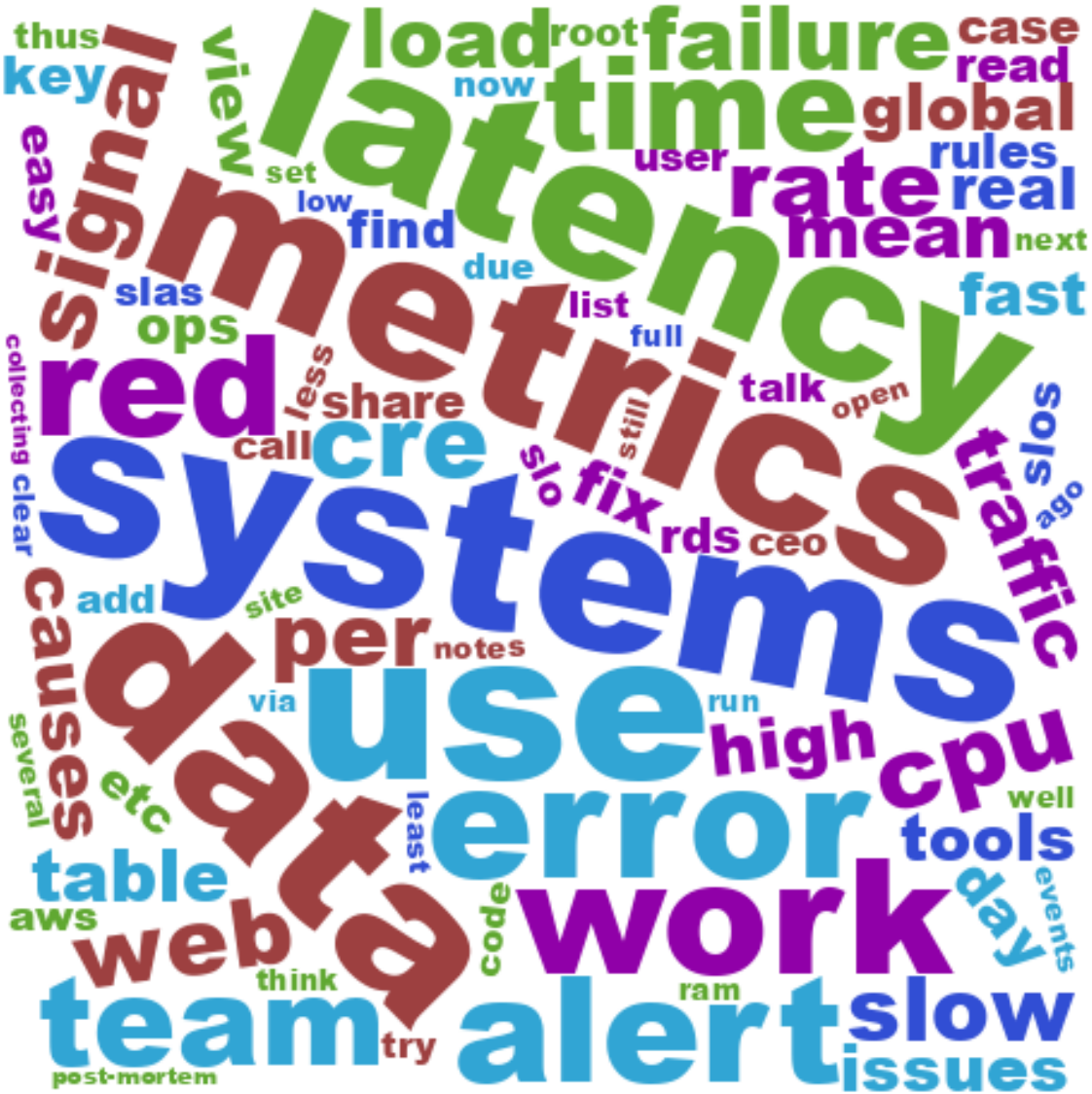
Using Prometheus and Grafana to build a Postgres Dashboard

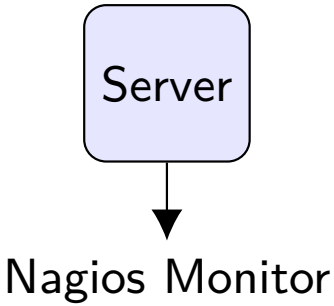
Gregory Stark

October 25, 2018

What is Monitoring?



Old School



New School

- We want to alert on global properties such as
  - The fraction of the fleet currently operating well
  - The average response time across the fleet
  - The consistency of the data across the fleet
- We want to alert based on historical data
  - Average rates over time period
  - Compare current data with 24h ago or 7d ago
- We want to alert on comparisons between services
  - Ratio of rates of transactions in database to application requests
  - Are there any database servers for which S3 does not contain a recent backup

The Tools

- Prometheus

  Database specifically designed for handling time series. It performs recorded queries regularly to synthesize new time series and to generate alerts.

- Alertmanager

  Part of Prometheus project. Handles generating notifications for alerts.

- node_exporter

  Agent for system statsitics. For more agents see: https://prometheus.io/docs/instrumenting/exporters/

- postgres_exporter

  Agent that exports statistics from `pg_stat_*` *views*

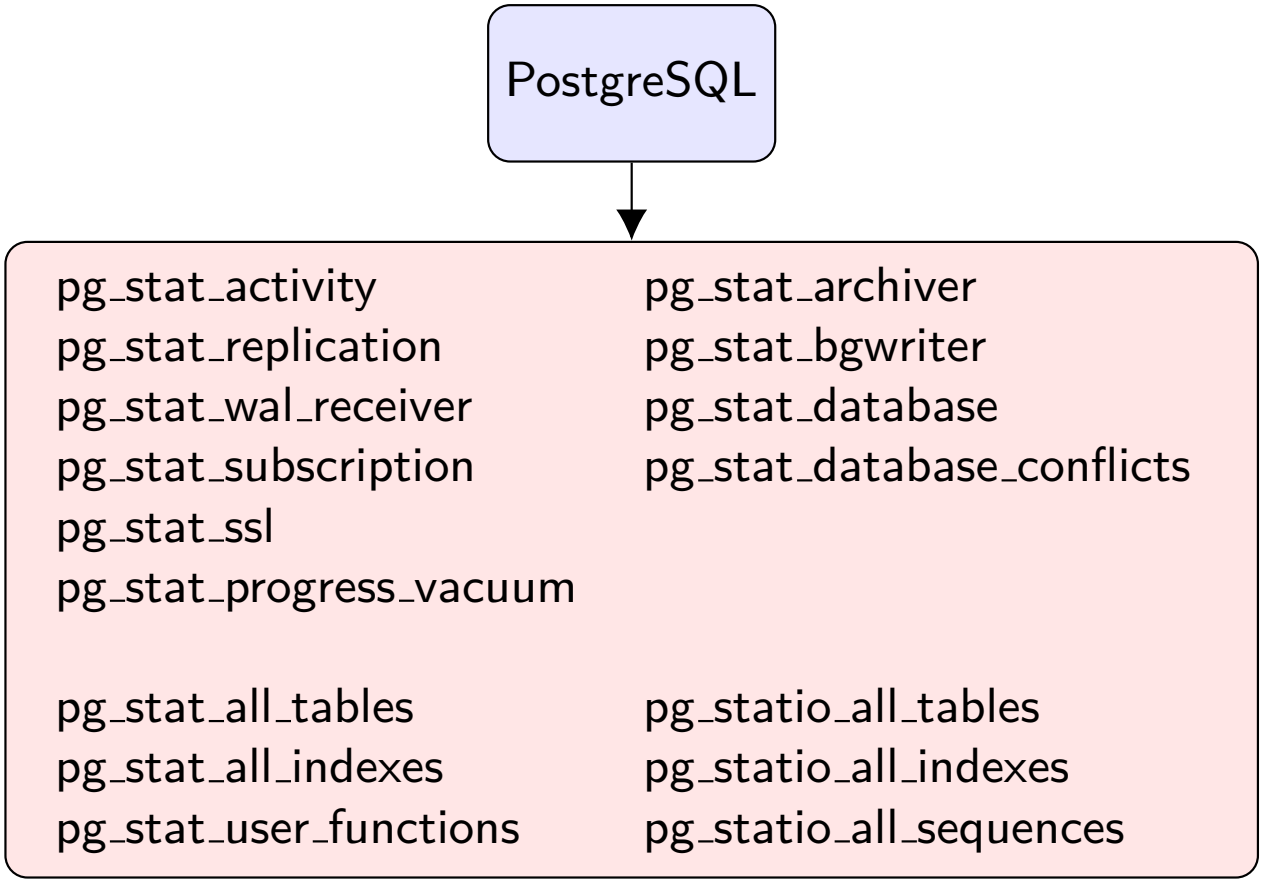
- mtail

  Useful to fill gaps where Postgres doesn't provide a statistics views to expose them. e.g. `log_min_duration`, `log_lock_waits`

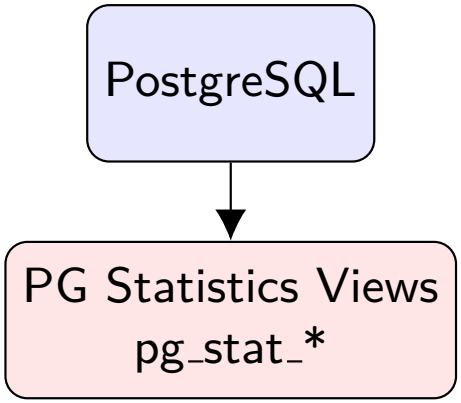
- Grafana

  WYSIWYG dashboard software.

