

BI Tools for Data Engineers: Apache Superset and Metabase

How open-source business intelligence tools connect to a data platform, and the decisions a data engineer owns: modelling for BI, where metrics live, performance, access control, embedding and operations.

Last reviewed 28 Sep 2026 ·

Prerequisites: [SQL](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/sql-reference/) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/sql-reference/) · [Data Modeling](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/data-modeling/) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/data-modeling/) · [Semantic Layer & Metrics](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/semantic-layer-metrics/) (https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/semantic-layer-metrics/)

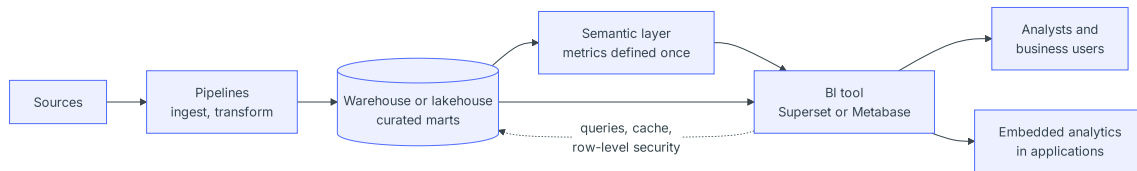
Related: [dbt](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/dbt-reference/) (https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/dbt-reference/) · [Snowflake](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/) · [BigQuery](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/bigquery-reference/) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/bigquery-reference/) · [Real-Time Analytics Databases](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/realtime-olap/) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/realtime-olap/) · [Governance & Lineage](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/) (https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/) · [Docker](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/docker-reference/) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/docker-reference/) · [Glossary](https://sarangambekar1997.github.io/data-engineering-handbook/99-reference/glossary/) (https://sarangambekar1997.github.io/data-engineering-handbook/99-reference/glossary/)

Overview

Challenge: A data platform is only useful when people can use its data. Business users want dashboards and self-service questions, product teams want charts inside their own applications, and analysts want to explore without writing a ticket. If each team builds its own reports on raw tables, numbers disagree, queries are slow and expensive, and access to sensitive rows is hard to control.

Solution: A BI tool is the consumption layer. It connects to the warehouse or lakehouse, presents tables as datasets or models people can explore, and turns queries into charts and dashboards. Apache Superset and Metabase are the two most widely used open-source options. They differ in style (Superset is a configurable, engineering-oriented

platform, Metabase is built for self-service) but they solve the same problem, and the data engineer's job is largely the same for both: give them well-modelled data, decide where business logic lives, keep queries fast, control access, and operate the service.



Relevance to data engineering: BI is where data quality problems and modelling shortcuts become visible to the business. A slow dashboard is usually a modelling or query problem, a wrong number is usually a definition that lives in the wrong place, and a data leak is usually an access rule that was set in the wrong layer. Understanding how these tools work helps you prevent all three.

On this page

Basic - [BI in the Data Platform](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#bi-in-the-data-platform) (https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#bi-in-the-data-platform) - [Superset and Metabase at a Glance](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#superset-and-metabase-at-a-glance) (https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#superset-and-metabase-at-a-glance) - [Preparing Data for BI](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#preparing-data-for-bi) (https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#preparing-data-for-bi)

Intermediate - [Apache Superset](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#apache-superset) (https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#apache-superset) - [Metabase](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#metabase) (https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#metabase) - [Where Business Logic Lives](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#where-business-logic-lives) (https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#where-business-logic-lives)

Advanced - [Performance](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#performance) (https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#performance) - [Access Control and Row-Level Security](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#access-control-and-row-level-security) (https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#access-control-and-row-level-security) - [Embedding Analytics in Applications](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#embedding-analytics-in-applications) (https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#embedding-analytics-in-applications) - [Operating a BI Service](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#operating-a-bi-service) (https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#operating-a-bi-service) - [Choosing a Tool](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#choosing-a-tool) (https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#choosing-a-tool)

Reference - [Common Pitfalls](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#common-pitfalls) (https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#common-pitfalls) - [Cheat Sheet](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#cheat-sheet) (https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#cheat-sheet) - [Interview Questions](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#interview-questions) (https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#interview-questions) - [Further Reading](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#further-reading) (https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/bi-tools/#further-reading)

BI in the Data Platform

A BI tool does not store your data. It stores its own **metadata** (definitions of datasets, charts, dashboards, users and permissions) in an **application database**, and it sends SQL to your data source when a chart loads. That has two consequences that shape everything else.

