Orchestrating workflows is a 'cron-ic' systems problem. Airflow is the modern solution.

A rapid introduction and tour of Airflow, the open-source task and workflow orchestration system.

Jack Bennett, PhD - CodeMash 2023 - 2023-01-12

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

Orchestrating workflows is a 'cron-ic' systems problem. Airflow is the modern solution.

A rapid introduction and tour of Airflow, the open-source task and workflow orchestration system.

Jack Bennett, PhD - CodeMash 2023 - 2023-01-12

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

Monday, 8:28 am.
No meetings today.
Ready to concentrate ALL YOUR ATTENTION on `Project X`.
(The company's #1 top priority.)

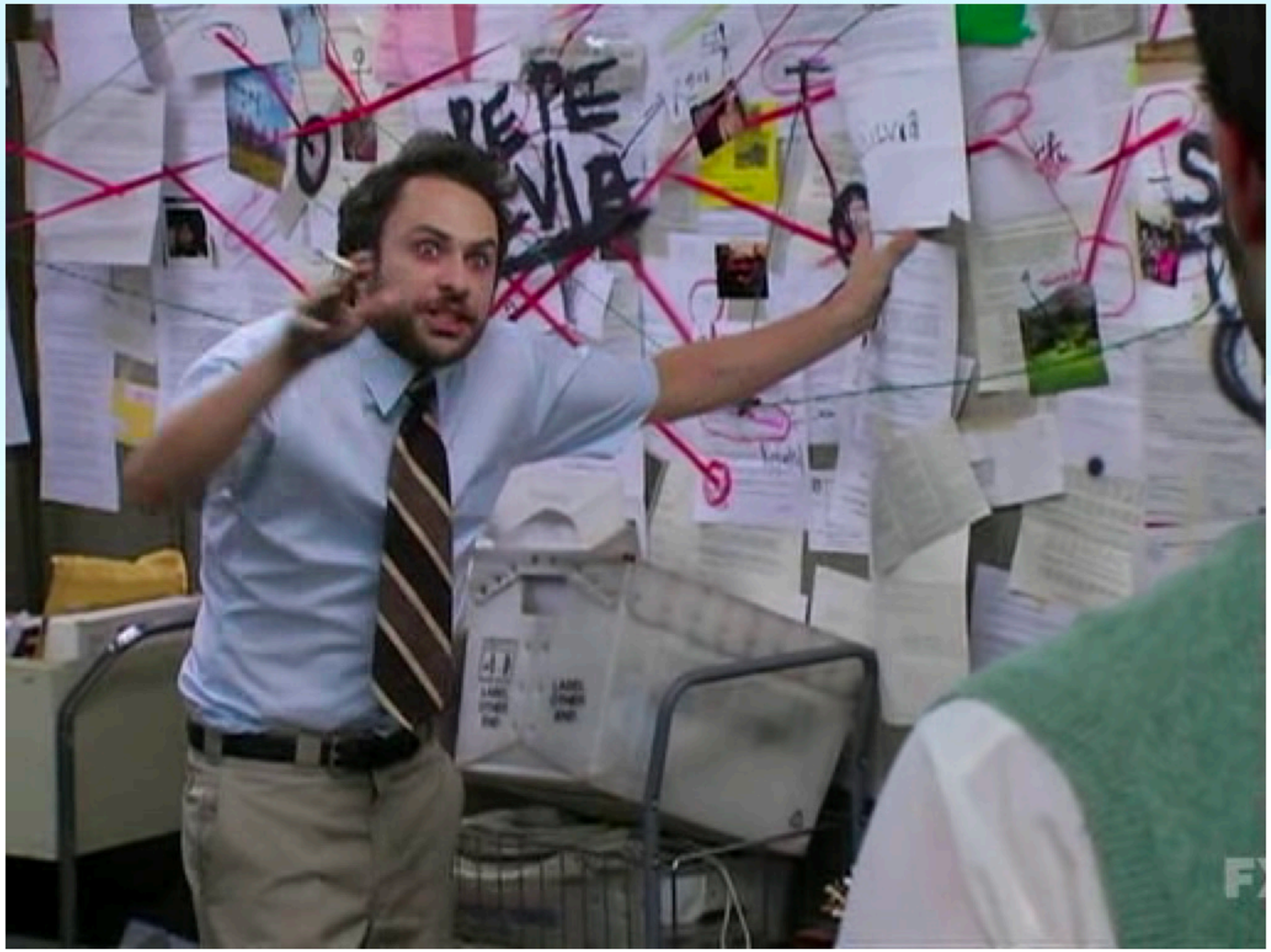
🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

“OK, I got two minutes before my next meeting. Bob told me we gotta to set up Air-something. So why should we use this Airflux? Airfly? I need you to get me up to speed on Airflow, OK bud? Get back to me before EOD. <*click*>”

-Your CTO, Monday 8:28 am, unexpected Zoom call

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

🧘‍♂️ ➡️



🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

Introduction

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

Over the course of this talk you will...

- Go from 0% to 100% in Airflow skill.
- Become a world-class Airflow expert.
- Learn all of the Python language if you don't know it already.
- Learn to write plugins, operators, DAGs, and other Airflow component code.
- Develop the ability to contribute to the Airflow open source project.
- Develop the ability to execute custom Airflow implementations for big 💰 for enterprise clients

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

The REAL goals of this talk ➡️🌱🌱

By the end of this talk you will be able to…

- Explain what Apache Airflow is and what it does.
- Compare and contrast Airflow with earlier, related tools (mostly `cron`)
- Describe some scenarios where you might want to use Airflow.
- Experiment with basic operation of the UI, features, and code.
- Understand Airflow's conceptual foundation and application architecture.
- Know where to find resources for learning more
- Have some fun 🎉😀 and plant some seeds 🌱🌱 of interest in Airflow

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

What is this talk *not*?

- NOT a deep dive into all the features, internal operation, and technical details of Airflow (impossible in only 1h).
- NOT a deep dive into source code … though we will step through some operational Python code.
- NOT a detailed review of how to run Airflow in production - CI/CD, testing, deployment, Docker, Kubernetes, managed Airflow solutions, etc.

⚠️ Warning ⚠️
May contain image memes, movie quotes, and terrible dad jokes.

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

What is Airflow?

Apache Airflow is an open-source platform for developing, scheduling, and monitoring batch-oriented workflows.

via https://airflow.apache.org/docs/apache-airflow/stable/index.html

Sounds good. What’s a workflow? 🤔

A Workflow is a set of related Tasks that we want to run automatically, in a specified order, on one or more hosts.

Sounds good. What’s a task? 🤔

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

A Task is a specific computational job.
e.g. get file from S3, export data from a database table, process a dataframe in `pandas`

Related Tasks: these Tasks have dependencies on each other, and/or on external conditions.

What's a dependency?

Answer: `$THING_1` has to happen before `$THING_2` happens.

