

Apache Airflow 101

Essential concepts and tips for beginners



Powered by Astronomer

Editor's Note

Welcome to the **Apache Airflow 101 ebook**, brought to you by Astronomer. We believe that Airflow is the best-in-class data orchestration technology for adapting to the ever-changing data landscape. In this ebook, we've gathered and explained key Airflow concepts and components to help you get started using this open source platform. We've also compiled some guides and tutorials for more advanced Airflow features. This ebook is everything you need to successfully kick off your Airflow journey.

Share this ebook with others!



Table of Contents

04	What is Apache Airflow?
08	Why Apache Airflow?
12	Why Open Source Software?
16	Apache Airflow 101 Core Concepts & Components
16	Core Concepts
22	Core Components
24	Apache Airflow 2
25	Major Features in Airflow 2.0
31	Major Features in Airflow 2.2
34	Major Features in Airflow 2.3
38	Useful Resources to Get Started with Airflow
38	Airflow Essentials
41	Integrations
46	About Astronomer

1. What is Apache Airflow?

Apache Airflow is the world's most popular data orchestration platform, a framework for programmatically authoring, scheduling, and monitoring data pipelines. It was created as an open-source project at Airbnb and later brought into the Incubator Program of the Apache Software Foundation, which named it a Top-Level Apache Project in 2019. Since then, it has evolved into an advanced data orchestration tool used by organizations spanning many industries, including software, healthcare, retail, banking, and fintech.

Today, with a thriving community of more than **2k contributors, 25k+ stars on Github**, and thousands of users—including organizations ranging from early-stage startups to Fortune 500s and tech giants—Airflow is widely recognized as the industry's leading data orchestration solution.

Apache Airflow's popularity as a tool for data pipeline automation has grown for a few reasons:

- **Proven core functionality for data pipelining**
Airflow's core capabilities are used by thousands of organizations in production, delivering value across scheduling, scalable task execution, and UI-based task management and monitoring.
- **An extensible framework**
Airflow was designed to make integrating existing data sources as simple as possible. Today, it supports over 80 providers, including AWS, GCP, Microsoft Azure, Salesforce, Slack, and Snowflake. Its ability to meet the needs of simple and complex use cases alike makes it both easy to adopt and scale.
- **Scalability**
From running a few pipelines to thousands every day, Airflow manages workflows using a reliable scheduler. Additionally, you can use parameters to fine-tune the performance of Airflow to fit any use case.
- **A large, vibrant community**
Airflow boasts thousands of users, and over 2k+ contributors who regularly submit features, plugins, content and bug fixes to ensure continuous momentum, and improvement. **So far Airflow has reached 15k+ commits and 25k+ GitHub stars.**


Apache Airflow continues to grow thanks to its active and expanding community. And because it belongs to the Apache Software Foundation and is governed by a group of PMC members, it can live forever.

As a result, hundreds of companies — including Fortune 500s, tech giants, and early-stage startups — are adopting Airflow as their preferred tool to programmatically author, schedule, and monitor workflows. Among the biggest names, you'll find Adobe, Bloomberg, Asana, Dropbox, Glassdoor, HBO, PayPal, Tesla, ThoughtWorks, WeTransfer, and more!

Apache Airflow – Built on a Strong and Growing Community

 **25k+**
Github Stars

 **21k+**
Slack community

 **8m+**
Monthly downloads

 **1.9k+**
Contributors

Apache Airflow Core Principles

Airflow is built on a set of core ideals that allow you to leverage the most popular open source workflow orchestrator on the market while maintaining enterprise-ready flexibility and reliability.

Flexible

Fully programmatic workflow authoring allows you to maintain full control of the logic you wish to execute.

Extensible

Leverage a robust ecosystem of open source mintegrations to connect natively to any third party datastore or API.

Open Source

Get the optionality of an open-source codebase while tapping into a buzzing and action-packed community.

Scalable

Scale your Airflow environment to infinity with a modular and highly-available architecture across a variety of execution frameworks.

Secure

Integrate with your internal authentication systems and secrets managers for an platform ops experience that your security team will love.

Modular

Plug into your internal logging and monitoring systems to keep all of the metrics you care about in one place.

2. Why Apache Airflow?



Viraj Parekh

Field CTO at Astronomer



Kenten Danas

Lead Developer Advocate at Astronomer

Our Field CTO Viraj Parekh and Lead Developer Advocate Kenten Danas gathered some of the most common questions data practitioners ask us on a daily basis. If you'd like to find out why Airflow is the best tool for data orchestration, here are their answers.

Q: Why would you run your pipelines as code?

At Astronomer, we believe using a code-based data pipeline tool like Airflow should be a standard. There are many benefits to that solution, but a few come to mind as high-level concepts:

Firstly, code-based pipelines are **dynamic**. If you can write it in code, then you can do it in your data pipeline. And that's really powerful.

Secondly, code-based pipelines are **highly extensible**. If your external system has an API, there's a way to integrate it with Airflow.

Finally, they are **more manageable**. Because everything is in code, these pipelines can integrate seamlessly into your source controls CI/CD and general developer workflows.

Q: What about a company that is looking at a drag-and-drop tool or other low/no-code approach? Why should they consider Airflow?

We never think of it as "Airflow or" – instead, it's always "Airflow and."

