



PostgresConf 2018

POSTGRESQL MONITORING INSIDE **MAP-MAKING PLATFORM**

Michał Gutkowski, Rafał Hawrylak

Agenda

1. About us
2. TomTom - what do we do?
3. Why monitoring is important?
4. Who should monitor?
5. What should we monitor?
6. Metrics & Tools
7. What changes when hundreds of databases have to be monitored?
8. Conclusion

About Us



Rafał Hawrylak

rafal.hawrylak@tomtom.com

Software developer and
database expert



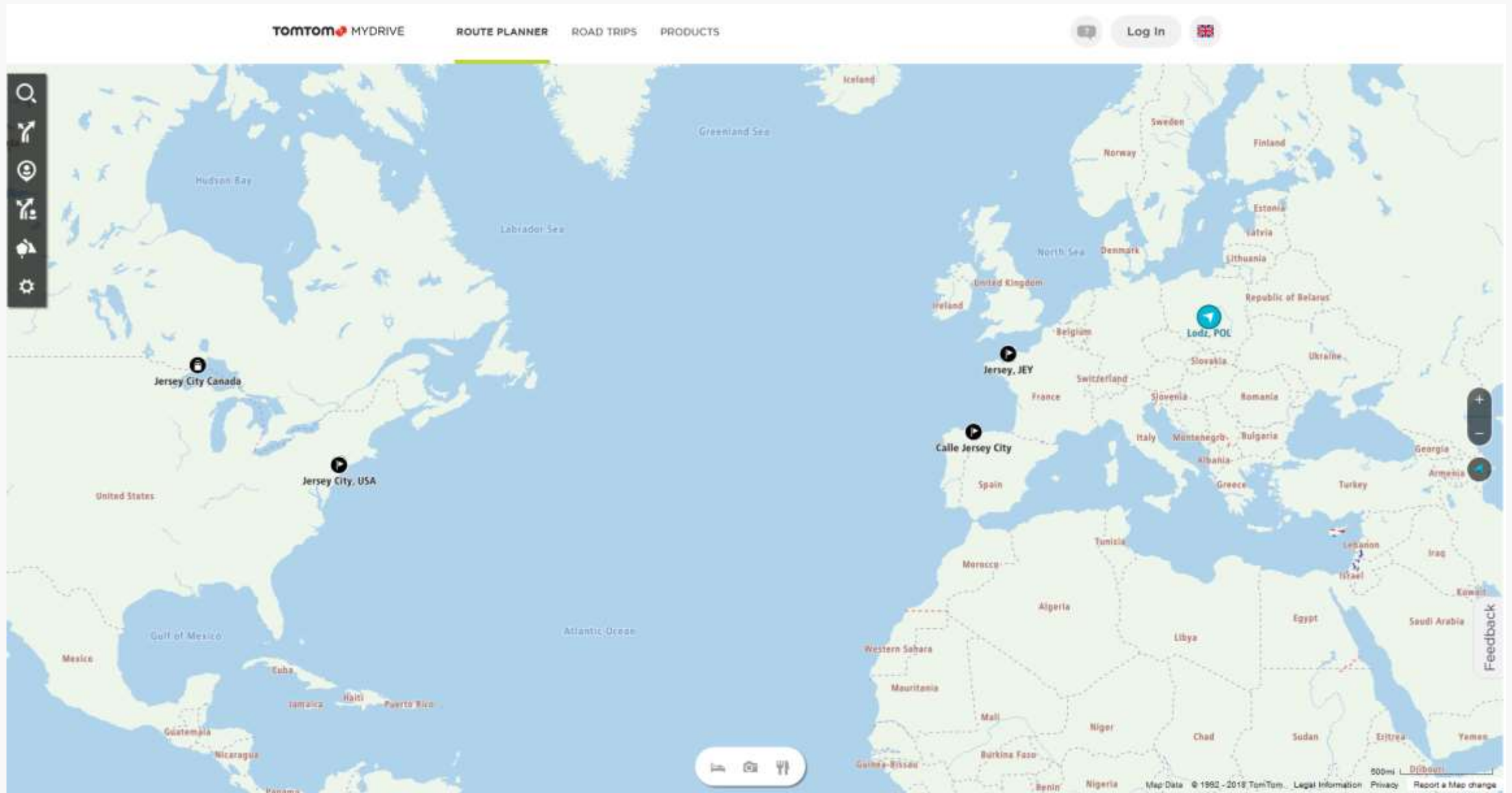
Michał Gutkowski

michal.gutkowski@tomtom.com

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



About Us - We Are From ŁÓDŹ, Poland!



