CYBERTEC

DATA SCIENCE & POSTGRESQL

Getting started with pgwatch

Prague PostgreSQL Meetup:
February Edition

Senior Consultant/Developer

Pavlo Golub

MAIL

pavlo.golub@cybertec.at

Social

pashagolub.github.io

WEB

www.cybertec-postgresql.com

CYBERTEC
DATA SCIENCE & POSTGRESQL
www.cybertec-postgresql.com

DATABASE - PRODUCTS

scalefield

CYPEX

CYBERTEC MIGRATOR

DATA MASKING

CYBERTEC PGEE

POSTGRESQL TDE

AGENDA

- Different levels of database monitoring
- PostgreSQL monitoring approaches
- Features of pgwatch
- Use cases

Getting started with pgwatch

Different levels of database monitoring

CYBERTEC
DATA SCIENCE & POSTGRESQL
www.cybertec-postgresql.com

Why to monitor?

- Failure / Downtime detection
- Slowness / Performance analysis
- Proactive predictions
- Maybe wasting money?

Different levels of database monitoring

- High level service availability
- System monitoring
- PostgreSQL land

High level service availability

- Try to periodically connect/query from an outside system

- DIY - e.g. a simple Cron script
- SaaS - lots of service providers

Who will guard the guards?

- You'll probably want two services for more critical stuff...

System monitoring

Too many tools, no real standards. Just make sure to understand what you're measuring!

- Do you know what does the CPU load number actually mean?
  - Is it a good metric?
- What's the difference between VIRT, RES, SHR memory values for a process?

PostgreSQL land

- Log analysis
- Statistics Collector
- Extensions

Log analysis

- “Just in case” storing of logs for possible ad hoc needs
  - Moving logs to a central place makes sense
  - rsync + Cron
- Active parsing
  - grep + Cron
  - DIY (file_fdw, Graylog, ELK, …)
  - pgBadger (JSON format)
  - Grafana Loki
  - Some cloud service (Loggly, Splunk, …)

Logging configuration

- Some settings to note
  - log_destination (CSV format recommended)
  - log_statement = ‘none’ (default)
  - log_min_duration_statement / log_duration
  - log_min_messages / log_min_error_statement

```
postgres=# SELECT count(*) FROM pg_settings
WHERE category LIKE 'Reporting and Logging%';
 count
-------
    43
```

Statistics Collector

- Not all `track_*` parameters enabled by default
- Dynamic views
  - `pg_stat_activity`, `pg_stat_(replication|wal_receiver)`
  - `pg_locks`, `pg_stat_ssl`, `pg_stat_progress_*`
- Cumulative views
  - Most `pg_stat_*` views
  - Long uptimes cause “lag” for problem detection
- Selective stats reset possible

Extensions

- Most notably **pg_stat_statements** (“top statements”)
- **pg_stat_monitor** (pg_stat_statements on steroids)
- **pgstattuple** (bloat)
- **pg_buffercache** (what’s in the shared buffers)
- **auto_explain** (e.g. to analyze “jumping runtimes”)
- **pg_qualstats** (WHERE predicate stats)
- **pg_stat_kcache** (to sample O/S metrics)
- …

PostgreSQL Monitoring Tools

FIND MORE

https://wiki.postgresql.org/wiki/Monitoring

No shortage of tools!

Getting started with pgwatch

Features

CYBERTEC
DATA SCIENCE & POSTGRESQL
www.cybertec-postgresql.com

Thanks!

