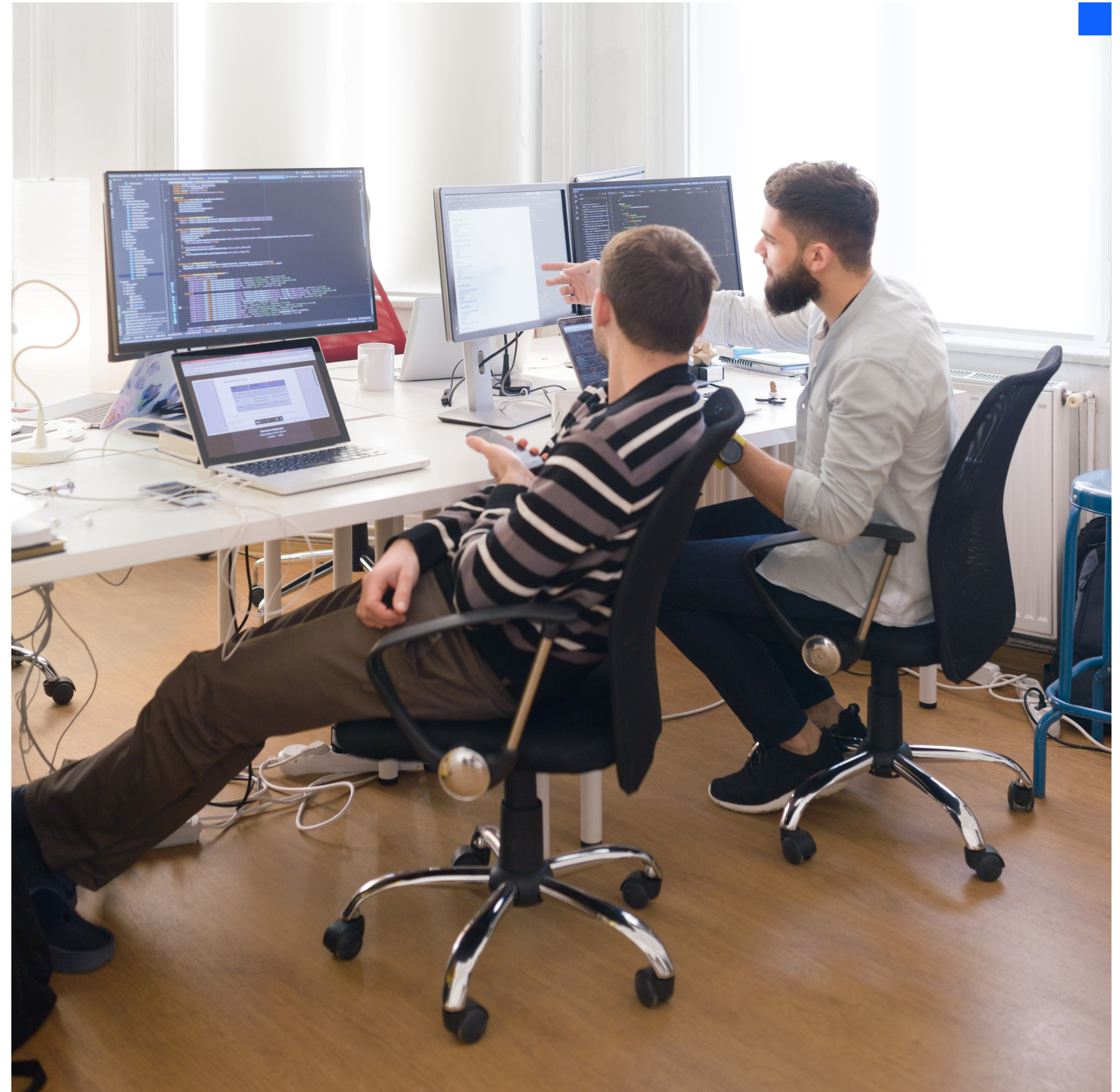


# Monitoring Apache Airflow with Prometheus, StatsD and Grafana



# Contents

01 →

Introduction

02 →

Building an open source  
Airflow monitoring solution

03 →

Prometheus, StatsD Exporter  
and metrics mapping

04 →

Airflow configuration

05 →

Configuring Grafana  
dashboards

06 →

Conclusion



# Introduction

To debug health problems or find the root cause of failures, a data engineer hops between the Apache Airflow UI, directed acyclic graph (DAG) logs, various monitoring tools and Python code.

It doesn't have to be this way.

You can use operational dashboards to get a bird's-eye view of the system, clusters and overall health.

In this guide, we'll explore the best practices for taking the open source route to building an operational dashboard.

The guide's goal is to easily answer questions such as:

- Is our cluster alive?
- How many DAGs do we have in a bag?
- Which operators succeeded and which failed lately?
- How many tasks are running right now?
- How long did it take for the DAG to complete?

# Building an open source Airflow monitoring solution

We'll need to configure a data observability dashboard. There are two routes you can take when looking for a data observability solution: an open source solution or a managed one. Both have advantages and disadvantages.



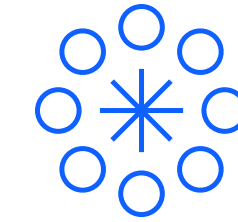
## Open source

### Pros

- Lower initial cost: your only cost for implementation is labor.
- Community support: get contributions from the global community.

### Cons

- Higher long-term cost: maintenance and troubleshooting can become difficult as your needs become more complex.
- Usability can be difficult: as your data team grows, ease of use, scalability and governance can become hard to manage.



## Managed service

### Pros

- Greater usability: better UI and automation can make your team more efficient.
- Better support: dedicated support is standing by.

### Cons

- Higher initial costs: the pricing model might not make sense for some organizations.
- Less flexibility: unless the managed service is built on open source code, functionality can be limited.

