

Distributed Data Systems

DIANE WOODBRIDGE, PH.D



UNIVERSITY OF SAN FRANCISCO
CHANGE THE WORLD FROM HERE

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 Data Systems (MSDS 697)

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



Distributed Computing (MSDS 694)

Distributed Computing Concept
Basic Spark Operations
Running Spark on Clusters



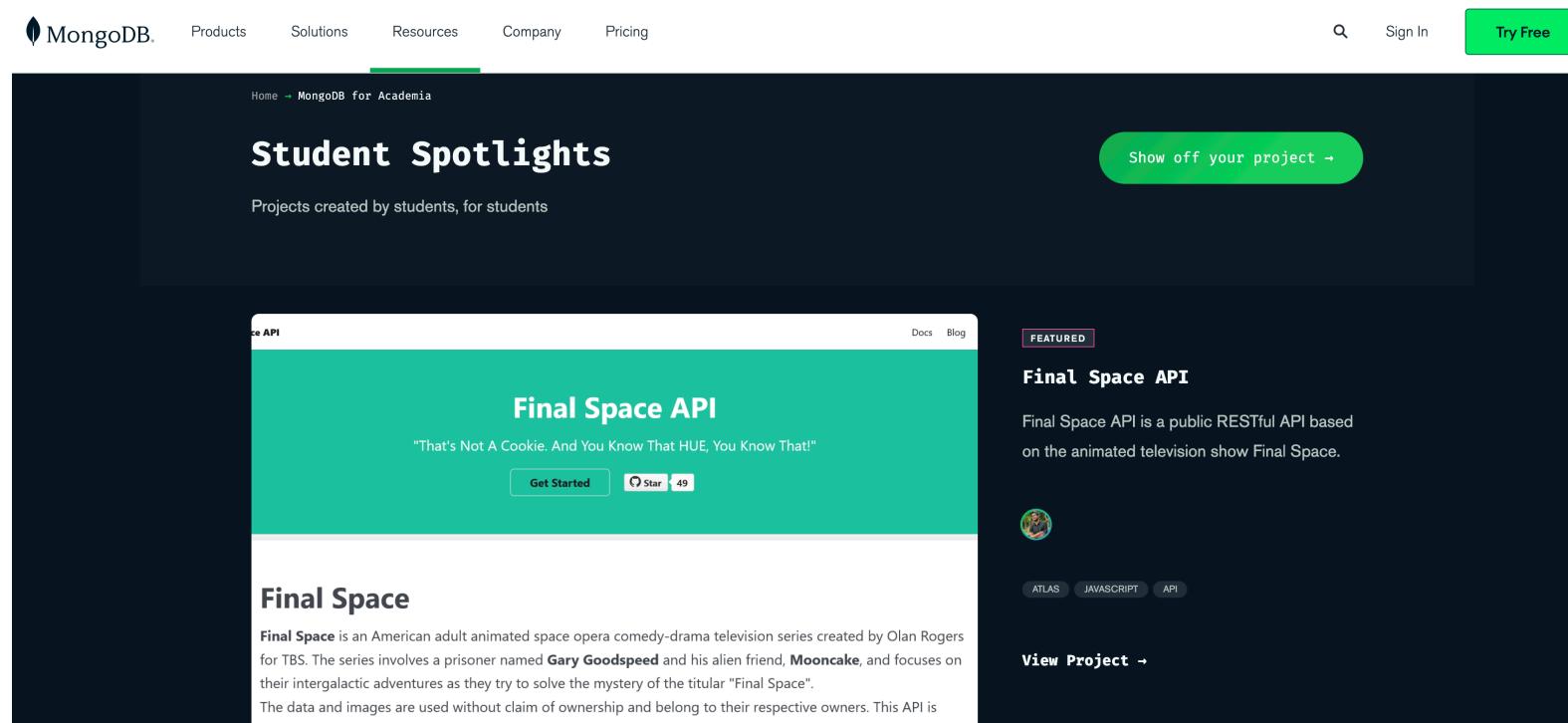
Course Learning Objectives

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

MongoDB Developer Topics ▾ Documentation Articles Tutorials Quickstarts Code Examples Podcasts Videos

