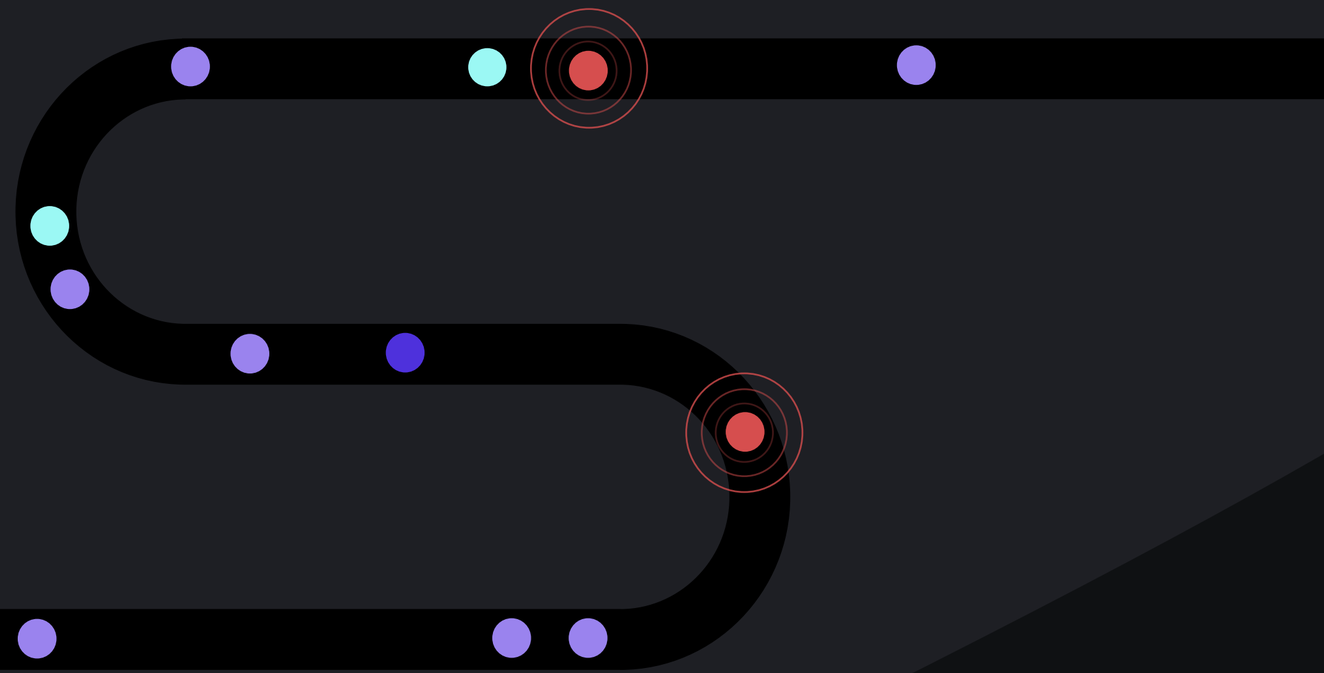
Databand

Everyday Data Engineering:

Monitoring Airflow with Prometheus, StatsD and Grafana



Everyday Data Engineering: Databand's series –
A guide on common data engineering activities,
written for data engineers by data engineers.

Databand

Monitoring Airflow can be painful.

In order to debug health problems or find the root cause of failures, a data engineer needs to hop between the Apache Airflow UI, 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 our system, clusters, and overall health.

In this guide, we’ll be exploring the best practices for going the open-source route to building an operational dashboard.

This guide’s goal is to easily answer questions like:

- *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

Databand

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. There are advantages and disadvantages to both, namely:

1 Open-source

Pros

- Lower initial cost—
  - Your only cost for implementation is labor.
- Community support—
  - Contribution 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.

2 Managed Service

Pros

- Greater usability—
  - Better UI and automation can make your team more efficient.
- Better support—
  - Dedicated support 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 leveraging this trio, you can find out whenever the scheduler is running, how many DAGs are in a bag now, or most other critical problems in the cluster's health.

Databand

1 StatsD

We'll start with StatsD. StatsD is 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 will be able to visualize them.

2 Prometheus

Prometheus is a popular solution for storing metrics and alerting. Because it is typically used to collect metrics from other sources, like RDBMSes and webservers, we will use Prometheus as the main storage for our metrics. 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 notice any difference between them.

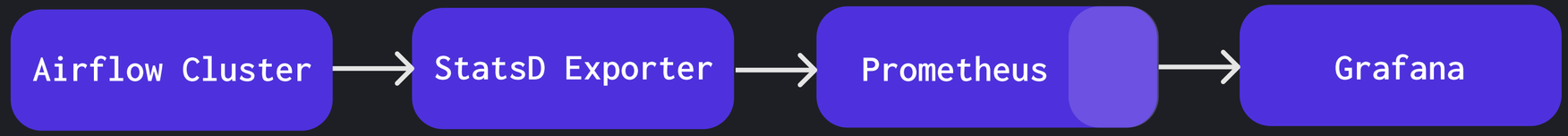
3 Grafana

Grafana is our preferred metrics visualization tool. It has native Prometheus support and we will use it to set up our Airflow Cluster Monitoring Dashboard.

