



**EDB**

Postgres® for the AI Generation

# Unified Observability: Monitoring Postgres Anywhere with OpenTelemetry

PgConf Nepal 2025

Yogesh Jain  
Staff SDE, EDB



# Yogesh Jain

Staff SDE @ EDB

## Curious One | Full Stack Developer




- Building a **Hybrid Control Plane** for managing **Postgres** across platforms
- **Focused** on observability: **collecting & visualizing metrics/logs** from Postgres and microservices
- Enabling a **single pane of glass** for **observability** across all deployments



# Key Challenges in Monitoring Modern Apps

Modern applications span **hybrid, distributed** systems — from **mobile** apps to **cloud** services, containers, and even mainframes.

## Common Challenges:

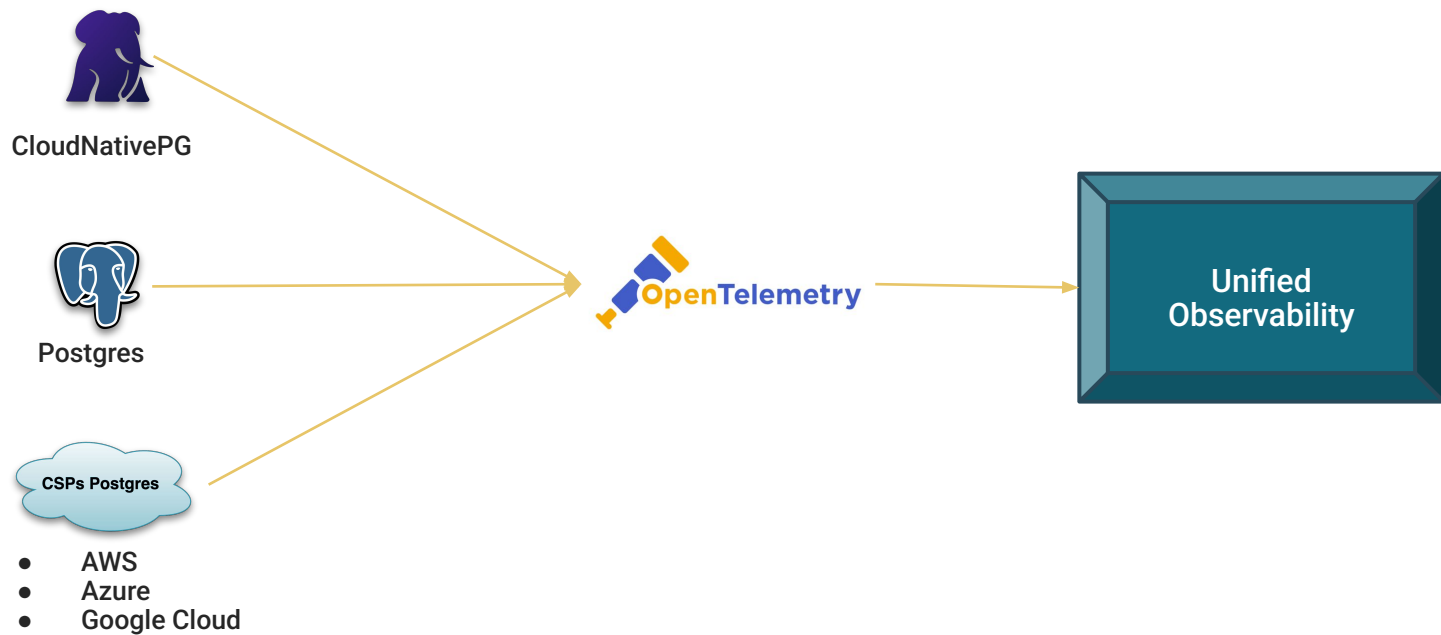
-  **No More Monoliths** - Applications are **fragmented across platforms** resulting in **lack of end-to-end visibility**
-  **Multiple Tools** - Most orgs rely on **4–7 disconnected monitoring tools**, making it **complex** to detect and **resolve** issues.
-  **Separated monitoring data** - Logs, metrics, & traces are collected separately - increasing complexity & **reducing correlations**.

 **Solution** - Unified frameworks like **OpenTelemetry** provide a **vendor-neutral**, open standard for collecting all telemetry signals.

