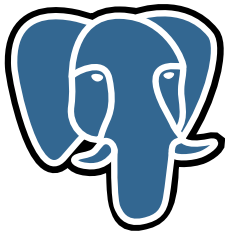
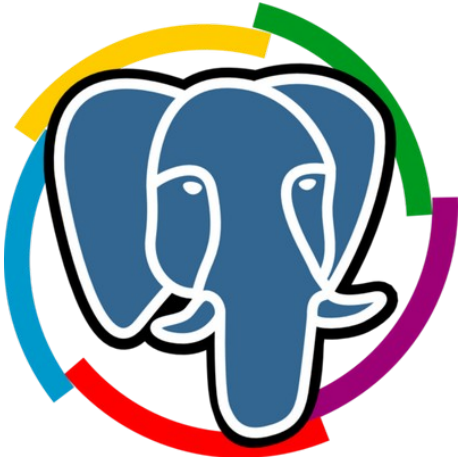


Instrumenter PostgreSQL avec Prometheus



camptocamp

INNOVATIVE SOLUTIONS
BY OPEN SOURCE EXPERTS

Présentation

- Formation : Scheme, ADA, Java
- Développement PHP, Mixage à la demande, Mastering
- Programmation PL/pgSQL, déplacement de la logique dans PostgreSQL
- Développeur géospatial à Camptocamp
- Infrastructure développeur à Camptocamp
- Open Source
- Formations Docker / Kubernetes / PostgreSQL / PostGIS

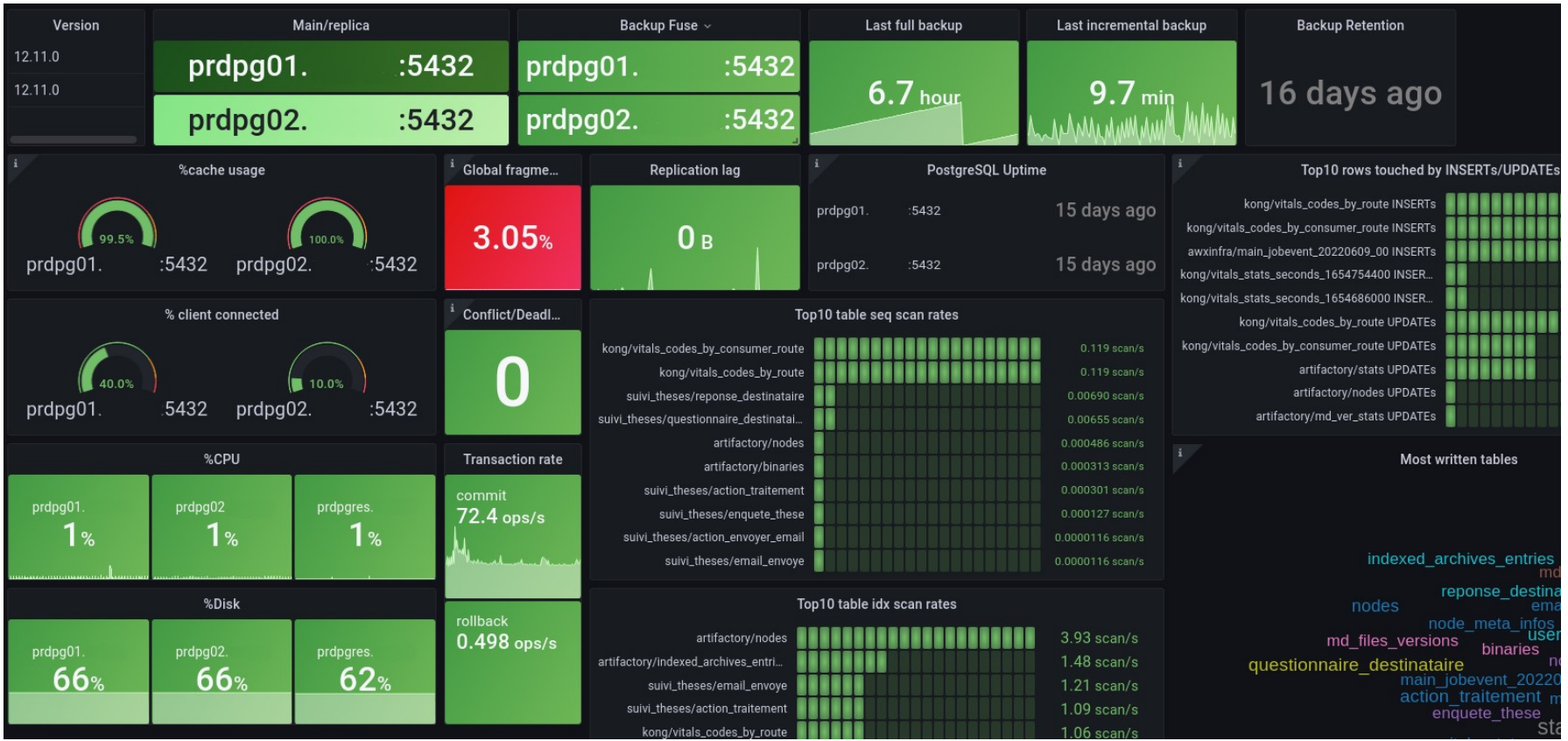
Instrumenter PostgreSQL avec Prometheus

- Observabilité
- Présentation de Prometheus et OpenMetrics
- Métriques du cluster :
  - CPU, RAM, Disques, Réseaux
  - Réplication, Sauvegarde
- Métriques des bases de données :
  - tailles, fragmentation,
  - connexions,
  - index,
  - Activité : lecture, écriture

