
data-pipelines-cli

GetInData

Apr 19, 2022

CONTENTS:

1	Installation	3
	Python Module Index	33
	Index	35

INSTALLATION

Use the package manager `pip` to install `dp` (data-pipelines-cli):

```
pip install data-pipelines-cli
```

1.1 Usage

First, create a repository with a global configuration file that you or your organization will be using. The repository should contain `dp.yml.tpl` file looking similar to this:

```
templates:
  my-first-template:
    template_name: my-first-template
    template_path: https://github.com/<YOUR_USERNAME>/<YOUR_TEMPLATE>.git
vars:
  username: YOUR_USERNAME
```

Thanks to the `copier`, you can leverage Jinja template syntax to create easily modifiable configuration templates. Just create a `copier.yml` file next to the `dp.yml.tpl` one and configure the template questions (read more at [copier documentation](#)).

Then, run `dp init <CONFIG_REPOSITORY_URL>` to initialize **dp**. You can also drop `<CONFIG_REPOSITORY_URL>` argument, **dp** will get initialized with an empty config.

1.1.1 Project creation

You can use `dp create <NEW_PROJECT_PATH>` to choose one of the templates added before and create the project in the `<NEW_PROJECT_PATH>` directory.

You can also use `dp create <NEW_PROJECT_PATH> <LINK_TO_TEMPLATE_REPOSITORY>` to point directly to a template repository. If `<LINK_TO_TEMPLATE_REPOSITORY>` proves to be the name of the template defined in **dp**'s config file, `dp create` will choose the template by the name instead of trying to download the repository.

`dp template-list` lists all added templates.

1.1.2 Project update

To update your pipeline project use `dp update <PIPELINE_PROJECT-PATH>`. It will sync your existing project with updated template version selected by `--vcs-ref` option (default HEAD).

1.1.3 Project configuration

dp as a tool depends on a few files in your project directory. In your project directory, it must be able to find a `config` directory with a structure looking similar to this:

```
config
├── base
│   ├── dbt.yml
│   ├── bigquery.yml
│   └── ...
├── dev
│   └── bigquery.yml
├── local
│   ├── dbt.yml
│   └── bigquery.yml
└── prod
    └── bigquery.yml
```

Whenever you call **dp**'s command with the `--env <ENV>` flag, the tool will search for `dbt.yml` and `<TARGET_TYPE>.yml` files in `base` and `<ENV>` directory and parse important info out of them, with `<ENV>` settings taking precedence over those listed in `base`. So, for example, for the following files:

```
# config/base/dbt.yml
target: env_execution
target_type: bigquery

# config/base/bigquery.yml
method: oauth
project: my-gcp-project
dataset: my-dataset
threads: 1

# cat config/dev/bigquery.yml
dataset: dev-dataset
```

`dp test --env dev` will run `dp test` command using values from those files, most notably with `dataset: dev-dataset` overwriting `dataset: my-dataset` setting.

dp synthesizes `dbt's profiles.yml` out of those settings among other things. However, right now it only creates `local` or `env_execution` profile, so if you want to use different settings amongst different environments, you should rather use `{{ env_var('VARIABLE') }}` as a value and provide those settings as environment variables. E.g., by setting those in your `config/<ENV>/k8s.yml` file, in `envs` dictionary:


```
# config/base/bigquery.yml
method: oauth
dataset: "{{ env_var('GCP_DATASET') }}"
project: my-gcp-project
threads: 1

# config/base/execution_env.yml
# ... General config for execution env ...

# config/base/k8s.yml
# ... Kubernetes settings ...

# config/dev/k8s.yml
envs:
  GCP_DATASET: dev-dataset

# config/prod/k8s.yml
envs:
  GCP_DATASET: prod-dataset
```

target and target_type

- target setting in config/<ENV>/dbt.yml should be set either to local or env_execution;
- target_type defines which backend dbt will use and what file **dp** will search for; example target_types are bigquery or snowflake.

Variables

You can put a dictionary of variables to be passed to dbt in your config/<ENV>/dbt.yml file, following the convention presented in [the guide at the dbt site](#). E.g., if one of the fields of config/<SNOWFLAKE_ENV>/snowflake.yml looks like this:

```
schema: "{{ var('snowflake_schema') }}"
```

you should put the following in your config/<SNOWFLAKE_ENV>/dbt.yml file:

```
vars:
  snowflake_schema: EXAMPLE_SCHEMA
```

and then run your `dp run --env <SNOWFLAKE_ENV>` (or any similar command).

You can also add “global” variables to your **dp** config file `$HOME/.dp.yml`. Be aware, however, that those variables get erased on every `dp init` call. It is a great idea to put *commonly used* variables in your organization’s `dp.yml.tpl` template and make **copier** ask for those when initializing **dp**. By doing so, each member of your organization will end up with a list of user-specific variables reusable across different projects on its machine. Just remember, **global-scoped variables take precedence over project-scoped ones**.

1.1.4 dbt sources and models creation

With the help of `dbt-codegen` and `dbt-profiler`, one can easily generate `source.yml`, source's base model SQLs, and model-related YAMLs. **dp** offers a convenient CLI wrapper around those functionalities.

First, add the **dbt-codegen** package to your `packages.yml` file:

```
packages:
- package: dbt-codegen
  version: 0.5.0 # or newer
```

Then, run `dp generate source-yaml YOUR_DATASET_NAME` to generate `source.yml` file in `models/source` directory. You can list more than one dataset, divided by space. After that, you are free to modify this file.

When you want to generate SQLs for your sources, run `dp generate source-sql`. It will save those SQLs in the directory `models/staging/YOUR_DATASET_NAME`.

Finally, when you have all your models prepared (in the form of SQLs), run `dp generate model-yaml MODELS_DIR` to generate YAML files describing them (once again, you are not only free to modify them but also encouraged to do so!). E.g., given such a directory structure:

```
models
├── staging
│   ├── my_source
│   │   ├── stg_table1.sql
│   │   └── stg_table2.sql
│   └── intermediate
│       ├── intermediate1.sql
│       ├── intermediate2.sql
│       └── intermediate3.sql
└── presentation
    └── presentation1.sql
```

`dp generate model-yaml models/` will create `models/staging/my_source/my_source.yml`, `models/staging/intermediate/intermediate.yml`, and `models/presentation/presentation.yml`. Beware, however, this command WILL NOT WORK if you do not have those models created in your data warehouse already. So remember to run `dp run` (or a similar command) beforehand.

If you add the **dbt-profiler** package to your `packages.yml` file too, you can call `dp generate model-yaml --with-meta MODELS_DIR`. **dbt-profiler** will add a lot of profiling metadata to descriptions of your models.

1.1.5 Project compilation

`dp compile` prepares your project to be run on your local machine and/or deployed on a remote one.

1.1.6 Local run

When you get your project configured, you can run `dp run` and `dp test` commands.

- `dp run` runs the project on your local machine,
- `dp test` run tests for your project on your local machine.

1.1.7 Project deployment

`dp deploy` will sync with your bucket provider. The provider will be chosen automatically based on the remote URL. Usually, it is worth pointing `dp deploy` to a JSON or YAML file with provider-specific data like access tokens or project names. The *provider-specific data* should be interpreted as the `**kwargs` (keyword arguments) expected by a specific `fsspec`'s `FileSystem` implementation. One would most likely want to look at the [S3FileSystem](#) or [GCSFileSystem](#) documentation.