- **Every chart is a query against your warehouse.** A dashboard with twelve charts sends twelve queries each time it loads, for every viewer. Cost, concurrency and latency are properties of your warehouse and your models, not of the BI tool.
- **Two databases matter.** The **data source** is your warehouse or lakehouse. The **application database** holds the BI tool's own state. Losing the second means losing every dashboard, so it needs the same care as any production database.

Term	Meaning
Data source	The warehouse, lakehouse or database the BI tool queries
Application (metadata) database	Where the BI tool stores dashboards, users, permissions and settings
Dataset or model	A table or query presented to users as something to explore
Chart (question)	A saved query with a visualisation
Dashboard	A collection of charts with shared filters
Cache	Stored query results, so repeated loads do not hit the warehouse

Superset and Metabase at a Glance

	Apache Superset	Metabase
Style	Configurable platform for data and BI engineers	Simple, self-service for a broad audience
Licence	Apache 2.0	Open-source edition under the AGPL. Some features are in paid plans under a commercial licence

	Apache Superset	Metabase
Runs as	A Python (Flask) web application plus optional workers	A single JAR, or a container
Application database	PostgreSQL or MySQL recommended. SQLite by default, which is not recommended for production	PostgreSQL recommended. An embedded H2 database by default, which is not for production
Data layer	Datasets (physical tables or virtual SQL)	Models (curated tables), and saved questions
Optional components	A cache such as Redis, Celery workers and a beat scheduler	None required
Caching	Flask-Caching, with Redis recommended	Duration, schedule and adaptive policies, on paid plans
Row-level security	Built-in filters attached to datasets and roles	Row and column security on paid plans
Embedding	Embedded SDK with guest tokens	Several options, some limited on the open-source edition
Current release (Sep 2026)	6.1.0	0.63.x

Neither tool is universally better. Superset gives more control over configuration, visualisation types and SQL exploration, and asks more of the team that runs it. Metabase gets a non-technical user to a first answer quickly, and its most advanced permissions and caching features are paid.

Preparing Data for BI

Whichever tool you pick, the work that matters most happens before it.

- **Point BI at curated marts, not raw tables.** A mart at the right grain, with clear column names, joins already done and business rules applied, makes charts fast and consistent. See [Data Modeling](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/data-modeling/) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/data-modeling/).
- **State the grain** of each table in its description, so nobody sums a column at the wrong level.

- **Use a read-only service account** for the BI connection, restricted to the schemas BI needs, with a query timeout and a warehouse or workload of its own so dashboards cannot starve pipelines.
- **Pre-aggregate expensive results.** A daily summary table is far cheaper to chart than the raw events beneath it.
- **Name things for humans.** Column names such as `net_revenue_usd` beat `nr_amt`, and descriptions are shown in the tools.
- **Document freshness.** People assume a dashboard is real time. Tell them when it last updated. See [Pipeline Observability](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/pipeline-observability/) (https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/pipeline-observability/).

Apache Superset

Architecture

Superset has two required components and several optional ones.

Component	Role	Required
Superset application	The Flask backend, API and React frontend. It serves visualisations and sends SQL to the data source	Yes
Metadata database	Stores chart and dashboard definitions, users and logs. PostgreSQL and MySQL are the tested options	Yes
Cache	Usually Redis. Stores query results so a chart loaded twice is served from the cache the second time, and acts as the message broker for workers	Optional, but needed for several features
Celery workers and beat	Workers run background tasks (async queries, report snapshots, emails) and the beat is the scheduler	Optional, but needed for Alerts and Reports, async queries and thumbnails

Docker Compose and Kubernetes installations provision all of these. A plain PyPI installation gives only the application and needs the rest to be configured by hand.

