Grafana Dashboards

DATA 503: Fundamentals of Data Engineering

Lucas P. Cordova, Ph.D.

2026-04-08

Introduces PostgreSQL views on the class practice database (same dashboard-ready definitions as the full Dashboards lecture), then a short path through Grafana on Railway aligned with the pipeline dashboard-service reading: template deploy, least-privilege users, data source setup, and (from the addendum) persisting Grafana metadata in Postgres. No panel-by-panel demo; see Lecture 13-2 for the full Grafana lab.

Table of contents

1 Learning objectives

1. Place Grafana in the stack: read-only analytics over Postgres, not a replacement for ETL
2. Explain why **views** and a **read-only role** are the right boundary for dashboard queries
3. Outline the Railway **template** steps and key **environment variables** for a Grafana service

4. Contrast the **dashboard-service** setup steps with the **addendum**: operational data vs Grafana metadata, persistence, and public URLs

. . .

Course connection: This is the serving layer: curated SQL becomes charts and tables stakeholders can refresh without touching pipelines.

2 Views

2.1 What a view is

A **view** is a **named** `SELECT` stored in the catalog. Clients read it like a table, but it does not duplicate rows on disk unless you use a **materialized** view.

For dashboards, views are the usual **contract**: one stable name, one definition of joins and aggregates, reused by Grafana and by anyone running SQL.

2.2 Same dataset as the Salem 13 SLM track

The **13-1 Advanced Queries (SLM)** deck restores `class_dbs` (local or Railway) with Census, business, temperature, and survey tables. After restore, you can `CREATE OR REPLACE VIEW` objects that match the analytical questions you want charts to answer.

The **named views** below are the same **dashboard-ready** definitions used in the full **Dashboards with Grafana (13-2)** lecture. They are intentionally shaped for bar charts, tables, and grouped comparisons.

2.3 Base tables (after restore)

Table	Role
`us_counties_pop_est_2019`	County population and migration components
`cbp_naics_72_establishments`	Food and accommodation establishments by area
`temperature_readings`	Daily readings for weather stations
`ice_cream_survey`	Long-format survey responses

2.4 Catalog of curated views

View	Idea	Panel-friendly shape
`v_state_population_summary`	State totals and natural increase	Sorted bars, tables
`v_state_migration_rates`	Net migration per thousand + direction label	Horizontal bar, color by category
`v_county_growth_summary`	County counts grouped by growth category	Pie, stat
`v_state_establs_per_capita`	NAICS 72 establishments per thousand residents	Ranked bars
`v_seasonal_temperatures`	Avg temps by station and season	Grouped bar
`v_population_tiers`	Metro to rural tiers	Table, shares

2.5 Example: `v_state_population_summary`

State-level aggregates from county rows. Natural **bar** or **table** panels.

```sql
CREATE OR REPLACE VIEW v_state_population_summary AS
SELECT
    state_name,
    count(*) AS num_counties,
    sum(pop_est_2019) AS total_pop,
    round(avg(pop_est_2019), 0) AS avg_county_pop,
    sum(births_2019) AS total_births,
    sum(deaths_2019) AS total_deaths,
    sum(births_2019) - sum(deaths_2019) AS natural_increase
FROM us_counties_pop_est_2019
GROUP BY state_name
ORDER BY total_pop DESC;
```

2.6 Example: `v_state_migration_rates`

Adds a **CTE**, a rate per thousand, and a **CASE** label for color or legend splits in Grafana.

```sql
CREATE OR REPLACE VIEW v_state_migration_rates AS
WITH state_migration AS (
    SELECT
        state_name,
        sum(international_migr_2019 + domestic_migr_2019) AS net_migration,
        sum(pop_est_2018) AS base_pop
    FROM us_counties_pop_est_2019
    GROUP BY state_name
)
SELECT
    state_name,
    base_pop,
    net_migration,
    round((net_migration / base_pop::numeric) * 1000, 2) AS migration_rate_per_thousand,
    CASE
        WHEN net_migration > 0 THEN 'Gaining'
        WHEN net_migration < 0 THEN 'Losing'
        ELSE 'Stable'
    END AS migration_direction
FROM state_migration
ORDER BY migration_rate_per_thousand DESC;
```

. . .

Full SQL for the other catalog views appears in **13-2 Dashboards**. Grant `SELECT` on these views to a read-only role, not necessarily every base table.

3 Where Grafana sits

3.1 Pipeline role

Grafana **reads** your database (usually through views). It does not run your batch jobs.

3.2 Two different uses of Postgres

Database role	What lives there
Pipeline / analytics	Facts, dimensions, **views** you chart
Grafana app config (optional)	Dashboard JSON, users, data source definitions

Keeping those separate means you can reload pipeline data without wiping dashboard UI work, once Grafana is configured to store metadata in its own database (addendum).

