Version: 1.10.0 (latest)

Load data from a REST API

This tutorial demonstrates how to extract data from a REST API using dlt's REST API source and load it into a destination. You will learn how to build a data pipeline that loads data from the Pokemon and the GitHub API into a local DuckDB database.

Extracting data from an API is straightforward with dlt: provide the base URL, define the resources you want to fetch, and dlt will handle the pagination, authentication, and data loading.

What you will learn

- How to set up a REST API source
- Configuration basics for API endpoints
- Configuring the destination database
- Relationships between different resources
- How to append, replace, and merge data in the destination
- Loading data incrementally by fetching only new or updated data

Prerequisites

- Python 3.9 or higher installed
- Virtual environment set up

Installing dlt

Before we start, make sure you have a Python virtual environment set up. Follow the instructions in the installation guide to create a new virtual environment and install dlt.

Verify that dlt is installed by running the following command in your terminal:

Get GPT-4 Help

```
dlt --version
```

If you see the version number (such as "dlt 0.5.3"), you're ready to proceed.

Setting up a new project

Initialize a new dlt project with a REST API source and DuckDB destination:

```
dlt init rest_api duckdb
```

`dlt init` creates multiple files and a directory for your project. Let's take a look at the project structure:

```
rest_api_pipeline.py
requirements.txt
.dlt/
    config.toml
    secrets.toml
```

Here's what each file and directory contains:

- `rest_api_pipeline.py`: This is the main script where you'll define your data pipeline. It contains two basic pipeline examples for Pokemon and GitHub APIs. You can modify or rename this file as needed.
- `requirements.txt`: This file lists all the Python dependencies required for your project.
- `.dlt/`: This directory contains the configuration files for your project:
  - `secrets.toml`: This file stores your API keys, tokens, and other sensitive information.
  - `config.toml`: This file contains the configuration settings for your dlt project.

Installing dependencies

Before we proceed, let's install the required dependencies for this tutorial. Run the following command to install the dependencies listed in the `requirements.txt` file:

```
pip install -r requirements.txt
```

Running the pipeline

Let's verify that the pipeline is working as expected. Run the following command to execute the pipeline:

```
python rest_api_pipeline.py
```

You should see the output of the pipeline execution in the terminal. The output will also display the location of the DuckDB database file where the data is stored:

```
Pipeline rest_api_pokemon load step completed in 1.08 seconds
1 load package(s) were loaded to destination duckdb and into dataset
rest_api_data
The duckdb destination used duckdb:////home/user-
name/quick_start/rest_api_pokemon.duckdb location to store data
Load package 1692364844.9254808 is LOADED and contains no failed jobs
```

Exploring the data

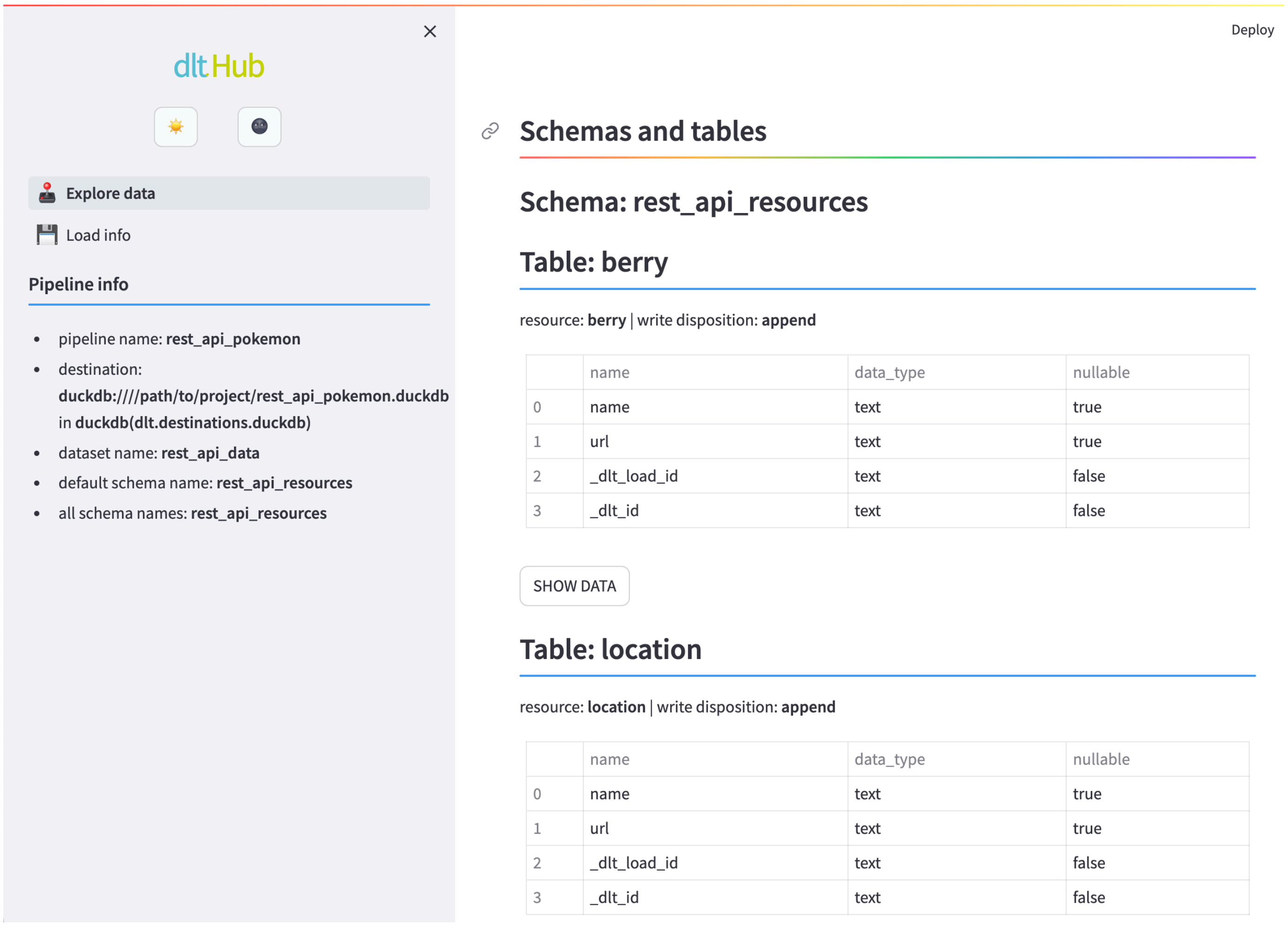
Now that the pipeline has run successfully, let's explore the data loaded into DuckDB. dlt comes with a built-in browser application that allows you to interact with the data. To enable it, run the following command:

```
pip install streamlit
```