E.g., to connect with Google Cloud Storage, one should run:

```
echo '{"token": "<PATH_TO_YOUR_TOKEN>", "project_name": "<YOUR_PROJECT_NAME>"}' > gs_
↪args.json
dp deploy --dags-path "gs://<YOUR_GS_PATH>" --blob-args gs_args.json
```

However, in some cases, you do not need to do so, e.g. when using **gcloud** with properly set local credentials. In such a case, you can try to run just the `dp deploy --dags-path "gs://<YOUR_GS_PATH>"` command and let `gcsfs` search for the credentials. Please refer to the documentation of the specific `fsspec`'s implementation for more information about the required keyword arguments.

dags-path as config argument

You can also list your path in the `config/base/airflow.yml` file, as a `dags_path` argument:

```
dags_path: gs://<YOUR_GS_PATH>
# ... rest of the 'airflow.yml' file
```

In such a case, you do not have to provide a `--dags-path` flag, and you can just call `dp deploy` instead.

1.1.8 Packing and publishing

The built project can be processed to a **dbt** package by calling `dp publish`. `dp publish` parses `manifest.json` and prepares a package that lists models outputted by transformations, saving it in the `build/package` directory.

1.1.9 Preparing dbt environment

Sometimes you would like to use standalone **dbt** or an application that interfaces with it (like VS Code plugin). `dp prepare-env` prepares your local environment to be more conformant with standalone **dbt** requirements, e.g., by saving `profiles.yml` in the home directory.

However, be aware that most of the time you do not need to do so, and you can comfortably use `dp run` and `dp test` commands to interface with the **dbt** instead.

1.1.10 Clean project

When finished, call `dp clean` to remove compilation-related directories.

1.2 CLI Commands Reference

If you are looking for extensive information on a specific CLI command, this part of the documentation is for you.

1.2.1 `dp`

```
dp [OPTIONS] COMMAND [ARGS]...
```

Options

`--version`

Show the version and exit.

`clean`

Delete local working directories

```
dp clean [OPTIONS]
```

`compile`

Create local working directories and build artifacts

```
dp compile [OPTIONS]
```

Options

--env <env>

Required Name of the environment

Default base

--docker-build

Whether to build a Docker image

--docker-tag <docker_tag>

Image tag of a Docker image to create

create

Create a new project using a template

```
dp create [OPTIONS] PROJECT_PATH [TEMPLATE_PATH]...
```

Arguments

PROJECT_PATH

Required argument

TEMPLATE_PATH

Optional argument(s)

deploy

Push and deploy the project to the remote machine

```
dp deploy [OPTIONS]
```

Options

--env <env>

Name of the environment

Default base

--dags-path <dags_path>

Remote storage URI

--blob-args <blob_args>

Path to JSON or YAML file with arguments that should be passed to your Bucket/blob provider

--docker-push

Whether to push image to the Docker repository

--datahub-ingest

Whether to ingest DataHub metadata

docs-serve

Generate and serve dbt documentation.

```
dp docs-serve [OPTIONS]
```

Options

--env <env>

Name of the environment

Default local

--port <port>

Port to be used by the 'dbt docs serve' command

Default 9328

generate

Generate additional dbt files

```
dp generate [OPTIONS] COMMAND [ARGS]...
```

model-yaml

Generate schema YAML using codegen or dbt-profiler

```
dp generate model-yaml [OPTIONS] [MODEL_PATH]...
```

Options

--env <env>

Name of the environment

Default local

--with-meta

Whether to generate dbt-profiler metadata

--overwrite

Whether to overwrite existing YAML files

Arguments

MODEL_PATH

Optional argument(s)

source-sql

Generate SQLs that represents tables in given dataset

```
dp generate source-sql [OPTIONS]
```

Options

--env <env>

Name of the environment

Default local

--source-yaml-path <source_yaml_path>

Required Path to the 'source.yml' schema file

Default /home/docs/checkouts/readthedocs.org/user_builds/data-pipelines-cli/checkouts/0.18.0/docs/models/source/source.yml

--staging-path <staging_path>

Required Path to the 'staging' directory

Default /home/docs/checkouts/readthedocs.org/user_builds/data-pipelines-cli/checkouts/0.18.0/docs/models/staging

--overwrite

Whether to overwrite existing SQL files

source-yaml

Generate source YAML using codegen

```
dp generate source-yaml [OPTIONS] [SCHEMA_NAME]...
```

Options

--env <env>

Name of the environment

Default local

--source-path <source_path>

Required Path to the 'source' directory

Default /home/docs/checkouts/readthedocs.org/user_builds/data-pipelines-cli/checkouts/0.18.0/docs/models/source

--overwrite

Whether to overwrite an existing YAML file

Arguments

SCHEMA_NAME

Optional argument(s)

init

Configure the tool for the first time

```
dp init [OPTIONS] [CONFIG_PATH]...
```

Arguments

CONFIG_PATH

Optional argument(s)

prepare-env

Prepare local environment for apps interfacing with dbt

```
dp prepare-env [OPTIONS]
```

Options

--env <env>

Name of the environment

publish

Create a dbt package out of the project

```
dp publish [OPTIONS]
```

Options

--key-path <key_path>

Required Path to the key with write access to repo with published packages

--env <env>

Required Name of the environment

Default base

run

Run the project on the local machine

```
dp run [OPTIONS]
```

Options

--env <env>

Name of the environment

Default local

template-list

Print a list of all templates saved in the config file

```
dp template-list [OPTIONS]
```

test

Run tests of the project on the local machine

```
dp test [OPTIONS]
```

Options

--env <env>

Name of the environment

Default local

update

Update project from its template

```
dp update [OPTIONS] [PROJECT_PATH] ...
```

Options

--vcs-ref <vcs_ref>

Git reference to checkout

Arguments

PROJECT_PATH

Optional argument(s)

1.3 API Reference

If you are looking for information on a specific function, class, or method, this part of the documentation is for you.

1.3.1 data_pipelines_cli package

data-pipelines-cli (dp) is a CLI tool designed for data platform.

dp helps data analysts to create, maintain and make full use of their data pipelines.

Subpackages

data_pipelines_cli.cli_commands package

Subpackages

data_pipelines_cli.cli_commands.generate package

Submodules

data_pipelines_cli.cli_commands.generate.generate module

data_pipelines_cli.cli_commands.generate.model_yaml module

```
class MacroArgName(**kwargs)
```

```
    Bases: dict
```

```
    arg_name: str
```

```
    deps_name: str
```

```
    macro_name: str
```

```
generate_model_yamls(env: str, with_meta: bool, overwrite: bool, model_paths: Sequence[pathlib.Path]) →  
    None
```

data_pipelines_cli.cli_commands.generate.source_sql module

generate_source_sqls(*env: str, source_yaml_path: pathlib.Path, staging_path: pathlib.Path, overwrite: bool*)
→ None

data_pipelines_cli.cli_commands.generate.source_yaml module

generate_source_yamls(*env: str, source_path: pathlib.Path, overwrite: bool, schema_names: Sequence[str]*) →
None

data_pipelines_cli.cli_commands.generate.utils module

generate_models_or_sources_from_single_table(*env: str, macro_name: str, macro_args: Dict[str, Any],
profiles_path: pathlib.Path*) → Dict[str, Any]

get_macro_run_output(*env: str, macro_name: str, macro_args: Dict[str, str], profiles_path: pathlib.Path*) → str

get_output_file_or_warn_if_exists(*directory: pathlib.Path, overwrite: bool, file_extension: str, filename:
Optional[str] = None*) → Optional[pathlib.Path]

Submodules**data_pipelines_cli.cli_commands.clean module**

clean() → None
Delete local working directories.

data_pipelines_cli.cli_commands.compile module

compile_project(*env: str, docker_tag: Optional[str] = None, docker_build: bool = False*) → None
Create local working directories and build artifacts.