- **Internal to Airflow**
  - An upstream task finishes running. ⬇️
- **External to Airflow**
  - *Data:* A file arrives in an SFTP directory or an S3 bucket. 📝
  - *Availability:* A downed service or API becomes available. ⛓️
  - *Server Load:* A database completes a CPU-intensive maintenance task. 💪
  - *Time:* The clock strikes midnight. ⏰

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

“That’s dumb. Don’t make it complicated. Just run a shell script from cron…”

-Hypothetical cranky sysadmin from 1998 😆

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

What is `cron`?

- Cron's name originates from **χρόνος** (*chronos)*, the Greek word for time.
- It is an original Unix utility for job scheduling.
- “Repeatedly execute **`$COMMAND(S)`** at **`$DATETIME_INTERVAL(S)`**”

The Cron command-line utility is a job scheduler on Unix-like operating systems.

Users who set up and maintain software environments use cron to schedule jobs (commands or shell scripts), also known as cron jobs, to run periodically at fixed times, dates, or intervals.

It typically automates system maintenance or administration—though its general-purpose nature makes it useful for things like downloading files from the Internet and downloading email at regular intervals.

Source: https://en.wikipedia.org/wiki/Cron

How does `cron` work?

- “crontab" (cron table) config files tell the host **what** to run **when**.
- Let's learn **`cron`** syntax! https://cron.help/
- Many later tools (including Airflow) adopted the same syntax to describe a repeating schedule.

Why is Airflow better[*] than cron?

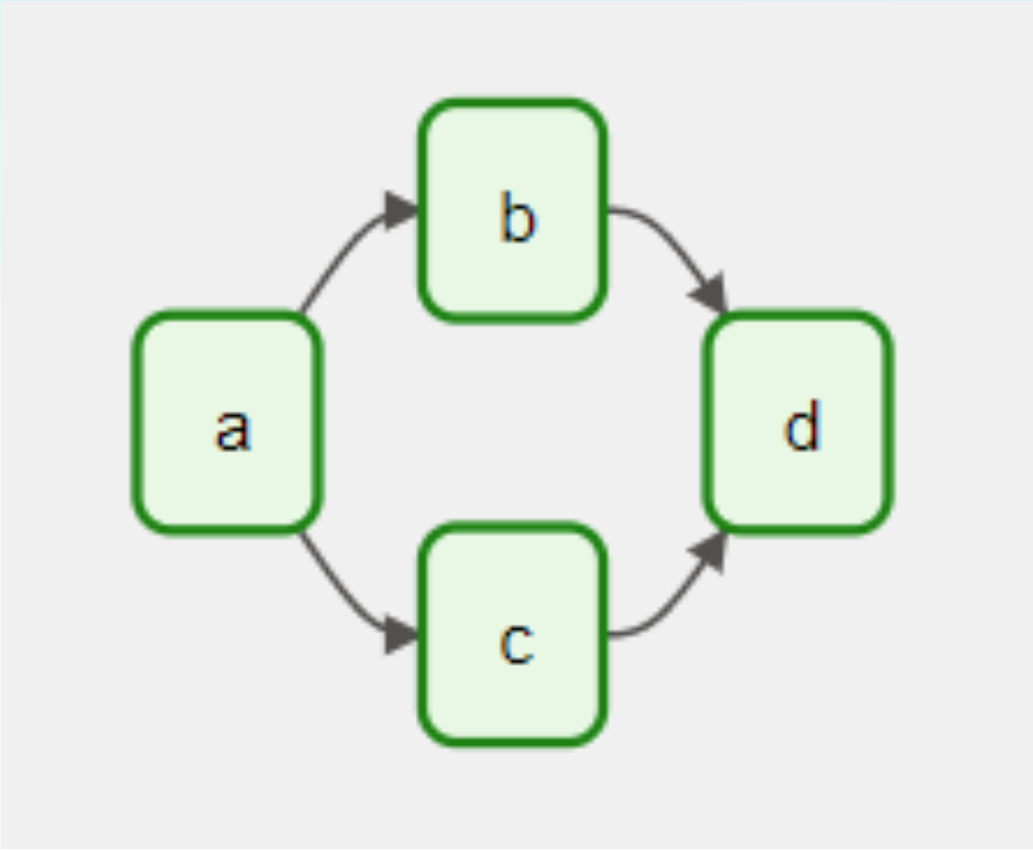
[*] “for some values of ‘better’” / “better for what?”

cron	Airflow
single host 1️⃣	multi-host or cloud 🔢
wall-clock time ⏰	complex dependencies 🧲
command-line terminal world 📄	web UI world 📲
run one hard-coded (literal) shell command 1️⃣	rich task management, execution, observation, etc⚙️
simple config file executes commands on schedule 📁	programmable in Python 🐍
no built-in alerts, things fail quietly 🤫	integrates with alerts and observability 👀
1975 📅	2014 📅
“run independent tasks in series, on fixed schedule”	“run complex workflow of interdependent tasks”

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

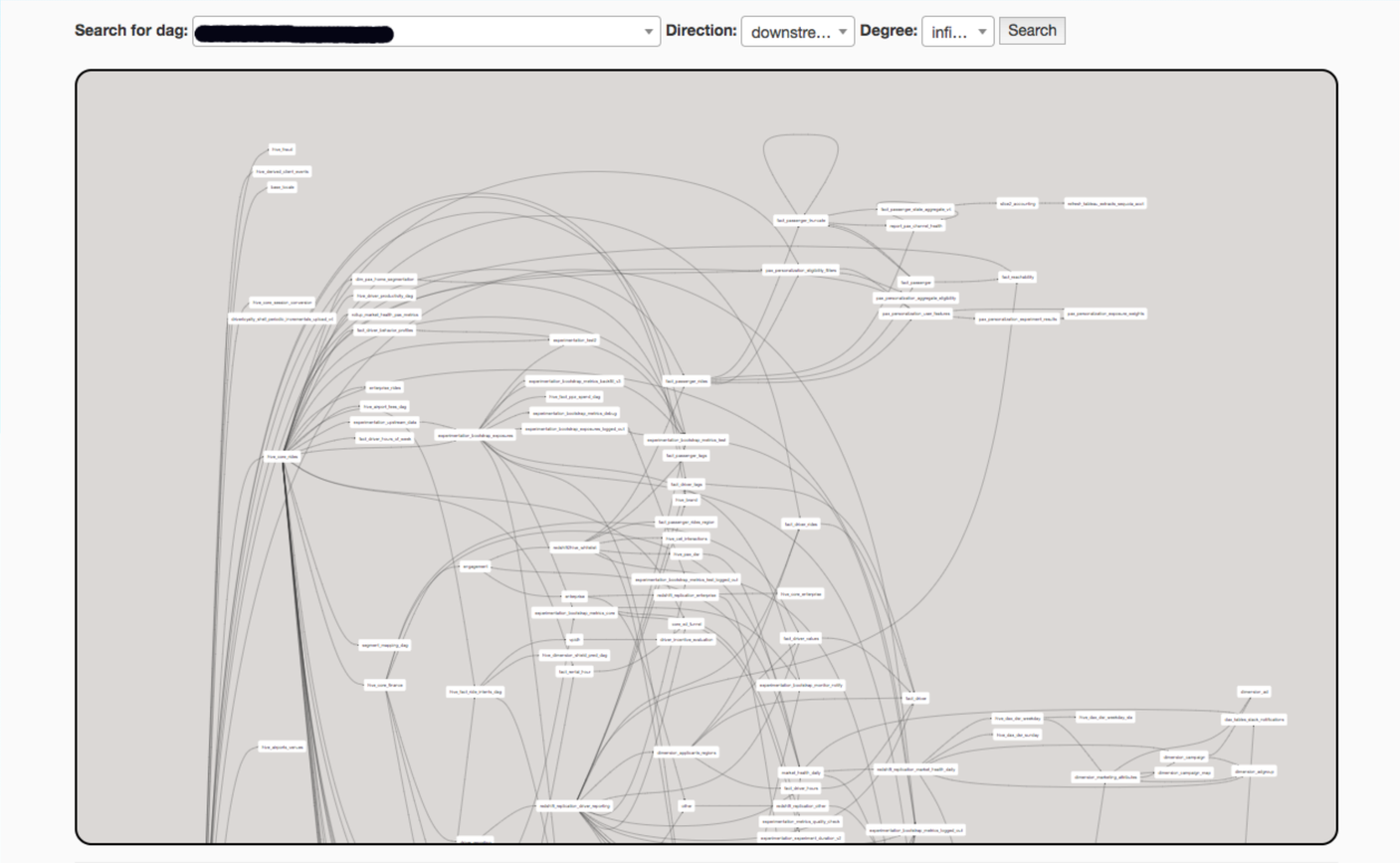
Well, you *could* run this simple workflow from cron…

