GSoC’25 Proposal

PostgreSQL

Upgrade Grafana dashboards to v11

Basic Details:

- **Full Name:** Gaurav Patidar
- **College:** Indian Institute of Technology Kharagpur (IIT Kharagpur)
- **Location:** Kharagpur, West Bengal, India
- **Languages:** Hindi, English
- **Timezone:** Indian Standard Time (UTC+5:30)

Contact Information:

- **Primary Email:** gaurav05082002@gmail.com
- **Secondary Email:** gauravpatidar@kgpian.iitkgp.ac.in
- **Phone:** (+91) 7689816680
- **GitHub:** https://github.com/Gaurav05082002
- **LinkedIn:** https://www.linkedin.com/in/gaurav-patidar-809997207/
- **Resume:** Resume link

Availability:

UTC 02:30 to UTC 20:30 (IST 08:00 to IST 02:00 next day).
I can adjust my schedule by starting my day 2 hours early or late if it helps to communicate with other developers and mentors. I will be reachable anytime through my mobile number and email.

Abstract:

This project is focused on **upgrading the existing pgwatch Grafana dashboards** to be fully compatible with **Grafana version 11**. While some dashboards have already been partially migrated, many still rely on **outdated or deprecated components** and require **manual refinement**. With v11 removing **AngularJS support** and introducing changes to **panel JSON structure** and **transformation capabilities**, the goal is to update all dashboards to leverage the **new visualization features**, ensure full functionality, and enhance usability. Alongside these upgrades, the project will also deliver **clear documentation** to support **future maintenance and updates** of the dashboards.

My Background / Technical Skills:

I’m Gaurav Patidar, a final-year undergraduate student at the Indian Institute of Technology Kharagpur (IIT KGP), India. My journey in tech began during my 12th grade, when I built a website for my school using raw HTML and JavaScript — and that small project is what sparked my long-term interest in software engineering.

After cracking the JEE exam, I joined **IIT Kharagpur** where I formally studied **Data Structures**, **Algorithms**, **Operating Systems**, and **Machine Learning**, along with **full-stack web development**. Early on, I built fun side projects like a Sudoku solver in JavaScript and a mini e-commerce site using React. These helped me learn how to take an idea from scratch to production.

In my second year, I was selected for the **Tech Team of Spring Fest**, one of Asia’s largest student-run college festivals. I worked on the core tech stack using React, Node.js, and Python, helping to manage thousands of event registrations, payments, and merchandise orders. As Tech Lead, I was responsible for setting up and maintaining **AWS infrastructure**, **Linux-based servers**, **PostgreSQL databases**, and managing production downtime during high-traffic periods.

Over the past few years, I’ve interned with organizations like **Samagra (Govt of India)**, **Density Exchange**, and **Simpl**, all of which involved working on **large microservice-based architectures**, structured **Postman API workspaces**, and tools like **Grafana**, **Linux**, and **Docker**. For instance, I worked with PostgreSQL databases on production workloads, built real-time survey platforms, configured monitoring dashboards, and wrote multithreaded automation scripts that improved system efficiency. Most recently, I was placed as a Software Development Engineer **(SDE) at PhonePe** on Day 1 of campus placements.

I'm comfortable working across both backend and devops pipelines. I code in **Python**, **C++**, **Go**, **SQL**, and **JavaScript**, and use tools like **Docker**, **Git**, **AWS**, **MongoDB**, and **Databricks**. My web dev stack includes **React**, **Flask**, **Express**, and **Node.js**, and I’ve worked with technologies like **JWT**, **SMTP**, **REST APIs**, and **PySpark**.

What excites me most about this project is the chance to work deeply with **Grafana dashboards**, dive into the **JSON modeling layer**, and handle **PostgreSQL metrics visualization** in a production-grade monitoring system like **pgwatch**. Having used Grafana and PostgreSQL in both internship and open-source work, I’m eager to improve their integration and contribute to more intuitive, modern dashboards using Grafana v11 features.

Outside of coursework, I’m an active contributor in the **open-source space** — through **C4GT (Digital Public Infrastructure)** by Samagra, **Hacktoberfest**, and the **Amazon AI Dev Hackathon**, where my team ranked in the top 10. With 500+ commits and PRs, I’m excited to bring my experience and commitment into Google Summer of Code (GSoC) this year.

Goals:

Here are the primary goals of this project:

- **Identify and replace deprecated AngularJS components** in existing Grafana dashboards to ensure full compatibility with Grafana v11.

- **Update and refactor panel JSON structures** to align with Grafana's new v11 schema, including updated field configurations, visual options, and panel types.

- **Integrate enhanced transformation capabilities** by applying Grafana v11's new data processing features such as field overrides, transformations, and dynamic thresholds.

- **Polish dashboard layouts and improve user experience**, ensuring consistency across panels, meaningful color schemes, tooltips, legends, and unit formatting.

- **Test and validate each upgraded dashboard** by generating live PostgreSQL workload data and verifying that all graphs and metrics render correctly without errors.

- **Export and organize the upgraded dashboards** under a dedicated `v11/` folder structure for PostgreSQL and Prometheus, maintaining version clarity.

- **Document the entire migration process**, outlining common issues, fixes, panel migration examples, and guidelines for future upgrades and contributions.

Implementation :

Upgrading the pgwatch Grafana dashboards is divided into the following MileStones.
I have upgraded some dashboards I will be using them as reference examples to explain the overall approach and methodology.
(All dates mentioned below are of 2025)

- ### Community Bonding Period (May 8 to June 1)

  - I will **actively engage with the community**, connect with my **mentors and fellow contributors**, and gain a deeper understanding of ongoing initiatives within the ecosystem.
  - I will walk my mentors through the setup and my approach, and **present the dashboards I have already upgraded** to seek feedback and suggestions for enhancing the methodology.
  - I will discuss the **JSON definitions** that I used to incorporate new v11 features, and check if there is a need for any modifications or improvements.
  - Align on the **documentation approach** for the migration process and plan guidelines for future dashboard updates.