Parameters

- **env** (*str*) – Name of the environment
- **docker_tag** (*Optional[str]*) – Image tag of a Docker image to create
- **docker_build** (*bool*) – Whether to build a Docker image

Raises *DataPipelinesError* –

replace_image_settings(*docker_args: data_pipelines_cli.data_structures.DockerArgs*) → None

data_pipelines_cli.cli_commands.create module

create(*project_path: str, template_path: Optional[str]*) → None

Create a new project using a template.

Parameters

- **project_path** (*str*) – Path to a directory to create
- **template_path** (*Optional[str]*) – Path or URI to the repository of the project template

Raises *DataPipelinesError* – no template found in *.dp.yml* config file

data_pipelines_cli.cli_commands.deploy module

class DeployCommand(*env: str, docker_push: bool, dags_path: Optional[str], provider_kwargs_dict: Optional[Dict[str, Any]], datahub_ingest: bool*)

Bases: object

A class used to push and deploy the project to the remote machine.

blob_address_path: str

URI of the cloud storage to send build artifacts to

datahub_ingest: bool

Whether to ingest DataHub metadata

deploy() → None

Push and deploy the project to the remote machine.

Raises

- *DependencyNotInstalledError* – DataHub or Docker not installed
- *DataPipelinesError* – Error while pushing Docker image

docker_args: Optional[data_pipelines_cli.data_structures.DockerArgs]

Arguments required by the Docker to make a push to the repository. If set to *None*, *deploy()* will not make a push

env: str

provider_kwargs_dict: Dict[str, Any]

Dictionary of arguments required by a specific cloud storage provider, e.g. path to a token, username, password, etc.

data_pipelines_cli.cli_commands.docs module

docs(*env: str, port: int*) → None

Generate and serve dbt documentation.

Parameters

- **env** (*str*) – Name of the environment
- **port** (*int*) – Port to serve dbt documentation on.

data_pipelines_cli.cli_commands.init module**init**(*config_path: Optional[str]*) → None

Configure the tool for the first time.

Parameters **config_path** (*Optional[str]*) – URI of the repository with a template of the config file**Raises** **DataPipelinesError** – user do not want to overwrite existing config file**data_pipelines_cli.cli_commands.prepare_env module****prepare_env**(*env: str*) → None

Prepare local environment for use with dbt-related applications.

Prepare local environment for use with applications expecting a “traditional” dbt structure, such as plugins to VS Code. If in doubt, use `dp run` and `dp test` instead.**Parameters** **env** (*str*) – Name of the environment**data_pipelines_cli.cli_commands.publish module****create_package**() → `pathlib.Path`

Create a dbt package out of the built project.

Raises **DataPipelinesError** – There is no model in ‘manifest.json’ file.**publish_package**(*package_path: pathlib.Path, key_path: str, env: str*) → None**data_pipelines_cli.cli_commands.run module****run**(*env: str*) → None

Run the project on the local machine.

Parameters **env** (*str*) – Name of the environment**data_pipelines_cli.cli_commands.template module****list_templates**() → None

Print a list of all templates saved in the config file.

data_pipelines_cli.cli_commands.test module**test**(*env: str*) → None

Run tests of the project on the local machine.

Parameters **env** (*str*) – Name of the environment

data_pipelines_cli.cli_commands.update module

update(*project_path*: str, *vcs_ref*: str) → None

Update an existing project from its template.

Parameters

- **project_path** (str) – Path to a directory to create
- **vcs_ref** (str) – Git reference to checkout in projects template

Submodules

data_pipelines_cli.cli module

cli() → None

data_pipelines_cli.cli_configs module

find_datahub_config_file(*env*: str) → pathlib.Path

data_pipelines_cli.cli_constants module

DEFAULT_GLOBAL_CONFIG: *data_pipelines_cli.data_structures.DataPipelinesConfig* = {'templates': {}, 'vars': {}}

Content of the config file created by *dp init* command if no template path is provided

IMAGE_TAG_TO_REPLACE: str = '<IMAGE_TAG>'

PROFILE_NAME_ENV_EXECUTION = 'env_execution'

Name of the dbt target to use for a remote machine

PROFILE_NAME_LOCAL_ENVIRONMENT = 'local'

Name of the environment and dbt target to use for a local machine

get_dbt_profiles_env_name(*env*: str) → str

Given a name of the environment, returns one of target names expected by the *profiles.yml* file.

Parameters **env** (str) – Name of the environment

Returns Name of the *target* to be used in *profiles.yml*

data_pipelines_cli.cli_utils module

echo_error(*text*: str, ***kwargs*: Any) → None

Print an error message to stderr using click-specific print function.

Parameters

- **text** (str) – Message to print
- **kwargs** –

echo_info(*text: str, **kwargs: Any*) → None

Print a message to stdout using click-specific print function.

Parameters

- **text** (*str*) – Message to print
- **kwargs** –

echo_suberror(*text: str, **kwargs: Any*) → None

Print a suberror message to stderr using click-specific print function.

Parameters

- **text** (*str*) – Message to print
- **kwargs** –

echo_subinfo(*text: str, **kwargs: Any*) → None

Print a subinfo message to stdout using click-specific print function.

Parameters

- **text** (*str*) – Message to print
- **kwargs** –

echo_warning(*text: str, **kwargs: Any*) → None

Print a warning message to stderr using click-specific print function.

Parameters

- **text** (*str*) – Message to print
- **kwargs** –

get_argument_or_environment_variable(*argument: Optional[str], argument_name: str, environment_variable_name: str*) → str

Given *argument* is not None, return its value. Otherwise, search for *environment_variable_name* amongst environment variables and return it. If such a variable is not set, raise [*DataPipelinesError*](#).

Parameters

- **argument** (*Optional[str]*) – Optional value passed to the CLI as the *argument_name*
- **argument_name** (*str*) – Name of the CLI's argument
- **environment_variable_name** (*str*) – Name of the environment variable to search for

Returns Value of the *argument* or specified environment variable

Raises [*DataPipelinesError*](#) – *argument* is None and *environment_variable_name* is not set

subprocess_run(*args: List[str], capture_output: bool = False*) → subprocess.CompletedProcess[bytes]

Run subprocess and return its state if completed with a success. If not, raise [*SubprocessNonZeroExitError*](#).

Parameters

- **args** (*List[str]*) – List of strings representing subprocess and its arguments
- **capture_output** (*bool*) – Whether to capture output of subprocess.

