data-pipelines-cli

GetInData

Dec 31, 2021

CONTENTS:

CHAPTER

ONE

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.tmpl` 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.tmpl` 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.tmpl` 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-repository-uri <docker_repository_uri>
URI of the Docker repository

--datahub-gms-uri <datahub_gms_uri>
URI of the DataHub ingestion endpoint

--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

--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 <docker_push>
Path 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_repository_uri: Optional[str] = None*, *datahub_gms_uri: Optional[str] = None*, *docker_build: bool = False*) → None
Create local working directories and build artifacts

Parameters

- **env** (*str*) – Name of the environment
- **docker_repository_uri** (*Optional[str]*) – URI of the Docker repository
- **datahub_gms_uri** (*Optional[str]*) – URI of the DataHub ingestion endpoint
- **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(*docker_push: Optional[str]*, *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

DATAHUB_URL_ENV: str = 'DATAHUB_URL'
DataHub URL environment variable to search for

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

DOCKER_REPOSITORY_URL_TO_REPLACE: str = '<DOCKER_REPOSITORY_URL>'

IMAGE_TAG_TO_REPLACE: str = '<IMAGE_TAG>'

INGEST_ENDPOINT_TO_REPLACE: str = '<INGEST_ENDPOINT>'

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*) → str
Given *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: str
Name of the *target* for dbt to run

copy_config_dir_to_build_dir() → None
Recursively copies *config* directory to *build/dag/config* working directory

copy_dag_dir_to_build_dir() → None
Recursively 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_yml(*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_yml_build_path(*env: str*) → pathlib.Path
Returns path to `build/profiles/<profile_name>/profiles.yml`, 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(*docker_repository_uri: 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

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*) → 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.vcs_utils module

`add_suffix_to_git_template_path`(*template_path: str*) → str
Adds `.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 0.8.0 - 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.

1.4.3 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]`).

1.4.4 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.

1.4.5 0.5.1 - 2021-12-14

Fixed

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

1.4.6 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.

1.4.7 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.

1.4.8 0.3.0 - 2021-12-06

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

1.4.9 0.2.0 - 2021-12-03

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

1.4.10 0.1.2 - 2021-12-02

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

1.4.11 0.1.1 - 2021-12-01

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

1.4.12 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

INDEX

Symbols

A

B

C

D

E

G

I

J

L

M

N

O

P

R

S

T

V