ChatGPT

Running the `devinpoc` PySpark/Luigi demo

Repository overview

The `devinpoc` repository under `ltibfspoc` contains a proof-of-concept data-processing pipeline built with **PySpark** for distributed data transformations and **Luigi** for workflow orchestration. The project is structured as a Python package in the `src` directory. Key sub-modules include:

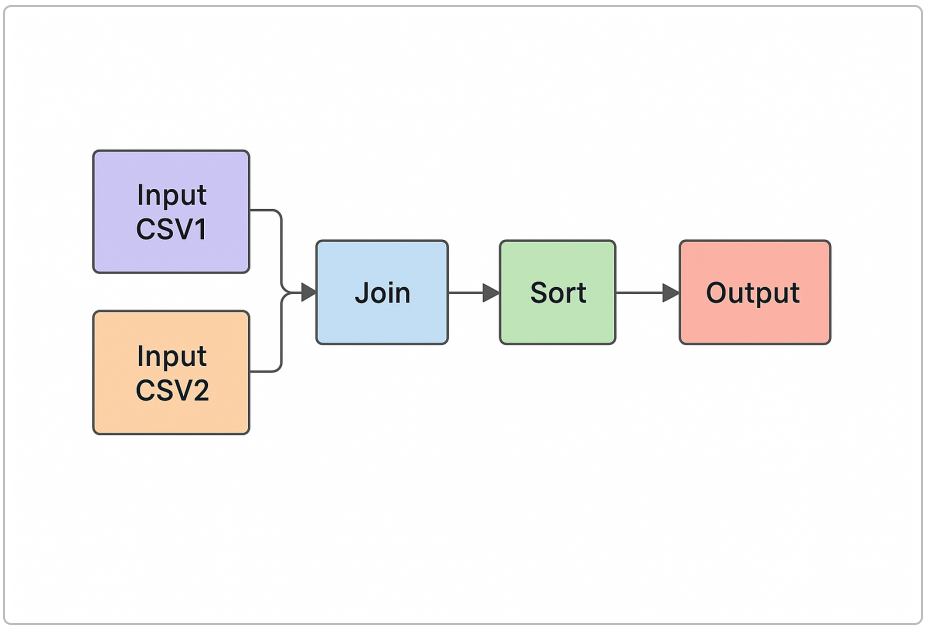
Directory/File	Purpose
`src/application.properties`	Defines where the pipeline will look for its JSON configuration. The file declares a section `[basicdetail]` with properties `jsonfilepath` and `jsonfilename`. In the current repository it points to the developer's local Windows path (`C:\\Users\\…\\src\\config`) [1]; this should be updated to a path on your machine.
`src/config/…`	Holds JSON files (`test1.json`, `test3.json`, etc.) that describe the tasks to perform. Each JSON file contains a list of **components** – such as input file readers, joins, sorts and partitions. For example, `test3.json` declares two input CSV components, a join on `Country` and `Origin`, a sort on `Country` and a partition step [2].
`src/data/…`	Sample data and schemas used by the JSON examples. `cars.csv`, `factbook.csv` and `carSchema.txt` are referenced in the JSON configs.
`src/etl/SparkMain.py`	The script that boots the pipeline. Its docstring explains that it starts a Spark session, parses the JSON configuration and then executes the operations defined in the JSON [3]. When run as a script it reads `src/application.properties`, builds the full path to the JSON file, then loads it and orchestrates tasks [4]. The call to Luigi (`luigi.build(...)`) is included but commented out, so you'll need to remove the triple-quotes to actually run the workflow.
`src/etl/SparkDriver.py`	Provides a `DriverProgram` class that converts the list of JSON components into a dependency graph and resolves the execution order. It uses helper functions in `ComponentDriver.py` and populates lists such as `tasks`, `tasksId` and `dependencylist` [5].

Directory/File	Purpose
`src/util/…`	Utility functions. `getSpark.py` defines a `get_spark_session()` function that creates a local Spark session using `SparkSession.builder.master("local[*]")` [6]. `read_json.py` contains `read_json_config()` that reads a JSON file and converts each component into a `Component` object [7].
`src/workflow/luigi.cfg`	Configuration for Luigi. It sets `check_unfulfilled_deps=False` and `check_complete_on_run=True` [8], telling Luigi not to fail when dependencies are missing and to check task completion on run.

How the pipeline works

1. **Configuration via JSON** – The pipeline is driven by JSON files in `src/config`. A JSON file's `component` array describes each processing step. In `test3.json`, for example, two input CSV components read `factbook.csv` and `cars.csv`; a join component specifies an inner join on `Country` and `Origin`; this is followed by a sort on `Country` and a partition step [2].
2. **Reading configuration** – When you run `SparkMain.py`, it loads `src/application.properties`, reads the `jsonfilepath` and `jsonfilename` values and then builds a full path to the JSON configuration [4]. It loads the JSON using `read_json.read_json_config()` [9], which turns each component into a Python object [7].
3. **Creating the Spark session** – The project uses the `get_spark_session()` function in `src/util/getSpark.py` to create a local Spark session. The builder specifies a local master (`local[*]`) and enables Hive support [6].
4. **Building the dependency graph** – `DriverProgram.getAllConfig()` resolves dependencies between components. It generates dictionaries of tasks and determines the execution order using helper functions like `CheckNode` [5].
5. **Executing with Luigi** – After building the dependency list, the script is intended to launch Luigi tasks. The call to `luigi.build([EnqueueTask(filename, unique_id)], workers=1, local_scheduler=True, detailed_summary=True)` is currently commented out in `SparkMain.py` [10]. Un-commenting these lines will let Luigi orchestrate the tasks using the local scheduler. Luigi's documentation notes that running tasks from Python code via `luigi.build()` with `local_scheduler=True` is appropriate for development and can accept additional arguments such as `workers` [11].