Next, run the following command to start the data browser:

```
dlt pipeline rest_api_pokemon show
```

The command opens a new browser window with the data browser application. `rest_api_pokemon` is the name of the pipeline defined in the `rest_api_pipeline.py` file. You can explore the loaded data, run queries, and see some pipeline execution details:



Pipeline info

- pipeline name: **rest_api_pokemon**
- destination: **duckdb:////path/to/project/rest_api_pokemon.duckdb** in **duckdb(dlt.destinations.duckdb)**
- dataset name: **rest_api_data**
- default schema name: **rest_api_resources**
- all schema names: **rest_api_resources**

Schemas and tables

Schema: rest_api_resources

Table: berry

resource: **berry** | write disposition: **append**

	name	data_type	nullable
0	name	text	true
1	url	text	true
2	_dlt_load_id	text	false
3	_dlt_id	text	false

SHOW DATA

Table: location

resource: **location** | write disposition: **append**

	name	data_type	nullable
0	name	text	true
1	url	text	true
2	_dlt_load_id	text	false
3	_dlt_id	text	false

Configuring the REST API source

Now that your environment and the project are set up, let's take a closer look at the configuration of the REST API source. Open the `rest_api_pipeline.py` file in your code editor and locate the following code snippet:

```
import dlt
from dlt.sources.rest_api import rest_api_source

def load_pokemon() -> None:
    pipeline = dlt.pipeline(
        pipeline_name="rest_api_pokemon",
        destination="duckdb",
        dataset_name="rest_api_data",
    )

    pokemon_source = rest_api_source(
        {
            "client": {
                "base_url": "https://pokeapi.co/api/v2/"
            },
            "resource_defaults": {
                "endpoint": {
                    "params": {
                        "limit": 1000,
                    },
                },
            },
            "resources": [
                "pokemon",
                "berry",
                "location",
            ],
        }
    )

    ...

    load_info = pipeline.run(pokemon_source)
    print(load_info)
```

Here's what's happening in the code:

1. With `dlt.pipeline()`, we define a new pipeline named `rest_api_pokemon` with DuckDB as the destination and `rest_api_data` as the dataset name.
2. The `rest_api_source()` function creates a new REST API source object.
3. We pass this source object to the `pipeline.run()` method to start the pipeline execution. Inside the `run()` method, dlt will fetch data from the API and load it into the DuckDB database.
4. The `print(load_info)` outputs the pipeline execution details to the console.

Let's break down the configuration of the REST API source. It consists of three main parts: `client`, `resource_defaults`, and `resources`.

```
config: RESTAPIConfig = {
    "client": {
        # ...
    },
    "resource_defaults": {
        # ...
    },
    "resources": [
        # ...
    ],
}
```

- The `client` configuration is used to connect to the web server and authenticate if necessary. For our simple example, we only need to specify the `base_url` of the API: `https://pokeapi.co/api/v2/`.
- The `resource_defaults` configuration allows you to set default parameters for all resources. Normally, you would set common parameters here, such as pagination limits. In our Pokemon API example, we set the `limit` parameter to 1000 for all resources to retrieve more data in a single request and reduce the number of HTTP API calls.
- The `resources` list contains the names of the resources you want to load from the API. REST API will use some conventions to determine the endpoint URL based on the resource name. For example, the resource name `pokemon` will be translated to the endpoint URL `https://pokeapi.co/api/v2/pokemon`.

