

---

# **data-pipelines-cli**

**GetInData**

**Jan 12, 2022**



**CONTENTS:**

<b>1</b>	<b>Installation</b>	<b>3</b>
	<b>Python Module Index</b>	<b>23</b>
	<b>Index</b>	<b>25</b>







## 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 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
```

(continues on next page)



(continued from previous page)

```
# 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**.

### 1.1.3 Project compilation

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

### 1.1.4 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.

### 1.1.5 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 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 `config/base/airflow.yml` file, as a `dags_path` argument:

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

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

### 1.1.6 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 a 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.

### 1.1.7 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

### 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

**--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

### 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

## 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

## 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

#### Subpackages

#### data\_pipelines\_cli.cli\_commands package

#### Submodules

#### data\_pipelines\_cli.cli\_commands.clean module

**clean()** → None  
Deletes local working directories

#### data\_pipelines\_cli.cli\_commands.compile module

**compile\_project**(*env: str, docker\_build: bool = False*) → None  
Create local working directories and build artifacts

##### Parameters

- **env** (*str*) – Name of the environment
- **docker\_build** (*bool*) – Whether to build a Docker image

**Raises** *DataPipelinesError* –

#### 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

**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.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

Prepares 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.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

## Submodules

### data\_pipelines\_cli.cli module

**cli**() → None

### 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\_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*) → strGiven *argument* is not None, returns its value. Otherwise, searches for *environment\_variable\_name* amongst environment variables and returns it. If such a variable is not set, raises [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]*) → subprocess.CompletedProcess[bytes]Runs subprocess and returns its state if completed with a success. If not, raises [SubprocessNonZeroExitError](#).**Parameters** **args** (*List[str]*) – List of strings representing subprocess and its arguments**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 copies *config* directory to *build/dag/config* working directory**copy\_dag\_dir\_to\_build\_dir()** → NoneRecursively copies *dag* directory to *build/dag* working directory**generate\_profiles\_dict**(*env: str, copy\_config\_dir: bool*) → Dict[str, *data\_pipelines\_cli.config\_generation.DbtProfile*]Generates and saves *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.Path*Generates and saves *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.Path*Returns 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]Reads dictionaries out of *file\_name* in both *base* and *env* directories, and compiles 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** DockerArgs(*env: 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

**commit\_sha:** str

Long hash of the current Git revision. Used as an image tag

**docker\_build\_tag()** → str

**Returns** Tag for Docker Python API build command.

**Return type** str

**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\_config()** → [data\\_pipelines\\_cli.data\\_structures.DataPipelinesConfig](#)

Parses *.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]

Reads *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 compiles 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*) → None

Runs 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

**Raises**

- *SubprocessNotFound* – dbt not installed
- *SubprocessNonZeroExitError* – dbt exited with error

### 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

**exception DataPipelinesError**(*message: str*)

Bases: Exception

Base class for all exceptions in *data\_pipelines\_cli* module

**message:** *str*

explanation of 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

**exception DockerNotInstalledError**

Bases: *data\_pipelines\_cli.errors.DependencyNotInstalledError*

Exception raised if ‘docker’ is not installed

**message:** *str*

explanation of the error

**exception JinjaVarKeyError**(*key: str*)  
 Bases: `data_pipelines_cli.errors.DataPipelinesError`  
**message: str**  
 explanation of the error

**exception NoConfigFileError**  
 Bases: `data_pipelines_cli.errors.DataPipelinesError`  
 Exception raised if `.dp.yml` does not exist  
**message: str**  
 explanation of the error

**exception SubprocessNonZeroExitError**(*subprocess\_name: str, exit\_code: int*)  
 Bases: `data_pipelines_cli.errors.DataPipelinesError`  
 Exception raised if subprocess exits with non-zero exit code  
**message: str**  
 explanation of 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

### `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*) → `None`  
 Sends local files to the remote directory and (optionally) deletes 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()** → Optional[str]

Tries to 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) → NonePerform 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.vcs\_utils module****add\_suffix\_to\_git\_template\_path**(template\_path: str) → strAdds `.git` suffix to *template\_path*, if necessary.Checks if *template\_path* starts with Git-specific prefix (e.g. `git://`), or `http://` or `https://` protocol. If so, then adds `.git` suffix if not present. Otherwise, it does not (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 0.10.0 - 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.

