```
name : example_emr_streaming
```

The frequency represents how often the pipeline is run on a schedule basis. Currently supported intervals are *hourly, daily, one-time*:

```
frequency : one-time
```

The load time is what time of day (in UTC) the pipeline is scheduled to run. It is in the format of HH:MM so 01:00 would set the pipeline to run at 1AM UTC:

```
load_time: 01:00  # Hour:Min in UTC
```

In your config file, you have the option of specifying a default Amazon Resource Name that will be messaged if the pipeline fails, if you would wish to override this default ARN, you may use the topic_arn property:

```
topic_arn: 'arn:aws:sns:example_arn'
```

If the pipeline includes an EMR-streaming step, the EMR instance can be configured. For example, you can configure the bootstrap, number of core instances, and instance types:

```
emr_cluster_config:
    num_instances: 1
    instance_size: m1.xlarge
    bootstrap:
        string: "s3://elasticmapreduce/bootstrap-actions/configure-hadoop,--yarn-key-value, yarn.sche
```

Note: Arguments in the bootstrap step are delimited by commas, not spaces.

5.2.1 Description

The description allows the creator of the YAML file to clearly explain the purpose of the pipeline.

CHAPTER 6

Steps and Pipeline Objects

Pipeline objects are classes that directly translate one-one from the dataduct classes to DP objects. A step is an abstraction layer that can translate into one or more pipeline objects based on the action type. For example a `sql-command` step translates into a `sql-activity` or a `transform` step translates into `shell command activity` and creates an output `s3 node`.

6.1 Definition of a Step

A step is defined as a series of properties in yaml. For example,

```
-   step_type: extract-s3
    name: get_file
    file_uri: s3://elasticmapreduce/samples/wordcount/wordSplitter.py
```

defines an `extract-s3` step with properties `name` and `file_uri`.

6.2 Common

These are the properties that all steps possess.

- `step_type`: The step type. Must be either a pre-defined step or a custom step. (Required)
- `name`: The user-defined name of the step. Will show up as part of the component name in DataPipeline.
- `input_node`: See input and output nodes.
- `depends_on`: This step will not run until the step(s) specified have finished.

6.3 Extract S3

Extracts the contents from the specified file or directory in S3. May used as input to other steps.

6.3.1 Properties

One of: (Required)

- `file_uri`: The location of a single file in S3.

- `directory_uri`: The location of a directory in S3.

6.3.2 Example

```
-   step_type: extract-s3
    file_uri: s3://elasticmapreduce/samples/wordcount/wordSplitter.py
```

6.4 Extract Local

Extracts the contents from the specified file locally. May be used as input to other steps. May only be used with one-time pipelines.

6.4.1 Properties

- `path`: The location of a single file. (Required)

6.4.2 Example

```
-   step_type: extract-local
    path: data/example_file.tsv
```

6.5 Extract RDS

Extracts the contents of a table from an RDS instance. May be used as input to other steps. Data is stored in TSV format.

6.5.1 Properties

- `host_name`: The host name to lookup in the `mysql` section of the configuration file. (Required)
- `database`: The database in the RDS instance in which the table resides. (Required)
- `output_path`: Output the extracted data to the specified S3 path.

One of: (Required)

- `sql`: The SQL query to execute to extract data.
- `table`: The table to extract. Equivalent to a sql query of `SELECT * FROM table`.

6.5.2 Example

```
-   step_type: extract-rds
    host_name: maestro
    database: maestro
    sql: |
        SELECT *
        FROM example_rds_table;
```

6.6 Extract Redshift

Extracts the contents of a table from a Redshift instance. May be used as input to other steps. Data is stored in TSV format.

6.6.1 Properties

- `schema`: The schema of the table. (Required)
- `table`: The name of the table. (Required)
- `output_path`: Output the extracted data to the specified S3 path. Optional.

6.6.2 Example

```
-   step_type: extract-redshift
    schema: prod
    table: example_redshift_table
```

6.7 Transform

Runs a specified script on an resource.