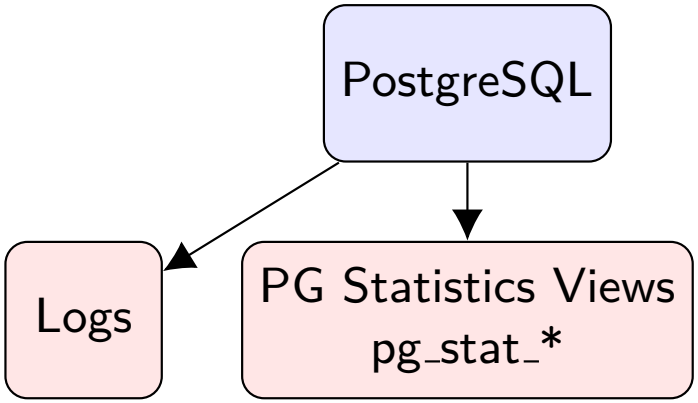
The Data Flow



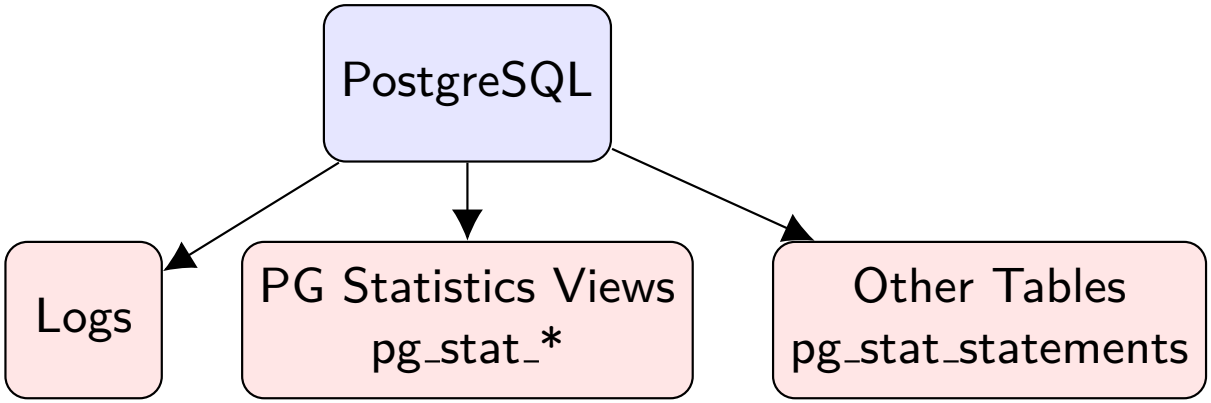
The Data Flow



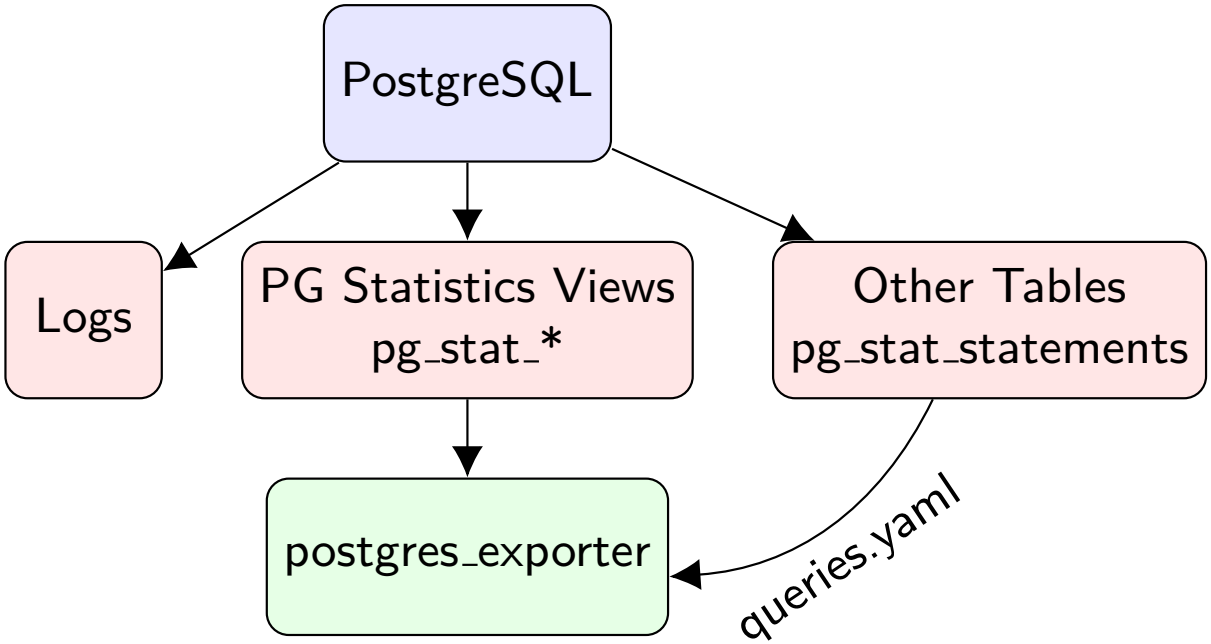
The Data Flow



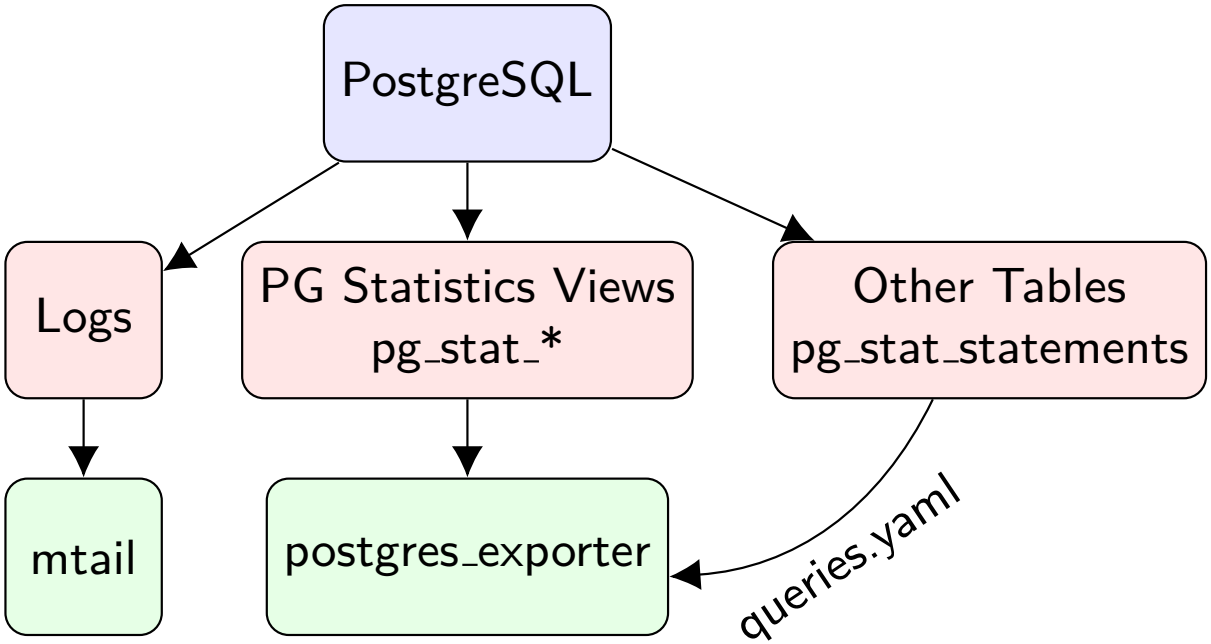
The Data Flow



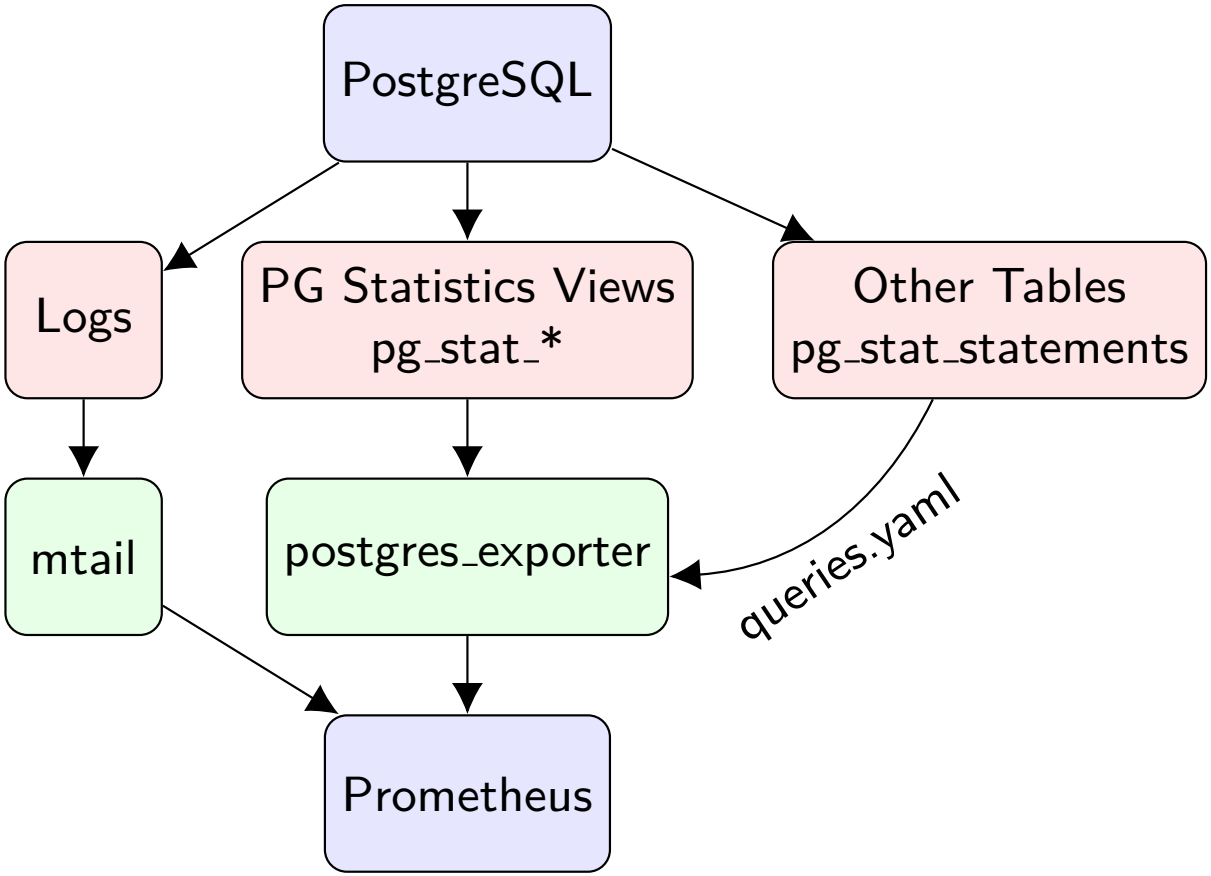
The Data Flow



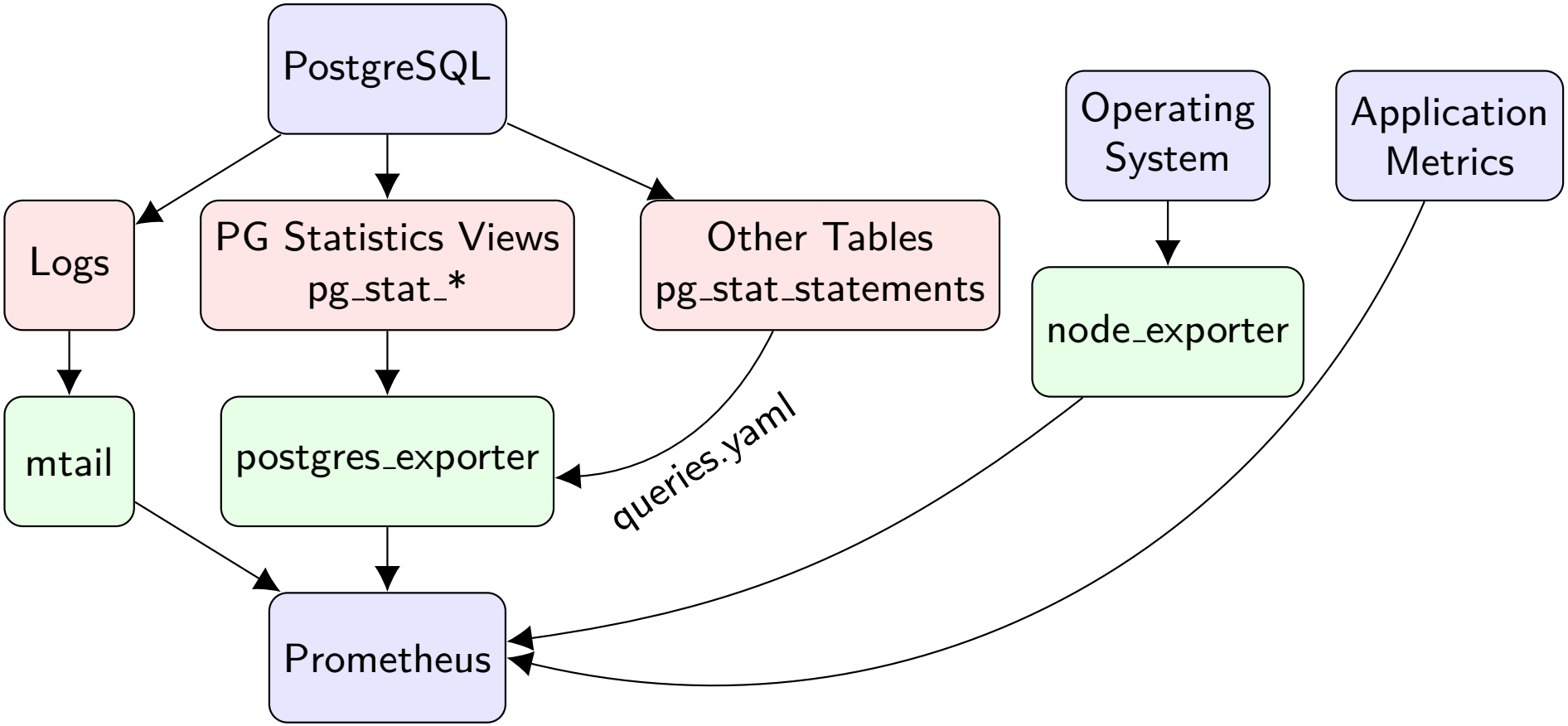
The Data Flow



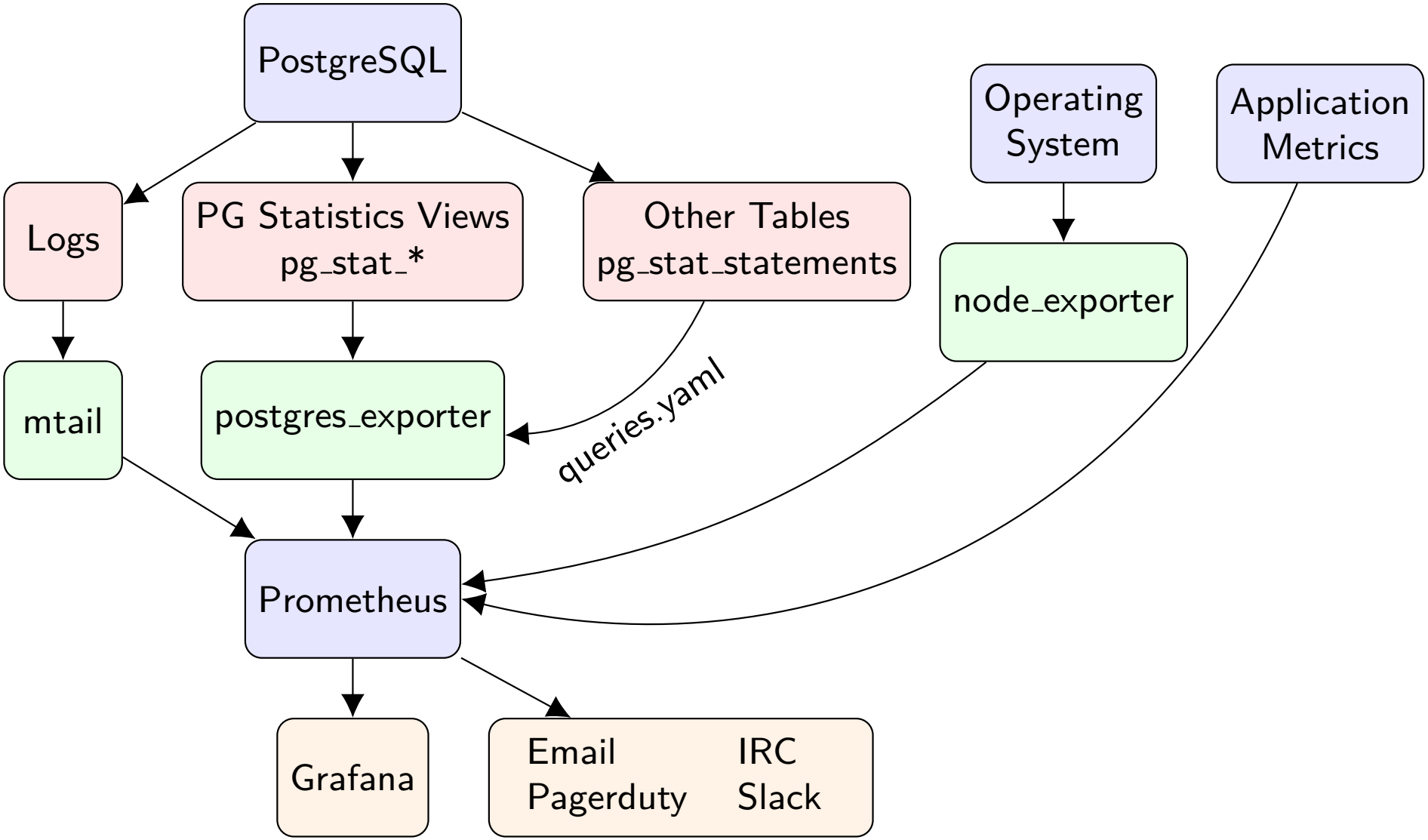
The Data Flow



The Data Flow



The Data Flow



USE

The USE method uses three key metrics for each component of a complex system:

- Utilization
- Saturation
- Errors

It was published in ACMQ as Thinking Methodically about Performance (2012):
https://queue.acm.org/detail.cfm?id=2413037

Further discussion:
http://www.brendangregg.com/usemethod.html

Presented at FISL13:
http://dtrace.org/blogs/brendan/2012/09/21/fisl13-the-use-method/

RED

The RED model uses latency (duration) instead of utilization:

- Rate
- Errors
- Duration

From:
https://www.weave.works/blog/the-red-method-key-metrics-for-microservices-architecture/

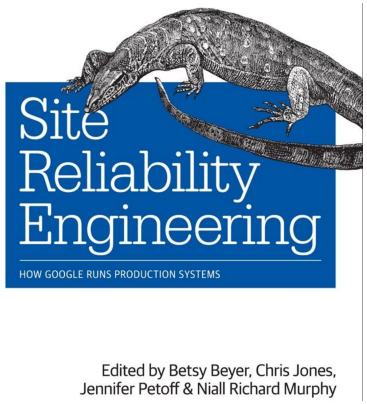
See also:
https://www.vividcortex.com/blog/monitoring-and-observability-with-use-and-red