### 1.4.3 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`

### 1.4.4 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.

### 1.4.5 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]`).

### 1.4.6 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.

### 1.4.7 0.5.1 - 2021-12-14

#### Fixed

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

### 1.4.8 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.

### 1.4.9 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.

### 1.4.10 0.3.0 - 2021-12-06

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

### 1.4.11 0.2.0 - 2021-12-03

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

### 1.4.12 0.1.2 - 2021-12-02

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

### 1.4.13 0.1.1 - 2021-12-01

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

### 1.4.14 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`, 10
- `data_pipelines_cli.cli`, 12
- `data_pipelines_cli.cli_commands`, 10
- `data_pipelines_cli.cli_commands.clean`, 10
- `data_pipelines_cli.cli_commands.compile`, 10
- `data_pipelines_cli.cli_commands.create`, 10
- `data_pipelines_cli.cli_commands.deploy`, 11
- `data_pipelines_cli.cli_commands.init`, 11
- `data_pipelines_cli.cli_commands.prepare_env`, 11
- `data_pipelines_cli.cli_commands.run`, 12
- `data_pipelines_cli.cli_commands.template`, 12
- `data_pipelines_cli.cli_commands.test`, 12
- `data_pipelines_cli.cli_constants`, 12
- `data_pipelines_cli.cli_utils`, 13
- `data_pipelines_cli.config_generation`, 14
- `data_pipelines_cli.data_structures`, 15
- `data_pipelines_cli.dbt_utils`, 16
- `data_pipelines_cli.errors`, 16
- `data_pipelines_cli.filesystem_utils`, 17
- `data_pipelines_cli.io_utils`, 18
- `data_pipelines_cli.vcs_utils`, 18



## Symbols

--blob-args <blob\_args>  
     dp-deploy command line option, 8  
 --dags-path <dags\_path>  
     dp-deploy command line option, 8  
 --datahub-ingest  
     dp-deploy command line option, 8  
 --docker-build  
     dp-compile command line option, 7  
 --docker-push  
     dp-deploy command line option, 8  
 --env <env>  
     dp-compile command line option, 7  
     dp-deploy command line option, 8  
     dp-prepare-env command line option, 9  
     dp-run command line option, 9  
     dp-test command line option, 10  
 --version  
     dp command line option, 7

## A

add\_suffix\_to\_git\_template\_path() (in module  
     data\_pipelines\_cli.vcs\_utils), 18  
 AirflowDagsPathKeyError, 16

## B

blob\_address\_path (DeployCommand attribute), 11

## C

clean() (in module data\_pipelines\_cli.cli\_commands.clean),  
     10  
 cli() (in module data\_pipelines\_cli.cli), 12  
 commit\_sha (DockerArgs attribute), 15  
 compile\_project() (in module  
     data\_pipelines\_cli.cli\_commands.compile),  
     10  
 CONFIG\_PATH  
     dp-init command line option, 9  
 copy\_config\_dir\_to\_build\_dir() (in module  
     data\_pipelines\_cli.config\_generation), 14  
 copy\_dag\_dir\_to\_build\_dir() (in module  
     data\_pipelines\_cli.config\_generation), 14

create() (in module data\_pipelines\_cli.cli\_commands.create),  
     10

## D

data\_pipelines\_cli  
     module, 10  
 data\_pipelines\_cli.cli  
     module, 12  
 data\_pipelines\_cli.cli\_commands  
     module, 10  
 data\_pipelines\_cli.cli\_commands.clean  
     module, 10  
 data\_pipelines\_cli.cli\_commands.compile  
     module, 10  
 data\_pipelines\_cli.cli\_commands.create  
     module, 10  
 data\_pipelines\_cli.cli\_commands.deploy  
     module, 11  
 data\_pipelines\_cli.cli\_commands.init  
     module, 11  
 data\_pipelines\_cli.cli\_commands.prepare\_env  
     module, 11  
 data\_pipelines\_cli.cli\_commands.run  
     module, 12  
 data\_pipelines\_cli.cli\_commands.template  
     module, 12  
 data\_pipelines\_cli.cli\_commands.test  
     module, 12  
 data\_pipelines\_cli.cli\_constants  
     module, 12  
 data\_pipelines\_cli.cli\_utils  
     module, 13  
 data\_pipelines\_cli.config\_generation  
     module, 14  
 data\_pipelines\_cli.data\_structures  
     module, 15  
 data\_pipelines\_cli.dbt\_utils  
     module, 16  
 data\_pipelines\_cli.errors  
     module, 16  
 data\_pipelines\_cli.filesystem\_utils  
     module, 17

