ADVANCED MONITORING OF LARGE-SCALE POSTGRESQL DEPLOYMENT IN TOMTOM

pgconf.eu 2017

TOMTOM

Agenda

- About us
- TomTom – what do we do?
- Why monitoring is important?
- Who should monitor?
- What should we monitor?
- Metrics & Tools
- What changes when hundreds of databases are to be monitored?
- Conclusion

About Us

Rafał Hawrylak

rafal.hawrylak@tomtom.com

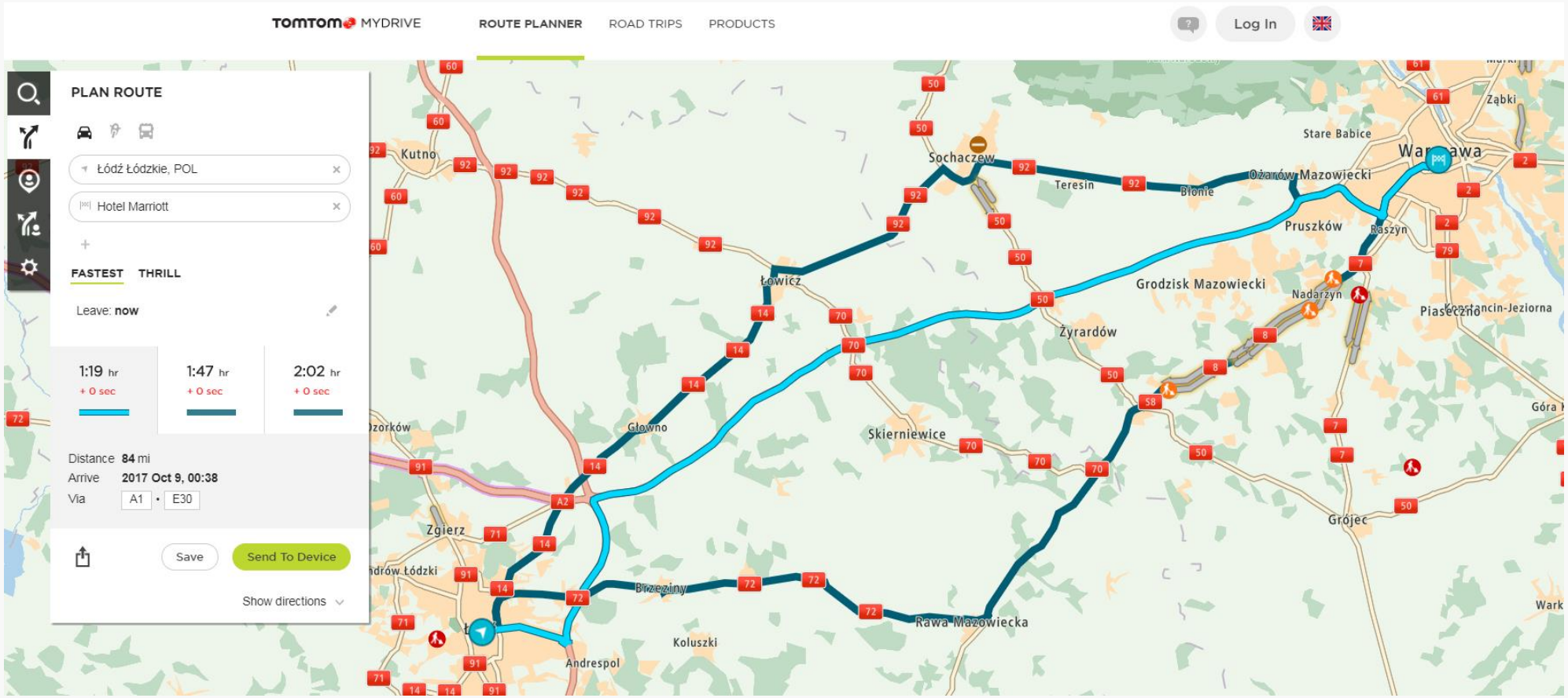
Software developer and database expert

Michał Gutkowski

michal.gutkowski@tomtom.com

Software engineer solving problems with Java, Python, Bash and SQL

About Us – We Are From Łódź!



We are from Lodz, Poland!

TomTom – What do we do?

TomTom - What do we do?

- Database with spatial features
- Transactional and versioned changes
- Massive automated tools editing map
- 2000+ of manual editors
- Billions of map objects

MapMaking Platform in 2017

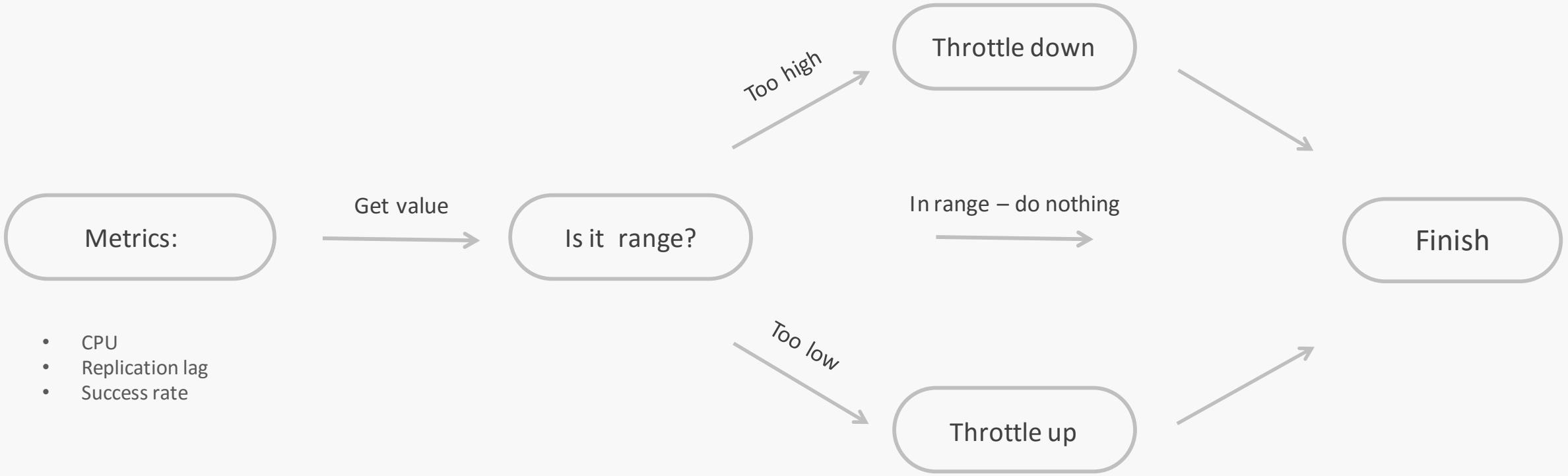
- PostgreSQL 9.5 + Postgis 2.2
- 150+ database machines – 32 cores, 256GB RAM, ssd drives in RAID
- Storage – 160TB
- Daily db size increase – 200GB
- Inserted rows count 15k per second
- Queries count – over 600k per second

TOMTOM

Why monitoring is important?