Google's SRE Golden Signals

SRE Golden Signals are very similar:

- Latency
- Traffic
- Errors
- Saturation

Orginally published in Site Reliability Book:



Also see discussion at:
https://medium.com/devopslinks/how-to-monitor-the-sre-golden-signals-1391cadc7524

PromQL

Prometheus Alerts Graph Status Help

Enable query history

pg_stat_activity_count

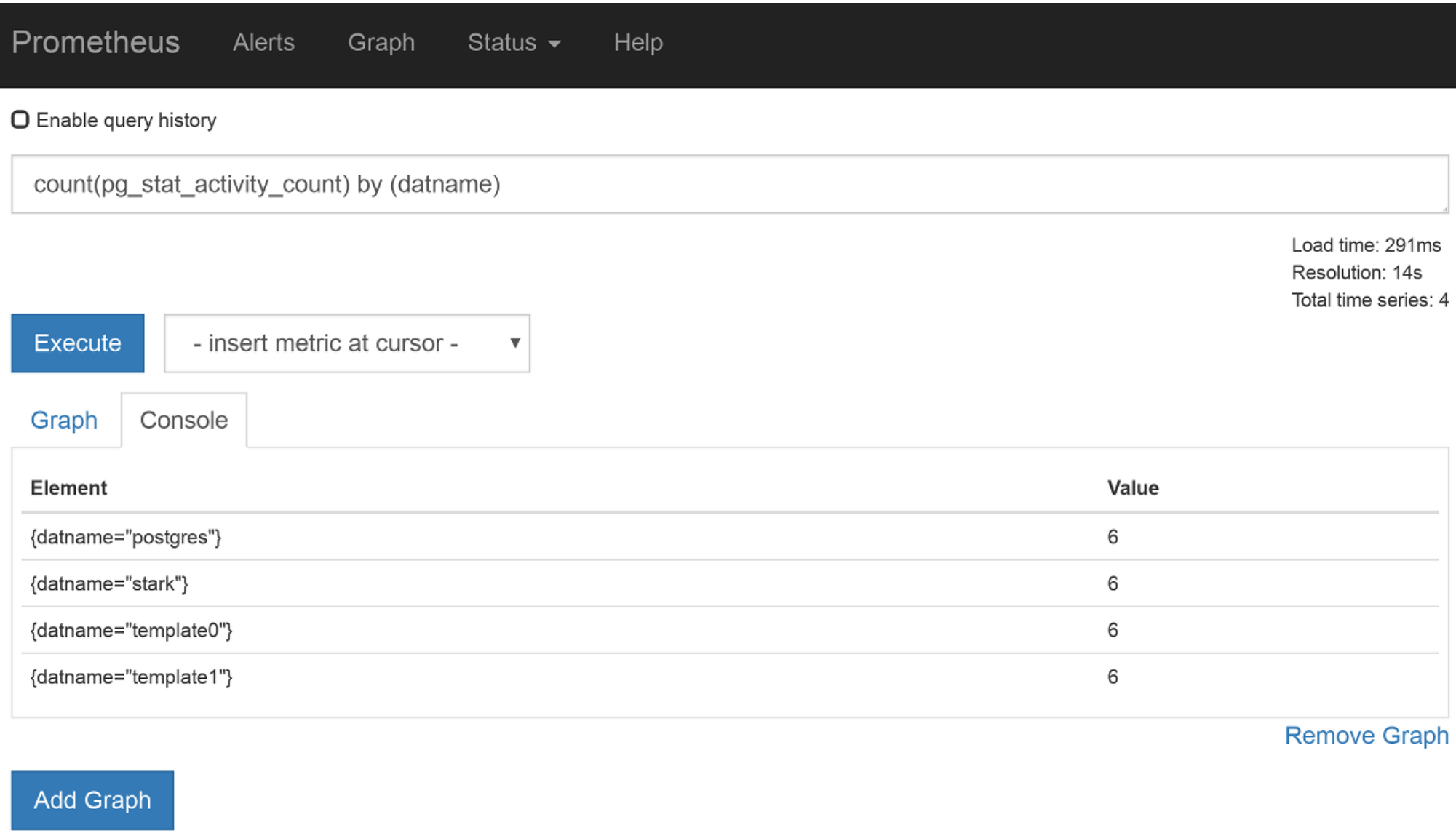
Load time: 290ms
Resolution: 14s
Total time series: 24

Execute - insert metric at cursor -

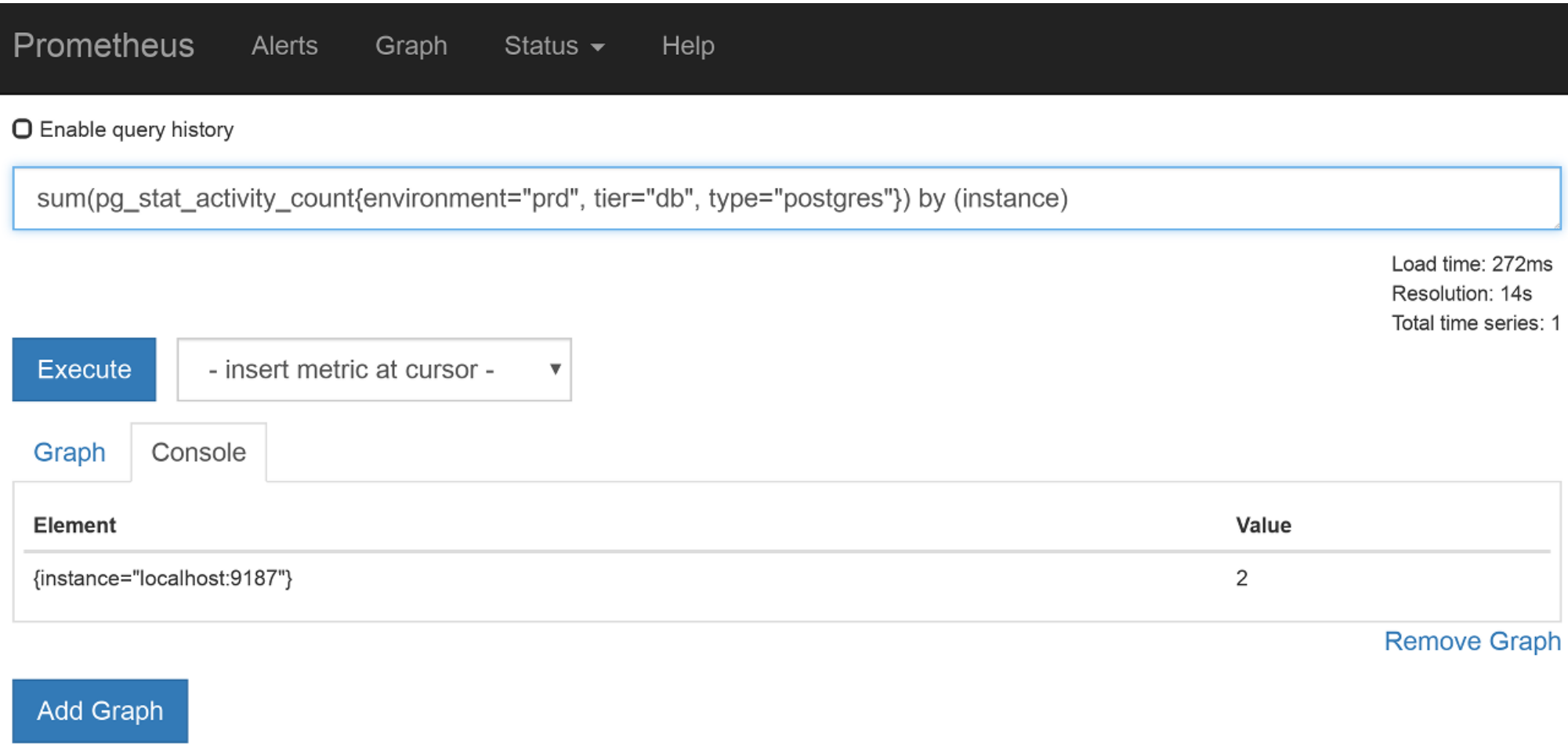
Graph Console

Element	Value
pg_stat_activity_count{datname="postgres",environment="prd",instance="localhost:9187",job="postgres",state="active",tier="db",type="postgres"}	0
pg_stat_activity_count{datname="postgres",environment="prd",instance="localhost:9187",job="postgres",state="disabled",tier="db",type="postgres"}	0
pg_stat_activity_count{datname="postgres",environment="prd",instance="localhost:9187",job="postgres",state="fastpath function call",tier="db",type="postgres"}	0
pg_stat_activity_count{datname="postgres",environment="prd",instance="localhost:9187",job="postgres",state="idle",tier="db",type="postgres"}	0
pg_stat_activity_count{datname="postgres",environment="prd",instance="localhost:9187",job="postgres",state="idle in transaction",tier="db",type="postgres"}	0
pg_stat_activity_count{datname="postgres",environment="prd",instance="localhost:9187",job="postgres",state="idle in transaction (aborted)",tier="db",type="postgres"}	0
pg_stat_activity_count{datname="stark",environment="prd",instance="localhost:9187",job="postgres",state="active",tier="db",type="postgres"}	1

PromQL


Prometheus
Alerts
Graph
Status
Help
Enable query history
count(pg_stat_activity_count) by (datname)
Load time: 291ms
Resolution: 14s
Total time series: 4
Execute
- insert metric at cursor -
Graph
Console
Element
Value
{datname="postgres"}
6
{datname="stark"}
6
{datname="template0"}
6
{datname="template1"}
6
Remove Graph
Add Graph