A Spotify Song and Playlist Recommendation Engine

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

FULL APPLICATION

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

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

View Code Try it →

Creators

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

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.

```
data = sc.textFile(input_file1)\n    .map(lambda x: x.split(","))\n    .filter(lambda x: len(x) == 5)\n\nsensor_type = sc.textFile(input_file2)\n    .map(lambda x: x.split(","))\n    .map(lambda x: (int(x[0]), x[1]))\n\nsensor_readings_types = data.groupBy(lambda x: int(x[1]))\n\nsensor_readings = data.map(lambda x: (x[1]+", "+x[0],\n    [float(x[2]), float(x[3]), float(x[4])]))\\n\n    .mapValues(lambda x: (1, x))\n    .reduceByKey(lambda x, y: (x[0]+y[0],\n        [x[1][0]+y[1][0],\n         x[1][1]+y[1][1],\n         x[1][2]+y[1][2]]))\\n\n    .mapValues(lambda x: (round(x[1][0]/x[0], 4),\n        round(x[1][1]/x[0], 4),\n        round(x[1][2]/x[0], 4)))\\n\n    .map(lambda x: (int(x[0].split(",")[0]),\n\n        [float(x[0].split(",")[1]), x[1]]))\n\nf = open(output_file, 'w')\nfor item in sensor_readings_types.leftOuterJoin(sensor_type)\\n\n    .sortByKey().collect():\n\n    f.write(str(item[0]) + " : " + str(item[1][1]) + "\n")\n    for prs in sensor_readings.groupByKey().collect():\n        if item[0] == prs[0]:\n\n            (64%)
```

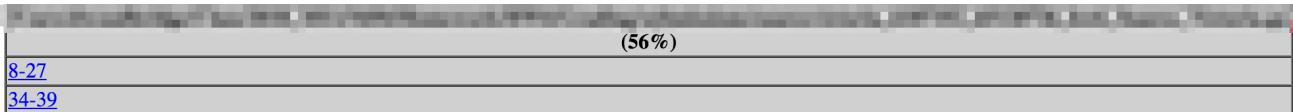
```
data = sc.textFile(input_file1)\\n    .map(lambda x: x.split(","))\\n    .filter(lambda x: len(x) == 5)\\n\nsensor_type = sc.textFile(input_file2)\\n    .map(lambda x: x.split(","))\\n    .map(lambda x: (int(x[0]), x[1]))\\n\nsensor_type_in_readings = data.keyBy(lambda x: int(x[1]))\\n    .groupByKey()\\n\n#Transform data : timestamp, sensortype, x axis, y axis, z axis ==> sensortype:timestamp,\n\nsensor_readings = data.map(lambda x : ((x[1]+ ":" + x[0]), [float(x[2]), float(x[3]), float(x[4])]))\\n\n#Calculate mean if there are multiple data with the same timestamp for each sensor.\n\npreprocessed_sensor_readings = sensor_readings.mapValues(lambda x: (1,x))\\n    .reduceByKey(lambda x, y : (x[0]+y[0],[x[1]+y[1],x[2]+y[2]]))\\n    .mapValues(lambda x : [round(x[1][0]/x[0], 4),\\n        round(x[1][1]/x[0], 4),\\n        round(x[1][2]/x[0], 4)])\\n\n.map(lambda x : (int(x[0].split(":")[0]),[f\n\n#Print sensor information and first and last n_element\nf = open(output_file,"w")\nfor type in sensor_type_in_readings.leftOuterJoin(sensor_type).sortByKey().collect():\n    #print readings\n    for preprocessed_reading in preprocessed_sensor_readings.groupByKey().collect():\n\n        if(type[0] == preprocessed_reading[0]):\n            f.write(str(type[0]) + " : " + str(type[1][1]) + "\n")\n            for i in sorted(list(preprocessed_reading[1]))[:n_element]:\n                f.write(str(i) + "\n")
```