- System health-check and maintenance
- Alerting and reliable notification system
- Detect performance regression – software, configuration, hardware changes
- Software optimisations: queries, batching
- Cost-efficiency – run it cheaper
- Business process improvements: scheduling, task queues, separate users with priorities
- Adjust business processes – self healing system

Adjust business process



- CPU
- Replication lag
- Success rate

Who should monitor?

- Production monitoring team is responsible for catching incidents
- Database team is responsible for administration, maintenance and tuning
- Every developer or tester has access to metrics from production environment
- Teams are responsible for delivering changes in software and database
- Full development cycle: design, implementation, deployment and monitoring on production

- Top-down responsibility

TOMTOM

What should we monitor?

- Collect both business and low level metrics (Kibana, Prometheus, Munin)
- Alerting should be built on top of business metrics

Throughput

Success rate

- Low level metrics should be used for root cause analysis

Queries

Storage

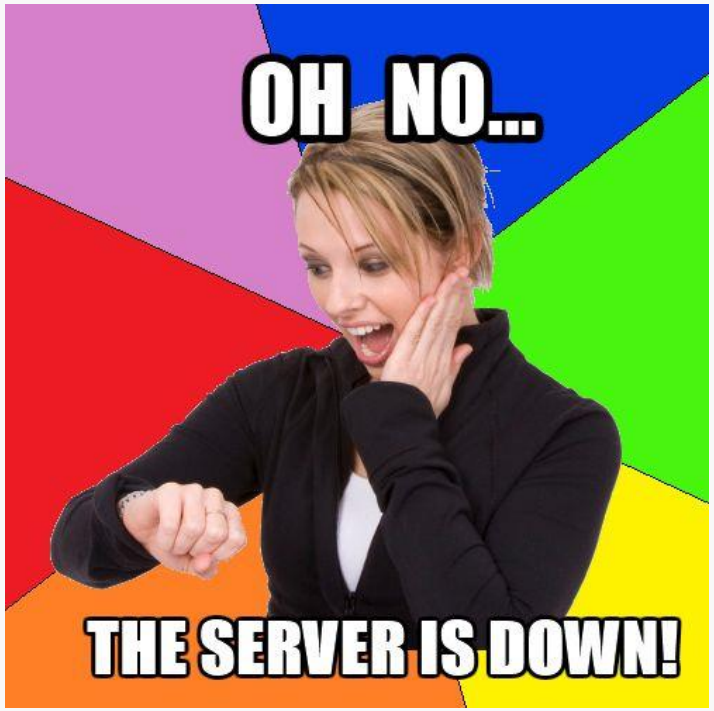
Replication

Operational system

TOMTOM

Alerting

- Define rules and thresholds for metrics
- Remember – only business metrics for alerting!
- Use automated notification (~~e-mails~~, Slack or PagerDuty)
- Sample business metrics:
  - Health check:
    - Success rate
  - Performance
    - Application response times
    - Requests per second