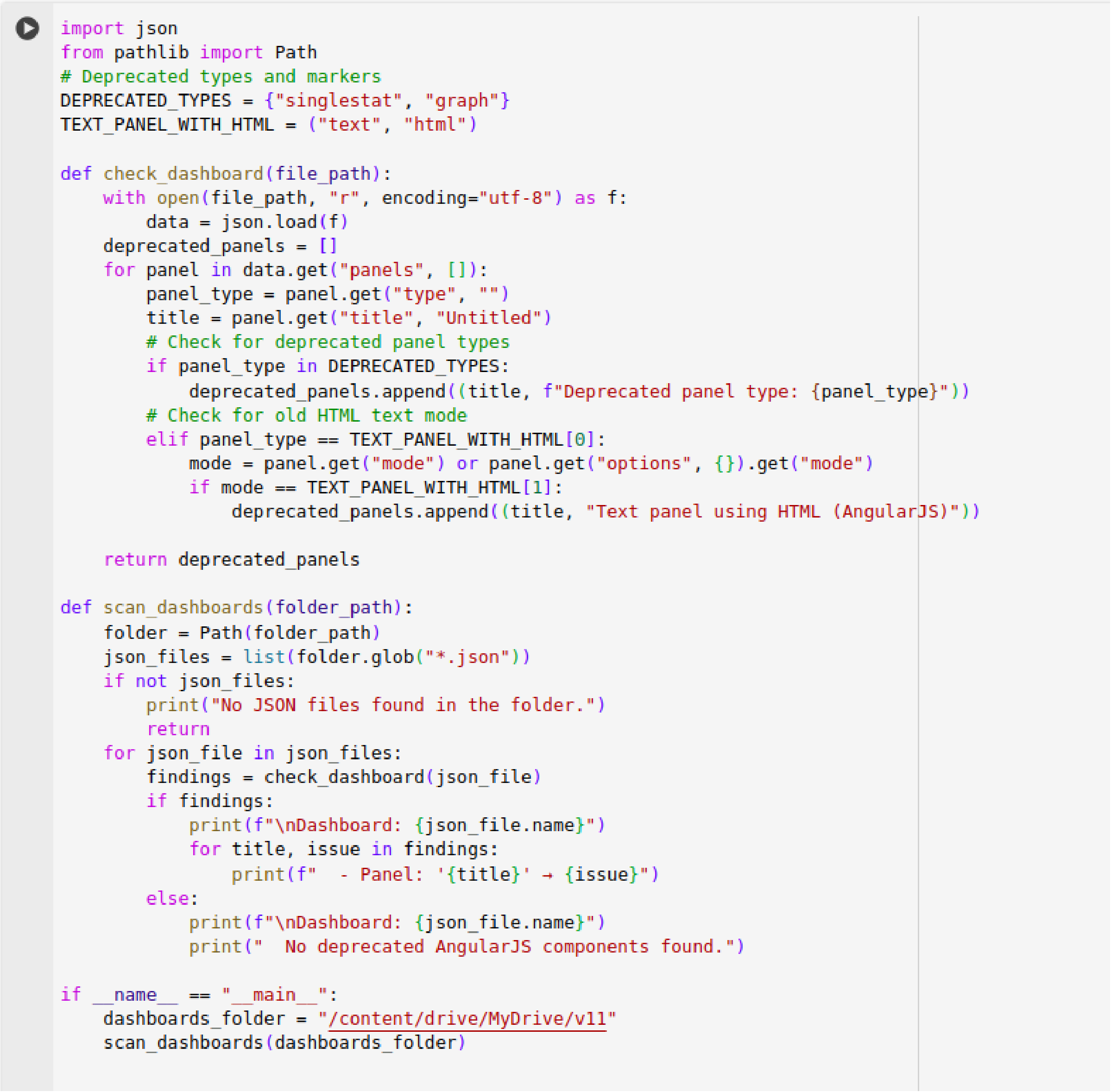
- Finalize the project **timeline and milestones** in consultation with my mentor, adjusting scope and deliverables as needed.

- **Milestone 1 (June 2 to June 16) - Detection and Replacement of Deprecated AngularJS-Based Components from v11 dashboards**

  - One of the key architectural changes in **Grafana v11** is the **complete removal of AngularJS-based components**, which were still supported in Grafana v10.
  - Although some dashboards have been automatically migrated, there is still a need to **review and validate** each dashboard to ensure **no deprecated components remain**.
  - Below is a table summarizing the **deprecated AngularJS components** and their respective **modern replacements** in Grafana v11.

Deprecated AngularJS Component	**Recommended Replacement for Grafana v11**
Text panel using HTML (AngularJS)	Text panel (mode: Markdown) or Stat panel if numeric
Deprecated panel type: `singlestat`	Stat panel
Deprecated panel type: `graph`	Time series panel
Use of yaxes, lines, fill properties	Use fieldConfig and options in panel JSON structure
Custom HTML in old panels	Rewrite using Markdown or transparent Stat panels with tooltips
Legacy transformations inside queries	Use Grafana transformations like merge, reduce, add field

  - To streamline this validation process, I **developed a Python script** that scans multiple dashboard JSON files and identifies any **legacy AngularJS-based panels or configurations**. The script logs each deprecated usage along with its associated panel title, enabling a systematic review and cleanup.



```python
import json
from pathlib import Path
# Deprecated types and markers
DEPRECATED_TYPES = {"singlestat", "graph"}
TEXT_PANEL_WITH_HTML = ("text", "html")

def check_dashboard(file_path):
    with open(file_path, "r", encoding="utf-8") as f:
        data = json.load(f)
    deprecated_panels = []
    for panel in data.get("panels", []):
        panel_type = panel.get("type", "")
        title = panel.get("title", "Untitled")
        # Check for deprecated panel types
        if panel_type in DEPRECATED_TYPES:
            deprecated_panels.append((title, f"Deprecated panel type: {panel_type}"))
        # Check for old HTML text mode
        elif panel_type == TEXT_PANEL_WITH_HTML[0]:
            mode = panel.get("mode") or panel.get("options", {}).get("mode")
            if mode == TEXT_PANEL_WITH_HTML[1]:
                deprecated_panels.append((title, "Text panel using HTML (AngularJS)"))

    return deprecated_panels

def scan_dashboards(folder_path):
    folder = Path(folder_path)
    json_files = list(folder.glob("*.json"))
    if not json_files:
        print("No JSON files found in the folder.")
        return
    for json_file in json_files:
        findings = check_dashboard(json_file)
        if findings:
            print(f"\nDashboard: {json_file.name}")
            for title, issue in findings:
                print(f"  - Panel: '{title}' → {issue}")
        else:
            print(f"\nDashboard: {json_file.name}")
            print("  No deprecated AngularJS components found.")

if __name__ == "__main__":
    dashboards_folder = "/content/drive/MyDrive/v11"
    scan_dashboards(dashboards_folder)
```

Img > Python code to check the angular js components in dashboards (JSON)

The following are the results of the script run on Grafana PostgreSQL v11 dashboards that were automatically migrated.
https://github.com/cybertec-postgresql/pgwatch/tree/master/grafana/postgres/v11

```
Dashboard: db-overview.json
 - Panel: 'Untitled' → Text panel using HTML (AngularJS)

Dashboard: single-query-details.json
 - Panel: 'Untitled' → Text panel using HTML (AngularJS)

Dashboard: show-plans-realtime.json
 - Panel: '' → Text panel using HTML (AngularJS)


Dashboard: system-stats-time-lag.json
```