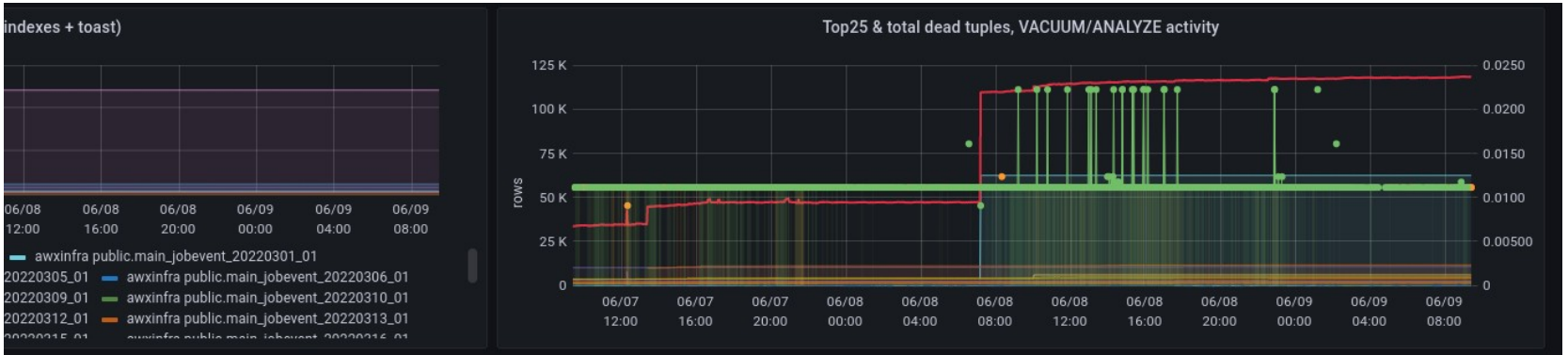
Instrumenter PostgreSQL avec Prometheus

- Métriques des tables :
  - Tailles, données «toastées», nombres d'enregistrements
  - Maintenance, fragmentation
- Métriques des requêtes :
  - requêtes longues, consommatrice de ressources
- Visualisation avec Grafana
- Création d'alerte

Instrumenter PostgreSQL avec Prometheus



Instrumenter PostgreSQL avec Prometheus



Tables Statistics

Live tuples	Dead Tuples ↓	% Fragmentation	Mod since analyse	Last Autovacuum	Last Vacuum	Last Auto Analyse	Last Analyse	Index Size	No
3085657	62371	2.02	26668	-	-	2022-06-08 07:11:01	-	335 MiB	
662	11620	1755	23896	-	-	-	-	96.1 MiB	
58799	10625	18.1	4074	-	-	2022-06-03 12:24:09	-	26.7 MiB	
31644	5943	18.8	1671	2022-06-01 18:01:06	-	2022-06-08 10:00:04	-	10.5 MiB	
33249	4572	13.8	2881	-	-	03:01:31	-	1.26 MiB	
65128	4336	6.66	2509	2022-06-04 02:20:38	-	2022-06-08 12:23:05	-	49.0 MiB	
32325	3687	11.4	400	2022-06-03 22:27:33	-	2022-06-08 22:33:24	-	10.5 MiB	
23250	3402	14.6	1058	-	-	2022-06-07 12:55:09	-	7.90 MiB	
14746	2457	16.7	1465	-	-	2022-06-01 08:02:03	-	7.80 MiB	

Pourquoi Prometheus ?