Monitoring: which metrics are important

Queries

- Connections
- Active queries
- Query statistics

Other

- Errors
- Statistics of tables
- Statistics of indexes

Storage

- Locks
- Objects size
- Vacuum processes
- Bloat
- bg writer and checkpoints

Replication

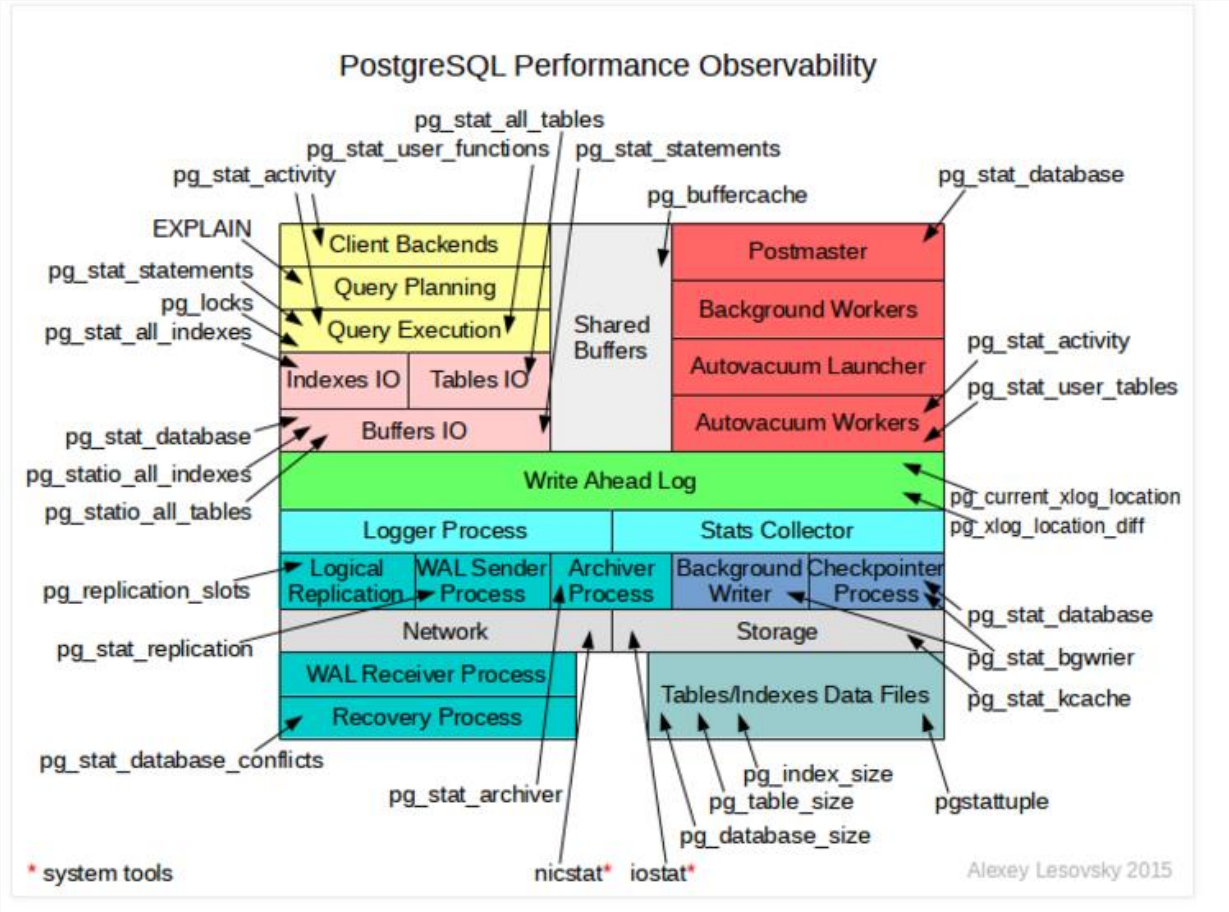
- Lag

Operational system

- CPU
- Disk IO
- Memory
- Network

TOMTOM

Monitoring: postgresql internals



http://blog.postgresql-consulting.com/2015/11/postgresql-observability-views.html

Monitoring: connections

Why?

- Indicate problems in higher tiers (application services are down, network problems)
- Changes in usage pattern of application layer (raised number of connections)
- More connections mean more resources utilized

How?

- SELECT usename, count(1) FROM pg_stat_activity WHERE state <> 'idle' GROUP BY 1 ORDER BY 2 desc;
- pg_view

Monitoring: pg_view

```
mastercore54-1 up 11 days, 22:23:21 32 cores Linux 4.9.32-15.41.amzn1.x86_64 load average 0.66 0.61 1.84
sys: utime 12.6  stime 1.3   idle 84.3  iowait 1.7   ctxt 20209 run 7   block 0
mem: total 240.1GB free 1.5GB  buffers 217.1MB cached 163.8GB dirty 640KB  limit 139.7GB as 10.3GB left 129.5GB
/var/lib/pgsql/9.5/data/core54 9.5.7 master connections: 393 of 7200 allocated, 11 active
type dev       fill total    left   read  write    await path_size path
data dm-0       0.0 5.1TB   2.4TB   91.4    0.0   1301.8     2.5TB /var/lib/pgsql/9.5/data/core54
xlog nvme3n1    0.0 1.7TB 175.6GB    0.0    0.0      0.0     1.4TB /var/lib/pgsql/9.5/data/core54/pg_xlog
  pid type    s utime stime guest read write   age  uss db  user        query
25610 backend R  14.6   1.8   0.0  3.2   0.0 00:00 15.5 cpp cpp_mdssnap WITH attribute_values_ids AS ( VALUES ('e2737fc
47238 backend S  14.6   0.9   0.0  4.3   0.0 00:00 15.2 cpp cpp_mdssnap WITH attribute_values_ids AS ( VALUES ('a373ed1
51933 backend S   0.9   0.0   0.0  0.0   0.0 00:00 20.3 cpp cpp_mdssnap WITH attribute_values_ids AS (VALUES ('eaf3b897
53241 backend S   2.7   1.8   0.0  1.4   0.0 00:00 20.6 cpp cpp_mdssnap WITH attribute_values_ids AS (VALUES ('a40a3aef
53703 backend S   5.5   0.0   0.0  1.6   0.0 00:00 20.2 cpp cpp_mdssnap WITH attribute_values_ids AS ( VALUES ('a1adbce
56416 backend S   3.7   0.0   0.0  1.2   0.0 00:00 15.7 cpp cpp_mdssnap WITH attribute_values_ids AS (VALUES ('82e3da24
59391 backend S   0.0   0.0   0.0  0.0   0.0 00:00 20.3 cpp cpp_mdssnap WITH attribute_values_ids AS (VALUES ('d816df16
59418 backend S   4.6   0.0   0.0  0.8   0.0 00:00 20.1 cpp cpp_mdssnap WITH attribute_values_ids AS (VALUES ('f93f4500
59483 backend S   8.2   0.0   0.0  0.5   0.0 00:00 11.3 cpp cpp_mdssnap  SELECT allMatching.id, branch, version, source
59634 backend S  10.0   1.8   0.0  2.0   0.0 00:00  8.7 cpp cpp_mdssnap WITH attribute_values_ids AS ( VALUES ('d3d1e16
60980 backend S   8.2   0.9   0.0  0.5   0.0 00:00 10.2 cpp cpp_mdssnap WITH attribute_values_ids AS ( VALUES ('cde4e9c
```

https://github.com/zalando/pg_view

TOMTOM

Monitoring: active queries

Why?

- Indicate currently long running queries
- Overview types of queries currently running

How?

- SELECT query, count(1) FROM pg_stat_activity WHEREstate <> 'idle' GROUP BY 1 ORDER BY 2 desc;
- pg_view
- pg_activity

