Distributed Data Systems

DIANE WOODBRIDGE, PH.D

Contents

Contents

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)

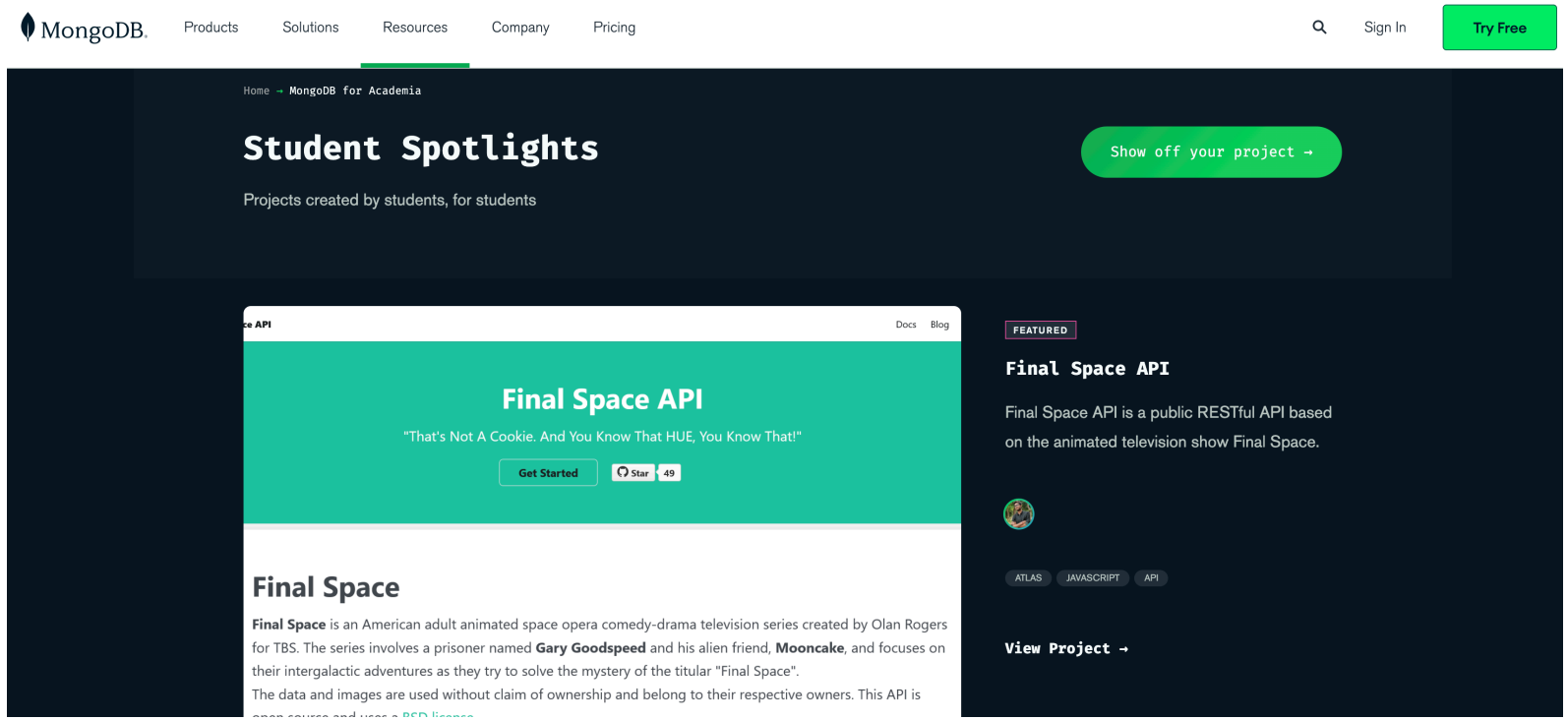
Workflow Management(Airflow)
Distributed Database (NoSQL) Concept
MongoDB and Operations
Running MongoDB on Clusters
Spark Dataframe/SQL
Spark ML

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 including Amazon Web Services, MongoDB Atlas.
3. Distributed Computing
   - Gain experience with SparkSQL and understand its relationship with RDD.
   - Apply distributed machine learning algorithms via Spark ML.
   - Be competent to work with Spark in a distributed computing environment including Google Cloud Platform and Databricks.

Possible Outcomes

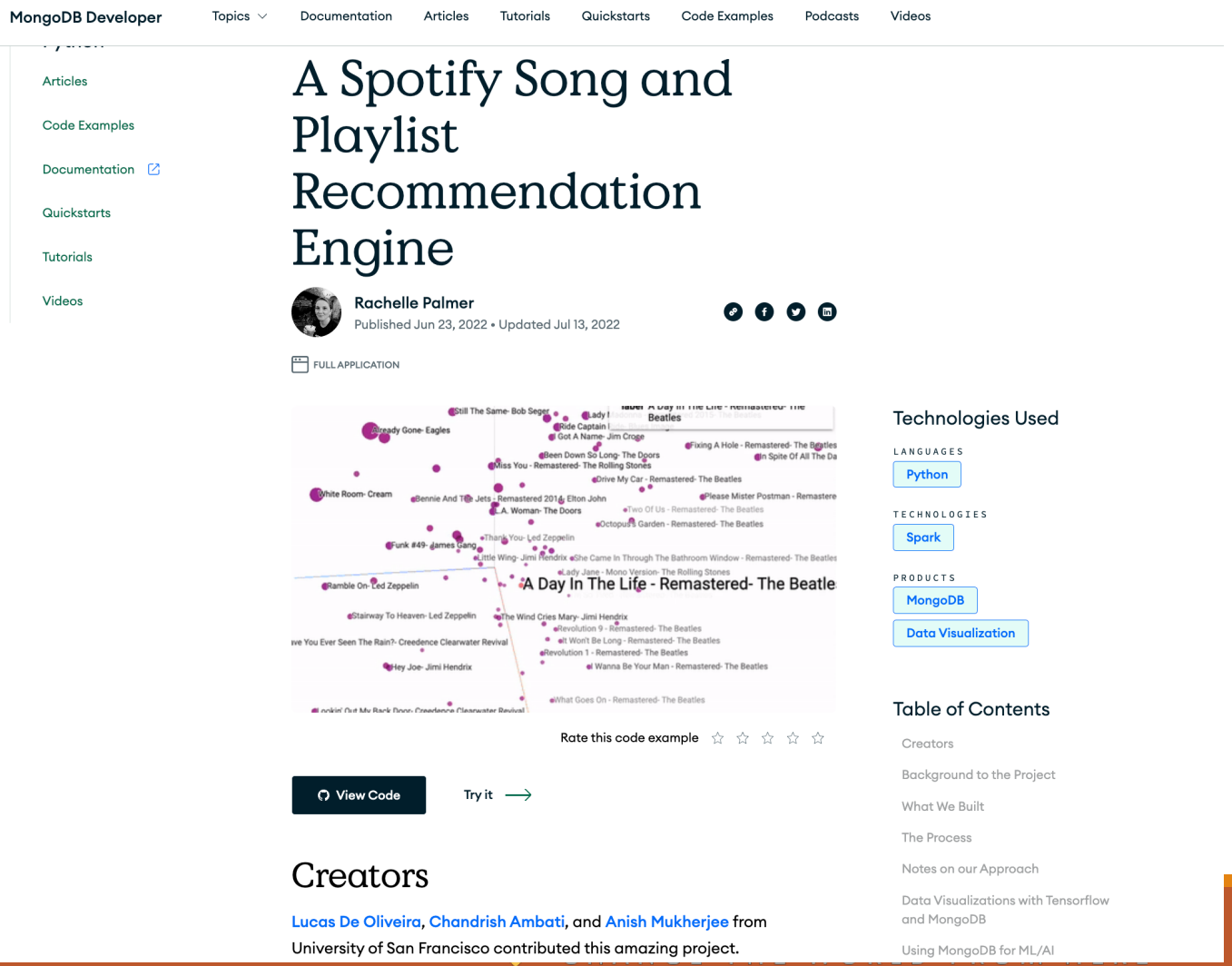
Partnering with MongoDB, selected teams' project will be highlighted at MongoDB Student Spotlight



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

Possible Outcomes

2022



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

Possible Outcomes

2021

Machine Learning-based Meal Detection Using Continuous Glucose Monitoring on Healthy Participants: An Objective Measure of Participant Compliance to Protocol. Victor Palacios, Diane Myung-kyung Woodbridge, Jean L. Fry. International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). November 2021.