6.7.1 Properties

- `output_node`: See input and output nodes.
- `script_arguments`: Arguments passed to the script.
- `script_name`: Required if `script_directory` is specified. Script to be executed in the directory.
- `additional_s3_files`: Additional files to include from S3.
- `output_path`: Save the script's output to the specified S3 path.
- `no_output`: If `true`, step will produce no extractable output. Default: `false`
- `resource_type`: If `ec2`, run step on the EC2 resource. If `emr`, run step on the EMR resource. Default: `ec2`

One of: (Required)

- `command`: A command to be executed directly.
- `script`: Local path to the script that should be executed.
- `script_directory`: Local path to a directory of scripts to be uploaded to the resource.

6.7.2 Example

```
-   step_type: transform
    script: scripts/example_script.py
    script_arguments:
    -   "--foo=bar"
```

6.8 SQL Command

Executes a SQL statement in a Redshift instance.

6.8.1 Properties

- `script_arguments`: Arguments passed to the SQL command.
- `queue`: Query queue that should be used.
- `wrap_transaction`: If `true`, SQL command will be wrapped inside a transaction. Default: `true`

One of: (Required)

- `command`: Command to be executed directly.
- `script`: Local path to the script that should be executed.

6.8.2 Example

```
-   step_type: sql-command
    command: SELECT * FROM dev.test_table;
```

6.9 EMR Streaming

Executes a map and an optional reduce script using Amazon Elastic MapReduce.

6.9.1 Properties

- `mapper`: Local path to the mapper script (Required)
- `reducer`: Local path to the reducer script
- `hadoop_params`: List of arguments to the hadoop command
- `output_path`: Save the script's output to the specified S3 path

6.9.2 Example

```
-   step_type: emr-streaming
    mapper: scripts/word_mapper.py
    reducer: scripts/word_reducer.py
```

6.10 Load Redshift

Loads the data from its input node into a Redshift instance.

6.10.1 Properties

- `schema`: The schema of the table. (Required)
- `table`: The name of the table. (Required)
- `insert_mode`: See Amazon's RedshiftCopyActivity documentation. Default: TRUNCATE
- `max_errors`: The maximum number of errors to be ignored during the load
- `replace_invalid_char`: Character to replace non-utf8 characters with

6.10.2 Example

```
-   step_type: load-redshift
    schema: dev
    table: example_table
```

6.11 Pipeline Dependencies

Keeps running until another pipeline has finished. Use with `depends_on` properties to stall the pipeline.

6.11.1 Properties

- `dependent_pipelines`: List of pipelines to wait for. (Required)
- `refresh_rate`: Time, in seconds, to wait between polls. Default: 300
- `start_date`: Date on which the pipelines started at. Default: Current day

6.11.2 Example

```
-   step_type: pipeline-dependencies
    refresh_rate: 60
    dependent_pipelines:
    -   example_transform
```

6.12 Create Load Redshift

Special transform step that loads the data from its input node into a Redshift instance. If the table it's loading into does not exist, the table will be created.

6.12.1 Properties

- `table_definition`: Schema file for the table to be loaded. (Required)
- `script_arguments`: Arguments for the runner.
  - `--max_error`: The maximum number of errors to be ignored during the load. Usage: `--max_error=5`

  - `--replace_invalid_char`: Character the replace non-utf8 characters with. Usage: `--replace_invalid_char='?'`
  - `--no_escape`: If passed, does not escape special characters. Usage: `--no_escape`
  - `--gzip`: If passed, compresses the output with gzip. Usage: `--gzip`
  - `--command_options`: A custom SQL string as the options for the copy command. Usage: `--command_options="DELIMITER '\t'"`
    * Note: If `--command_options` is passed, script arguments `--max_error`, `--replace_invalid_char`, `--no_escape`, and `--gzip` have no effect.