Because Airflow is so extensible, you can still use those drag-and-drop tools and then orchestrate and integrate them with other pipelines using Airflow. **This approach allows you to have a one-stop-shop for orchestration without necessarily having to make a giant migration effort upfront or convince everybody at one time.**

As those internal teams get more comfortable with Airflow and see all the benefits, they'll naturally start to transfer to Airflow.

Q: How can functional data pipelines improve my business?

Data pipelines help users manage data workflows that need to be run on a consistent, reliable schedule. You can use data pipelines to help you with simple use cases, such as running a report about yesterday's sales. Data pipelines empower those users who are looking for baseline insights into the business.

You can also use data pipelines to help with more complex use cases, such as running machine learning models, training AI, and generating complex business intelligence reports that help you make critical decisions.

So, if you want something that's flexible, standardized, and scalable that has the ongoing support of the open-source community— Airflow is perfect for you.

Q: Is Apache Airflow hard to learn?

Many resources exist for learning how to write code for Airflow, including Astronomer's [Airflow Guides](#). Airflow becomes more difficult to learn once you try to run it at business scale. Luckily, there are tools like [Astro](#) that allow for abstracting and managing the infrastructure easily and efficiently.

Q: What companies use Apache Airflow?

Walmart, Tinder, JP Morgan, Tesla, Revolut....just to name a few. Currently, hundreds of companies around the world, from various industries like healthcare, AI, fintech, or e-commerce, use Apache Airflow as part of their modern data stack to help them make sense of their data and drive better business decisions.



Get Apache Airflow Support

Schedule time with Astronomer experts to learn how we can help you address your Airflow issues.

Contact Us

3. Why Open Source Software?

The massive increase in open source software (OSS) projects is changing both the tech scene and the business scene on a global scale. So why do we love open-source? **Because it is:**

Innovative

According to [this 2016 article in Wired](#), Facebook decided to stop “treating data center design like **Fight Club**” by making its architecture open and accessible to everyone. Companies like Microsoft, HP, and even Google followed suit by open-sourcing many of their technologies that had been kept secret for years. By making code for their projects accessible to the public, companies could freely adapt golden standards in the open-source world and create better integrations between products, all while freeing up time to innovate elsewhere.

Reliable

We’ve all had “technical difficulties” when our technology operated unreliably. Professionally, that can cause a variety of outcomes, from mild annoyance to genuine detriment. When everybody’s using the same components, everybody’s optimizing the same components. As Linux Creator Linus Torvalds said, “Given enough eyeballs, all bugs are shallow” (**Linus’s Law**). In other words, the more people who can see and test a set of code, the more likely any flaws will be caught and fixed quickly.

Diverse

It’s a huge benefit to have a large, diverse community of developers look at and contribute to the same system. Instead of a few people with the same view of the world-building components, an entire community with varying perspectives and strengths contributes to the system (and has a stake in its success and adoption), which makes the system more resilient.

Transparent

When software is transparent people aren’t tied down to a proprietary system protected in a black box. Users and builders of open source software can see everything; they can open GitHub, look at the source code, and trust that the code has been reviewed publicly. This gives decision-makers insight into what’s going on — where components can be swapped around — and the control they need to do it.

Agile

When components live in a community-curated open source world, they are built to “play nicely” with other components. This flexibility allows us to stay on the cutting edge. With open-source, any organization can take a piece of code and customize it to best fit their needs. When it comes to tech, we want to live in a world where we’re free to explore, invent best-in-class technology and use it in revolutionary ways. We can do it through open source.



Apache Airflow Fundamentals

The Astronomer Certification for Apache Airflow Fundamentals is a course and exam focused on imparting an understanding of the basics of the Airflow architecture, and the ability to create basic data pipelines for scheduling and monitoring tasks.

Concepts Covered:

- User Interface
- Scheduling
- Backfill & Catchup
- DAG Structure
- Architectural Components
- Use Cases

Get Certified

4. Apache Airflow 101

Core Concepts & Components

Airflow Core Concepts

DAGs

In Airflow, a DAG is your data pipeline and represents a set of instructions that must be completed in a specific order. This is beneficial to data orchestration for a few reasons:

- **DAG dependencies** ensure that your data tasks are executed in the same order every time, making them reliable for your everyday data infrastructure.

- **The graphing component of DAGs** allows you to visualize dependencies in Airflow's user interface.
- **Because every path in a DAG is linear**, it's easy to develop and test your data pipelines against expected outcomes.

Learn more:



ARTICLE

What exactly is a DAG?

[SEE ARTICLE →](#)

Operators

Operators are the building blocks of Airflow. They contain the logic of how data is processed in a pipeline. There are different operators for different types of work: some operators execute general types of code, while others are designed to complete very specific types of work.

```
1 operator
2     file = open("myfile", "r")
3     print (f.read() )
```

- **Action Operators** execute pieces of code. For example, a Python action operator will run a Python function, a bash operator will run a bash script, etc.
- **Transfer Operators** are more specialized, designed to move data from one place to another.
- **Sensor Operators**, frequently called “sensors,” are designed to wait for something to happen — for example, for a file to land on S3, or for another DAG to finish running.

When you create an instance of an operator in a DAG and provide it with its required parameters, it becomes a task.