Depression Level Prediction in People with Parkinson's Disease during the COVID-19 Pandemic. Hashneet Kaur, Patrick Ka-Cheong Poon, Sophie Yuefei Wang, Diane Myung-kyung Woodbridge.International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). November 2021.

2020

A Machine Learning Approach to Detecting Low Medication State with Wearable Technologies. Andy Cheon, Stephanie Yeoju Jung, Collin Prather, Matt Sarmiento, Kevin Wong, Diane Woodbridge. International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). July 2020.

Sensor Selection for Activity Classification at Smart Home Environments. Nithish Bolleddula, Geoffrey Hung, Daren Ma, Hoda Noorian, Diane Woodbridge. International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). July 2020.

Possible Outcomes

2019

A Scalable Smartwatch-based Medication Adherence Monitoring System using Distributed Machine Learning. Donya Fozoonmayeh, Hai Vu Le, Ekaterina Wittfoth, Chong Geng, Natalie Ha, Jingjue Wang, Maria Vasilenko, Yewon Ahn, Diane Myung-kyung Woodbridge. Journal of Medical Systems (JOMS)

Predicting Unethical Physician Behavior At Scale: A Distributed Computing Framework. Anastasia Quinn Keck, Miguel Romero, Robert Sandor, Diane Myung-kyung Woodbridge, Paul Intrevado. IEEE Smart World Congress. August 2019.

A Scalable and Reliable Model for Real-time Air Quality Prediction. Liying Li, Zhi Li, Lara G. Reichmann, Diane Myung-kyung Woodbridge. IEEE Smart World Congress. August 2019.

The Impact of Bike-Sharing Ridership on Air Quality: A Scalable Data Science Framework. Nina Hua, Victoria Suarez, Rebecca Reilly, Philip Trinh, Paul Intrevado, Diane Myung-kyung Woodbridge. IEEE International Conference on Smart City Innovations. August 2019.

Distributed Data Analytics Framework for Cluster Analysis of Parking Violation. Nan Lin, Evan Liu, Fiorella Tenorio, Xi Yang, Diane Woodbridge. IEEE International Conference on Smart City Innovations. August 2019.

Scalable Real-time Prediction and Analysis of San Francisco Fire Department Response Times. Xu Lian, Sarah Melancon, Jon-Ross Presta, Adam Reevesman, Brian J. Spiering, Diane Myung-kyung Woodbridge. IEEE International Conference on Ubiquitous Intelligence and Computing. August 2019.

Scalable Motor Movement Recognition from Electroencephalography using Machine Learning. Aditi Sharma, Shivee Singh, Brian Wright, Alan Perry, Diane Myung-Kyung Woodbridge, Abbie Popa. IEEE International Workshop on Integrated Smart Healthcare (WISH). July 2019.

A Scalable Automated Diagnostic Feature Extraction System for EEGs. Prakhar Agrawal, Divya Bhargavi, Gokul Krishna G, Xiao Han, Neha Tevathia, Abbie Popa, Nicholas Ross, Diane Myung-Kyung Woodbridge, Barbie Zimmerman-Bier and William Bosl. IEEE International Workshop on Medical Computing (MediComp). July 2019.

2018

Distributed Data Analytics Framework for Smart Transportation. Alexander Howard, Tim Lee, Sara Mahar, Paul Intrevado, Diane Woodbridge. IEEE International Conference on Smart City (SmartCity). 2018

Forecasting Smart Meter Energy Usage using Distributed Systems and Machine Learning. Chris Dong, Lingzhi Du , Feiran Ji , Zizhen Song, Yuedi Zheng, Paul Intrevado, Diane Woodbridge. IEEE International Conference on Smart City (SmartCity). 2018

2017

Machine Learning-Based Product Recommendation using Apache Spark. Lin Chen, Rui Li, Yige Liu, Ruixuan Zhang, Diane Myung-kyung Woodbridge. IEEE UIC International Workshop on Data Science and Computational Intelligence (DSCI). 2017.

Course Evaluation

Attendance and Professionalism - 5 %

Individual Assignment - 20%

Group Project - 25 % (Based on the peer review, the group assignment grades will be weighed.)

- Data Selection/Loading on Google Cloud Storage and Cloud Composer (Airflow).
- Data Preprocessing and Store (Spark & MongoDB).
- Machine Learning, Final Report and Code Submission.

Quiz - 50 %

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.

The expected final score for this course is 85 ± 3 (and close to normal distributions). The following grades will be given if the class grade distribution falls within the expectation.

Score	Letter Grades
90 - 100	A+, A and A-
80 - 90	B+, B and B-
70 - 80	C+, C and C-
Below 70	F

However, if the grade distribution does not meet the aforementioned criteria, grades will be curved.

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.

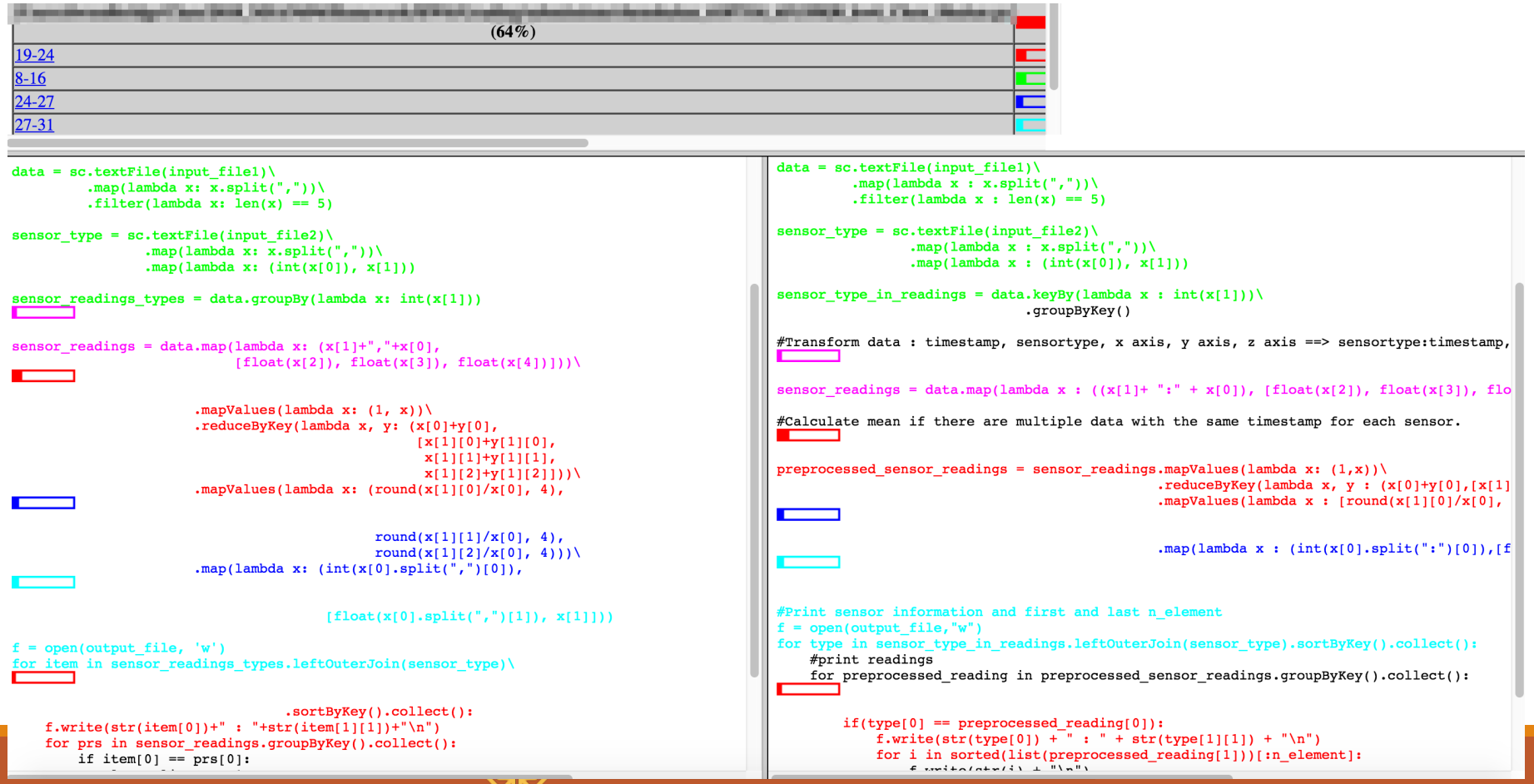
Plagiarized works will receive at least 0 for the assignment or the assignment category. If plagiarism happens, your professionalism score will be 0. Also, further action will be taken based on the program policy (https://myusf.usfca.edu/arts-sciences/data-science/program-policies).

