# Season of Docs 2019 [ARCHIVED]

The Apache Airflow community is happy to share that we have applied to participate in the first edition of Season of Docs.

Season of Docs is a program organized by Google Open Source to match technical writers with mentors to work on documentation for open source projects. We, at Apache Airflow, couldn't be more excited about this opportunity, because as a small, but fast growing project, we need to make sure that our documentation stays up to date, and in good condition.

After a discussion that involved many members of the community, where lots of ideas were shared, we were able to gather a set of projects that we would love a technical writer to work on. In fact, we have asked for two technical writers in our application, as we believe there is a big interest in the community, and lots of work to be done.

If you like the idea of spending a few months working in our awesome open source project, in a welcoming community that will be thrilled to see your contributions, please take a look at the projects list, and consider applying for Season of Docs. If you have any questions, do not hesitate to reach out to us in the Apache Airflow mailing list at dev@airflow.apache.org (you will need to subscribe first by emailing dev-subscribe@airflow.apache.org), and the Airflow slack channel.

Our team of mentors are looking forward to hearing from you <3

## Project Ideas

### 1. Apache Airflow architecture

Jira issue: AIRFLOW-4368

#### Project description

The Apache Airflow website does not have an architectural overview section. An overview would enable new contributors and users to develop a mental model of Apache Airflow and to start contributing sooner.

This project involves documenting the different parts of Apache Airflow and how they are developed. This documentation should answer:

- What are the different components of Apache Airflow[1]?
- Which components are stateful or stateless?
- How does Apache Airflow distribute tasks to workers[2]?
- How can Apache Airflow run on different deployment architectures and databases?

#### Expected deliverables

- A page that describes the architecture
- The page should have detailed descriptions of each of the following components:
    - Scheduler
    - Web Server
    - Worker
    - Metadata DB
- The page should contain a diagram of the Apache Airflow architecture (e.g. [1])
- Description of how Apache Airflow schedules tasks[2]
- Detailed examples with diagrams and text [3]

#### Related resources

[1] https://imgur.com/a/YGpg5Wa

[2] https://blog.sicara.com/using-airflow-with-celery-workers-54cb5212d405

[3] https://github.com/plantuml/plantuml

### 2. Deployment

Jira issue: AIRFLOW-4369

#### Project description

Apache Airflow automates and orchestrates complex workflows. It hides the complexity of managing dependencies between operators and scheduling tasks, enabling users to focus on the logic of their workflows.

On the other hand, deploying Apache Airflow in a resilient manner is the first step to using it. There are many strategies to deploy it, and each has advantages and disadvantages. By documenting deployment, project newcomers will be able to adopt Apache Airflow with confidence.

This project will document strategies to deploy Apache Airflow in the following environments:

- Cloud (AWS / GCP / Azure)

- On-premises
- Special attention can be given to the Kubernetes executor, as Kubernetes is a very popular technology to manage workloads.

### Expected deliverables

- A page that introduces and describes deployment techniques
- A page that describes the deployment models and helps users choose the best one (full management, GKE-like service, PaaS - Astronomer /Google Composer)
- A page that helps users to choose the best executor
- A section that describes how to deploy Apache Airflow with Kubernetes. The section should include snippets for Kubernetes files, scripts, a PlantUML diagram for clarity.
- A section on running Apache Airflow on different cloud providers (AWS / Azure / GCP)
- A table comparing different executors

### Related resources

[1] https://github.com/jghoman/awesome-apache-airflow

[2] https://apache-airflow.slack.com/archives/CCV3FV9KL/p1554319091033300; https://apache-airflow.slack.com/archives/CCV3FV9KL/p1553708569192000

[3] https://gtoonstra.github.io/etl-with-airflow/

## 3. Testing

Jira issue: AIRFLOW-4370

### Project description

Apache Airflow enables people to perform complex workflows that might affect many components in their infrastructure. It is important to be able to test an Apache Airflow workflow and ensure that it works as intended when run in a production environment.

The existing documentation does not have information that helps users test their workflows (known as DAGs), schedule DAGs properly, and write their own custom operators.

Users who know best practices for creating Apache Airflow DAGs and using operators will be able to adopt Apache Airflow more easily and with fewer mishaps.

### Expected deliverables

- A page or section that introduces testing workflows. The page should include information about the following testing stages:
    - Unit tests: applies to one class
    - DAG integrity tests: checks DAG code for missing variables, imports, etc.
    - System tests
    - Data tests: checks if the DAG performs its purpose
- A page for designing and testing DAGs and includes the following information:
    - Tips and working examples on good practices for designing DAGs
    - Descriptions on how to perform DAG dry-runs
    - Descriptions on how to write unit tests for DAGs
    - Snippets with working examples of DAGs and tests for them, including diagrams where possible
- A section on how to develop operators that are testable

### Related resources

[1] https://github.com/jghoman/awesome-apache-airflow

[2] https://airflow.apache.org/scheduler.html

[3] https://github.com/PolideaInternal/airflow/blob/simplified-development-workflow/CONTRIBUTING.md

[4] Airflow Breeze

## 4. How to create a workflow

Jira issue: AIRFLOW-4371

### Project description

In Apache Airflow, workflows are saved as a code.  DAGs use operators to build complex workflow. A DAG is a collection of all the tasks you want to run, organized in a way that reflects their relationships and dependencies. A developer can describe the relationships in several ways. Task logic is saved in operators. Apache Airflow has operators that integrate with many services, but often developers need to write their own operators. Tasks can use the xcom metabase to communicate.