Learn more:

 **TUTORIAL**
Operators 101
[SEE TUTORIAL →](#)

 **TUTORIAL**
Deferrable Operators
[SEE TUTORIAL →](#)


 **TUTORIAL**
Sensors 101
[SEE TUTORIAL →](#)

Tasks

A task is an instance of an operator. In order for an operator to complete work within the context of a DAG, it must be instantiated through a task. Generally speaking, you can use tasks to configure important context for your work, including when it runs in your DAG.

Learn more:


 **GUIDE**
Using Task Groups in Airflow
[SEE GUIDE →](#)

 **GUIDE**
Passing Data Between Airflow Tasks
[SEE GUIDE →](#)

A data pipeline

A “data pipeline” describes the general process by which data moves from one system into another. From an engineering perspective, it can also be represented by a DAG. Each task in a DAG is defined by an operator, and there are specific downstream or upstream dependencies set between tasks. A DAG run either extracts, transforms, or loads data - becoming a data pipeline, essentially.

Learn more:

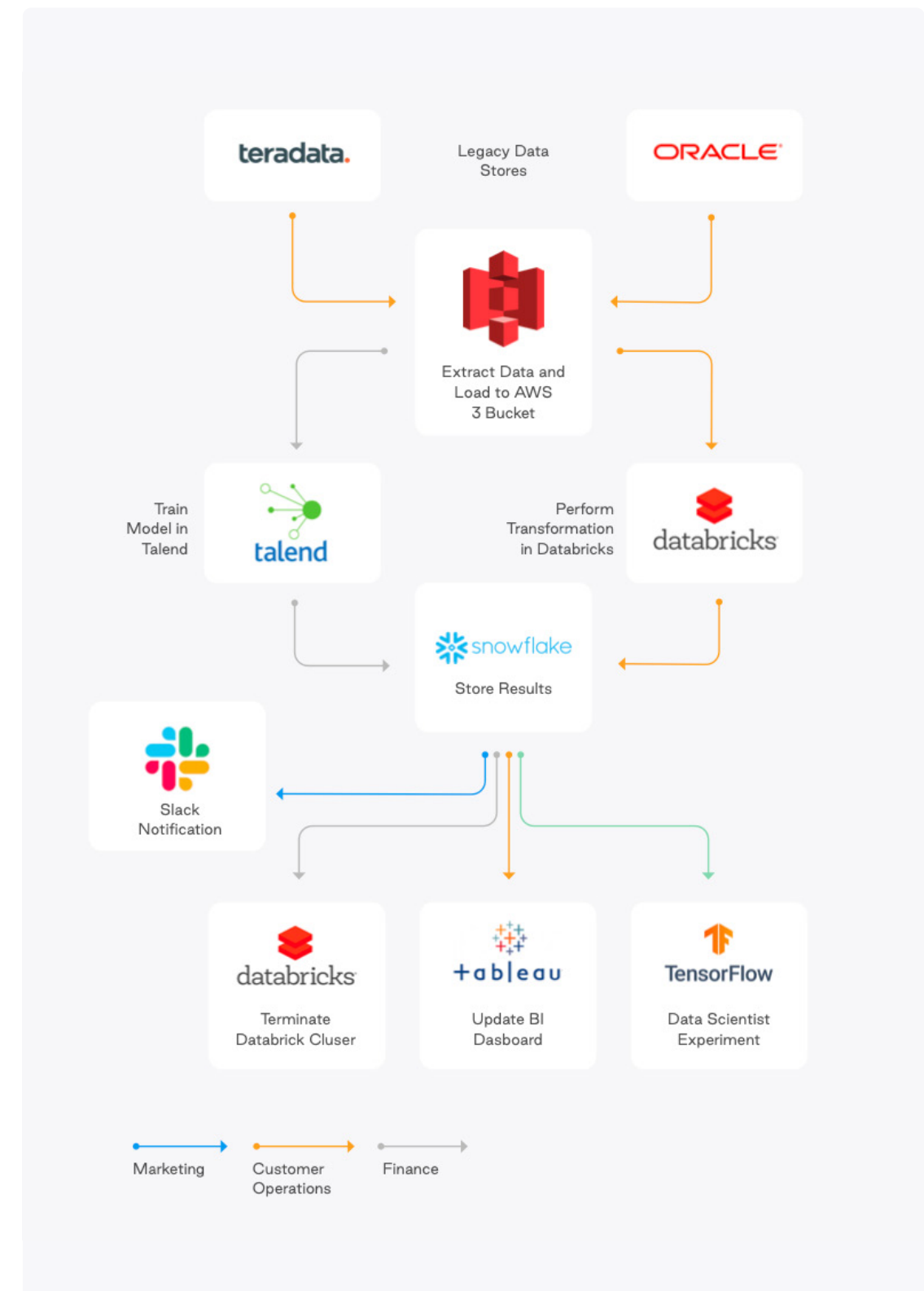
 **ARTICLE**
Data Pipeline: Components, Types, and Best Practices
[SEE ARTICLE →](#)

Providers

Airflow providers are Python packages that contain all of the relevant Airflow modules for interacting with external services. Airflow is designed to be an agnostic workflow orchestrator: You can do your work within Airflow, but you can also use other tools with it, like AWS, Snowflake, or Databricks.

Most of these tools already have community-built Airflow modules, giving Airflow spectacular flexibility. Check out the [Astronomer Registry](#) to find all the providers.