Kaarel Moppel

The author or pgwatch v2

CYBERTEC
DATA SCIENCE & POSTGRESQL
www.cybertec-postgresql.com

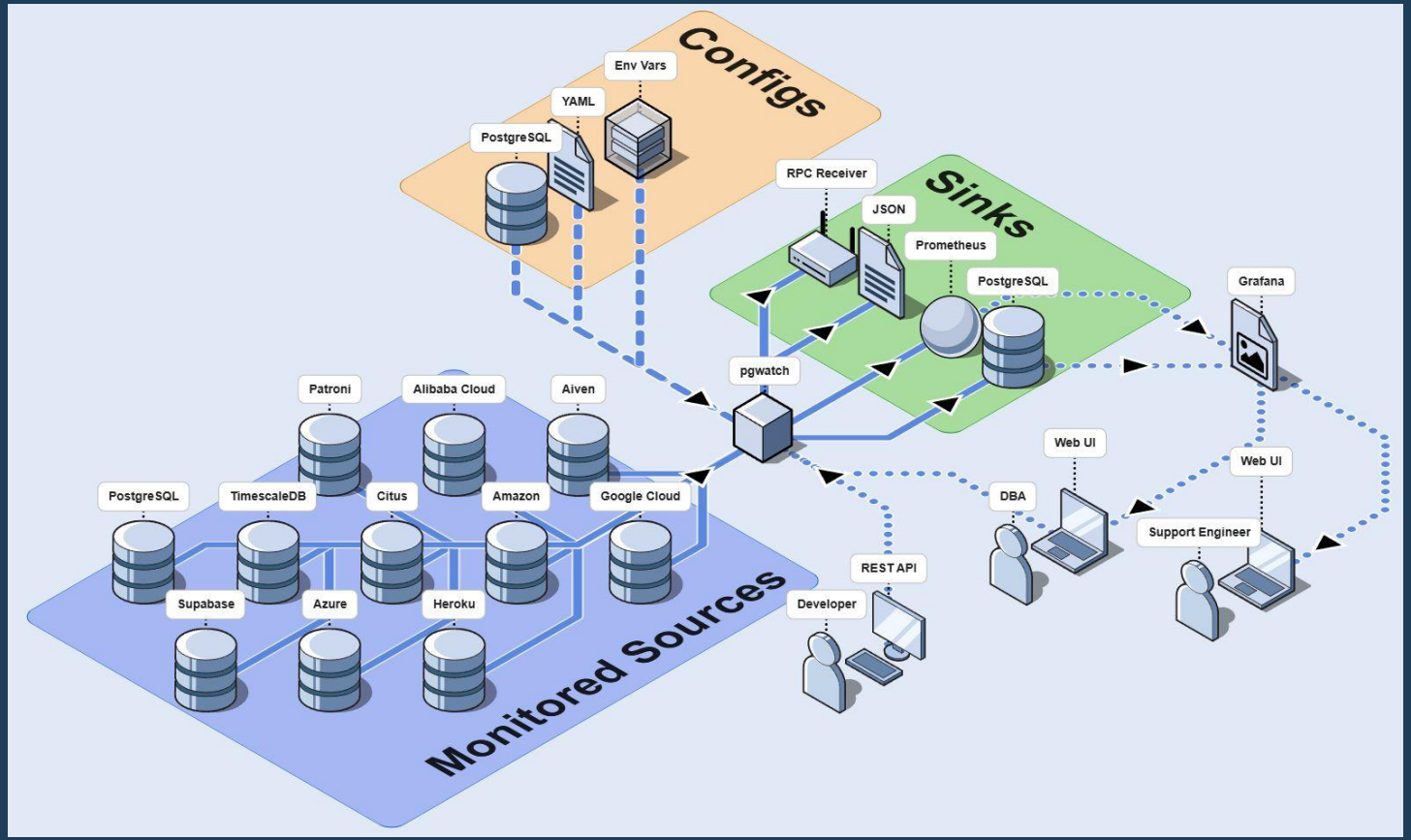
Main principles - why another tool?

- 1-minute setup
  - One binary!
- User changeable visuals / dashboarding
- Non-invasive
  - No extensions or superuser needed for base functionality
- Easy extensibility, do minimal work needed
  - SQL metrics
- Simple alerting via Grafana possible

Architecture components

- Metrics gathering daemon (Golang)
- Configuration
  - PostgreSQL compatible database
  - YAML files
  - ENV vars
- Measurements sinks
  - PostgreSQL (with TimescaleDB optionally)
  - Prometheus
  - Local file
  - RPC
- Optional simple Web UI for administration
- Grafana for intuitive point-and-click dashboarding

Architecture components



Features

- “Ready to go”
  - Default metrics cover all `pg_stat*` views
  - Pre-configured dashboards for almost all metrics
- Supports Postgres 11+ out of the box
- Configurable security - passwords, SSL
- Reuse of existing Postgres, Grafana, Prometheus installations possible
- Kubernetes/OpenStack ready