NOTE

Pagination

You may have noticed that we didn't specify any pagination configuration in the `rest_api_source()` function. That's because for REST APIs that follow best practices, dlt can automatically detect and handle pagination. Read more about configuring pagination in the REST API source documentation.

Appending, replacing, and merging loaded data

Try running the pipeline again with `python rest_api_pipeline.py`. You will notice that all the tables have duplicated data. This happens because, by default, dlt appends the data to the destination table. In dlt, you can control how the data is loaded into the destination table by setting the `write_disposition` parameter in the resource configuration. The possible values are:

- `append`: Appends the data to the destination table. This is the default.
- `replace`: Replaces the data in the destination table with the new data.
- `merge`: Merges the new data with the existing data in the destination table based on the primary key.

Replacing the data

In our case, we don't want to append the data every time we run the pipeline. Let's start with the simpler `replace` write disposition.

To change the write disposition to `replace`, update the `resource_defaults` configuration in the `rest_api_pipeline.py` file:

```
...
pokemon_source = rest_api_source(
    {
        "client": {
            "base_url": "https://pokeapi.co/api/v2/",
        },
        "resource_defaults": {
            "endpoint": {
                "params": {
                    "limit": 1000,
                },
            },
            "write_disposition": "replace", # Setting the write disposition
to `replace`
        },
        "resources": [
            "pokemon",
            "berry",
            "location",
        ],
    }
)
...
```

Run the pipeline again with `python rest_api_pipeline.py`. This time, the data will be replaced in the destination table instead of being appended.

Merging the data

When you want to update the existing data as new data is loaded, you can use the `merge` write disposition. This requires specifying a primary key for the resource. The primary key is used to match the new data with the existing data in the destination table.

Let's update our example to use the `merge` write disposition. We need to specify the primary key for the `pokemon` resource and set the write disposition to `merge`:

```
...
pokemon_source = rest_api_source(
    {
        "client": {
            "base_url": "https://pokeapi.co/api/v2/",
        },
        "resource_defaults": {
            "endpoint": {
                "params": {
                    "limit": 1000,
                },
            },
            # For the `berry` and `location` resources, we keep
            # the `replace` write disposition
            "write_disposition": "replace",
        },
        "resources": [
            # We create a specific configuration for the `pokemon` resource
            # using a dictionary instead of a string to configure
            # the primary key and write disposition
            {
                "name": "pokemon",
                "primary_key": "name",
                "write_disposition": "merge",
            },
            # The `berry` and `location` resources will use the default
            "berry",
            "location",
        ],
    }
)
```

Run the pipeline with `python rest_api_pipeline.py`, the data for the `pokemon` resource will be merged with the existing data in the destination table based on the `name` field.

Loading data incrementally

When working with some APIs, you may need to load data incrementally to avoid fetching the entire dataset every time and to reduce the load time. APIs that support incremental loading usually provide a way to fetch only new or changed data (most often by using a timestamp field like `updated_at`, `created_at`, or incremental IDs).

To illustrate incremental loading, let's consider the GitHub API. In the `rest_api_pipeline.py` file, you can find an example of how to load data from the GitHub API incrementally. Let's take a look at the configuration:

```python
import dlt
from dlt.sources.rest_api import rest_api_source

pipeline = dlt.pipeline(
    pipeline_name="rest_api_github",
    destination="duckdb",
    dataset_name="rest_api_data",
)

github_source = rest_api_source({
    "client": {
        "base_url": "https://api.github.com/repos/dlt-hub/dlt/",
    },
    "resource_defaults": {
        "primary_key": "id",
        "write_disposition": "merge",
        "endpoint": {
            "params": {
                "per_page": 100,
            },
        },
    },
    "resources": [
        {
            "name": "issues",
            "endpoint": {
                "path": "issues",
                "params": {
                    "sort": "updated",
                    "direction": "desc",
                    "state": "open",
                    "since": {
                        "type": "incremental",
                        "cursor_path": "updated_at",
                        "initial_value": "2024-01-25T11:21:28Z",
                    },
                },
            },
        },
    ],
})

load_info = pipeline.run(github_source)
print(load_info)
```

In this configuration, the `since` parameter is defined as a special incremental parameter. The `cursor_path` field specifies the JSON path to the field that will be used to fetch the updated data, and we use the `initial_value` for the initial value for the incremental parameter. This value will be used in the first request to fetch the data.

When the pipeline runs, dlt will automatically update the `since` parameter with the latest value from the response data. This way, you can fetch only the new or updated data from the API.

Read more about incremental loading in the REST API source documentation.

What's next?

Congratulations on completing the tutorial! You've learned how to set up a REST API source in dlt and run a data pipeline to load the data into DuckDB.

Interested in learning more about dlt? Here are some suggestions:

- Learn more about the REST API source configuration in the REST API source documentation
- Learn how to create a custom source in the advanced tutorial.

Edit this page