The Fundamentals of Type-safe, Reproducible, and Scalable Data Pipelines with Flyte

Niels Bantilan
Chief ML Engineer @ Union.ai
@cosmicbboy / cosmicbboy
April 19, 2023
Who am I?

- Chief ML Engineer @ Union.ai
- Open source creator: Pandera ✅, UnionML 🤖
- Core maintainer: Flyte ✈
- Background: Biology, Public Health 🕒
- Mission: build open source tools for DS/ML practitioners ⭐
My ML Journey

- 2012: pandas
- 2017: DASK
- 2022: W&B

I wrote pandera and UnionML.
Motivation
The Life of an ML Engineer
“My umpteenth ML project”
Insights

- ML pipelines are data pipelines
- Software is stateless, data is stateful
- If data shifts, models deteriorate
Why we built Flyte

i.e. the challenges of working with data
Challenge 1
The ecosystem of tools is constantly and rapidly evolving.

Credit: Sandeep Uttamchandani
Challenge 2

Developing datasets & models can be wasteful and inefficient.
**Challenge 3**

Data/ML infrastructure doesn’t scale well across teams/orgs

![Diagram]

1. Team A
   - Updated Critical Complex Pipeline
   - Pipeline W
   - Pipeline X

2. Team B
   - Pipeline Y
   - Pipeline Z
   - Breaks
Challenge 4

Complex ML workflows require a dedicated infrastructure team

- Provisioning CPU/GPU/Memory
- Framework/Library Independence
- Multi-tenancy
- Auto-scaling
- Efficiency: Caching, Model Checkpoints
- Cost Controls: Spot Machines
- Data Quality Assurance
- Model Monitoring
Challenge 5

ML pipelines require dynamism, i.e. the execution graph depends on the inputs.
Orchestration to the rescue! 🌟
Orchestrators coordinate the logical flow of computations needed to get data from its raw state into a desired state.
### What would I want out of an orchestrator?

<table>
<thead>
<tr>
<th>Challenges</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>🎈 Rapidly evolving ecosystem</td>
<td>A future-proof system that’s language- and framework- agnostic.</td>
</tr>
<tr>
<td>🗑 Data/model development inefficiency</td>
<td>Out-of-the-box support for data lineage tracking, caching, immutability, and versioning.</td>
</tr>
<tr>
<td>⛰ Poor scaling across teams/orgs</td>
<td>Isolated units of compute that can be arbitrarily composed together and reused across many different pipelines.</td>
</tr>
<tr>
<td>🔧 Need for infrastructure expertise</td>
<td>Declarative provisioning of compute, memory, disk requirements.</td>
</tr>
<tr>
<td>🔄 Dynamic execution graphs</td>
<td>Support for DAGs whose structure can be determined at runtime</td>
</tr>
</tbody>
</table>
How does Flyte address these challenges?
Flyte Overview

Create Tasks and Workflows

# workflows.py
from flytekit import task, workflow
...

Package & Register

Compiled Workflow

Container

Workflow Execution

Kubernetes Cluster

Flyte Cluster

Config
Tasks

The smallest unit of work in Flyte.

*addresses [👟,⛰️]

inputs

Task

outputs

Containerized

Versioned

Strongly Typed
Workflows

Compositions of Tasks to achieve complex computations

*addresses [🗑, ⛰]
Launch Plans

Customizing and scheduling the invocation behavior of workflows

*addresses [🗑, ⛰️]

Launch Plan

Workflow

inputs

outputs

Schedule: "* * * * *"

Default Args: {...}
Dynamic Workflows

Compositions of Tasks to achieve complex computations

*addresses [🔗]({"learning_rate": [0.1, 0.01, 0.001, 0.0001]})
Projects and Domains

Logical groupings of tasks and workflows for built-in multi-tenancy and isolation.

*addresses [▲]
What’s unique about Flyte?
**Type Safety**

Get errors about your execution graph at compile-time, even before executing your code.

*addresses [🗑, ⛅]*
Language-independence

Create Workflows in Python, Java, and Scala.

*addresses [운동, 퇴사, 산나라]
Declarative Infrastructure

Declaratively provisions ephemeral cluster, CPU/GPU, and memory resources.

*addresses [🔧]
Abstracted Data Persistence

Don’t worry about how data is serialized/deserialized as your execution graph runs

*addresses [🗑]
Building a Model Training Pipeline with Flyte
pip install flytekit
Training a penguin species classification model

https://allisonhorst.github.io/palmerpenguins/
The data:

https://allisonhorst.github.io/palmerpenguins/
Development Lifecycle
from typing import Tuple

import pandas as pd
from palmerpenguins import load_penguins
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

from flytekit import task, workflow, LaunchPlan, CronSchedule

TARGET = "species"
FEATURES = [
    "bill_length_mm",
    "bill_depth_mm",
    "flipper_length_mm",
    "body_mass_g",
]
Define the Task

Building Blocks 🧱
Compose a Workflow

```python
@workflow
def training_workflow(
    hyperparameters: dict,
    test_size: float = 0.2,
    random_state: int = 42,
) -> Tuple[LogisticRegression, float, float]:
    # get and split data
    data = get_data()
    train_data, test_data = split_data(
        data=data, test_size=test_size, random_state=random_state
    )
    # train model on the training set
    model = train_model(data=train_data, hyperparameters=hyperparameters)
    # evaluate the model
    train_acc = evaluate(model=model, data=train_data)
    test_acc = evaluate(model=model, data=test_data)
    # return model with results
    return model, train_acc, test_acc
```
Build a Container