Features

- Per DB setup with optional auto-discovery of all DBs of a cluster
- Change detection for table/index/sproc/configuration events
- AWS RDS CloudWatch metrics support
- PgBouncer & Pgpool2 metrics support
- Patroni support
- Very low resource requirements
- A Ping mode to test connectivity to monitored databases
- Extensible
  - Custom metrics via SQL, i.e. usable also for business layer!
  - Built-in log parsing and OS level metrics
  - Grafana has lot of plugins as well

CYBERTEC
DATA SCIENCE & POSTGRESQL
www.cybertec-postgresql.com

Alerting

- Quite easy with Grafana, “point-and-click”
- Alertmanager if Prometheus envolved
- Use **pg_timetable** scheduler to analyze and send alerts

Among other things

- TimescaleDB as a metric storage is available out of the box
  - compression is on
  - 45M rows of real world stat_statements data (6db * 3mo)
    - current storage: 70 GB → 8 GB compressed (ratio 8.9x)
    - upcoming feat: 11 GB → 763 MB (ratio 14.5x)

Among other things

- Parallel measurement storages

```
src  master ≡ ?1  1.22.0  ./pgwatch3 --config=sources/sample.sources.yaml
  --sink=jsonfile://metrics.json  --sink=postgresql://pgwatch3@localhost/pgwatch3_metrics -
-sink=prometheus://:9127
2024-04-11 13:00:03.544 [INFO] [sink:jsonfile://metrics.json] measurements sink actviated
2024-04-11 13:00:03.683 [INFO] [sink:postgresql://pgwatch3@localhost/pgwatch3_metrics] init
ialising the measurement database ...
2024-04-11 13:00:03.690 [INFO] [sink:postgresql://pgwatch3@localhost/pgwatch3_metrics] meas
urements sink actviated
2024-04-11 13:00:03.690 [INFO] [sink:prometheus://:9127] measurements sink actviated
2024-04-11 13:00:03.691 [INFO] [metrics:73] [sources:1] host info refreshed
2024-04-11 13:00:03.867 [INFO] Connect OK. [test1] is on version 16.1 (in recovery: false)
2024-04-11 13:00:03.867 [INFO] Trying to create helper functions if missing for "test1" ...
2024-04-11 13:00:03.952 [INFO] [sink:postgres] [rows:1] [db:pgwatch3_metrics] [elapsed:5.83
72ms] measurements written
```

CYBERTEC
DATA SCIENCE & POSTGRESQL
www.cybertec-postgresql.com

Among other things

- Docker images
  - are multiplatform and are slimmer
  - no more monstrous all-in-one image
    - docker compose is shipped instead

```
docker images -a
REPOSITORY                          TAG       IMAGE ID         SIZE
cybertec/pgwatch2-postgres          latest    7a889edafe50   1.17GB
cybertecpostgresql/pgwatch-demo     latest    5e3b24fb5a6a   753MB
cybertecpostgresql/pgwatch          latest    2cbd8fc36e28   44.1MB
```

Getting started with pgwatch

Use cases

CYBERTEC
DATA SCIENCE & POSTGRESQL
www.cybertec-postgresql.com

Use case 1: Live demo

1.
```
docker run -d --name pw3 \
    -p 5432:5432 -p 3000:3000 -p 8080:8080 \
    -e PW_TESTDB=true \
    cybertecpostgresql/pgwatch-demo
```
2. Wait some seconds and open browser at `localhost:8080`
3. Insert your DB connection strings
4. Start viewing/editing dashboards in 5 min...

or check online demo at **https://demo.pgwatch.com**

Use case 2: Find slow queries

1. pgwatch is running with “stat_statements” metric enabled
2. “pg_stat_statements” extension enabled in monitored database
3. Data collected for some time after an hour of regular workload or after peak hours
4. Examine “Stat Statements Top” dashboards

Use case 3: Detect performance spikes

1. pgwatch is running with system and PostgreSQL resource metrics enabled.
2. High-load event occurs (e.g., sudden CPU, memory, or I/O spike).
3. Examine dashboards:

   System Stats → CPU, memory, and disk I/O usage

   DB Overview → General database activity

   Sessions Overview → Detect connection spikes

   Stat Statements Top / Visual → Identify queries causing load

   Checkpointer / Bgwriter / Block IO Stats → Spot I/O-heavy operations
4. Correlate findings to pinpoint the root cause and take action.