TomTom - location services



Mobile Usage



Navigation



Ride Planning



Drive Range Calculation

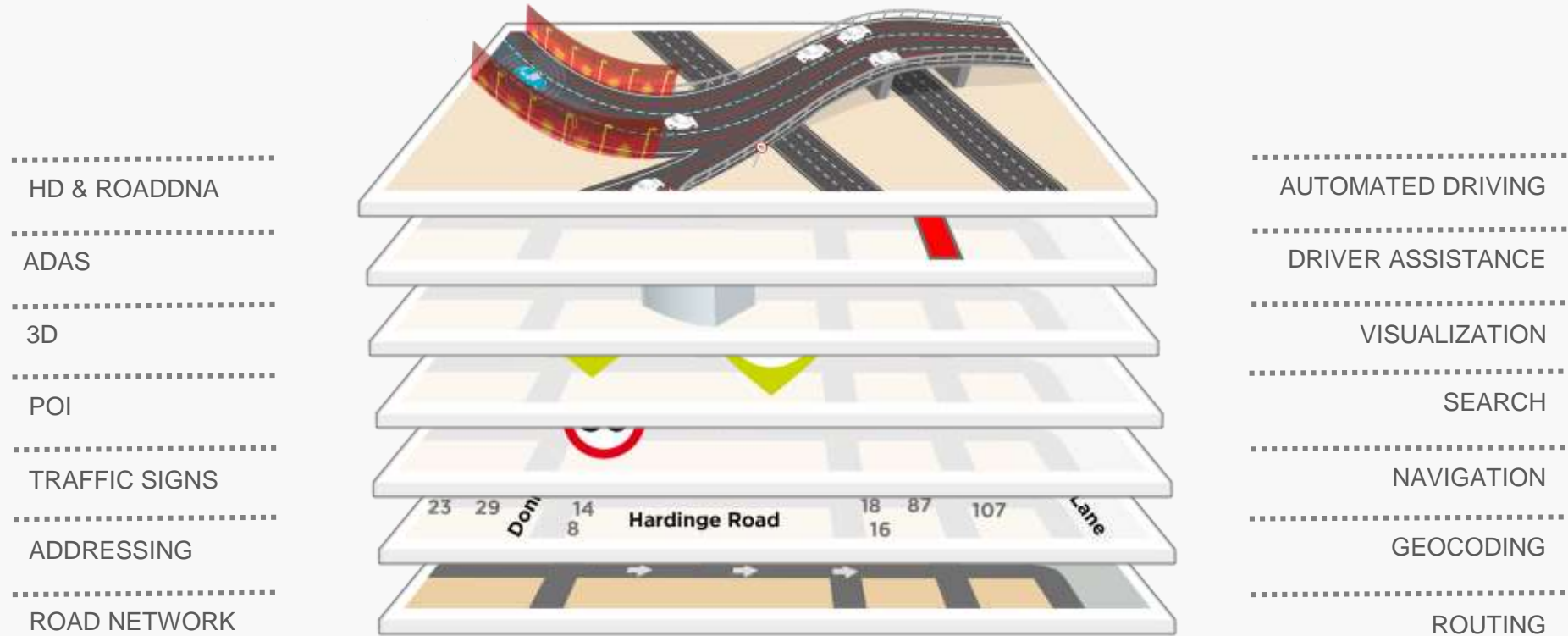


Internet of Things



Autonomous Driving

TomTom - location content



TomTom - map-making platform

- Transactional and versioned changes
- Database with spatial functions
- Massive automated tools editing map
- Manual editors working in parallel
- Real-time quality checks
- Billions of map objects



Map-making platform in 2018

- PostgreSQL + Postgis
- 100+ database machines in AWS
- Sharding and scalable reads
- 150TB of data
- Daily db size increase: 400GB - up to 15k rows / sec
- Daily db transfers: 200TB - up to 500k queries / sec

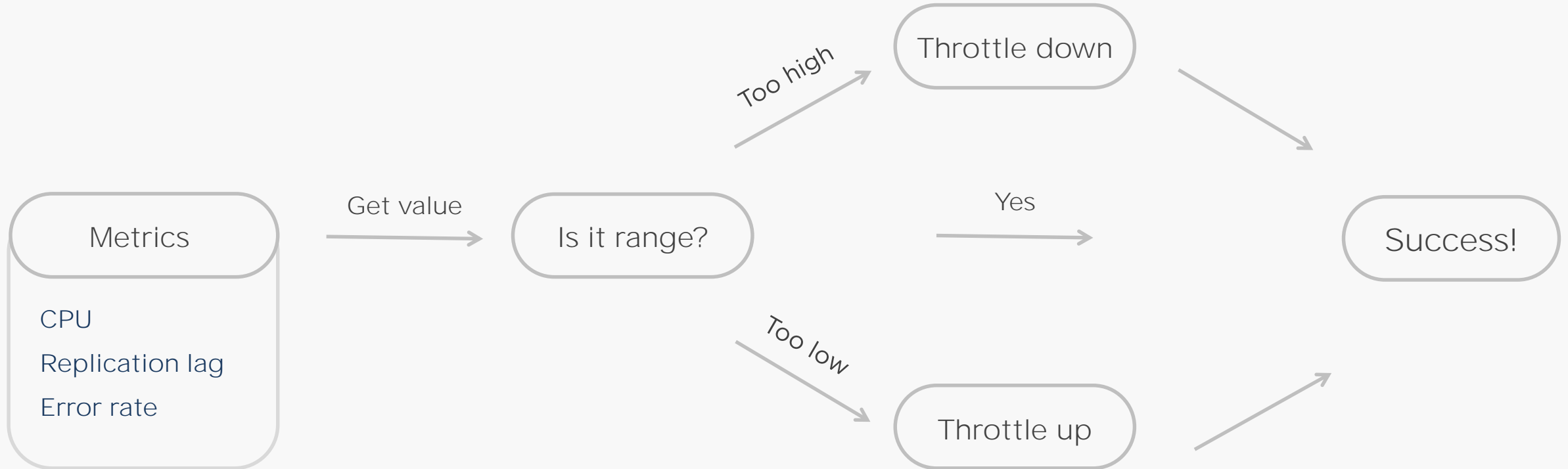


Why monitoring is important?