Returns State of the completed process

Return type subprocess.CompletedProcess[bytes]

Raises [*SubprocessNonZeroExitError*](#) – subprocess exited with non-zero exit code

data_pipelines_cli.config_generation module**class DbtProfile(**kwargs)**

Bases: dict

POD representing dbt's *profiles.yml* file.**outputs:** Dict[str, Dict[str, Any]]Dictionary of a warehouse data and credentials, referenced by *target* name**target:** strName of the *target* for dbt to run**copy_config_dir_to_build_dir()** → NoneRecursively copy *config* directory to *build/dag/config* working directory.**copy_dag_dir_to_build_dir()** → NoneRecursively copy *dag* directory to *build/dag* working directory.**generate_profiles_dict**(*env*: str, *copy_config_dir*: bool) → Dict[str, *data_pipelines_cli.config_generation.DbtProfile*]Generate and save *profiles.yml* file at *build/profiles/local* or *build/profiles/env_execution*, depending on *env* argument.**Parameters**

- **env** (str) – Name of the environment
- **copy_config_dir** (bool) – Whether to copy config directory to build working directory

Returns Dictionary representing data to be saved in *profiles.yml***Return type** Dict[str, *DbtProfile*]**generate_profiles_yaml**(*env*: str, *copy_config_dir*: bool = True) → pathlib.PathGenerate and save *profiles.yml* file at *build/profiles/local* or *build/profiles/env_execution*, depending on *env* argument.**Parameters**

- **env** (str) – Name of the environment
- **copy_config_dir** (bool) – Whether to copy config directory to build working directory

Returns Path to *build/profiles/{env}***Return type** pathlib.Path**get_profiles_dir_build_path**(*env*: str) → pathlib.PathReturns path to *build/profiles/<profile_name>/*, depending on *env* argument.**Parameters** **env** (str) – Name of the environment**Returns****Return type** pathlib.Path**read_dictionary_from_config_directory**(*config_path*: Union[str, os.PathLike[str]], *env*: str, *file_name*: str) → Dict[str, Any]Read dictionaries out of *file_name* in both *base* and *env* directories, and compile them into one. Values from *env* directory get precedence over *base* ones.**Parameters**

- **config_path** (*Union[str, os.PathLike[str]]*) – Path to the *config* directory
- **env** (*str*) – Name of the environment
- **file_name** (*str*) – Name of the YAML file to parse dictionary from

Returns Compiled dictionary

Return type Dict[str, Any]

data_pipelines_cli.data_structures module

class DataPipelinesConfig(***kwargs*)

Bases: dict

POD representing *.dp.yml* config file.

templates: Dict[str, [data_pipelines_cli.data_structures.TemplateConfig](#)]

Dictionary of saved templates to use in *dp create* command

vars: Dict[str, str]

Variables to be passed to dbt as *--vars* argument

class DbtModel(***kwargs*)

Bases: dict

POD representing a single model from 'schema.yml' file.

columns: List[[data_pipelines_cli.data_structures.DbtTableColumn](#)]

description: str

identifier: str

meta: Dict[str, Any]

name: str

tags: List[str]

tests: List[str]

class DbtSource(***kwargs*)

Bases: dict

POD representing a single source from 'schema.yml' file.

database: str

description: str

meta: Dict[str, Any]

name: str

schema: str

tables: List[[data_pipelines_cli.data_structures.DbtModel](#)]

tags: List[str]

```
class DbtTableColumn(**kwargs)
```

Bases: dict

POD representing a single column from 'schema.yml' file.

description: str

meta: Dict[str, Any]

name: str

quote: bool

tags: List[str]

tests: List[str]

```
class DockerArgs(env: str, image_tag: Optional[str])
```

Bases: object

Arguments required by the Docker to make a push to the repository.

Raises *DataPipelinesError* – repository variable not set or git hash not found

docker_build_tag() → str

Prepare a tag for Docker Python API build command.

Returns Tag for Docker Python API build command

Return type str

image_tag: str

An image tag

repository: str

URI of the Docker images repository

```
class TemplateConfig(**kwargs)
```

Bases: dict

POD representing value referenced in the *templates* section of the *.dp.yml* config file.

template_name: str

Name of the template

template_path: str

Local path or Git URI to the template repository

```
read_env_config() → data_pipelines_cli.data_structures.DataPipelinesConfig
```

Parse *.dp.yml* config file, if it exists. Otherwise, raises *NoConfigFileError*.

Returns POD representing *.dp.yml* config file, if it exists

Return type *DataPipelinesConfig*

Raises *NoConfigFileError* – *.dp.yml* file not found

data_pipelines_cli.dbt_utils module**read_dbt_vars_from_configs**(*env: str*) → Dict[str, Any]

Read *vars* field from dp configuration file (\$HOME/.dp.yml), base dbt.yml config (config/base/dbt.yml) and environment-specific config (config/{env}/dbt.yml) and compile into one dictionary.

Parameters *env* (*str*) – Name of the environment

Returns Dictionary with *vars* and their keys

Return type Dict[str, Any]

run_dbt_command(*command: Tuple[str, ...]*, *env: str*, *profiles_path: pathlib.Path*, *log_format_json: bool = False*, *capture_output: bool = False*) → subprocess.CompletedProcess[bytes]

Run dbt subprocess in a context of specified *env*.

Parameters

- **command** (*Tuple[str, ...]*) – Tuple representing dbt command and its optional arguments
- **env** (*str*) – Name of the environment
- **profiles_path** (*pathlib.Path*) – Path to the directory containing *profiles.yml* file
- **log_format_json** (*bool*) – Whether to run dbt command with *-log-format=json* flag
- **capture_output** (*bool*) – Whether to capture stdout of subprocess.

Returns State of the completed process

Return type subprocess.CompletedProcess[bytes]

Raises

- [*SubprocessNotFound*](#) – dbt not installed
- [*SubprocessNonZeroExitError*](#) – dbt exited with error

data_pipelines_cli.docker_response_reader module**class DockerReadResponse**(*msg: str*, *is_error: bool*)

Bases: object

POD representing Docker response processed by [*DockerResponseReader*](#).

is_error: bool

Whether response is error or not

msg: str

Read and processed message

class DockerResponseReader(*logs_generator: Iterable[Union[str, Dict[str, Union[str, Dict[str, str]]]]]*)

Bases: object

Read and process Docker response.

Docker response turns into processed strings instead of plain dictionaries.

cached_read_response:

Optional[List[[*data_pipelines_cli.docker_response_reader.DockerReadResponse*](#)]]

Internal cache of already processed response

click_echo_ok_responses() → None

Read, process and print positive Docker updates.

Raises *DockerErrorResponseError* – Came across error update in Docker response.

logs_generator: `Iterable[Union[str, Dict[str, Union[str, Dict[str, str]]]]]`

Iterable representing Docker response

read_response() → `List[data_pipelines_cli.docker_response_reader.DockerReadResponse]`

Read and process Docker response.