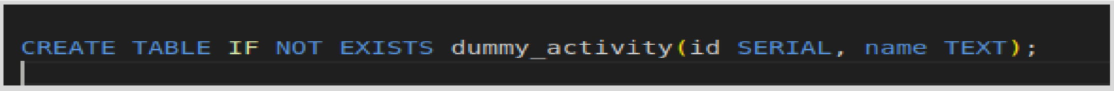
```
 - Panel: 'CPU utilization' → Deprecated panel type: graph
 - Panel: 'IO Wait' → Deprecated panel type: graph
 - Panel: 'Memory used (%)' → Deprecated panel type: graph
 - Panel: 'Memory available' → Deprecated panel type: graph
 - Panel: 'Swap used (%)' → Deprecated panel type: graph
 - Panel: 'Total bytes read per second' → Deprecated panel type: graph
 - Panel: 'Total bytes written per second' → Deprecated panel type:
graph

Dashboard: health-check.json
 - Panel: 'Instance state' → Deprecated panel type: singlestat
 - Panel: 'Instance uptime' → Deprecated panel type: singlestat
 - Panel: 'PG version num.' → Deprecated panel type: singlestat
 - Panel: 'Longest query runtime' → Deprecated panel type: singlestat
 - Panel: 'Active connections' → Deprecated panel type: singlestat
 - Panel: 'Max. connections' → Deprecated panel type: singlestat
 - Panel: 'Blocked sessions' → Deprecated panel type: singlestat
 - Panel: 'Shared Buffers hit pct.' → Deprecated panel type:
singlestat
 - Panel: 'TX rollback pct. (avg.)' → Deprecated panel type:
singlestat
 - Panel: 'TPS (avg.)' → Deprecated panel type: singlestat
 - Panel: 'QPS (avg.)' → Deprecated panel type: singlestat
 - Panel: '"Idle in TX" count' → Deprecated panel type: singlestat
 - Panel: 'DB size (last)' → Deprecated panel type: singlestat
 - Panel: 'DB size change (diff.)' → Deprecated panel type: singlestat
 - Panel: 'DATADIR disk space left' → Deprecated panel type:
singlestat
 - Panel: 'Query runtime (avg.)' → Deprecated panel type: singlestat
 - Panel: 'Config change events' → Deprecated panel type: singlestat
 - Panel: 'Table changes' → Deprecated panel type: singlestat
 - Panel: 'WAL archiving status' → Deprecated panel type: singlestat
 - Panel: 'WAL folder size' → Deprecated panel type: singlestat
 - Panel: 'Invalid / duplicate indexes' → Deprecated panel type:
singlestat
 - Panel: 'Autovacuum issues' → Deprecated panel type: singlestat
 - Panel: 'Checkpoints requested' → Deprecated panel type: singlestat
 - Panel: 'Approx. table bloat' → Deprecated panel type: singlestat
 - Panel: 'WAL per second (avg.)' → Deprecated panel type: singlestat
 - Panel: 'Temp. bytes per second (avg.)' → Deprecated panel type:
singlestat
 - Panel: 'Longest AUTOVACUUM duration' → Deprecated panel type:
singlestat
 - Panel: 'Seq. scans on >100 MB tables per minute (avg.)' →
Deprecated panel type: singlestat
 - Panel: 'INSERT-s per minute (avg.)' → Deprecated panel type:
singlestat
 - Panel: 'Backup duration' → Deprecated panel type: singlestat
 - Panel: 'Max. table FREEZE age' → Deprecated panel type: singlestat
 - Panel: 'Max. XMIN horizon age' → Deprecated panel type: singlestat
 - Panel: 'Inactive repl. slots' → Deprecated panel type: singlestat
 - Panel: 'Max. replication lag' → Deprecated panel type: singlestat
```

SEE MORE ...... see results for each dashboard

https://github.com/Gaurav05082002/PostgreSQL_GSOC/blob/main/Detection%20Script%20and%20result.md

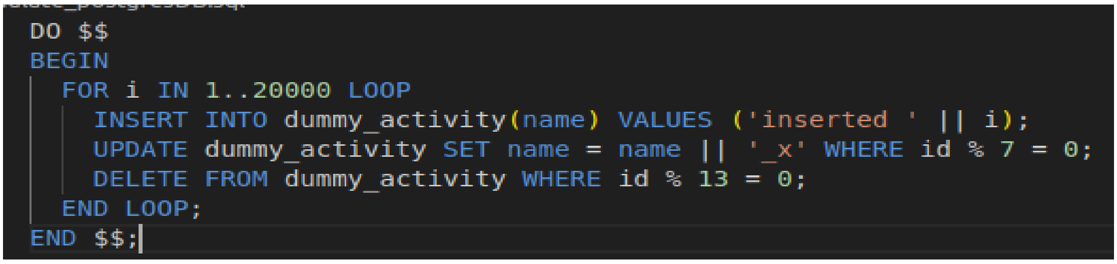
- **Milestone 2 (June 17 to July 10) - Data Simulation and Functional Validation of Dashboard Components**
    - Performed **manual validation** to ensure that v11 dashboards not only render correctly but also fully utilize updated features and resolve any discrepancies introduced by differences between Grafana v10 and v11.
    - For each dashboard, simulations will be done for **query execution, table level activity, and database state changes** to verify that every panel is receiving and displaying the expected data.
    - I used the postgres database as the target and successfully **simulated data flow into the DB Overview dashboard**, including TPS, QPS, WAL, and buffer metrics.
    - **Simulation Steps & Queries Used** for DB overview dashboard are as follows

Step 1: Create an activity table



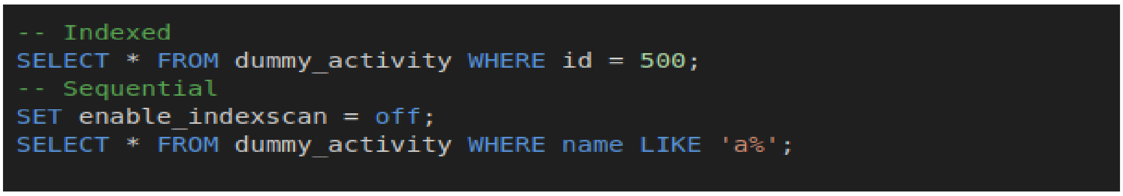
```
CREATE TABLE IF NOT EXISTS dummy_activity(id SERIAL, name TEXT);
```

Step 2: Insert/Update/Delete operations



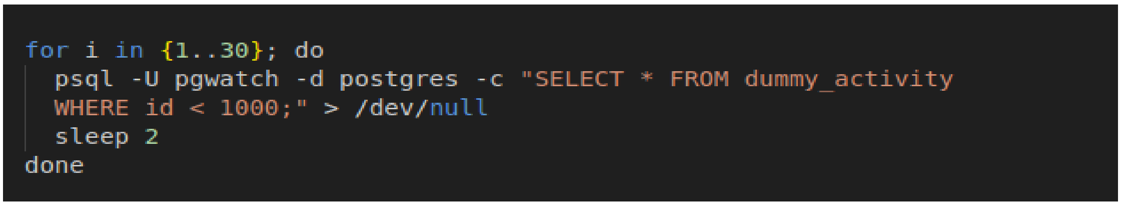
```
DO $$
BEGIN
  FOR i IN 1..20000 LOOP
    INSERT INTO dummy_activity(name) VALUES ('inserted ' || i);
    UPDATE dummy_activity SET name = name || '_x' WHERE id % 7 = 0;
    DELETE FROM dummy_activity WHERE id % 13 = 0;
  END LOOP;
END $$;
```