6.12.2 Example

```
-   step_type: create-load-redshift
    table_definition: tables/dev.example_table.sql
```

6.13 Load, Reload, Primary Key Check

Combine `create-load-redshift`, `reload` and `primary-key-check` into one single step.

6.13.1 Properties

- `staging_table_definition`: Intermidiate staging schema file for the table to be loaded into. (Required)
- `production_table_definition`: Production schema file for the table to be reloaded into. (Required)
- `script_arguments`: Arguments for the runner.
  - `--max_error`: The maximum number of errors to be ignored during the load. Usage: `--max_error=5`
  - `--replace_invalid_char`: Character the replace non-utf8 characters with. Usage: `--replace_invalid_char='?'`
  - `--no_escape`: If passed, does not escape special characters. Usage: `--no_escape`
  - `--gzip`: If passed, compresses the output with gzip. Usage: `--gzip`
  - `--command_options`: A custom SQL string as the options for the copy command. Usage: `--command_options="DELIMITER '\t'"`
    * Note: If `--command_options` is passed, script arguments `--max_error`, `--replace_invalid_char`, `--no_escape`, and `--gzip` have no effect.

6.13.2 Example

```
-   step_type: load-reload-pk
    staging_table_definition: tables/staging.example_table.sql
    production_table_definition: tables/dev.example_table.sql
    script_arguments:
    -   "--foo=bar"
```

6.14 Upsert

Extracts data from a Redshift instance and upserts the data into a table. Upsert = Update + Insert. If a row already exists (by matching primary keys), the row will be updated. If the row does not already exist, insert the row. If the table it's upserting into does not exist, the table will be created.

6.14.1 Properties

- `destination`: Schema file for the table to upsert into. (Required)
- `enforce_primary_key`: If true, de-duplicates data by matching primary keys. Default: true
- `history`: Schema file for the history table to record the changes in the destination table.
- `analyze_table`: If true, runs `ANALYZE` on the table afterwards. Default: true

One of: (Required)

- `sql`: The SQL query to run to extract data.
- `script`: Local path to a SQL query to run.
- `source`: The table to extract. Equivalent to a sql query of `SELECT * FROM source`.

6.14.2 Example

```
-   step_type: upsert
    source: tables/dev.example_table.sql
    destination: tables/dev.example_table_2.sql
```

6.15 Reload

Extracts data from a Redshift instance and reloads a table with the data. If the table it's reloading does not exist, the table will be created.

6.15.1 Properties

- `destination`: Schema file for the table to reload. (Required)
- `enforce_primary_key`: If true, de-duplicates data by matching primary keys. Default: true
- `history`: Schema file for the history table to record the changes in the destination table.
- `analyze_table`: If true, runs `ANALYZE` on the table afterwards. Default: true

One of: (Required)

- `sql`: The SQL query to run to extract data.
- `script`: Local path to a SQL query to run.
- `source`: The table to extract. Equivalent to a sql query of `SELECT * FROM source`.

6.15.2 Example

```
-   step_type: reload
    source: tables/dev.example_table.sql
    destination: tables/dev.example_table_2.sql
```

6.16 Create Update SQL

Creates a table if it exists and then runs a SQL command.

6.16.1 Properties

- `table_definition`: Schema file for the table to create. (Required)
- `script_arguments`: Arguments for the SQL script.
- `non_transactional`: If true, does not wrap the command in a transaction. Default: false
- `analyze_table`: If true, runs `ANALYZE` on the table afterwards. Default: true

One of: (Required)

- `command`: SQL command to execute directly.
- `script`: Local path to a SQL command to run.

6.16.2 Example

```
-   step_type: create-update-sql
    command: |
        DELETE FROM dev.test_table WHERE id < 0;
        INSERT INTO dev.test_table
        SELECT * FROM dev.test_table_2
        WHERE id < %s;
    table_definition: tables/dev.test_table.sql
    script_arguments:
    -   4
```

6.17 Primary Key Check

Checks for primary key violations on a specific table.