Returns List of processed lines of response

Return type `List[DockerReadResponse]`

data_pipelines_cli.errors module

exception AirflowDagsPathKeyError

Bases: *data_pipelines_cli.errors.DataPipelinesError*

Exception raised if there is no dags_path in *airflow.yml* file.

message: `str`

explanation of the error

submessage: `Optional[str]`

additional informations for the error

exception DataPipelinesError(*message: str, submessage: Optional[str] = None*)

Bases: `Exception`

Base class for all exceptions in data_pipelines_cli module

message: `str`

explanation of the error

submessage: `Optional[str]`

additional informations for the error

exception DependencyNotInstalledError(*program_name: str*)

Bases: *data_pipelines_cli.errors.DataPipelinesError*

Exception raised if certain dependency is not installed

message: `str`

explanation of the error

submessage: `Optional[str]`

additional informations for the error

exception DockerErrorResponseError(*error_msg: str*)

Bases: *data_pipelines_cli.errors.DataPipelinesError*

Exception raised if there is an error response from Docker client.

message: `str`

explanation of the error

submessage: Optional[str]
 additional informations for the error

exception DockerNotInstalledError

Bases: *data_pipelines_cli.errors.DependencyNotInstalledError*

Exception raised if 'docker' is not installed

message: str
 explanation of the error

submessage: Optional[str]
 additional informations for the error

exception JinjaVarKeyError(*key: str*)

Bases: *data_pipelines_cli.errors.DataPipelinesError*

message: str
 explanation of the error

submessage: Optional[str]
 additional informations for the error

exception NoConfigFileError

Bases: *data_pipelines_cli.errors.DataPipelinesError*

Exception raised if *.dp.yml* does not exist

message: str
 explanation of the error

submessage: Optional[str]
 additional informations for the error

exception NotAProjectDirectoryError(*project_path: str*)

Bases: *data_pipelines_cli.errors.DataPipelinesError*

Exception raised if *.copier-answers.yml* file does not exist in given dir

message: str
 explanation of the error

submessage: Optional[str]
 additional informations for the error

exception SubprocessNonZeroExitError(*subprocess_name: str, exit_code: int, subprocess_output: Optional[str] = None*)

Bases: *data_pipelines_cli.errors.DataPipelinesError*

Exception raised if subprocess exits with non-zero exit code

message: str
 explanation of the error

submessage: Optional[str]
 additional informations for the error

exception SubprocessNotFound(*subprocess_name: str*)

Bases: `data_pipelines_cli.errors.DataPipelinesError`

Exception raised if subprocess cannot be found

message: `str`

explanation of the error

submessage: `Optional[str]`

additional informations for the error

data_pipelines_cli.filesystem_utils module

class LocalRemoteSync(*local_path: Union[str, os.PathLike[str]], remote_path: str, remote_kwargs: Dict[str, str]*)

Bases: `object`

Synchronizes local directory with a cloud storage's one.

local_fs: `fsspec.spec.AbstractFileSystem`

FS representing local directory

local_path_str: `str`

Path to local directory

remote_path_str: `str`

Path/URI of the cloud storage directory

sync(*delete: bool = True*) \rightarrow `None`

Send local files to the remote directory and (optionally) delete unnecessary ones.

Parameters **delete** (*bool*) – Whether to delete remote files that are no longer present in local directory

data_pipelines_cli.io_utils module

git_revision_hash() \rightarrow `Optional[str]`

Get current Git revision hash, if Git is installed and any revision exists.

Returns Git revision hash, if possible.

Return type `Optional[str]`

replace(*filename: Union[str, os.PathLike[str]], pattern: str, replacement: str*) \rightarrow `None`

Perform the pure-Python equivalent of in-place *sed* substitution: e.g., `sed -i -e 's/{pattern}/{replacement}' '{filename}'`.

Beware however, it uses Python regex dialect instead of *sed*'s one. It can introduce regex-related bugs.

data_pipelines_cli.jinja module

replace_vars_with_values(*templated_dictionary*: Dict[str, Any], *dbt_vars*: Dict[str, Any]) → Dict[str, Any]

Replace variables in given dictionary using Jinja template in its values.

Parameters

- **templated_dictionary** (Dict[str, Any]) – Dictionary with Jinja-templated values
- **dbt_vars** (Dict[str, Any]) – Variables to replace

Returns Dictionary with replaced variables

Return type Dict[str, Any]

Raises *JinjaVarKeyError* – Variable referenced in Jinja template does not exist

data_pipelines_cli.vcs_utils module

Utilities related to VCS.

add_suffix_to_git_template_path(*template_path*: str) → str

Add .git suffix to *template_path*, if necessary.

Check if *template_path* starts with Git-specific prefix (e.g. *git://*), or *http://* or *https://* protocol. If so, then add .git suffix if not present. Does nothing otherwise (as *template_path* probably points to a local directory).

Parameters **template_path** (str) – Path or URI to Git-based repository

Returns *template_path* with .git as suffix, if necessary

Return type str

1.4 Changelog

1.4.1 Unreleased

1.4.2 0.18.0 - 2022-04-19

Added

- `dp docs-serve` command acting as a wrapper for `dbt docs serve`.

1.4.3 0.17.0 - 2022-04-11

Added

- `pip install data-pipelines-cli[ADAPTER_PROVIDER]` installs adapter alongside **dbt-core**, e.g. `pip install data-pipelines-cli[bigquery]`.

Changed

- `dp compile` accepts additional command line argument `--docker-tag`, allowing for custom Docker tag instead of relying on Git commit SHA. Moreover, if `--docker-tag` is not provided, **dp** searches for tag in `build/dag/config/<ENV>/execution_env.yml`. If it is present instead of `<IMAGE_TAG>` to be replaced, **dp** chooses it over Git commit SHA.

1.4.4 0.16.0 - 2022-03-24

Added

- `dp generate source-yaml` and `dp generate model-yaml` commands that automatically generate YAML schema files for project's sources or models, respectively (using `dbt-codegen` or `dbt-profiler` under the hood).
- `dp generate source-sql` command that generates SQL representing sources listed in `source.yml` (or a similar file) (again, with the help of `dbt-codegen`).

1.4.5 0.15.2 - 2022-02-28

Changed

- Bumped dbt to 1.0.3.

1.4.6 0.15.1 - 2022-02-28

Fixed

- Pinned MarkupSafe==2.0.1 to ensure that Jinja works.

1.4.7 0.15.0 - 2022-02-11

- Migration to dbt 1.0.1

1.4.8 0.14.0 - 2022-02-02

1.4.9 0.13.0 - 2022-02-01

1.4.10 0.12.0 - 2022-01-31

- `dp publish` will push generated sources to external git repo

1.4.11 0.11.0 - 2022-01-18

Added

- `dp update` command
- `dp publish` command for creation of dbt package out of the project.

Changed

- Docker response in `deploy` and `compile` gets printed as processed strings instead of plain dictionaries.
- `dp compile` parses content of `datahub.yml` and replaces Jinja variables in the form of `var` or `env_var`.
- `dags_path` is read from an env'd `airflow.yml` file.

1.4.12 0.10.0 - 2022-01-12

Changed

- Run `dbt deps` at the end of `dp prepare-env`.

Fixed

- `dp run` and `dp test` are no longer pointing to `profiles.yml` instead of the directory containing it.

1.4.13 0.9.0 - 2022-01-03

Added

- `--env` flag to `dp deploy`.

Changed

- Docker repository URI gets read out of `build/config/{env}/k8s.yml`.