Use case 4: Detect bloated tables

1. pgwatch is running with table-level statistics enabled.
2. Examine dashboards:
   - Tables Top → Identify tables with high dead tuples.
   - Table Details → Inspect individual table statistics.
   - Checkpointer / Bgwriter / Block IO Stats → Assess I/O impact.
3. Look for high dead tuples and table size growth.
4. Decide on action:
   - VACUUM / VACUUM FULL if autovacuum isn't sufficient.
   - REINDEX if index bloat is detected.
   - pg_squeeze for online reclaiming of space.

CYBERTEC
DATA SCIENCE & POSTGRESQL
www.cybertec-postgresql.com

Use case 5: Investigate lock contention

1. pgwatch is running with lock monitoring enabled.
2. Examine dashboards:
    - Lock Details → Identify locked and blocking queries.
    - Stat Activity (Realtime) → View active queries and waiting states.
    - Sessions Overview → Check session and connection states.
3. Look for queries holding locks too long.
4. Resolve the issue:
    - Optimize queries to reduce lock duration.
    - Kill problematic sessions (pg_terminate_backend).
    - Adjust transaction isolation levels if needed.

Use case 6: Detect replication lag

1. pgwatch is running with replication monitoring enabled.
2. Examine dashboards:

   Replication → Check replication slots, lag in bytes/time.
3. Look for increasing lag trends.
4. Identify causes & resolve:

   Check network performance.

   Tune replication parameters (wal_keep_size, max_wal_senders).

Use case 7: Analyzing Index Efficiency

1. pgwatch is running with index monitoring enabled.
2. Examine dashboards:
    - Index Overview → Identify frequently and rarely used indexes.
    - Tables Top → Detect large tables where indexing is crucial.
    - Table Details → Review index sizes and hit rates.
3. Look for indexes with very few scans (potentially unnecessary) and duplicate or redundant indexes.
4. Decide on action:
    - Drop unused indexes.
    - Rebuild bloated indexes (REINDEX).
    - Consider index type changes (e.g., B-tree vs. GIN/GiST).

CYBERTEC
DATA SCIENCE & POSTGRESQL
www.cybertec-postgresql.com

Use case 8: Detect pooling issues

1. pgwatch is running with PgBouncer metrics enabled.
2. Examine dashboards:
   - PgBouncer Stats → Monitor connection pool usage and efficiency.
3. Look for:
   - High numbers of waiting clients in PgBouncer.
   - Too many active backend connections, reducing pooling efficiency.
   - Connections stuck in idle-in-transaction state, causing bloat.

Decide on action:

- Tune PgBouncer settings (max_client_conn, default_pool_size).
- Optimize transaction vs. session pooling.
- Adjust PostgreSQL connection limits (max_connections).

Use case 9: Detect changes in settings

1. pgwatch is running with configuration tracking enabled.
2. Examine dashboards:
   - Change Events → Look for recent modifications in PostgreSQL settings.
   - DB Overview time lag → Compare overall performance before/after.
   - System Stats → Identify CPU/memory/disk usage variations.
3. Look for:
   - Critical changes (work_mem, shared_buffers, max_connections, etc.).
   - Unexpected changes (e.g., autovacuum disabled, logging settings altered).
   - Correlation between system performance and configuration changes.
4. Decide on action:
   - Verify when and who changed the settings and restore previous values
   - Implement change tracking (e.g., version control for postgresql.conf).

Be Inspired

“I would rather have questions that can't be answered than answers that can't be questioned.”

Richard Feynman

DON'T BE A STRANGER

PERSONAL PAGE

pashagolub.github.io

PERSONAL GITHUB

www.github.com/pashagolub

CYBERTEC BLOG

www.cybertec-postgresql.com/en/blog/

CYBERTEC GITHUB

www.github.com/cybertec-postgresql

#StandWithUkraine

CYBERTEC
DATA SCIENCE & POSTGRESQL
www.cybertec-postgresql.com