Configuration

Superset reads a Python module that overrides its defaults. Point to it with `SUPERSET_CONFIG_PATH`.

```
# superset_config.py
import os

# Superset will not start without a SECRET_KEY. Generate
one with: openssl rand -base64 42
SECRET_KEY = os.environ["SUPERSET_SECRET_KEY"]

# The metadata database: PostgreSQL (10.x to 17.x) or MySQL
(5.7, 8.x). Not SQLite in production.
SQLALCHEMY_DATABASE_URI =
os.environ["SUPERSET_METADATA_URI"] # e.g.
postgresql://user:pass@host/superset

FEATURE_FLAGS = {
    "EMBEDDED_SUPERSET": True,      # needed for embedded
dashboards (see below)
}
```

```
export SUPERSET_CONFIG_PATH=/app/superset_config.py
export SUPERSET_SECRET_KEY=$(openssl rand -base64 42)
```

The `SECRET_KEY` is used to sign session cookies and to encrypt sensitive information. Keep it in a secret manager, and do not change it casually, since values encrypted with the old key may no longer be readable. Load configuration from environment variables so the same file works in every environment.

Datasets, charts and dashboards

Superset works with **datasets**, which are either **physical** (a table) or **virtual** (a saved SQL query). Charts are built on a dataset, and dashboards combine charts with shared filters. Because a virtual dataset runs its SQL as a subquery every time, keep them simple and push heavy logic into the warehouse as a view or table.

Caching and asynchronous work

Superset uses Flask-Caching for its caches. Redis is the recommended backend, and other options include Memcached, an in-memory cache, S3-compatible storage and the local filesystem. Caching serves two roles: it stores query results, and it is the message broker for the Celery workers. SQL Lab runs queries asynchronously only if you enable asynchronous query execution on the database connection, and SQL Lab query results then use a configured results backend.

Metabase

Architecture

Metabase runs as a single application, either a JAR file or a container. For the JAR, the documentation recommends a **Java 25** runtime (Eclipse Temurin) and states that earlier Java versions are not supported. It listens on port 3000.

```
java --add-opens java.base/java.nio=ALL-UNNAMED -jar
metabase.jar
```

```
docker run -d -p 3000:3000 --name metabase
metabase/metabase
```

By default it stores its own state in an embedded **H2** database on the local filesystem. This is fine for a trial and not for production: H2 is an on-disk database that is sensitive to filesystem errors, so without backups it can be corrupted and you can lose every question, dashboard and collection. It also cannot be shared between instances, so you cannot run several Metabase instances behind a load balancer with it. In a container it lives inside the container, so removing the container removes the data. For production, use **PostgreSQL** as the application database:

```
docker run -d -p 3000:3000 \
  -e "MB_DB_TYPE=postgres" \
  -e "MB_DB_DBNAME=metabaseappdb" \
  -e "MB_DB_PORT=5432" \
  -e "MB_DB_USER=name" \
  -e "MB_DB_PASS=password" \
  -e "MB_DB_HOST=my-database-host" \
  --name metabase metabase/metabase
```

Take these credentials from your secret manager, not from a script in Git. Migrate to a production application database before you have content worth losing.

Models

A Metabase **model** is a curated data layer: it curates data from one or more tables of the same database, anticipating the questions people will ask. A model lets an administrator set display names, descriptions, column types and visibility, and it appears in search so people reuse it instead of starting from raw tables. It can be built with the query builder, where

Metabase fills in the metadata, or with SQL, where you configure the metadata yourself. Models can be **persisted** for faster loading. Treat models as the interface between your marts and the business, and keep the logic in the warehouse, with the model adding names and descriptions.