# Agenda

- Observability
- Monitoring PostgreSQL
- CloudNativePG
- OpenTelemetry
- Monitoring Postgres Anywhere with OpenTelemetry
- Demo

# Goal








# Observability

# Observability

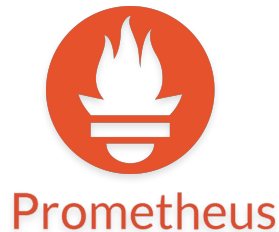
**Observability** is the ability to understand a system's internal state by examining its external outputs – also known as telemetry data.

This includes three core pillars:

-  **Metrics** – Numerical data that reflects the system's health and performance.
-  **Logs** – Text records that capture events and state changes.
-  **Traces** – Detailed records of request flows across services.

# Making a System Observable

- To achieve observability, a **system must be instrumented** — that means the application code must emit logs, metrics, or traces.
- Once instrumented, this telemetry data is sent to an observability backend, where it can be **analyzed** to gain insights, detect issues, and optimize performance.



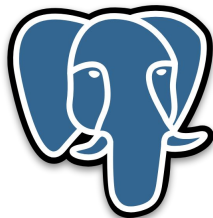




# Monitoring PostgreSQL

The World's Most Advanced Open Source Relational Database

# PostgreSQL - Monitoring - Metrics



We can **capture critical metrics** over time to identify trends and perform root cause analysis, for example:



**Resource Usage:** Track CPU, memory, disk I/O, storage usage



**Query Performance:** Monitor slow queries, locks, & execution plans



**Connections:** Track active sessions, connection limits, & idle connections

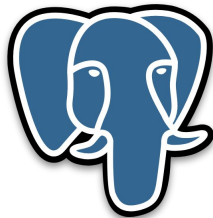


**Database Health:** Monitor replication lag, bloat, & vacuum stats



**Errors:** Keep an eye on failure rates, & warnings

# PostgreSQL - Monitoring - Logs



PostgreSQL logs provide a rich source of information to help **monitor** the **database's health, performance, and security**.

## 🔑 Some Key Logs to Monitor:

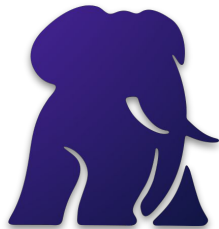
- **Error Logs:** Capture critical errors and crashes like **deadlock** detected.
- **Warning Logs:** Flag potential issues like **slow queries** or low resources.
- **Connection Logs:** Track connections and **failed login** attempts.
- **Statement Logs:** Record **SQL queries** for **auditing** and troubleshooting.



# CloudNativePG

PostgreSQL Operator for Kubernetes

# CloudNativePG



CloudNativePG is a [CNCF Sandbox project](#) — an **open-source Kubernetes operator** for managing **PostgreSQL** workloads in **Kubernetes**.



## Kubernetes-Native by Design

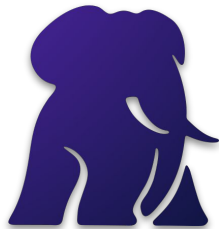
- Defines a **custom k8s resource: Cluster** – represents a PostgreSQL cluster with one primary & optional replicas.
- **Fully declarative** & integrates directly with the Kubernetes API.



## Full Lifecycle Management

- Manages deployment, **scaling**, **failover**, and updates.
- Uses native **streaming replication** for **high availability** (primary/standby architecture)
- Does not rely on StatefulSets. Manages its own **PVCs** for storing **PGDATA**.

# CloudNativePG - Monitoring



CloudNativePG integrates seamlessly with observability stacks using Prometheus and Kubernetes-native logging.



## Metrics Export

- **Exposes native Prometheus metrics**
- Each PostgreSQL instance includes a dedicated exporter
- Predefined metrics included & **Custom queries** can be added **for deeper insights**



## Logging

- **Outputs JSON**-formatted logs (including PostgreSQL logs) to stdout
- No disk persistence — improves security and statelessness



# OpenTelemetry




Open Source Observability Framework

# OpenTelemetry (OTel)



OpenTelemetry is an open-source **Observability Framework** &

toolkit for: **Generating** → **Collecting** → **Processing** → **Exporting** telemetry data:

-  Metrics
-  Logs
-  Traces

 Important to Note:

OpenTelemetry is not an observability backend – it **does not store** or visualize telemetry data.

Instead, it **standardizes and transports** it to the backend of your choice.