- Start all the jobs from cron
- Create $LOCKFILE_A when job A starts.
- Job A deletes $LOCKFILE_A right before it exits.
- Job B and job C wait while periodically polling the file system to see if $LOCKFILE_A is gone.
- When jobs B and C detect that $LOCKFILE_A is gone, they create $LOCKFILE_B and $LOCKFILE_C, respectively, and start their actual work.
- Job D waits while periodically polling the file system to see if both $LOCKFILE_B and $LOCKFILE_C are gone.
- When job D detects that both $LOCKFILE_B and $LOCKFILE_C are gone, it can start working.



That was easy … now do this one. 😂😂

Would you like to handle this workflow with cron and lock files?



Source https://eng.lyft.com/running-apache-airflow-at-lyft-6e53bb8fccff (2018)

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

⚠️Disclaimer⚠️

We are not beating up on cron and praising Airflow

- **Both of these tools**
  - **Do the jobs that their programmers made them to do.**
  - **Do these jobs well.**
- We must remember:
  - cron (1975) ~4,800 lines of C
  - Airflow (2014) ~620,000 lines of Python
- Created at very different times and in radically different worlds.

`cron` vs Airflow

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

Summary: `cron` vs Airflow

- `cron` is best for running
  - **independent tasks**, each launched from a command line
  - on a **single host**
  - on a **predictable and manually-defined schedule** (defined in crontab file)
- **Airflow** is best for running
  - **complex workflows**
  - in a **multi-host** or **cloud** environment
  - with **complex dependencies** on **variable conditions** that are both **internal and external to the workflow** itself

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

What is Airflow *not*?

What is Airflow not?	Why not?
NOT a high-volume streaming platform and/or datastore like Kafka	“Big Data” does not flow through Airflow itself.
NOT a complex event processing (CEP) system like Storm	Nodes launch jobs that perform business logic. They don’t directly perform the logic themselves (mostly).
NOT a low-latency / real-time platform.	Operational timescales are typically minutes, hours, days.
NOT for running infinite-lifetime jobs (servers, etc).	Every task and every workflow has a definite start and end.

Other tools for task orchestration

DAGs: the conceptual model at the heart of Airflow

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

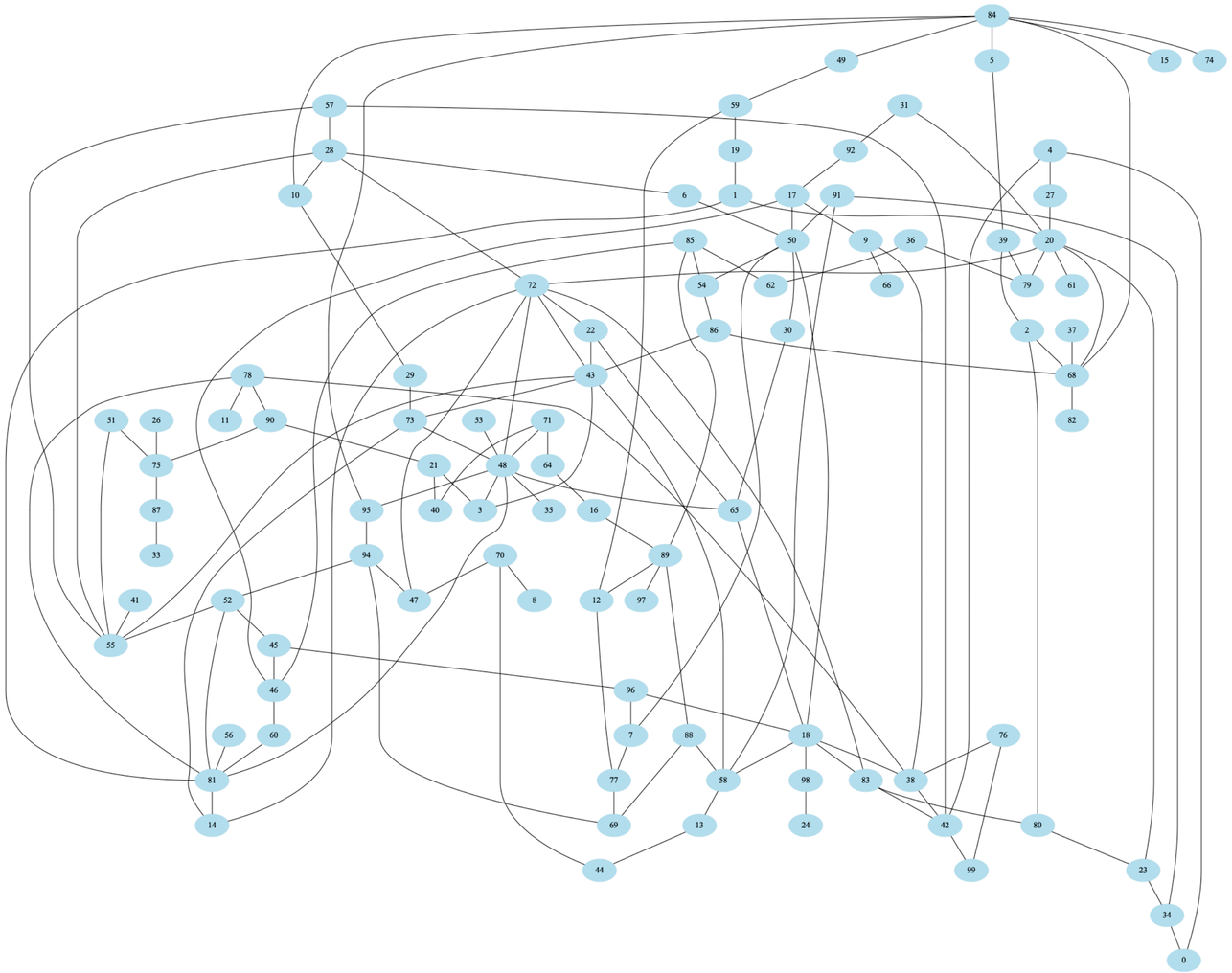
DAGS, D'YA LIKE DAGS?