Step 3: Simulate index and sequential scans

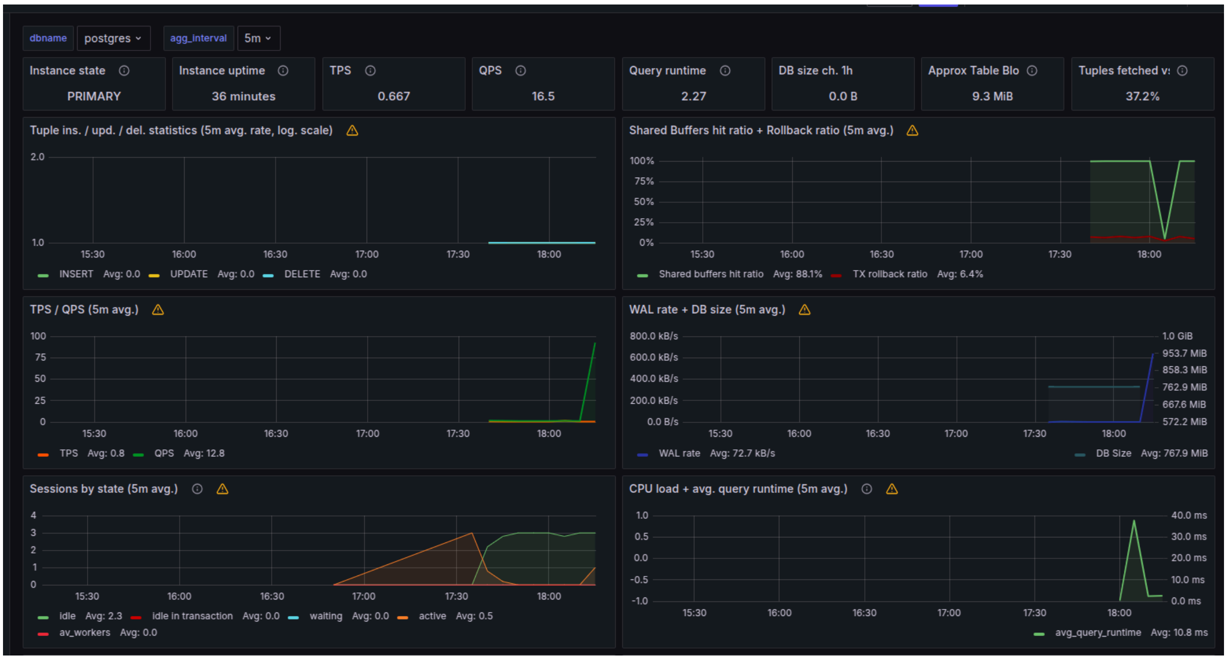


```
-- Indexed
SELECT * FROM dummy_activity WHERE id = 500;
-- Sequential
SET enable_indexscan = off;
SELECT * FROM dummy_activity WHERE name LIKE 'a%';
```

Step 4: Loop query to trigger QPS



```
for i in {1..30}; do
  psql -U pgwatch -d postgres -c "SELECT * FROM dummy_activity
  WHERE id < 1000;" > /dev/null
  sleep 2
done
```

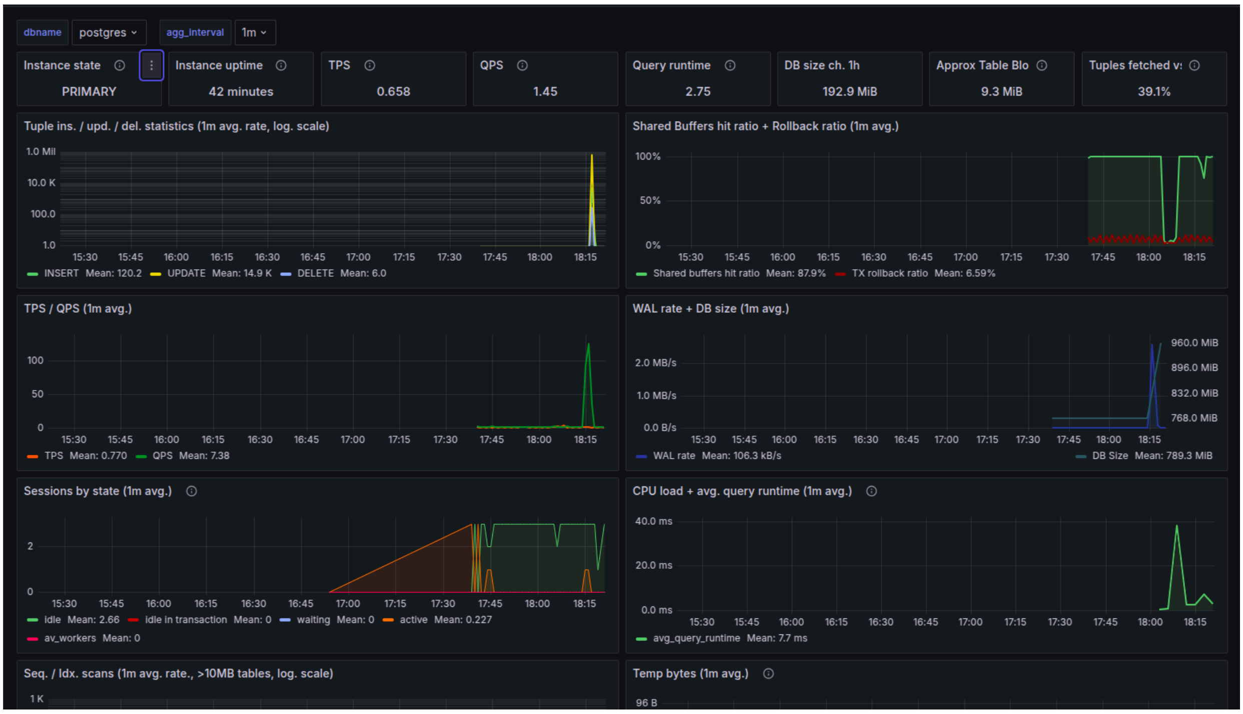


dbname postgres agg_interval 5m

Instance state	Instance uptime	TPS	QPS	Query runtime	DB size ch. 1h	Approx Table Blo	Tuples fetched v:
PRIMARY	36 minutes	0.667	16.5	2.27	0.0 B	9.3 MiB	37.2%

Tuple ins. / upd. / del. statistics (5m avg. rate, log. scale) ⚠
2.0
1.0
15:30 16:00 16:30 17:00 17:30 18:00
INSERT Avg: 0.0 UPDATE Avg: 0.0 DELETE Avg: 0.0
Shared Buffers hit ratio + Rollback ratio (5m avg.) ⚠
100% 75% 50% 25% 0%
15:30 16:00 16:30 17:00 17:30 18:00
Shared buffers hit ratio Avg: 88.1% TX rollback ratio Avg: 6.4%
TPS / QPS (5m avg.) ⚠
100 75 50 25 0
15:30 16:00 16:30 17:00 17:30 18:00
TPS Avg: 0.8 QPS Avg: 12.8
WAL rate + DB size (5m avg.) ⚠
800.0 kB/s 600.0 kB/s 400.0 kB/s 200.0 kB/s 0.0 B/s
1.0 GiB 953.7 MiB 858.3 MiB 762.9 MiB 667.6 MiB 572.2 MiB
15:30 16:00 16:30 17:00 17:30 18:00
WAL rate Avg: 72.7 kB/s DB Size Avg: 767.9 MiB
Sessions by state (5m avg.) ⚠
4 3 2 1 0
15:30 16:00 16:30 17:00 17:30 18:00
idle Avg: 2.3 idle in transaction Avg: 0.0 waiting Avg: 0.0 active Avg: 0.5
av_workers Avg: 0.0
CPU load + avg. query runtime (5m avg.) ⚠
1.0 0.5 0.0 -0.5 -1.0
40.0 ms 30.0 ms 20.0 ms 10.0 ms 0.0 ms
15:30 16:00 16:30 17:00 17:30 18:00
avg_query_runtime Avg: 10.8 ms