# OTel - Key Characteristics



✓ **Open Source**

🔧 **Vendor-Neutral & Tool-Agnostic**

🔄 **Pluggable** with multiple backends:

- Open source Tools (e.g., Prometheus, Loki, Jaeger)
- Commercial platforms (e.g., Grafana Cloud, Datadog, New Relic)

🎯 **Built for Easy & Consistent Instrumentation**

Works across:

👤 **Multiple Programming Language**

🏗️ **Any Infrastructure** - Kubernetes, Bare-metal, VMs

# OTel - Data Pipeline



## Instrumentation



App emits telemetry data:

-  Metrics
-  Logs
-  Traces

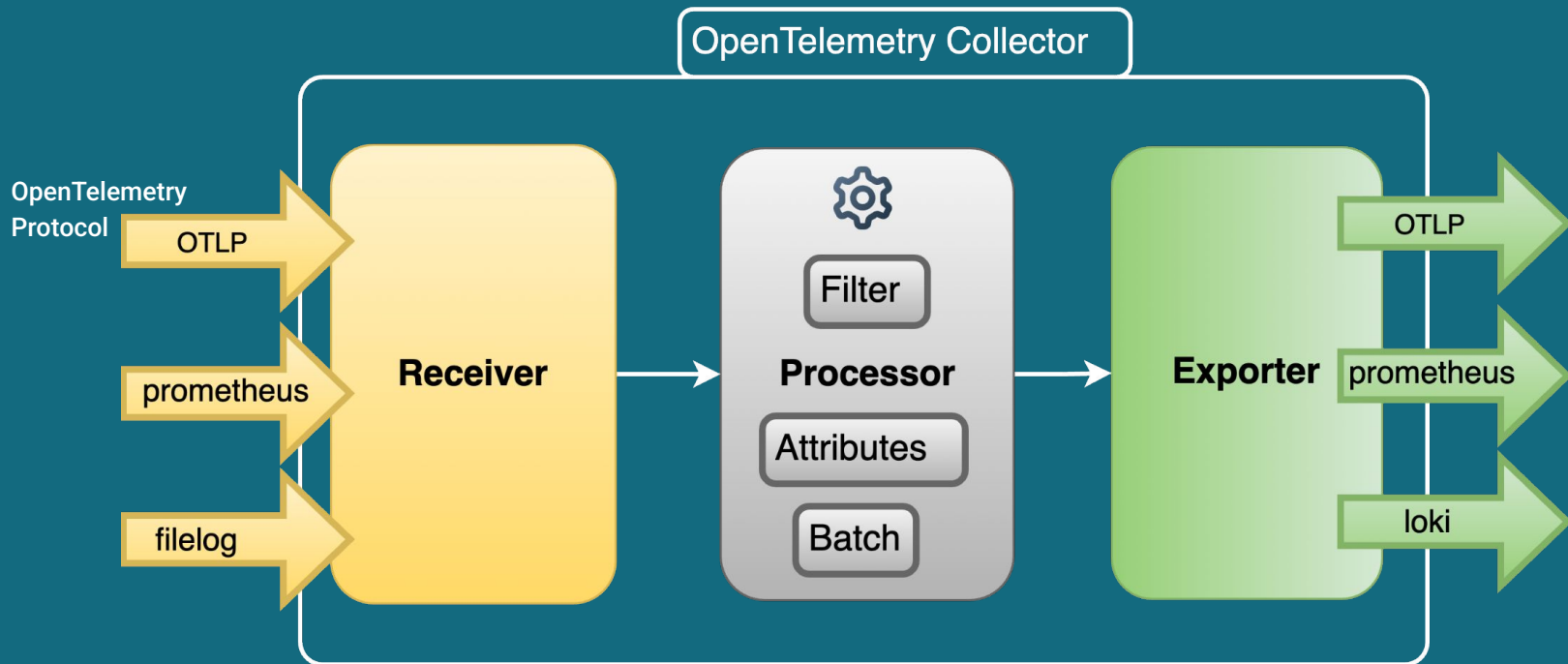
## OTEL Collector

OTel Collector **Receives** → **Processes** → **Exports** telemetry data.