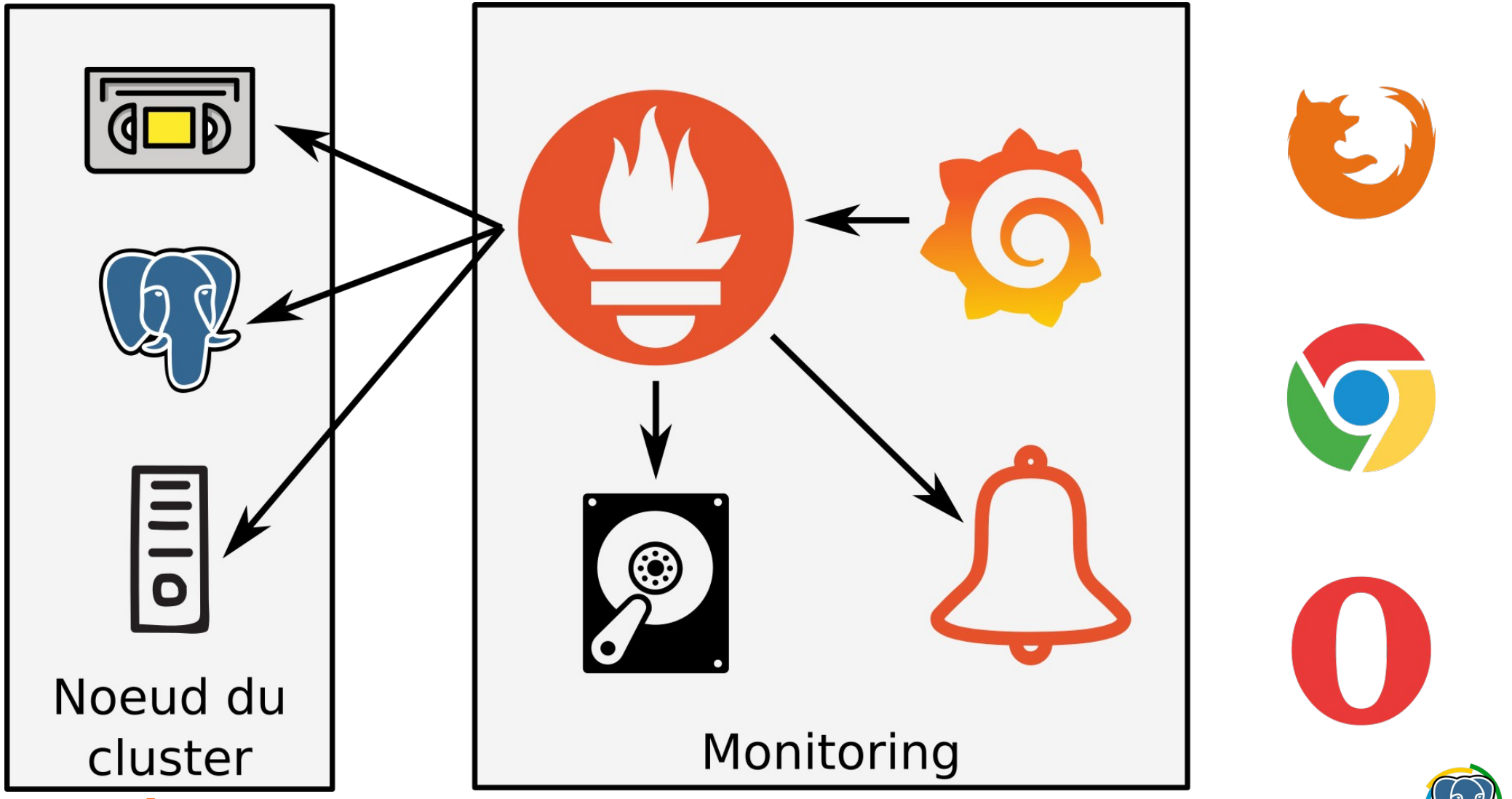
- Outils de monitoring généraliste
- Langage de requête puissant : agrégation, jointure
- Corrélation avec d'autres sources de données :
  - Métriques systèmes
  - Sauvegarde
  - Cloud Provider
- Utilisable par les développeurs et les administrateurs

Pourquoi Prometheus ?

- Performance
- Pas de stockage en local, centralisation des métriques
- Accès aux indicateurs avec un navigateur
- Métriques personnalisées, facilement extensible
- Analyse après incident
- Pas de valeur instantanées, décalage de 10-30 sec
- Open Source

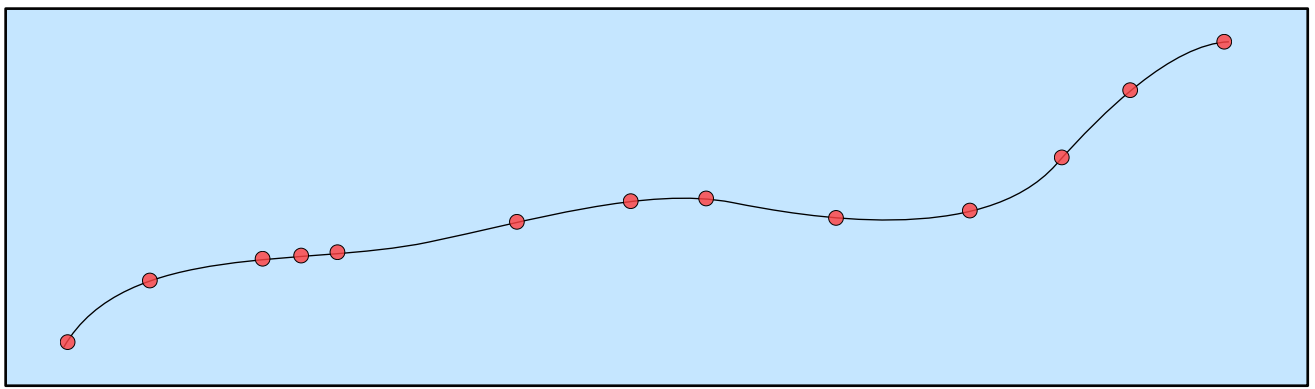
Fonctionnement de Prometheus

- « Pull » métriques à partir d'« exporter »
- Communication HTTP, PushProx
- Stocke des séries temporelles
- Interface Web
- Langage de requête PromQL