*Img > Old version of the **DB-Overview** dashboard running on Grafana v11. The presence of the "⚠️" warning icon indicates the use of **deprecated AngularJS-based components***



dbname postgres agg_interval 1m

Instance state	Instance uptime	TPS	QPS	Query runtime	DB size ch. 1h	Approx Table Blo	Tuples fetched v:
PRIMARY	42 minutes	0.658	1.45	2.75	192.9 MiB	9.3 MiB	39.1%

Tuple ins. / upd. / del. statistics (1m avg. rate, log. scale)
1.0 Mil 10.0 K 100.0 1.0
15:30 15:45 16:00 16:15 16:30 16:45 17:00 17:15 17:30 17:45 18:00 18:15
INSERT Mean: 120.2 UPDATE Mean: 14.9 K DELETE Mean: 6.0
Shared Buffers hit ratio + Rollback ratio (1m avg.)
100% 50% 0%
15:30 15:45 16:00 16:15 16:30 16:45 17:00 17:15 17:30 17:45 18:00 18:15
Shared buffers hit ratio Mean: 87.9% TX rollback ratio Mean: 6.59%
TPS / QPS (1m avg.)
100 50 0
15:30 15:45 16:00 16:15 16:30 16:45 17:00 17:15 17:30 17:45 18:00 18:15
TPS Mean: 0.770 QPS Mean: 7.38
WAL rate + DB size (1m avg.)
2.0 MB/s 1.0 MB/s 0.0 B/s
960.0 MiB 896.0 MiB 832.0 MiB 768.0 MiB
15:30 15:45 16:00 16:15 16:30 16:45 17:00 17:15 17:30 17:45 18:00 18:15
WAL rate Mean: 106.3 kB/s DB Size Mean: 789.3 MiB
Sessions by state (1m avg.)
2 0
15:30 15:45 16:00 16:15 16:30 16:45 17:00 17:15 17:30 17:45 18:00 18:15
idle Mean: 2.66 idle in transaction Mean: 0 waiting Mean: 0 active Mean: 0.227
av_workers Mean: 0
CPU load + avg. query runtime (1m avg.)
40.0 ms 20.0 ms 0.0 ms
15:30 15:45 16:00 16:15 16:30 16:45 17:00 17:15 17:30 17:45 18:00 18:15
avg_query_runtime Mean: 7.7 ms
Seq. / Idx. scans (1m avg. rate., >10MB tables, log. scale)
1 K
Temp bytes (1m avg.)
96 B

*Img > Updated **DB-Overview** dashboard on Grafana v11 with all **deprecated AngularJS components removed** (no "⚠️" warning icons).*
Legacy visualizations, such as graph have been replaced with v11-supported components like timeseries.
*The simulation is active, and data is successfully populated across all graphs and panels, indicating that the dashboard is now **fully functional and ready for production use**.*

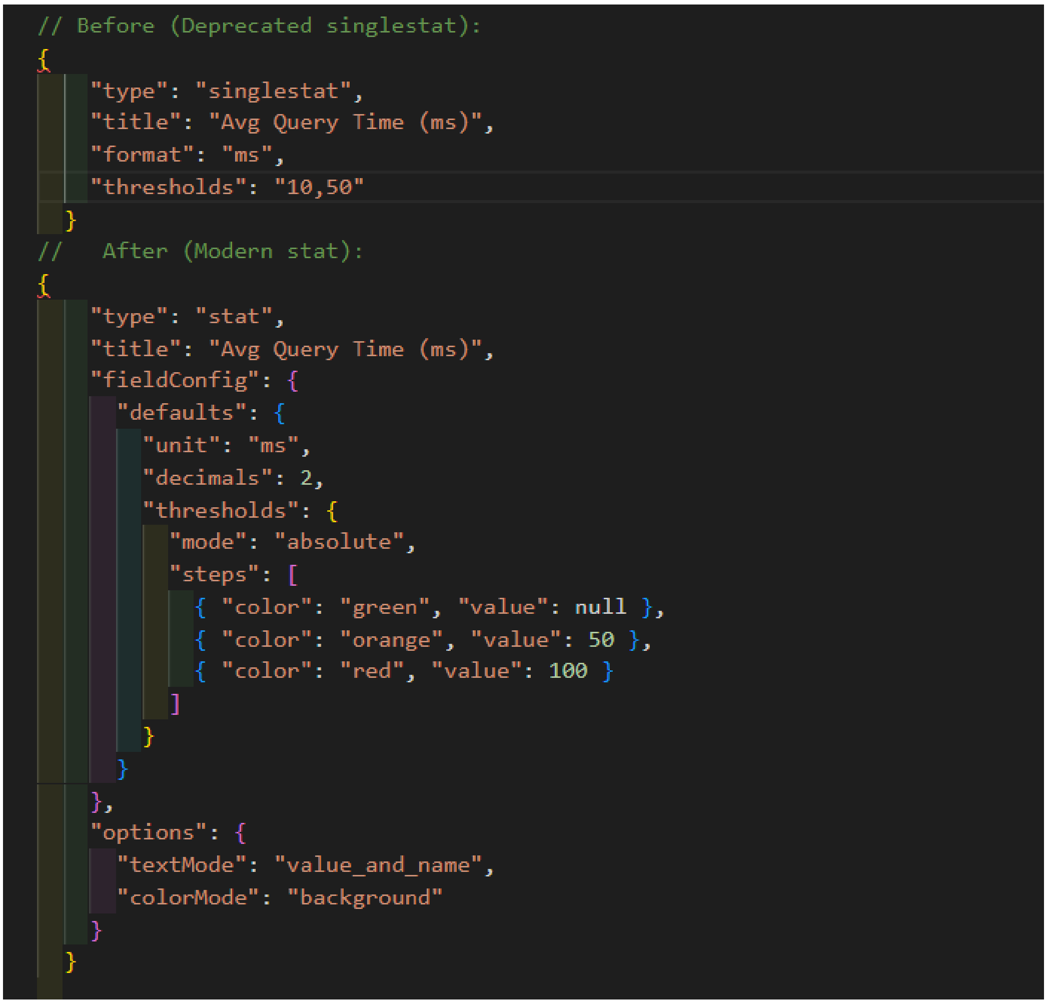
- **Milestone 3 (July 11 to July 31) - JSON Refinement with Grafana v11 Features and UX Enhancements**
    - Reviewed and **polished the JSON definitions** of each dashboard to ensure alignment with Grafana v11's schema. This included replacing legacy fields like yaxes, lines, and fill with the modern fieldConfig, options, and overrides structure
    - Incorporating **new visualization capabilities**, such as replacing deprecated graph panels with timeseries, configuring stat panels with value_and_name text mode, setting thresholds, and defining units to make metrics more interpretable.
    - Applying **Grafana transformations** like merge, add field from calculation, and reduce to simplify complex queries, reduce backend load, and improve the responsiveness and clarity of the dashboards for end users.
    - I have done the following JSON refinementand ux changes on the **Single Query Dashboard**