TOMTOM

Monitoring: pg_activity

```
 PostgreSQL 9.5.7 - mastercore55-1 - postgres@localhost:5432/postgres - Ref.: 2s
  Size:     1.21T -      3.75M/s          | TPS:          3923
  Mem.:    34.30% -     80.27G/240.10G    | IO Max:     43242/s
  Swap:     0.00% -      0.00B/0.00B      | Read :      95.96M/s -   24564/s
  Load:     6.61  6.68  6.52              | Write:      79.27K/s -      19/s
                                                                                                         RUNNING QUERIES
PID     DATABASE                 USER              CLIENT   CPU% MEM%    READ/s   WRITE/s      TIME+  W   IOW  Query
102870 cpp                     postgres              None     7.4  0.3   94.62M     0.00B  05:40.21  N    N   autovacuum: ANALYZE vmds_rp
98337  cpp                     cpp_live      172.29.20.209     3.5  0.0    1.33M    79.27K  0.666824  N    N   select target_branch_uuid a
 _r2.funnels where target_branch_uuid = $1 AND target_version > $2 AND target_version <= $3
68776  cpp                     cpp_live      172.29.22.162     0.0  0.0    0.00B     0.00B  0.362072  N    N   SELECT id , object_id, obje
 ((M.branch_id=$2 AND M.object_version > $3 AND M.object_version <= $4) OR (M.branch_id=$5 AND M.object_version <= $6)) ORDER BY
79208  cpp                     cpp_proc       172.29.22.55     0.0  0.0    0.00B     0.00B  0.007092  N    N   INSERT INTO metadata_rprod_
 ($1, $2, metadata_rprod_cpp_r2.metadata_branch_version_increment($3), $4, $5)
101922 cpp                     cpp_proc      172.29.23.178     0.0  0.0    0.00B     0.00B  0.000000  N    N   SELECT * FROM (SELECT M.id,
 ype ORDER BY object_version DESC) FROM metadata_rprod_cpp_r2.metadata_objects M WHERE ((M.branch_id=$1 AND M.object_version <=
88107  cpp                     cpp_proc      172.29.21.229     0.0  0.0    0.00B     0.00B  0.000000  N    N   INSERT INTO vmds_rprod_cpp_
79117  cpp                     cpp_proc      172.29.23.178     0.0  0.0    0.00B     0.00B  0.000000  N    N   INSERT INTO metadata_rprod_
 ($1, $2, metadata_rprod_cpp_r2.metadata_branch_version_increment($3), $4, $5)
77746  cpp                     cpp_proc       172.29.23.64     0.0  0.0    0.00B     0.00B  0.000000  N    N   UPDATE journal_rprod_cpp_r2
80535  cpp                     cpp_proc       172.29.21.44     0.0  0.0    0.00B     0.00B  0.000000  N    N   INSERT INTO vmds_rprod_cpp_
 NOTHING
```

https://github.com/julmon/pg_activity

TOMTOM

Monitoring: query statistics

Why?

- What queries are executed:
  - Types of queries
  - Top queries (number, total time)
  - Parameters
- Find slow queries requiring optimization
- Check resource usage by particular queries
- Find queries causing timeouts

How?

- pg_stat_statements
- postgres logs
- munin

TOMTOM

Monitoring: errors

Why?

- Data corruption
- Database was shutdown
- Database not being able to start up
- Data not accessible:

  wrong user priviliges, full disk

```
07:00:49 UTC [82046]: [3222-1] db=,user= LOG:  could not fork new process for connection: Cannot allocate memory
07:00:49 UTC [82046]: [3223-1] db=,user= LOG:  could not fork new process for connection: Cannot allocate memory
07:24:25 UTC [106627]: [1-1] db=cpp,user=stat_collector FATAL:  out of memory
07:36:10 UTC [112362]: [1-1] db=cpp,user=stat_collector FATAL:  out of memory
07:36:10 UTC [82046]: [3224-1] db=,user= LOG:  could not fork new process for connection: Cannot allocate memory
07:36:10 UTC [82046]: [3225-1] db=,user= LOG:  could not fork new process for connection: Cannot allocate memory
07:36:10 UTC [82046]: [3226-1] db=,user= LOG:  could not fork new process for connection: Cannot allocate memory
07:48:00 UTC [117985]: [1-1] db=cpp,user=stat_collector FATAL:  out of memory
```

```
23:59:59 UTC [74395]: [1-1] db=,user= LOG:  started streaming WAL from primary at 46A0/75000000 on timeline 5
23:59:59 UTC [74395]: [2-1] db=,user= FATAL:  could not write to file "pg_xlog/xlogtemp.74395": No space left on device
```

How?

- zgrep –i fatal /var/log/db/postgresql-* | less

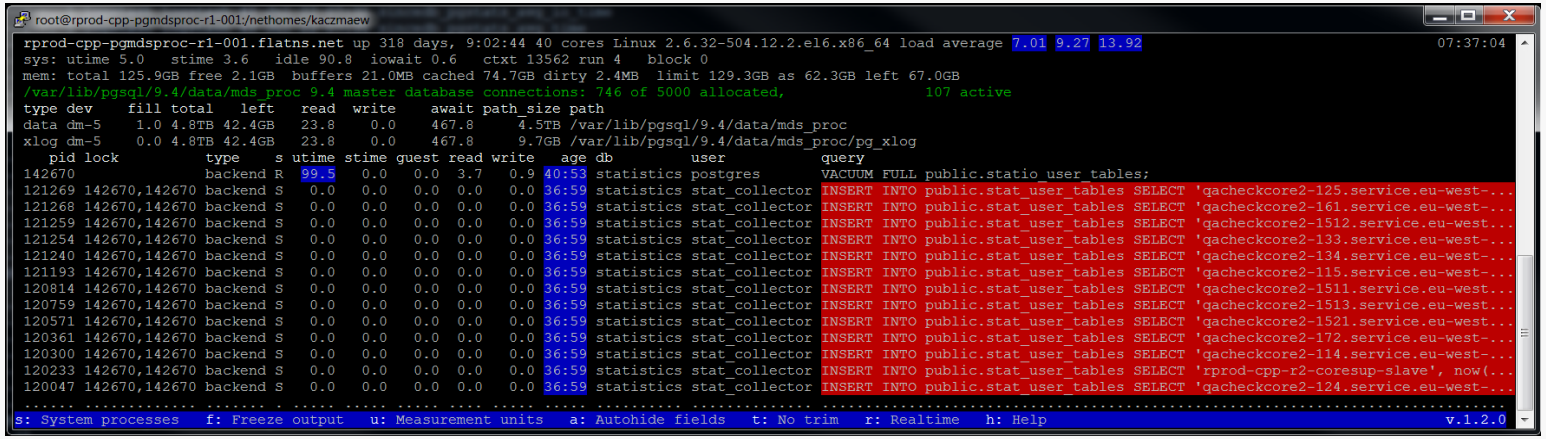
Monitoring: locks

Why?

- Verify if some offline processes do not block applications (operations freezing big chunks of data like whole tables)
- Verify if some application processes do not block other applications processes
- Deadlocks

How?

- SELECT * FROM pg_locks WHERE granted = false;
- munin
- pg_view, pg_activity
- postgres logs (for deadlocks)



Monitoring: objects size

Why?

- Control diskspace
- Know the largest objects, control increase
- Changes in usage pattern of application layer (tuples count, average tuple size)

How?

- Munin to catch trend
- pg_total_relation_size(relid) – table + indexes size
- pg_relation_size(relid) – tables or index size
- pgstattuple(regclass) – for precise results
- pgstatindex(regclass) – for precise results
- SELECT reltuples AS approximate_row_count FROM pg_class WHERE relname = 'tbl';

TOMTOM

Monitoring: statistics of tables

Why?