- System health-check and maintenance
- Alerting and reliable notification system
- Detect performance regression
- Measure optimizations - software and business process
- Best value for money - maximum utilization
- Adjust business processes - self healing system



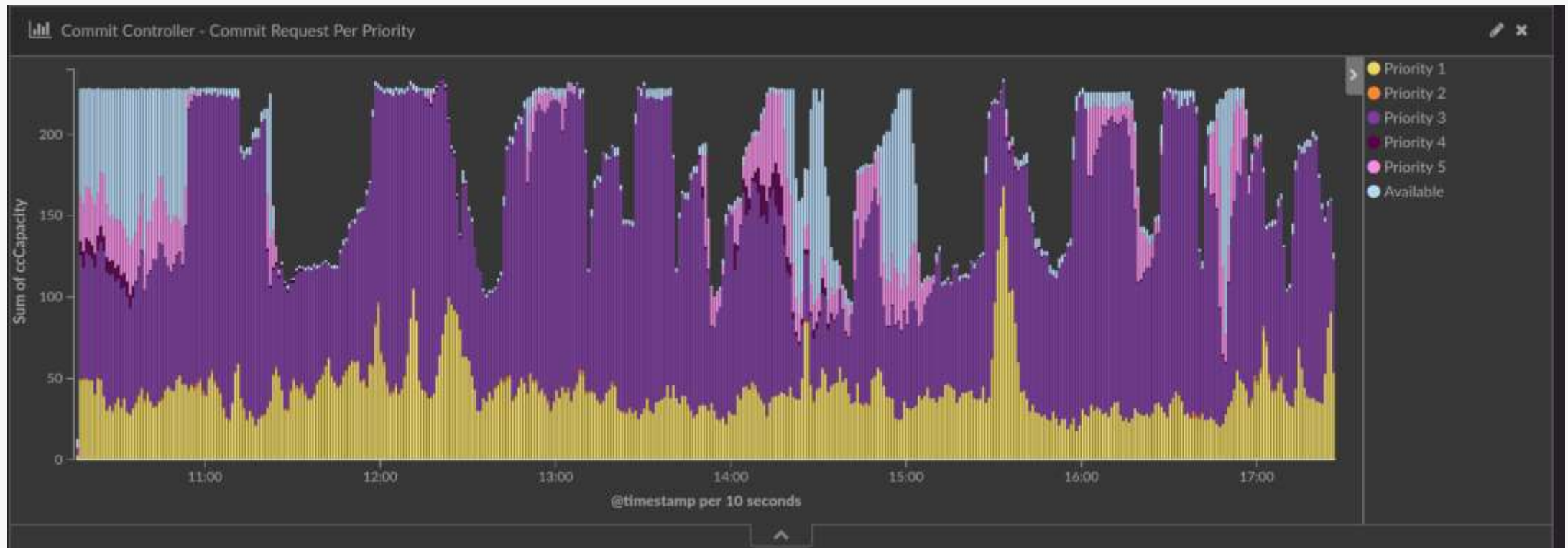
Why monitoring is important?



Why monitoring is important?



Why monitoring is important?



Who should monitor?

- Production monitoring
- Database team
- Every developer and tester
- Top-down responsibility: teams are responsible for delivering changes in software, databases and monitoring



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 and warnings

Queries

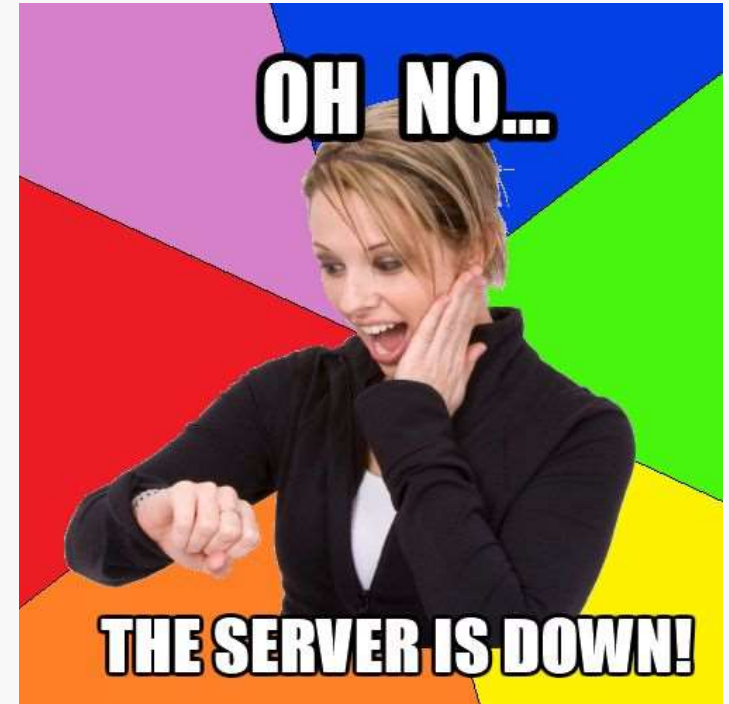
Storage

Replication

Operational system

Alerting

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



Metrics & Tools

Monitoring: which metrics are important

Queries

- Connections
- Active queries
- Query statistics
- Locks

Storage

- Statistics of tables and indexes
- Objects size
- Vacuum processes
- Bloat
- bg writer and checkpoints