- data\_pipelines\_cli.io\_utils  
     module, 18  
 data\_pipelines\_cli.vcs\_utils  
     module, 18  
 datahub\_ingest (*DeployCommand* attribute), 11  
 DataPipelinesConfig (class in *data\_pipelines\_cli.data\_structures*), 15  
 DataPipelinesError, 16  
 DbtProfile (class in *data\_pipelines\_cli.config\_generation*), 14  
 DEFAULT\_GLOBAL\_CONFIG (in *data\_pipelines\_cli.cli\_constants*), 12  
 DependencyNotInstalledError, 16  
 deploy() (*DeployCommand* method), 11  
 DeployCommand (class in *data\_pipelines\_cli.cli\_commands.deploy*), 11  
 docker\_args (*DeployCommand* attribute), 11  
 docker\_build\_tag() (*DockerArgs* method), 15  
 DockerArgs (class in *data\_pipelines\_cli.data\_structures*), 15  
 DockerNotInstalledError, 16  
 dp command line option  
     --version, 7  
 dp-compile command line option  
     --docker-build, 7  
     --env <env>, 7  
 dp-create command line option  
     PROJECT\_PATH, 8  
     TEMPLATE\_PATH, 8  
 dp-deploy command line option  
     --blob-args <blob\_args>, 8  
     --dags-path <dags\_path>, 8  
     --datahub-ingest, 8  
     --docker-push, 8  
     --env <env>, 8  
 dp-init command line option  
     CONFIG\_PATH, 9  
 dp-prepare-env command line option  
     --env <env>, 9  
 dp-run command line option  
     --env <env>, 9  
 dp-test command line option  
     --env <env>, 10
- ## E
- echo\_error() (in *module data\_pipelines\_cli.cli\_utils*), 13  
 echo\_info() (in *module data\_pipelines\_cli.cli\_utils*), 13  
 echo\_subinfo() (in *module data\_pipelines\_cli.cli\_utils*), 13  
 echo\_warning() (in *module data\_pipelines\_cli.cli\_utils*), 13
- ## G
- generate\_profiles\_dict() (in *module data\_pipelines\_cli.config\_generation*), 14  
 generate\_profiles\_yaml() (in *module data\_pipelines\_cli.config\_generation*), 14  
 get\_argument\_or\_environment\_variable() (in *module data\_pipelines\_cli.cli\_utils*), 13  
 get\_dbt\_profiles\_env\_name() (in *module data\_pipelines\_cli.cli\_constants*), 12  
 get\_profiles\_dir\_build\_path() (in *module data\_pipelines\_cli.config\_generation*), 14  
 git\_revision\_hash() (in *module data\_pipelines\_cli.io\_utils*), 18
- ## I
- IMAGE\_TAG\_TO\_REPLACE (in *module data\_pipelines\_cli.cli\_constants*), 12  
 init() (in *module data\_pipelines\_cli.cli\_commands.init*), 11
- ## J
- JinjaVarKeyError, 16
- ## L
- list\_templates() (in *module data\_pipelines\_cli.cli\_commands.template*), 12  
 local\_fs (*LocalRemoteSync* attribute), 17  
 local\_path\_str (*LocalRemoteSync* attribute), 17  
 LocalRemoteSync (class in *data\_pipelines\_cli.filesystem\_utils*), 17
- ## M
- message (*AirflowDagsPathKeyError* attribute), 16  
 message (*DataPipelinesError* attribute), 16  
 message (*DependencyNotInstalledError* attribute), 16  
 message (*DockerNotInstalledError* attribute), 16  
 message (*JinjaVarKeyError* attribute), 17  
 message (*NoConfigFileError* attribute), 17  
 message (*SubprocessNonZeroExitError* attribute), 17  
 message (*SubprocessNotFound* attribute), 17  
 module  
     data\_pipelines\_cli, 10  
     data\_pipelines\_cli.cli, 12  
     data\_pipelines\_cli.cli\_commands, 10  
     data\_pipelines\_cli.cli\_commands.clean, 10  
     data\_pipelines\_cli.cli\_commands.compile, 10  
     data\_pipelines\_cli.cli\_commands.create, 10  
     data\_pipelines\_cli.cli\_commands.deploy, 11  
     data\_pipelines\_cli.cli\_commands.init, 11