Course Evaluation

Changed lines and
variable names

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



```
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], sensorID), [x[1]]))
countbyTimeStamp = ts.countByKey()

timeseries = ts.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.collect():
        f.write(str(int(sensorID[0][0])) + ' : ' + str(sensorID[0][1]) + '\n')
        for value in sensorID[1][:-(n_element)]:
            f.write(str(list(value)) + '\n')
        f.write('...' + '\n')
        for value in sensorID[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/>

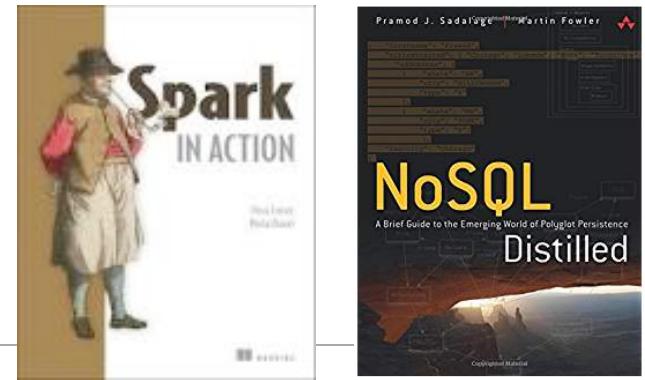
MongoDB. Online Documentation, <https://docs.mongodb.com/>

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

The screenshot shows the Canvas LMS interface with the 'Modules' page selected. The left sidebar lists various navigation options. The main area displays a list of modules, each with a title, preview, and three-dot menu.

- Home**
- Assignments
- Grades
- People
- Pages
- Syllabus
- Quizzes
- Modules**
- Echo360 ALP
- LockDown Browser
- Zoom
- Files
- Discussions
- Collaborations
- Conferences
- Outcomes
- Announcements
- Rubrics
- New Analytics
- Accessibility Report
- Settings

Module	Description	Status	Actions
Syllabus	ProductAnalyticsSyllabus_2020.pdf	Completed	Checkmark, +, ...
Week0	2018 Product Analytics Class Demo 2019 Product Analytics Class Demo	Completed	Checkmark, +, ...
Week1	Tech Session (Watch Before the Discussion Session) Week1 - Deploy and Continuous Integration Week 1 - Slides Installing Sphinx and Publishing on Git Example Used for Week1 (READ) Week 1 - Deploy and Continuous Integration Discussion Week 1 - Sphinx Discussion	Completed	Checkmark, +, ...

Student Engagement

Canvas
Echo 360

The screenshot shows a video call interface from Echo 360. On the right, there is a video feed of a woman with short dark hair, smiling. On the left, there is a code editor window displaying Python code related to file handling and bucket operations. Below the video feed, there is a toolbar with various icons for file operations like 'View Files', 'Always Add', and 'Don't Ask Again'. At the bottom, there is a navigation bar with icons for different applications and a timer indicating 00:18.

echo Week5-4 : Bootstrap Dropdown + ...

Diane Woodbridge - Diane...

Spring 2020

Home

Assignments

Grades

People

Pages

Syllabus

Quizzes

Modules

Echo360 ALP

LockDown Browser

Zoom

Files

Discussions

Collaborations

Conferences

Outcomes

Announcements

Rubrics

New Analytics

Accessibility Report

Settings

```
@application.errorhandler(401)
def re_route():
    return redirect(url_for('login'))

@application.route("/examples")
@login_required
def examples():
    return render_template("example.html", authenticated_user = current_user.is_authenticated)