DAG != 🐶 (sorry, Mickey)

For us Airflow users:
DAG == “**D**irected **A**cyclic **G**raph”

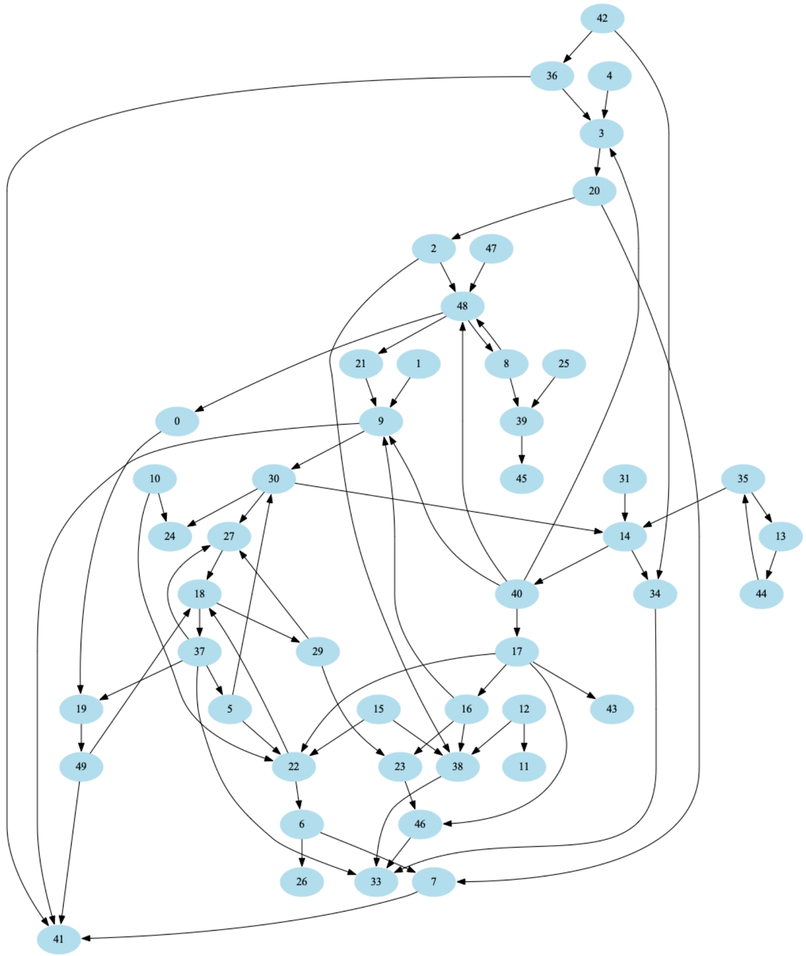
“DAGs. D’ya like DAGs?” When you use Airflow, you learn to love the Directed Acyclic Graph.

What is a GRAPH?



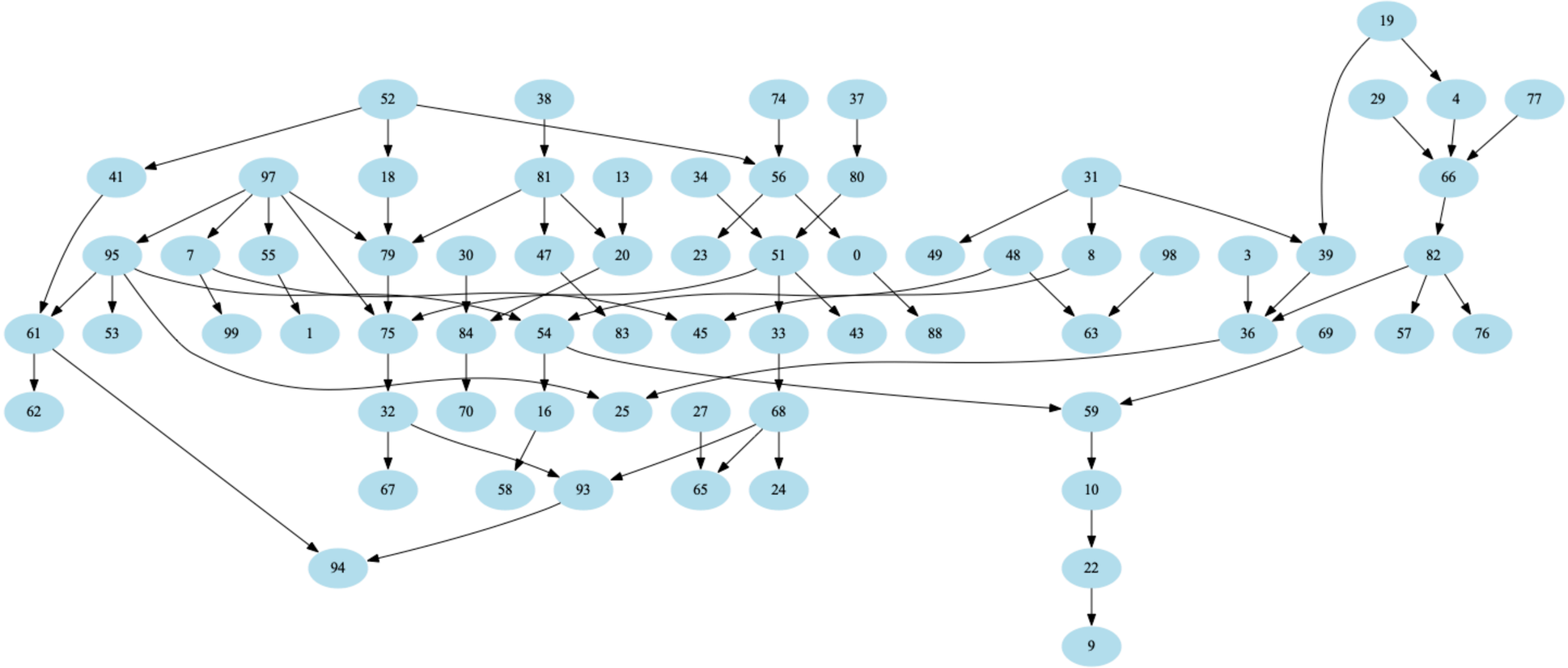
🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

What is a DIRECTED GRAPH?



🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

What is a DIRECTED ACYCLIC GRAPH?

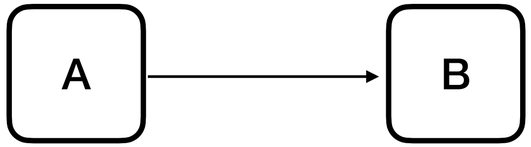


🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

Why does Airflow model workflows as DAGs?

