Distributed Data Systems

MAHESH CHAUDHARI, PH.D

Contents

Course Overview

Workflow Management

Apache Airflow

Airflow DAG Creation

Contents

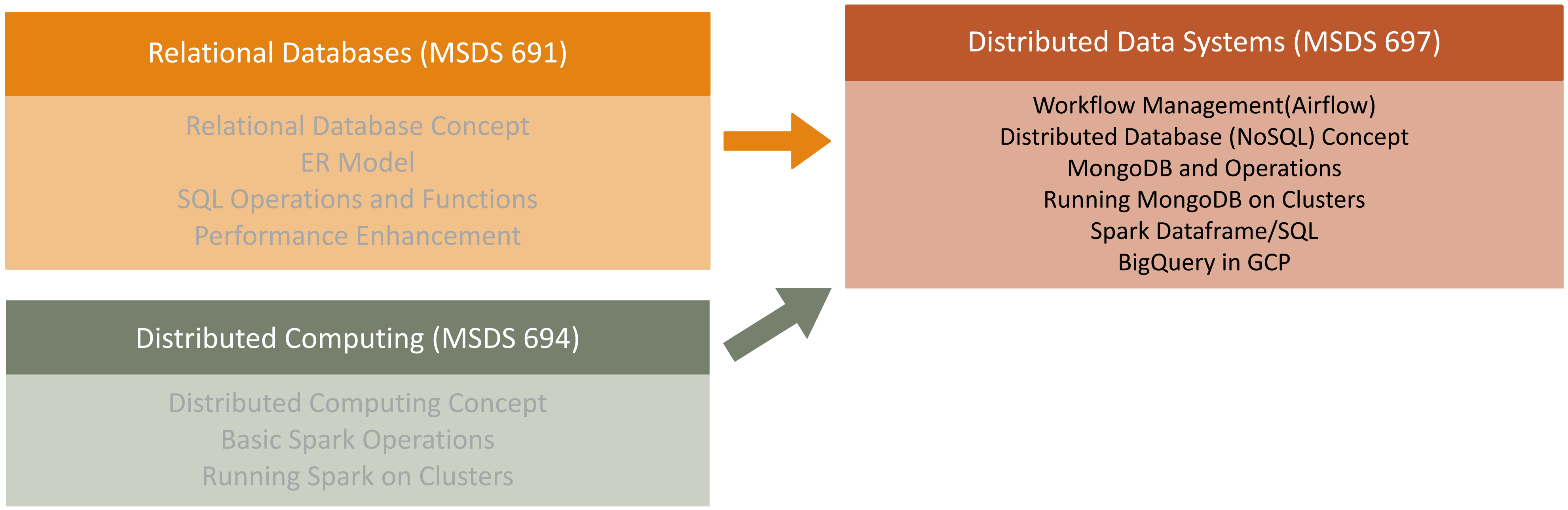
Course Overview

Workflow Management

Apache Airflow

Airflow DAG Creation

About Distributed Data Systems



Relational Databases (MSDS 691)

Relational Database Concept
ER Model
SQL Operations and Functions
Performance Enhancement

Distributed Computing (MSDS 694)

Distributed Computing Concept
Basic Spark Operations
Running Spark on Clusters

Distributed Data Systems (MSDS 697)

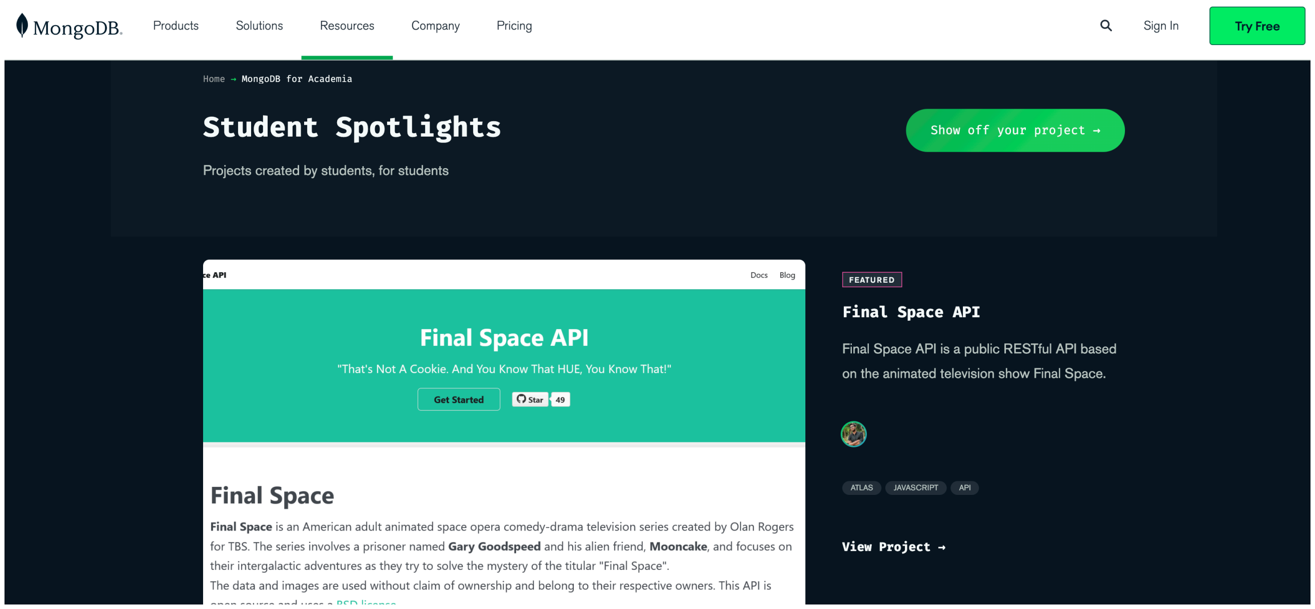
Workflow Management(Airflow)
Distributed Database (NoSQL) Concept
MongoDB and Operations
Running MongoDB on Clusters
Spark Dataframe/SQL
BigQuery in GCP

Course Learning Objectives

1. Understand the needs and concepts of distributed systems and build an automated and scalable ETL(Extract, Transform and Load) pipeline.
2. Distributed Database Management Systems
   - Understand the needs and characteristics of schemaless non-relational databases for storing and querying data in distributed settings.
   - Gain insights to make judgments to choose SQL or NoSQL (and which NoSQL) with various hands-on exercises.
   - Gain experience with NoSQL databases including MongoDB through homework and in-class exercises.
   - Be competent to work with Spark and MongoDB in a distributed environment.
3. Distributed Computing
   - Gain experience with SparkSQL and understand its relationship with RDD.
   - Be competent to work with Spark in a distributed computing environment including Google Cloud Platform.
   - Use BigQuery as a distributed data warehouse to understand the workings of BigQuery, understand native and external tables, load data and process data in BigQuery using SQL.

