Zabbix 3.0+

Where do we go
Agenda
What we are building?

- Monitoring platform you can trust
- Not limited to IT only
- Trust: reliable, stable, correct, predictable
- Suitable for environments of any size
Let's go back into 1998

Will we meet Perl there?
Toolbox from 1998

- Perl was used initially, then switched to:
  - C language for all critical parts
  - PHP language for the WEB interface
  - SQL back-end
C language

Properties:

+ Low level language
+ Efficient code: fast
+ Lowest resource usage (CPU/mem)
+ Almost no dependencies
+ Write once compile & run everywhere
  
- Slower development
- Memory, lock, pointer related errors
PHP language

+ Fast learning curve
+ Available for all platforms
+ Very actively developed nowadays
- Dynamically typed
- Discipline is required for good code
- Interpreted: errors tend to come up at runtime
SQL back-end

MySQL, PostgreSQL, Oracle, DB2, SQLite

+ Transactional storage engine: consistency
+ Standard API: SQL
+ Easy to deploy
- Scalability
- High-availability

SQL (/ˈɛs ˈkjuː, ˈɛl/) or /ˈsiːkwəl/; Structured Query Language (SQL) is a special-purpose programming language designed for managing data held in a relational database management system (RDBMS).
How C, PHP, SQL affect Zabbix

+ Zabbix is super compact
How C, PHP, SQL affect Zabbix

+ Almost no dependencies
+ Easy to maintain
+ High-performance
+ Low resource usage
+ Supported on all Unix platforms

- Regressions and unfortunate issues (undefined variables)
Some techniques: cache

- used only for backend (Server, Proxy)
Some techniques: bulk operations

- used only for backend (Server, Proxy)

- No individual INSERTS, UPDATE
  We combine them in bulk INSERTs/UPDATEs
Architecture: the good parts

+ Good separation of logic: data collection, problem detection, alerting, visualization, API, etc

+ Multi process application: scales to all available CPU cores

+ Data is always in consistent state
It was a good foundation in 1998

Is it now?
Present challenges

Code duplication (back-end, front-end)

```c
static int validate_host(zbx_uint64_t hostid, zbx_vector_uint64_t *templateids,
                          char *error, size_t max_error_len)
{
    const char
    DB_RESULT
    DB_SCHEMA
    DB_ROW
    DB_SCHEMA
    zbx_graph_items
    size_t
    sql_alloc = 256, sql_offset;
    gitems = NULL, *chd_gitems = NULL;
    gitems_alloc = 0, gitems_num = 0,
    chd_gitems_alloc = 0, chd_gitems_num = 0;
```
Present challenges

Different technology: back-end and front-end

Speed of development

Regressions and quality

Maintaining high quality: tests vs better tools

Historical PHP code

Performance
5 things I'd like to improve in Zabbix

... to start with
WEB interface: facts

- Navigation is not efficient: menu!
- Too many clicks for basic work-flow
  
  “Click until die”, Lukas, Zabbix Conference 2014
- Disconnected information
- Monitoring/administration is strictly separated
- Drop downs: memory usage, performance, usability
WEB interface: UX

• Improve usability

• Navigation: be object-centric
  • Selected host → All information about the host is one-click away

• Information should be interconnected

• Make it faster (also related to API performance)
API: some facts

- Can be extremely slow
- Generates too much SQL queries
- No strict validation
- Weak error reporting
API: faster and more reliable

- Make it **10-100x** faster
  - More efficient algorithms
  - Bulk operations
- Make it **1st class citizen**: possibly move to Zabbix Server side
  - That's where we have the most efficient code
- Implement strict validation
- Error reporting
- Composability
So much valuable data in the database but:

- Quite limited reporting capabilities
- No analytics
- No way to create ad-hoc reports
- No way to memorize parameters of executed reports
• Real-time and analytics

• Important: response time and throughput
Scalability: facts

- Zabbix becomes (much) slower as data volume grows
- Requires special techniques to make it scalable
  (database partitioning)
- Not easy to deliver HA/redundancy
Scalability: terabytes!

• Horizontal scalability of storage engine
  • Standard SQL engines do not deliver it

• Separate storage for historical data

• New distributed monitoring

• Front-end performance is important, sub-sec response
Encryption: facts

- Encryption and authentication are not supported out-of-the-box

- Can be implemented using 3\textsuperscript{rd} party tools (stunnel, OpenVPN, etc)
Encryption & authentication

- Must be part of the product
- Must be easy to enable and maintain
- Note quite sure about strong (SSL/TLS) encryption for Agents
Am I satisfied with existing tools and design decisions?
Am I satisfied with existing tools and design decisions? Not quite.
What changes we may expect in Zabbix 3.0?
What changes we may expect in Zabbix 3.0?
I don't know.
Any questions?

:)