Slide 1: Welcome!
Slide 2: My intro
Slide 3: Notes on the session
Slide 4: Code of Conduct
Slide 5: Session Goal
Slide 6: Agenda

Slide 8: In a study held by enlyft it was found that: There are actually 866 companies that use Apache Airflow. The companies using Apache Airflow are most often found in United States and in the Computer Software industry. Apache Airflow is most often used by companies with 50-200 employees and >1000M dollars in revenue. Our data for Apache Airflow usage goes back as far as 1 years and 3 months.

Slide 9 History:
In 2015, Airbnb experienced a problem. They were growing like crazy and had a massive amount of data that was only getting larger. To achieve the vision of becoming a fully data-driven organization, they had to grow their workforce of data engineers, data scientists, and analysts — all of whom had to regularly automate processes by writing scheduled batch jobs. To satisfy the need for a robust scheduling tool, Maxime Beauchemin created and open-sourced Airflow with the idea that it would allow them to quickly author, iterate on, and monitor their batch data pipelines.
Since Maxime's first commit way back then, Airflow has come a long way. The project joined the official Apache Foundation Incubator in April of 2016, where it lived and grew until it graduated as a top-level project on January 8th, 2019. Almost two years later, as of December 2020, Airflow has over 1,400 contributors, 11,230 commits, and 19,800 stars on Github. On December 17th 2020, Airflow 2.0 was released, bringing with it major upgrades and powerful new features. Airflow is used by thousands of Data Engineering teams around the world and continues to be adopted as the community grows stronger.

Slide 12 Overview:
Apache Airflow is a platform for programmatically authoring, scheduling, and monitoring workflows. It is completely open source and is especially useful in architecting and orchestrating complex data pipelines. Airflow was originally created to solve the issues that come with long-running cron tasks and hefty scripts, but it's since grown to become one of the most powerful open source data pipeline platforms out there.
Airflow has a couple of key benefits, namely:
* It's dynamic: Anything you can do in Python, you can do in Airflow.
* It's extensible: Airflow has readily available plugins for interacting with most common external systems. You can also create your own plugins as needed.
* It's scalable: Teams use Airflow to run thousands of different tasks per day.
With Airflow, workflows are architected and expressed as Directed Acyclic Graphs (DAGs), with each node of the DAG representing a specific task. Airflow is designed with the belief that all

data pipelines are best expressed as code, and as such is a code-first platform where you can quickly iterate on workflows. This code-first design philosophy provides a degree of extensibility that other pipeline tools can't match.

Slide 13 Video Airflow:

Slide 14 Data Pipeline:
A data pipeline is a series of data processing steps. If the data is not currently loaded into the data platform, then it is ingested at the beginning of the pipeline. Then there are a series of steps in which each step delivers an output that is the input to the next step. This continues until the pipeline is complete. In some cases, independent steps may be run in parallel.
Data pipelines consist of three key elements: a source, a processing step or steps, and a destination. In some data pipelines, the destination may be called a sink. Data pipelines enable the flow of data from an application to a data warehouse, from a data lake to an analytics database, or into a payment processing system, for example. Data pipelines also may have the same source and sink, such that the pipeline is purely about modifying the data set. Any time data is processed between point A and point B (or points B, C, and D), there is a data pipeline between those points.

Slide 17: Video Core Concepts

Slide 18 DAG:
In Airflow, pipelines are directed acyclic graphs (DAGs)
Mathematical Background
Directed Graph: A directed graph is any graph where the vertices and edges have some sort of order or direction associated with them.
Directed Acyclic Graph: Finally, a directed acyclic graph is a directed graph without any cycles. A cycle is just a series of vertices that connect back to each other in a closed chain.
In Airflow, each node in a DAG (soon to be known as a task) represents some form of data processing:
Node A could be the code for pulling data out of an API.
Node B could be the code for anonymizing the data and dropping any IP address.
Node D could be the code for checking that no duplicate record ids exist.
Node E could be putting that data into a database.
Node F could be running a SQL query on the new tables to update a dashboard.
Each of the vertices have a specific direction showing the relationship between nodes - data can only follow the direction of the vertices (from the example above, the IP addresses cannot be anonymized until the data has been pulled).
Node B is downstream from Node A - it won't execute until Node A finishes.