Why DIRECTED?

- Arrow direction ➡️ movement of time and program activity.
- Task **DEPENDENCY** needs a “directed” workflow graph: dependency (**UPSTREAM**) nodes point at dependent (**DOWNSTREAM**) nodes.

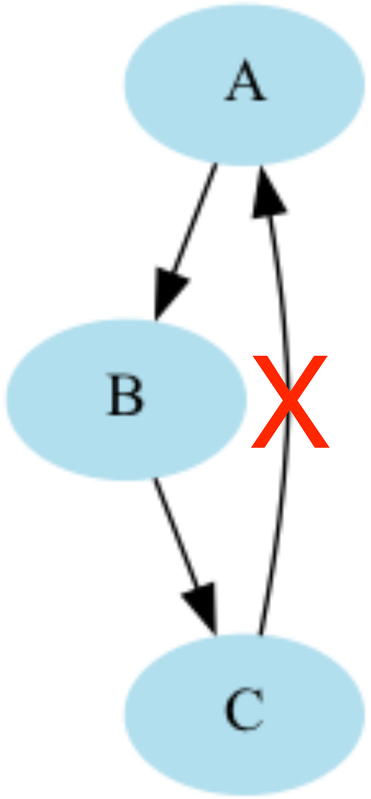


A is **UPSTREAM** of B
A is a **DEPENDENCY** of B

B is **DOWNSTREAM** of A
B is **DEPENDENT** on A

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

Why ACYCLIC?



⚠️Upstream/Downstream vs Next/Previous⚠️

Getting the terminology clear

- **Upstream** and **Downstream:**
  - Suppose Task B depends on Task A. Task A is **upstream** from Task B, and Task B is **downstream** from Task A.
  - The arrow in the DAG points from Task A ➡️ Task B: **upstream** to **downstream**.
  - Time and the active task(s) flow from **upstream** to **downstream**.
- **Next** and **Previous**:
  - These terms are used only to refer separate runs of the same DAG or the same Task.

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

Summary: Why DAGs?

- Workflow **GRAPH** shows individual **TASKS** (nodes) and **RELATIONSHIPS** (edges).
- Workflow graph must be **DIRECTED** because of
  - Task dependency ⬇️
  - Time ordering ⏰
- Workflow graph must be **ACYCLIC** because
  - Cycle ♻️ == no independent starting point. A cyclic workflow can never start.

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

Airflow Demo: Code, UI, API

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

Magnets, how do they work?

🔗 📄

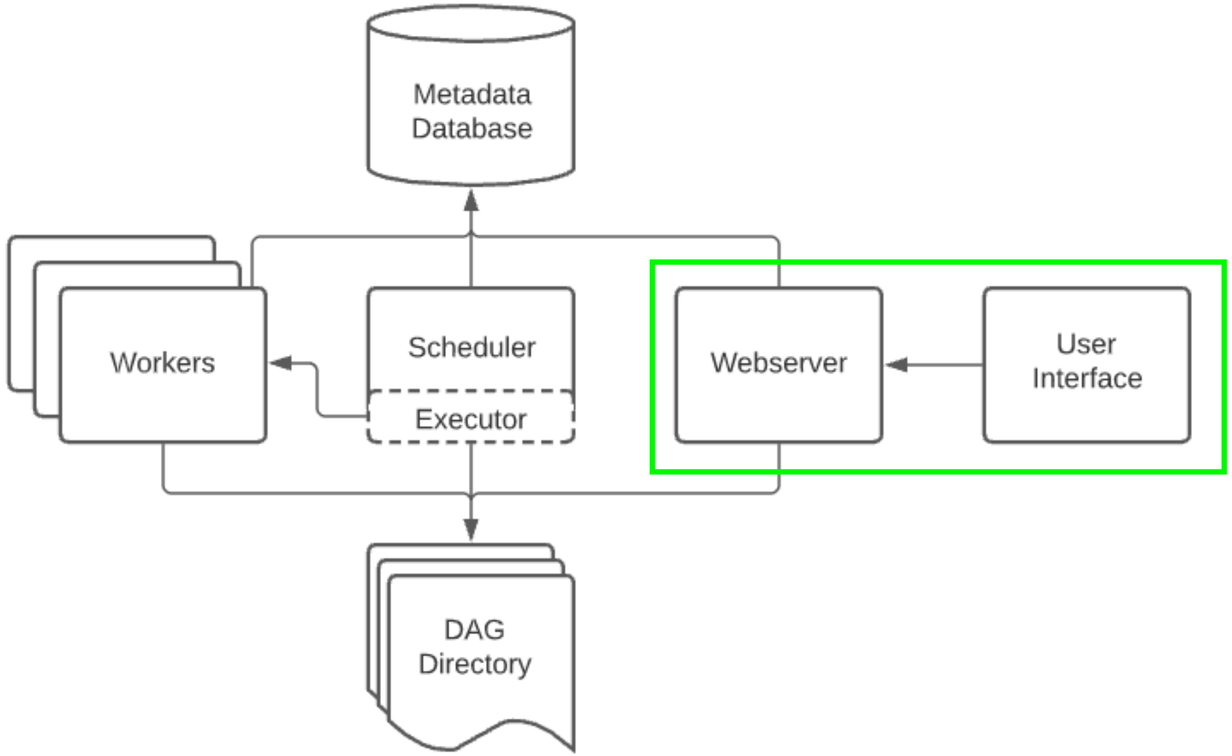
Demonstrating Airflow

- Launch the standalone demo version from terminal
- Demonstrate the web UI and some of its functionality
  - DAG organization and scheduling
  - DAG code and graphs - "*Workflows As Code*"
- Show some DAGs in live operation
- Look at the REST API (Swagger UI)