- Changes in usage pattern of application layer
- Types of search (need for indexes)
- Number of inserted, updated, deleted tuples
- Analyze and vacuum info

How?

- pg_stat_user_tables

```
-[ RECORD 1 ]-------+------------------------------
relid               | 1234235602
schemaname          | locks_rprod_cpp_r2
relname             | lock
seq_scan            | 2712
seq_tup_read        | 187136
idx_scan            | 2910
idx_tup_fetch       | 113882
n_tup_ins           | 24163
n_tup_upd           | 0
n_tup_del           | 24064
n_tup_hot_upd       | 0
n_live_tup          | 54
n_dead_tup          | 29478
n_mod_since_analyze | 1506
last_vacuum         | 2017-10-24 09:05:01.223045+00
last_autovacuum     | 2017-10-24 09:04:39.577809+00
last_analyze        |
last_autoanalyze    | 2017-10-24 09:04:39.58479+00
vacuum_count        | 1
autovacuum_count    | 5
analyze_count       | 0
autoanalyze_count   | 5
```

Monitoring: statistics of indexes

Why?

- Changes in usage pattern of application layer
- Types of search (not used indexes may be dropped)

How?

- pg_stat_user_indexes

```
 relname |                 indexrelname                 | idx_scan | idx_tup_read | idx_tup_fetch
---------+----------------------------------------------+----------+--------------+---------------
 lock    | lock_new_expired_named_area_id_branch_id_idx1 |     1771 |       558822 |          6667
 lock    | lock_new_pkey1                               |        0 |            0 |             0
(2 rows)
```


TOMTOM

Monitoring: vacuum process

Why?

- Vacuum effectiveness
- To know how much resources are used by vacuum process

How?

- pg_view
- htop
- iotop
- postgres logs

Monitoring: bloat

Why?

- If you do a lot of updates or deletes and readings at the same time your tables and indexes

get bloated

- Having bloated tables or indexes causes: uneffective space usage, slower reads and writes

How?

- pg_stats (estimated) – implemented also in check_postgres scripts
- pgstattuple extension (exact, but slow query) - includes pgstatindex
- pgstattuple_approx (quite exact, quite fast)

TOMTOM

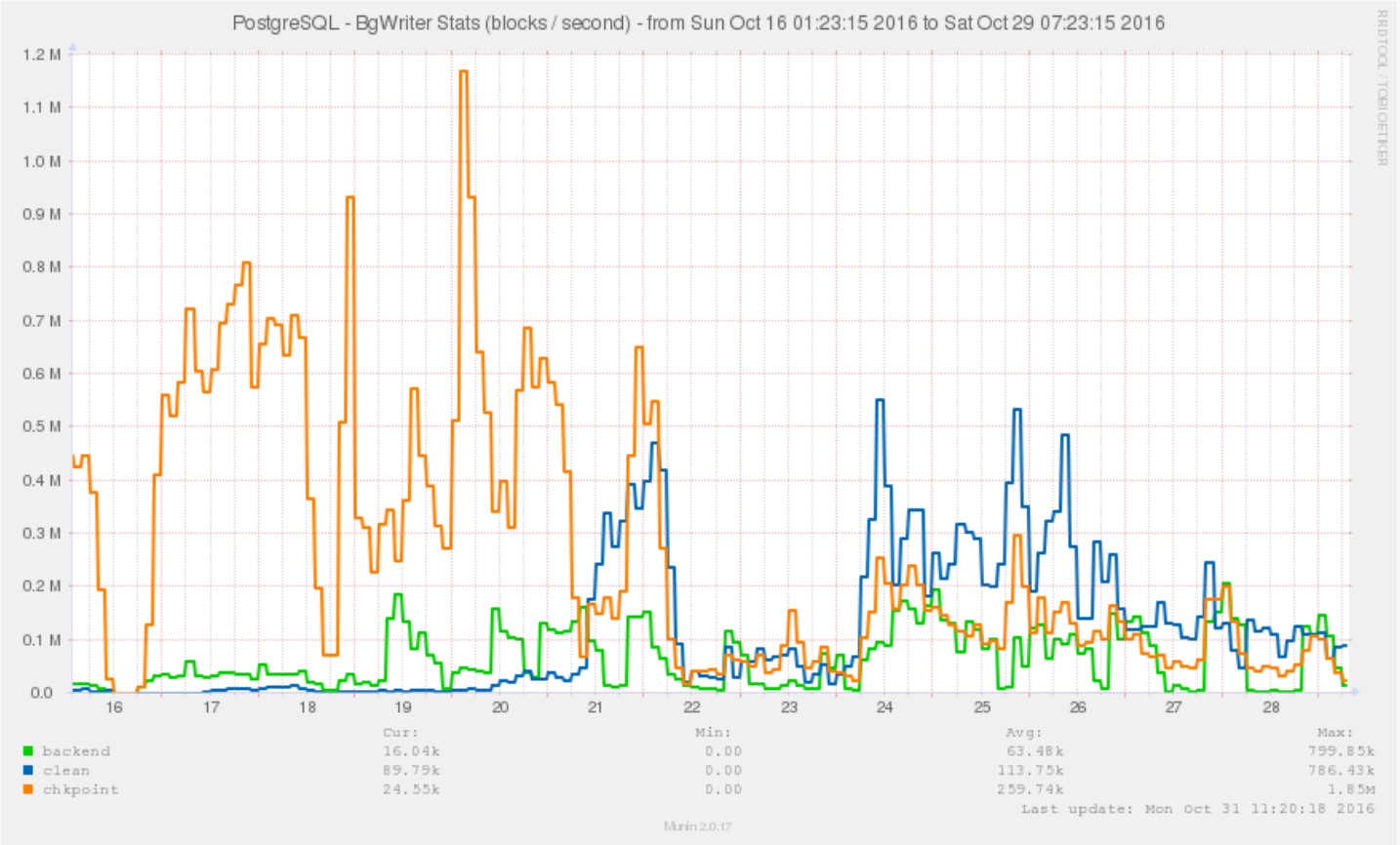
Monitoring: bg writer and checkpoints

Why?

- Influence on write performance

How?