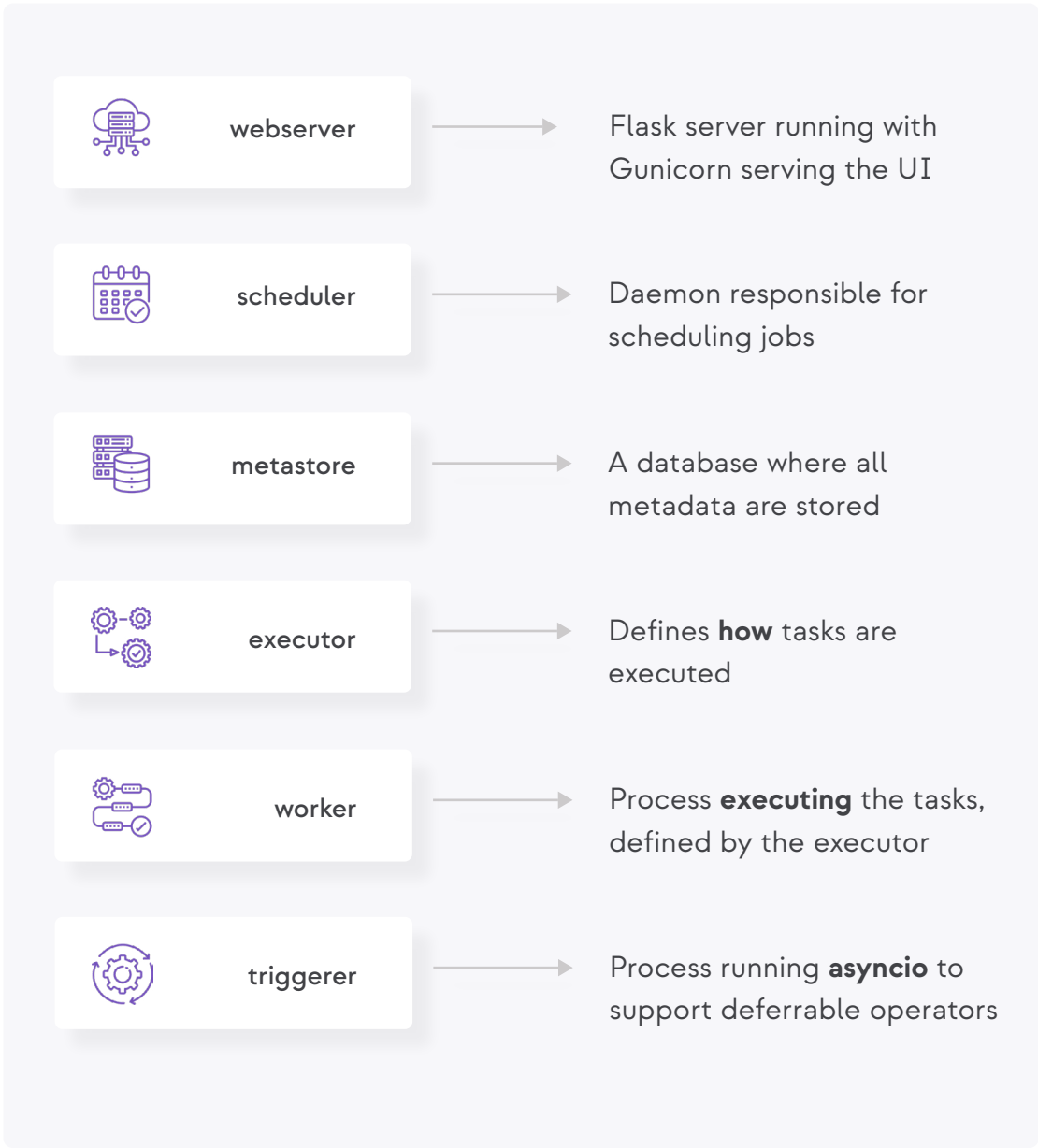
The following diagram shows how these concepts work in practice. As you can see, by writing a single DAG file in Python using existing provider packages, you can begin to define complex relationships between data and actions.



Airflow Core Components

When working with Airflow, it is important to understand the underlying components of its infrastructure. Even if you mostly interact with Airflow as a DAG author, knowing which components are “under the hood” and why they are needed can be helpful for developing your DAGs, debugging, and running Airflow successfully. The first four components below run at all times, and the last two are situational components that are used only to run tasks or make use of certain features.

The infrastructure:



5. Apache Airflow 2

As [Apache Airflow](#) grew in adoption in recent years, a major release to expand on the project's core strengths became long overdue. Astronomer was delighted to release Airflow 2.0 together with the community in December 2020.

Since then, various organizations and leaders within the Airflow community have worked in close collaboration, refining the scope of Airflow 2.0, enhancing existing functionality, and introducing changes to make Airflow faster, more reliable, and more performant at scale.

Major Features in Airflow 2.0

Airflow 2.0 includes hundreds of features and bug fixes both large and small. Many of the significant improvements were influenced and inspired by feedback from [Airflow's 2019 Community Survey](#), which garnered over 300 responses.