@application.route("/list", methods=['GET', 'POST'])
@login_required
def list():
    bucket_name = "msds603"
    net
    setattr(__object, __name, __value)
    slice
    sorted(__iterable, key, reverse)
    staticmethod
    str
    sum(__iterable)
    super
    examples
    current_user
    classes
    <-->
    Date->In basic, --> replace, Need To:
```

Externally added files can be added to Git

View Files Always Add Don't Ask Again

Event Log

23:05 / 27:36

CHANGE THE WORLD FROM HERE

22

Student Engagement

Canvas

Piazza - For discussing lectures and assignments. (Preferable)

The screenshot shows the Piazza platform interface for the class MSDS-691. The top navigation bar includes links for LIVE Q&A, Drafts, lecture_-_week1, hw1, lecture_-_week2, hw2, lecture_-_week3, hw3, lecture_-_week4, hw4, lecture_-_week5, hw5, lecture_-_week6, hw6, and more. The user profile of Diane Woodbridge is visible. The main dashboard features a "Class at a Glance" section with the following statistics:

Category	Value
no unread posts	6 total posts
no unanswered questions	6 total contributions
no unresolved followups	0 instructors' responses
	0 students' responses
	n/a avg. response time

The "Student Enrollment" section shows 103 enrolled students, with a note that 103 out of 90 are estimated. A "Share Your Class" section provides a demo link: https://piazza.com/demo_login?nid=kfzsup0ak1p6rn&auth=deb6dd8. A product update message at the bottom right states: "Product Updates: October 2, 2020" and lists several new features.

Left sidebar menu items include: Fall 2020, Home, Announcements, Syllabus, Zoom, Modules, Piazza, Quizzes, Honorlock, Grades, People, Collaborations, Rubrics, Discussions, Files, Pages, Outcomes, Conferences, Assignments, and Settings.

Top navigation bar items include: Setup, Q & A, Resources, Statistics, Manage Class, and a user profile for Diane Woodbridge.

Bottom right corner of the dashboard displays a "Product Updates: October 2, 2020" message with a list of new features.

Questions?

Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollev.com/app

Contents

Course Overview

Workflow Management

Apache Airflow

Airflow DAG Creation



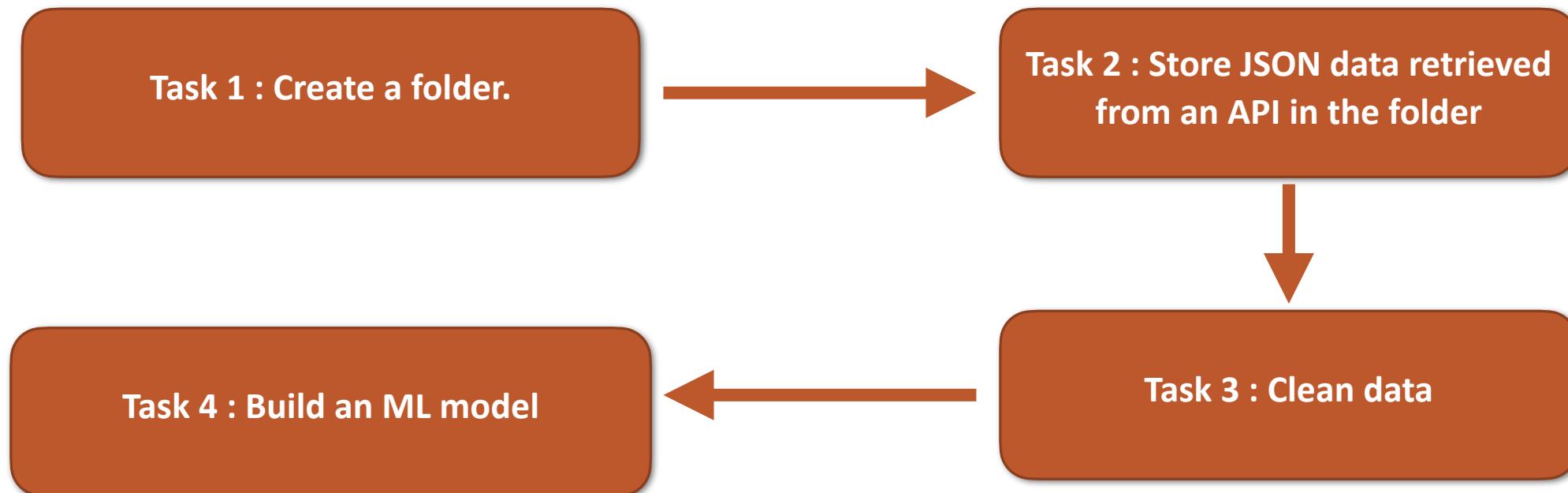
Building an Automated Scalable ETL Pipeline in MSDS697



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

Course Overview

Workflow Management

Apache Airflow

Airflow DAG Creation





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.

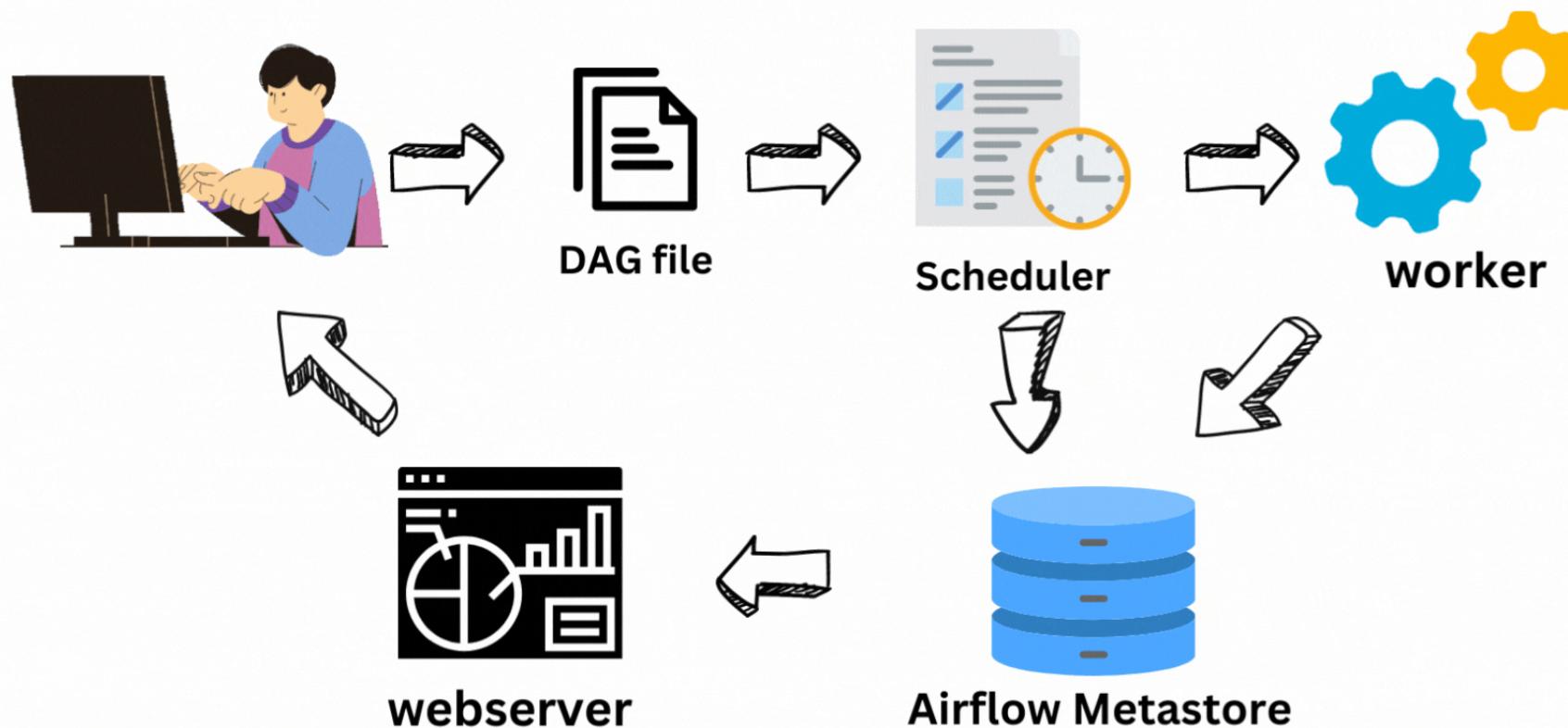
<https://airflow.apache.org>

Airflow



Apache

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

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
```