6.17.1 Properties

- `table_definition`: Schema file for the table to check. (Required)
- `script_arguments`: Arguments for the runner script.
- `log_to_s3`: If true, logs the output to a file in S3. Default: false

6.17.2 Example

```
-   step_type: primary-key-check
    table_definition: tables/dev.test_table.sql
```

6.18 Count Check

Compares the number of rows in the source and destination tables/SQL scripts.

6.18.1 Properties

- `source_host`: The source host name to lookup in the `mysql` section of the configuration file. (Required)
- `tolerance`: Tolerance threshold, in %, for the difference in count between source and destination. Default: 1
- `log_to_s3`: If true, logs the output to a file in S3. Default: false
- `script`: Replace the default count script.
- `script_arguments`: Arguments for the script.

One of: (Required)

- `source_sql`: SQL query to select rows to count for the source.
- `source_count_sql`: SQL query that returns a count for the source.
- `source_table_name`: Name of source table to count. Equivalent to a source_count_sql of `SELECT COUNT(1) from source_table_name`.

One of: (Required)

- `destination_sql`: SQL query to select rows to count for the destination.
- `destination_table_name`: Name of the destination table to count.
- `destination_table_definition`: Schema file for the destination table to count.

6.18.2 Example

```
-   step_type: count-check
    source_sql: "SELECT id, name FROM networks_network;"
    source_host: maestro
    destination_sql: "SELECT network_id, network_name FROM prod.networks"
    tolerance: 2.0
    log_to_s3: true
```

6.19 Column Check

Compares a sample of rows from the source and destination tables/SQL scripts to see if they match

6.19.1 Properties

- `source_host`: The source host name to lookup in the `mysql` section of the configuration file. (Required)
- `source_sql`: SQL query to select rows to check for the source. (Required)
- `sql_tail_for_source`: Statement to append at the end of the SQL query for the source
- `sample_size`: Number of samples to check. Default: 100
- `tolerance`: Tolerance threshold, in %, for mismatched rows. Default: 1
- `log_to_s3`: If true, logs the output to a file in S3. Default: false
- `script`: Replace the default column check script.
- `script_arguments`: Arguments for the script.

One of: (Required)

- `destination_sql`: SQL query to select rows to check for the destination.
- `destination_table_definition`: Schema file for the destination table to check.

6.19.2 Example

```
-   step_type: column-check
    source_sql: "SELECT id, name FROM networks_network;"
    source_host: maestro
    destination_sql: "SELECT network_id, network_name FROM prod.networks"
    sql_tail_for_source: "ORDER BY RAND() LIMIT LIMIT_PLACEHOLDER"
    sample_size: 10
    log_to_s3: true
```

CHAPTER 7

Input and Output Nodes

In dataduct, data is shared between two activities using S3. After a step is finished, it saves its output to a file in S3 for successive steps to read. Input and output nodes abstract this process, they represent the S3 directories in which the data is stored. A step's input node determines which S3 file it will read as input, and its output node determines where it will store its output. In most cases, this input-output node chain is taken care of by dataduct, but there are situations where you may want finer control over this process.

7.1 Input Nodes

The default behaviour of steps (except Extract- and Check-type steps) is to link its input node with the preceding step's output node. For example, in this pipeline snippet

```
-   step_type: extract-local
    path: data/test_table1.tsv

-   step_type: create-load-redshift
    table_definition: tables/dev.test_table.sql
```

the output of the `extract-local` step is fed into the `create-load-redshift` step, so the pipeline will load the data found inside `data/test_table1.tsv` into `dev.test_table.sql`. This behaviour can be made explicit through the `name` and `input_node` properties.

```
# This pipeline has the same behaviour as the previous pipeline.
-   step_type: extract-local
    name: extract_data
    path: data/test_table1.tsv

-   step_type: create-load-redshift
    input_node: extract_data
    table_definition: tables/dev.test_table.sql
```