A New Scheduler: Low-Latency + High-Availability

The Airflow scheduler as a core component has been key to the growth and success of the project following its creation in 2014. In fact, "Scheduler Performance" was the most asked for improvement in the Community Survey. Airflow users have found that while the Celery and [Kubernetes Executors](#) allow for task execution at scale, the scheduler often limits the speed at which tasks are scheduled and queued for execution. While effects vary across use cases, it's not unusual for users to grapple with induced downtime and a long recovery in the case of a failure and experience high latency between short-running tasks.

It is for that reason we introduced a new, refactored scheduler with the Airflow 2.0 release. The most impactful Airflow 2.0 change in this area is support for running multiple schedulers concurrently in an active/active model. Coupled with [DAG](#) Serialization, Airflow's refactored scheduler is now highly available, significantly faster, and infinitely scalable. Here's a quick overview of the new functionality:

- 1. Horizontal Scalability.** If task load on one scheduler increases, a user can now launch additional "replicas" of the scheduler to increase the throughput of their Airflow Deployment.
- 2. Lowered Task Latency.** In Airflow 2.0, even a single scheduler has proven to schedule tasks at much faster speeds with the same level of CPU and Memory.

3. **Zero Recovery Time.** Users running 2+ schedulers see zero downtime and no recovery time in the case of a failure.
4. **Easier Maintenance.** The Airflow 2.0 model allows users to make changes to individual schedulers without impacting the rest and inducing downtime.

The scheduler's now-zero recovery time and readiness for scale eliminate it as a single point of failure within Apache Airflow. Given the significance of this change, our team published "[The Airflow 2.0 Scheduler](#)," a blog post that dives deeper into the story behind scheduler improvements alongside an architecture overview and benchmark metrics.

For more information on how to run more than one scheduler concurrently, refer to the [official documentation on the Airflow Scheduler](#).

Full REST API

[Data engineers](#) have been using Airflow's "Experimental API" for years, most often for [triggering DAG runs programmatically](#). With that said, the API has historically remained narrow in scope and lacked critical elements of functionality, including a robust authorization and permissions framework. Airflow 2.0 introduces a new, comprehensive REST API that sets a strong foundation for a new Airflow UI and CLI in the future. Additionally, the new API:

- **Makes for easy access by third-parties.**
- **Is based on the [Swagger/OpenAPI Spec](#).**
- **Implements CRUD (Create, Update, Delete) operations on all Airflow resources.**
- **Includes authorization capabilities (parallel to those of the Airflow UI).**

These capabilities enable a variety of use cases and create new opportunities for automation. For example, users now have the ability to programmatically set Connections and Variables, show import errors, create Pools, and monitor the status of the Metadata Database and scheduler.

For more information, reference [Airflow's REST API documentation](#).

TaskFlow API

While Airflow has historically shined in scheduling and running idempotent tasks, it has historically lacked a simple way to pass information between tasks. Let's say you are writing a DAG to train some set of Machine Learning models. The first set of tasks in that DAG generates an identifier for each model and the second set of tasks outputs the results generated by each of those models. In this scenario, what's the best way to pass output from that first set of tasks to the latter?

Historically, [XComs](#) have been the standard way to pass information between tasks and would be the most appropriate method to tackle the use case above. As most users know, however, XComs are often cumbersome to use and require redundant boilerplate code to set return variables at the end of a task and retrieve them in downstream tasks.

With Airflow 2.0, we were excited to introduce the TaskFlow API and Task Decorator to address this challenge. The TaskFlow API implemented in 2.0 makes DAGs significantly easier to write by abstracting the task and dependency management layer from users. Here's a breakdown of the functionality:

- **A framework that automatically creates `PythonOperator` tasks from Python functions and handles variable passing.** Now, variables such as Python Dictionaries can simply be passed between tasks as return and input variables for cleaner and more efficient code.
- **Task dependencies are abstracted and inferred as a result of the Python function invocation.** This again makes for much cleaner and simpler [DAG writing](#) for all users.
- **Support for Custom XCom Backends. Airflow 2.0 includes support for a new `xcom_backend` parameter that allows users to pass even more objects between tasks.** Out-of-the-box support for S3, HDFS, and other tools is coming soon.

It's worth noting that the underlying mechanism here is still XCom and data is still stored in Airflow's Metadata Database, but the XCom operation itself is hidden inside the `PythonOperator` and is completely abstracted from the DAG developer. Now, Airflow users can pass information and manage dependencies between tasks in a standardized Pythonic manner for cleaner and more efficient code.

To learn more, refer to [Airflow documentation on the TaskFlow API](#) and the [accompanying tutorial](#).

Task Groups

[Airflow SubDAGs](#) have long been limited in their ability to provide users with an easy way to manage a large number of tasks. The lack of parallelism coupled with confusion around the fact that SubDAG tasks can only be executed by the Sequential Executor, regardless of which executor is employed for all other tasks, made for a challenging and unreliable user experience.

Airflow 2.0 introduced [Task Groups](#) as a UI construct that doesn't affect task execution behavior but fulfills the primary purpose of SubDAGs. Task Groups give a DAG author the management benefits of "grouping" a logical set of tasks with one another without having to look at or process those tasks any differently.

While Airflow 2.0 continues to support the SubDAG Operator, Task Groups are intended to replace it in the long term.

Independent Providers

One of Airflow's signature strengths is its [sizeable collection of community-built operators, hooks, and sensors](#) — all of which enable users to integrate with external systems like AWS, GCP, Microsoft Azure, Snowflake, Slack and many more.