<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

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

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)
```

When poll is active, respond at **pollev.com/msds**

Text **MSDS** to **37607** once to join



Which one is same as @yearly?

0 0 * * 1

0 0 1 1 *

* * 1 1 *

0 * * * *

When poll is active, respond at **pollev.com/msds**

Text **MSDS** to **37607** once to join



Which one is same as @yearly?

0 0 * * 1
0 0 1 1 *
* * 1 1 *
0 * * * *



Which one is same as @yearly?

0 0 * * 1
0 0 1 1 *
* * 1 1 *
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:
```

Executes a Python callable

Id of task

Python Function Name

```
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
```

<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
```

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

**Task 1 : Create a folder
(named yesterday's date in
YYYY_MM_DD format).**



**Task 2 : Download SF law
enforcement data and store in the
created folder**



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

```
$ airflow db init
```

Only for the first time

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
```

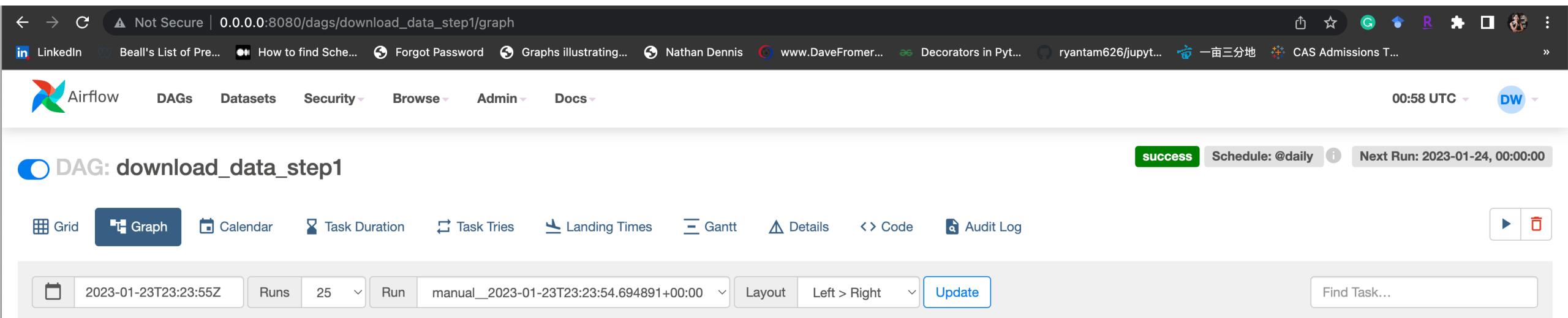
<https://airflow.apache.org/docs/apache-airflow/1.10.13/howto/initialize-database.html>



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



The screenshot shows the Airflow Webserver interface at the URL 0.0.0.0:8080/dags/download_data_step1/graph. The top navigation bar includes links for LinkedIn, Beall's List of Pre..., How to find Sche..., Forgot Password, Graphs illustrating..., Nathan Dennis, www.DaveFromer..., Decorators in Pyt..., ryantam626/jupy..., 一亩三分地, CAS Admissions T..., and a search bar. The header also displays the time as 00:58 UTC and a DW button.