# Building an open source Airflow monitoring solution

For this guide, we will focus on monitoring and visualizing Airflow Cluster metrics. These types of metrics are great indicators of cluster and infrastructure health and should be constantly monitored.

Airflow exposes metrics such as DAG bag size, number of currently running tasks and task duration time every moment the cluster is running. You can find a list of all the different metrics exposed, along with descriptions, in the [official Airflow documentation](#).

By taking advantage of the trio below, you can find out whenever the scheduler is running, how many DAGs are in a bag, and most other critical problems in the cluster's health.

## 1. StatsD

We'll start with StatsD, a widely used service for collecting and aggregating metrics from various sources. Airflow has built-in support for sending metrics into the StatsD server. Once configured, Airflow will then push metrics to the StatsD server and we'll be able to visualize them.

## 2. Prometheus

Prometheus is a popular solution for storing metrics and alerting. Because it's typically used to collect metrics from other sources such as relational database management systems (RDBMSes) and web servers, we'll use Prometheus as the main storage for our metrics. And because Airflow doesn't have an integration with Prometheus, we'll use Prometheus StatsD Exporter to collect metrics and transform them into a Prometheus-readable format. StatsD Exporter acts as a regular StatsD server, and Airflow won't respond any differently to it.

## 3. Grafana

Grafana is our preferred metrics visualization tool. It has native Prometheus support and we'll use it to set up our Airflow Cluster monitoring dashboard.

## Let's start!

The basic architecture of our monitoring solution will look like this:

Airflow Cluster → StatsD Exporter → Prometheus → Grafana

Airflow Cluster reports metrics to StatsD Exporter, which performs transformations and aggregations and passes them to Prometheus. Grafana then queries Prometheus and displays everything in a dashboard. But first, we'll need to set up all of those pieces.

We'll configure Airflow, then StatsD Exporter and then Grafana.