Possible Outcomes

Partnering with MongoDB, selected teams' project will be highlighted at MongoDB Student Spotlight



Student Spotlights

Projects created by students, for students

Final Space API

"That's Not A Cookie. And You Know That HUE, You Know That!"

Final Space

Final Space is an American adult animated space opera comedy-drama television series created by Olan Rogers for TBS. The series involves a prisoner named **Gary Goodspeed** and his alien friend, **Mooncake**, and focuses on their intergalactic adventures as they try to solve the mystery of the titular "Final Space".

The data and images are used without claim of ownership and belong to their respective owners. This API is

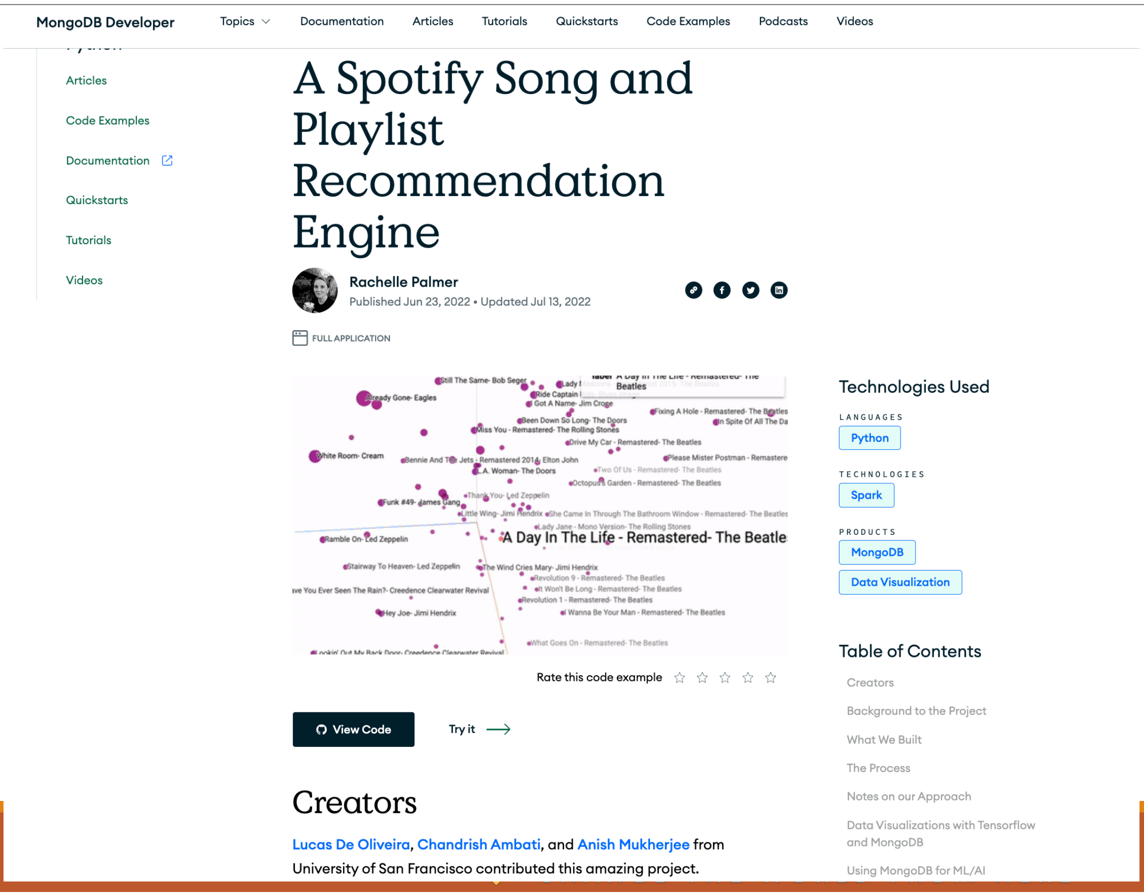
Final Space API

Final Space API is a public RESTful API based on the animated television show Final Space.

https://www.mongodb.com/developer/academia/students/

Possible Outcomes

2022



MongoDB Developer Topics Documentation Articles Tutorials Quickstarts Code Examples Podcasts Videos

- Articles
- Code Examples
- Documentation
- Quickstarts
- Tutorials
- Videos

A Spotify Song and Playlist Recommendation Engine

Rachelle Palmer
Published Jun 23, 2022 • Updated Jul 13, 2022

FULL APPLICATION

Still The Same- Bob Seger
Already Gone- Eagles
Got A Name- Jim Croce
Been Down So Long- The Doors
Miss You - Remastered- The Rolling Stones
Fixing A Hole - Remastered- The Beatles
White Room- Cream
Bennie And The Jets - Remastered 2014- Elton John
L.A. Woman- The Doors
Please Mister Postman - Remastered
Octopus's Garden - Remastered- The Beatles
Funk #49- James Gang
Thank You- Led Zeppelin
Little Wing- Jimi Hendrix
She Came In Through The Bathroom Window - Remastered- The Beatles
A Day In The Life - Remastered- The Beatle
Ramble On- Led Zeppelin
Stairway To Heaven- Led Zeppelin
The Wind Cries Mary- Jimi Hendrix
Revolution 9 - Remastered- The Beatles
Revolution 1 - Remastered- The Beatles
I Wanna Be Your Man - Remastered- The Beatles
Hey Joe- Jimi Hendrix
What Goes On - Remastered- The Beatles

Rate this code example ☆ ☆ ☆ ☆ ☆

View Code Try it

Technologies Used

LANGUAGES
Python

TECHNOLOGIES
Spark

PRODUCTS
MongoDB
Data Visualization

Table of Contents

Creators
Background to the Project
What We Built
The Process
Notes on our Approach
Data Visualizations with Tensorflow and MongoDB
Using MongoDB for ML/AI

Creators

Lucas De Oliveira, Chandrish Ambati, and Anish Mukherjee from University of San Francisco contributed this amazing project.

https://www.mongodb.com/developer/code-examples/python/song-recommendations-example-app/

Vocalize your work - Research Papers

· **Bhumika Srinivas**, Eren Bardak, Ireri Avila, Jessica Brungard, Yihan Cao and Mahesh B. Chaudhari, Streamlining Social Music Analytics with Distributed Data Processing: Apache Airflow and MongoDB and Spark SQL, 2025 **International Conference on Computing, Intelligence, and Application** (CIACON 2025), Durgapur, India, July 2025, https://doi.org/10.1109/CIACON65473.2025.11189649.