- pg_stat_bgwriter



Monitoring: replication lag

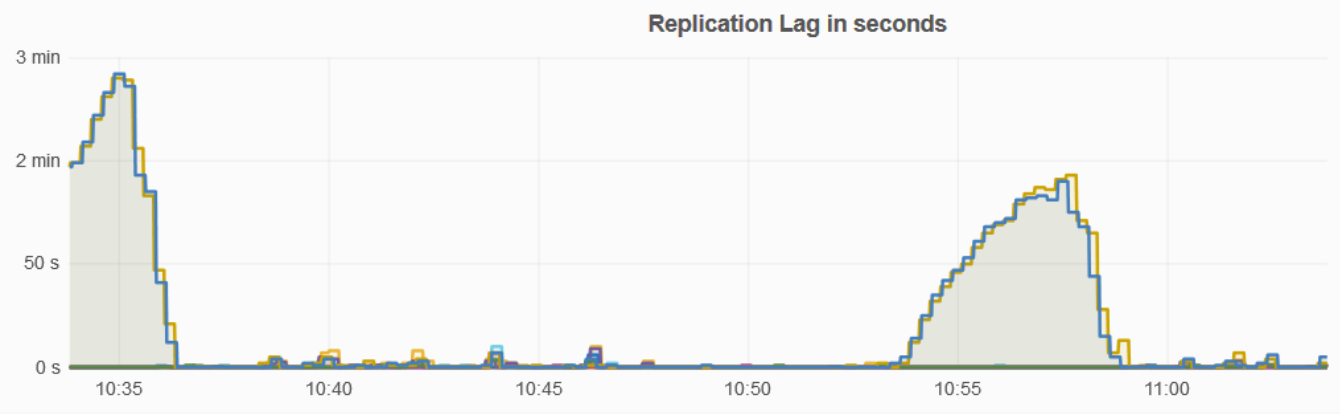
Why?

- Usability of standbys in terms of fresh data
- Are standbys in sync?
- Streaming replication depends on resources utilization (network, cpu, disk io)

How?

- Primary: pg_stat_replication
- Standby: SELECT

now() - pg_last_xact_replay_timestamp();



TOMTOM

Monitoring at system level: cpu

Why?

- To find processes consuming most cpu
- Having cpu utilization more than 60-70% usually leads to significant drop of performance due to cpu context switching
- To have knowledge what is usual consumption of cpu by specific processes (queries, autovacuum, replication)
- Find areas to optimize

How?

- top, htop
- munin
- pg_stat_statements

TOMTOM

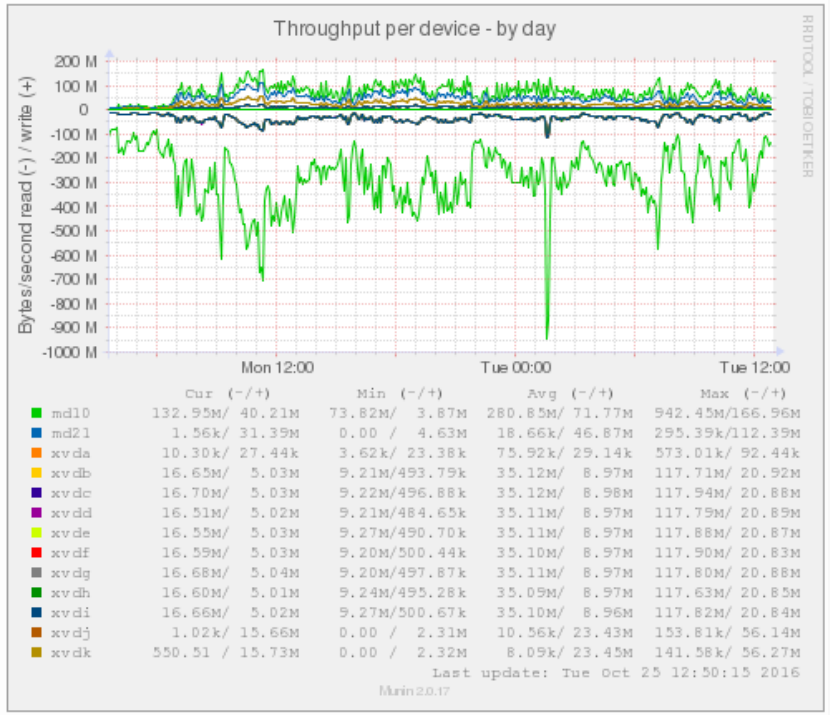
Monitoring at system level: disk io

Why?

- To find processes consuming disk io most
- Having disk reads more than 90% of hardware capabilities usually leads to significant drop of performance of both reads and writes
- To have knowledge what is usual consumption of disk io by specific processes (queries, autovacuum, replication, bg writer, checkpoints, maintenance)
- Find areas to optimize

How?

- iotop
- munin
- pg_stat_statement



Monitoring at system level: memory

Why?

- To find processes consuming most RAM
- Having memory utilization less than 60-70% usually is a waste (possibly cache hit ratio is low)
- To have knowledge what is usual consumption of ram by specific processes (queries mostly)
- Find areas to optimize

How?

- htop
- munin
- Unfortunately there are no statistics related to ram consumption inside PostgreSQL

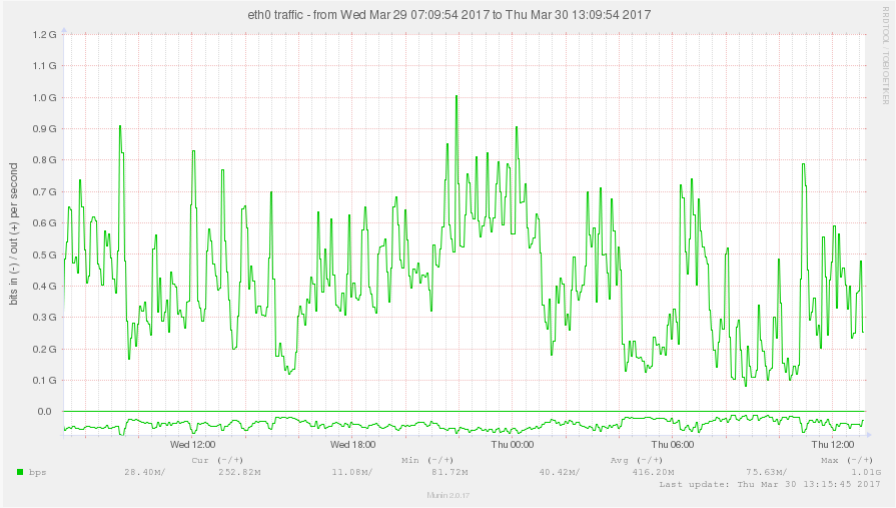
Monitoring at system level: network

Why?