Providers have historically been bundled into the core Airflow distribution and versioned alongside every Apache Airflow release. As of Airflow 2.0, they are now split into their own [airflow/providers](#) directory such that they can be released and versioned independently from the core Apache Airflow distribution. Cloud service release schedules often don't align with the Airflow release schedule and either result in incompatibility errors or prohibit users from being able to run the latest versions of certain providers. The separation in Airflow 2.0 allows the most up-to-date versions of Provider packages to be made generally available and removes their dependency on core Airflow releases.

It's worth noting that some operators, including the Bash and Python Operators, remain in the core distribution given their widespread usage.

To learn more, refer to [Airflow documentation on Provider Packages](#).

Simplified Kubernetes Executor

Airflow 2.0 includes a re-architecture of the [Kubernetes Executor](#) and [KubernetesPodOperator](#), both of which allow users to dynamically launch tasks as individual Kubernetes Pods to optimize overall resource consumption.

Given the known complexity users previously had to overcome to successfully leverage the executor and operator, we drove a concerted effort towards simplification that ultimately involved removing over 3,000 lines of code. The changes incorporated in Airflow 2.0 made the executor and operator easier to understand, faster to execute and offers far more flexibility in configuration.

Data Engineers have now access to the full Kubernetes API to create a yaml 'pod_template_file' instead of being restricted to a partial set of configurations through parameters defined in the airflow.cfg file. We've also replaced the executor_config dictionary with the `pod_override` parameter, which takes a Kubernetes V1Pod object for a clear 1:1 override setting.

For more information, we encourage you to follow the documentation on the new [pod_template_file](#) and [pod_override](#) functionality.

UI/UX Improvements

Perhaps one of the most welcomed sets of changes brought by Airflow 2.0 has been the visual refreshment of the [Airflow UI](#).

In an effort to give users a more sophisticated and intuitive front-end experience, we've made over 30 UX improvements.

Major Features in Airflow 2.2

Airflow 2.2 Big Features

Deferrable tasks

Turn any task into a super-efficient asynchronous loop

Custom timetables

Run DAGs exactly when you want

@task.docker

Easily run Python functions in separate docker containers

Custom Timetables

Cron expressions got us as far as regular time intervals, which is only convenient for predictable data actions. Before Airflow 2.2.0, for example, you could not run a DAG at 9am and 12:30pm, because that schedule has different hours and minutes, which is impossible with Cron. Now that the custom timetables are finally here, the scheduling sky is the limit. With this feature, it's also now possible to look at a given data for a specific period of time.

New concept: `data_interval` — A period of data that a task should operate on.

Since the concept of `execution_date` was confusing to every new user, a better version is now available. No more "why didn't my [DAG](#) run?", as this feature has been replaced with `data_interval`, which is the period of data that a task should operate on.

It includes:

- `logical_date` (aka `execution_date`)
- `data_interval_start` (same value as `execution_date` for cron)
- `data_interval_end` (aka `next_execution_date`)

Key Benefits of Custom Timetables

- Flexibility
- Run DAGs
- Introducing explicit “data interval”
- Full back-compatibility maintained — `schedule_interval` not going away

Bonus for Astronomer Customers: NYSE Trading Timetable

The trading hours timetable allows users to run DAGs based on the start and end of trading hours for NYSE and Nasdaq. It includes historic trading hours as well as holidays and half-days where the markets have irregular hours.

Deferrable Operators

Many longtime Airflow users are familiar with the frustration of tasks or sensors clogging up worker resources when waiting for external systems

and events. Airbnb introduced smart sensors, the first tackle to this issue. Deferrable operators go further than smart sensors — they are perfect for anything that submits a job to an external system and then polls for status. With this feature, operators or sensors can postpone themselves until a light-weight async check succeeds, at which time they can resume executing. This causes the worker slot, as well as any resources it uses, to be returned to Airflow. A deferred task does not consume a worker slot while in deferral mode — instead, one triggerer process (a new component, which is the daemon process that executes the asyncio event loop) can run 100s of async deferred tasks concurrently. As a result, tasks like monitoring a job on an external system or watching for an event become far less expensive.

Custom `@task` decorators and `@task.docker`

The ‘`@task.docker`’ decorator allows for running a function inside a docker container. Airflow handles putting the code into the container and returning xcom. This is especially beneficial when there are competing dependencies between Airflow and tasks that must run.

Key Benefits of Deferrable Tasks

- Turn any task into a super-efficient asynchronous loop
- Resilient against restarts
- Resilient against restarts
- Doesn't use worker resources when deferred
- **Paves the way to event-based DAGs!**

Major Features in Airflow 2.3

Note: Airflow 2.3 was released in April 2022

Dynamic Task Mapping

Dynamic task mapping permits Airflow’s scheduler to trigger tasks based on context: given this input — for example, a set of files in an S3 bucket — run these tasks. The number of tasks can change at run time based on the number of files; you no longer have to configure a static number of tasks.

As of Airflow 2.3, you can use dynamic task mapping to hook into S3, list any new files, and run separate instances of the same task for each file. You don’t have to worry about keeping your Airflow workers busy, because Airflow’s scheduler automatically optimizes for available parallelism.