Refactored the JSON panel structure of Single Query Dashboard :

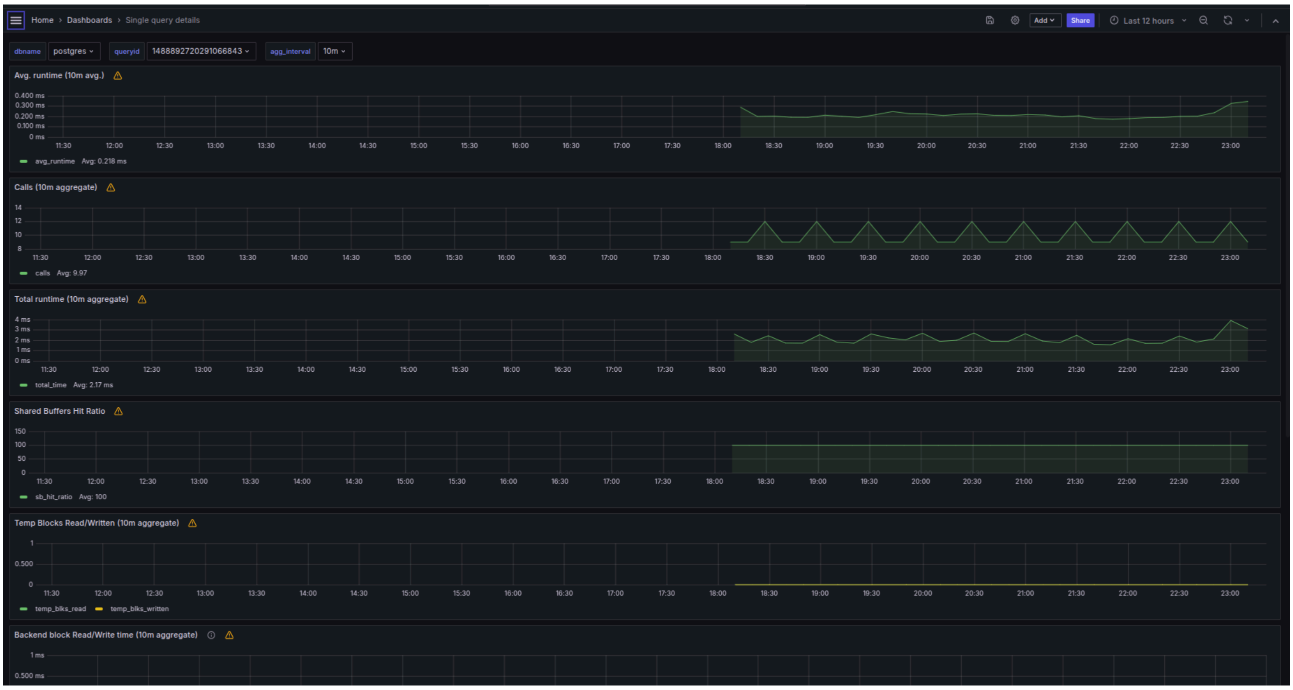
Removed outdated fields (`yaxes`, `lines`, `fill`, etc.)

Introduced `fieldConfig`, `overrides`, and `options` blocks

Set meaningful units (`ms`, `ops`, `%`), threshold colors, and legends



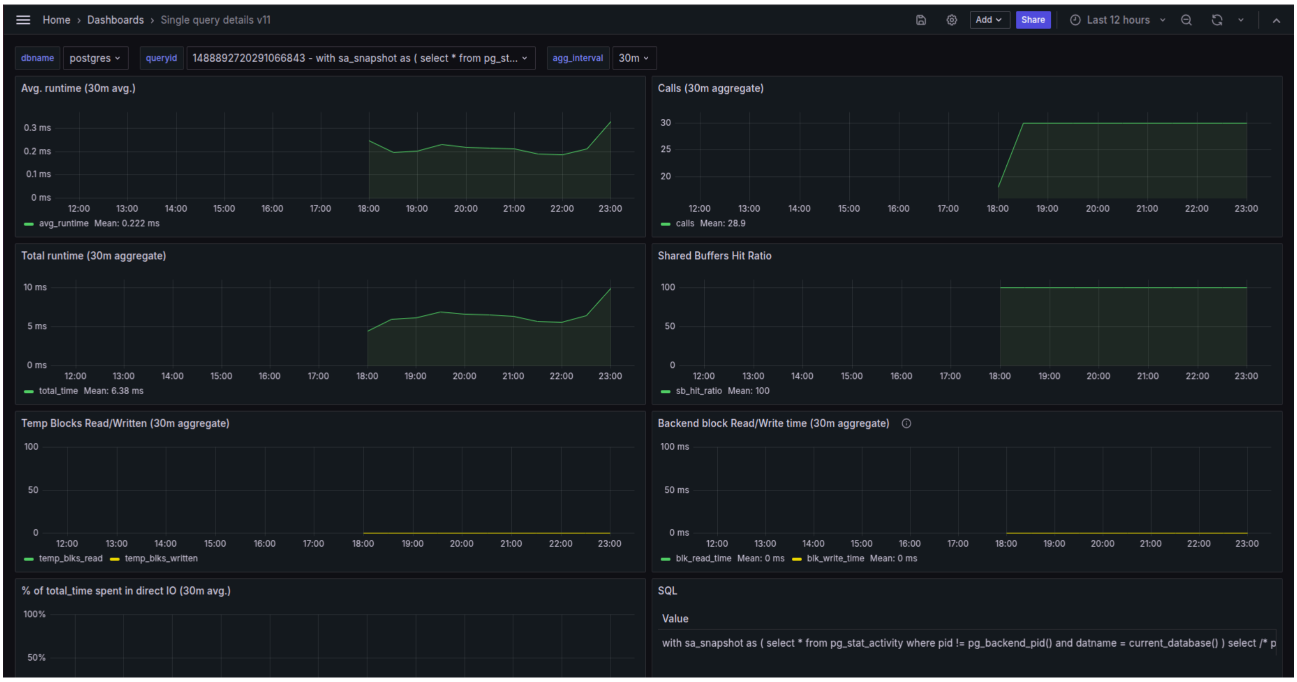
```
// Before (Deprecated singlestat):
{
    "type": "singlestat",
    "title": "Avg Query Time (ms)",
    "format": "ms",
    "thresholds": "10,50"
  }
//   After (Modern stat):
{
    "type": "stat",
    "title": "Avg Query Time (ms)",
    "fieldConfig": {
      "defaults": {
        "unit": "ms",
        "decimals": 2,
        "thresholds": {
          "mode": "absolute",
          "steps": [
            { "color": "green", "value": null },
            { "color": "orange", "value": 50 },
            { "color": "red", "value": 100 }
          ]
        }
      }
    },
    "options": {
      "textMode": "value_and_name",
      "colorMode": "background"
    }
  }
```



dbname postgres queryid 1488892720291066843 agg_interval 10m

Img > Old version of the ***Single query details*** *dashboard running on Grafana v11. "⚠️" indicates* ***deprecated components*** *Follows* ***horizontal arrangement*** *for components.*
At top we have to ***select query id*** *and this query run to get the data*

```
SELECT DISTINCT tag_data->>'queryid' FROM stat_statements WHERE time >
current_date - 3 AND dbname = '$dbname' ORDER BY 1;
```



dbname postgres queryid 1488892720291066843 - with sa_snapshot as (select * from pg_st... agg_interval 30m