Data can be sent to any preferred **observability backend**:

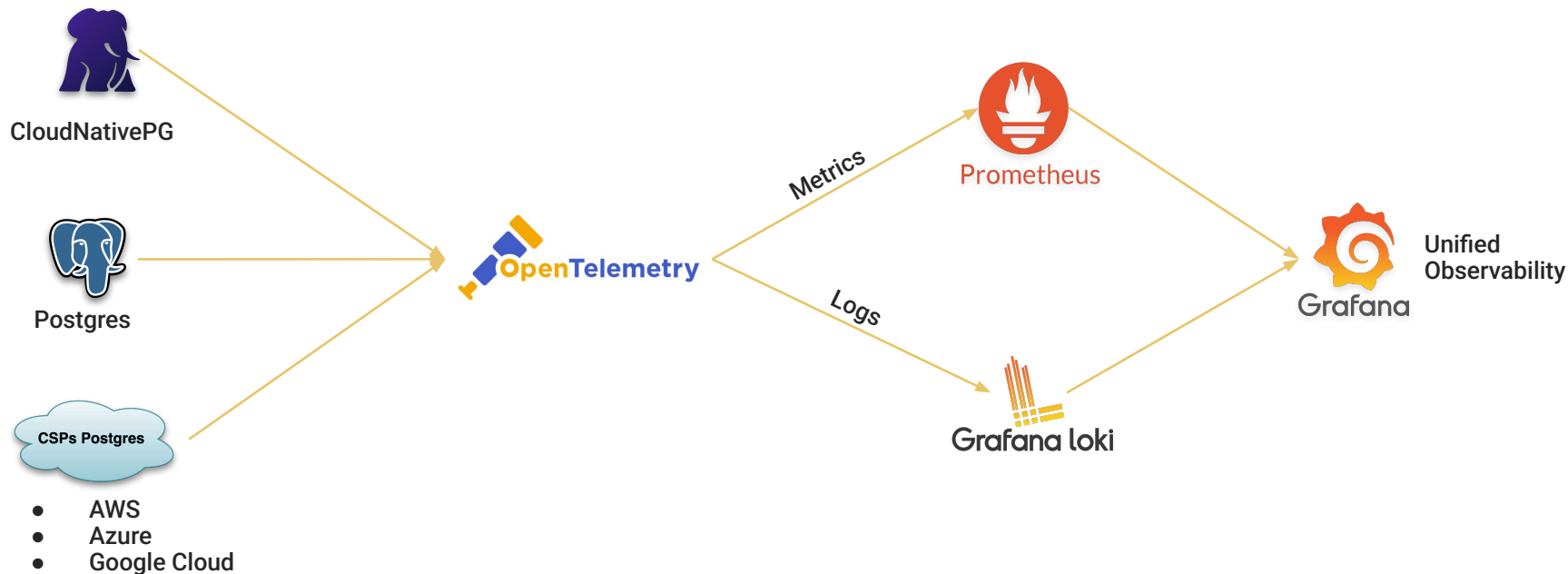
-  Prometheus, Loki
-  Datadog, Grafana Cloud, etc.

# OTel Collector Design



# Monitoring Postgres Anywhere with OpenTelemetry

# Monitoring Postgres Anywhere with OTel



# OTel - Config - Overview for Postgres



A simple pipeline to collect, process, and export PostgreSQL telemetry:

## Receivers

- **hostmetrics** – Gathers system-level metrics
- **postgresql** – Collects PostgreSQL metrics
- **prometheus** – Scrapes metrics from Prometheus - postgres exporters
- **filelog** – Reads PostgreSQL logs

## Processors

- **attributes** – Adds, updates, deletes custom labels
- **filter** – Keeps only relevant metrics/logs (**save storage cost**)
- **batch** – Batches telemetry data for performance