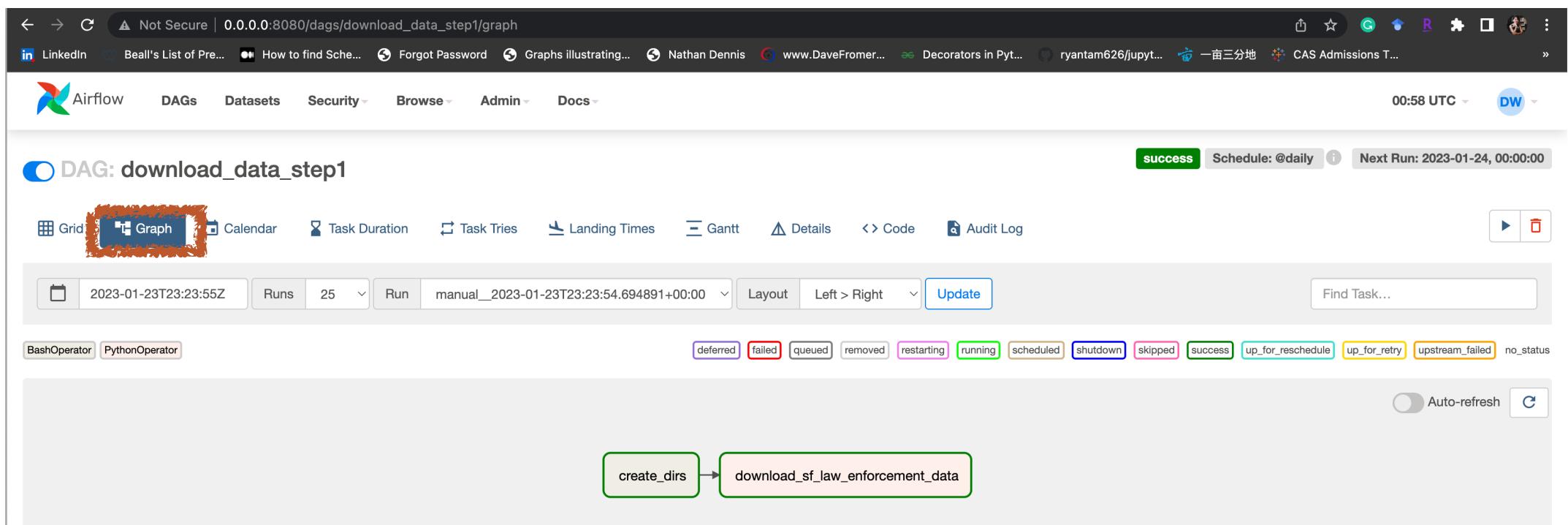
The main content area displays the DAG details for "download_data_step1". The DAG status is shown as "success" with a green button, "Schedule: @daily", and the next run scheduled for "2023-01-24, 00:00:00".

Below the status, there are several tabs: Grid, Graph (which is selected), Calendar, Task Duration, Task Tries, Landing Times, Gantt, Details, Code, and Audit Log. At the bottom of the screen, there are buttons for Date, Time, Runs (set to 25), Run, Layout, Left > Right, Update, and Find Task....

Airflow Webserver

DAG visualization

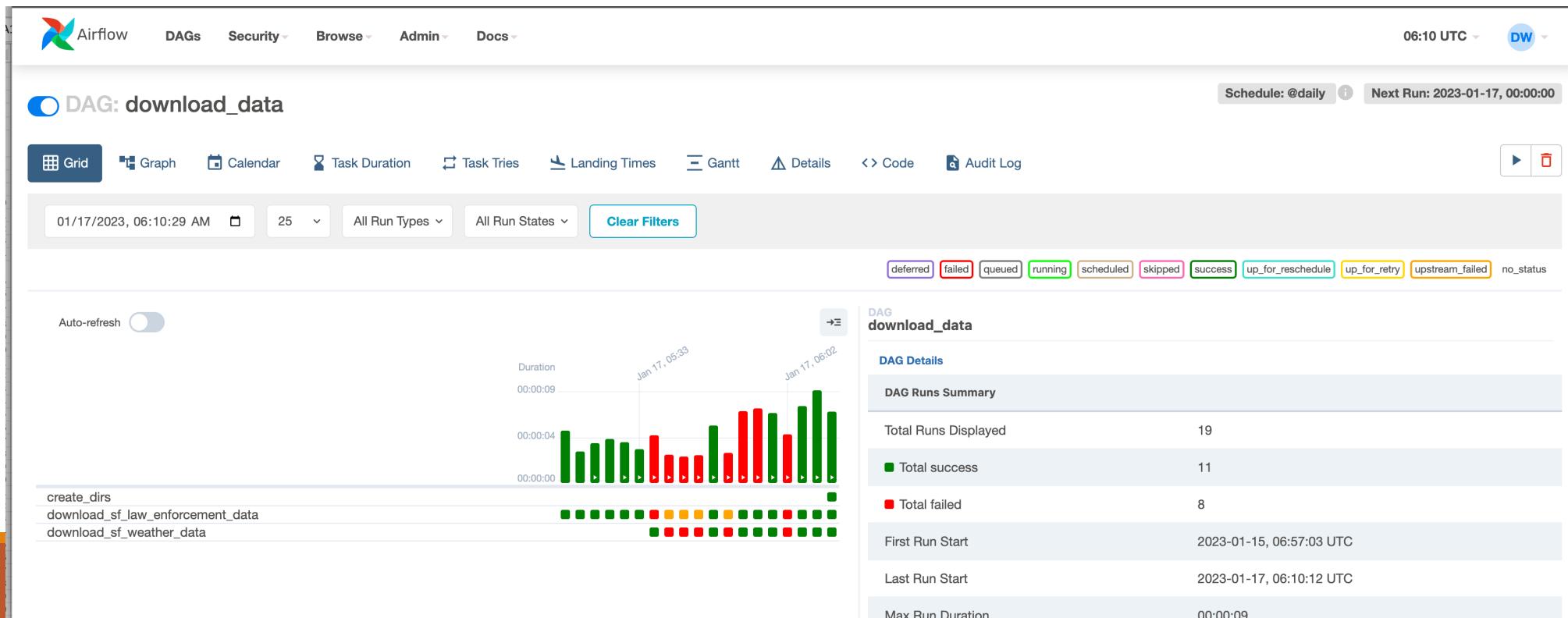
- `dag_id` ➔ Graph shows a dependency graph of tasks.