Databand

Let’s start!

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



Airflow Cluster reports metrics to StatsD Exporter which performs transformations and aggregations and passes them to Prometheus. Grafana then queries Prompetheus and displays everything in a gorgeous dashboard. In order for this to happen, we will need to set up all of those pieces.

First, we will configure Airflow, then StatsD Exporter and then Grafana.

Databand

Prometheus, StatsD Exporter and metrics mapping

This guide assumes you're already familiar with Prometheus. If you aren't yet, it has great documentation and is really easy to get started with as Prometheus doesn't require special configuration. StasD 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 will need to re-map them.

By default, Airflow exposes a lot of metrics, which labels are composed from DAG names. For convenience, these metrics should be properly mapped. By utilizing 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 you have and for each DAG you have, it will report duration for each Task producing combinatorics 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 Grafana configuration.

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

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

Prometheus, StatsD Exporter and metrics mapping

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"
```

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 detalization levels.

We will repeat re-labeling config for each metric exposed by Airflow. See "Source Code" section at the end of the article for a complete example.

Airflow configuration

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

The very basic config section in airflow.cfg will look like this:

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

For the more details, please refer to Airflow Metrics documentation.

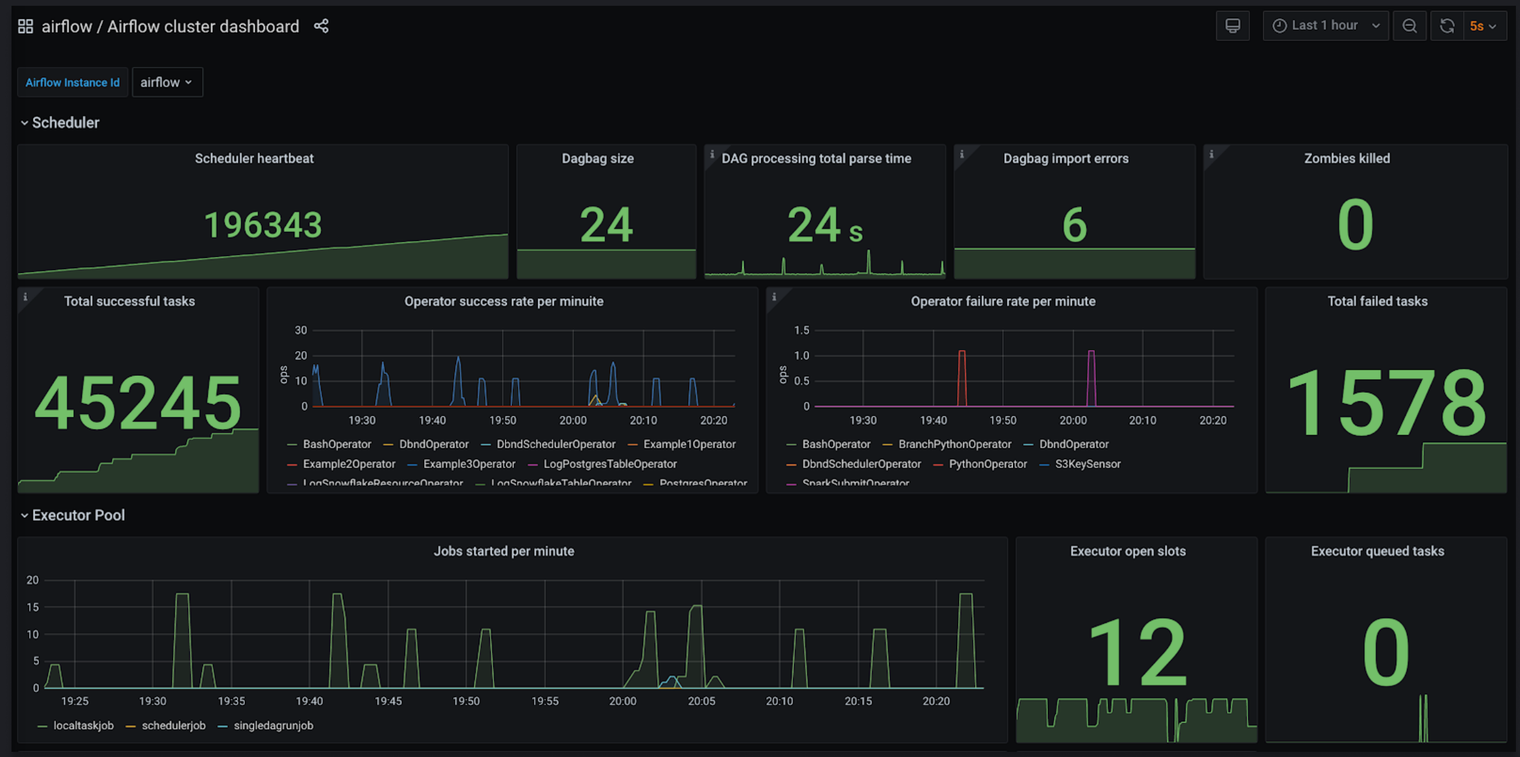
> "A constant challenge is ensuring my data engineers have a good contract with data scientists and know how to take products from them and smoothly integrate them into the system. Even with pods, it's not always smooth."
>
> -Data Engineering Team Lead

Databand

Configuring Grafana Dashboards

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

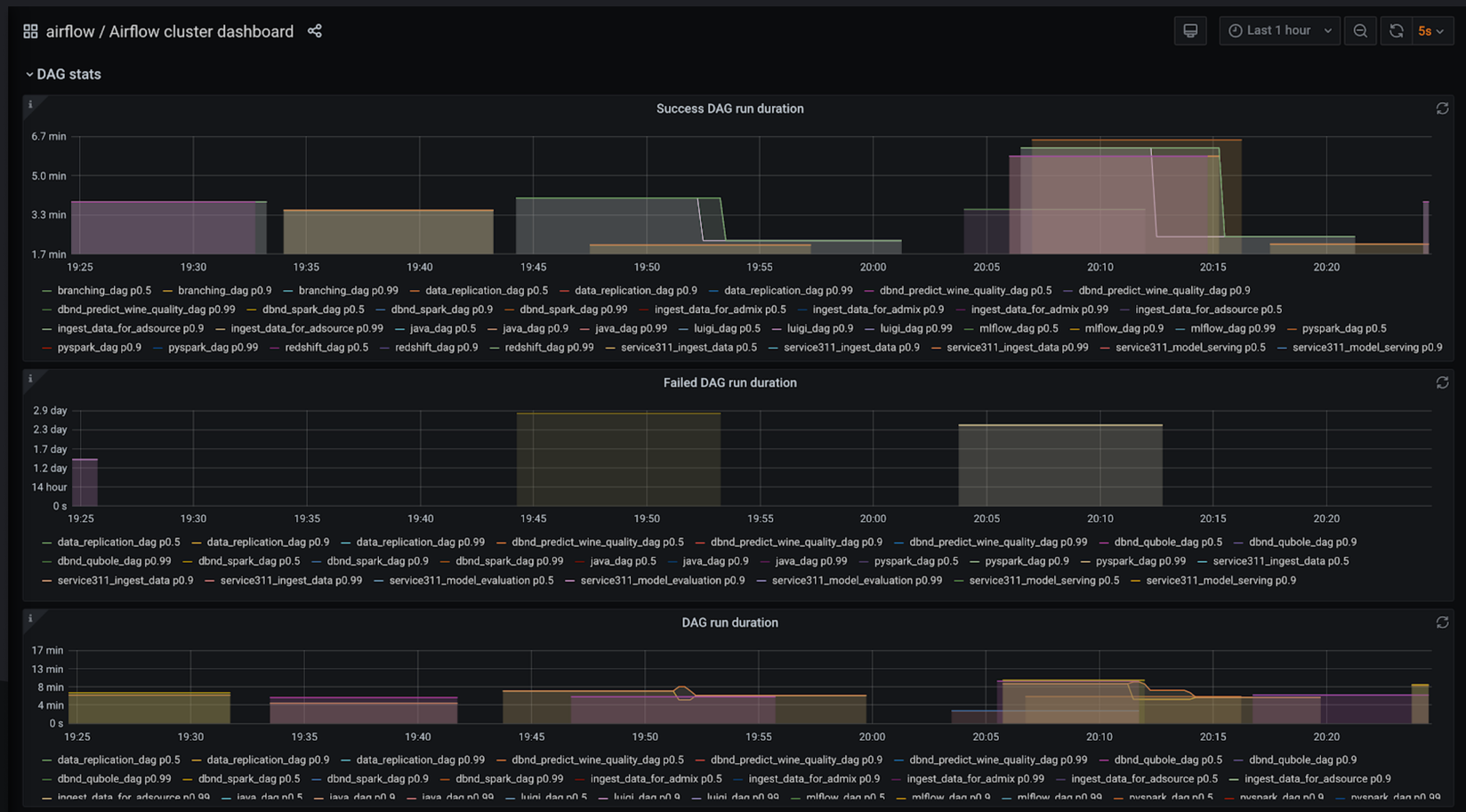