- 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.



(64%)

19-24

8-16

24-27

27-31

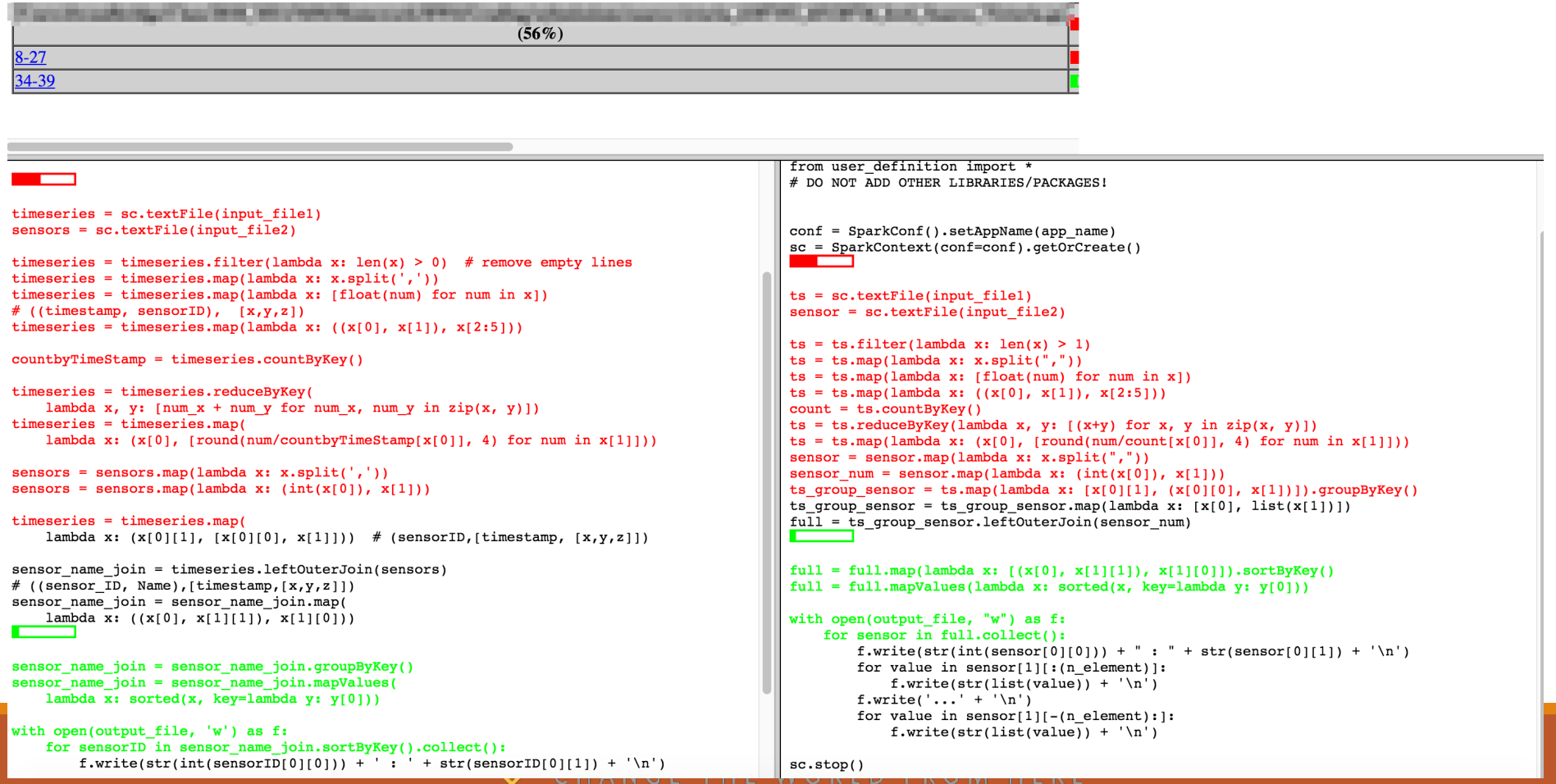
```
data = sc.textFile(input_file1)\
         .map(lambda x: x.split(","))\
         .filter(lambda x: len(x) == 5)

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.



(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 - T/F : 10:00 - 11:50, Room 155-156

Session 2 - T : 1:00 - 2:50, F : 2:30 - 4:30, Room 155-156

Course Schedule

LEARNING OBJECTIVES

- Week 1 - Course Intro, Workflow and Apache Airflow, NoSQL Concept
- Week 2 - 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 - Databricks and Lab, Spark ML
- Week 7 - Final Project

Course Schedule

- Individual Assignment

Date	Assignment
2023-02-06	HW1
2023-02-13	HW2
2023-02-20	HW3
2023-02-27	HW4

- Group Assignment

Date	Assignment
2023-02-07	Task 1
2023-02-25	Task 2
2023-03-10	Task 3 : Final Project Submission

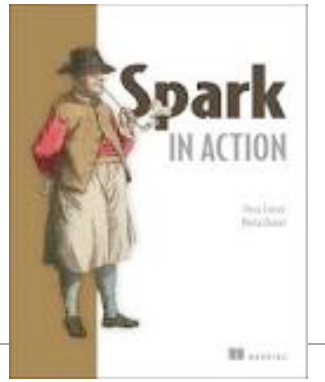
- Quiz

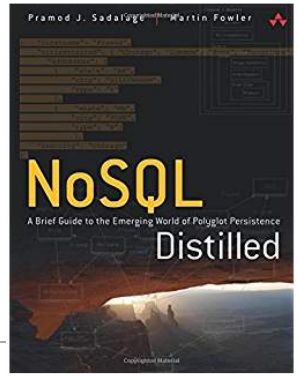
Date	Assignment
2023-02-14	Quiz1
2023-02-28	Quiz2
2023-03-07	Quiz3

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