4 Reading: dashboard-service

4.1 What that page is for

The **dashboard-service** write-up is a **checklist** from "data in Postgres" to "first live panels":

- Introduce **views** as the stable interface for visualization (hide joins, control permissions)
- Deploy **Grafana on Railway** from a **template**
- Lock down access with a **Grafana-specific database user**
- Register **PostgreSQL as a Grafana data source** and build a first dashboard

That checklist assumes you already publish **views** (or tightly scoped queries) as the Grafana-facing surface, as in the **Views** section above.

4.2 Add Grafana from a Railway template (high level)

These steps match the **template-based** path in the reading (search the template gallery for **Grafana**, often published under community profiles such as **Andre Lademann**):

1. In your Railway project: **Create** then **Template**
2. Search for **Grafana** and select the template your materials reference
3. Set the **admin** credentials and any template-required variables, then **Deploy**

4.3 Environment variables (template checklist)

The reading suggests variables along these lines (exact names may match your template's prompts):

Key	Purpose
`GF_SECURITY_ADMIN_USER`	Initial Grafana admin username
`GF_SECURITY_ADMIN_PASSWORD`	Initial Grafana admin password
`GF_DEFAULT_INSTANCE_NAME`	Friendly name for the instance (if the template asks)

Leave template defaults for unrelated keys unless the template docs say otherwise. After variables validate, deploy and wait until the service is healthy.

4.4 Serverless on Grafana

After deploy, in Grafana **Settings**, enable **Serverless** if your platform offers it so the service **sleeps when idle** and wakes on demand. Useful for class projects with periodic use.

4.5 Least-privilege database user

In Postgres, create a user that can **only** read what dashboards need (often `SELECT` on specific **views**):

- `CREATE USER ...`
- `GRANT USAGE ON SCHEMA public ...`
- `GRANT SELECT ON` the view (or views), not carte blanche on all tables if you can avoid it

4.6 Connect Grafana to Postgres (high level)

1. From Railway, copy connection details: **host**, **port**, database name (often `railway`), and credentials
2. In Grafana: **Connections** then **Data sources** then **PostgreSQL**
3. Fill host, database, user, password; use the SSL mode Railway documents for your endpoint (**internal** vs **public** hostnames differ)
4. **Save and test**

4.7 First dashboard (no demo here)

At a high level: **Home** then create a **dashboard**, **add visualization**, pick the PostgreSQL source, choose a **table or view**, map time and value fields, **run query**, pick a visualization. The full **Lecture 13-2** deck walks panels step by step.

5 Reading: dashboard-service addendum

5.1 Why an addendum exists

Out of the box, Grafana may store state in a way that does not survive redeploys the way you expect on a PaaS. The **addendum** addresses two goals:

1. **Persistence:** Store Grafana's own configuration (dashboards, users, data source records) in **PostgreSQL**
2. **Sharing:** Serve Grafana at a stable **public URL** and use Grafana's **share** features for read-only links

5.2 Before you change storage

If you already built dashboards in a non-persistent setup, **export** dashboards (and optionally settings) as **JSON** before switching backends so you can re-import.

5.3 Addendum steps (high level)

1. **Create a dedicated database** on your Railway Postgres instance (commonly named `grafana`) for Grafana metadata only
2. **Set Grafana environment variables** so it uses Postgres as its database, for example:
   - `GF_DATABASE_TYPE=postgres`
   - `GF_DATABASE_HOST` (often the **internal** host and port from Railway when Grafana runs in the same project)
   - `GF_DATABASE_NAME=grafana`
   - `GF_DATABASE_USER` / `GF_DATABASE_PASSWORD` (service credentials from Postgres)
   - `GF_DATABASE_SSL_MODE` (often `disable` for internal Railway networking, or `require` if you use the public endpoint; follow Railway's connector docs)
   - `GF_SERVER_ROOT_URL` set to your Grafana **public HTTPS URL** (from the Deployments tab)
3. **Redeploy or restart** Grafana
4. **Verify:** create a test dashboard, restart Grafana, confirm the dashboard still exists
5. **Public link:** use Grafana's **Share** then **External link** flow if you need a link that works outside your account (understand the security implications)

. . .

Idea: Pipeline data stays in your main database; the **`grafana` database** holds only Grafana's application state.

6 Summary

Topic	Takeaway
Views	Named `SELECT` on `class_dbs`; catalog in **Views** section, full set in **13-2**
Template deploy	Railway **Create** then **Template** then **Grafana**, then configure admin env vars and deploy
Security	Read-only DB user scoped to views or minimal schema access
Data source	PostgreSQL connector in Grafana with correct host and SSL mode for your network path
Addendum	Separate **`grafana` database** and `GF_DATABASE_*` (plus `GF_SERVER_ROOT_URL`) for **persistent** Grafana and stable **public** access

6.1 References

1. Grafana documentation
2. Grafana PostgreSQL data source
3. Railway documentation
4. Course lecture: Dashboards with Grafana (13-2)
5. PostgreSQL CREATE VIEW
6. SLM restore context: Advanced Query Techniques (13-1 SLM)