From LLD to SuperDiscovery

How to involve developers in monitoring process
Who am I?

- **Ilya Ableev**, Head of Monitoring Department in Badoo
- Zabbix experience – 7 years (certified specialist)
- Zabbix Moscow Meetup leader for 3 years
- Author of “Zabbix-in-Telegram” tool
Badoo
Bigger than dating :)

- 1 monitoring department
- 5 data centres
- >200 developers
- >300M users
- 2 dell platforms
- 2 offices
- 3K Linux servers
- 2 releases a day
Monitoring Department in Badoo is:

- Controlling everything’s state
- Detecting problems, researching these
- **Notifying** responsible people
- Escalating ongoing problems
- Keeping an eye on things

And doing something with Zabbix :)
Monitoring Department in Badoo

THIS IS FINE.
Zabbix in Badoo
monitors:

**Availability**
Servers, network equipment (simple checks)

**Hardware statistics**
Free space, memory, CPU, load average (standard agent check)

**Services**
Daemons, MySQL, Nginx, PHP-FPM, Docker, Tarantool, a lot of in-house services

**Application metrics**
Queues, RPS, response time, consistency, data freshness

**User activity**
Online users, registrations, messages count, notifications deliverability

And everything we want to know about...
How things became monitored?

When do we need it:

- Thought about monitoring before releasing to production
- We faced a problem, decided to check it constantly
How things became monitored?

What to do:

- Describe requirements (metrics, thresholds, severities, docs)
- Write monitoring code
- Implement it to Zabbix – templates, hosts, items, formulas, triggers
- Enable notifications – users media, actions
Who is being notified?

Operations team:

- One primary engineer on duty ("many to one")
- Quite standard environment (N metrics multiply amount of servers)
- Easy way of managing thresholds (templates)
- No notifications: our team investigates problems, escalates those
Who is being notified?

Developers (large departments and small teams):

- A lot of different applications with completely different logic
- Even more metrics with different behaviour
- Reviews required, something changes or becomes old
- List of responsible people should be up to date: ex-employees, newcomers, vacations, collaboration
- Notifications should remain “fresh”
How was this handled before?

- Need to get info from developers about app’s logic
- Write scripts on our own: bash, perl, php, python, sql, telnet, grep…
- Deploy scripts
- Add items to Zabbix: `system.run[script.php]`
- Clarify limits for triggers, fix them after flapping (**continuously**!)
- Create notifications for everything and everyone, change those from time to time by request from developers (a bit annoying)
- **Click**, click, click, every time we change something
What could have been improved?

- We have a squad of developers who could help us write the code
- No need to write code
- No need to deploy the code
- No need to dig inside an app

\{ saving tons of time \}
- Still should deploy written code
- Still change it in Zabbix
- Still support notifications
- Still clicking
Initial results

- We got rid of maintaining monitoring code
- Developers could create and change it code by themselves

- But we are still supporting items, triggers, notifications – not fast, not flexible
How to become faster and more flexible?

- Provide **admin** access to Zabbix web for developers
- Share knowledge about Zabbix API (allocate another instance)

Both were rejected :)

- Needed to learn what are… hosts, items, triggers, functions, actions…
- Too hard to understand interface
- New universe of documentation about API
Tried low level discovery as an experiment

- Got task to monitor queue processor
- Perfectly fit LLD

Item:
\texttt{count[#QUEUE]} – queue size (trapper)

Function:
\texttt{queue:count[#QUEUE].last(0) > #LIMIT}

Trigger:
Queue size of \{#NAME\} > \{#LIMIT\}

```json
{
  "data": [
    {
      "#QUEUE": 1,
      "#NAME": "name 1",
      "#LIMIT": 1000
    }, {
      ...
    }, {
      "#QUEUE": 99,
      "#NAME": "name 99",
      "#LIMIT": 5000
    }
  ]
}
```
Expanding the experiment

- **Added contacts to LLD!!!** – easily changeable by developers
- Split limits by two severities, used only as indicator
- Item “status[]” – to allow developers to mute trigger

```json
{
    "data": [{
        "{#ID}": 1,
        "{#NAME}": "app stuff",
        "{#RESPONSIBLE}": "user1,user2,manager1",
        "{#PHONES}": "7915123456789,7901987654321"
    }]
}
```
Unifying solution: problems to solve

Hosts

- **Problem**: we can’t create items or trigger without hosts
- **Resolution**: create “fake” hosts (“virtual”) to attach metric to them; developers should provide list of available hosts; hosts could bring some valuable information, e.g. domain zone

Example:

  - `pushopens.dc1` – push opening rates in datacenter 1
  - `emailclicks.dc2` – clicks from emails in datacenter 2
Unifying solution: problems to solve

Metrics (LLD keys, items)

- **Problem**: zabbix must generate items based on LLD’s JSON
- **Resolution**: developers are maintaining “core” code to provide JSON arrays by host
  
  array keys: id, item name, trigger name, emails of responsible people, their numbers (for sms)

Example:

  - `php /opt/www/getJson.php --host=pushopens.dc1`
  - `php /opt/www/getJson.php --host=emailclicks.dc2`
Unifying solution: problems to solve

Data (history)