· **Irene Garcia Montoya**, Kabir Nawani, Fred Serfati, Gaurav Goyal, Sai Pranavi Avadhanam, and Mahesh B. Chaudhari, Incorporating Building Data Pipelines in Data-Engineering Curriculum for Supporting Classification Models in Production, **Journal of Computing Sciences in Colleges**, 40, 10 (July 2025), 49-57 (Proceedings of Consortium for Computing Sciences in Colleges, MidSouth Conference, Clarksville, AR, April 2025), https://dl.acm.org/doi/abs/10.5555/3744992.3745001.

· **Ranjeet Nagarkar**, Colin Bennie, Kejia Wang, Mark Lam, Daniel Gonzalez and Mahesh B. Chaudhari, Integrating Multiple Cloud Platforms to Build a Data Pipeline for Recommendation Systems, 7th **International Conference on Data Science and Information Technology** (DSIT 2024), Nanjing, China, December 2024, pp. 326-330. http://doi.org/10.1109/DSIT61374.2024.10881634.

ILivia Ellen, Yi Zhuang, Ajay Kallepalli, Nirant Khot and Jahnavi Paliwal, Mahesh Chaudhari, Optimizing Data Science Workflows through Scalable Data Engineering Pipelines: A Case Study on Michelin Star Analysis

- Accepted at the 11th **International Conference on Big Data Analytics**, ICBDA 2026, Waseda University, Tokyo, Japan, April 2026

Course Evaluation

Individual Assignment - 20% (MongoDB (1, 2) , PySpark + SQL (3))

Group Project - 30 %

- Phase-1: Data Selection/Loading on Google Cloud Storage and Cloud Composer (Airflow).
- Phase-2: Data Preprocessing and Store (Spark & MongoDB).
- Phase-3: Adding ML/AI, Final Report and Code Submission.

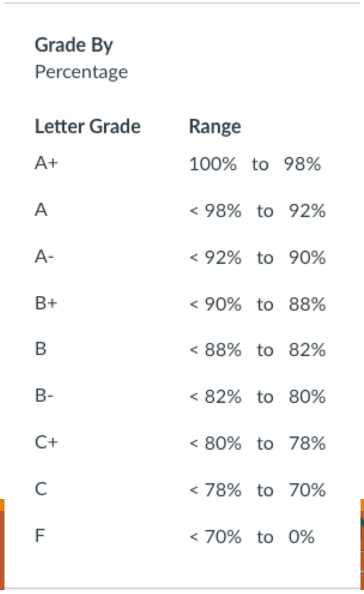
Quiz-1 - 20 %

Final Exam - 30 %

Course Evaluation

This class is a standard, graded course with letter grades A - F.

- I consider an A grade to be above and beyond what most students have achieved.
- A B grade is an average grade for a student or what you could call "competence" in a business setting.
- A C grade means that you either did not or could not put forth the effort to achieve competence.
- Below C (F) implies you did very little work or had great difficulty with the class compared to other students.



Grade By
Percentage

Letter Grade	Range
A+	100% to 98%
A	< 98% to 92%
A-	< 92% to 90%
B+	< 90% to 88%
B	< 88% to 82%
B-	< 82% to 80%
C+	< 80% to 78%
C	< 78% to 70%
F	< 70% to 0%

Course Evaluation

All assignments should be submitted on time via Canvas.

- No late submission is allowed.
- No submissions via Slack is allowed.

In order to pass this course, students are expected to receive at least 50% of each category. (Ex. receiving less than 29/60 on your quiz might not be a satisfying grade.)

Questions/discussions regarding assignments and course topics should be posted on Piazza (rather than slack or email) and visible to the instructor and other classmates.

Quizzes will be monitored and recorded by the instructor and proctoring system.

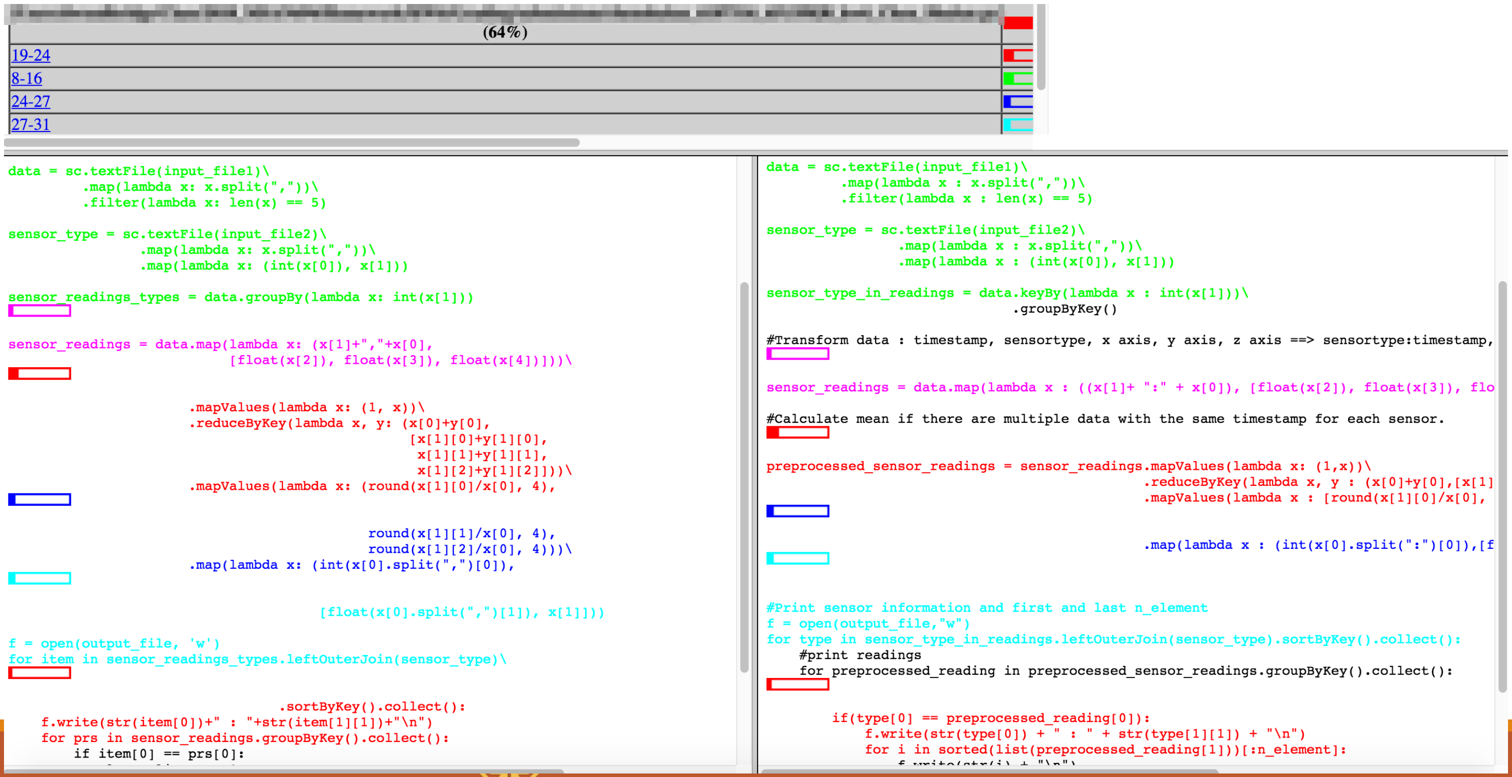
Plagiarized works will receive at least 0 for the assignment or the assignment category. If plagiarism happens, your professionalism score will be 0. Also, further action will be taken based on the program policy (https://myusf.usfca.edu/arts-sciences/data-science/program-policies).