Databricks Online Documentation, https://docs.databricks.com/

Zecevic, Petar, et al. Spark in Action, Manning, 2016.

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

Student Engagement

Example Data : https://github.com/dianewoodbridge/2023-msds697-distributed-data-systems

Piazza : https://piazza.com/usfca/spring2023/msds697

Poll : https://pollev.com/msds

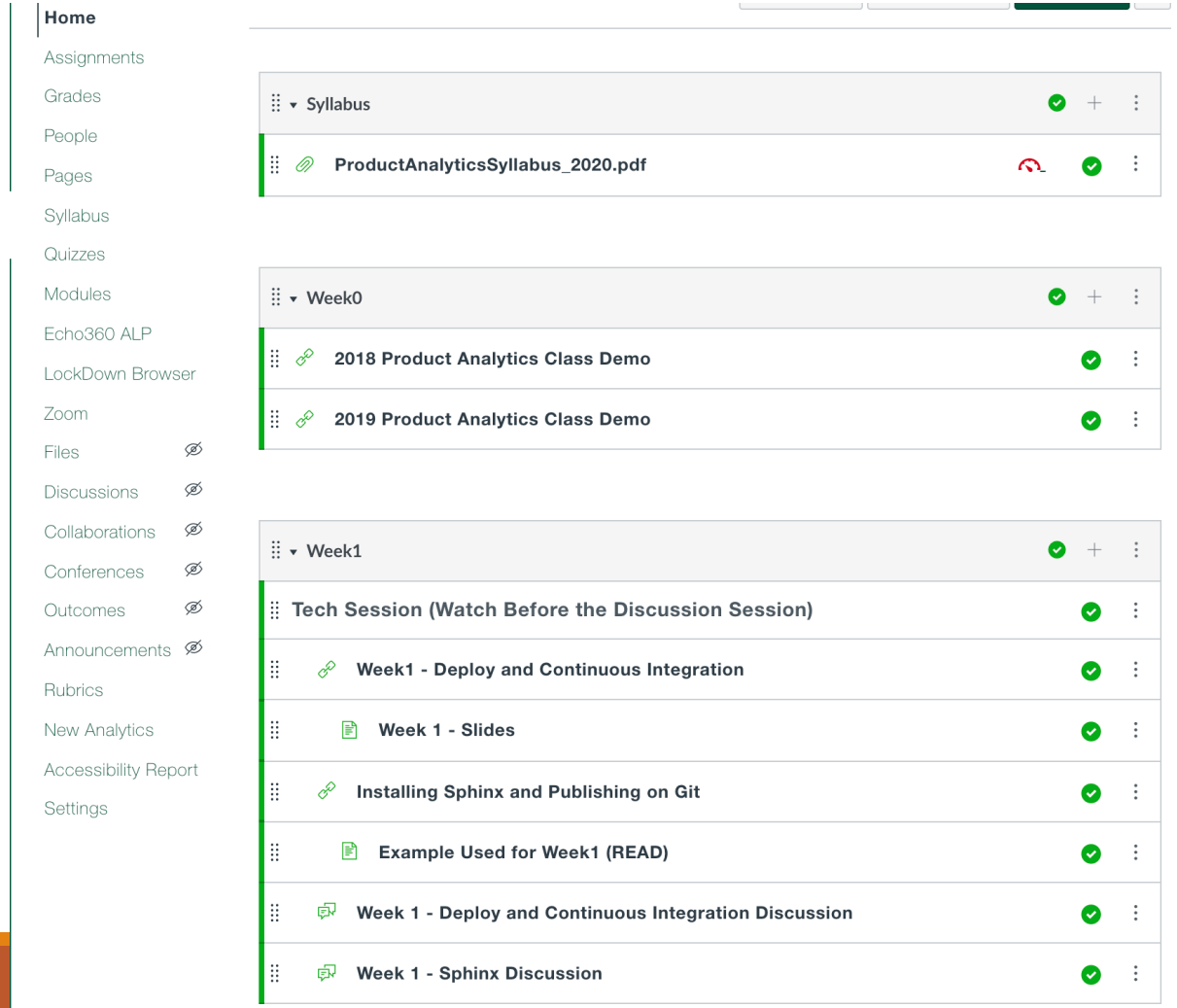
Canvas

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

Student Engagement

Canvas

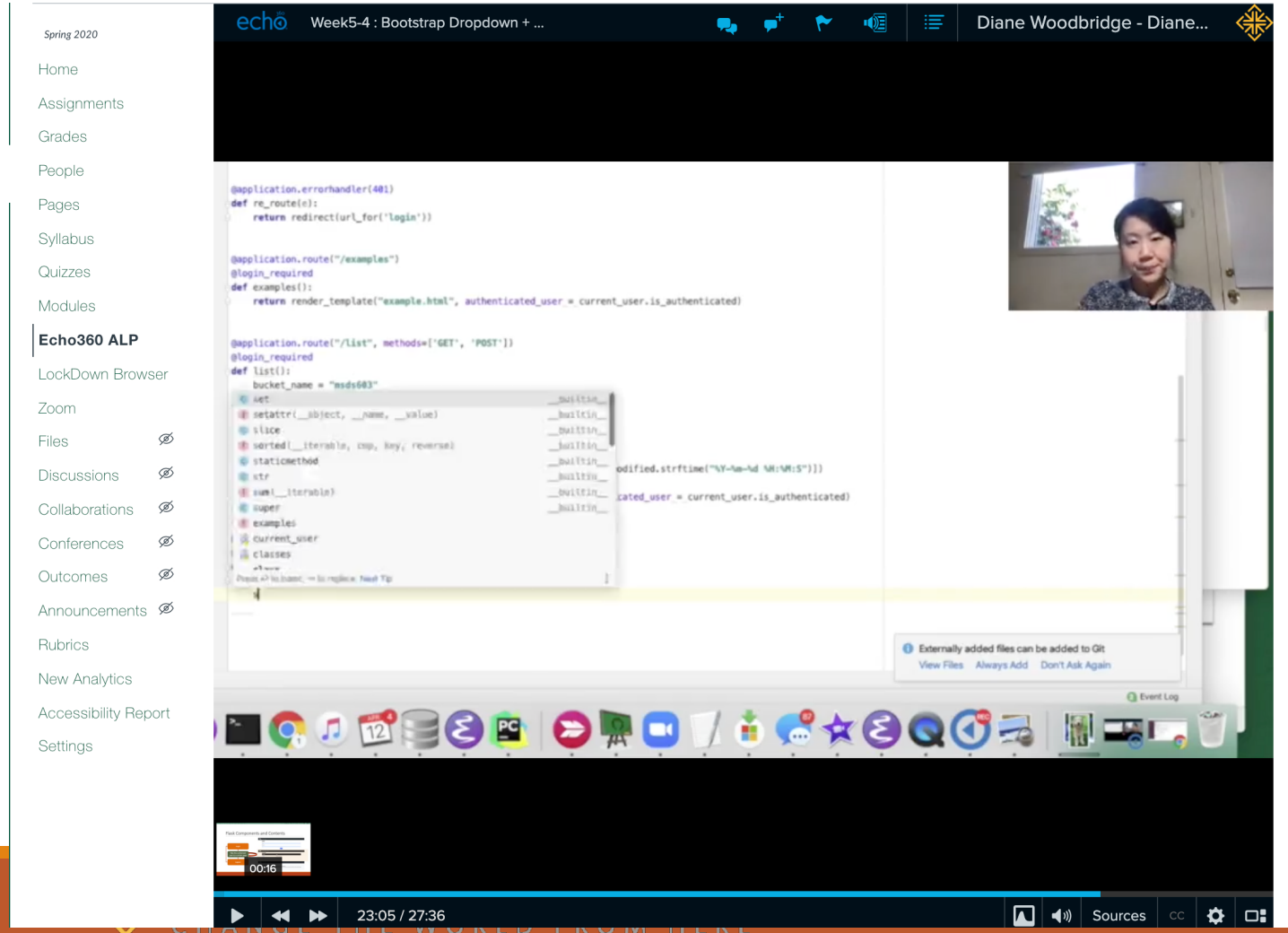
- Modules



Student Engagement

Canvas

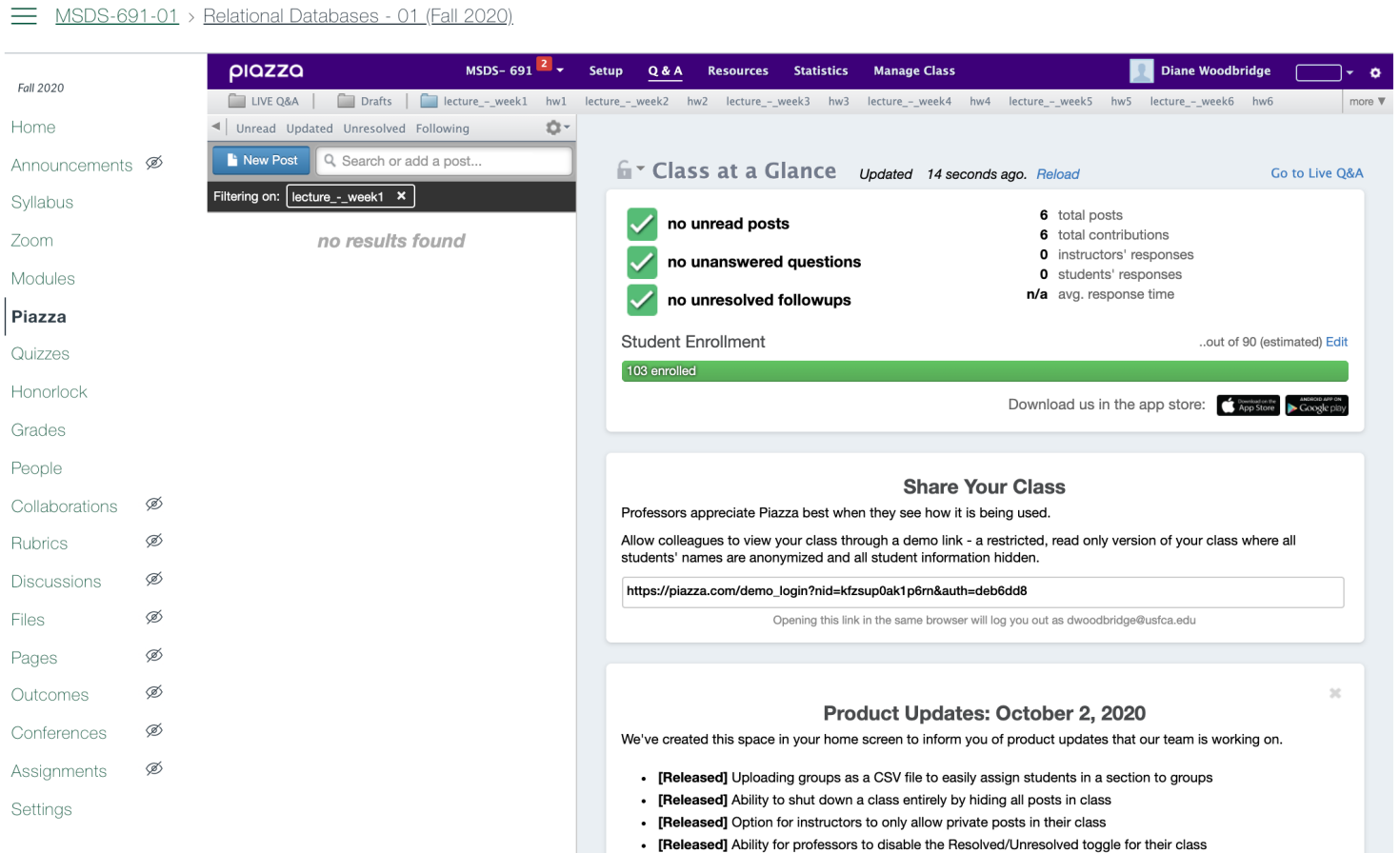
- Echo 360



Student Engagement

Canvas

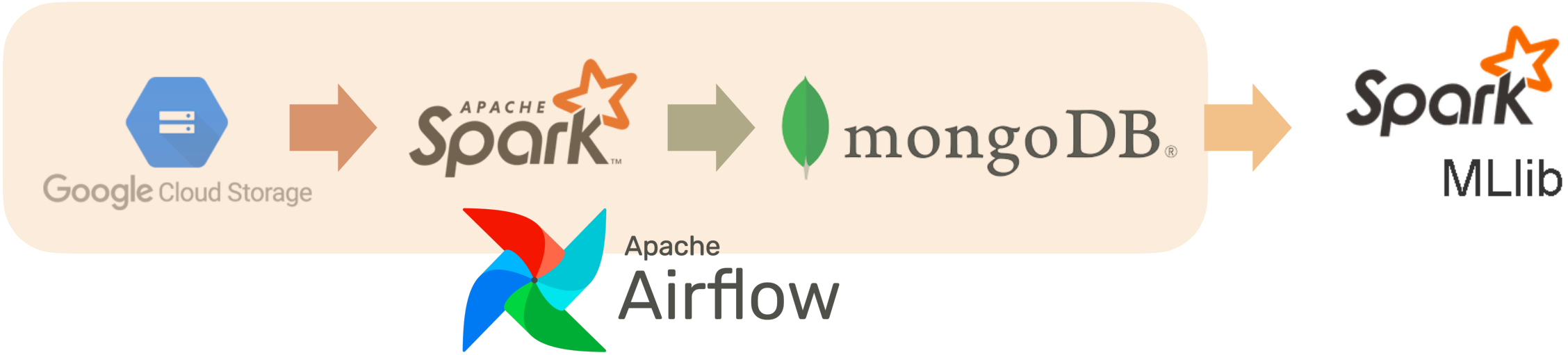
Piazza - For discussing lectures and assignments. (Preferable)



Questions?

Contents

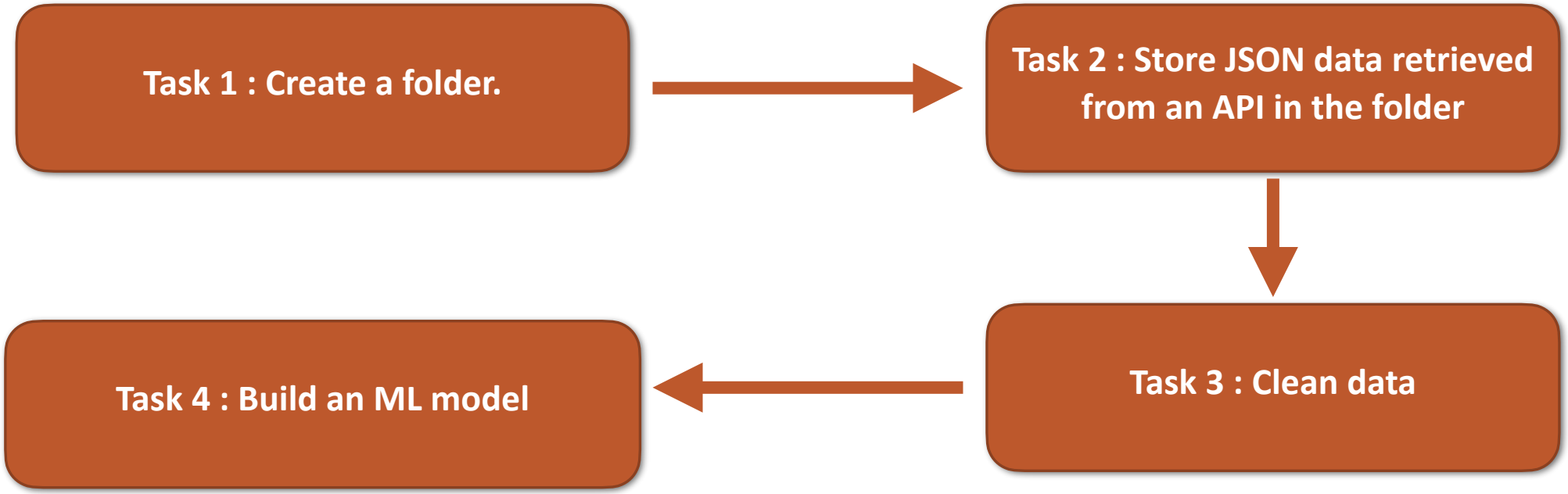
Building an Automated Scalable ETL Pipeline in MSDS697



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)**



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

Contents

Airflow

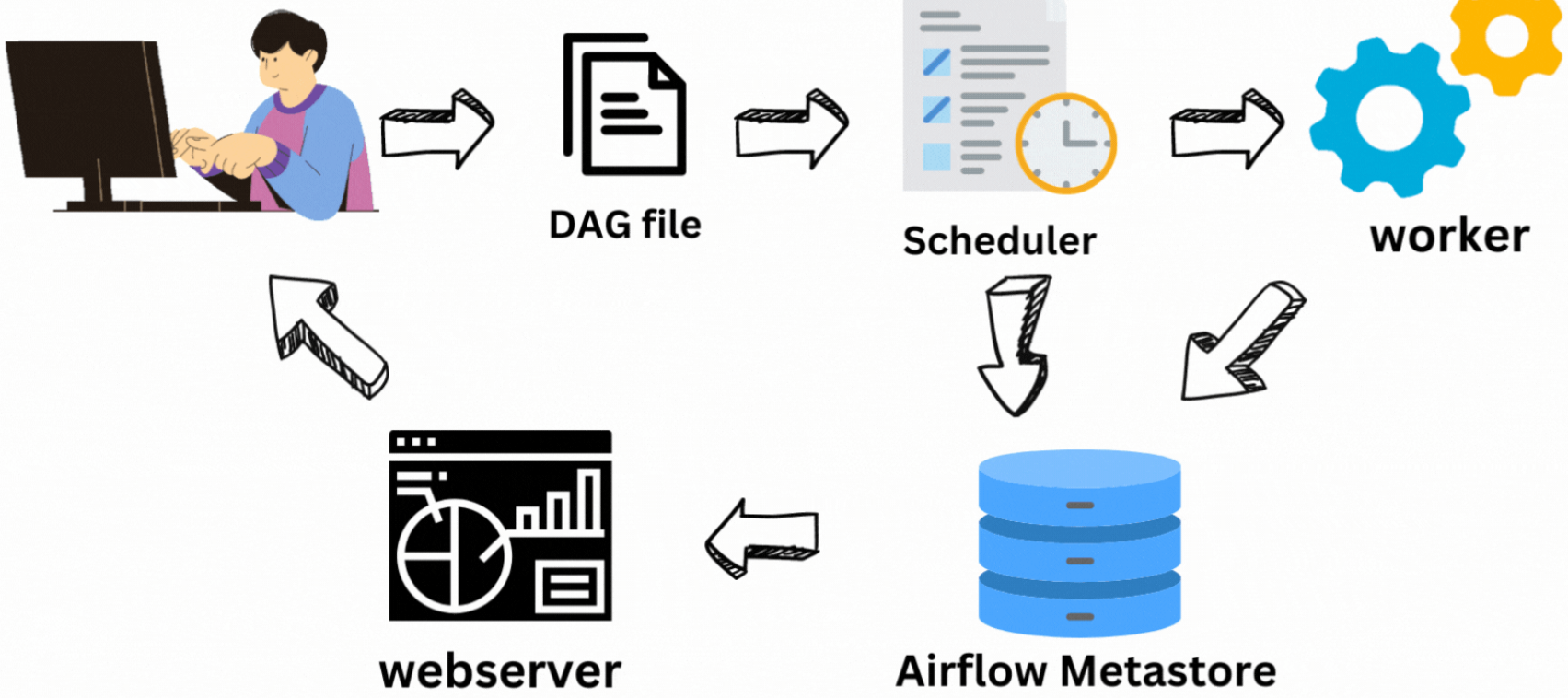
Apache Airflow

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 (3)
  - Scheduler - Parse DAGs, check schedules, and schedule tasks.
  - Workers - Executing tasks.
  - Webserver - Visualize DAGs and provide main interface to monitor DAGs.

https://airflow.apache.org

Airflow



1. User writes DAG
2. Scheduler parses DAG and schedules tasks, considering schedules and dependencies.
3. Worker executes scheduled tasks.
4. User monitors workflow execution schedules and results displayed by the webserver.

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

https://airflow.apache.org

What are the three main components of Apache Airflow?

- Scheduler, Workers, Webserver
- DAG, Scheduler, Webserver
- DAG, Worker, Webserver
- Node, Edge, Direction
- None of the above

What are the three main components of Apache Airflow?

Scheduler, Workers, Webserver

DAG, Scheduler, Webserver

DAG, Worker, Webserver

Node, Edge, Direction

None of the above

What are the three main components of Apache Airflow?

Scheduler, Workers, Webserver

DAG, Scheduler, Webserver

DAG, Worker, Webserver

Node, Edge, Direction

None of the above

Airflow Installation