When an input -> output node link is created, implicitly or explicitly, dependencies are created automatically between the two steps. This behaviour can be made explicit through the `depends_on` property.

```
# This pipeline has the same behaviour as the previous pipeline.
-   step_type: extract-local
    name: extract_data
    path: data/test_table1.tsv

-   step_type: create-load-redshift
    input_node: extract_data
```

```
    depends_on: extract_data
    table_definition: tables/dev.test_table.sql
```

You can use input nodes to communicate between steps that are not next to each other.

```
-   step_type: extract-local
    name: extract_data
    path: data/test_table1.tsv

-   step_type: extract-local
    path: data/test_table2.tsv

# This step will use the output of the first extract-local step (test_table1.tsv)
-   step_type: create-load-redshift
    input_node: extract_data
    table_definition: tables/dev.test_table.sql
```

Without the use of `input_node`, the `create-load-redshift` step would have used the data from `test_table2.tsv` instead.

You can also use input nodes to reuse the output of a step.

```
-   step_type: extract-local
    name: extract_data
    path: data/test_table1.tsv

-   step_type: create-load-redshift
    input_node: extract_data
    table_definition: tables/dev.test_table1.sql

-   step_type: create-load-redshift
    input_node: extract_data
    table_definition: tables/dev.test_table2.sql
```

Sometimes, you may not want a step to have any input nodes. You can specify this by writing `input_node: []`.

```
-   step_type: extract-local
    name: extract_data
    path: data/test_table1.tsv

# This step will not receive any input data
-   step_type: transform
    input_node: []
    script: scripts/example_script.py
```

If you are running your own script (e.g. through the Transform step), the input node's data can be found in the directory specified by the `INPUT1_STAGING_DIR` enviroment variable.

```
-   step_type: extract-local
    name: extract_data
    path: data/test_table1.tsv

# manipulate_data.py takes in the input directory as a script argument and
# converts the string into the enviroment variable.
-   step_type: transform
    script: scripts/manipulate_data.py
    script_arguments:
    -   --input=INPUT1_STAGING_DIR
```

7.2 Output Nodes

Dataduct usually handles a step's output nodes automatically, saving the file into a default path in S3. You can set the default path through your dataduct configuration file. However, some steps also have an optional `output_path` property, allowing you to choose an S3 directory to store the step's output.

7.2.1 Transform Step and Output Nodes

Transform steps allow you to run your own scripts. If you want to save the results of your script, you can store data into the output node by writing to the directory specified by the `OUTPUT1_STAGING_DIR` enviroment variable.

```
# generate_data.py takes in the output directory as a script argument and
# converts the string into the enviroment variable.
-   step_type: transform
    script: scripts/generate_data.py
    script_arguments:
    -   --output=OUTPUT1_STAGING_DIR

-   step_type: create-load-redshift
    table_definition: tables/dev.test_table.sql
```

You may wish to output more than one set of data for multiple proceeding steps to use. You can do this through the `output_node` property.

```
-   step_type: transform
    script: scripts/generate_data.py
    script_arguments:
    -   --output=OUTPUT1_STAGING_DIR
    output_node:
    -   foo_data
    -   bar_data

-   step_type: create-load-redshift
    input_node: foo_data
    table_definition: tables/dev.test_table1.sql

-   step_type: create-load-redshift
    input_node: bar_data
    table_definition: tables/dev.test_table2.sql
```

In this case, the script must save data to subdirectories with names matching the output nodes. In the above example, `generate_data.py` must save data in `OUTPUT1_STAGING_DIR/foo_data` and `OUTPUT1_STAGING_DIR/bar_data` directories. If the subdirectory and output node names are mismatched, the output nodes will not be generated correctly.

CHAPTER 8

Hooks

Dataduct has some endpoints you can use to execute python scripts before and after certain events when using the CLI and library locally.

8.1 Available Hooks

- `activate_pipeline`, which hooks onto the `activate_pipeline` function in `dataduct.etl.etl_actions`.
- `connect_to_redshift`, which hooks onto the `redshift_connection` function in `dataduct.data_access`.