[data\\_pipelines\\_cli.cli\\_commands.prepare\\_env](#), 11  
[data\\_pipelines\\_cli.cli\\_commands.run](#), 12  
[data\\_pipelines\\_cli.cli\\_commands.template](#), 12  
[data\\_pipelines\\_cli.cli\\_commands.test](#), 12  
[data\\_pipelines\\_cli.cli\\_constants](#), 12  
[data\\_pipelines\\_cli.cli\\_utils](#), 13  
[data\\_pipelines\\_cli.config\\_generation](#), 14  
[data\\_pipelines\\_cli.data\\_structures](#), 15  
[data\\_pipelines\\_cli.dbt\\_utils](#), 16  
[data\\_pipelines\\_cli.errors](#), 16  
[data\\_pipelines\\_cli.filesystem\\_utils](#), 17  
[data\\_pipelines\\_cli.io\\_utils](#), 18  
[data\\_pipelines\\_cli.vcs\\_utils](#), 18

## N

[NoConfigFileError](#), 17

## O

[outputs](#) (*DbtProfile* attribute), 14

## P

[prepare\\_env\(\)](#) (in module [data\\_pipelines\\_cli.cli\\_commands.prepare\\_env](#)), 11  
[PROFILE\\_NAME\\_ENV\\_EXECUTION](#) (in module [data\\_pipelines\\_cli.cli\\_constants](#)), 12  
[PROFILE\\_NAME\\_LOCAL\\_ENVIRONMENT](#) (in module [data\\_pipelines\\_cli.cli\\_constants](#)), 12  
[PROJECT\\_PATH](#)  
 dp-create command line option, 8  
[provider\\_kwargs\\_dict](#) (*DeployCommand* attribute), 11

## R

[read\\_config\(\)](#) (in module [data\\_pipelines\\_cli.data\\_structures](#)), 15  
[read\\_dbt\\_vars\\_from\\_configs\(\)](#) (in module [data\\_pipelines\\_cli.dbt\\_utils](#)), 16  
[read\\_dictionary\\_from\\_config\\_directory\(\)](#) (in module [data\\_pipelines\\_cli.config\\_generation](#)), 14  
[remote\\_path\\_str](#) (*LocalRemoteSync* attribute), 17  
[replace\(\)](#) (in module [data\\_pipelines\\_cli.io\\_utils](#)), 18  
[repository](#) (*DockerArgs* attribute), 15  
[run\(\)](#) (in module [data\\_pipelines\\_cli.cli\\_commands.run](#)), 12  
[run\\_dbt\\_command\(\)](#) (in module [data\\_pipelines\\_cli.dbt\\_utils](#)), 16

## S

[subprocess\\_run\(\)](#) (in module [data\\_pipelines\\_cli.cli\\_utils](#)), 13

[SubprocessNonZeroExitError](#), 17  
[SubprocessNotFound](#), 17  
[sync\(\)](#) (*LocalRemoteSync* method), 17

## T

[target](#) (*DbtProfile* attribute), 14  
[template\\_name](#) (*TemplateConfig* attribute), 15  
[TEMPLATE\\_PATH](#)  
 dp-create command line option, 8  
[template\\_path](#) (*TemplateConfig* attribute), 15  
[TemplateConfig](#) (class in [data\\_pipelines\\_cli.data\\_structures](#)), 15  
[templates](#) (*DataPipelinesConfig* attribute), 15  
[test\(\)](#) (in module [data\\_pipelines\\_cli.cli\\_commands.test](#)), 12

## V

[vars](#) (*DataPipelinesConfig* attribute), 15