Removed

- `--docker-repository-uri` and `--datahub-gms-uri` from `dp compile` and `dp deploy` commands.
- `dp compile` no longer replaces `<INGEST_ENDPOINT>` in `datahub.yml`, or `<DOCKER_REPOSITORY_URL>` in `k8s.yml`

1.4.14 0.8.0 - 2021-12-31

Changed

- `dp init` and `dp create` automatically adds `.git` suffix to given template paths, if necessary.
- When reading dbt variables, global-scoped variables take precedence over project-scoped ones (it was another way around before).
- Address argument for `dp deploy` is no longer mandatory. It should be either placed in `airflow.yml` file as value of `dags_path` key, or provided with `--dags-path` flag.

1.4.15 0.7.0 - 2021-12-29

Added

- Add documentation in the style of [Read the Docs](#).
- Exception classes in `errors.py`, deriving from `DataPipelinesError` base exception class.
- Unit tests to massively improve code coverage.
- `--version` flag to `dp` command.
- Add `dp prepare-env` command that prepares local environment for standalone **dbt** (right now, it only generates and saves `profiles.yml` in `$HOME/.dbt`).

Changed

- `dp compile`:
 - `--env` option has a default value: `base`,
 - `--datahub` is changed to `--datahub-gms-uri`, `--repository` is changed to `--docker-repository-uri`.
- `dp deploy`'s `--docker-push` is not a flag anymore and requires a Docker repository URI parameter; `--repository` got removed then.
- `dp run` and `dp test` run `dp compile` before actual **dbt** command.
- Functions raise exceptions instead of exiting using `sys.exit(1)`; `cli.cli()` entrypoint is expecting exception and exits only there.
- `dp deploy` raises an exception if there is no Docker image to push or `build/config/dag` directory does not exist.
- Rename `gcp` to `gcs` in requirements (now one should run `pip install data-pipelines-cli[gcs]`).

1.4.16 0.6.0 - 2021-12-16

Modified

- **dp** saves generated `profiles.yml` in either `build/local` or `build/env_execution` directories. **dbt** gets executed with `env_execution` as the target.

1.4.17 0.5.1 - 2021-12-14

Fixed

- `_dbt_compile` is no longer removing replaced `<IMAGE_TAG>`.

1.4.18 0.5.0 - 2021-12-14

Added

- `echo_warning` function prints warning messages in yellow/orange color.

Modified

- Docker image gets built at the end of `compile` command.
- **dbt**-related commands do not fail if no `$HOME/.dp.yml` exists (e.g., `dp run`).

Removed

- Dropped `dbt-airflow-manifest-parser` dependency.

1.4.19 0.4.0 - 2021-12-13

Added

- `dp run` and `dp test` commands.
- `dp clean` command for removing `build` and `target` directories.
- File synchronization tests for Google Cloud Storage using `gcp-storage-emulator`.
- Read vars from config files (`$HOME/.dp.yml`, `config/$ENV/dbt.yml`) and pass to `dbt`.

Modified

- `profiles.yml` gets generated and saved in `build` directory in `dp compile`, instead of relying on a local one in the main project directory.
- `dp dbt <command>` generates `profiles.yml` in `build` directory by default.
- `dp init` is expecting `config_path` argument to download config template with the help of the `copier` and save it in `$HOME/.dp.yml`.
- `dp template list` is renamed as `dp template-list`.
- `dp create` allows for providing extra argument called `template-path`, being either name of one of templates defined in `.dp.yml` config file or direct link to Git repository.

Removed

- Support for manually created `profiles.yml` in main project directory.
- `dp template new` command.
- `username` field from `$HOME/.dp.yml` file.

1.4.20 0.3.0 - 2021-12-06

- Run `dbt deps` alongside rest of `dbt` commands in `dp compile`

1.4.21 0.2.0 - 2021-12-03

- Add support for GCP and S3 syncing in `dp deploy`

1.4.22 0.1.2 - 2021-12-02

- Fix: do not use styled `click.secho` for Docker push response, as it may not be a `str`

1.4.23 0.1.1 - 2021-12-01

- Fix Docker SDK for Python's bug related to tagging, which prevented Docker from pushing images.

1.4.24 0.1.0 - 2021-12-01**Added**

- Draft of `dp init`, `dp create`, `dp template new`, `dp template list` and `dp dbt`
- Draft of `dp compile` and `dp deploy`

PYTHON MODULE INDEX

d

- `data_pipelines_cli`, 14
- `data_pipelines_cli.cli`, 18
- `data_pipelines_cli.cli_commands`, 14
- `data_pipelines_cli.cli_commands.clean`, 15
- `data_pipelines_cli.cli_commands.compile`, 15
- `data_pipelines_cli.cli_commands.create`, 16
- `data_pipelines_cli.cli_commands.deploy`, 16
- `data_pipelines_cli.cli_commands.docs`, 16
- `data_pipelines_cli.cli_commands.generate`, 14
- `data_pipelines_cli.cli_commands.generate.generate`, 14
- `data_pipelines_cli.cli_commands.generate.model_yaml`, 14
- `data_pipelines_cli.cli_commands.generate.source_sql`, 15
- `data_pipelines_cli.cli_commands.generate.source_yaml`, 15
- `data_pipelines_cli.cli_commands.generate.utils`, 15
- `data_pipelines_cli.cli_commands.init`, 17
- `data_pipelines_cli.cli_commands.prepare_env`, 17
- `data_pipelines_cli.cli_commands.publish`, 17
- `data_pipelines_cli.cli_commands.run`, 17
- `data_pipelines_cli.cli_commands.template`, 17
- `data_pipelines_cli.cli_commands.test`, 17
- `data_pipelines_cli.cli_commands.update`, 18
- `data_pipelines_cli.cli_configs`, 18
- `data_pipelines_cli.cli_constants`, 18
- `data_pipelines_cli.cli_utils`, 18
- `data_pipelines_cli.config_generation`, 20
- `data_pipelines_cli.data_structures`, 21
- `data_pipelines_cli.dbt_utils`, 23
- `data_pipelines_cli.docker_response_reader`, 23
- `data_pipelines_cli.errors`, 24
- `data_pipelines_cli.filesystem_utils`, 26
- `data_pipelines_cli.io_utils`, 26
- `data_pipelines_cli.jinja`, 27
- `data_pipelines_cli.vcs_utils`, 27

Symbols

- blob-args
 - dp-deploy command line option, 9
- dags-path
 - dp-deploy command line option, 9
- datahub-ingest
 - dp-deploy command line option, 9
- docker-build
 - dp-compile command line option, 9
- docker-push
 - dp-deploy command line option, 9
- docker-tag
 - dp-compile command line option, 9
- env
 - dp-compile command line option, 9
 - dp-deploy command line option, 9
 - dp-docs-serve command line option, 10
 - dp-generate-model-yaml command line option, 10
 - dp-generate-source-sql command line option, 11
 - dp-generate-source-yaml command line option, 11
 - dp-prepare-env command line option, 12
 - dp-publish command line option, 12
 - dp-run command line option, 13
 - dp-test command line option, 13
- key-path
 - dp-publish command line option, 12
- overwrite
 - dp-generate-model-yaml command line option, 10
 - dp-generate-source-sql command line option, 11
 - dp-generate-source-yaml command line option, 11
- port
 - dp-docs-serve command line option, 10
- source-path
 - dp-generate-source-yaml command line option, 11
- source-yaml-path