8.2 Creating a hook

Dataduct tries to find available hooks by searching in the directory specified by the `HOOKS_BASE_PATH` config variable in the `etl` section, matching them by their filename. For example, a hook for the `activate_pipeline` endpoint would saved as `activate_pipeline.py` in that directory.

Each hook has two endpoints: `before_hook` and `after_hook`. To implement one of these endpoints, you declare them as functions inside the hook. You may implement only one or both endpoints per hook.

`before_hook` is called before the hooked function is executed. The parameters passed into the hooked function will also be passed to the `before_hook`. The `before_hook` is designed to allow you to manipulate the arguments of the hooked function before it is called. At the end of the `before_hook`, return the `args` and `kwargs` of the hooked function as a tuple.

Example `before_hook`:

```
# hooked function signature:
# def example(arg_one, arg_two, arg_three='foo')

def before_hook(arg_one, arg_two, arg_three='foo'):
    return [arg_one + 1, 'hello world'], {'arg_three': 'bar'}
```

`after_hook` is called after the hooked function is executed. The result of the hooked function is passed into `after_hook` as a single parameter. The `after_hook` is designed to allow you to access or manipulate the result of the hooked function. At the end of the `after_hook`, return the (modified) result of the hooked function.

Example `after_hook`:

```
# hooked function result: {'foo': 1, 'bar': 'two'}

def after_hook(result):
    result['foo'] = 2
    result['bar'] = result['bar'] + ' three'
    return result
```

CHAPTER 9

Code documentation

9.1 dataduct.config package

9.1.1 Subpackages

dataduct.config.tests package

Submodules

dataduct.config.tests.test_credentials module

Module contents

9.1.2 Submodules

9.1.3 dataduct.config.config module

9.1.4 dataduct.config.config_actions module

9.1.5 dataduct.config.constants module

9.1.6 dataduct.config.credentials module

9.1.7 dataduct.config.logger_config module

9.1.8 Module contents

9.2 dataduct.data_access package

9.2.1 Submodules

9.2.2 dataduct.data_access.connection module

9.2.3 Module contents

9.3 dataduct.database package

9.3.1 Subpackages

dataduct.database.parsers package

dataduct.database.parsers.tests package

Submodules

dataduct.database.parsers.tests.test_create_table module

dataduct.database.parsers.tests.test_create_view module

dataduct.database.parsers.tests.test_select_query module

dataduct.database.parsers.tests.test_transfrom module

Module contents

Submodules

dataduct.database.parsers.create_table module

dataduct.database.parsers.create_view module

dataduct.database.parsers.helpers module

dataduct.database.parsers.select_query module

dataduct.database.parsers.transform module

dataduct.database.parsers.utils module

Module contents

dataduct.database.sql package

Subpackages

dataduct.database.sql.tests package

Submodules

dataduct.database.sql.tests.test_sql_script module

dataduct.database.sql.tests.test_sql_statement module

dataduct.database.sql.tests.test_sql_utils module

Module contents

Submodules

dataduct.database.sql.sql_script module

dataduct.database.sql.sql_statement module

dataduct.database.sql.transaction module

dataduct.database.sql.utils module

Module contents

dataduct.database.tests package

Submodules

dataduct.database.tests.test_database module

dataduct.database.tests.test_history_table module

Module contents

9.3.2 Submodules

9.3.3 dataduct.database.column module

9.3.4 dataduct.database.database module

9.3.5 dataduct.database.history_table module

9.3.6 dataduct.database.relation module

9.3.7 dataduct.database.select_statement module

9.3.8 dataduct.database.table module

9.3.9 dataduct.database.view module

9.3.10 Module contents

9.4 dataduct.etl package

9.4.1 Subpackages

dataduct.etl.tests package

Submodules

dataduct.etl.tests.test_etl_actions module

dataduct.etl.tests.test_etl_pipeline module

Module contents

CHAPTER 10

Indices and tables

- genindex
- modindex
- search