`docker build . -t my_image:latest
docker push my_image:latest`

```dockerfile
# Docker containers unlock OS-level reproducibility

FROM python:3.9-slim-buster

WORKDIR /root
ENV VENV /opt/venv
ENV LANG C.UTF-8
ENV LC_ALL C.UTF-8
ENV PYTHONPATH /root
ENV GIT_PYTHON_REFRESH quiet

RUN apt-get update && apt-get install -y build-essential
RUN pip3 install awscli

ENV VENV /opt/venv

# Virtual environment
RUN python3 -m venv ${VENV}
ENV PATH=":${VENV}/bin:${PATH}"

# Install Python dependencies
COPY ./requirements.txt /root
RUN pip install -r /root/requirements.txt

# Copy the actual code
COPY . /root
RUN pip install -e /root
```
Package and Register to a Flyte Backend 📦

```
pyflyte run --remote workflows.example_00_intro training_workflow ...
pyflyte register workflows ...
```
Run on Flyte Console 🎥
Inspect your Workflow results on the UI 📊
Schedule your workflows with launch plans 🕒

```python
training_launchplan = LaunchPlan.create(
    "scheduled_training_workflow",
    training_workflow,

    # run every hour
    schedule=CronSchedule(schedule="@hourly"),

    # use default inputs
    default_inputs={"hyperparameters": {"C": 0.1, "max_iter": 1000}}
)
```
Visualize inputs and outputs with Flyte Decks 🖼️

```python
@task(requests=resources, limits=resources, disable_deck=False)
def get_data() -> pd.DataFrame:
    # Flyte Decks allow you to render html in the Flyte console so you can visualize and document metadata associated with a task.
    penguins = load_penguins()[(TARGET + FEATURES]
    Deck("data_profile", FrameProfilingRenderer("penguins"), to_html(penguins))
    return penguins
```
Create rich static reports with custom renderers 📊

class ConfusionMatrixRenderer:

def to_html(self, cm_display: ConfusionMatrixDisplay) -> str:
buff = BytesIO()
    cm_display.plot().fig.sframe().fig.savefig(buf, format="png")
    encoded = base64.b64encode(buf.getvalue()).decode("utf-8")
    return f"<img src='data:image/png;base64,{encoded}'/>"
Optimizing your Workflows
Cache intermediary results to avoid repeat computation 🧪

*addresses [🗑️]
Use `map_task` to parallelize tasks

```
@workflow
def tune_model(
    hyperparam_grid: List[dict],
    tune_data: StructuredDataset,
    val_size: float,
    random_state: int,
) -> Tuple[LogisticRegression, float]:
    """And finally, a workflow that performs grid search."""
    train_data, val_data = split_data(
        data=tune_data, test_size=val_size, random_state=random_state
    )
    # wrapping the 'train_model' task in `map_task` allows us to parallelize
    # our grid search.
    models = map_task(train_model, concurrency=5)(
        train_args=prepare_train_args(
            train_data=train_data, hyperparam_grid=hyperparam_grid
        )
    )
    return get_best_model(models=models, val_data=val_data)
```
Provision ephemeral distributed computing frameworks 🏆

*addresses 🔧

*also supports Ray, Dask
Provision the GPUs, CPUs, and memory needed for your ML workloads.

*addresses [🔗]

*also supports Kubeflow Tensorflow, MPI, Horovod*
How are People Using Flyte?
<table>
<thead>
<tr>
<th>Use Case</th>
<th>Flyte’s Impact</th>
</tr>
</thead>
</table>
| **Spotify**                  | Use Case: End-to-end profit & loss forecasting  
Flyte’s Impact:  
- Increases velocity to deliver models  
- Breaks down silos between teams |
| **freenome**                 | Use Case: Early detection of cancer via ML  
Flyte’s Impact:  
- Powers full data and ML stack  
- Accelerates clinical research |
| **blackshark.ai**            | Use Case: Digital 3D Mapping of the World  
Flyte’s Impact:  
- Enables processing ~2.5 petabytes of data  
- Unlocks multi-cloud provider capabilities |
| **gojek**                    | Use Case: Price optimization, rider matching, etc.  
Flyte’s Impact:  
- Easily roll back critical pipeline bugs  
- Unlocks scale and reliability of pipelines |
Summary

✈ Flyte orchestrates compute, data, and infrastructure

🚨 Type safety means you can catch compile-time bugs early

📍 Container-native tasks ensures reproducibility

🏔 Scales your production workflows seamlessly

🤖 Supports the canonical data science and ML tech stack

🛠 Easily customizable and extendable

🤝 Breaks the data and model silos between teams
Getting Started with Flyte

📖 Follow our Getting Started Guide:

👋 Join us on Slack:
https://slack.flyte.org/

💻 See the Code:
https://github.com/flyteorg/flyte
Union Cloud

Frictionless Machine Learning to Production

Union Cloud is a managed version of Flyte™. It frees data and ML teams from infrastructure constraints and setup so they can focus on ML production.

Try Union Cloud
We’re Hiring!

https://www.union.ai/careers
Take Flyte

pip install flytekit
pyflyte init my_app

https://github.com/flyteorg/flyte

@cosmicbboy / cosmicbboy 🌞