## Exporters

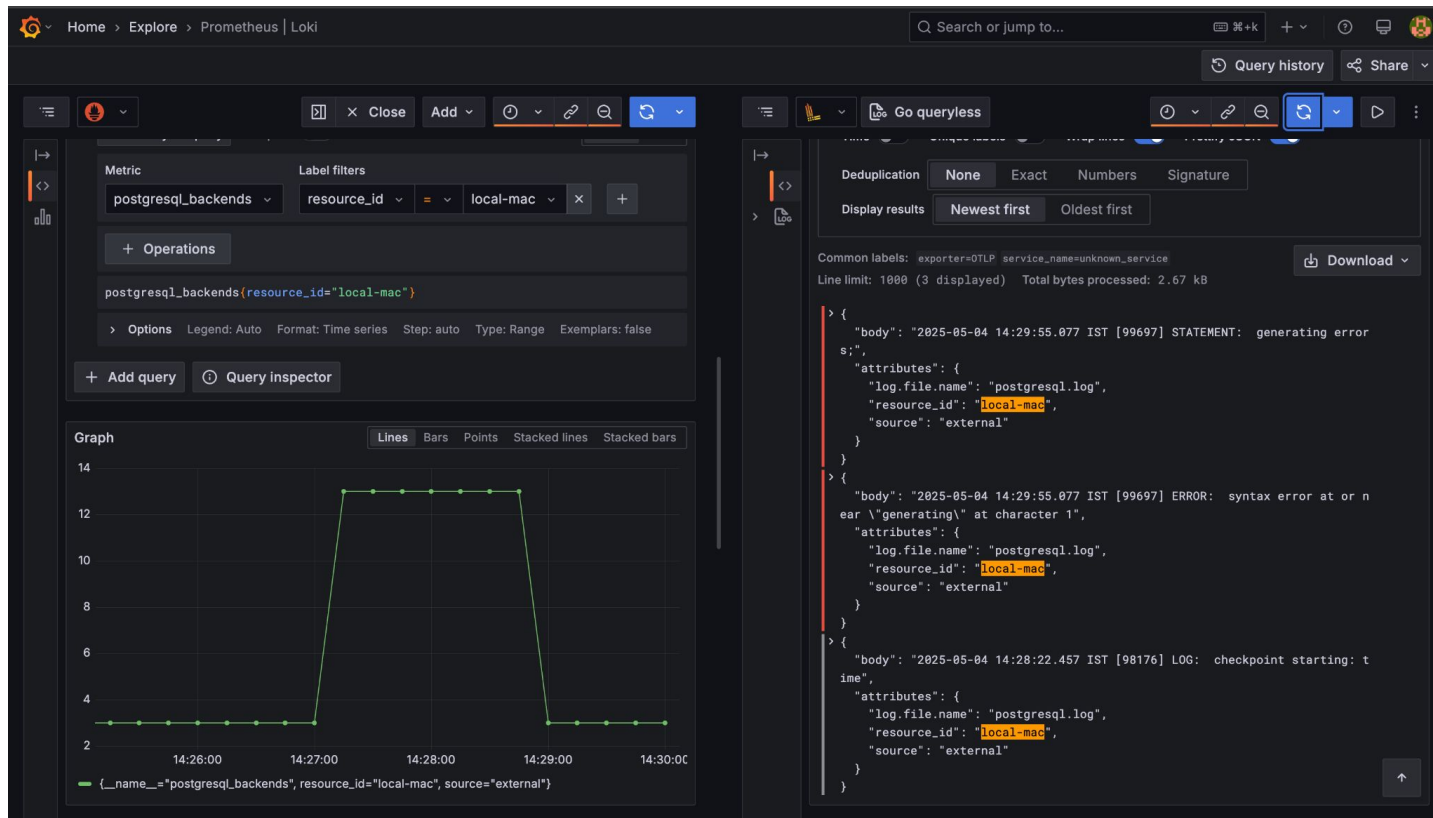
- **prometheusremotewrite** – Sends metrics to Prometheus-compatible backends
- **loki** – Streams logs to Grafana Loki
- **otlp** – Exports to any OpenTelemetry-compatible observability backend

# OTel - Config - Pipeline Example



```
service:
  pipelines:
    metrics:
      receivers: [postgresqlreceiver, hostmetricsreceiver, prometheus]
      processors: [attributes, filter, batch]
      exporters: [prometheusremotewrite, oltp]
    logs:
      receivers: [filelogreceiver]
      processors: [filter, batch]
      exporters: [loki, oltp]
```