Permissions

Permissions are granted to **groups**, not people. A person in several groups gets the most permissive access across all of them, so an over-broad group cannot be undone by adding someone to a stricter one. There are two kinds:

Kind	Controls
Data permissions	Access to databases and tables: viewing data, creating queries, downloading results, managing the database
Collection permissions	Access to saved questions, dashboards, models and other content

Row and column security, database routing (sending the same question to a different database depending on the customer) and connection impersonation (using roles you define in your database) are for multi-tenant setups and are available on paid plans.

Caching

Metabase caches query results using **duration**, **schedule** or **adaptive** policies, which can be set at four levels with priority from the most specific: question, dashboard, database and the whole instance. These policies are limited to paid plans. Whatever your plan, the most reliable performance work happens in the warehouse.

Where Business Logic Lives

The most consequential design decision is where a metric is defined. There are three places, and they are not equal.

Where	How it works	Trade-off
In the BI tool (calculated columns, saved questions, virtual datasets)	Logic is written in the tool	Fast to build, but definitions are hidden in dashboards, differ between tools and are hard to test or review

Where	How it works	Trade-off
In the warehouse (tables and views built by dbt or similar)	Logic is versioned SQL that produces a mart	Reviewed, tested and shared by every consumer. The BI tool reads a simple table
In a semantic layer (metrics defined once, queried by name)	One definition serves BI tools, notebooks and applications	The strongest consistency, and it needs the layer to be supported. See Semantic Layer & Metrics (https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/semantic-layer-metrics/)

A sound default: **put business logic in the warehouse or a semantic layer, and keep the BI tool to presentation**. A BI tool that only selects, filters, groups and formats can be replaced or supplemented without redefining "revenue". When two dashboards disagree, the cause is nearly always a definition that lives in two places.

Keep the link between the two visible. dbt **exposures** record which dashboards depend on which models, so a change to a model shows who is affected, and lineage tools can pull that into a catalogue. See [dbt](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/dbt-reference/) (<https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/dbt-reference/>) and [Governance & Lineage](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/) (<https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/>).

Performance

A slow dashboard is a query problem. In order of impact:

- 1. Model for the question.** Point charts at a mart or a summary table at the right grain, not at a raw events table.
- 2. Reduce the data scanned.** Partition or cluster large tables on the columns dashboards filter on, and make sure a dashboard's default filter (usually a date range) uses them. See [BigQuery](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/bigquery-reference/) (<https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/bigquery-reference/>) and [Snowflake](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/) (<https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/>).
- 3. Pre-aggregate.** Build daily or hourly summary tables in the pipeline, and chart those.
- 4. Cache.** Use the BI tool's cache for dashboards that many people open, remembering that a cached chart is as stale as its cache lifetime. Superset's cache works with Redis, and Metabase's policies are on paid plans.
- 5. Limit the work per load.** Fewer charts per dashboard, sensible default filters, and no `SELECT *` from wide tables.
- 6. Isolate the workload.** Give BI its own warehouse, queue or resource group, with a query timeout, so a heavy dashboard cannot slow the pipelines or other users.

7. **Use a faster engine where it is justified.** For dashboards that need sub-second answers over fresh event data, serve them from a real-time analytics database. See [Real-Time Analytics Databases](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/realtime-olap/) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/realtime-olap/).

Measure before you change anything: the warehouse's query history shows which dashboard queries are slow, how much data they scan and how often they run.

Access Control and Row-Level Security

Decide **where** each rule is enforced, because a rule set only in the BI tool can be bypassed by anyone with direct database access.

Layer	Enforces	Best for
Warehouse (roles, row access policies, masking)	Who can read which tables, rows and columns, for every tool	Sensitive data and compliance. The strongest guarantee
BI tool (groups, collections, row-level security)	Who can see which dashboards, datasets and rows within the tool	Content access, and convenient row filtering where the warehouse cannot
Application (embedding with signed tokens)	What an embedded viewer can see	Customer-facing analytics

Superset row-level security attaches a filter clause to a dataset and to a set of roles. The clause is added to the WHERE clause of the SQL that Superset generates for that dataset, so members of that role only ever see the matching rows. **Metabase** offers row and column security on paid plans, and connection impersonation, which uses roles you define in your database, keeps the rule in the warehouse. See [Data Security & Privacy](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/data-security-privacy/) (https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/data-security-privacy/).