```
conda install -c conda-forge airflow
```

```
(DistributedComputing) dwoodbridge@ML-ITS-210588 Class % conda install -c conda-forge airflow
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Solving environment: failed with repodata from current_repodata.json, will retry with next repodata source.
Collecting package metadata (repodata.json): done
```

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 web server.

Contents

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/

Which one is same as @yearly?

- 00**1
- 0011*
- **11*
- 0****

Which one is same as @yearly?

- `00**1`
- `0011*`
- `**11*`
- `0****`

Which one is same as @yearly?

- 00**1
- 0011*
- **11*
- 0****

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
```

Executes a Python callable

Id of task

Python Function Name

a dictionary of keyword arguments used in func_1

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

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
```

Executes a Bash script

Id of task

Bash commands

Note : Other operator types in Airflow (https://airflow.apache.org/docs/apache-airflow/stable/_api/airflow/operators/index.html)
https://airflow.apache.org/docs/apache-airflow/1.10.4/_api/airflow/operators/python_operator/index.html
https://airflow.apache.org/docs/apache-airflow/stable/howto/operator/bash.html

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
```

Task dependency - The direction of edges between tasks.

```
first_task >> [second_task, third_task]
third_task << fourth_task
```

https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/dags.html

Example 1

Create a DAG called “download_data_step1" where it starts on 2023-01-23 and called daily.

- Task 1 (create_dirs) : create {data_dir}/{yesterday} if not exists.
- Task 2 (download_sf_law_enforcement_data) : calls the download_sf_law_enforcement_data function with keyword arguments.
- Task 2 should happen if Task 1 is successful.

Execute it daily

Exe

Example 1

Create a DAG called “download_data_step1" where it starts on 2023-01-23 and called daily.

```
with DAG(dag_id="download_data_step1",
         start_date=datetime(2023, 1, 14),
         schedule_interval='@daily') as dag:

    # https://github.com/apache/airflow/discussions/24463
    os.environ["no_proxy"] = "*"  # set this for airflow errors.

    create_dirs_op = BashOperator(task_id="create_dirs",
                                  bash_command=f"mkdir -p {data_dir}/{yesterday}")

    download_sf_law_enforcement = PythonOperator(task_id="download_sf_law_enforcement",
                                                 python_callable=download_sf_law_enforcement_data,
                                                 op_kwargs={'url': url,
                                                            'app_token': app_token,
                                                            'data_dir': data_dir,
                                                            'sub_uri': sub_uri,
                                                            'data_limit': data_limit,
                                                            'yesterday': yesterday})


    create_dirs_op >> download_sf_law_enforcement
```

Airflow Execution

1. Initialize the metastore

Only for the first time

```
$ airflow db init
```

2. Create a dags directory under ~/airflow

```
$ mkdir ~/airflow/dags
```

This is where your .py files should located.

3. Create a user

```
$ airflow users create --username admin --firstname FIRST_NAME --lastname LAST_NAME --role Admin
--email admin@example.org
```

Airflow requires users to authenticated to login to the web server.