# Prometheus, StatsD Exporter and metrics mapping

This guide assumes you're already familiar with Prometheus. If you aren't yet, Prometheus has great documentation and is easy to get started with because it doesn't require special configuration.

StatsD Exporter will be used to receive metrics and provide them to Prometheus. Usually it doesn't require much configuration, but because of the way Airflow sends metrics, we'll need to remap them.

By default, Airflow exposes a lot of metrics. For convenience, these metrics should be properly mapped. By using mapping, we can then build Grafana dashboards with per-Airflow instances and per-DAG views.

Let's take `dag.<dag_id>.<task_id>.duration` metric, for example.

The raw metric name sent by Airflow will look like `airflow_dag_sample_dag_dummy_task_duration`. For each Airflow instance and DAG you have, it will report duration for each task, producing a combinatorial explosion of the metrics. For simple DAGs, it's not an issue. But when tasks add up, things start being more complicated, and you wouldn't want to bother with a Grafana configuration.

To solve this, StatsD Exporter provides a built-in relabeling configuration. You can find great documentation and examples of these on the StatsD Exporter page.

Now let's apply this to our DAG duration metric.

The relabel config will look like this:

```
mappings:
- match: ".*.dag.*.*.duration"
  match_metric_type: observer
  name: "af_agg_dag_task_duration"
  labels:
    airflow_id: "$1"
    dag_id: "$2"
    task_id: "$3"
```

Figure 1. Relabel config

We are extracting three labels from this metric:

1. Airflow instance ID, which should be different across the instances
2. DAG ID
3. Task ID

Prometheus will then take these labels and we'll be able to configure dashboards with instance/DAG/task selectors and provide observability on different detailization levels.

We will repeat relabeling config for each metric exposed by Airflow.

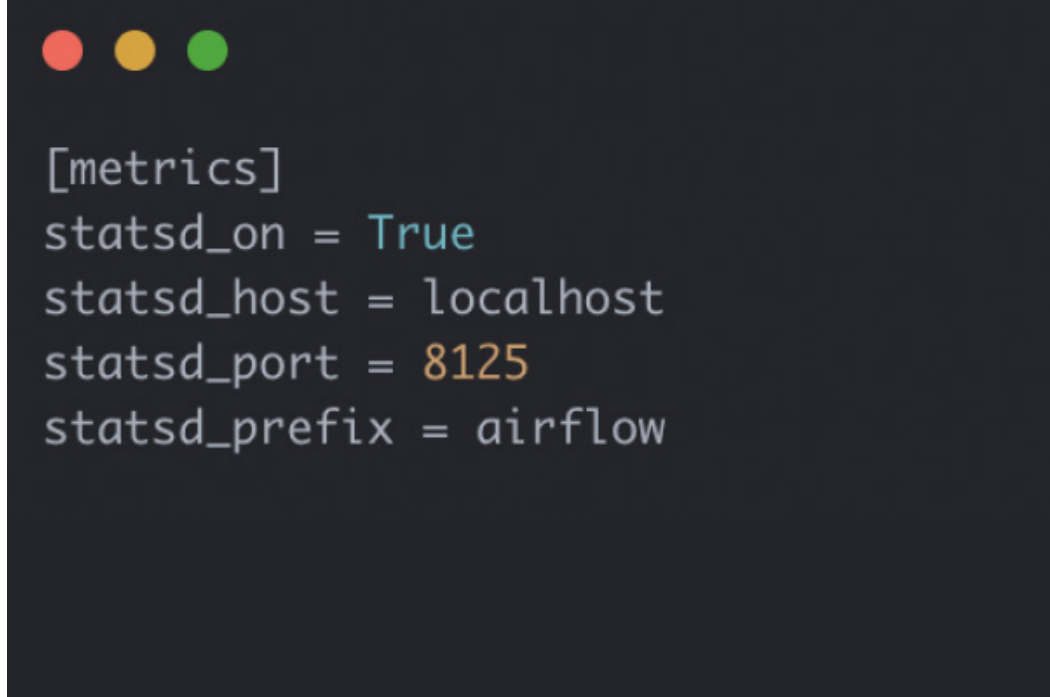
# Airflow configuration

A couple of options should be added to `airflow.cfg`. Please note that Airflow will fail to start if the StatsD server isn't available at the start-up time. Make sure you have a StatsD Exporter instance up and running.