PromQL


Prometheus
Alerts
Graph
Status
Help
Enable query history
sum(pg_stat_activity_count{environment="prd", tier="db", type="postgres"}) by (instance)
Load time: 272ms
Resolution: 14s
Total time series: 1
Execute
- insert metric at cursor -
Graph
Console
Element
Value
{instance="localhost:9187"}
2
Remove Graph
Add Graph

Alerts - Rate

```
groups:
- name: postgresql.rules
  rules:
  - alert: PostgreSQL_CommitRateTooLow
    expr: |
      rate(pg_stat_database_xact_commit{datname="gitlabhq_production",
                                        environment="prd"}[1m]) < 1000
    for: 2m
    labels:
      severity: warn
      channel: database
    annotations:
      description: |
        Commits/s on {{$labels.instance}} database {{$labels.datname}}
        is {{$value | printf "%.0f" }} which is implausibly low.
        Perhaps the application is unable to connect
      runbook: troubleshooting/postgresql.md#availability
      title: 'Postgres seems to be processing very few transactions'
```

Alerts - Errors

```
groups:
- name: postgresql.rules
  rules:
  - alert: PostgreSQL_RollbackRateTooHigh
    expr: |
      rate(pg_stat_database_xact_rollback{datname="gitlabhq_production"}[5m])
        / ON(instance, datname)
      rate(pg_stat_database_xact_commit{datname="gitlabhq_production"}[5m])
        > 0.02
    for: 5m
    labels:
      severity: warn
      channel: database
    annotations:
      description: |
        Ratio of transactions being aborted compared to committed is
        {{$value | printf "%.2f" }} on {{$labels.instance}}
      runbook: troubleshooting/postgresql.md#errors
      title: 'Postgres transaction rollback rate is high'
```

Alerts - Saturation

```
groups:
- name: postgresql.rules
  rules:
  - alert: PostgreSQL_ConnectionsTooHigh
    expr: |
      sum(pg_stat_activity_count) BY (environment, instance)
        > ON(instance)
      pg_settings_max_connections * 0.75
    for: 10m
    labels:
      severity: warn
      channel: database
    annotations:
      runbook: troubleshooting/postgresql.md#connections
      title: |
        Postgres has {{$value}} connections on {{$labels.instance}}
        which is close to the maximum
```

Alerts - more Errors

```
# Count of specific types of errors -- notably statement timeouts
counter postgresql_logs_total by severity
counter postgresql_errors_total by type

/^[0-9_:.-]* [a-z0-9-]* postgresql: (?P<date>\d\d\d\d-\d\d-\d\d \d\d:\d\d:\d\d [A-Z]{3}) \[[0-9]*\]:
  \[[0-9]*-1\] (?P<severity>DEBUG[1-5]|INFO|NOTICE|WARNING|ERROR|LOG|FATAL|PANIC):  / {
    postgresql_logs_total[$severity]++

    /ERROR:  (?P<message>.*)$/ {
      /canceling statement due to statement timeout/ {
        postgresql_errors_total["statement_timeout"]++
      }
      /canceling autovacuum task/ {
        postgresql_errors_total["canceled_autovacuum"]++
      }
      /deadlock detected/ {
        postgresql_errors_total["deadlock_detected"]++
      }
      /duplicate key value violates unique constraint/ {
        postgresql_errors_total["duplicate_key"]++
      }
      otherwise {
        postgresql_errors_total["other"]++
      }
    }
}
```

Alerts

```
groups:
- name: postgresql.rules
  rules:
  - alert: PostgreSQL_StatementTimeout_Errors
    expr: |
      rate(postgresql_errors_total{type="statement_timeout"}[1m]) > 0.5
    for: 5m
    labels:
      severity: warn
      channel: database
    annotations:
      descrition: |
        Database {{$labels.instance}} is logging
        {{ $value | printf "%.1f" }} statement timeouts per second
      runbook: troubleshooting/postgresql.md#errors
      title: 'Postgres transactions showing high rate of statement timeouts'
```

Alerts - Exposing hidden problems

```
pg_replication:
  query: |
    SELECT EXTRACT(epoch FROM (
             now() - pg_last_xact_replay_timestamp()
           ))::int AS lag,
           CASE WHEN pg_is_in_recovery() THEN 1 ELSE 0 END AS is_replica
  metrics:
    - lag:
        usage: "GAUGE"
        description: "Replication lag behind primary in seconds"
    - is_replica:
        usage: "GAUGE"
        description: "Indicates if this host is a replica"
```

Alerts - Exposing hidden problems

```
groups:
- name: postgresql.rules
  rules:
  - alert: PostgreSQL_ReplicationLagTooLarge
    expr: |
      (pg_replication_lag > 120)
        AND ON(instance)
      (pg_replication_is_replica == 1)
    annotations:
      description: |
        Replication lag on server {{$labels.instance}} is currently
        {{$value | humanizeDuration }}
      runbook: troubleshooting/postgres.md#replication-is-lagging-or-has-stopped
      title: 'Postgres Replication lag is over 2 minutes'
```

Alerts - Exposing hidden problems

```
pg_replication_slots:
  query: |
    SELECT slot_name, slot_type,
           case when active then 1.0 else 0.0 end AS active,
           age(xmin) AS xmin_age,
           age(catalog_xmin) AS catalog_xmin_age,
      FROM pg_replication_slots
  metrics:
    - slot_name:
        usage: "LABEL"
        description: "Slot Name"
    - slot_type:
        usage: "LABEL"
        description: "Slot Type"
    - active:
        usage: "GAUGE"
        description: "Boolean flag indicating whether this slot has a consumer streaming from it"
    - xmin_age:
        usage: "GAUGE"
        description: "Age of oldest transaction that cannot be vacuumed due to this replica"
    - catalog_xmin_age:
        usage: "GAUGE"
        description: "Age of oldest transaction that cannot be vacuumed from catalogs due to this replica (use
```

Alerts - Exposing hidden problems

```
groups:
- name: postgresql.rules
  rules:
  - alert: PostgreSQL_UnusedReplicationSlot
    expr: 'pg_replication_slots_active == 0'
    for: 30m
    labels:
      severity: warn
      channel: database
    annotations:
      description: |
        Unused {{$labels.slot_type}} slot "{{$labels.slot_name}}"
        on {{$labels.instance}}
```

Alerts - Exposing hidden problems

```
- alert: PostgreSQL_SplitBrain
  expr: 'count(pg_replication_is_replica == 0) BY (environment) != 1'
  annotations:
    title: |
      Split Brain: more than one postgres databases in environment
      {{$labels.environment}} in read-write (primary) mode

- alert: PostgreSQL_SplitBrain_Replicas
  expr: |
    count(
      count(pg_stat_wal_receiver_status >= 0) BY (environment, upstream_host)
    ) BY (environment) > 1
  annotations:
    title: |
      Split Brain: replicas in environment {{$labels.environment}}
      have different upstream databases configured
```

Alerts - Miscellaneous

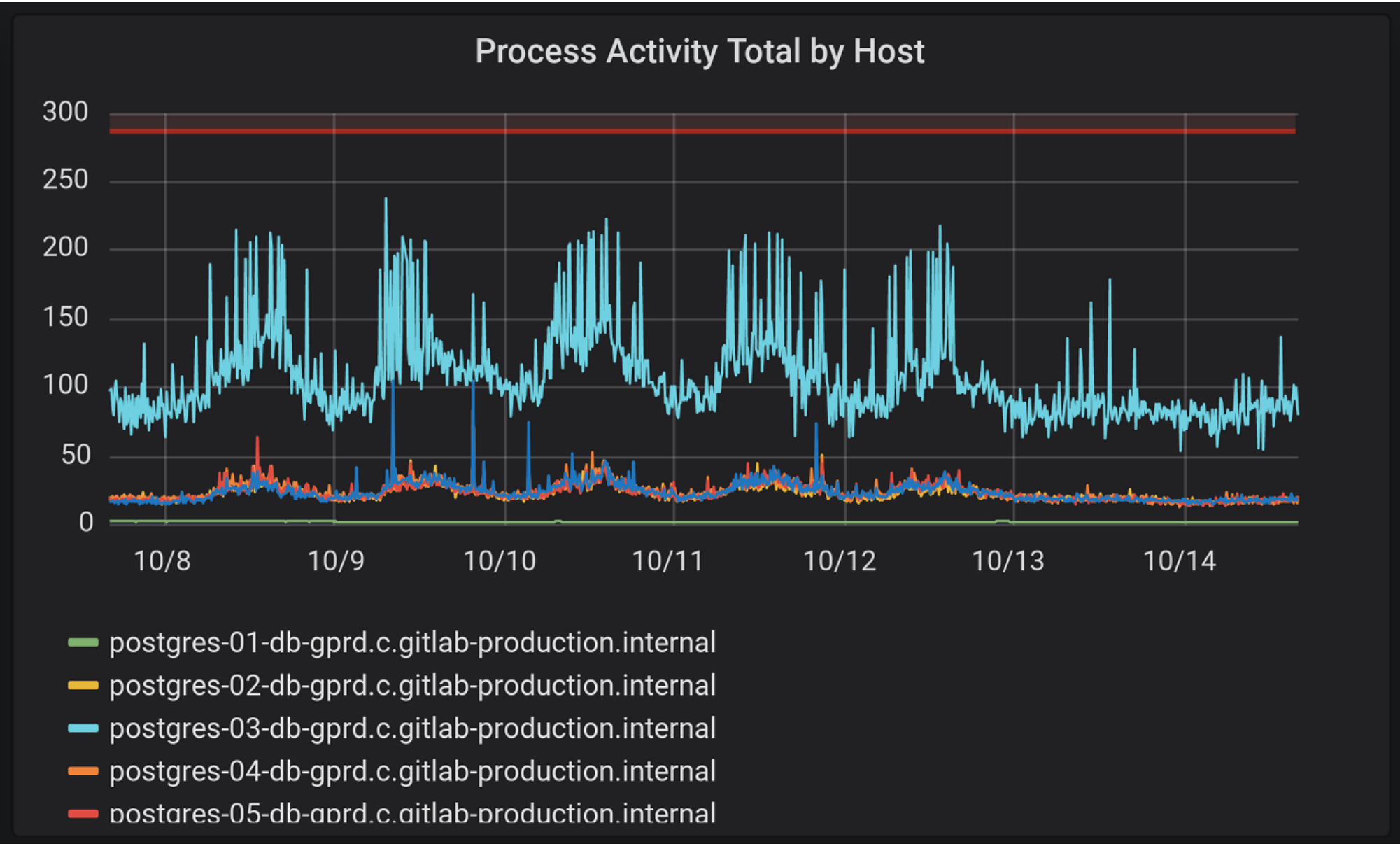
```
- alert: PostgreSQL_FleetSizeChange
  expr: 'postgres:databases != postgres:databases OFFSET 2m'
  annotations:
    description: 'There are now {{$value}} databases in "{{$labels.environment}}"'
    title: 'Number of PostgreSQL Databases in {{$labels.environment}} has changed'

- alert: PostgreSQL_RoleChange
  expr: 'pg_replication_is_replica and changes(pg_replication_is_replica[1m]) > 0'
    title: 'Postgres Database replica promotion occurred in "{{$labels.environment}
```