Whatever the tool, follow least privilege. Give the BI connection read-only access to only the schemas it needs, give users the lowest role that works, review who is in privileged groups, and remember that, in Metabase, the most permissive group wins.

Embedding Analytics in Applications

Embedding puts charts and dashboards inside your own application for your customers. The main risk is exposing the wrong data, so the design goal is that **the viewer's access is decided by your backend, not by the browser.**

Superset

Enable the `EMBEDDED_SUPERSET` feature flag, install the embedded SDK (`@superset-ui/embedded-sdk`) in the front end, and mount the dashboard using its embed ID. The browser needs a **guest token**, which your backend requests from Superset using a service account. Never request it from client-side code.

```
import requests

SUPERSET = "https://superset.example.com"

def guest_token_for(customer_department: str, dashboard_id: str, service_token: str) -> str:
    response = requests.post(
        f"{SUPERSET}/api/v1/security/guest_token/",
        headers={"Authorization": f"Bearer {service_token}"},
        json={
            "user": {"username": "guest_user",
                    "first_name": "Guest", "last_name": "User"},
            "resources": [{"type": "dashboard", "id": dashboard_id}],
            "rls": [{"clause": f"department = '{customer_department}'"}], # see the warning below
        },
        timeout=10,
    )
    response.raise_for_status()
    return response.json()["token"]
```

The token carries the dashboard it is allowed to show and the row-level security rules to apply, and it expires: `GUEST_TOKEN_JWT_EXP_SECONDS` defaults to 300 seconds, so your application should refresh it before then. Restrict the **allowed domains** in each dashboard's embed settings, since an empty list allows any origin. For cross-site embedding, session cookies need `SESSION_COOKIE_SAMESITE = "None"` and `SESSION_COOKIE_SECURE = True`, and the Talisman `frame_ancestors` setting must list the host site.

Warning: The example builds a SQL clause from a value. Never insert unvalidated user input into a clause. Take the value from your own authenticated session or a lookup, and validate it against a known list. Otherwise a crafted value can widen the filter or inject SQL.

Metabase

Metabase offers several ways to embed: **guest embedding** for view-only charts and dashboards, **modular embedding** of individual components with a drop-in script, a **modular embedding SDK** for React, and **full app embedding** in an iframe with your data permissions. Guest embeds are view-only, so viewers cannot drill through. On the open-source edition you can embed components only without single sign-on, and the full integration with permissions is a paid feature, so check the plan's limits before designing around a feature.

Operating a BI Service

A BI tool is a production service that people rely on.

- **Back up the application database**, and test a restore. It holds every dashboard, and for Metabase's default H2 database a corrupted file is the main way to lose everything.
 - **Use a managed PostgreSQL** for the application database, and run more than one instance of the application behind a load balancer where availability matters.
 - **Secrets:** the Superset `SECRET_KEY` and the data source credentials belong in a secret manager. Rotate the data source credentials on a schedule.
 - **Authentication:** connect single sign-on so accounts follow your identity provider, and remove access when people leave.
 - **Versions:** pin the version, read the release notes before upgrading, and upgrade a staging copy first. Metabase's Java requirement, for example, has changed between releases.
 - **Dashboards as code:** where the tool can export dashboards and datasets to files, keep them in Git and promote them from a development instance to production through CI, instead of editing production by hand. See [Testing and CI/CD](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/testing-cicd/) (<https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/testing-cicd/>).
 - **Monitor:** watch the error rate, slow queries, cache hit rate, and the age of the data behind each dashboard.
 - **Deploy with containers**, for repeatability. See [Docker](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/docker-reference/) (<https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/docker-reference/>) and [Kubernetes](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/kubernetes-for-de/) (<https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/kubernetes-for-de/>).
-