Airflow Architecture

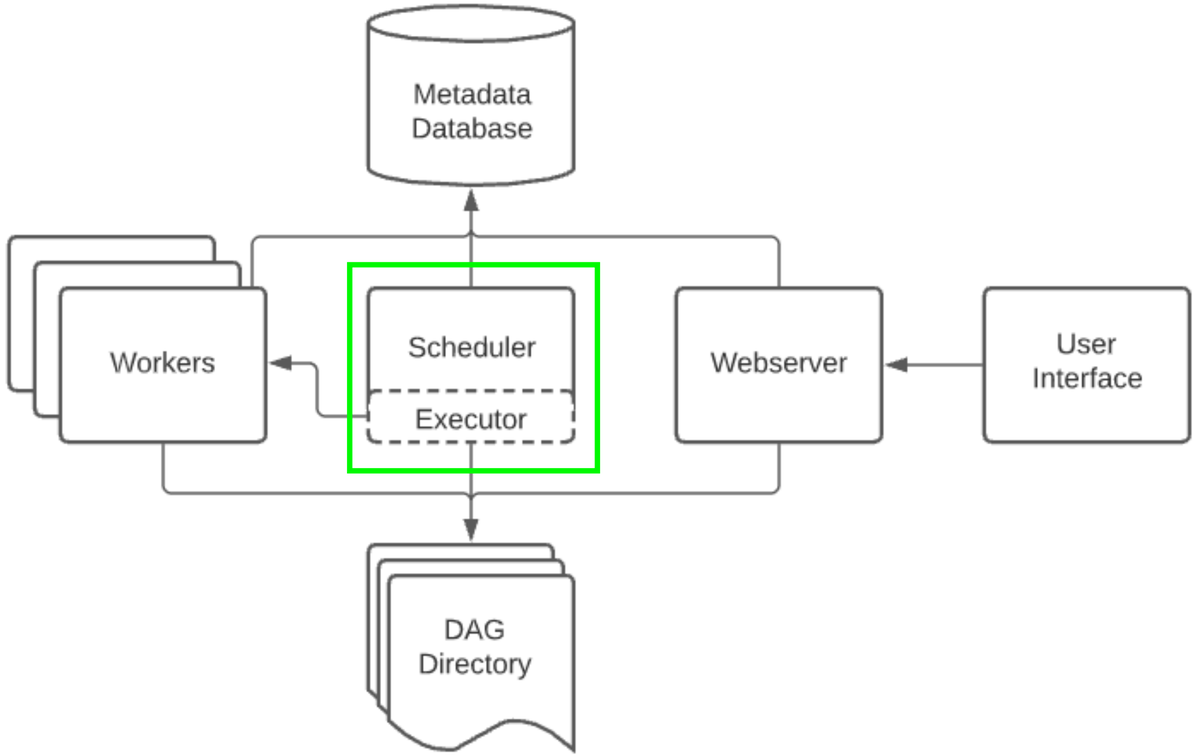
🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