- Individual assignments should be completed independently based on the university's academic integrity policy.
  - If you have any questions or need to discuss, please post on Pizza or come to the office hour.
- For programming assignments, plagiarism will be checked by Stanford MOSS (https://theory.stanford.edu/~aiken/moss/).
  - Codes with similarity scores higher than 40% would be reviewed.

Course Evaluation

Changed lines and tweaked logics

Please no plagiarism! – Zero tolerance and will received 0.



[redacted]

(64%)
19-24
8-16
24-27
27-31

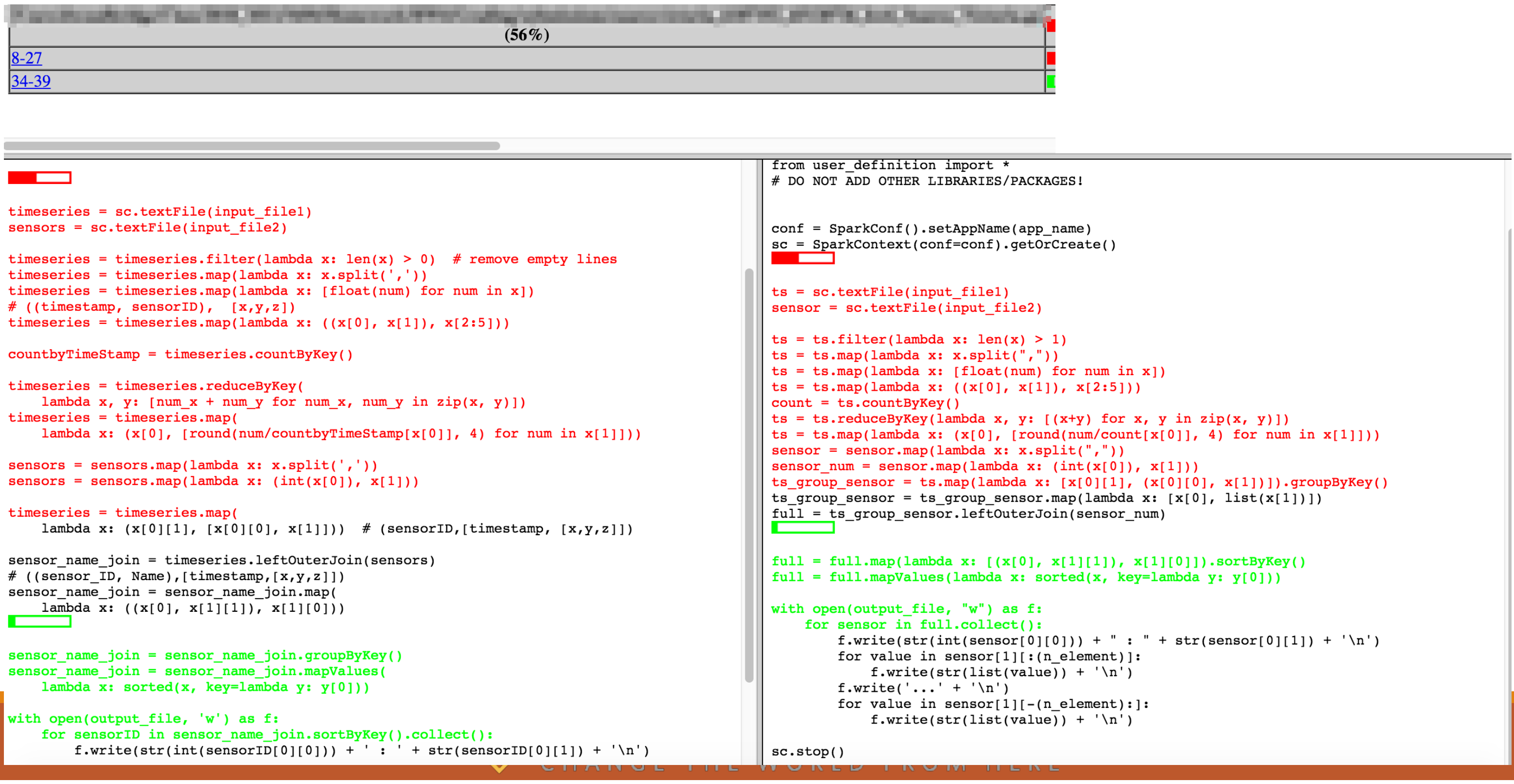
```
data = sc.textFile(input_file1)\
         .map(lambda x: x.split(","))\
         .filter(lambda x: len(x) == 5)

sensor_type = sc.textFile(input_file2)\
                .map(lambda x: x.split(","))\
                .map(lambda x: (int(x[0]), x[1]))

sensor_readings_types = data.groupBy(lambda x: int(x[1]))

sensor_readings = data.map(lambda x: (x[1]+","+x[0],
                           [float(x[2]), float(x[3]), float(x[4])]))\

                      .mapValues(lambda x: (1, x))\
                      .reduceByKey(lambda x, y: (x[0]+y[0],
                                                 [x[1][0]+y[1][0],
                                                  x[1][1]+y[1][1],
                                                  x[1][2]+y[1][2]]))\
                      .mapValues(lambda x: (round(x[1][0]/x[0], 4),

                                            round(x[1][1]/x[0], 4),
                                            round(x[1][2]/x[0], 4)))\
                      .map(lambda x: (int(x[0].split(",")[0]),

                                      [float(x[0].split(",")[1]), x[1]]))

f = open(output_file, 'w')
for item in sensor_readings_types.leftOuterJoin(sensor_type)\

                                   .sortByKey().collect():
    f.write(str(item[0])+" : "+str(item[1][1])+"\n")
    for prs in sensor_readings.groupByKey().collect():
        if item[0] == prs[0]:
```

```
data = sc.textFile(input_file1)\
         .map(lambda x : x.split(","))\
         .filter(lambda x : len(x) == 5)

sensor_type = sc.textFile(input_file2)\
                .map(lambda x : x.split(","))\
                .map(lambda x : (int(x[0]), x[1]))

sensor_type_in_readings = data.keyBy(lambda x : int(x[1]))\
                               .groupByKey()

#Transform data : timestamp, sensortype, x axis, y axis, z axis ==> sensortype:timestamp,

sensor_readings = data.map(lambda x : ((x[1]+ ":" + x[0]), [float(x[2]), float(x[3]), flo

#Calculate mean if there are multiple data with the same timestamp for each sensor.

preprocessed_sensor_readings = sensor_readings.mapValues(lambda x: (1,x))\
                                              .reduceByKey(lambda x, y : (x[0]+y[0],[x[1]
                                              .mapValues(lambda x : [round(x[1][0]/x[0],

                                              .map(lambda x : (int(x[0].split(":")[0]),[f


#Print sensor information and first and last n_element
f = open(output_file,"w")
for type in sensor_type_in_readings.leftOuterJoin(sensor_type).sortByKey().collect():
    #print readings
    for preprocessed_reading in preprocessed_sensor_readings.groupByKey().collect():

        if(type[0] == preprocessed_reading[0]):
            f.write(str(type[0]) + " : " + str(type[1][1]) + "\n")
            for i in sorted(list(preprocessed_reading[1]))[:n_element]:
```