Replication

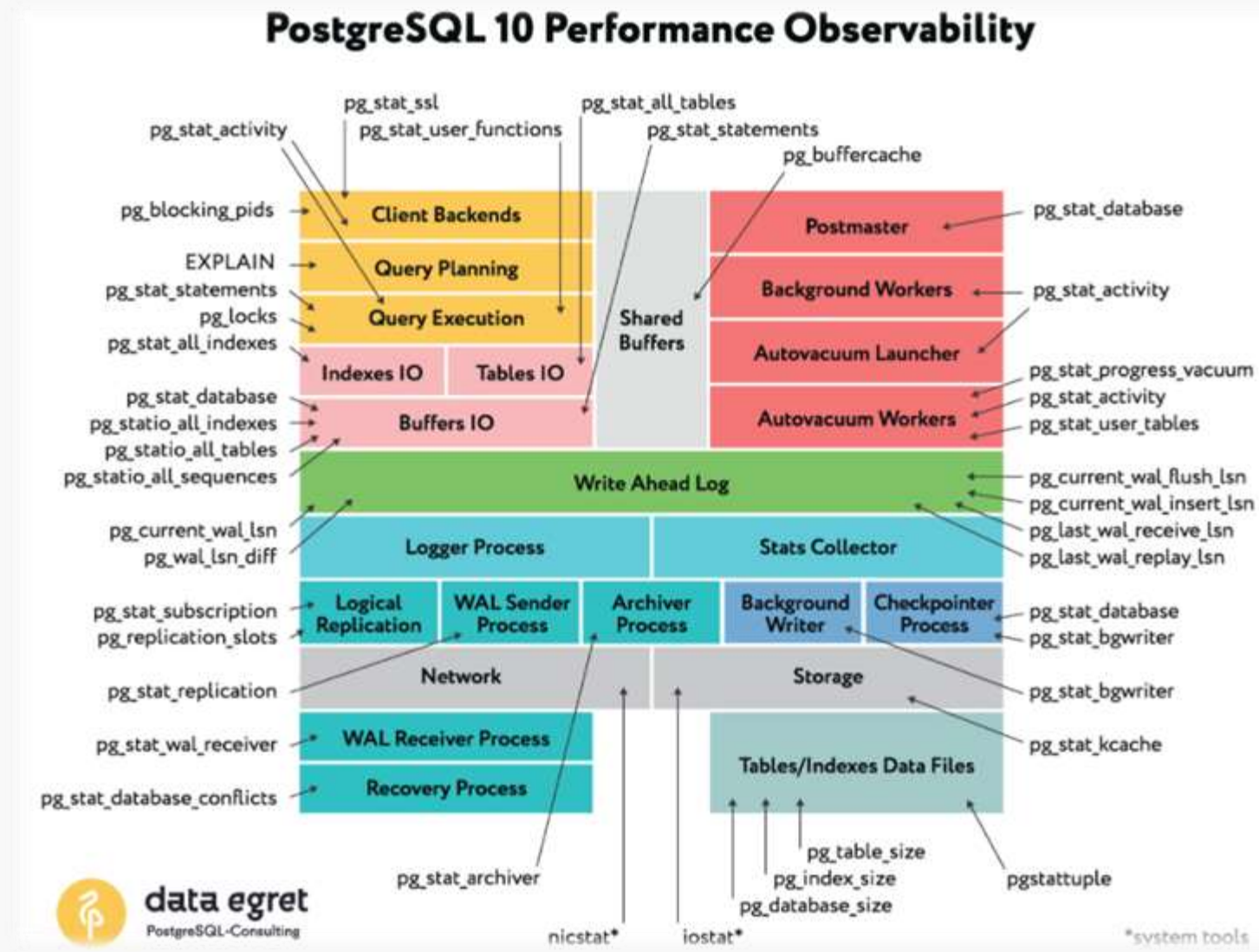
- Replication tree
- Lag on standby

OS stats

- CPU
- Disk IO
- Memory
- Network

ERRORS

Monitoring: postgresql internals



Monitoring: errors

Why?

- Data corruption
- Database was shutdown
- Database not being able to start up
- Data not accessible
- Wrong user privileges
- Full disk

How?

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

```
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
```

Monitoring: connections and queries

Why?

- indicate problems in higher tiers
- changes in usage pattern
- queries requiring optimization
- resource usage
- timeouts

How?