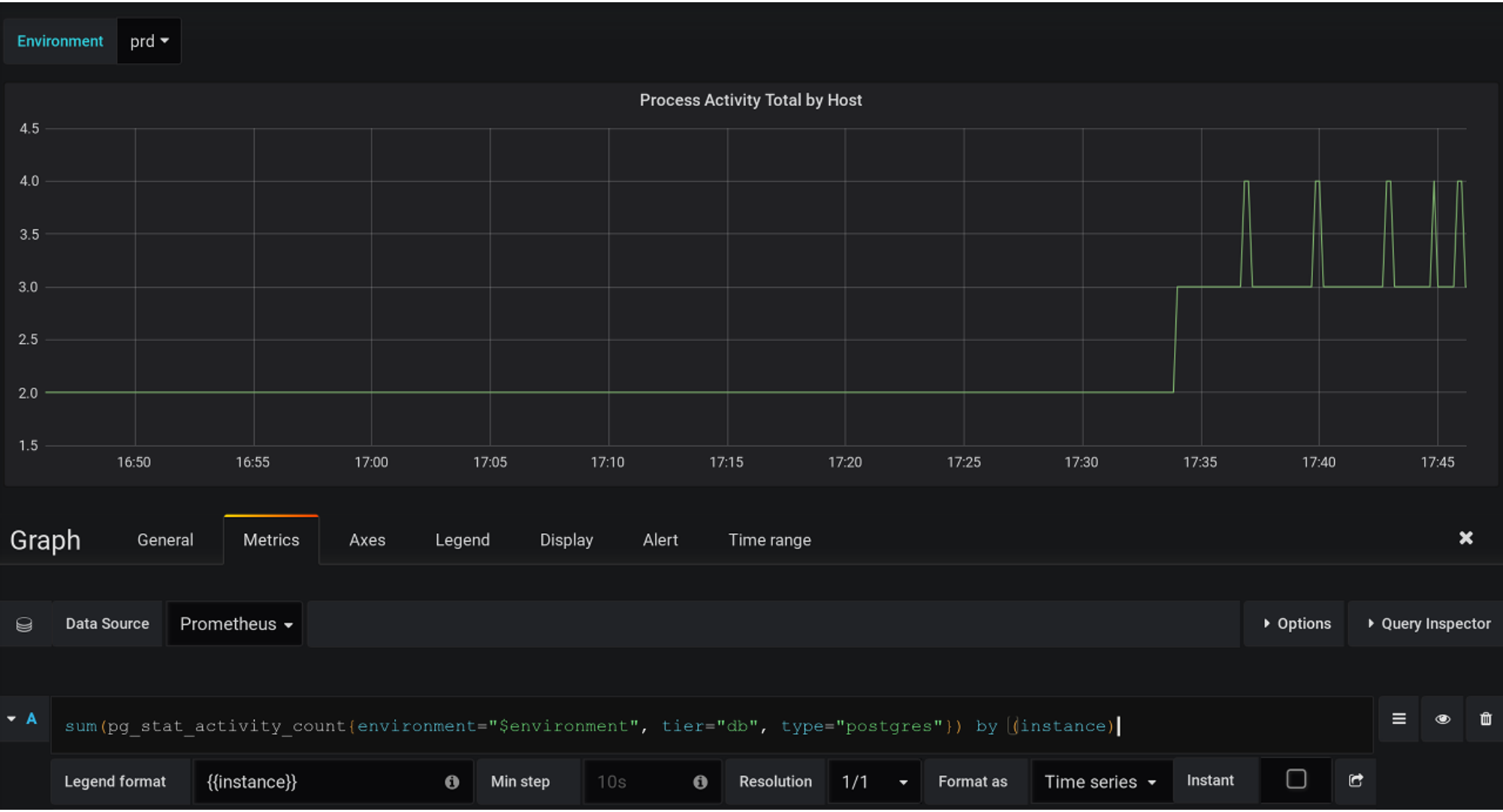
```
- alert: PostgreSQL_ConfigurationChange
  expr: |
        {__name__=~"pg_settings_.*"} !=
        ON(__name__, instance)
        {__name__=~"pg_settings_.*",__name__!="pg_settings_transaction_read_only"}
        OFFSET 10m
```

The GUI Dashboard

The GUI Dashboard



The GUI Dashboard



pg_stat_statements

```
pg_stat_statements:
  query: |
    SELECT
      pg_get_userbyid(userid) as user,
      pg_database.datname,
      pg_stat_statements.queryid,
      pg_stat_statements.calls,
      pg_stat_statements.total_time as time_milliseconds,
      pg_stat_statements.rows,
      pg_stat_statements.shared_blks_hit,
      pg_stat_statements.shared_blks_read,
      pg_stat_statements.shared_blks_dirtied,
      pg_stat_statements.shared_blks_written,
      pg_stat_statements.local_blks_hit,
      pg_stat_statements.local_blks_read,
      pg_stat_statements.local_blks_dirtied,
      pg_stat_statements.local_blks_written,
      pg_stat_statements.temp_blks_read,
      pg_stat_statements.temp_blks_written,
      pg_stat_statements.blk_read_time,
      pg_stat_statements.blk_write_time
      FROM pg_stat_statements
      JOIN pg_database
        ON pg_database.oid = pg_stat_statements.dbid
```

pg_stat_statements

```
metrics:
  - user:
      usage: "LABEL"
      description: "The user who executed the statement"
  - datname:
      usage: "LABEL"
      description: "The database in which the statement was executed"
  - queryid:
      usage: "LABEL"
      description: "Internal hash code, computed from the statement's parse tree"
  - calls:
      usage: "COUNTER"
      description: "Number of times executed"
  - time_milliseconds:
      usage: "COUNTER"
      description: "Total time spent in the statement, in milliseconds"
  - rows:
      usage: "COUNTER"
      description: "Total number of rows retrieved or affected by the statement"
  - shared_blks_hit:
      usage: "COUNTER"
      description: "Total number of shared block cache hits by the statement"
  - shared_blks_read:
      usage: "COUNTER"
      description: "Total number of shared blocks read by the statement"
  - shared_blks_dirtied:
      usage: "COUNTER"
      description: "Total number of shared blocks dirtied by the statement"
  - shared_blks_written:
      usage: "COUNTER"
      description: "Total number of shared blocks written by the statement"
  - local_blks_hit:
      usage: "COUNTER"
      description: "Total number of local block cache hits by the statement"
```