Course Evaluation

Changed lines and variable names

Please no plagiarism! – Zero tolerance and will received 0.



[redacted]

(56%)
8-27
34-39

```
timeseries = sc.textFile(input_file1)
sensors = sc.textFile(input_file2)

timeseries = timeseries.filter(lambda x: len(x) > 0)  # remove empty lines
timeseries = timeseries.map(lambda x: x.split(','))
timeseries = timeseries.map(lambda x: [float(num) for num in x])
# ((timestamp, sensorID),  [x,y,z])
timeseries = timeseries.map(lambda x: ((x[0], x[1]), x[2:5]))

countbyTimeStamp = timeseries.countByKey()

timeseries = timeseries.reduceByKey(
    lambda x, y: [num_x + num_y for num_x, num_y in zip(x, y)])
timeseries = timeseries.map(
    lambda x: (x[0], [round(num/countbyTimeStamp[x[0]], 4) for num in x[1]]))

sensors = sensors.map(lambda x: x.split(','))
sensors = sensors.map(lambda x: (int(x[0]), x[1]))

timeseries = timeseries.map(
    lambda x: (x[0][1], [x[0][0], x[1]]))  # (sensorID,[timestamp, [x,y,z]])

sensor_name_join = timeseries.leftOuterJoin(sensors)
# ((sensor_ID, Name),[timestamp,[x,y,z]])
sensor_name_join = sensor_name_join.map(
    lambda x: ((x[0], x[1][1]), x[1][0]))

sensor_name_join = sensor_name_join.groupByKey()
sensor_name_join = sensor_name_join.mapValues(
    lambda x: sorted(x, key=lambda y: y[0]))

with open(output_file, 'w') as f:
    for sensorID in sensor_name_join.sortByKey().collect():
        f.write(str(int(sensorID[0][0])) + ' : ' + str(sensorID[0][1]) + '\n')
```

```
from user_definition import *
# DO NOT ADD OTHER LIBRARIES/PACKAGES!


conf = SparkConf().setAppName(app_name)
sc = SparkContext(conf=conf).getOrCreate()


ts = sc.textFile(input_file1)
sensor = sc.textFile(input_file2)

ts = ts.filter(lambda x: len(x) > 1)
ts = ts.map(lambda x: x.split(","))
ts = ts.map(lambda x: [float(num) for num in x])
ts = ts.map(lambda x: ((x[0], x[1]), x[2:5]))
count = ts.countByKey()
ts = ts.reduceByKey(lambda x, y: [(x+y) for x, y in zip(x, y)])
ts = ts.map(lambda x: (x[0], [round(num/count[x[0]], 4) for num in x[1]]))
sensor = sensor.map(lambda x: x.split(","))
sensor_num = sensor.map(lambda x: (int(x[0]), x[1]))
ts_group_sensor = ts.map(lambda x: [x[0][1], (x[0][0], x[1])]).groupByKey()
ts_group_sensor = ts_group_sensor.map(lambda x: [x[0], list(x[1])])
full = ts_group_sensor.leftOuterJoin(sensor_num)


full = full.map(lambda x: [(x[0], x[1][1]), x[1][0]]).sortByKey()
full = full.mapValues(lambda x: sorted(x, key=lambda y: y[0]))

with open(output_file, "w") as f:
    for sensor in full.collect():
        f.write(str(int(sensor[0][0])) + " : " + str(sensor[0][1]) + '\n')
        for value in sensor[1][:(n_element)]:
            f.write(str(list(value)) + '\n')
        f.write('...' + '\n')
        for value in sensor[1][-(n_element):]:
            f.write(str(list(value)) + '\n')

sc.stop()
```

Course Schedule

Session 1 - M/Th : 10:00 - 11:50, Room 527

Session 2 - M/Th : 1:00 - 2:50, Room 527

Office Hours: M/Th: 3:00 - 4:00 #524

Grader: Eunice Tu

Office Hours: TBD

Course Schedule

LEARNING OBJECTIVES

- Week 1 - Course Intro, Workflow and Apache Airflow
- Week 2 - NoSQL Concept, MongoDB - Datatypes, CRUD
- Week 3 - MongoDB - Aggregation Pipeline, Index
- Week 4 - MongoDB - Replication and Sharding, CAP Theorem, MongoDB Atlas and Lab
- Week 5 - Apache Spark Recap, Data Frame Creation, Functions, Load Data Frame using SparkSQL, Register DataFrame
- Week 6 - Advanced SQL with Spark SQL + BigQuery in GCP
- Week 7 - Final Project

Course Schedule

- Individual Assignment

Date	Assignment
2025-02-09	HW1
2025-02-23	HW2
2025-03-07	HW3

- Group Assignment

Date	Assignment
2025-02-07	Task 1 - Finalize your storyline
2025-02-21	Task 2 - Airflow, MongoDB, SparkSQL
2025-03-14	Task 3 : +ML/AI, Report, Final Project Submission

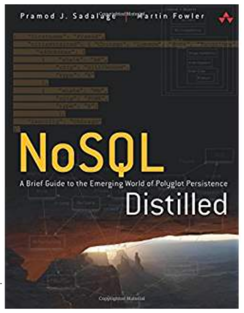
- Exams

Date	Assignment
2025-02-20	Quiz-1
2025-03-12	Final Exam

Attendance Policy

- Please make sure to attend the session that you are assigned to. If you want to switch a session, please email me at least 24 hours in advance including reasons (ex. Doctor’s note).
- To enhance student learning, I might publish a short recorded video. Make sure to watch a recorded session, if I assign them.
- No cellphones, social media, slack, texting during the class .
- Please only use laptops for class-related purposes.
- Follow the USF Covid-19 Policy.