Noeud du cluster
Monitoring

Traitement à la demande

- Données brutes récupérées régulièrement (< 30sec)
- Dérivée temporelle calculée à la visualisation
- Gestion de la réinitialisation des compteurs
- Gestion de l’irrégularité des intervalles
- Prédiction Linéaire

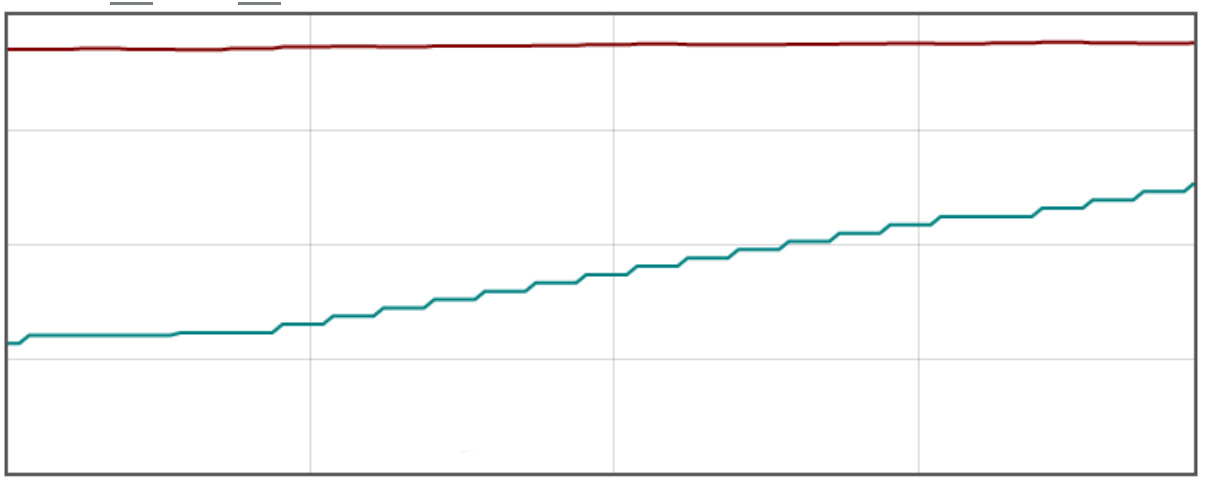


Séries temporelles et Labels

- Ensemble de valeurs numériques horodatées
- Exemple :
  - Taille d'une base de données
  - Nombre de scan séquentiel d'une table
  - Nombre de connexions
- 2 principaux types : «Gauge», «Counter»

Séries temporelles et Labels

- 9h10 : taille_db_1 = 72 Mo
- 9h10 : taille_db_2 = 618 Mo
- 9h11 : taille_db_1 = 73 Mo
- 9h11 : taille_db_2 = 623 Mo



Séries temporelles et Labels

- Ensemble de clés/valeurs associé aux séries temporelles
- Exemple :
  - Taille d'une base de données : **nom de la base de donnée**
  - Nombre de scan séquentiel d'une table : **nom de la table**
  - Nombre de connexions : **nom de la base de données**
- Valeur d'un label : ensemble fini énumérable
- L'agrégation doit donner un résultat « utile »

Séries temporelles et Labels

- `9h10 : taille_db{nom="db_1"} = 72 Mo`
- `9h10 : taille_db{nom="db_2"} = 618 Mo`
- `9h11 : taille_db{nom="db_1"} = 73 Mo`
- `9h11 : taille_db{nom="db_2"} = 623 Mo`