For the first dashboard we will have the Airflow instance selector:



Databand

Configuring Grafana Dashboards

You can see here all vital metrics: like scheduler heartbeat, dagbag size, queued/running tasks count, currently running DAGs aggregated by tasks etc:

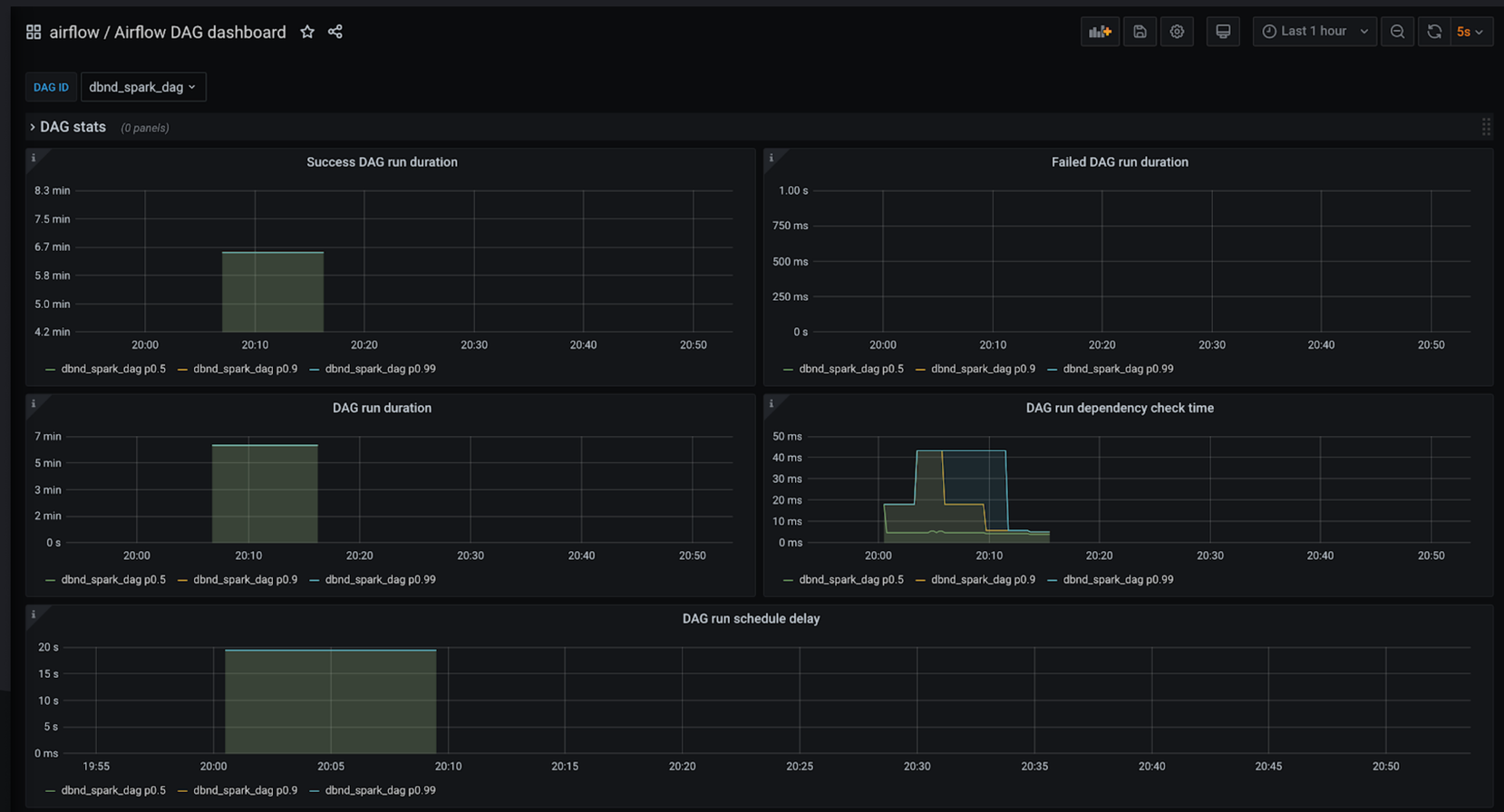


Databand

Configuring Grafana Dashboards

For the second dashboard we will have the DAG selector:

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



Conclusion

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

Stay tuned for more guides!

Happy engineering! 🙂

Source Code

Complete source code including StatsD mapping config, two Grafana dashboards—one for Cluster overview and another for DAG stats, can be found in our GitHub project: https://github.com/databand-ai/airflow-dashboards/

Databand

www.databand.ai

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 who depend on accurate, on-time data flows, that's a huge problem. We built Databand to help data engineers scale their infrastructure alongside their organization while maintaining data health standards.

Make big data observability manageable.

Learn more about Databand.ai