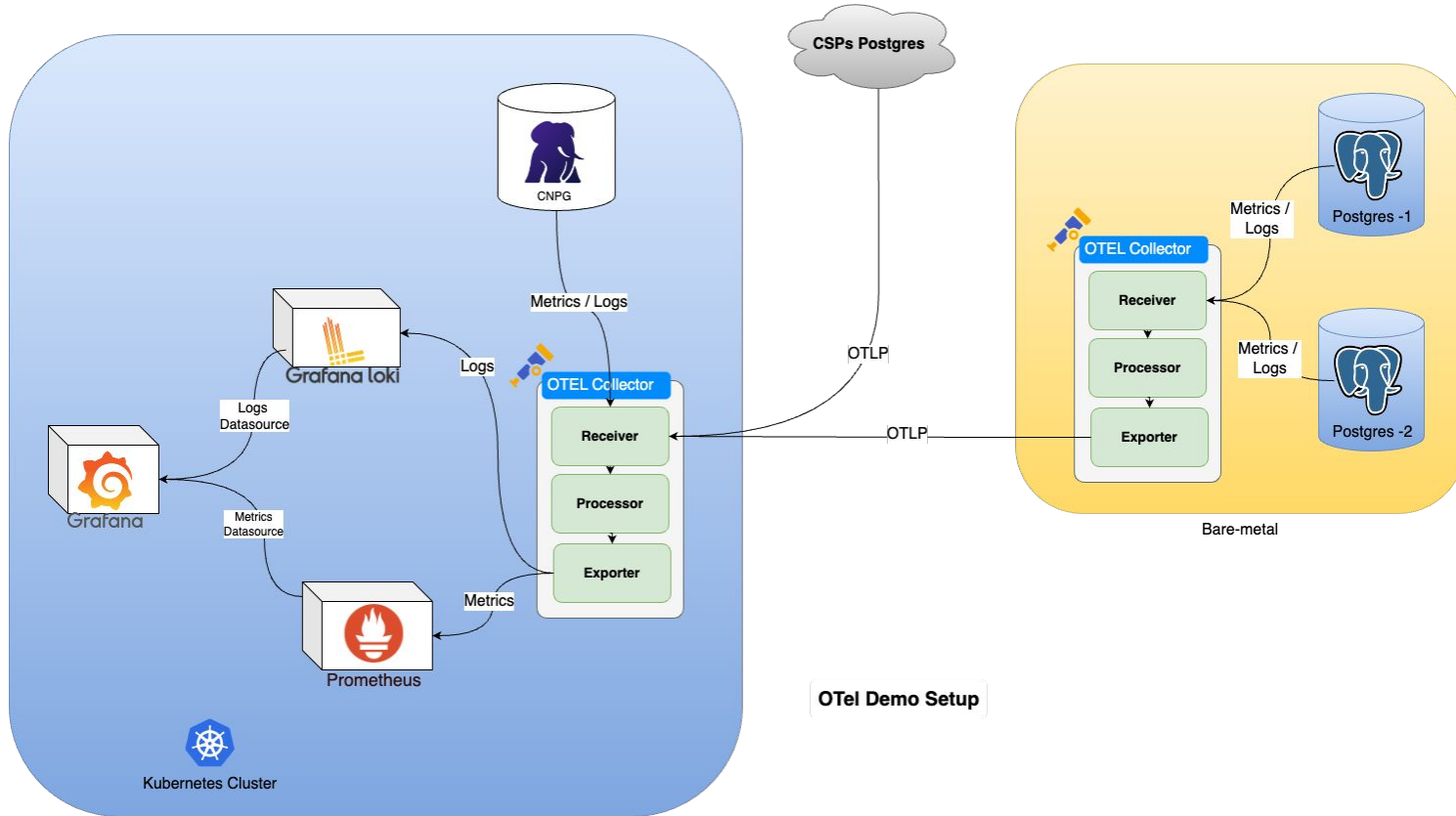
# Unified Observability – Metrics & Logs





# DEMO

# Demo Setup Overview



# Resources/ Useful Links



- [CloudNativePG](#) ( [CNCF Sandbox Project](#) )
- [CloudNativePG - Monitoring](#)
- [OpenTelemetry](#) ( [CNCF Incubating Project](#) )
  - [OpenTelemetry - Receivers List](#)
  - [OpenTelemetry - Processors List](#)
  - [OpenTelemetry - Exporter List](#)
- [Prometheus](#) ( [CNCF Graduated Project](#) )
- [Loki](#)
- [Grafana](#)



# Yogesh Jain

Staff SDE @ EDB

👋 Let's connect to talk about:

- Observability
- Postgres
- Kubernetes
- Open Source Softwares

🔗 Contact:

- [yogesh.jain@enterprisedb.com](mailto:yogesh.jain@enterprisedb.com)
- [LinkedIn - yogeshjain96](#)
- [Blog - curiousone.in](#)