Airflow Architecture: Webserver + UI

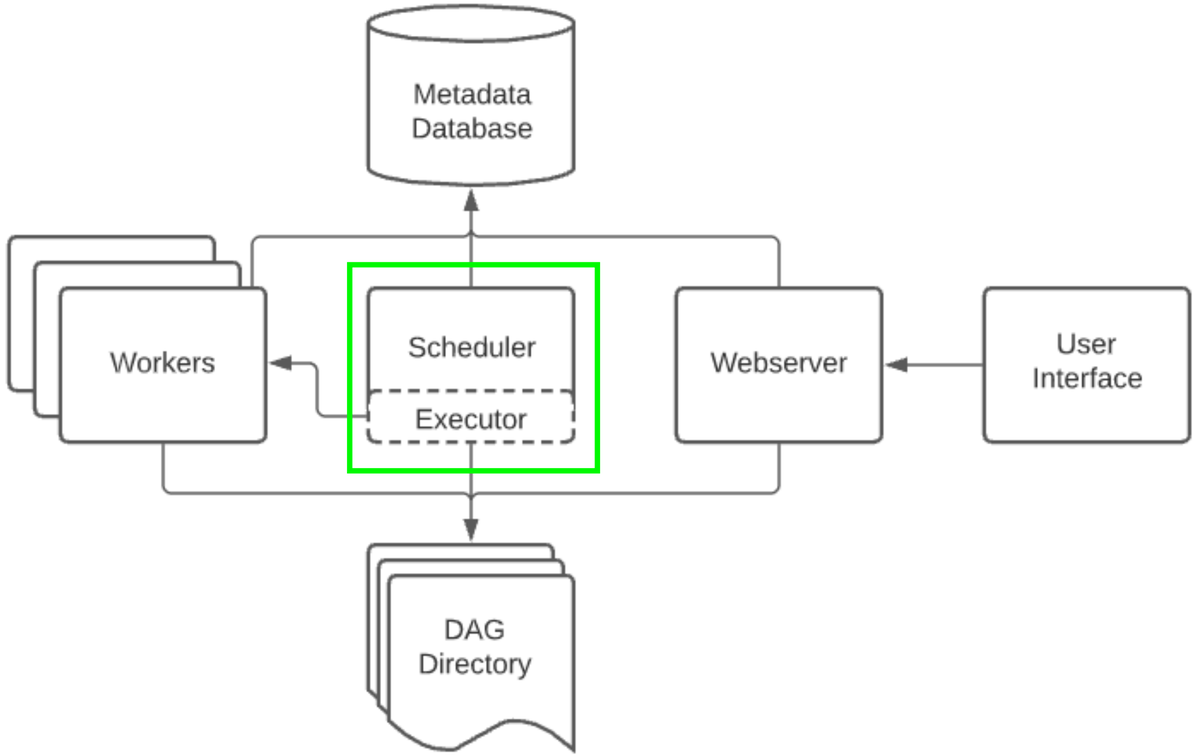


🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

Airflow Architecture: Scheduler

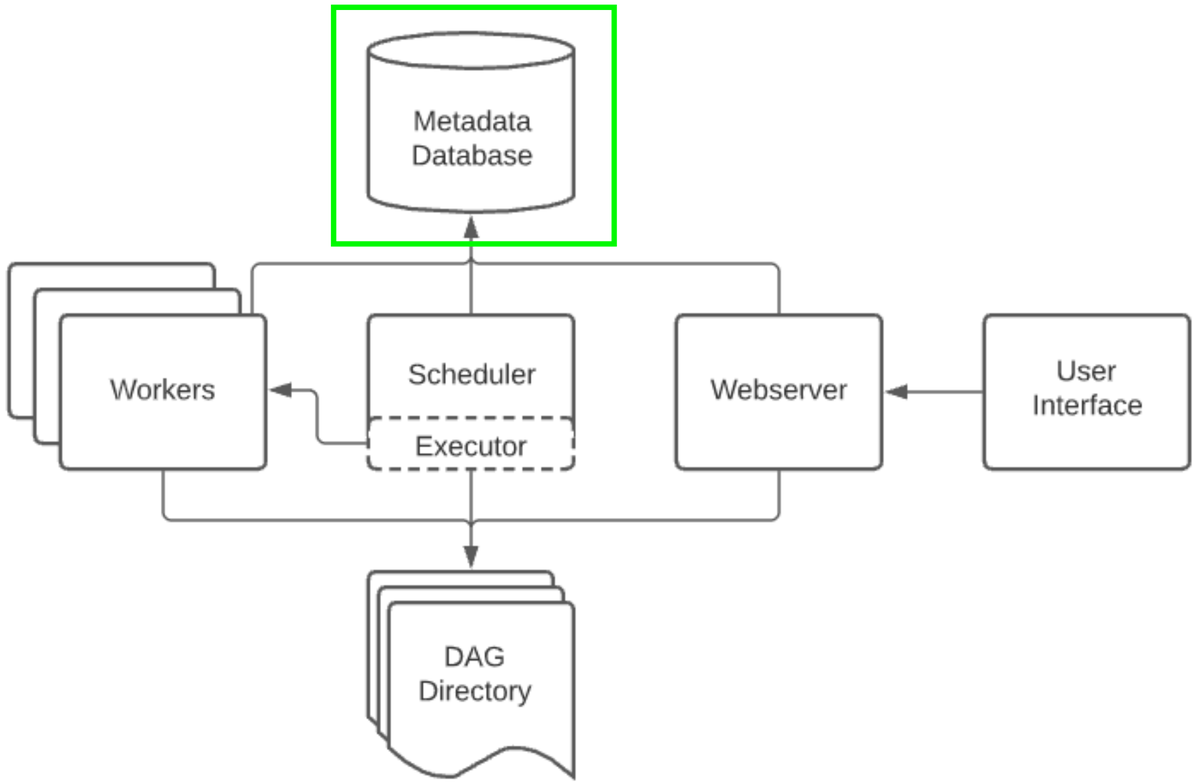


Airflow Architecture: Executor



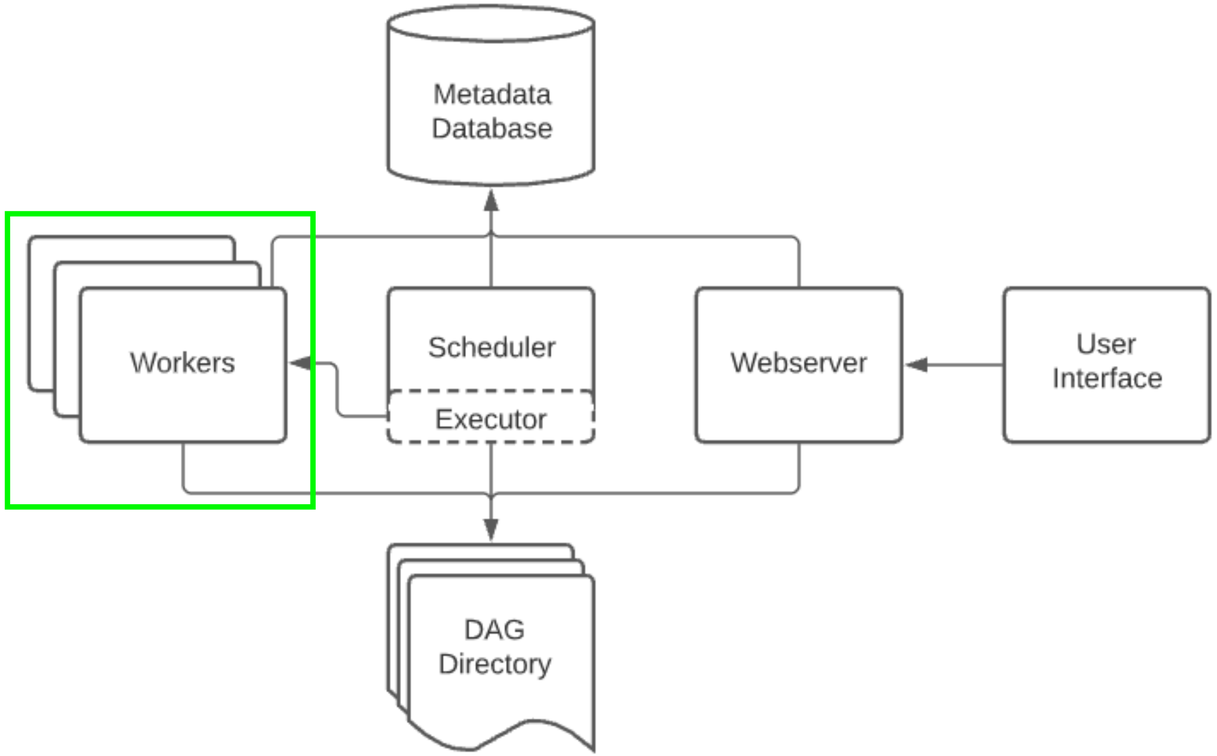
🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

Airflow Architecture: Metadata Database



🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

Airflow Architecture: Workers



🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

Operators

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

Operators

- A class that represents a specific category of task.
- Think of it as **class = Operator**, **instance of the class = Task**
- Intended to save time, effort, and repetition for common Tasks. Where possible, don't write custom code, call an operator to define a Task.
- Many pre-defined operators already exist for our use.
  - **`BashOperator`** - executes a bash command
  - **`PythonOperator`** - calls an arbitrary Python function
  - **`EmailOperator`** - sends an email

Sensors

Sensors

- A specific category of Operator that polls some external resource and executes when that resource meets a certain condition.
- Examples:
  - Look for a file matching a pattern using **`S3KeySensor`**
  - Observe when a build completes in Jenkins using **`JenkinsBuildSensor`**.

Hooks

nett

Hooks

- **Hook:** A high-level abstract interface to an external platform, datastore, service, etc.
- Goal: use these external resources via a common interface and without using special SDKs or APIs.
- Hooks are generally used as building blocks for operators.
- Examples: **`SSHHook`**, **`SFTPHook`**, **`PostgresHook`**

Connections

Connections

- Abstraction that holds parameters for connecting to common external resources (file server, database, etc).
- Example parameters: username, password, hostname, protocol.
- Every connection has a unique identifier, the **`conn_id`**.
- Manage connections through the UI or CLI; see Managing Connections for more information on creating, editing and managing connections. There are customizable connection storage and backend options.
- Use Connections directly, through Hooks, or in templates

⚠️Data movement in Airflow⚠️

- Data flow within Airflow **itself** is relatively lightweight (Metadata DB, parameters passed in to workers, etc).
- The data that Airflow directly manages is all **task metadata**: task descriptions, schedules, state, etc.
- Airflow **Tasks** can and do perform heavy data lifting and moving. However, the high-volume data movement generally takes place within the apps/programs that get launched by Airflow. It does not take place within Airflow itself!

XCom (“cross-communications”)

- XCom is Airflow’s module for data communication directly between separate tasks.
- Remember, depending on the **Executor** you choose, Airflow worker tasks aren't necessarily on the same host.
- XCom operates as a key-value store. Writer task stores data using **`xcom_push`** and reader task retrieves data using **`xcom_pull`**.
- An XCom is uniquely identified by a **`key`**, **`task_id`** and **`dag_id`**
- If you use XCom at all, use it only to transfer **small** amounts of data.
- Alternative approach: if a shared datastore is already part of your infrastructure (Redis, Mongo, RDBMS, etc) ... you may want to use that instead.

If you try to move massive data sets within Airflow itself...

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

Extending and customizing Airflow's capabilities

Providers and Plugins

- **Provider:** An extension package that can contain operators, hooks, and sensors, to interface with external systems and applications, and/or to extend Airflow core capabilities.
- **Plugin:** A mechanism for enabling users to add capabilities and functionalities to a local Airflow installation (e.g. macros and web views) by placing custom code into **`$AIRFLOW_HOME/plugins`**

Architecture: who needs to know this?

- Architects and Airflow administrators will benefit from learning more of these details so that they can design/deploy/troubleshoot systems and allocate resources that play nicely with Airflow's architecture.
- Developers focusing on DAG implementations don't necessarily need to know all the underlying details (though it won't hurt).
- Analysts and other end-users can interact with Airflow via the UI.

Zooming out: The big picture

“Oh, yeah, all that other stuff.” -Developers

“I guess it’s important, right?”-Also developers

- Airflow’s technology ecosystem
- FOSS
- Providers and plugins
- Production deployment
- Documentation
- Business cases

“How to sell Airflow to your CTO.”