Format OpenMetrics

- Standard en version texte ou binaire
- HTTP GET sur `/metrics`

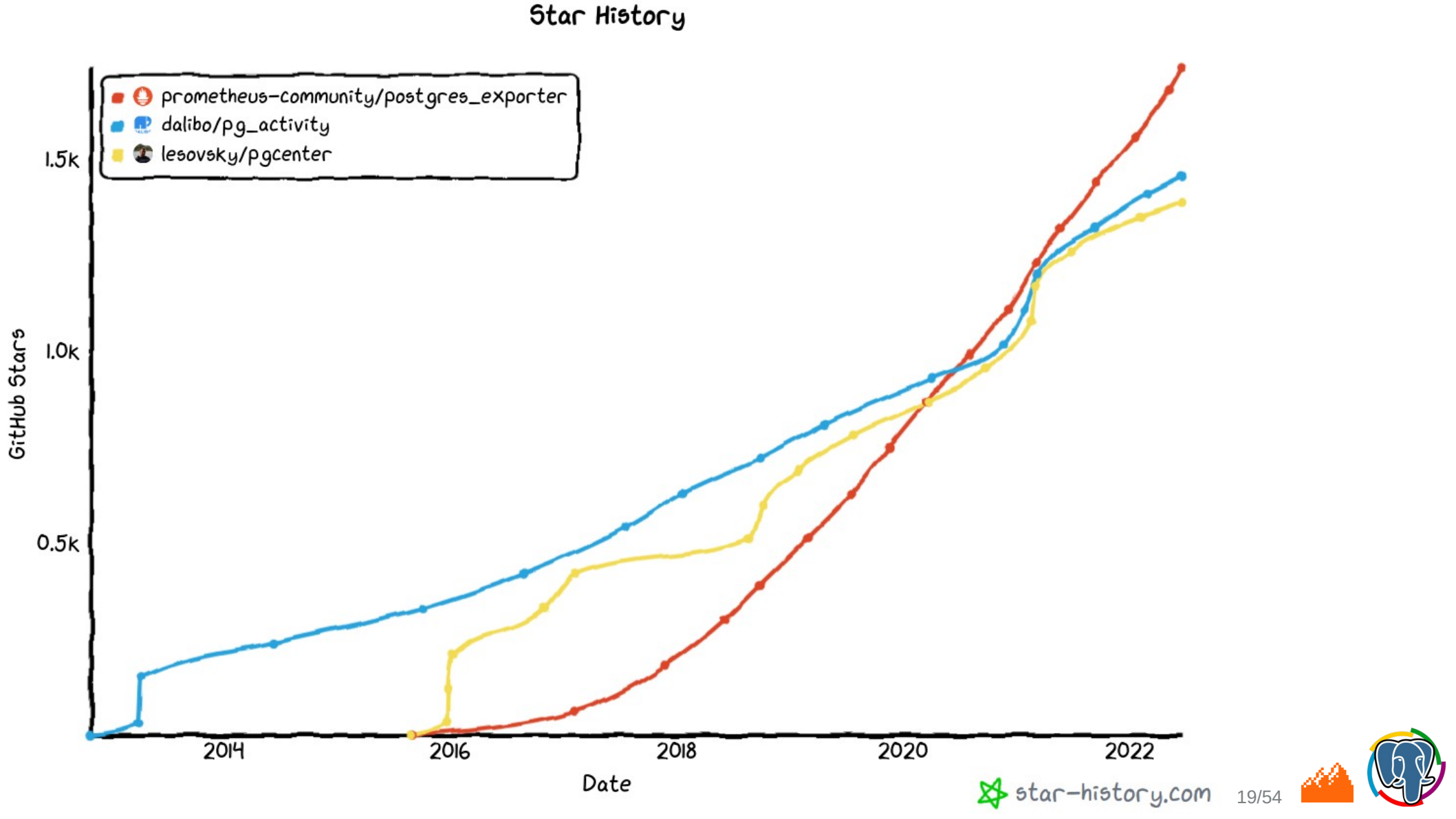
```
# HELP disk_usage_percent Usage of disk in percent (0-100)
# TYPE disk_usage_percent gauge
disk_usage_percent{partition="/"}    63.4
disk_usage_percent{partition="/var"} 37.6
disk_usage_percent{partition="/tmp"} 12.3

# HELP http_requests_total The total number of HTTP requests.
# TYPE http_requests_total counter
http_requests_total{method="GET", code="200"} 1234027 1655884333
http_requests_total{method="POST", code="200"}   1027 1655884333
http_requests_total{method="POST", code="400"}      3 1655884333
```


Compagnons

- Grafana : affichage des métriques (camembert, graphe, VU-mètre)
- Thanos : Stockage au long terme, « downsampling »
- Alertmanager : Routage d’alerte
- PushProx : Proxy pour accéder aux exporter derrière un pare-feu

La concurrence

- pg_view : tourne sur la machine
- pg_activity : statistiques instantanées
- pgcenter : interface similaire à «top», nécessite des droits
- pganalyse : pas opensource
- pganalyze/collector : exploite la vue pg_stat_plans
- pgSCV : permet des requêtes personnalisées


Star History
prometheus-community/postgres_exporter
dalibo/pg_activity
lesovsky/pgcenter
1.5k
1.0k
0.5k
GitHub Stars
2014
2016
2018
2020
2022
Date
star-history.com