Dynamic task mapping not only gives you a means to easily parallelize these operations across your available Airflow workers, but also makes it easier to rerun individual tasks (e.g., in the case of failure) by giving you vastly improved visibility into the success or failure of these tasks. Imagine that you create a monolithic task to process all of the files in the S3 bucket, and that one or more steps in this task fail. In past versions of Airflow, you’d have to parse the Airflow log file generated by your monolithic task to determine which step failed and why. In 2.3, when a dynamically mapped task fails, it generates a discrete alert for that step in Airflow. This enables you to zero in on the specific task and troubleshoot from there. If appropriate, you can easily requeue the task and run it all over again.

Learn more:



GUIDE

Dynamic Tasks in Airflow

[SEE GUIDE →](#)

New Grid View

Airflow’s new grid view replaces the tree view, which was not ideal for representing DAGs and their topologies, since a tree cannot natively represent a DAG that has more than one path, such as a task with branching dependencies. The tree view could only represent these paths by displaying multiple, separate instances of the same task. So if a task had three paths, Airflow’s tree view would show three instances of the same task — confusing even for expert users.

The grid view offers first-class support for Airflow task groups. The tree view chained task group IDs together, resulting in repetitive text and, occasionally, broken views. In the new grid view, task groups display summary information and can be expanded or collapsed as needed. The new grid view also dynamically generates lines and hover effects based on the task you’re inspecting, and displays the durations of your DAG runs; this lets you quickly track performance — and makes it easier to spot potential problems. These are just a few of the improvements the new grid view brings.

A New LocalKubernetesExecutor

Airflow 2.3's new local K8s executor allows you to be selective about which tasks you send out to a new pod in your K8s cluster — i.e., you can either use a local Airflow executor to run your tasks within the scheduler service or send them out to a discrete K8s pod. Kubernetes is powerful, to be sure, but it's overkill for many use cases. (It also adds layers of latency that the local executor does not.) The upshot is that for many types of tasks — especially lightweight ones — it's faster (and, arguably, just as reliable) to run them in Airflow's local executor, as against spinning up a new K8s pod.

Storing Airflow Connections in JSON Instead of URI Format

And the new release's [ability to store Airflow connections in JSON](#) (rather than in Airflow URI) format is a relatively simple feature that — for some users — could nevertheless be a hugely welcomed change. Earlier versions stored connection information in Airflow's URI format, and there are cases in which that format can be tricky to work with. JSON provides a simple, human-readable alternative.

Altogether, Airflow 2.3 introduces more than a dozen new features, including, in addition to the above, a new command in the Airflow CLI [for reserializing DAGs](#), a new [listener plugin API](#) that tracks TaskInstance state changes, a new [REST API endpoint for bulk-pausing/resuming DAGs](#), and other ease-of-use (or fit-and-finish) features. Learn more about [Airflow 2.3 on our blog](#).

Ready for more?

Discover our guides, covering everything from introductory content to advanced tutorials around Airflow.

Discover Guides

6. Useful Resources to Get Started with Airflow

Airflow Essentials



WEBINAR

Intro to Airflow

[SEE WEBINAR →](#)



WEBINAR

Intro to Data Orchestration with Apache Airflow

[SEE WEBINAR →](#)



VIDEO

Coding Your First DAG for Beginners

[SEE VIDEO →](#)



WEBINAR

Best Practices For Writing DAGs in Airflow 2

[SEE WEBINAR →](#)



GUIDE

Understanding the Airflow UI

[SEE GUIDE →](#)



GUIDE

Airflow Executors Explained

[SEE GUIDE →](#)



GUIDE

Managing Your Connections in Apache Airflow

[SEE GUIDE →](#)



GUIDE

DAG Writing Best Practices

[SEE GUIDE →](#)



GUIDE

Using Airflow to Execute SQL

[SEE GUIDE →](#)



GUIDE

Using TaskGroups in Airflow

[SEE GUIDE →](#)



GUIDE

Debugging DAGs

[SEE GUIDE →](#)



GUIDE

Rerunning DAGs

[SEE GUIDE →](#)



GUIDE

Airflow Decorators

[SEE GUIDE →](#)



GUIDE

Deferrable Operators

[SEE GUIDE →](#)



WEBINAR

Scaling out Airflow

[SEE WEBINAR →](#)



WEBINAR

Scheduling in Airflow

[SEE WEBINAR →](#)

Integrations

Make the most of Airflow by connecting it to other tools.

Airflow + dbt



ARTICLE

Building a Scalable Analytics Architecture With Airflow and dbt

[SEE ARTICLE →](#)



ARTICLE

Airflow and dbt, Hand in Hand

[SEE ARTICLE →](#)



GUIDE

Integrating Airflow and dbt

[SEE GUIDE →](#)

Airflow + Great Expectations



GUIDE

Integrating Airflow and Great Expectations

[SEE GUIDE →](#)

Airflow + Azure



GUIDE

Executing Azure Data Factory Pipelines with Airflow

[SEE GUIDE →](#)



GUIDE

Executing Azure Data Explorer Queries with Airflow

[SEE GUIDE →](#)



GUIDE

Orchestrating Azure Container Instances with Airflow

[SEE GUIDE →](#)

Airflow + Redshift



GUIDE

Orchestrating Redshift Operations from Airflow

[SEE GUIDE →](#)

Airflow + Sagemaker



GUIDE

Using Airflow with SageMaker

[SEE GUIDE →](#)