3. Copy the DAG into the DAGs directory

```
$ cp *.py ~/airflow/dags/
```

4. Start the scheduler and web server (Open two separate terminals)

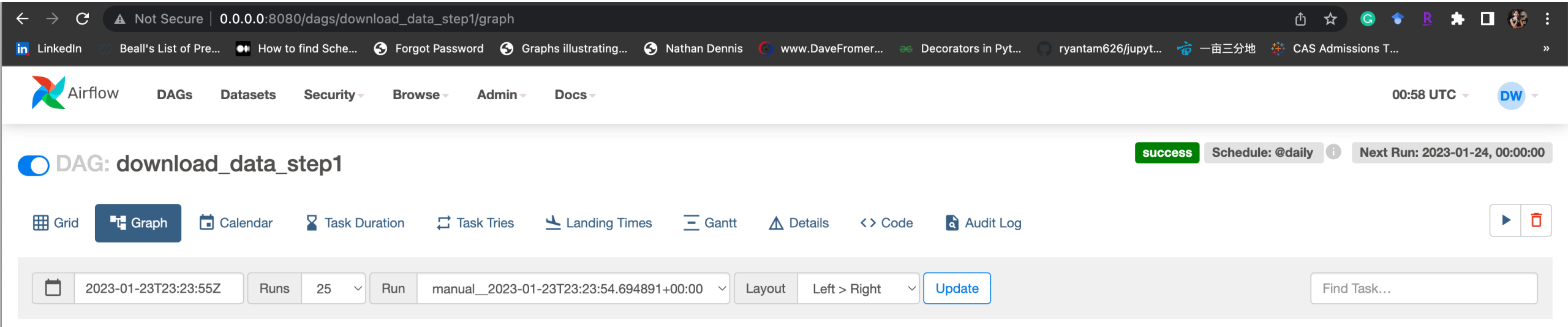
```
$ airflow scheduler
$ airflow webserver
```

https://airflow.apache.org/docs/apache-airflow/1.10.13/howto/initialize-database.html

Airflow Webserver

You can access the Airflow Webserver via 0.0.0.0:8080 when runs locally.

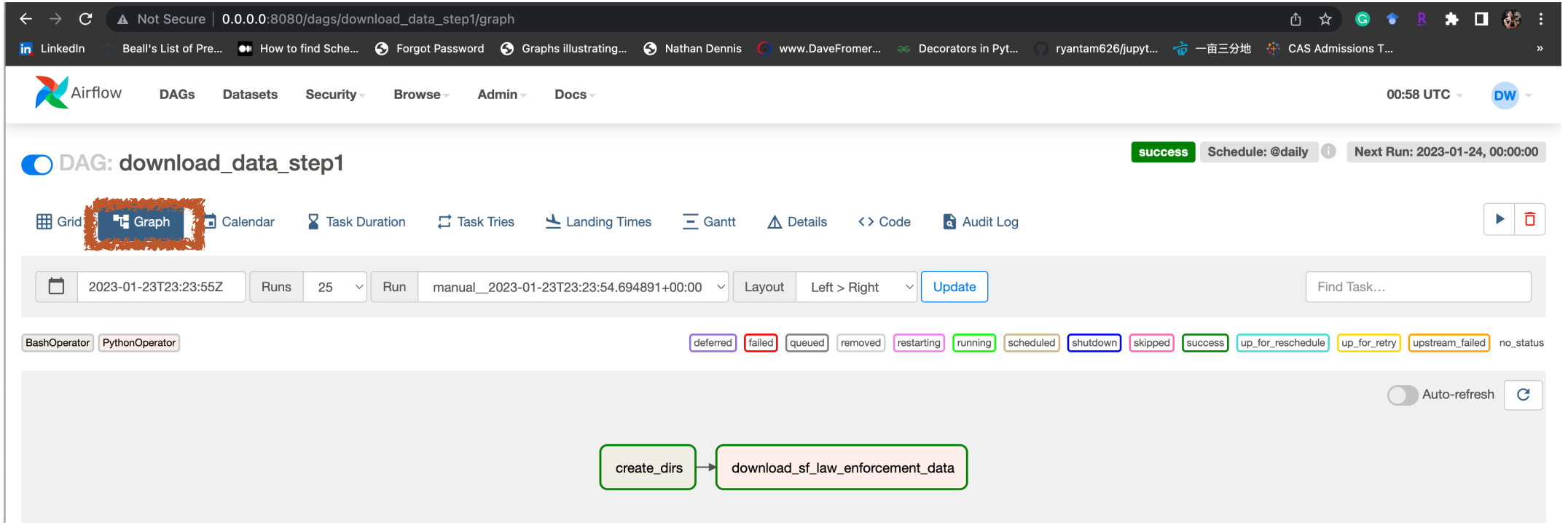
-  : Turn on/off the DAG
- Click the dag_id to see more details
-  : Execute the DAG manually



Airflow Webserver

DAG visualization

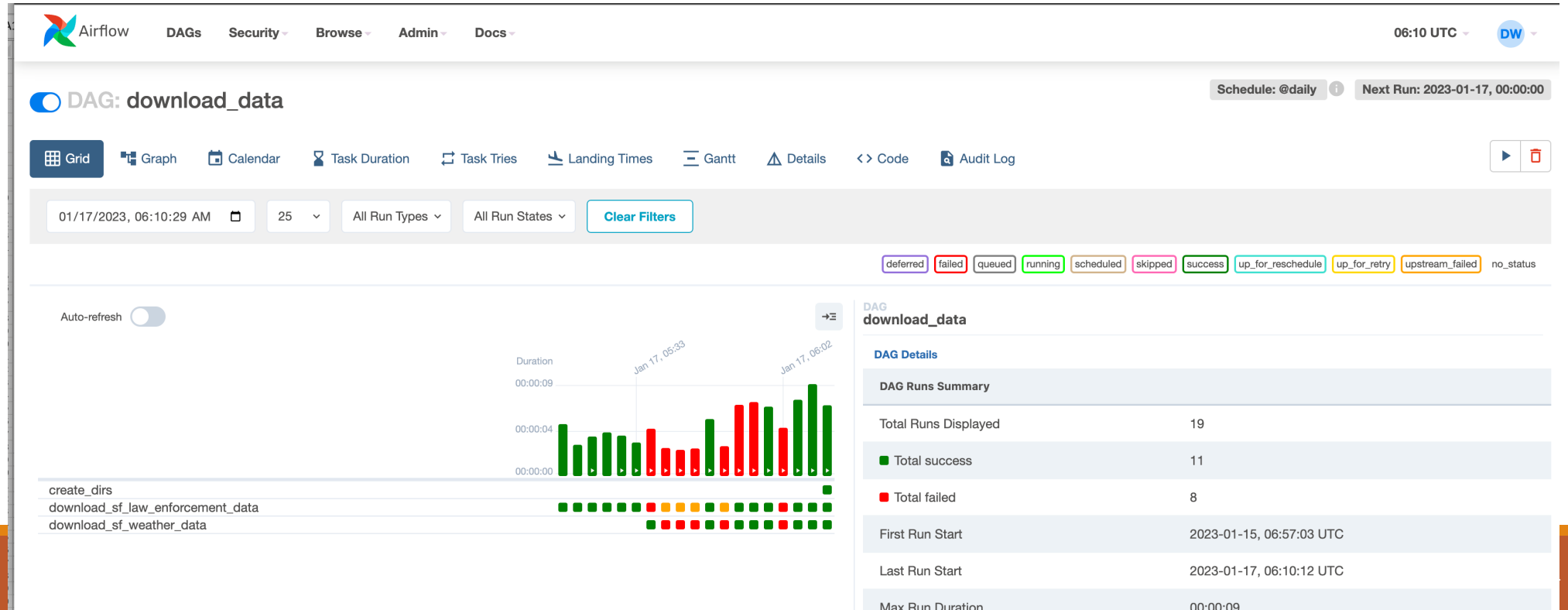
- dag_id ➡ Graph shows a dependency graph of tasks.