- **Problem**: we need to get data somehow
- **Resolution**: developers are maintaining code which pushes new data to us from our cloud of scripts using `zabbix_sender`;

  items: count, status, limit_{warning, high, disaster} (used for notifications)
### Discovery rules

<table>
<thead>
<tr>
<th>Name</th>
<th>Items</th>
<th>Triggers</th>
<th>Graphs</th>
<th>Hosts</th>
<th>Key</th>
<th>Interval</th>
<th>Type</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuperDiscovery Item prototypes 5</td>
<td>Trigger prototypes 3</td>
<td>Graph prototypes</td>
<td>Host prototypes</td>
<td>system.run(&quot;php /opt/www/getJson.php --host= {HOST.DNS}&quot;</td>
<td>5m</td>
<td>Zabbix agent</td>
<td>Enabled</td>
<td></td>
</tr>
</tbody>
</table>

Displaying 1 of 1 found
### Item prototypes

<table>
<thead>
<tr>
<th>Name</th>
<th>Key</th>
<th>Interval</th>
<th>History</th>
<th>Trends</th>
<th>Type</th>
<th>Applications</th>
<th>Create enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>count_of_{#NAME}</td>
<td>count[#ID]</td>
<td>1d</td>
<td>0d</td>
<td></td>
<td>Zabbix trapper</td>
<td>SuperDiscovery</td>
<td>Yes</td>
</tr>
<tr>
<td>limit_disaster_{#NAME}</td>
<td>limit_disaster[#ID]</td>
<td>1d</td>
<td>0d</td>
<td></td>
<td>Zabbix trapper</td>
<td>SuperDiscovery</td>
<td>Yes</td>
</tr>
<tr>
<td>limit_fail_{#NAME}</td>
<td>limit_fail[#ID]</td>
<td>1d</td>
<td>0d</td>
<td></td>
<td>Zabbix trapper</td>
<td>SuperDiscovery</td>
<td>Yes</td>
</tr>
<tr>
<td>limit_warning_{#NAME}</td>
<td>limit_warning[#ID]</td>
<td>1d</td>
<td>0d</td>
<td></td>
<td>Zabbix trapper</td>
<td>SuperDiscovery</td>
<td>Yes</td>
</tr>
<tr>
<td>status_of_{#NAME}</td>
<td>status[#ID]</td>
<td>1d</td>
<td>0d</td>
<td></td>
<td>Zabbix trapper</td>
<td>SuperDiscovery</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## How it looks

### Trigger prototypes

<table>
<thead>
<tr>
<th>Severity</th>
<th>Name</th>
<th>Expression</th>
<th>Create enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>{#TRIGGER} @e: [[#RESPONSIBLE]] @t: [[#PHONES]]</td>
<td>{TZabbix_SuperDiscovery_TEMPLATE:status[[#ID]].count(#5,1)=5 and TZabbix_SuperDiscovery_TEMPLATE:count[[#ID]].date(0)&gt;0 and TZabbix_SuperDiscovery_TEMPLATE:limit_warning[[#ID]].date(0)&gt;0</td>
<td>Yes</td>
</tr>
<tr>
<td>High</td>
<td>{#TRIGGER} @e: [[#RESPONSIBLE]] @t: [[#PHONES]]</td>
<td>{TZabbix_SuperDiscovery_TEMPLATE:status[[#ID]].count(#5,2)=5 and TZabbix_SuperDiscovery_TEMPLATE:count[[#ID]].date(0)&gt;0 and TZabbix_SuperDiscovery_TEMPLATE:limit_fail[[#ID]].date(0)&gt;0</td>
<td>Yes</td>
</tr>
<tr>
<td>Disaster</td>
<td>{#TRIGGER} @e: [[#RESPONSIBLE]] @t: [[#PHONES]]</td>
<td>{TZabbix_SuperDiscovery_TEMPLATE:status[[#ID]].count(#5,3)=5 and TZabbix_SuperDiscovery_TEMPLATE:count[[#ID]].date(0)&gt;0 and TZabbix_SuperDiscovery_TEMPLATE:limit_disaster[[#ID]].date(0)&gt;0</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Displaying 3 of 3 found
Example

Too few amount of pushes @e: [ableev,zorin,lobashev] @t: [7915123456789,7900987654321]

Trigger name: “Too few amount of pushes”

Email: ableev@..., zorin@... and lobashev@... – one email with three recipients (hello, ZBXNEXT-3126)

SMS: 7915123456789 and 7900987654321
Summary: flow

- **Developers** are responsible for metrics and thresholds
- They can measure “problem”, they can switch something off for awhile
- **Monitoring** provides them with a semi-API, one dedicated template for each team
- No need to configure actions, those rules apply by developers as well
- Ex-employees/newcomers removed and added automatically
- No need to create zabbix users with media for one action anymore!
Summary: technical details

- Alert script parses `@t` and `@e` arrays from trigger description
- Only one email – killer feature for non-zabbix users
- There is a limitation for amount of keys (def. 200), in case if someone goes “crazy”
- Not only one template: we can disable “bad” templates (flapping, etc)
Questions?

Me: @ableev (Tg, GitHub, Twitter, FB)

Zabbix Moscow Meetup: https://zabbix.moscow

https://techblog.badoo.com