Choosing a Tool

If	Consider
You have engineers to run it and want control over configuration, many chart types and SQL exploration	Superset
You want non-technical users answering their own questions quickly, with little setup	Metabase
You need row-level security, advanced caching or full embedding with SSO in Metabase	A paid Metabase plan, or Superset, which includes row-level security in the open-source project
You need a managed, governed enterprise BI tool with a built-in semantic layer	A commercial platform such as Looker, Tableau or Power BI
Your dashboards need sub-second answers over fresh event data	Serve them from a real-time analytics database, whichever BI tool fronts it
You already have a BI tool the business uses	Keep it, and improve the models and metrics beneath it

Evaluate on a real dataset and your ten most important dashboards: query performance on your warehouse, the row-level security you need, the embedding you need, licence terms (the AGPL in particular matters for some organisations), and how much operating effort your team can afford. The choice of tool matters less than the quality of the models beneath it.

Common Pitfalls

Pitfall	Symptom	Fix
BI pointed at raw tables	Slow dashboards, inconsistent numbers	Curated marts and summary tables
Business logic inside dashboards	Two dashboards show different revenue	Define logic in the warehouse or a semantic layer
Default application database in production	Lost dashboards after a disk error or a container removal	PostgreSQL for the application database, with backups

Pitfall	Symptom	Fix
Metabase H2 behind a load balancer	Cannot run more than one instance	A shared PostgreSQL application database
Superset started without a <code>SECRET_KEY</code>	It refuses to start	Generate one with <code>openssl rand -base64 42</code> and store it as a secret
A shared, over-privileged BI service account	A leak exposes the whole warehouse	Read-only access to the schemas BI needs, with timeouts
Row-level security only in the BI tool	Direct database access bypasses it	Enforce sensitive rules in the warehouse
Metabase group permissions that overlap	A user has more access than intended	The most permissive group wins, so review memberships
Guest token requested from the browser	Anyone can craft a token	Request it from the backend with a service account
Building a row-level security clause from raw input	SQL injection or a widened filter	Validate against a known list, or use a lookup
No allowed domains on an embedded Superset dashboard	Any site can embed it	Set allowed domains, <code>frame_ancestors</code> and cookie settings
Over-long cache lifetimes	People act on stale numbers	Match the cache to the data's freshness and show when it updated
Dashboards with dozens of charts	Slow loads and warehouse cost	Fewer charts, pre-aggregated tables, sensible default filters
Editing production dashboards by hand	No history, no review, no rollback	Export to files, review in Git and promote through CI
Not knowing which dashboards use a model	A model change breaks reports	dbt exposures and lineage
Assuming an open-source feature is available in every edition	A missing feature late in a project	Check the plan and licence for caching, row-level security and embedding first

Cheat Sheet

Task	Superset	Metabase
Run	Docker Compose or Kubernetes (provision all components), or PyPI	<code>docker run -d -p 3000:3000 --name metabase metabase/metabase · java -jar metabase.jar</code>
Required runtime	Python application plus a metadata database	Java 25 for the JAR
Application database	<code>SQLALCHEMY_DATABASE_URI</code> (PostgreSQL or MySQL)	<code>MB_DB_TYPE=postgres</code> with <code>MB_DB_*</code> variables
Secret	<code>SECRET_KEY (openssl rand -base64 42)</code>	Application database credentials from a secret store
Config file	<code>superset_config.py</code> via <code>SUPERSET_CONFIG_PATH</code>	Environment variables
Curated layer	Datasets (physical or virtual)	Models (persistable)
Cache	Redis via Flask-Caching	Duration, schedule and adaptive policies (paid)
Background work	Celery workers and beat	Built in
Row-level security	<u>RLS</u> filters on datasets and roles	Row and column security (paid)
Embed	<code>EMBEDDED_SUPERSET</code> and a guest token from your backend	Guest, modular, SDK or full app embedding
Token lifetime	<code>GUEST_TOKEN_JWT_EXP_SECONDS</code> (default 300)	Per embedding method
Default port	Set by the install method	3000
Licence	Apache 2.0	AGPL (open source), commercial for paid plans