- `SELECT username, count(1) FROM pg_stat_activity`
- `pg_view` or `pg_activity` or `pgcenter`
- `munin`
- `pg_stat_statements`
- postgres logs

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 ('e2737fd
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

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_rpr
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.funnel 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

Monitoring: locks

```
root@rprod-cpp-pgmdsproc-r1-001:/nethomes/kaczmaew
rprod-cpp-pgmdsproc-r1-001.flatns.net up 318 days, 9:02:44 40 cores Linux 2.6.32-504.12.2.el6.x86_64 load average 7.01 9.27 13.92 07:37:04
sys: utime 5.0 stime 3.6 idle 90.8 iowait 0.6 ctxt 13562 run 4 block 0
mem: total 125.9GB free 2.1GB buffers 21.0MB cached 74.7GB dirty 2.4MB limit 129.3GB as 62.3GB left 67.0GB
/var/lib/pgsql/9.4/data/mds_proc 9.4 master database connections: 746 of 5000 allocated, 107 active
type dev fill total left read write await path_size path
data dm-5 1.0 4.8TB 42.4GB 23.8 0.0 467.8 4.5TB /var/lib/pgsql/9.4/data/mds_proc
xlog dm-5 0.0 4.8TB 42.4GB 23.8 0.0 467.8 9.7GB /var/lib/pgsql/9.4/data/mds_proc/pg_xlog
 pid lock type s utime stime guest read write age db user query
142670 backend R 99.5 0.0 0.0 3.7 0.9 40:53 statistics postgres VACUUM FULL public.statio user_tables;
121269 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'qacheckcore2-125.service.eu-west-...
121268 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'qacheckcore2-161.service.eu-west-...
121259 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'qacheckcore2-1512.service.eu-west-...
121254 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'qacheckcore2-133.service.eu-west-...
121240 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'qacheckcore2-134.service.eu-west-...
121193 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'qacheckcore2-115.service.eu-west-...
120814 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'qacheckcore2-1511.service.eu-west-...
120759 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'qacheckcore2-1513.service.eu-west-...
120571 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'qacheckcore2-1521.service.eu-west-...
120361 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'qacheckcore2-172.service.eu-west-...
120300 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'qacheckcore2-114.service.eu-west-...
120233 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'rprod-cpp-r2-coresup-slave', now(...
120047 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'qacheckcore2-124.service.eu-west-...
.....
s: System processes f: Freeze output u: Measurement units a: Autohide fields t: No trim r: Realtime h: Help v.1.2.0
```

Monitoring: objects size

Why?

- Control disk space
- 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(releid)` - table + indexes size
- `pg_relation_size(releid)` - 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';`

Monitoring: statistics of tables and indexes

Why?

- Changes in usage pattern of application layer
- Types of search (need for indexes)
- Number of inserted, updated, deleted tuples
- Analyze and vacuum info
- Types of search (not used indexes may be dropped)

How?