Workflows, particularly around those processing data, have to have a point of "completion." This especially holds true in batch architectures to be able to say that a certain "batch" ran successfully.

Slide 19:
A Directed Acyclic Graph, or DAG, is a data pipeline defined in Python code. Each DAG represents a collection of tasks you want to run and is organized to show relationships between tasks in Airflow's UI. When breaking down the properties of DAGs, their usefulness becomes clear:
* Directed: If multiple tasks with dependencies exist, each must have at least one defined upstream or downstream task.
* Acyclic: Tasks are not allowed to create data that goes on to self-reference. This is to avoid creating infinite loops.
* Graph: All tasks are laid out in a clear structure with processes occurring at clear points with set relationships to other tasks.
For example, the image below shows a valid DAG on the left with a couple of simple dependencies, in contrast to an invalid DAG on the right that is not acyclic.

Slide 20 Recap:
DAGs are a natural fit for batch architecture - they allow you to model natural dependencies that come up in data processing without and force you to architect your workflow with a sense of "completion."
Directed - If multiple tasks exist, each must have at least one defined upstream (previous) or downstream (subsequent) tasks, although they could easily have both.
Acyclic - No task can create data that goes on to reference itself. This could cause an infinite loop that would be, um, it'd be bad. Don't do that.
Graph - All tasks are laid out in a clear structure with discrete processes occurring at set points and clear relationships made to other tasks.

Slide 21 Tasks:
Tasks represent each node of a defined DAG. They are visual representations of the work being done at each step of the workflow, with the actual work that they represent being defined by operators.

A Task is the basic unit of execution in Airflow. Tasks are arranged into DAGs, and then have upstream and downstream dependencies set between them into order to express the order they should run in.
There are three basic kinds of Task:
* Operators, predefined task templates that you can string together quickly to build most parts of your DAGs.
* Sensors, a special subclass of Operators which are entirely about waiting for an external event to happen.
* A TaskFlow-decorated @task, which is a custom Python function packaged up as a Task.
Internally, these are all actually subclasses of Airflow's BaseOperator, and the concepts of Task and Operator are somewhat interchangeable, but it's useful to think of them as separate

concepts - essentially, Operators and Sensors are templates, and when you call one in a DAG file, you're making a Task.

Like in POO when instantiating a class,

Slide 22 Operators:
Operators are the building blocks of Airflow, and determine the actual work that gets done. They can be thought of as a wrapper around a single task, or node of a DAG, that defines how that task will be run. DAGs make sure that operators get scheduled and run in a certain order, while operators define the work that must be done at each step of the process.
There are three main categories of operators:
* Action Operators execute a function, like the PythonOperator or BashOperator
* Transfer Operators move data from a source to a destination, like the S3ToRedshiftOperator
* Sensor Operators wait for something to happen, like the ExternalTaskSensor
Operators are defined individually, but they can pass information to other operators using XComs.

Slide 24: At a high level, the combined system of DAGs, operators, and tasks looks like this:

Slide 25: Hooks
Hooks are Airflow's way of interfacing with third-party systems. They allow you to connect to external APIs and databases like Hive, S3, GCS, MySQL, Postgres, etc. They act as building blocks for operators. Secure information such as authentication credentials are kept out of hooks - that information is stored via Airflow connections in the encrypted metadata db that lives under your Airflow instance.

Slide 26 Recap:
Explain with your own words

Links:
https://www.youtube.com/watch?v=Axm35hOUzyQ
https://www.astronomer.io/guides/dags
https://github.com/jghoman/awesome-apache-airflow#best-practices-lessons-learned-and-cool-use-cases
https://github.com/astronomer/astro-cli
https://www.astronomer.io/guides/intro-to-airflow
https://livebook.manning.com/book/data-pipelines-with-apache-airflow/chapter-5/v-5/1
https://medium.com/hashmapinc/orchestration-and-dag-design-in-apache-airflow-two-approaches-35edd3eaf7c0
https://enlyft.com/tech/products/apache-airflow
https://airflow.apache.org/docs/apache-airflow/stable/concepts/tasks.html
https://airflow.apache.org/docs/apache-airflow-providers/index.html