Rule: curated marts in the warehouse · metrics in one place · read-only BI account · enforce sensitive rules in the warehouse · back up the application database

Interview Questions

Q: What are the two databases involved when you run a BI tool, and why does the distinction matter? A: The data source, which is your warehouse or lakehouse, is queried whenever a chart loads, so dashboard cost and speed are properties of your models and warehouse. The application database holds the BI tool's own state: dashboards, users and permissions. It needs to be a production database with backups, because losing it means losing every dashboard. For Metabase the default embedded H2 database is not suitable for production, and Superset's default SQLite is not recommended either.

Q: Where should business metrics be defined, and why? A: In the warehouse or a semantic layer, not in individual dashboards. Logic there is versioned, reviewed and tested, and every consumer shares one definition, so two dashboards cannot show different revenue. The BI tool then handles presentation. Definitions inside a BI tool are hidden in charts, differ between tools and are hard to test.

Q: A dashboard is slow. How do you investigate and fix it? A: Look at the warehouse query history to see which queries the dashboard sends, how much data they scan and how long they take. Then fix in order of impact: point charts at a mart or pre-aggregated table at the right grain, partition or cluster on the filter columns, cut the number of charts and set a default date filter, cache what many people open, and isolate BI on its own warehouse with timeouts. For sub-second needs over fresh events, use a real-time analytics database.

Q: How do you implement row-level security for an embedded Superset dashboard? A: Enable the embedded feature flag and use the embedded SDK in the front end, and have the backend request a guest token from Superset's API using a service account, naming the dashboard and passing row-level security clauses. The token expires, 5 minutes by default, so the application refreshes it. I would set allowed domains, configure cookies and the frame ancestors for the host site, and build clauses only from validated values, never from raw user input.

Q: What are the limits of enforcing security in the BI tool? A: A rule enforced only in the BI tool applies only to people using that tool. Anyone with direct database access, or another tool on the same connection, bypasses it. For sensitive data I enforce rules in the warehouse with roles, row access policies and masking, and use the BI tool's groups and row-level security for content access and convenience. Connection impersonation keeps the rules in the database.

Q: How would you choose between Superset and Metabase? A: On the team and the requirements. Superset suits an engineering team that wants configuration control, many chart types and SQL exploration, and it includes row-level security in the open-source project, but it has more components to run. Metabase gets non-technical users to answers quickly with a simple deployment, but caching policies, row and column security and full embedding with single sign-on are on paid plans. I would trial both on real data and dashboards, and weigh licence terms, embedding needs and operating effort.

Further Reading

- [Apache Superset documentation](https://superset.apache.org/) and its [architecture](https://superset.apache.org/admin-docs/installation/architecture/)
- [Configuring Superset](https://superset.apache.org/admin-docs/configuration/configuring-superset/) and [caching](https://superset.apache.org/admin-docs/configuration/cache/)
- [Embedding Superset](https://superset.apache.org/user-docs/using-superset/embedding/)
- [Metabase documentation](https://www.metabase.com/docs/latest/) and [running the JAR](https://www.metabase.com/docs/latest/installation-and-operation/running-the-metabase-jar-file)
- [Metabase: migrating to a production application database](https://www.metabase.com/docs/latest/installation-and-operation/migrating-from-h2)
- [Metabase: models](https://www.metabase.com/docs/latest/data-modeling/models), [permissions](https://www.metabase.com/docs/latest/permissions/introduction) and [caching](https://www.metabase.com/docs/latest/configuring-metabase/caching)

Previous: [Semantic Layer & Metrics](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/semantic-layer-metrics/) • **Next:** [Data Quality](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/data-quality/) • **Back to:** [Index](https://sarangambekar1997.github.io/data-engineering-handbook/)

 September 28, 2026