Airflow Webserver

Debugging

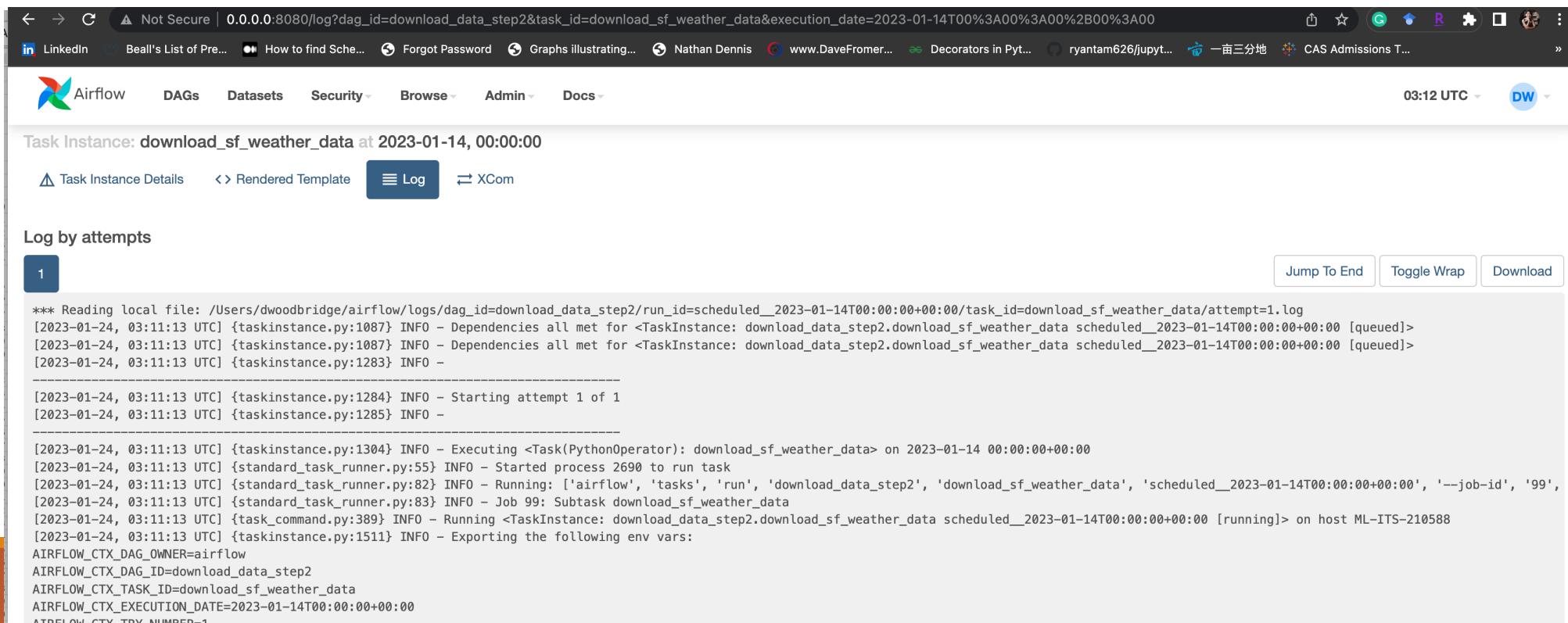
- If fails and needs to debug, go to “Details” ➔ Click “Failed” ➔ Click “task_id” for the failed DAG ➔ Click “Logs”