Reference Materials



Apache Airflow Online Documentation, https://airflow.apache.org/docs/apache-airflow/

Spark Online Documentation, http://spark.apache.org/docs/latest/

MongoDB. Online Documentation, https://docs.mongodb.com/

Google Cloud Platform Online Documentation, https://cloud.google.com/docs

Shannon Bradshaw, Eoin Brazil, Kristina Chodorow. MongoDB: The Definitive Guide. O'Reilly Media, Inc. 2019.

Jean-Georges Perrin. Spark in Action, O'Reilly Publications, 2020.

Sadalage, Pramod J., and Martin Fowler. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education, 2012.

Student Engagement

Poll : https://pollev.com/pollmsds

Canvas

- Under each module, there are learning outcome, slides, homework, example tests, tests, etc.
- Please use Piazza for discussions/questions.

Contents

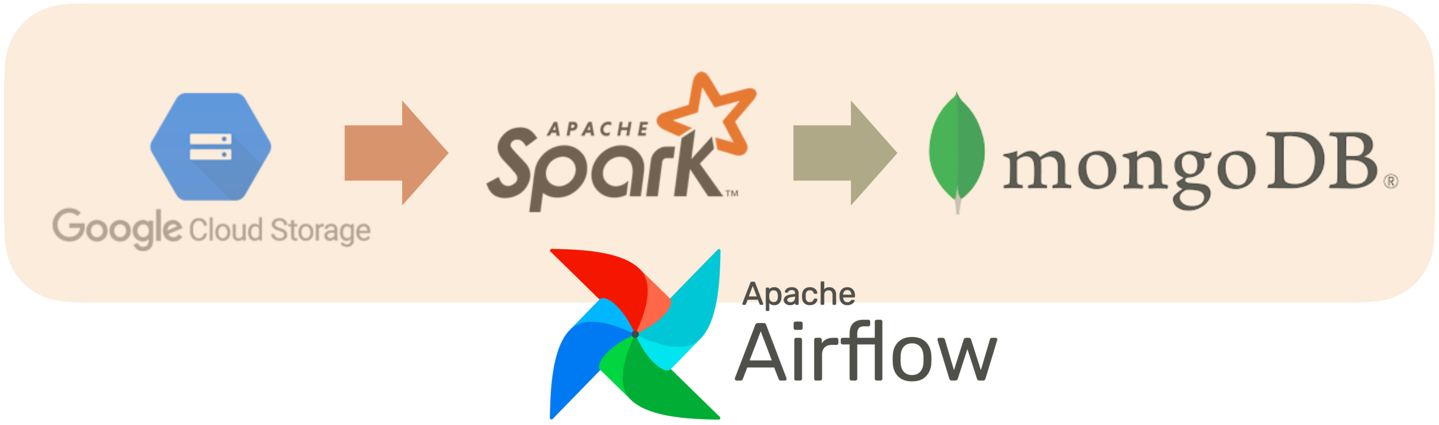
Course Overview

Workflow Management

Apache Airflow

Airflow DAG Creation

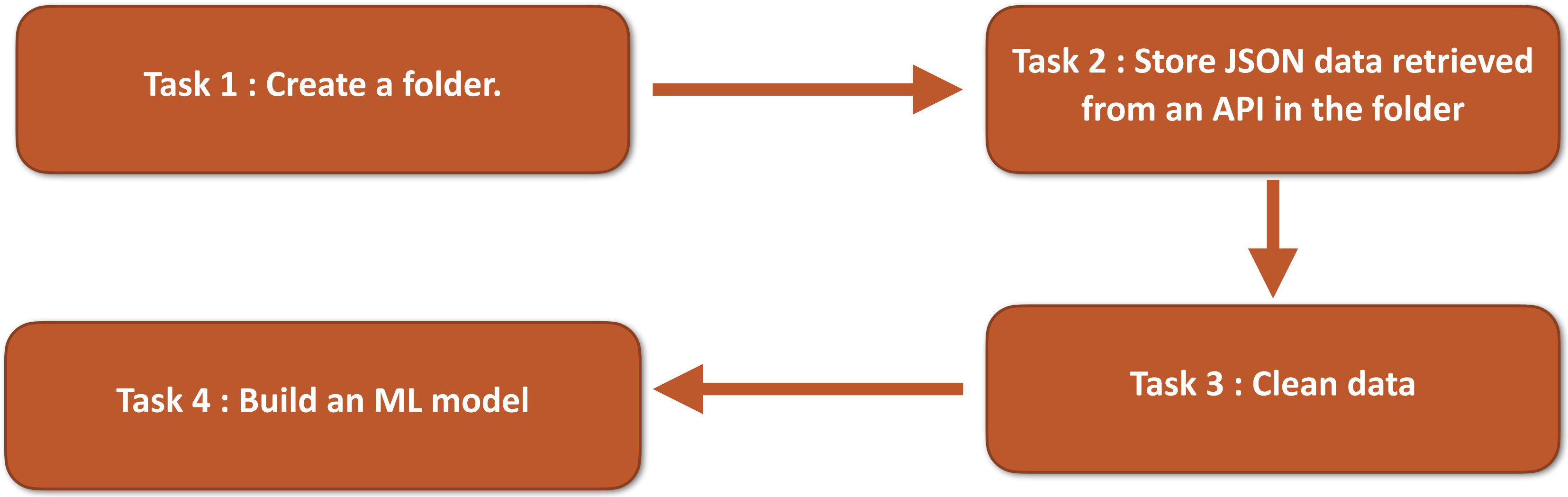
Building an Automated Scalable ETL Pipeline in MSDS-697 - Group Project



Data Pipeline

A set of data processing tasks, where the output of one task is the input of the next one or dependent on it.