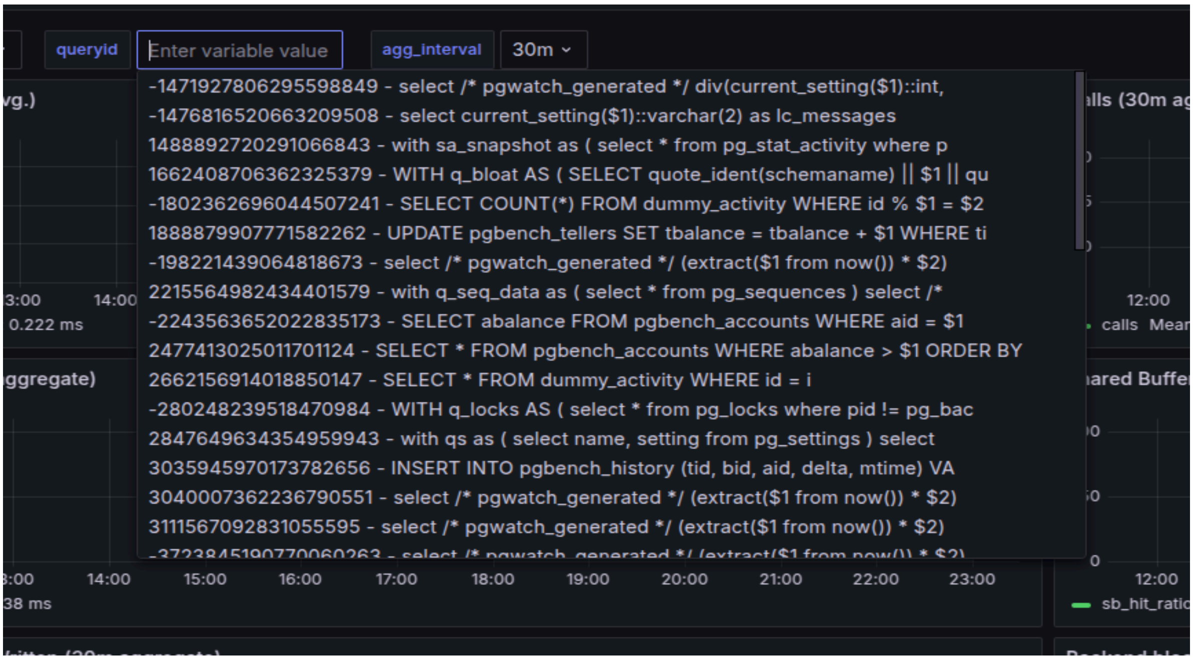
Value

with sa_snapshot as (select * from pg_stat_activity where pid != pg_backend_pid() and datname = current_database()) select /* p

Img > Upgraded ***Single Query Details*** *dashboard. To improve user experience, a* ***vertical panel arrangement*** *has been adopted.* ***Query metadata*** *has been added to the selection dropdown, allowing users to easily identify and select queries based on their content.*

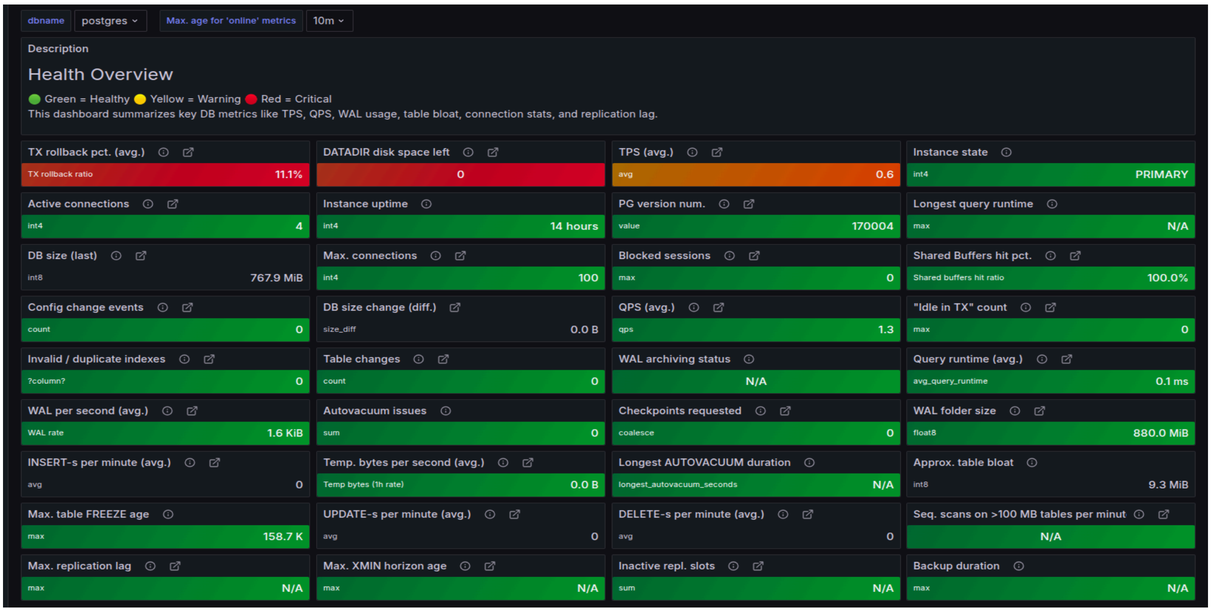
Updated query allowing users to easily identify , select queries based on their content.

```
SELECT DISTINCT (tag_data->>'queryid') || ' - ' || LEFT(tag_data->>'query',
60) AS __text,
  tag_data->>'queryid' AS __value FROM stat_statements WHERE time >
current_date - 3
  AND dbname = '$dbname' ORDER BY 1;
```



```
-1471927806295598849 - select /* pgwatch_generated */ div(current_setting($1)::int,
-1476816520663209508 - select current_setting($1)::varchar(2) as lc_messages
1488892720291066843 - with sa_snapshot as ( select * from pg_stat_activity where p
1662408706362325379 - WITH q_bloat AS ( SELECT quote_ident(schemaname) || $1 || qu
-1802362696044507241 - SELECT COUNT(*) FROM dummy_activity WHERE id % $1 = $2
1888879907771582262 - UPDATE pgbench_tellers SET tbalance = tbalance + $1 WHERE ti
-1982214390648186873 - select /* pgwatch_generated */ (extract($1 from now()) * $2)
2215564982434401579 - with q_seq_data as ( select * from pg_sequences ) select /*
-2243563652022835173 - SELECT abalance FROM pgbench_accounts WHERE aid = $1
2477413025011701124 - SELECT * FROM pgbench_accounts WHERE abalance > $1 ORDER BY
2662156914018850147 - SELECT * FROM dummy_activity WHERE id = i
-2802482395184709084 - WITH q_locks AS ( select * from pg_locks where pid != pg_bac
2847649634354959943 - with qs as ( select name, setting from pg_settings ) select
3035945970173782656 - INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VA
3040007362236790551 - select /* pgwatch_generated */ (extract($1 from now()) * $2)
3111567092831055595 - select /* pgwatch_generated */ (extract($1 from now()) * $2)
-3723845190770060263 - select /* pgwatch_generated */ (extract($1 from now()) * $2)
```

*Img > Enhanced user experience with **query metadata displayed alongside IDs**, enabling easier and more intuitive query selection.*



Description

Health Overview

🟢 Green = Healthy 🟡 Yellow = Warning 🔴 Red = Critical
This dashboard summarizes key DB metrics like TPS, QPS, WAL usage, table bloat, connection stats, and replication lag.