- dp-generate-source-sql command line option, 11
- staging-path
 - dp-generate-source-sql command line option, 11
- vcs-ref
 - dp-update command line option, 13
- version
 - dp command line option, 8
- with-meta
 - dp-generate-model-yaml command line option, 10

A

- add_suffix_to_git_template_path() (in module *data_pipelines_cli.vcs_utils*), 27
- AirflowDagsPathKeyError, 24
- arg_name (*MacroArgName* attribute), 14

B

- blob_address_path (*DeployCommand* attribute), 16

C

- cached_read_response (*DockerResponseReader* attribute), 23
- clean() (in module *data_pipelines_cli.cli_commands.clean*), 15
- cli() (in module *data_pipelines_cli.cli*), 18
- click_echo_ok_responses() (*DockerResponseReader* method), 23
- columns (*DbtModel* attribute), 21
- compile_project() (in module *data_pipelines_cli.cli_commands.compile*), 15
- CONFIG_PATH
 - dp-init command line option, 12
- copy_config_dir_to_build_dir() (in module *data_pipelines_cli.config_generation*), 20
- copy_dag_dir_to_build_dir() (in module *data_pipelines_cli.config_generation*), 20
- create() (in module *data_pipelines_cli.cli_commands.create*), 16

`create_package()` (in *module* `data_pipelines_cli.cli_commands.publish`), 17

D

`data_pipelines_cli` *module*, 14
`data_pipelines_cli.cli` *module*, 18
`data_pipelines_cli.cli_commands` *module*, 14
`data_pipelines_cli.cli_commands.clean` *module*, 15
`data_pipelines_cli.cli_commands.compile` *module*, 15
`data_pipelines_cli.cli_commands.create` *module*, 16
`data_pipelines_cli.cli_commands.deploy` *module*, 16
`data_pipelines_cli.cli_commands.docs` *module*, 16
`data_pipelines_cli.cli_commands.generate` *module*, 14
`data_pipelines_cli.cli_commands.generate.generate` *module*, 14
`data_pipelines_cli.cli_commands.generate.model_yaml` *module*, 14
`data_pipelines_cli.cli_commands.generate.source_sql` *module*, 15
`data_pipelines_cli.cli_commands.generate.source_yaml` *module*, 15
`data_pipelines_cli.cli_commands.generate.utils` *module*, 15
`data_pipelines_cli.cli_commands.init` *module*, 17
`data_pipelines_cli.cli_commands.prepare_env` *module*, 17
`data_pipelines_cli.cli_commands.publish` *module*, 17
`data_pipelines_cli.cli_commands.run` *module*, 17
`data_pipelines_cli.cli_commands.template` *module*, 17
`data_pipelines_cli.cli_commands.test` *module*, 17
`data_pipelines_cli.cli_commands.update` *module*, 18
`data_pipelines_cli.cli_configs` *module*, 18
`data_pipelines_cli.cli_constants` *module*, 18
`data_pipelines_cli.cli_utils` *module*, 18
`data_pipelines_cli.config_generation` *module*, 20
`data_pipelines_cli.data_structures` *module*, 21
`data_pipelines_cli.dbt_utils` *module*, 23
`data_pipelines_cli.docker_response_reader` *module*, 23
`data_pipelines_cli.errors` *module*, 24
`data_pipelines_cli.filesystem_utils` *module*, 26
`data_pipelines_cli.io_utils` *module*, 26
`data_pipelines_cli.jinja` *module*, 27
`data_pipelines_cli.vcs_utils` *module*, 27
`database` (*DbtSource* attribute), 21
`datahub_ingest` (*DeployCommand* attribute), 16
`DataPipelinesConfig` (class in *data_pipelines_cli.data_structures*), 21
`DataPipelinesError`, 24
`DbtModel` (class in *data_pipelines_cli.data_structures*), 21
`DbtProfile` (class in *data_pipelines_cli.config_generation*), 20
`DbtSource` (class in *data_pipelines_cli.data_structures*), 21
`DbtTableColumn` (class in *data_pipelines_cli.data_structures*), 21
`DEFAULT_GLOBAL_CONFIG` (in *data_pipelines_cli.cli_constants*), 18
`DependencyNotInstalledError`, 24
`deploy()` (*DeployCommand* method), 16
`DeployCommand` (class in *data_pipelines_cli.cli_commands.deploy*), 16
`deps_name` (*MacroArgName* attribute), 14
`description` (*DbtModel* attribute), 21
`description` (*DbtSource* attribute), 21
`description` (*DbtTableColumn* attribute), 22
`docker_args` (*DeployCommand* attribute), 16
`docker_build_tag()` (*DockerArgs* method), 22
`DockerArgs` (class in *data_pipelines_cli.data_structures*), 22
`DockerErrorResponseError`, 24
`DockerNotInstalledError`, 25
`DockerReadResponse` (class in *data_pipelines_cli.docker_response_reader*), 23
`DockerResponseReader` (class in *data_pipelines_cli.docker_response_reader*), 23

docs() (in module *data_pipelines_cli.cli_commands.docs*), 16

dp command line option

- version, 8

dp-compile command line option

- docker-build, 9
- docker-tag, 9
- env, 9

dp-create command line option

- PROJECT_PATH, 9
- TEMPLATE_PATH, 9

dp-deploy command line option

- blob-args, 9
- dags-path, 9
- datahub-ingest, 9
- docker-push, 9
- env, 9

dp-docs-serve command line option

- env, 10
- port, 10

dp-generate-model-yaml command line option

- env, 10
- overwrite, 10
- with-meta, 10
- MODEL_PATH, 11

dp-generate-source-sql command line option

- env, 11
- overwrite, 11
- source-yaml-path, 11
- staging-path, 11

dp-generate-source-yaml command line option

- env, 11
- overwrite, 11
- source-path, 11
- SCHEMA_NAME, 12

dp-init command line option

- CONFIG_PATH, 12

dp-prepare-env command line option

- env, 12

dp-publish command line option

- env, 12
- key-path, 12

dp-run command line option

- env, 13

dp-test command line option

- env, 13

dp-update command line option

- vcs-ref, 13
- PROJECT_PATH, 14

E

echo_error() (in module *data_pipelines_cli.cli_utils*), 18

echo_info() (in module *data_pipelines_cli.cli_utils*), 18

docs() (in module *data_pipelines_cli.cli_commands.docs*), 16

echo_suberror() (in module *data_pipelines_cli.cli_utils*), 19

echo_subinfo() (in module *data_pipelines_cli.cli_utils*), 19

echo_warning() (in module *data_pipelines_cli.cli_utils*), 19

env (DeployCommand attribute), 16

F

find_datahub_config_file() (in module *data_pipelines_cli.cli_configs*), 18

G

generate_model_yamls() (in module *data_pipelines_cli.cli_commands.generate.model_yaml*), 14

generate_models_or_sources_from_single_table() (in module *data_pipelines_cli.cli_commands.generate.utils*), 15

generate_profiles_dict() (in module *data_pipelines_cli.config_generation*), 20

generate_profiles_yaml() (in module *data_pipelines_cli.config_generation*), 20

generate_source_sqls() (in module *data_pipelines_cli.cli_commands.generate.source_sql*), 15