The diagram below illustrates the example pipeline defined in `test3.json`. Two CSV sources are joined on a common key, then the result is sorted and partitioned:



Prerequisites

To run the project on your own machine you need:

1. **Python 3.7 or later** – The project uses standard Python modules, dataclasses and typing annotations.
2. **Java (JDK)** – PySpark requires a Java runtime. Install JDK 8 or later and ensure the `JAVA_HOME` environment variable points to your Java installation.
3. **PySpark** – Install PySpark using pip (the builder will download the appropriate Spark distribution):

   ```
   pip install pyspark
   ```

4. **Luigi** – Install Luigi for workflow orchestration:

   ```
   pip install luigi
   ```

5. **Optional dependencies** – If your JSON uses components that rely on additional libraries (for example, reading Parquet files or writing to a database), you may need to install extra packages. The provided examples only depend on PySpark and Luigi.

Setup and running the pipeline

Follow these steps to run the pipeline locally:

1. **Clone the repository**

```
git clone https://github.com/ltibfspoc/devinpoc.git
cd devinpoc
```

2. **Create and activate a virtual environment (optional but recommended)**

```
python -m venv venv
source venv/bin/activate  # On Windows use `venv\Scripts\activate`
```

3. **Install dependencies**

```
pip install pyspark luigi
```

4. **Adjust** `application.properties`

In `src/application.properties`, update `jsonfilepath` to point to the `src/config` directory within your cloned repository and choose which JSON configuration file to use:

```
[basicdetail]
jsonfilepath=/path/to/devinpoc/src/config
jsonfilename=test3.json
```

Replace `/path/to/devinpoc` with the absolute path on your system. The default values in the repository point to the developer's local Windows directories [1] and will not work elsewhere.

1. **Update JSON file paths**

The sample JSON files in `src/config` reference absolute Windows paths to the data (`C:\\Users\\Rishabh monster\\PycharmProjects\\pyspark\\src\\data\\…`) [1][2]. Edit the JSON file you intend to use so that `input_data_file_path` and `input_schema_file_path` point to the correct locations in your cloned repository (e.g., `src/data/factbook.csv`, `src/data/cars.csv`, `src/data/carSchema.txt`).

1. **(Optional) Re-enable Luigi execution**

In `src/etl/SparkMain.py` lines 56–71 the call to Luigi is commented out [10]. To actually run the pipeline, remove the triple quotes around this block so that `luigi.build([EnqueueTask(filename, unique_id)], workers=1, local_scheduler=True, detailed_summary=True)` executes. If you leave it commented, the script will only print the resolved dependency lists without executing tasks.

1. **Run the pipeline**

Execute the pipeline by passing the JSON filename (not the full path) as a command-line argument. The script will read `application.properties` to determine the full path:

```
python src/etl/SparkMain.py test3.json
```

The program will: - Read `application.properties` to construct the full JSON file path [4] . - Load the JSON configuration into component objects [7] . - Create a Spark session using `get_spark_session()` [6] . - Build the dependency graph and (once uncommented) run Luigi tasks in the proper order [5] .

1. **Monitor results**

2. When running with Luigi's local scheduler, logs will be printed to the console. You can also inspect the data written to the output location specified in your JSON configuration.

3. If you set `check_unfulfilled_deps=False` in `luigi.cfg` [8] , Luigi will not raise an error if a task's dependency is missing; ensure your dependencies are defined correctly.

Tips and troubleshooting

- **Spark cannot find Java** – Ensure `JAVA_HOME` is set and the Java runtime is on your `PATH`.
- **Path errors** – Absolute Windows paths in the JSON configuration will not work on other systems. Convert them to either absolute paths on your machine or relative paths within the repository.
- **Luigi not executing tasks** – Double-check that the call to `luigi.build()` in `SparkMain.py` is uncommented and that you are passing `local_scheduler=True`. Luigi's documentation notes that running tasks from Python code requires passing `local_scheduler=True` when using the local scheduler [11] .
- **Scaling beyond local mode** – The current implementation uses a local Spark master (`local[*]`). For cluster execution, modify `get_spark_session()` to specify the appropriate master URL and pass any additional Spark configuration.

With these adjustments, you should be able to run the `devinpoc` pipeline locally and explore how PySpark and Luigi can be combined to build modular data workflows.

---

1 raw.githubusercontent.com
https://raw.githubusercontent.com/ltibfspoc/devinpoc/main/src/application.properties

2 12 raw.githubusercontent.com
https://raw.githubusercontent.com/ltibfspoc/devinpoc/main/src/config/test3.json

3 4 9 10 raw.githubusercontent.com
https://raw.githubusercontent.com/ltibfspoc/devinpoc/main/src/etl/SparkMain.py

5 raw.githubusercontent.com
https://raw.githubusercontent.com/ltibfspoc/devinpoc/main/src/etl/SparkDriver.py

6 raw.githubusercontent.com
https://raw.githubusercontent.com/ltibfspoc/devinpoc/main/src/util/getSpark.py

7 raw.githubusercontent.com
https://raw.githubusercontent.com/ltibfspoc/devinpoc/main/src/util/read_json.py

8 devinpoc/src/luigi.cfg at main · ltibfspoc/devinpoc · GitHub
https://github.com/ltibfspoc/devinpoc/blob/main/src/luigi.cfg

11 Running Luigi — Luigi 3.6.0 documentation
https://luigi.readthedocs.io/en/stable/running_luigi.html