Airflow + Notebooks



GUIDE

Executing Notebooks with Airflow

[SEE GUIDE →](#)

Airflow + Kedro



GUIDE

Deploying Kedro Pipelines to Apache Airflow

[SEE GUIDE →](#)

Airflow + Databricks



GUIDE

Orchestrating Databricks Jobs with Airflow

[SEE GUIDE →](#)

Airflow + Talend

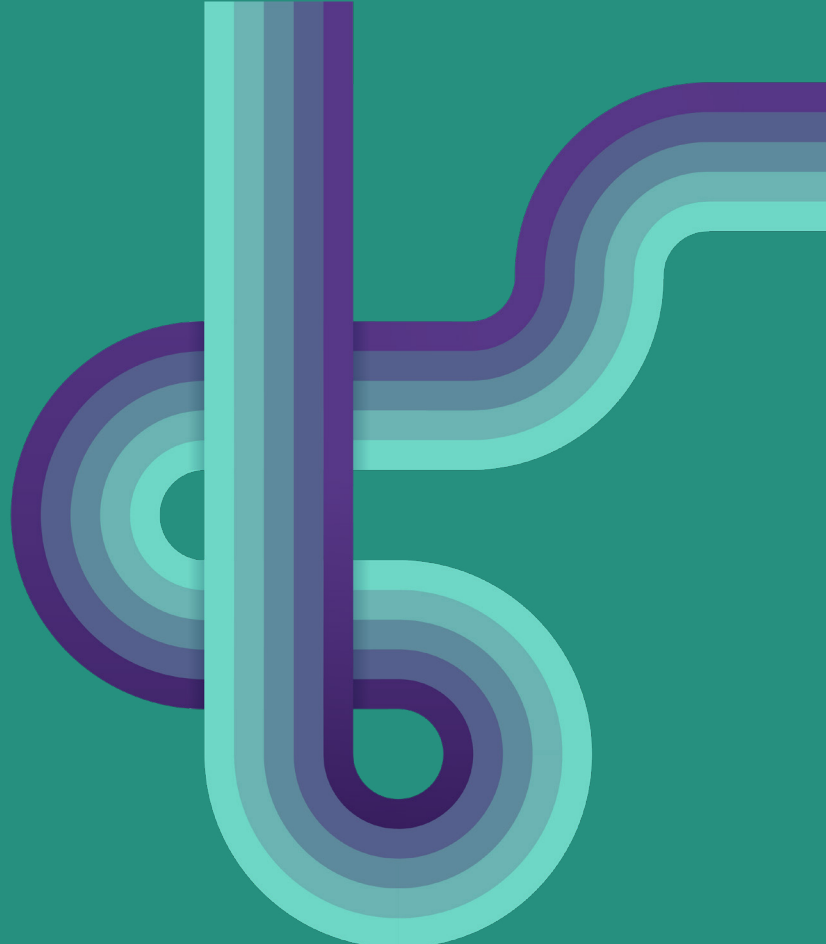
 **GUIDE**
Executing Talend Jobs with Airflow
[SEE GUIDE →](#)

Airflow + AWS Lambda

 **GUIDE**
Best Practices Calling AWS Lambda from Airflow
[SEE GUIDE →](#)

Setting Up Error Notifications in Airflow Using Email, Slack, and SLAs

 **GUIDE**
Error Notifications in Airflow
[SEE GUIDE →](#)



Can't find
the integration you're
looking for?

Check out the Astronomer Registry — a go-to discovery and distribution hub for Apache Airflow integrations.

[Browse Registry](#)

7. About Astronomer

With its rapidly growing number of downloads and contributors, Airflow is at the center of the data management conversation—a conversation Astronomer is helping to drive.

Our commitment is evident in our people. We've got a robust Airflow Engineering team and sixteen active committers on board, including seven PMC members: **Ash Berlin-Taylor**, **Kaxil Naik**, **Daniel Imberman**, **Jed Cunningham**, **Bolke de Bruin**, **Sumit Maheshwari**, and **Ephraim Anierobi**.

A significant portion of the work all these people do revolves around Airflow releases—creating new features, testing, troubleshooting, and ensuring that the project continues to improve and grow. Their hard work, combined with that of the community at large, allowed us to deliver Airflow 2.0 in late 2020, Airflow 2.2 in 2021, and Airflow 2.3 in early 2022.

With every new release, Astronomer is unlocking more of Airflow's potential and moving closer to the goal of democratizing the use of Airflow, so that all members of the data team can work with or otherwise benefit from it. And we've also built [Astro](#)—the essential data orchestration platform. With the strength and vibrancy of Airflow at its core, Astro offers a scalable, adaptable, and efficient blueprint for successful data orchestration.

[If you'd like to learn more about how Astronomer drives the Apache Airflow project together with the community check out this recent article.](#)



ASTRONOMER

Thank you

We hope you've enjoyed our guide to Airflow. Please follow us on Twitter and LinkedIn, and share your feedback, if any.



Start building your next-generation data platform with Astro

Get Started

Experts behind this ebook:

Viraj Parekh | Field CTO at Astronomer
Kenten Danas | Lead Developer Advocate at Astronomer
Jake Witz | Technical Writer at Astronomer

Created by ©Astronomer 2022

ASTRONOMER