For more details, refer to Airflow metrics documentation.

“A constant challenge is ensuring my data engineers have a good contract with data scientists and know how to take products from them and smoothly integrate them into the system. Even with pods, it's not always smooth.”

**Data engineering  
team lead**



```
[metrics]
statsd_on = True
statsd_host = localhost
statsd_port = 8125
statsd_prefix = airflow
```

Figure 2. The very basic config section in `airflow.cfg`

# Configuring Grafana dashboards

When we have all our metrics properly mapped, we can proceed to creating the dashboards. We'll have two dashboards—one for cluster overview and another for DAG metrics.

For the first dashboard, we'll have the Airflow instance selector:

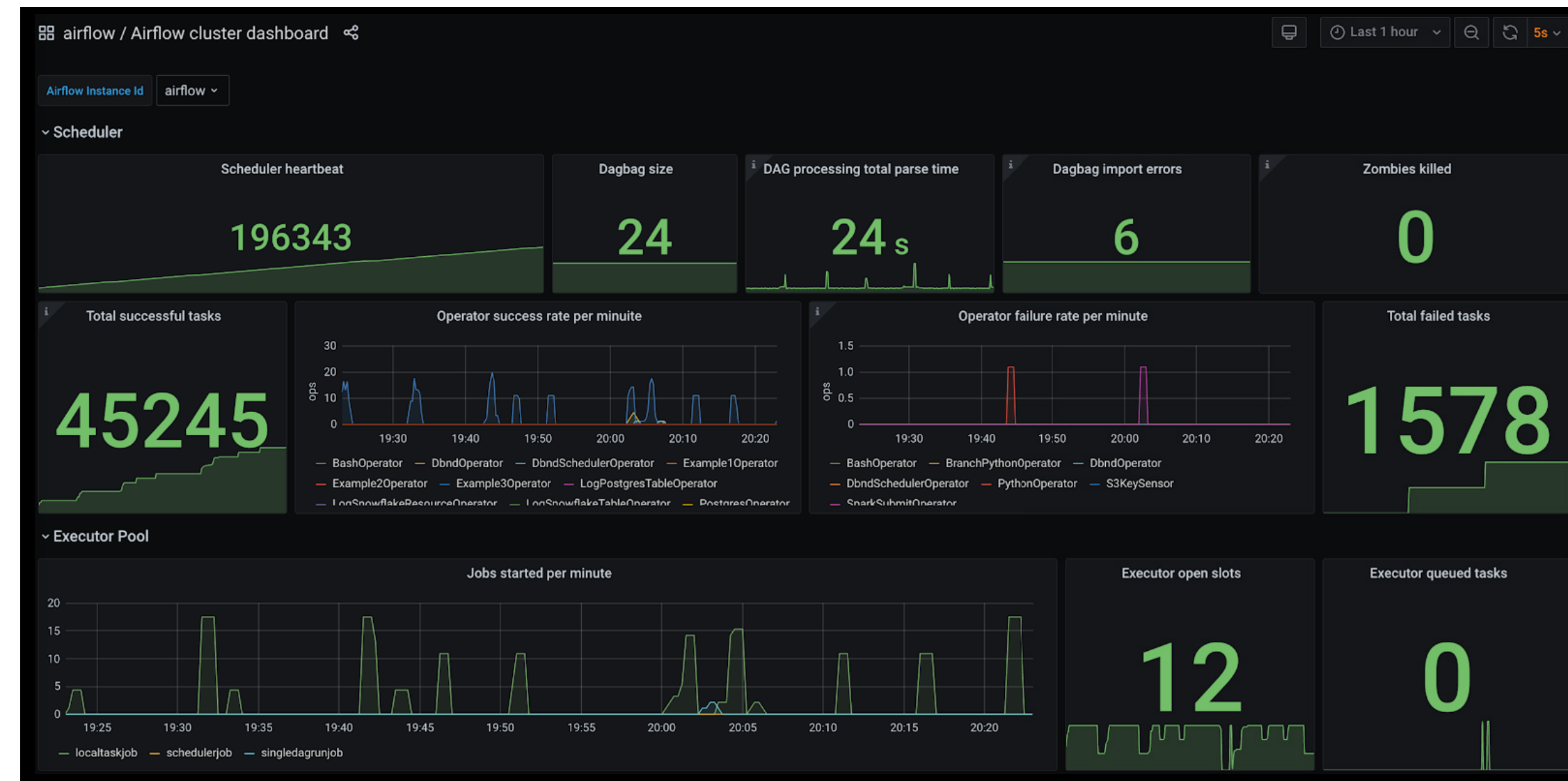


Figure 3. Airflow Grafana dashboard



# Configuring Grafana dashboards

You can see all vital metrics here, such as scheduler heartbeat, dagbag size, queued/running tasks count, currently running DAGs aggregated by tasks, and so on:

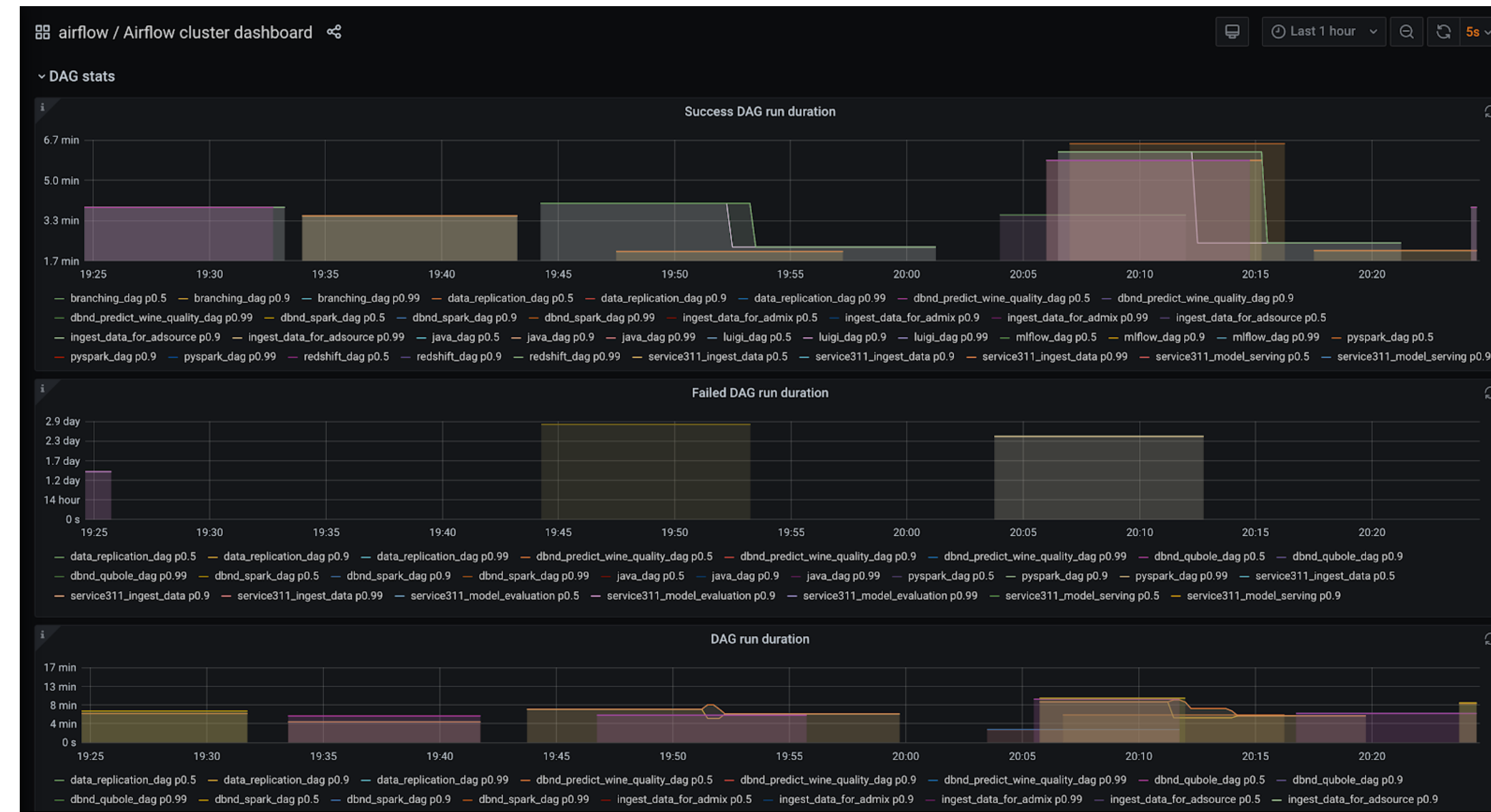


Figure 4. Airflow Grafana DAG view

# Configuring Grafana dashboards

For the second dashboard, we'll have the DAG selector.

You can see DAG-related metrics—successful DAG run duration, failed DAG run duration, DAG run dependency check time and DAG run schedule delay:

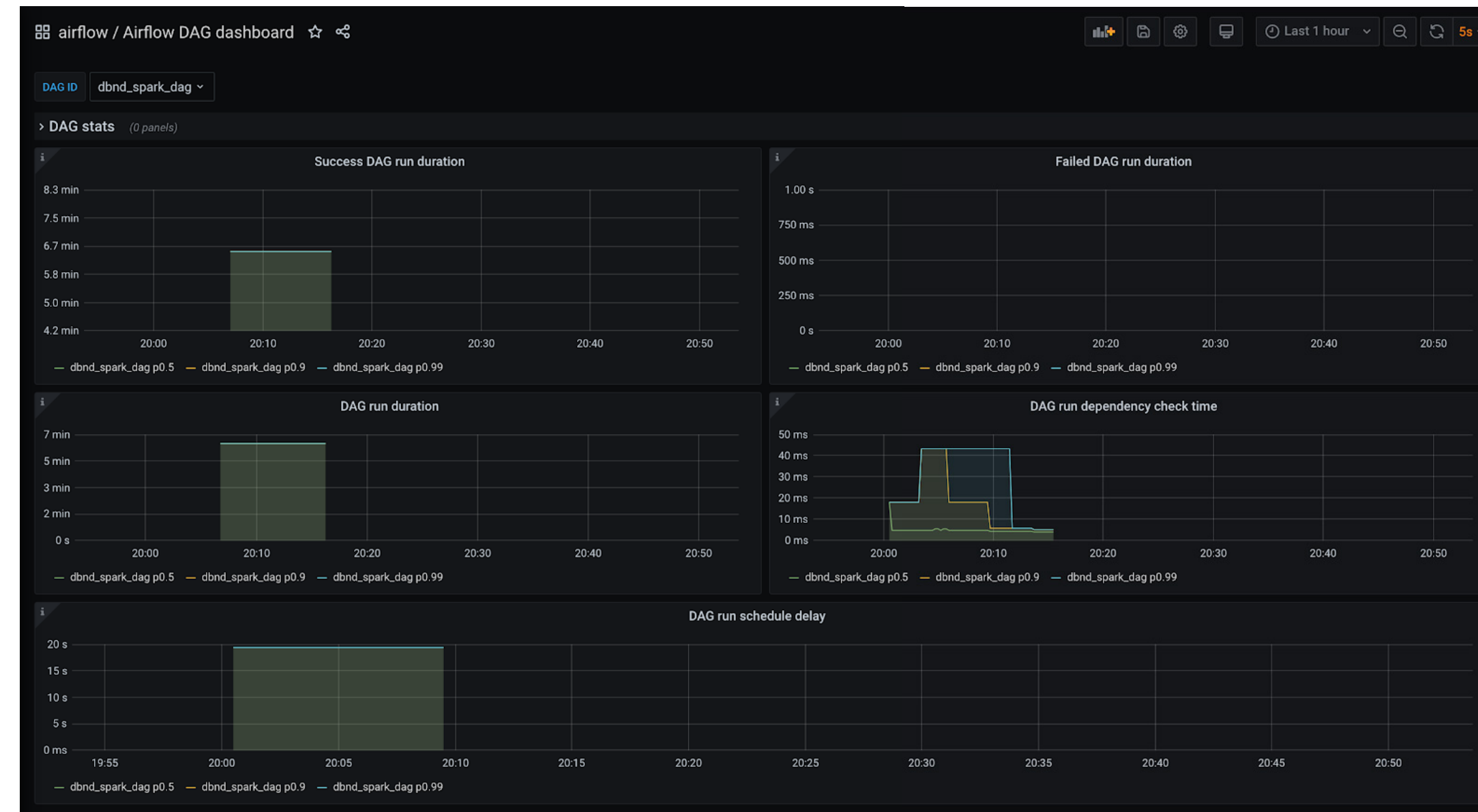


Figure 5. Airflow Grafana DAG status

## Conclusion



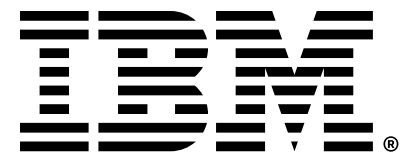
Airflow provides a decent out-of-the-box solution for monitoring DAGs, but it lacks accessibility and comprehensiveness. In this tutorial, we configured Airflow, StatsD Exporter and Grafana to get useful dashboards. Dashboards like these can save a lot of time when troubleshooting cluster health issues such as executors being down or DAG parsing being stuck because it has some heavyweight database query. For more robust and convenient monitoring, alerts should also be configured, but this is beyond the scope of the current guide.

**Better data quality starts at ingestion**  
Data engineers are the backbone of modern data teams. But for the average data engineer, it's a challenge to make sure jobs are running successfully, data is meeting quality standards, and business stakeholders are satisfied. For companies that depend on accurate, on-time data flows, that's a huge problem. Databand, an IBM Company, was built to help data engineers scale their infrastructure alongside their organization while maintaining data health standards.

Make big data observability manageable.

### Why IBM?

IBM® Databand delivers trusted data to your business. Learn more about how [IBM Databand](#) can help your organization automatically observe dynamic data pipelines, promote data quality and reliability, and continuously monitor AI and machine learning reliability.



© Copyright IBM Corporation 2023

IBM Corporation  
New Orchard Road  
Armonk, NY 10504

Produced in the United States of America  
January 2023

IBM and the IBM logo are trademarks or registered trademarks of International Business Machines Corporation, in the United States and/or other countries. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on [ibm.com/trademark](http://ibm.com/trademark).

This document is current as of the initial date of publication and may be changed by IBM at any time. Not all offerings are available in every country in which IBM operates.

THE INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT ANY WARRANTY, EXPRESS OR IMPLIED, INCLUDING WITHOUT ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND ANY WARRANTY OR CONDITION OF NON-INFRINGEMENT. IBM products are warranted according to the terms and conditions of the agreements under which they are provided.