- Can be represented as an **directed acyclic graph (DAG)**


Task 1 : Create a folder.
Task 2 : Store JSON data retrieved from an API in the folder
Task 3 : Clean data
Task 4 : Build an ML model

Workflow Manager

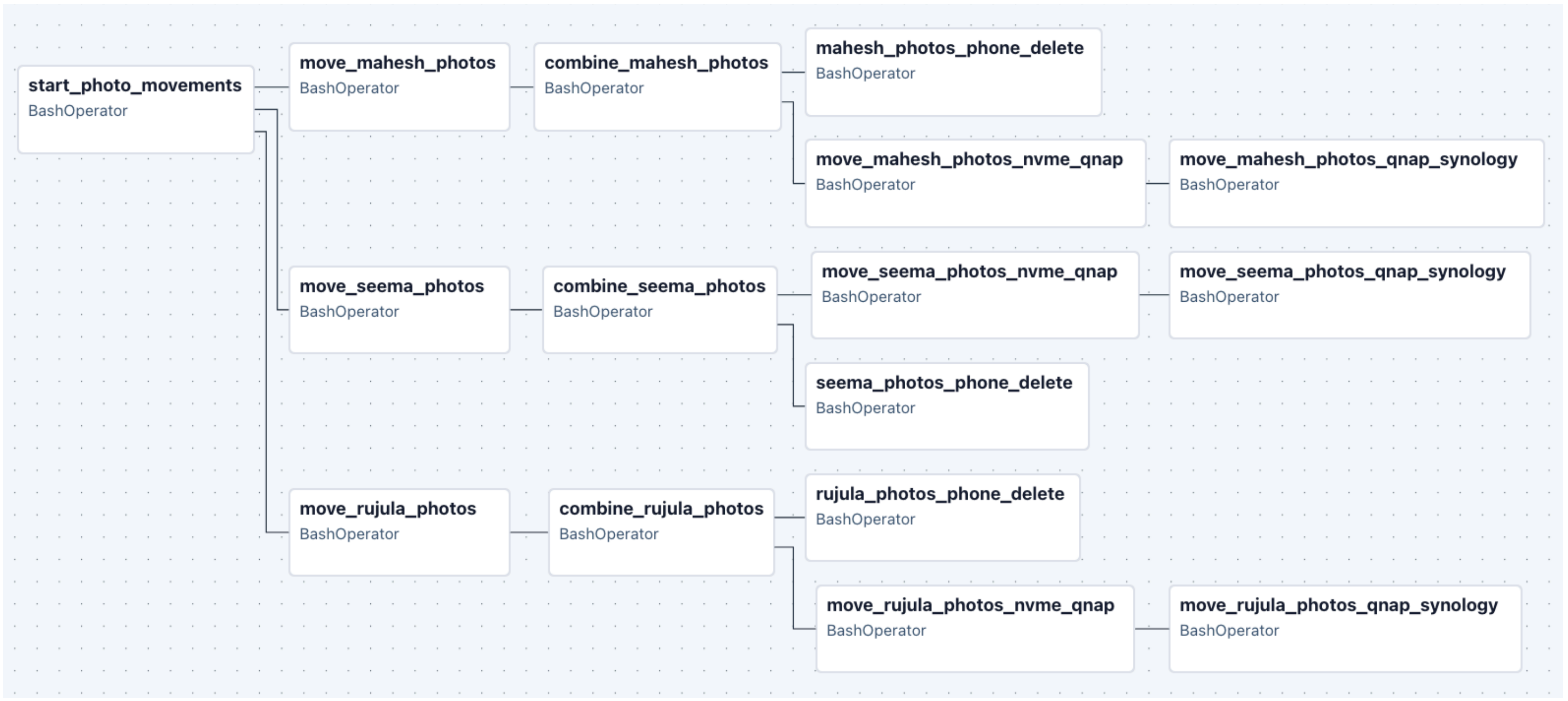
Allow to define graphs of tasks as pipelines.

Automate the tasks in pipelines on given conditions and schedules.

- Ex. Every Monday at 9 am, retrieve the practicum weekly report from Canvas ➡ If submitted, create an email and send to the corresponding mentors.

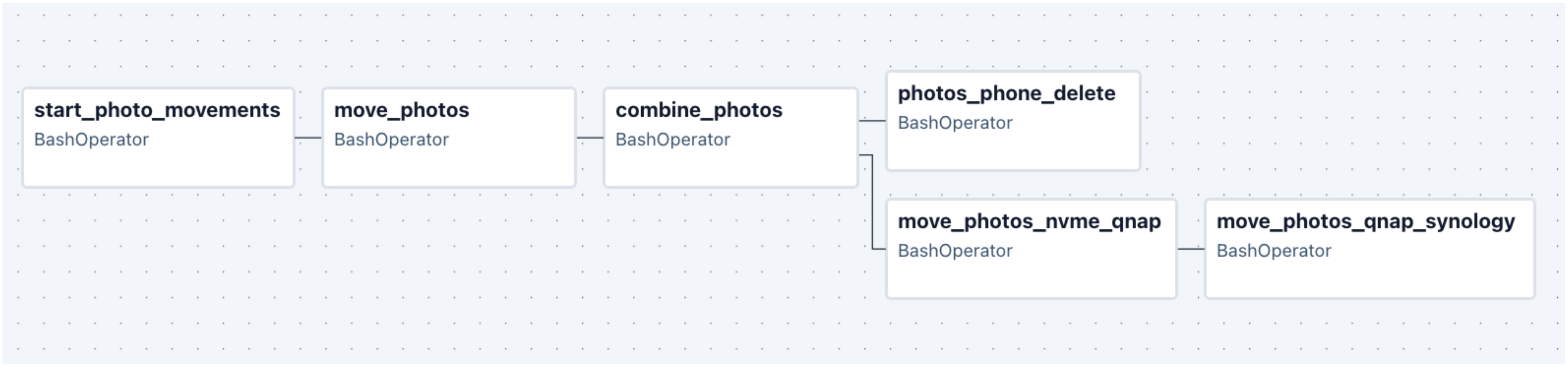
https://github.com/pditommaso/awesome-pipeline

Example


start_photo_movements
BashOperator
move_mahesh_photos
BashOperator
combine_mahesh_photos
BashOperator
mahesh_photos_phone_delete
BashOperator
move_mahesh_photos_nvme_qnap
BashOperator
move_mahesh_photos_qnap_synology
BashOperator
move_seema_photos
BashOperator
combine_seema_photos
BashOperator
move_seema_photos_nvme_qnap
BashOperator
move_seema_photos_qnap_synology
BashOperator
seema_photos_phone_delete
BashOperator
move_rujula_photos
BashOperator
combine_rujula_photos
BashOperator
rujula_photos_phone_delete
BashOperator
move_rujula_photos_nvme_qnap
BashOperator
move_rujula_photos_qnap_synology
BashOperator