- To find processes consuming network
- Having network utilization more than 90% for a longer period usually means the investigation must be done (which may result in optimizations or infrastructure enhancement)
- To have knowledge what is usual consumption of network by specific processes (queries, replication, backup)
- Find areas to optimize

How?

- netstat
- munin



What changes when hundreds of databases are to be monitored?

Metrics collectors

- Prometheus + exporter plugins
- Munin + plugins
- AppDynamics & Java agents
- custom collectors (queries statistics)

Metrics aggregators

- Elastic Search
- AppDynamics

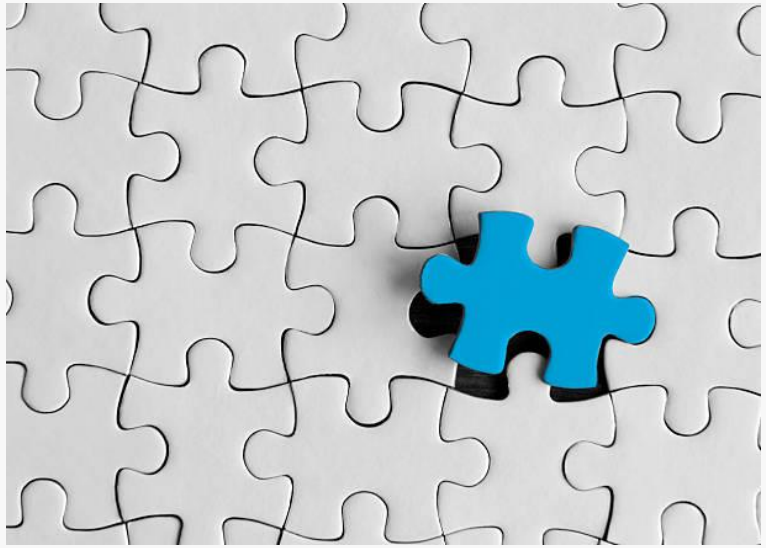
Visualization

- Kibana, Grafana
- AppDynamics
- Munin

TOMTOM

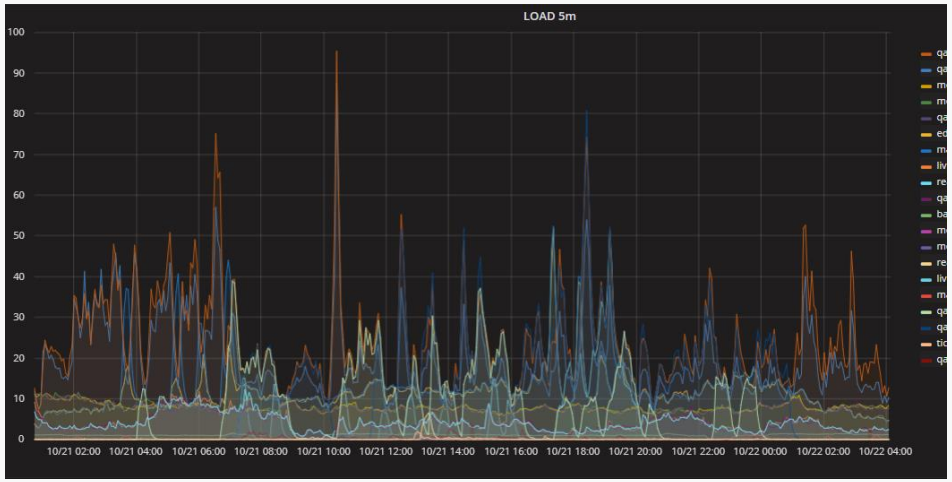
What changes when hundreds of databases are to be monitored?

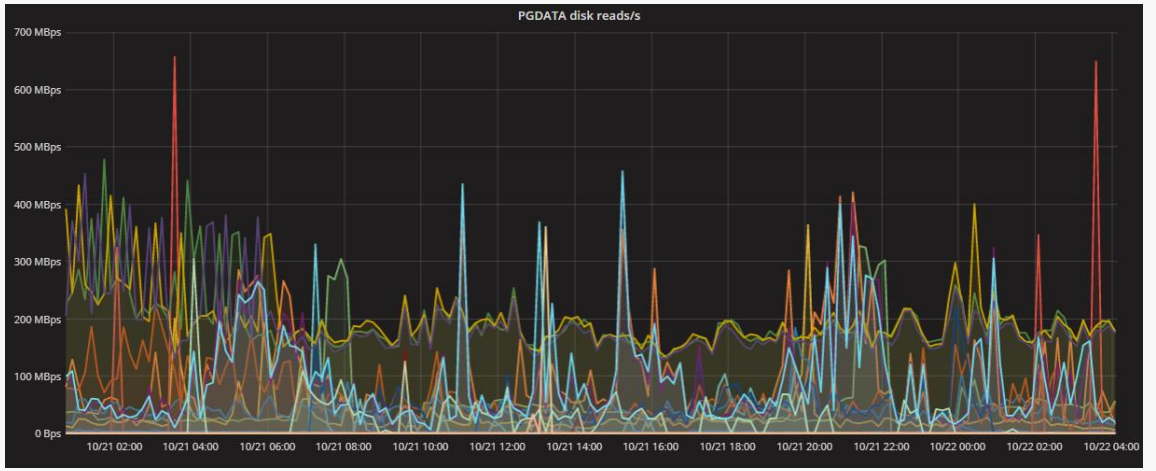
- Don’t rely on manual setups!
- Git – configuration is versioned and kept in external storage
- Ansible – automated configuration management
  - Defines which collectors / agents / plugins need to be installed
  - Settings for database and system
- Jenkins – automate your job



What changes when hundreds of databases are to be monitored?

- Prometheus + Grafana
  - Prometheus for storing huge amount of metrics
  - Existing exporters for system and database metrics
  - Allow collecting custom metrics as timeseries data
  - Multiple databases on single chart - aggregation





What changes when hundreds of databases are to be monitored?