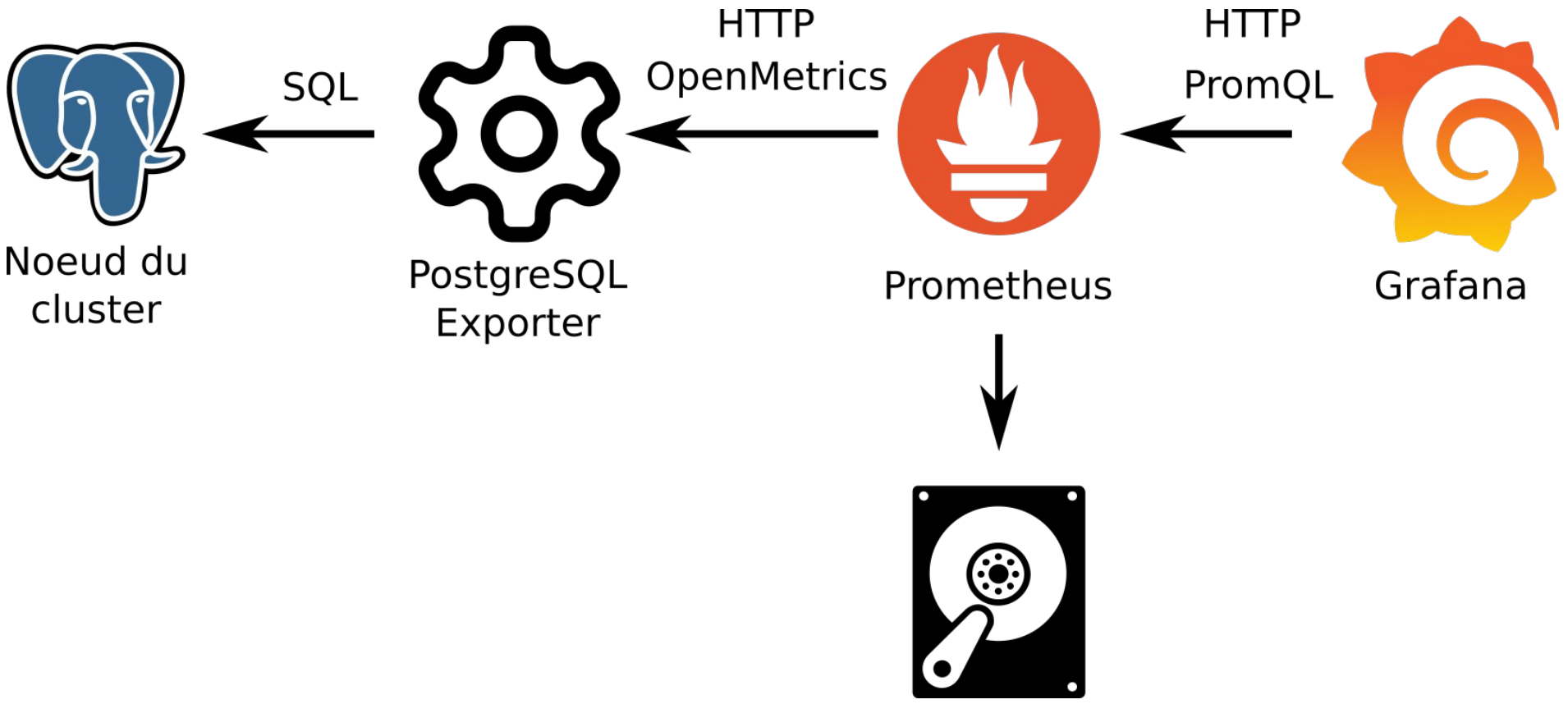
L’«exporter» Prometheus pour PostgreSQL

- Open Source :
  - https://github.com/prometheus-community/postgres_exporter
- Disponible sous forme de :
  - Binaire
  - Image Docker
- Développé en Golang
- Exploite les vues de PostgreSQL
- Expose les métriques au format OpenMetrics

L’«exporter» Prometheus pour PostgreSQL

- Sur un nœud du cluster PostgreSQL :
  - Binaire
  - Image Docker
- Machine de monitoring
- Cluster Kubernetes

L'«exporter» Prometheus pour PostgreSQL



Métriques standards

- Paramètres de configuration numériques
- Version, connectivité vers le serveur
- Vue `pg_locks` : agrégation par base de données
- Vue `pg_stat_activity` : Connexions par état et base de données
- Vue `pg_stat_bg_writer` :
  - Cache
  - Checkpoint

Métriques standards

- Vue `pg_stat_database` :
  - I/O
  - enregistrement lu, écrit, mis à jour
  - Transaction
  - fichier temporaire

Métriques standards

- Vue `pg_stat_archiver` :
  - échec, succès
  - durée depuis le dernier archivage

Ajout de métriques

- Une simple requête SQL permet de créer une nouvelle série temporelle :

```
SELECT datname,
       pg_database_size(datname) AS size_bytes
FROM pg_database
WHERE datname NOT IN ('template0', 'template1');
```


Ajout de métriques

```
pg_database:
  query: |
    SELECT datname, pg_database_size(datname) AS size_bytes
    FROM pg_database
    WHERE datname NOT IN ('template0', 'template1')
  master: true
  metrics:
    - size_bytes:
        usage: "GAUGE"
        description: "Database size in bytes"
    - datname:
        usage: "LABEL"
        description: "Name of current database"
```


Ajout de métriques

```
# HELP pg_database_size_bytes Database size in bytes
# TYPE pg_database_size_bytes gauge
pg_database_size_bytes{datname="postgres"} 8602115
pg_database_size_bytes{datname="db1"}      72385095
pg_database_size_bytes{datname="db2"}    618385754
```