Airflow Webserver

Debugging

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



The screenshot shows the Airflow Webserver interface at the URL `0.0.0.0:8080/log?dag_id=download_data_step2&task_id=download_sf_weather_data&execution_date=2023-01-14T00%3A00%3A00%2B00%3A00`. The browser title bar indicates it's a Not Secure connection. The page header includes links for LinkedIn, Beall's List of Pre..., How to find Sche..., Forgot Password, Graphs illustrating..., Nathan Dennis, www.DaveFromer..., Decorators in Py..., ryantam626/jupyter..., 一亩三分地, CAS Admissions T..., and a search bar. The main navigation menu has items for Airflow, DAGs, Datasets, Security, Browse, Admin, and Docs. A timestamp of 03:12 UTC is shown. The main content area is titled "Task Instance: download_sf_weather_data at 2023-01-14, 00:00:00". Below this, there are tabs for Task Instance Details, Rendered Template, Log (which is selected), and XCom. The Log section is titled "Log by attempts" and shows attempt 1. It contains log entries from 2023-01-24 at 03:11:13 UTC, detailing the execution of the Python operator task. The log ends with environment variable exports for AIRFLOW_CTX_DAG_OWNER, AIRFLOW_CTX_DAG_ID, AIRFLOW_CTX_TASK_ID, AIRFLOW_CTX_EXECUTION_DATE, and AIRFLOW_CTX_TRY_NUMBER.

```
*** Reading local file: /Users/dwoodbridge/airflow/logs/dag_id=download_data_step2/run_id=scheduled__2023-01-14T00:00+00:00/task_id=download_sf_weather_data/attempt=1.log
[2023-01-24, 03:11:13 UTC] {taskinstance.py:1087} INFO - Dependencies all met for <TaskInstance: download_data_step2.download_sf_weather_data scheduled__2023-01-14T00:00+00:00 [queued]>
[2023-01-24, 03:11:13 UTC] {taskinstance.py:1087} INFO - Dependencies all met for <TaskInstance: download_data_step2.download_sf_weather_data scheduled__2023-01-14T00:00+00:00 [queued]>
[2023-01-24, 03:11:13 UTC] {taskinstance.py:1283} INFO -
[2023-01-24, 03:11:13 UTC] {taskinstance.py:1284} INFO - Starting attempt 1 of 1
[2023-01-24, 03:11:13 UTC] {taskinstance.py:1285} INFO -
[2023-01-24, 03:11:13 UTC] {taskinstance.py:1304} INFO - Executing <Task(PythonOperator): download_sf_weather_data> on 2023-01-14 00:00:00+00:00
[2023-01-24, 03:11:13 UTC] {standard_task_runner.py:55} INFO - Started process 2690 to run task
[2023-01-24, 03:11:13 UTC] {standard_task_runner.py:82} INFO - Running: ['airflow', 'tasks', 'run', 'download_data_step2', 'download_sf_weather_data', 'scheduled__2023-01-14T00:00+00:00', '--job-id', '99',
[2023-01-24, 03:11:13 UTC] {standard_task_runner.py:83} INFO - Job 99: Subtask download_sf_weather_data
[2023-01-24, 03:11:13 UTC] {task_command.py:389} INFO - Running <TaskInstance: download_data_step2.download_sf_weather_data scheduled__2023-01-14T00:00+00:00 [running]> on host ML-ITS-210588
[2023-01-24, 03:11:13 UTC] {taskinstance.py:1511} INFO - Exporting the following env vars:
AIRFLOW_CTX_DAG_OWNER=airflow
AIRFLOW_CTX_DAG_ID=download_data_step2
AIRFLOW_CTX_TASK_ID=download_sf_weather_data
AIRFLOW_CTX_EXECUTION_DATE=2023-01-14T00:00+00:00
AIRFLOW_CTX_TRY_NUMBER=1
```

Example 2

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

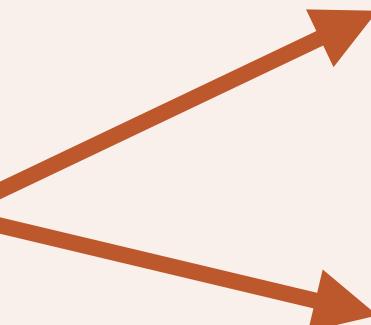
- Task 1 (create_dirs) : create {data_dir}/{yesterday} {data_dir}/{three_days_ago} if not exists.
- Task 2 (download_sf_law_enforcement_data) : calls the download_sf_law_enforcement_data function with keyword arguments.
- Task 3 (download_sf_weather_data) : calls the download_sf_weather_data function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

Execute it daily

Task 1 : Create a folder
(named yesterday's date in
YYYY_MM_DD format).

Task 2 : Download SF law
enforcement data and store in the
created folder

Task 3 : Download SF weather data
and store in the created folder



When poll is active, respond at **pollev.com/msds**

Text **MSDS** to **37607** once to join



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

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

Contents

Course Overview

Workflow Management

Apache Airflow

Airflow DAG Creation



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.