### Expected deliverables

- A page for how to create a DAG that also includes:
    - Revamping the page related to scheduling a DAG
    - Adding tips for specific DAG conditions, such as rerunning a failed task
- A page for developing custom operators that includes:
    - Describing mechanisms that are important when creating an operator, such as template fields, UI color, hooks, connection, etc.
    - Describing the responsibility between the operator and the hook
    - Considerations for dealing with shared resources (such as connections and hooks)
- A page that describes how to define the relationships between tasks. The page should include information about:
    - >> <<
    - set_upstream/set_downstream
    - helpers method ex. chain
- A page that describes the communication between tasks that also includes:
    - Revamping the page related to macros and XCOM

## 5. Documenting using local development environments

Jira issue: AIRFLOW-4372

### Project description

Currently, people who want to join Apache Airflow Community and start contributing to Apache Airflow might find it very difficult to on-board. Setting up a local development environment is difficult. Depending on the level of testing needed, Apache Airflow might require manual setup of a combination of environment variables, external dependencies (Postgres, MySQL databases, Kerberos, and others), proper configuration and database initialization. Additionally, they have to know how to run the tests. There are scripts that run in CI environment to help, but the scripts are typically used for running a full set of tests not individual tests.

All 3600+ tests in Apache Airflow are executed in CI, and the problem we are trying to solve is that it's very difficult for developers to recreate failures in CI locally and fix them. It takes time and effort to run and re-run the tests and iterate while trying to fix the failures. Also Apache Airflow project is being continuously developed, and there are lots of changes from multiple people. It's hard to keep up with the changes in your local development environment (currently, it requires full rebuild after every change).

There are three different types of environments, and it's not easy to decide which one to use based on the limitations and benefits of those environments. The environments are: Local virtualenv for IDE integration and running unit tests, self-managed docker image based environment for simple integration tests with SQLite, and CI Docker-compose based environment.

We have a Simplified Development Environment (work in progress), which makes it very easy to create and manage the CI Docker Based environment. You can have a working environment in fewer than 10 minutes from scratch that is self managed, as it is being developed by others. You can run tests immediately and iterate quickly (re-running tests have sub-seconds overhead compared to 20-30 seconds previously and it has a built in self-management features). The environment rebuilds incrementally when there are incoming changes from other people. The environment is called Breeze (like "It's a Breeze to develop Apache Airflow")

We would like to not only give  the environment but also improve the documentation so that it is easy to discover and understand when and how to use it. The benefit is a faster learning curve for new developers joining the project, thus opening community to more developers. It will also help the experienced developers be able to iterate faster while fixing problems and implementing new features.

The relevant documentation is currently a work in progress, but it is already completed [1][2].

There are two relevant documents: CONTRIBUTING.md and BREEZE.rst. But we can think about different structure.

### Expected deliverables

- A chapter or page of onboarding documentation that will be easy to find for new developers joining Apache Airflow community or someone who wants to start working on Apache Airflow development on a new PC. Ideally, the documentation could be a step-by-step guide, interactive tutorial, or a video guide—generally something easy to follow. Specifically, it should be clear that there are different local development environments from a local virtualenv through Docker image to full-blown replica of CI integration testing environment and that choosing one depends on your needs and experience level.

[1] https://github.com/PolideaInternal/airflow/blob/simplified-development-workflow/CONTRIBUTING.md

[2]Airflow Breeze

# 6. System maintenance

Jira issue: AIRFLOW-4373

### Project description

Users rely on Apache Airflow to provide a reliable scheduler to orchestrate and run tasks. This means that an Airflow deployment should be resilient and low maintenance. This project involves documenting how to ensure a reliable deployment and maintain a healthy Apache Airflow Instance.

Examples of things that can be documented include:

- Good practices on how to ensure continuous, trouble-free system operation
- Methods and mechanisms for system monitoring

- Description of the SLA mechanism, such as:
    - Monitoring a running Apache Airflow instance and doing health checks, etc.
    - Setting up Prometheus and Grafana, the two most common monitoring tools, to monitor Apache Airflow metrics

### Expected deliverables

- Instructions and a step-by-step guide on how to set up monitoring for Apache Airflow, including Prometheus and Grafana (two most common monitoring tools)

---