Simplified Example

Parameterized and reusable



Future: Trigger based, file detection

Contents

Course Overview

Workflow Management

Apache Airflow

Airflow DAG Creation

Airflow

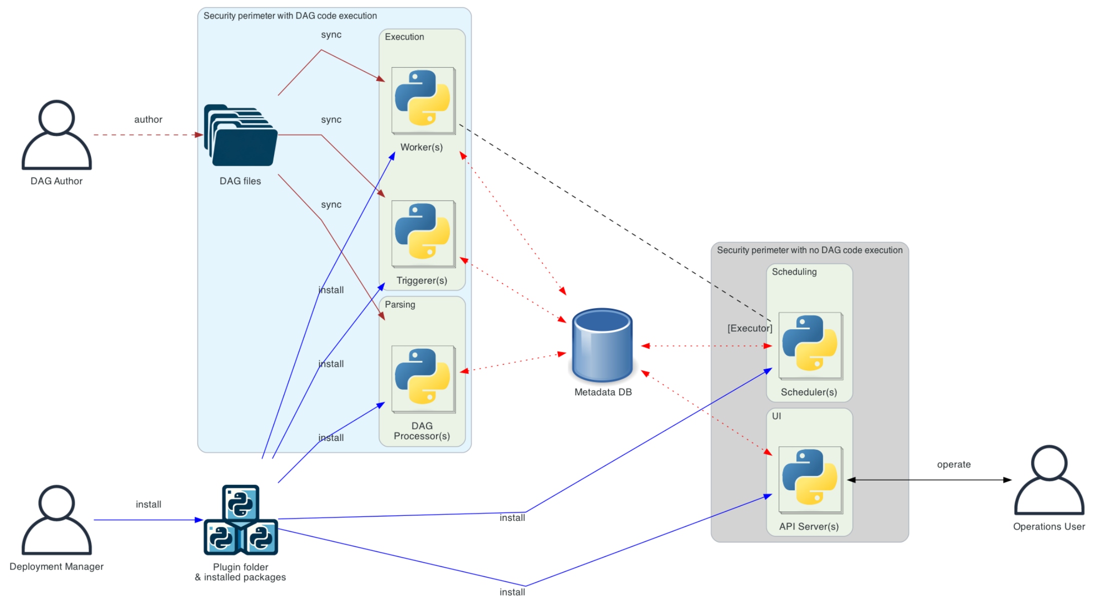
An open-source scalable workflow manager for developing and monitoring batch-oriented workflows (developed by Airbnb).

- You can define DAGs using Python.
  - Should include a set of tasks and their dependencies.
  - Flexible - you can dynamically generate optional tasks based on conditions.
- Main Components
  - Scheduler - monitors all tasks and DAGs, triggers the task instances once their dependencies are complete.
  - Workers - Executing tasks.
  - api-server - Visualize DAGs and provide main interface to monitor DAGs.
  - dag-processor - dedicated component responsible for continuously parsing Python files that define workflows (DAGs) and storing a serialized version of them in the metadata database

https://airflow.apache.org

Airflow

https://airflow.apache.org



1. User writes DAG
2. dag-processor detects the new DAG, parses it and serializes it into the airflow database.
3. Scheduler parses DAG and schedules tasks, considering schedules and dependencies.
4. Worker executes scheduled tasks.
5. User monitors workflow execution schedules and results displayed by the api-server.

* Note : Results/logs from each step is stored in metastores to track schedules and progress.

Airflow Installation

- Use Conda environment with python=3.12 Airflow with latest Python has some issues.
- Create conda environment with python version 3.12

```
conda create --name dds-spring-2026 python=3.12
```

- Activate the environment

```
conda activate dds-spring-2026
```

- Install Airflow on your local machine:

```
conda install conda-forge::airflow
```

- Initiate/Migrate the existing SQLLite database

```
airflow db migrate
```

Executing DAG via Airflow

Step 1. Define DAG, schedules, configurations, its tasks belonging to it and their dependencies.

Step 2. Add the DAG in your airflow/dags folder.

Step 3. Run Airflow scheduler and api-server and dag-processor.

Open 3 separate terminal windows with conda env activated:

- airflow scheduler
- airflow api-server
- airflow dag-processor

Step 4. Access Airflow UI at http://localhost:8080

Contents

Course Overview

Workflow Management

Apache Airflow

Airflow DAG Creation

Airflow DAG definition

DAG

- Includes tasks and their dependencies.
- Declaration

```
with DAG(dag_id="dag_name",
         start_date=datetime(2022, 1, 1),
         end_date=datetime(2023, 12, 31),
         schedule="@daily") as dag:

    task1 = PythonOperator(task_id="task1",
                           python_callable=func_1,
                           op_kwargs={'arg1':val1, 'arg2':val2}
                           )

    task2 = BashOperator(task_id="task2",
                         bash_command="enter bash command")

    task1 >> task2
```

Str, ID of DAG

Datetime, timestamp from which the scheduler will attempt to run/backfill

Datetime, timestamp beyond which your DAG won't run, leave to None for open-ended.

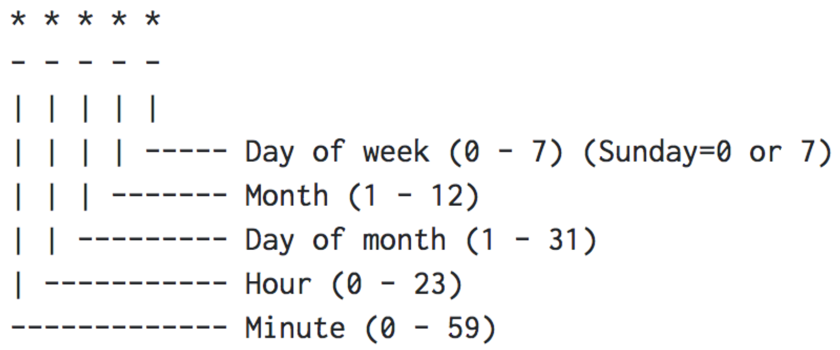
Scheduling rules. You can use `@once`,`@hourly`, `@daily`, `@weekly`, `@monthly`, `@yearly` and also crontab-like scheduling

https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/dags.html
https://airflow.apache.org/docs/apache-airflow/stable/_api/airflow/models/dag/index.html#airflow.models.dag.DAG

Airflow DAG definition

Crontab

- Format
  - 1st : Minute (0-59)
  - 2nd: Hour (0-23)
  - 3rd : Day (1-31)
  - 4th : Month(1-12)
  - 5th : Day of week (0-7. 0/7-Sun. 1-Mon, 2-Tue..)
- Value : * means any value.



```
* * * * *
- - - - -
| | | | |
| | | | ----- Day of week (0 - 7) (Sunday=0 or 7)
| | | ------- Month (1 - 12)
| | --------- Day of month (1 - 31)
| ----------- Hour (0 - 23)
------------- Minute (0 - 59)
```

Crontab generator - https://crontab-generator.org/

References

Apache Airflow Online Documentation, https://airflow.apache.org/docs/apache-airflow/

Spark Online Documentation, http://spark.apache.org/docs/latest/

MongoDB. Online Documentation, https://docs.mongodb.com/

Google Cloud Platform Online Documentation, https://cloud.google.com/docs

Shannon Bradshaw, Eoin Brazil, Kristina Chodorow. MongoDB: The Definitive Guide. O’Reilly Media, Inc. 2019.

Jean-Georges Perrin. Spark in Action, O’Reilly Publications, 2020.

Sadalage, Pramod J., and Martin Fowler. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education, 2012.