Configuration de la connexion

- Par variable d'environnement : DATA_SOURCE_NAME
  - `postgresql://postgres:pgpass@postgres:5432/postgres?sslmode=disable`
  - `"user=postgres host=/var/run/postgresql/ sslmode=disable"`
- Autre variables disponibles :
  - DATA_SOURCE_URI : idem mais sans login/mot de passe
  - DATA_SOURCE_USER : nom d'utilisateur
  - DATA_SOURCE_PASS : mot de passe

Configuration du fichier de requêtes

- Par variable d'environnement ou de la ligne de commande :
  - --extend.query-path
  - PG_EXPORTER_EXTEND_QUERY_PATH
- Format YAML
- Correspondance des colonnes avec soit :
  - Métriques : «gauge» ou «counter»
  - Labels
- Nom de la métrique : concaténation de la section du YAML et du nom de colonne

Configuration du fichier de requêtes

```
pg_database:
  query: |
    SELECT datname, pg_database_size(datname) AS size_bytes
    FROM pg_database
    WHERE datname NOT IN ('template0', 'template1')
  master: true
  metrics:
    - size_bytes:
        usage: "GAUGE"
        description: "Database size in bytes"
    - datname:
        usage: "LABEL"
        description: "Name of current database"
```


Ajout de métriques

```
# HELP pg_database_size_bytes Database size in bytes
# TYPE pg_database_size_bytes gauge
pg_database_size_bytes{datname="postgres"} 8602115
pg_database_size_bytes{datname="db1"}      72385095
pg_database_size_bytes{datname="db2"}    618385754
```


Découverte automatique des bases de données

- Base de connexion
- Autres bases de données avec droit de connexion
  - --auto-discover-databases
- Paramètre `master`
- Possibilité d'exclure certaines bases ou de spécifier une liste précise :
  - --exclude-databases
  - --include-databases

Métriques au niveau du cluster - Réplication

- Vue pg_stat_replication
- Adresse et position des replicas
- Vu du «primary»

```
SELECT client_addr,
  coalesce(pg_wal_lsn_diff(pg_current_wal_flush_lsn(),
                           replay_lsn), 0) AS lag
FROM pg_stat_replication
WHERE state <> 'backup'
```

Métriques au niveau du cluster - Réplication

- Rôle d'un nœud :
  - Primary
  - Replica

```
SELECT
  CASE WHEN pg_is_in_recovery()
    THEN 1
    ELSE 0
  END
```


Métriques au niveau du cluster - Réplication

- Lag de réplication calculé à partir de la position dans le flux de réplication

```
SELECT
  CASE WHEN pg_is_in_recovery()
    THEN pg_wal_lsn_diff(pg_last_wal_replay_lsn(),
                         '0/0')
    ELSE pg_wal_lsn_diff(pg_current_wal_flush_lsn(),
                         '0/0')
  END
```


Métriques au niveau du cluster - Réplication

- Lag de réplication vu depuis le replica

```
SELECT
  CASE WHEN NOT pg_is_in_recovery()
    THEN -1
  WHEN pg_last_wal_receive_lsn() = pg_last_wal_replay_lsn()
    THEN 0
  ELSE EXTRACT (EPOCH FROM now() - pg_last_xact_replay_timestamp())
  END
```

Métriques au niveau du cluster - Réplication

```
pg_is_in:
  query: "SELECT CASE WHEN pg_is_in_recovery() THEN 1 ELSE 0 END AS recovery"
  master: true
  metrics:
    - recovery:
        usage: "GAUGE"
        description: "0 for primary, 1 for replicas"

pg_xlog:
  query: "SELECT CASE WHEN pg_is_in_recovery() THEN
pg_wal_lsn_diff(pg_last_wal_replay_lsn(), '0/0') ELSE
pg_wal_lsn_diff(pg_current_wal_flush_lsn(), '0/0') END AS position"
  master: true
  metrics:
    - position:
        usage: "COUNTER"
        description: "Position in the WAL"
```


Métriques au niveau du cluster - Réplication

```
pg_replication:
  query: "SELECT CASE WHEN NOT pg_is_in_recovery() THEN -1 WHEN pg_last_wal_receive_lsn() =
pg_last_wal_replay_lsn() THEN 0 ELSE EXTRACT (EPOCH FROM now() - pg_last_xact_replay_timestamp()) *
1000 END AS replag"
  master: true
  metrics:
    - replag:
        usage: "GAUGE"
        description: "Replication lag behind primary in milliseconds"

pg_replication_replay:
  query: "SELECT client_addr, coalesce(pg_wal_lsn_diff(pg_current_wal_flush_lsn(), replay_lsn), 0)
AS lag FROM pg_stat_replication WHERE state <> 'backup'"
  master: true
  metrics:
    - lag:
        usage: "GAUGE"
        description: "Replication lag behind primary in bytes"
    - client_addr:
        usage: "LABEL"
        description: "Address of replica"
```


Métriques au niveau du cluster - Réplication

- Alertes lié à la réplication
- Décalage entre les positions XLOG :
  - `max(pg_xlog_position) – min(pg_xlog_position) > 1000000000`
- Décalage en temps (> 1 min)
  - `pg_replication_replag > 60000`
- Décalage en données (> 10 Mo)
  - `pg_replication_replay_lag > 1000000`

