Flink Applications Powered by Python on a locally running Apache Flink Cluster in Docker

Discover how Apache Flink® can transform your data pipelines! Explore hands-on examples of Flink applications using the DataStream API, Table API, and Flink SQL—all built in Python with PyFlink, which compiles these apps to Java. You'll see how these technologies integrate seamlessly with AWS Services, Apache Kafka, and Apache Iceberg.

Curious about the differences between the DataStream API and Table API? Click here to learn more and find the best fit for your next project.

Table of Contents

1.0 Important Note(s)

> As previously mentioned, PyFlink does not have a built-in Python-based data generator source for creating data streams. Therefore, before executing any of the Flink applications, it is necessary to pre-populate the Kafka topics `airline.skyone` and `airline.sunset` with data. To achieve this, run the Java-based Flink application named `DataGeneratorApp` first, which is designed to generate sample records and populate these Kafka topics with the necessary data. This step ensures that subsequent Flink applications have data to work with and can properly perform their intended operations.

2.0 Power up the Apache Flink Docker containers

> **Prerequisite**
>
> Before you can run `./deploy-flink.sh` Bash script, you need to install the `aws2-wrap` utility. If you have a Mac machine, run this command from your Terminal:
>
> ```
> brew install aws2-wrap
> ```
>
> If you do not, make sure you have Python3.x installed on your machine, and run this command from your Terminal:
>
> ```
> pip install aws2-wrap
> ```

This section guides you through the local setup (on one machine but in separate containers) of the Apache Flink cluster in Session mode using Docker containers with support for Apache Iceberg. Run the `bash` script below to start the Apache Flink cluster in Session Mode on your machine:

```
./deploy-flink.sh <on | off> --profile=<AWS_SSO_PROFILE_NAME>
                              --chip=<amd64 | arm64>
                              --flink-language=python
```

Argument placeholder	Replace with
`<DOCKER_SWITCH>`	`on` to start up your very own local Apache Cluster running in Docker containers, otherwise `off` to stop the Docker containers.
`<AWS_SSO_PROFILE_NAME>`	your AWS SSO profile name for your AWS infrastructue that host your AWS Secrets Manager.
`<CHIP>`	if you are running on a Mac with M1, M2, or M3 chip, use `arm64`. Otherwise, use `amd64`.
`<FLINK_LANGUAGE>`	`python` to specify Python is the language base of the Flink Apps you plan on running. Otherwise, specifiy `java` if the language base of the Flink Apps are Java.

To learn more about this script, click here.

3.0 Discover What You Can Do with These Flink Apps

To access the Flink JobManager (`apache_flink-kickstarter-jobmanager-1`) container, open the interactive shell by running:

```
docker exec -it -u root -w /opt/flink/python_apps/src apache_flink-kickstarter-jobmanager-1 /bin/bash
```

Jump right into the container and take charge! You'll have full control to run commands, explore the file system, and tackle any tasks you need. You'll land directly in the `/opt/flink/python_apps/src` directory—this is the headquarters for all the Python scripts in the repo.

Ready to launch one of those cool Java-to-Python Flink apps? Just use the `flink run` command with the correct options, and kick off the Python Flink app script with its corresponding parameters below. Adventure awaits!

3.1 JSON formatted data

Flink App	Flink Run Command

Flink App	Flink Run Command
`json_flight_consolidator_app`	`uv run flink run --pyFiles kickstarter/python_files.zip --python kickstarter/json_flight_consolidator_app.py`
`json_flyer_stats_app`	`uv run flink run --pyFiles kickstarter/python_files.zip --python kickstarter/json_flyer_stats_app.py`

> | Argument placeholder | Replace with |
> |---|---|
> | `<AWS_S3_BUCKET>` | specify name of the AWS S3 bucket you chosen during the Terraform creation or created yourself separately. The AWS S3 bucket is used to store the Apache Iceberg files (i.e., data files, manifest files, manifest list file, and metadata files). |
> | `<AWS_REGION_NAME>` | specify the AWS Region your AWS Glue infrastructure resides. |

3.2 Did you notice I prepended `uv run` to `flink run`?

You maybe asking yourself why. Well, `uv` is an incredibly fast Python package installer and dependency resolver, written in **Rust**, and designed to seamlessly replace `pip`, `pipx`, `poetry`, `pyenv`, `twine`, `virtualenv`, and more in your workflows. By prefixing `uv run` to a command, you're ensuring that the command runs in an optimal Python environment.

Now, let's go a little deeper into the magic behind `uv run`:

- When you use it with a file ending in `.py` or an HTTP(S) URL, `uv` treats it as a script and runs it with a Python interpreter. In other words, `uv run file.py` is equivalent to `uv run python file.py`. If you're working with a URL, `uv` even downloads it temporarily to execute it. Any inline dependency metadata is installed into an isolated, temporary environment—meaning zero leftover mess! When used with `-`, the input will be read from `stdin`, and treated as a Python script.
- If used in a project directory, `uv` will automatically create or update the project environment before running the command.
- Outside of a project, if there's a virtual environment present in your current directory (or any parent directory), `uv` runs the command in that environment. If no environment is found, it uses the interpreter's environment.

So what does this mean when we put `uv run` before `flink run`? It means `uv` takes care of all the setup—fast and seamless—right in your local Docker container. If you think AI/ML is magic, the work the folks at Astral have done with `uv` is pure wizardry!

Curious to learn more about Astral's `uv`? Check these out:

- Documentation: Learn about `uv`.
- Video: `uv` IS THE FUTURE OF PYTHON PACKING!

4.0 Resources

Flink Python Docs

PyFlink API Reference

Apache Flink® Table API on Confluent Cloud - Examples

How to create a User-Defined Table Function (UDTF) in PyFlink to fetch data from an external source for your Flink App?

Apache Iceberg in Action with Apache Flink using Python