- `pg_stat_user_tables`
- `pg_stat_user_indexes`

```
-[ 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: vacuum process

Why?

- vacuum effectiveness
- resources utilization

How?

- pg_stat_progress_vacuum (PG10)
- pg_view
- htop
- iotop
- postgres logs (log_autovacuum_min_duration)

Monitoring: bloat

Why?

- 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)

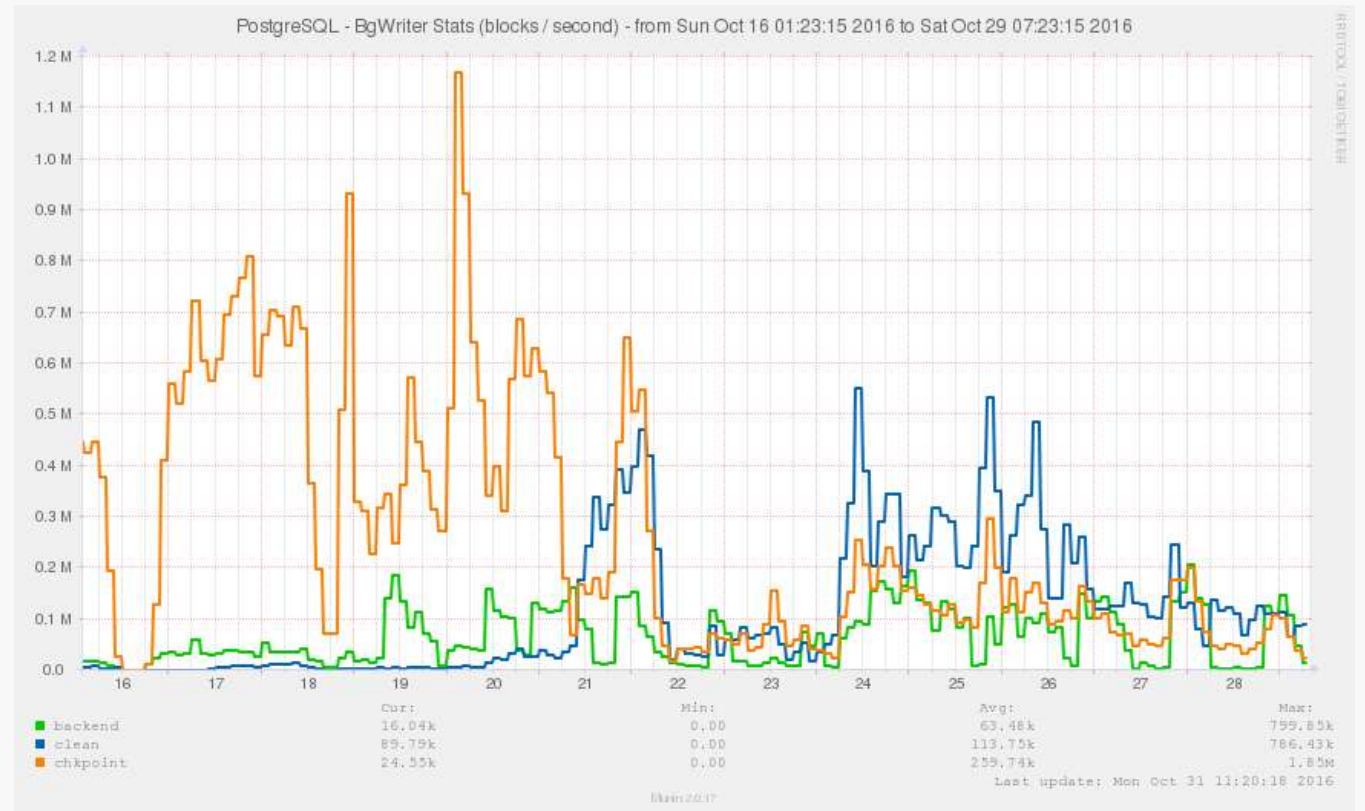
Monitoring: bg writer and checkpoints

Why?

- Influence on write performance

How?

- `pg_stat_bgwriter`



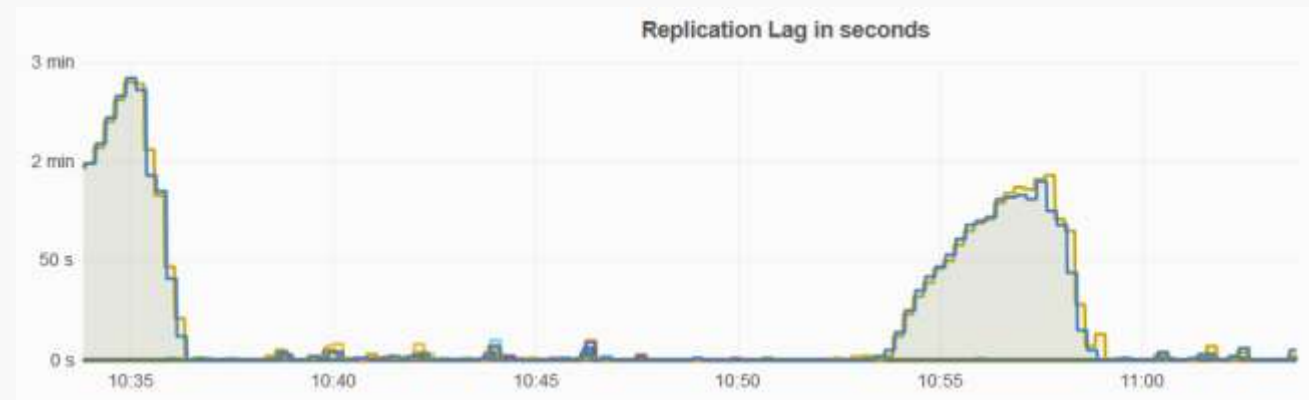
Monitoring: replication lag

Why?

- Usability of standbys in terms of fresh data
- Are standbys in sync?
- Resources used by replication (network, cpu, disk utilization)

How?

- Primary: `pg_stat_replication`
- Standby: `SELECT`
`now() - pg_last_xact_replay_timestamp();`



Monitoring at system level: cpu / disk io / memory / network

Why?

- most consuming processes
- learn your 100%
- usual consumption
- find bottlenecks and areas to optimize

How?

- cpu: top, htop, munin, pg_stat_statements
- disk io: iotop, munin, pg_stat_statements
- memory: htop, munin
- network: netstat, munin

What changes when hundreds of databases are to be monitored?

- Versioned PostgreSQL configuration - Git