Définition des alertes

- Configurer au niveau de Prometheus

```
groups:
- name: replication
  rules:
  - alert: PGReplicationLagXLOG
    expr: max(pg_xlog_position) - min(pg_xlog_position) > 1000000000
    labels:
      severity: critical
    annotations:
      summary: Replication Lag
```


Métriques au niveau du cluster - Connexions

- Vue `pg_stat_activity`
- Identifiant de processus, nom de base, nom du rôle, IP d'origine
- Champ "application name"
- Date et heure de :
  - connexion
  - début de transaction
  - début de requête
- État
- Requête en cours

Métriques au niveau du cluster - Connexions

```
SELECT count(datname) AS count,
  datname,
  usename,
  application_name AS appname,
  client_addr
FROM pg_stat_activity
WHERE backend_type = 'client backend'
GROUP BY datname, usename, client_addr, application_name
```


Métriques au niveau du cluster - Connexions

```
pg_client_connections:
  query: "SELECT count(datname) as count, datname, usename, application_name as appname,
client_addr FROM pg_stat_activity WHERE backend_type = 'client backend' GROUP BY datname, usename,
client_addr, application_name"
  master: true
  metrics:
    - datname:
        usage: "LABEL"
        description: "Name of the database"
    - usename:
        usage: "LABEL"
        description: "Username connected as"
    - appname:
        usage: "LABEL"
        description: "Application Name connection string"
    - client_addr:
        usage: "LABEL"
        description: "Client connection address"
    - count:
        usage: "GAUGE"
        description: "Clients connected"
```


Métriques au niveau du cluster - Connexions

- Alerte lié aux connexions
- Utilisation > 90 % du pool de connexions
  - `sum by (exported_server, instance, job, server) (pg_stat_database_numbackends) / pg_settings_max_connections > 0.9`

Métriques au niveau du cluster - Verrou

- Jointure entre la vue des verrou et la vue des connexions
- Permet de récupérer :
  - l'age du verrou
  - Requête liée au verrou

```
SELECT datname, relation::regclass, pl.pid, mode, query,
  xact_start
FROM pg_locks pl
LEFT JOIN pg_stat_activity psa ON pl.pid = psa.pid
WHERE datname = current_database()
AND granted = true
```


Métriques au niveau du tables

- Vue `pg_stat_user_tables` :
  - scan séquentiel, utilisation d'index
  - enregistrements lus, écrit, mis à jour
  - nombre de modification depuis le dernier `ANALYSE`
  - estimation du nombre d'enregistrement et des "dead tuples"
- Dernier événements de maintenance
  - VACUUM
  - ANALYSE

Métriques au niveau du tables - Tailles

- Utilisation des fonctions :
  - pg_indexes_size()
  - pg_total_relation_size()
- Récupération de la table «toast»

Métriques au niveau du tables - Tailles

```
SELECT current_database() AS datname,
  *,
  total_bytes - index_bytes - toast_bytes AS table_bytes
FROM (SELECT c.oid,nspname AS schemaname,
        relname,
        c.reltuples AS row_estimate,
        pg_total_relation_size(c.oid) AS total_bytes,
        pg_indexes_size(c.oid) AS index_bytes,
        COALESCE(pg_total_relation_size(reltoastrelid), 0) AS toast_bytes
      FROM pg_class c
      LEFT JOIN pg_namespace n ON n.oid = c.relnamespace
      WHERE relkind = 'r') a
```


Métriques au niveau des requêtes

- Vue pg_stat_statements
- 5000 requêtes les plus souvents lancées
- nécessite : `shared_preload_libraries = 'pg_stat_statements'`
- Texte factorisé de la requête
- Temps de planification (total, min, max, moyen)
- Temps d'execution
- Nombre d'execution
- Utilisation du cache local et partagé
- Écriture dans les fichiers WAL

Métriques au niveau des requêtes

- 5000 séries temporelles pour :
  - Temps de planification
  - Temps d’exécution
  - Utilisation du cache
- Nécessité de réduire le nombre de série temporelles en prenant le top 25 pour :
  - Temps de planification + exécution
  - Nombre d’enregistrement
  - Fréquence d’exécution

Métriques au niveau des requêtes

```
SELECT *,
  regexp_replace(substring(query FOR 100), '[ \t\n]+', ' ', 'g') AS query,
  calls
FROM pg_stat_statements
ORDER BY calls DESC LIMIT 25

SELECT *,
  regexp_replace(substring(query FOR 100), '[ \t\n]+', ' ', 'g') AS query,
  (total_exec_time + total_plan_time) / 1000 AS total_time_seconds,
FROM pg_stat_statements
ORDER BY total_time_seconds DESC LIMIT 25
```


Et après ?

- Création de séries temporelles à partir de données métiers
- Corrélation de séries temporelles :
  - Systèmes
  - Applicatives
  - Métiers
- Possibilité de développer des nouveaux «exporter»

Merci !

- Démo disponible sur github.com/Vampouille/demo-prometheus-postgres
- Des questions ?