Airflow Webserver

Debugging

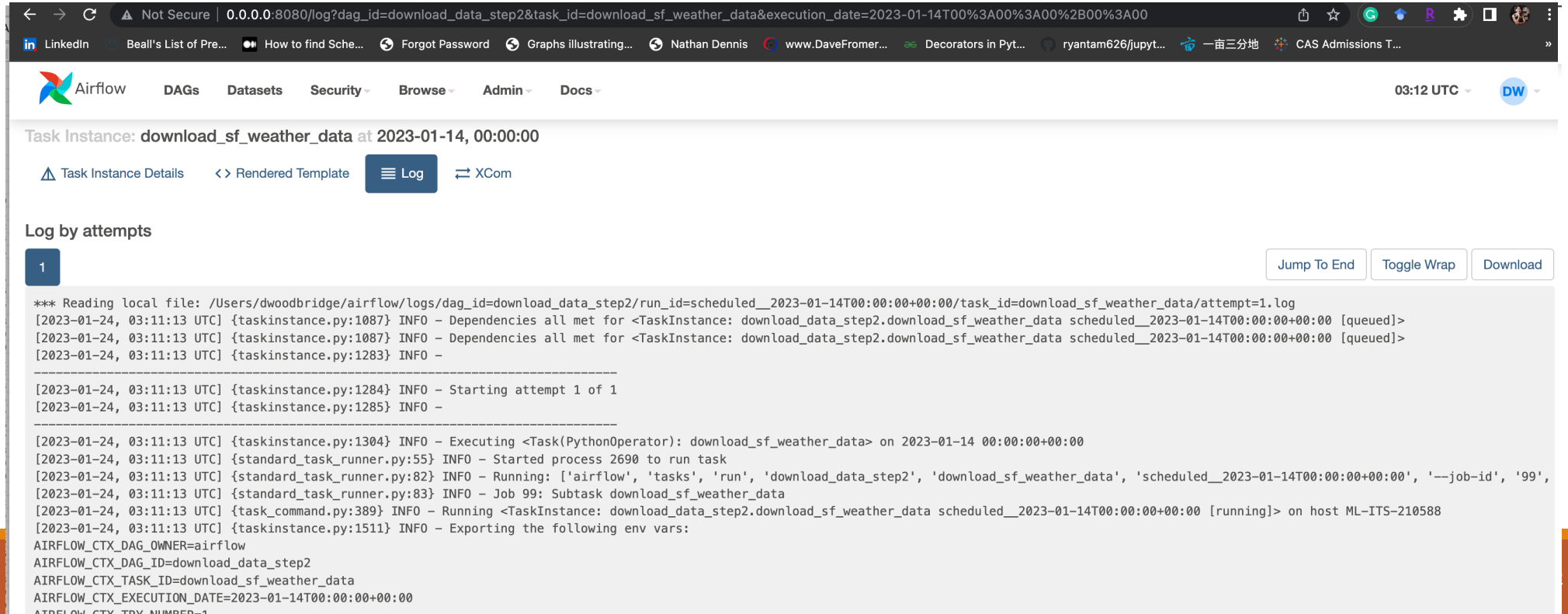
- If fails and needs to debug, go to “Details” ➡ Click “Failed” ➡ Click “task_id” for the failed DAG ➡ Click “Logs”



Airflow Webserver

Debugging

- If fails and needs to debug, go to “Details” ➡ Click “Failed” ➡ Click “task_id” for the failed DAG ➡ Click “Logs”

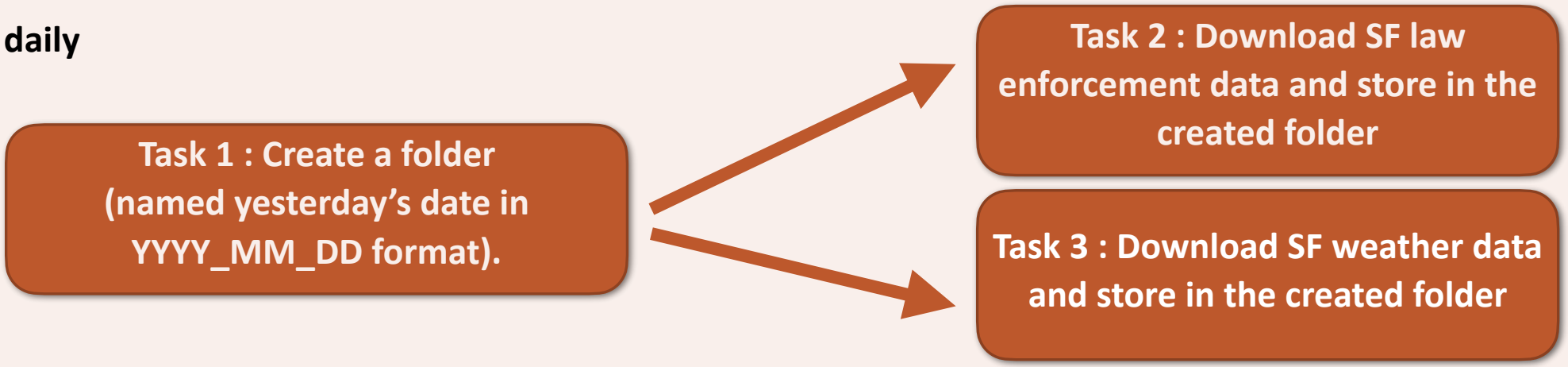


Example 2

Create a DAG called “download_data_step2” where it starts on 2023-01-23 and called daily.

- Task 1 (create_dirs) : create {data_dir}/{yesterday} {data_dir}/{three_days_ago} if not exists.
- Task 2 (download_sf_law_enforcement_data) : calls the download_sf_law_enforcement_data function with keyword arguments.
- Task 3 (download_sf_weather_data) : calls the download_sf_weather_data function with keyword arguments
- Task 2 and Task 3 should happen if Task 1 is successful.

Execute it daily



Choose the dependency representing Example 2

Task1 >> Task2 >> Task3

Task1 >> [Task2, Task3]

Choose the dependency representing Example 2

Task1 >> Task2
>> Task3

Task1 >>
[Task2, Task3]

Choose the dependency representing Example 2

Task1 >> Task2
>> Task3

Task1 >>
[Task2, Task3]

Example 2

```
with DAG(dag_id="download_data_step2",
         start_date=datetime(2023, 1, 14),
         schedule_interval='@daily') as dag:

    # https://github.com/apache/airflow/discussions/24463
    os.environ["no_proxy"] = "*"  # set this for airflow errors.

    create_dirs_op = BashOperator(task_id="create_dirs",
                                  bash_command=f"mkdir -p {data_dir}/{yesterday} " +
                                               f"{data_dir}/{three_days_ago}")

    download_sf_law_enforcement = PythonOperator(task_id="download_sf_law_enforcement",
                                                 python_callable=download_sf_law_enforcement_data,
                                                 op_kwargs={'url': sf_data_url,
                                                            'sub_uri': sf_data_sub_uri,
                                                            'data_limit': data_limit,
                                                            'app_token': sf_data_app_token,
                                                            'data_dir': data_dir,
                                                            'yesterday': str(yesterday)})

    download_sf_weather_data = PythonOperator(task_id="download_sf_weather_data",
                                              python_callable=download_sf_weather_data,
                                              op_kwargs={'noaa_token': noaa_token,
                                                         'api_url': noaa_api_url,
                                                         'three_days_ago': three_days_ago,
                                                         'data_dir': data_dir})

    create_dirs_op >> [download_sf_law_enforcement, download_sf_weather_data]
```

Should I leave my computer on for Airflow?

Google Cloud Composer

Google Cloud Composer Environment

- Self-contained Airflow deployed on Google Kubernetes Engine
- A recorded tutorial will be posted.

Google Cloud

=

Google Cloud Composer

https://cloud.google.com/composer/docs/how-to/managing/creating

Comments (What you liked/disliked so far? What should I do for you?)

Contents

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

Databricks Online Documentation, https://docs.databricks.com/

Zecevic, Petar, et al. Spark in Action, Manning, 2016.

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