- Long, open-source, project development history.
- Active developer and user community.
- High-quality documentation, tutorials, examples.
- Abundant resources exist for getting dev teams up to speed.
- Several managed solutions are available
- **Conclusion: Airflow is ready for production use.**

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

Airflow in 2023: basic facts and stats

- Open source (Apache 2.0 license)
- GitHub stats (2023-01-10). These values are moving targets. 💨🎯
  - 622,406 lines of Python code
  - 18,529 commits
  - 28,700 stars
  - 11,800 forks
  - 2,324 committers
- Written primarily in Python (UI elements are in TS/JS/HTML)
- Started in 2014 by Maxime Beauchemin. Current version is 2.5.0 (2023-01-10)

```
commit 1047940ca4363b04044c4963b9c88f7632746407
Author: Maxime Beauchemin <maxime.beauchemin@apache.org>
Date:   Mon Oct 6 14:29:38 2014 -0700

    First official commit
```

Running in production 🐇🕳️

- On-prem vs DIY in cloud vs Managed in cloud
- “Usual **`$CURRENT_YEAR`** best-practice devops stuff.” (You're doing all this already, right?)
- DAGs are Python source code == version control your DAGs in **`git`**.
- Test, merge, build, deploy with your existing CI/CD methodology.
- Abundant Docker, Kubernetes, etc, approaches available.
- Managed Airflow instances on AWS, GCP, Astronomer.io.

Managed Airflow solutions

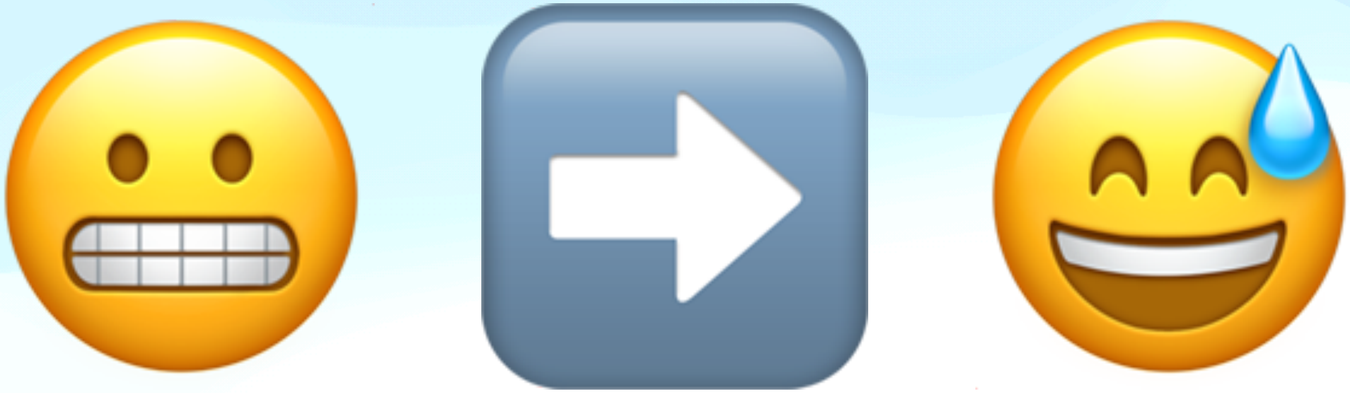
AWS, GCP and astronomer.io

- https://itsvit.com/blog/google-cloud-composer-vs-astronomer-what-to-choose/
- $300/mo for GCP vs $110/mo for astronomer (circa 2020)

[Redux]

“OK, I got two minutes before my next meeting. Bob told me we gotta to set up Air-something. So why should we use this Airflux? Airfly? I need you to get me up to speed on Airflow, OK bud? Get back to me before EOD. <*click*>”

-Your CTO, on an unexpected Zoom call



🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

Executive Summary

What is Airflow?

- Apache Airflow is **an open-source platform for developing, scheduling, and monitoring batch-oriented workflows.**

Executive Summary

What is our current situation and pain points?

- **We launch and run hundreds of individual computational batch jobs across the organization.**
- These are currently implemented in many different systems and platforms and in many different ways.
- Sometimes they are cron jobs, sometimes they are launched out of individual platforms like SQL Server or Windows Task Scheduler.
- We have **no unified view into what launches what or what depends on what.**

Executive Summary

What will Airflow do for us?

- Using Airflow will **save us thousands of hours of engineer and analyst time per year**, with the **associated cost saving**.
- The **integrated logging, job tracking, and observability** features will **reduce troubleshooting and debugging time by 20%** (estimated).
- These improvements will help us achieve **faster resolution of customer issues** and ultimately create **greater customer satisfaction**.

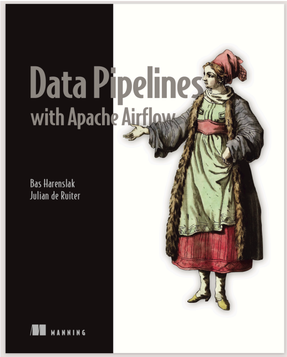
Executive Summary

Can we use Airflow safely for production operation?

- Airflow is a **mature and stable open source project** with **over eight years in development**, and **thousands of contributors**.
- Many **large enterprises** such as Lyft, EA, and Credit Suisse **use Airflow in production for business-critical tasks and workflows**.
- Several **managed cloud Airflow solutions are available**, such as AWS, GCP, or Astronomer, if we prefer to delegate the responsibility of administrating our own instances.

Resources and references

- The official documentation is very good. It should be your starting point to look something up or learn. https://airflow.apache.org/docs/
- Best Practices https://airflow.apache.org/docs/apache-airflow/stable/best-practices.html
- I know of one published book so far: Harenslak and de Ruiter, *Data Pipelines with Apache Airflow* (Manning, 2021)



🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

https://twitter.com/ManningBooks/status/1612871663003045889

> **Manning Publications** @ManningBooks
> Hey #codemash! Save 35% on #coding, #Java and other Manning books, liveProjects and videos with CODE ctwcodemash23 at: http://mng.bz/Elgr
> Twitter | Yesterday at 12:59 PM

Acknowledgements 🙏❤️

- 💪 Thanks to the CodeMash organizers, volunteers, and sponsors.
- 👔 Thanks to my employer Cardinal Health Fuse who has supported this effort
- 👨‍👩‍👧 Thanks to my family for giving me the space and time to prepare this talk
- 👊 Thanks to YOU … the attendees and audience. You are the ultimate reason why so many people invest this effort to make CodeMash 2023 great!

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

Thank You! 🙏🏾

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett

Upcoming Talks by Fuse Teammates

Building a Shared Component Library in Lit
Presented by: Tim Huddle
Time: Thursday 11:45 AM - 12:45 PM
Room: Salon D

Paired Programming: From Pain to Profit
Presented by: Austin Kucera
Time: Friday 4:00 PM - 5:00 PM
Room: Salon E

🔗 https://linkedin.com/in/ajbennett 📄 https://github.com/ajbennett