TX rollback pct. (avg.) — TX rollback ratio 11.1%
DATADIR disk space left — 0
TPS (avg.) — avg 0.6
Instance state — int4 PRIMARY
Active connections — int4 4
Instance uptime — int4 14 hours
PG version num. — value 170004
Longest query runtime — max N/A
DB size (last) — int8 767.9 MiB
Max. connections — int4 100
Blocked sessions — max 0
Shared Buffers hit pct. — Shared buffers hit ratio 100.0%
Config change events — count 0
DB size change (diff.) — size_diff 0.0 B
QPS (avg.) — qps 1.3
"Idle in TX" count — max 0
Invalid / duplicate indexes — ?column? 0
Table changes — count 0
WAL archiving status — N/A
Query runtime (avg.) — avg_query_runtime 0.1 ms
WAL per second (avg.) — WAL rate 1.6 KiB
Autovacuum issues — sum 0
Checkpoints requested — coalesce 0
WAL folder size — float8 880.0 MiB
INSERT-s per minute (avg.) — avg 0
Temp. bytes per second (avg.) — Temp bytes (1h rate) 0.0 B
Longest AUTOVACUUM duration — longest_autovacuum_seconds N/A
Approx. table bloat — int8 9.3 MiB
Max. table FREEZE age — max 158.7 K
UPDATE-s per minute (avg.) — avg 0
DELETE-s per minute (avg.) — avg 0
Seq. scans on >100 MB tables per minut — N/A
Max. replication lag — max N/A
Max. XMIN horizon age — max N/A
Inactive repl. slots — sum N/A
Backup duration — max N/A

*Img > Upgraded **Health Overview** dashboard with `"textMode": "value_and_name"` for clearer metrics, **panels sorted by criticality**, and **descriptions added** to enhance user experience.*

- **Milestone 4 (Aug 1 to Aug 25) - Documenting Migration Workflow and Exploring Additional Feature Integrations**

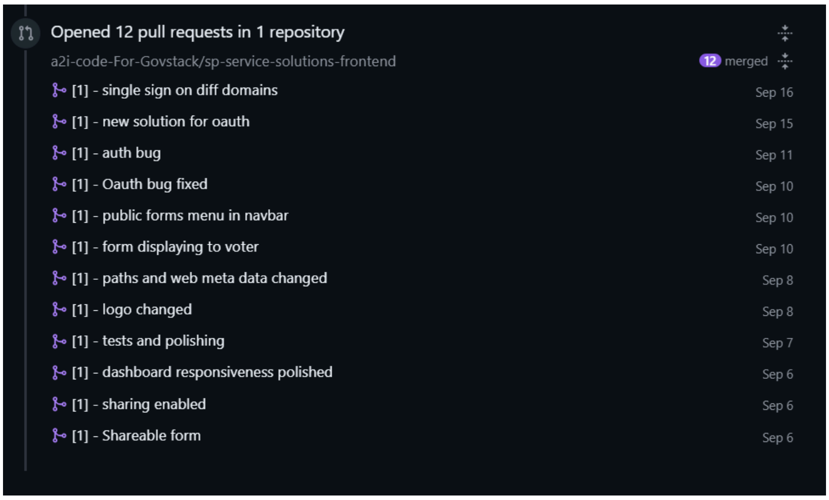
    - **Documenting the complete migration process**, including steps for identifying deprecated components, updating panel JSON structure, and validating dashboards with live data. Also adding **guidelines for future updates** and version management.
    - **Integrating Grot (Grafana's AI assistant)** into the **left panel of Grafana's home UI** to provide contextual help for queries, visualizations, and troubleshooting within dashboards.
    - Addressed Grafana's limitation around **real-time panel reordering** by proposing a **summary panel** that highlights critical metrics using sorting and transformation features.
    - Creating a **"Generate Summary" feature** for dashboards like *Single Query Details*, which auto-generates key query insights using calculated fields and reduces transformations.

Project Timeline:

Period / Milestone	Focus Area
May 8 – June 1 (Community Bonding Period)	Engagement, Setup Walkthrough, Feedback, Timeline Finalization
June 2 – June 16 (Milestone 1)	Detection and Replacement of Deprecated AngularJS Components
June 17 – July 10 (Milestone 2)	Data Simulation and Functional Validation of Dashboard Components
July 11 – July 31 (Milestone 3)	JSON Refinement with Grafana v11 Features and UX Enhancements
Aug 1 – Aug 25 (Milestone 4)	Documentation and Integration of Additional Features

Contributions to open source:

CV buddy: https://github.com/praneeth-rdy/CV-Buddy/pull/10
Code mystic: https://github.com/codemistic/Web-Development/issues/292
Code mystic: https://github.com/codemistic/Web-Development/pull/298
Code mystic: https://github.com/codemistic/Web-Development/pull/297
Code mystic: https://github.com/codemistic/Web-Development/pull/291
C4GT Open Source Program Organization A2I (PR From 23 to 34)
https://github.com/a2i-code-For-Govstack/sp-service-solutions-frontend/pull/23



Opened 12 pull requests in 1 repository

a2i-code-For-Govstack/sp-service-solutions-frontend 12 merged

[1] - single sign on diff domains	Sep 16
[1] - new solution for oauth	Sep 15
[1] - auth bug	Sep 11
[1] - Oauth bug fixed	Sep 10
[1] - public forms menu in navbar	Sep 10
[1] - form displaying to voter	Sep 10
[1] - paths and web meta data changed	Sep 8
[1] - logo changed	Sep 8
[1] - tests and polishing	Sep 7
[1] - dashboard responsiveness polished	Sep 6
[1] - sharing enabled	Sep 6
[1] - Shareable form	Sep 6

My Projects:

Visual Image Search: https://github.com/Gaurav05082002/Visual_Image_Search
Survey & Analysis App: https://github.com/a2i-code-For-Govstack/sp-service-solutions
AI chat pdf: https://github.com/Gaurav05082002/ChatPDF
SF Main Website: https://bit.ly/springfest-23
SQL Editor: https://gaurav-sql-editor.netlify.app/
Animated Website: https://animated-web-eight.vercel.app/

References:

Project Details: https://wiki.postgresql.org/wiki/GSoC_2025#Project_Description
Prototype Repo: https://github.com/Gaurav05082002/PostgreSQL_GSOC