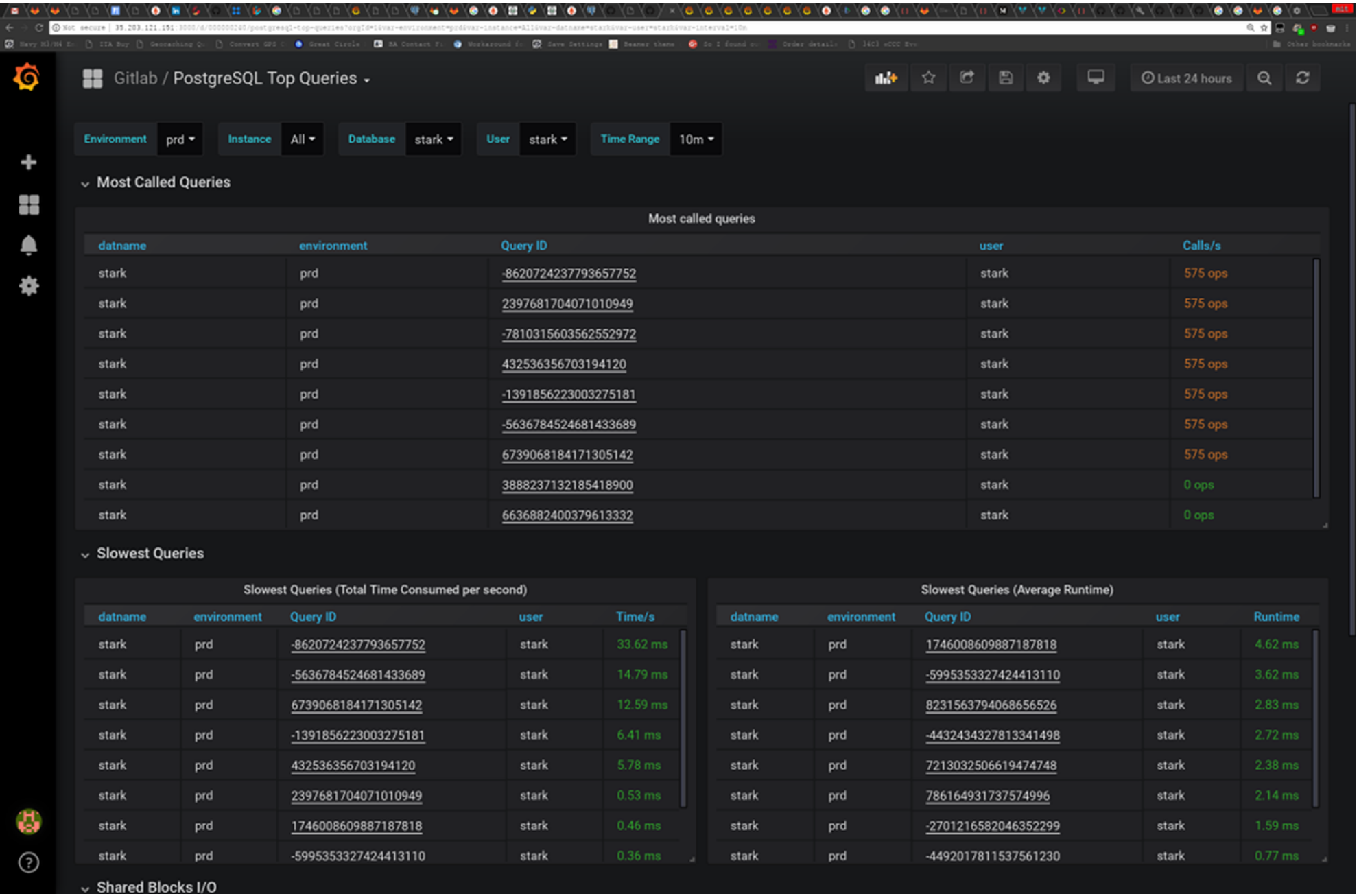
pg_stat_statements

This has some issues with Cardinality....

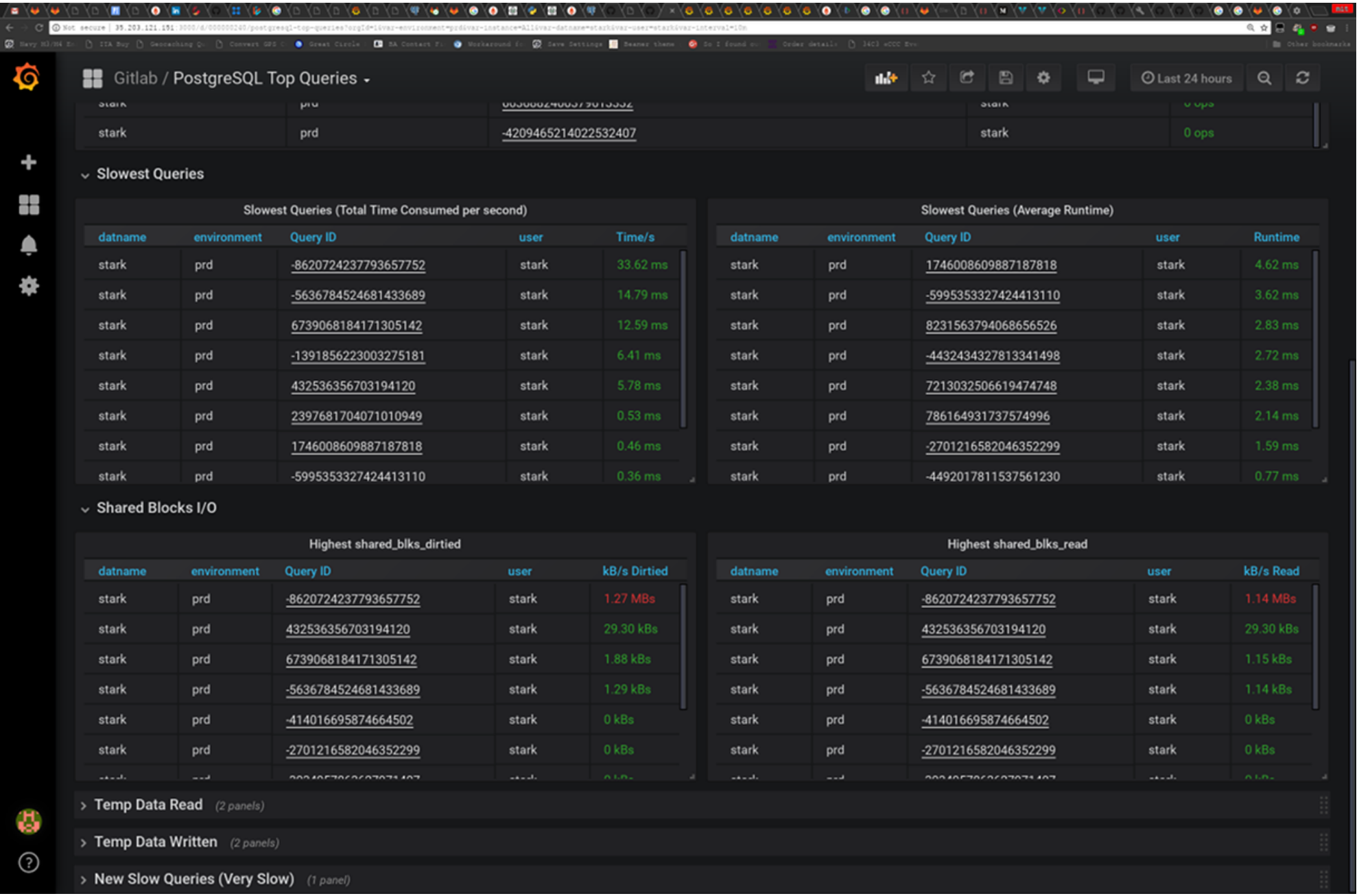
- 15 metrics
- for each of 5000 queryids (or more)
- for each database
- every 15s

This can quickly become performance issue for Prometheus.