- Automated configuration management - Ansible/Puppet
 - OS settings
 - Defines which collectors / agents / plugins need to be installed
 - Includes into monitoring (dashboards)
- Automated deployment and replication - Jenkins
- Automated backups



What changes when hundreds of databases are to be monitored?

Amounts of logs collected

- 12 billions searchable events
(2% of total events, 200-400M events daily)
- 30 days retention for 95% of the events



What changes when hundreds of databases are to be monitored?

Metrics collectors

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

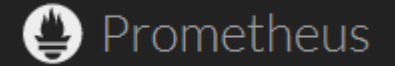
Metrics aggregators

- Elastic Search
- AppDynamics

Visualization

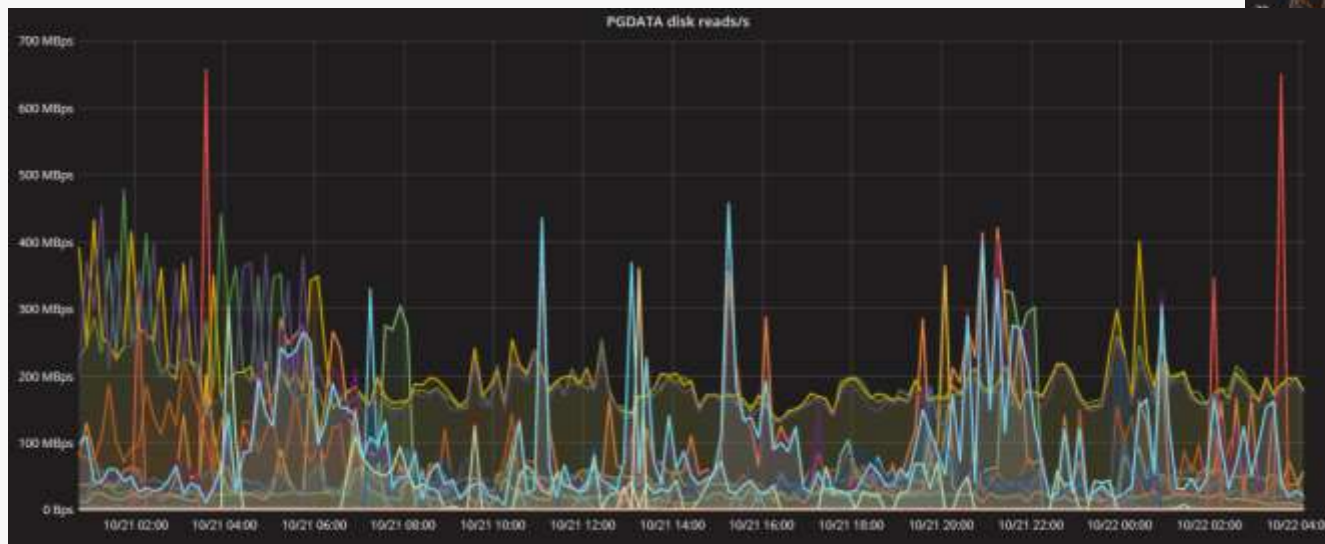
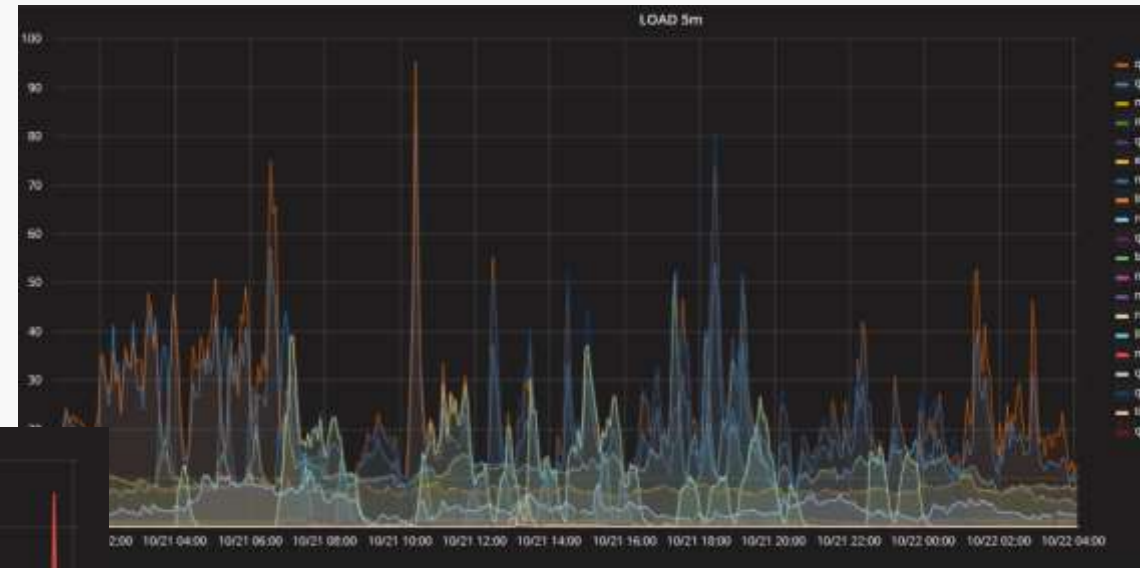
- Kibana, Graphana
- AppDynamics
- Munin

What changes when hundreds of databases are to be monitored?

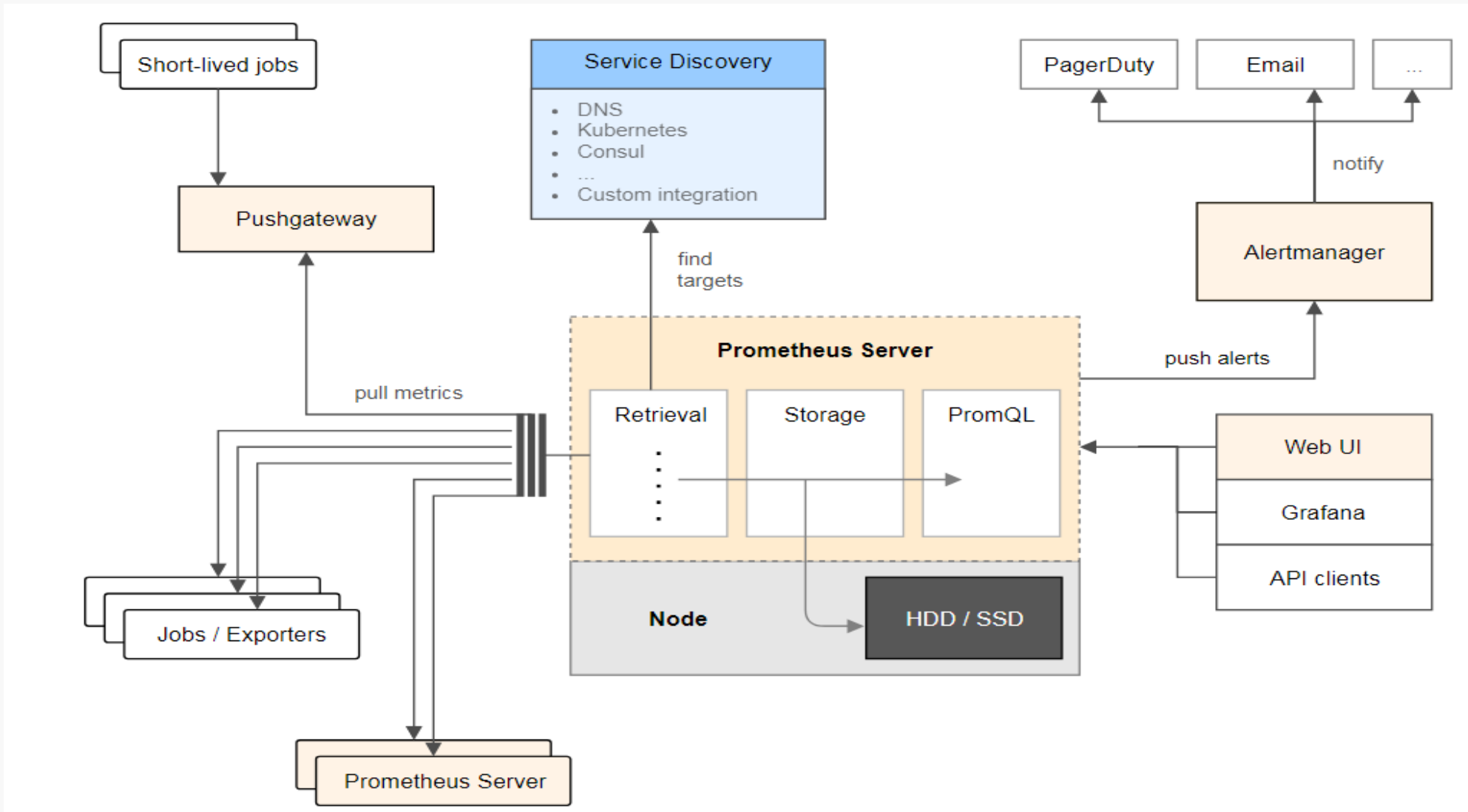


Prometheus + Grafana

- Prometheus for pulling and storing metrics
- Ready-to-use exporters for OS and database metrics
- Allows creating custom exporters
- Grafana for visualization of different datasources



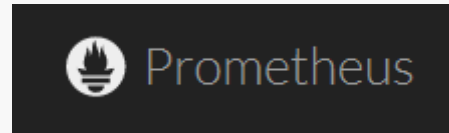
What changes when hundreds of databases are to be monitored?



What changes when hundreds of databases are to be monitored?