- Custom collectors + Elastic Search + Kibana
  - Elastic Search for collecting metrics
  - Kibana for Visualization

- Munin
  - Plugins: built-in and external
  - Does not aggregate metrics into single chart

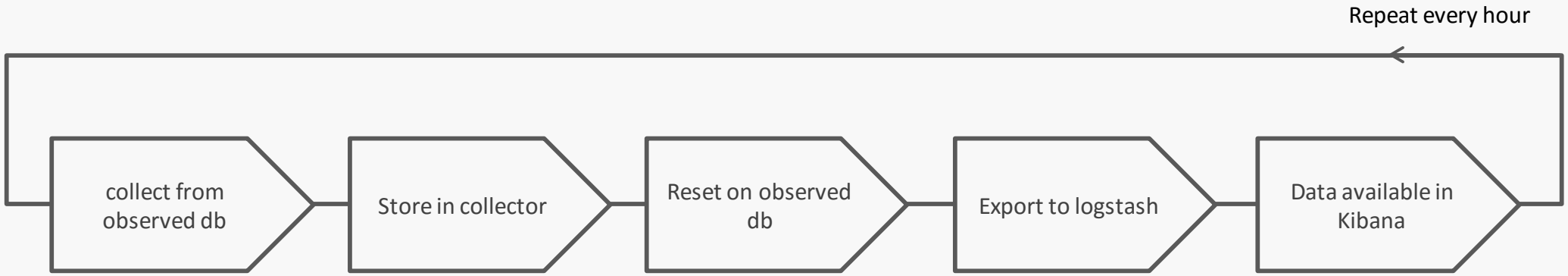
- AppDynamics
  - Collecting metrics on application level from many instances
  - Dynamic instrumentation
  - Alerts on incidents
  - Track down the root cause

https://prometheus.io/
https://grafana.com/
https://www.elastic.co/products/kibana
http://munin-monitoring.org/

TOMTOM

Monitoring: Multiple database instances vs pg_stat_statements

- Gathers a bunch of useful statistics of query execution
- The best way to track lots of short queries
- One cumulative sack
- Not usable if you need track query behavior changes



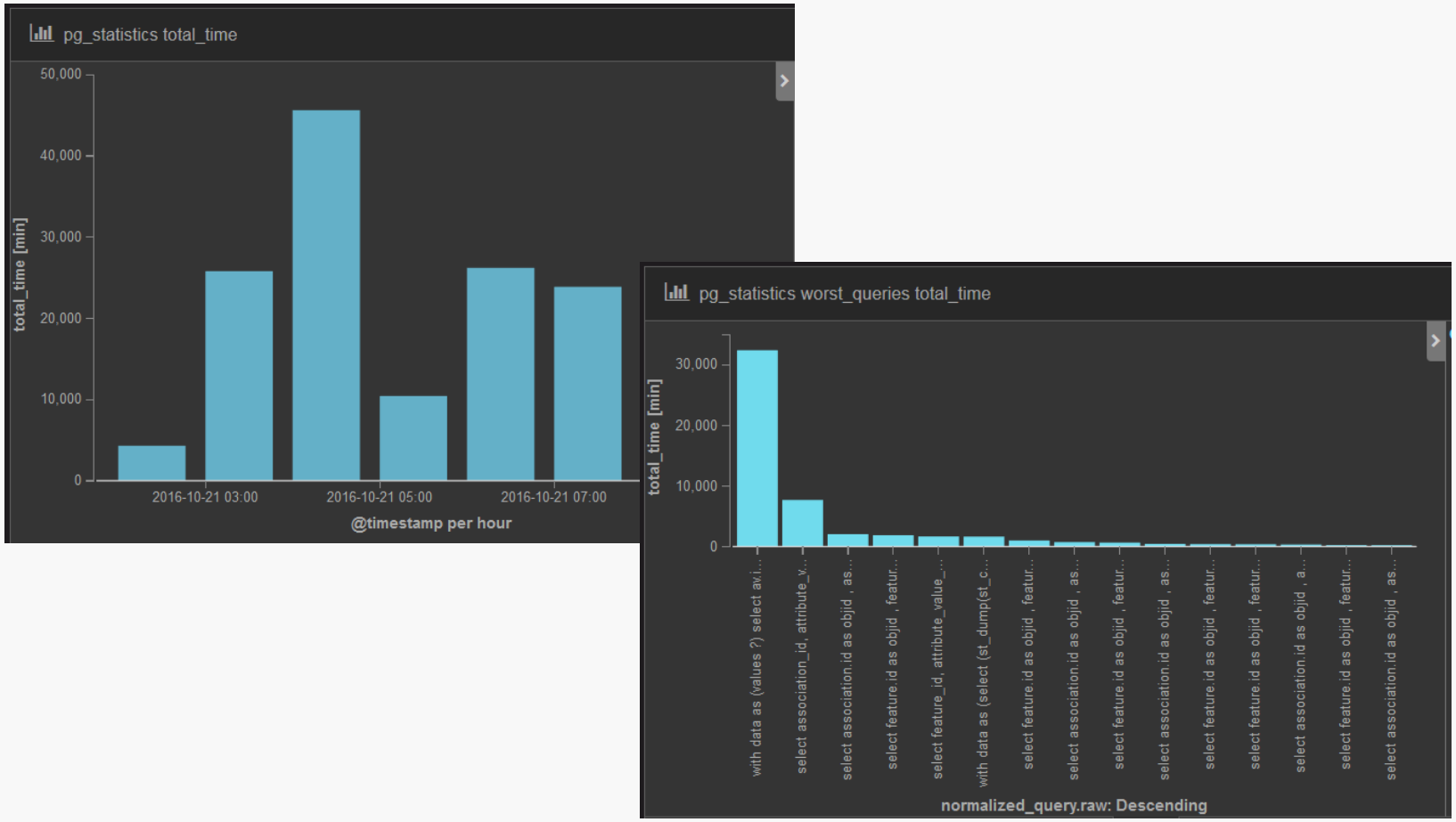
ElastAlert https://github.com/Yelp/elastalert

TOMTOM

Monitoring: stat_statements in Kibana

In Kibana, we can easily observe for each particular statement on each and every machine separately (if we want to):

- total execution time
- cpu execution time
- io execution time
- number of calls
- number of rows returned / affected
- average execution time
- average cpu execution time
- average io execution time
- average number of calls
- average number of rows returned / affected



Conclusion

- PostgreSQL is great database capable of reaching big goals
- It is scalable and provides good monitoring tools

But it is not enough

- Needs constant monitoring (metrics collection)
- Knowledge sharing: software developers should know how to read basic metrics
- For many instances:
    - Aggregated overview on metrics
    - Alerting on top of business metrics – not on low level instance metrics

Questions?

We are hiring! https://tomtom.com/careers/