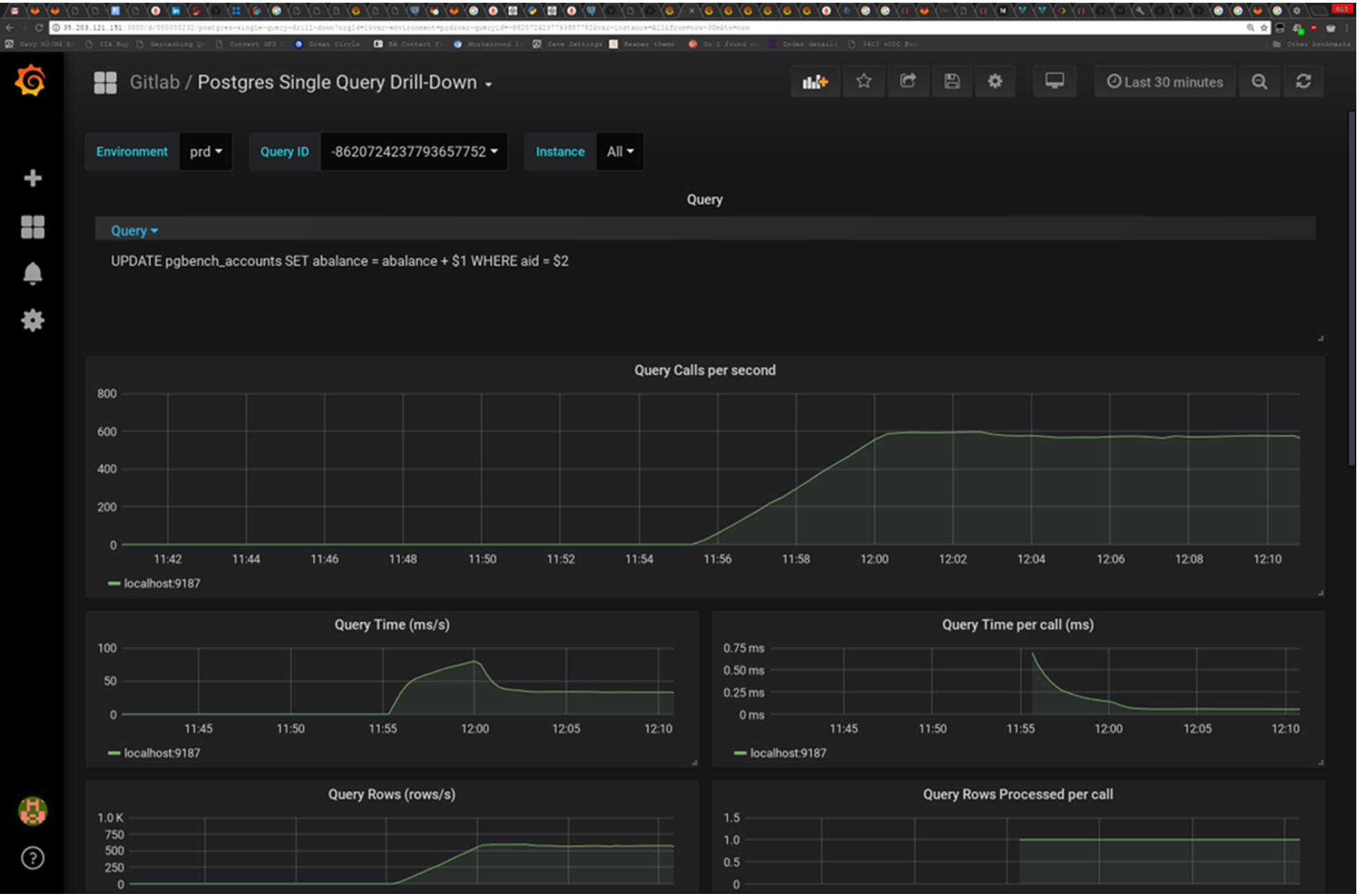
pg_stat_statements



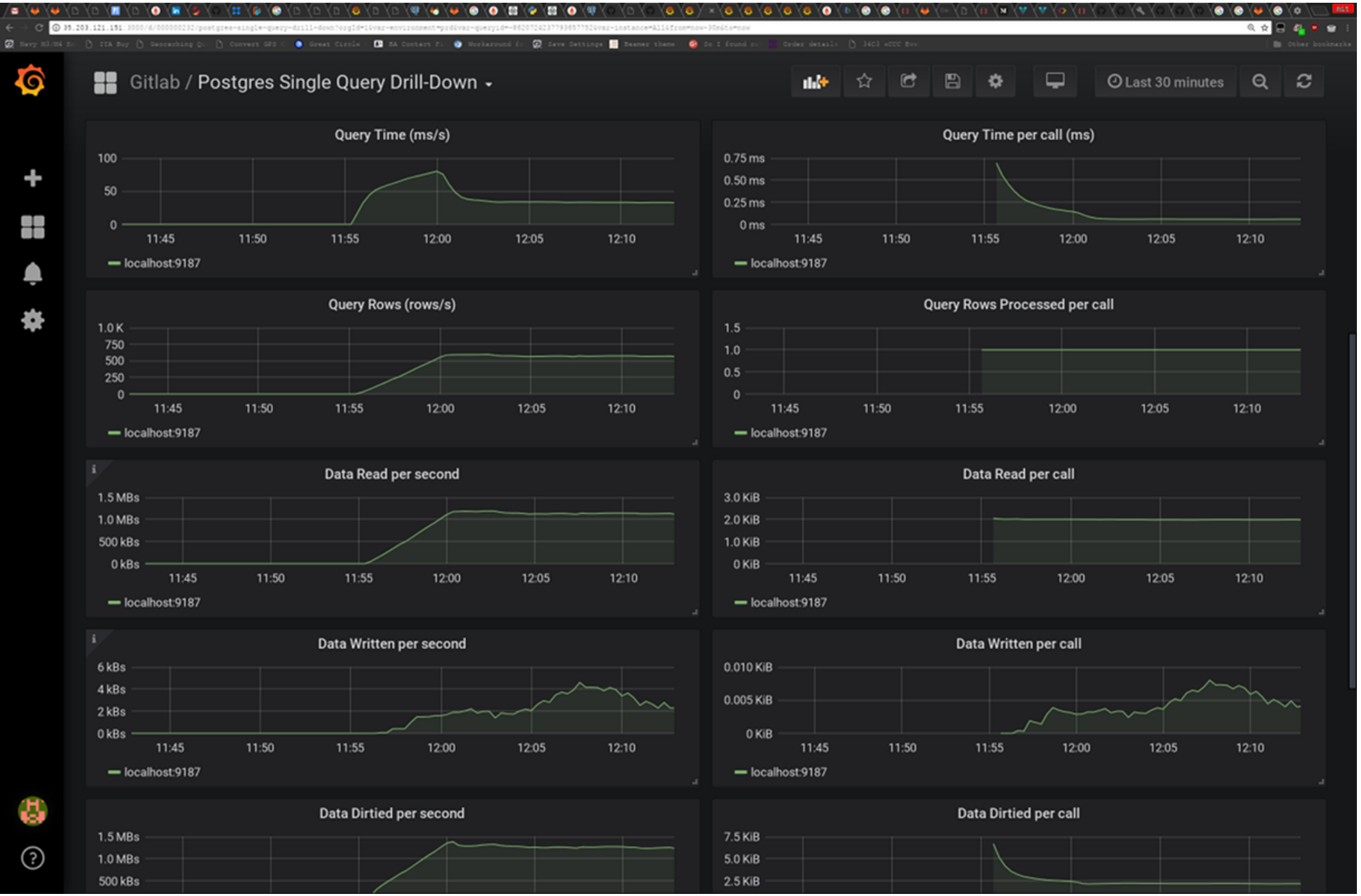
pg_stat_statements



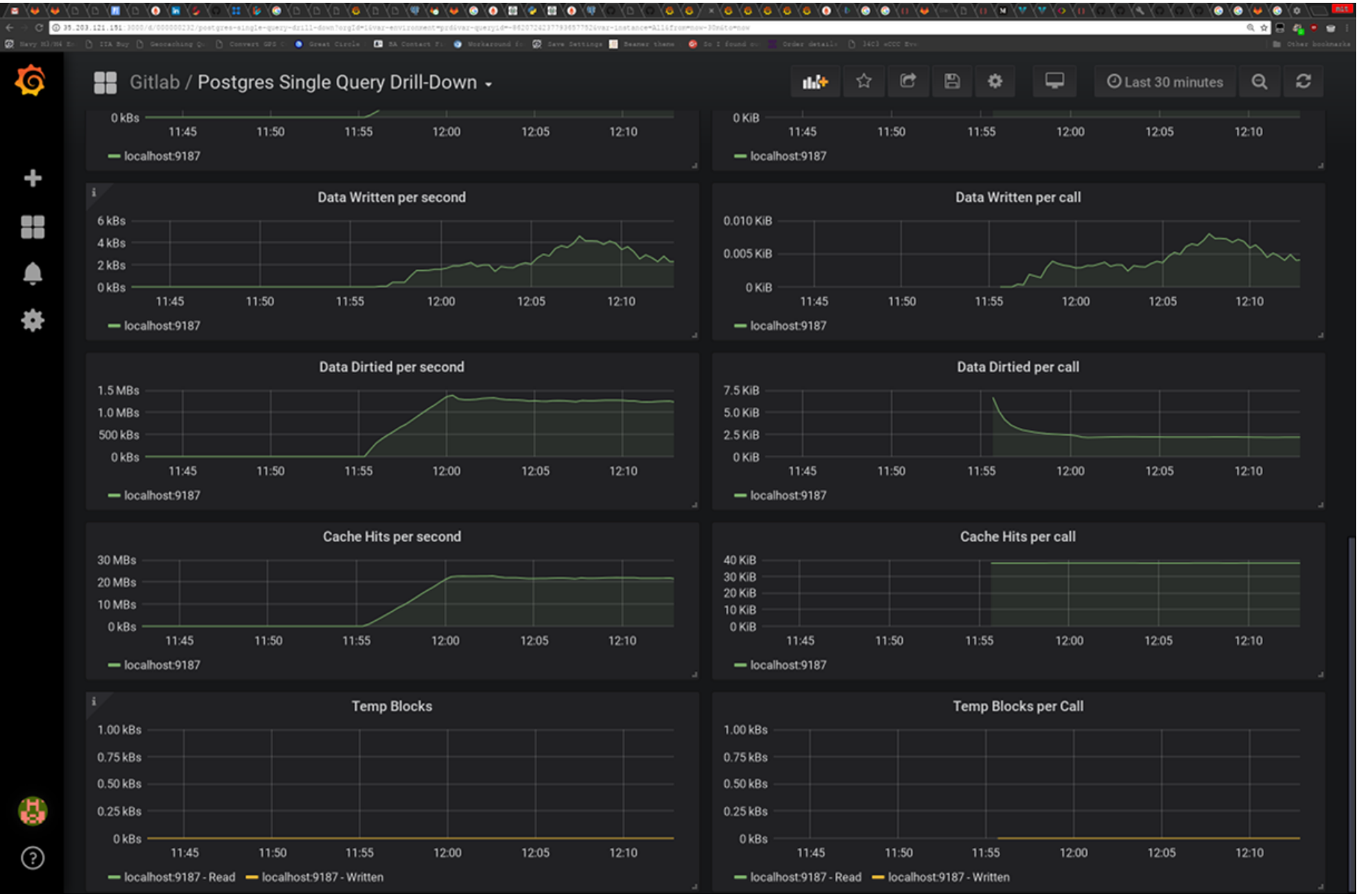
pg_stat_statements



pg_stat_statements



pg_stat_statements



Future things to address

- Missing stats in postgres_exporter
  Queries.yaml requires a good understanding of Postgres *and* Prometheus to write. It also makes rules and dashboards non-portable which is a major downside. It's considered an "anti-pattern" in Prometheus exporter world.

- Missing data in `pg_stats_*`: errors, lock timing, slow queries
  If you have more please tell me, I'll be working on adding these in the future.

- Saturation is basically impossible to measure in Postgres
  `pg_stat_activity` does not really represent saturation very well when applications keep persistent connections and use pooling of any form. If you filter on `state=active` it's useful but still very coarse and incomplete representation. There's a pgbouncer exporter as well and you can add instrumentation to your application to address this. But it would be good to identify standard ways of measuring Postgres saturation.

Future things to address

- postgres_exporter should be eliminated entirely
  It would be much preferable to have Postgres speak common monitoring protocols directly. That would make the statistics more consistent, reliable, and easier to deploy.

- Distributed Tracing
  This is different from but complementary to monitoring and is a major gap that would help expose the connections between database metrics and application metrics.

More information

- This presentation is online at:
  `https://_stark.gitlab.io/monitoring-postgres-pgconf.eu-2018/monitoring.pdf`
- Gitlab Project for presentation at:
  `https://gitlab.com/_stark/monitoring-postgres-pgconf.eu-2018`
- Source code for presentation at:
  `https://gitlab.com/_stark/monitoring-postgres-pgconf.eu-2018.git`
- Author contact address:
  `stark@mit.edu`