- **Prometheus + Grafana**

- Prometheus for collecting metrics
- Grafana for visualization and alerting



- **Custom collectors + Elastic Search + Kibana**

- ElasticSearch for collecting metrics
- Kibana for Visualization



- **Munin**

- Built-in and plugin collectors
- Does not automatically aggregate metrics into single chart

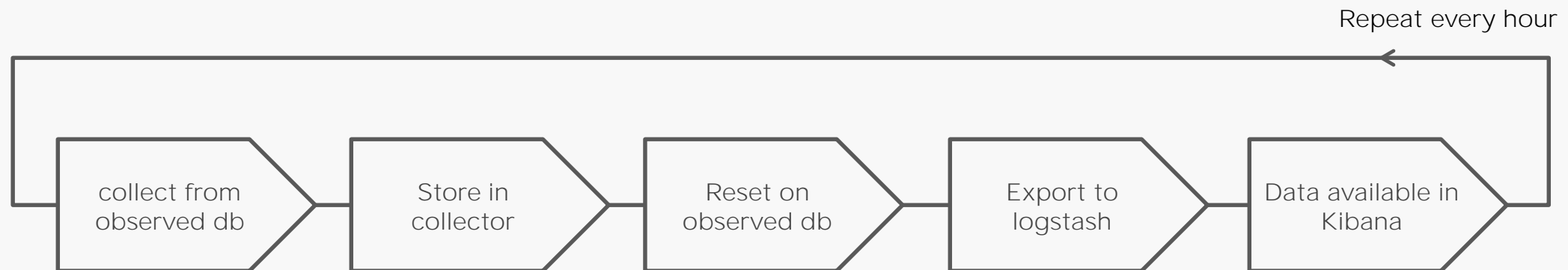


- **Appdynamics**

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

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



Monitoring: pg_stat_statements in Kibana

Dashboard:

- total / avg execution time
- total / avg CPU execution time
- total / avg IO execution time
- total / avg number of calls
- total / avg 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)
- For many instances:
 - It needs aggregated overview on metrics
 - Alerting on top of business metrics - not on low level instance metrics

During this presentation

- db size increased by 12,5GB, 40,5M rows
- generated WALs: 6k of total size 94GB
- all those WALs were streamed to 6 standbys
- db transfered (in/out): 6TB
- queries run: 1,08 bilion





TomTom data via API:

<https://developer.tomtom.com>

Questions?



<https://developer.tomtom.com/>

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