generate_source_yamls() (in module *data_pipelines_cli.cli_commands.generate.source_yaml*), 15

get_argument_or_environment_variable() (in module *data_pipelines_cli.cli_utils*), 19

get_dbt_profiles_env_name() (in module *data_pipelines_cli.cli_constants*), 18

get_macro_run_output() (in module *data_pipelines_cli.cli_commands.generate.utils*), 15

get_output_file_or_warn_if_exists() (in module *data_pipelines_cli.cli_commands.generate.utils*), 15

get_profiles_dir_build_path() (in module *data_pipelines_cli.config_generation*), 20

git_revision_hash() (in module *data_pipelines_cli.io_utils*), 26

I

identifier (DbtModel attribute), 21

image_tag (DockerArgs attribute), 22

IMAGE_TAG_TO_REPLACE (in module *data_pipelines_cli.cli_constants*), 18

init() (in module *data_pipelines_cli.cli_commands.init*), 17

is_error (DockerReadResponse attribute), 23

J

JinjaVarKeyError, 25

L

list_templates() (in module
data_pipelines_cli.cli_commands.template), 17
local_fs (LocalRemoteSync attribute), 26
local_path_str (LocalRemoteSync attribute), 26
LocalRemoteSync (class in
data_pipelines_cli.filesystem_utils), 26
logs_generator (DockerResponseReader attribute), 24

M

macro_name (MacroArgName attribute), 14
MacroArgName (class in
data_pipelines_cli.cli_commands.generate.model_yaml), 14
message (AirflowDagsPathKeyError attribute), 24
message (DataPipelinesError attribute), 24
message (DependencyNotInstalledError attribute), 24
message (DockerErrorResponseError attribute), 24
message (DockerNotInstalledError attribute), 25
message (JinjaVarKeyError attribute), 25
message (NoConfigFileError attribute), 25
message (NotAProjectDirectoryError attribute), 25
message (SubprocessNonZeroExitError attribute), 25
message (SubprocessNotFound attribute), 26
meta (DbtModel attribute), 21
meta (DbtSource attribute), 21
meta (DbtTableColumn attribute), 22
MODEL_PATH
dp-generate-model-yaml command line
option, 11
module
data_pipelines_cli, 14
data_pipelines_cli.cli, 18
data_pipelines_cli.cli_commands, 14
data_pipelines_cli.cli_commands.clean, 15
data_pipelines_cli.cli_commands.compile, 15
data_pipelines_cli.cli_commands.create, 16
data_pipelines_cli.cli_commands.deploy, 16
data_pipelines_cli.cli_commands.docs, 16
data_pipelines_cli.cli_commands.generate, 14
data_pipelines_cli.cli_commands.generate.generate, data_pipelines_cli.cli_constants), 18
data_pipelines_cli.cli_commands.generate.model_yaml, 14
data_pipelines_cli.cli_commands.generate.model_yaml, 14
data_pipelines_cli.cli_commands.generate.source_yaml, 15

data_pipelines_cli.cli_commands.generate.source_yaml, 15
data_pipelines_cli.cli_commands.generate.utils, 15
data_pipelines_cli.cli_commands.init, 17
data_pipelines_cli.cli_commands.prepare_env, 17
data_pipelines_cli.cli_commands.publish, 17
data_pipelines_cli.cli_commands.run, 17
data_pipelines_cli.cli_commands.template, 17
data_pipelines_cli.cli_commands.test, 17
data_pipelines_cli.cli_commands.update, 18
data_pipelines_cli.cli_configs, 18
data_pipelines_cli.cli_constants, 18
data_pipelines_cli.cli_utils, 18
data_pipelines_cli.config_generation, 20
data_pipelines_cli.data_structures, 21
data_pipelines_cli.dbt_utils, 23
data_pipelines_cli.docker_response_reader, 23
data_pipelines_cli.errors, 24
data_pipelines_cli.filesystem_utils, 26
data_pipelines_cli.io_utils, 26
data_pipelines_cli.jinja, 27
data_pipelines_cli.vcs_utils, 27
msg (DockerReadResponse attribute), 23

N

name (DbtModel attribute), 21
name (DbtSource attribute), 21
name (DbtTableColumn attribute), 22
NoConfigFileError, 25
NotAProjectDirectoryError, 25

O

outputs (DbtProfile attribute), 20

P

prepare_env() (in module
data_pipelines_cli.cli_commands.prepare_env), 17
PROFILE_NAME_ENV_EXECUTION (in module
data_pipelines_cli.cli_constants), 18
PROFILE_NAME_LOCAL_ENVIRONMENT (in module
data_pipelines_cli.cli_constants), 18
PROJECT_PATH
dp-generate command line option, 9
dp-update command line option, 14
provide_kwargs_dict (DeployCommand attribute), 16

- publish_package() (in module data_pipelines_cli.cli_commands.publish), 17
- ## Q
- quote (DbtTableColumn attribute), 22
- ## R
- read_dbt_vars_from_configs() (in module data_pipelines_cli.dbt_utils), 23
- read_dictionary_from_config_directory() (in module data_pipelines_cli.config_generation), 20
- read_env_config() (in module data_pipelines_cli.data_structures), 22
- read_response() (DockerResponseReader method), 24
- remote_path_str (LocalRemoteSync attribute), 26
- replace() (in module data_pipelines_cli.io_utils), 26
- replace_image_settings() (in module data_pipelines_cli.cli_commands.compile), 15
- replace_vars_with_values() (in module data_pipelines_cli.jinja), 27
- repository (DockerArgs attribute), 22
- run() (in module data_pipelines_cli.cli_commands.run), 17
- run_dbt_command() (in module data_pipelines_cli.dbt_utils), 23
- ## S
- schema (DbtSource attribute), 21
- SCHEMA_NAME
- dp-generate-source-yaml command line option, 12
- submessage (AirflowDagsPathKeyError attribute), 24
- submessage (DataPipelinesError attribute), 24
- submessage (DependencyNotInstalledError attribute), 24
- submessage (DockerErrorResponseError attribute), 24
- submessage (DockerNotInstalledError attribute), 25
- submessage (JinjaVarKeyError attribute), 25
- submessage (NoConfigFileError attribute), 25
- submessage (NotAProjectDirectoryError attribute), 25
- submessage (SubprocessNonZeroExitError attribute), 25
- submessage (SubprocessNotFound attribute), 26
- subprocess_run() (in module data_pipelines_cli.cli_utils), 19
- SubprocessNonZeroExitError, 25
- SubprocessNotFound, 25
- sync() (LocalRemoteSync method), 26
- ## T
- tags (DbtModel attribute), 21
- tags (DbtSource attribute), 21
- tags (DbtTableColumn attribute), 22
- target (DbtProfile attribute), 20
- template_name (TemplateConfig attribute), 22
- TEMPLATE_PATH
- dp-create command line option, 9
- template_path (TemplateConfig attribute), 22
- TemplateConfig (class in data_pipelines_cli.data_structures), 22
- templates (DataPipelinesConfig attribute), 21
- test() (in module data_pipelines_cli.cli_commands.test), 17
- tests (DbtModel attribute), 21
- tests (DbtTableColumn attribute), 22
- ## U
- update() (in module data_pipelines_cli.cli_commands.update), 18
- ## V
- vars (DataPipelinesConfig attribute), 21