initMAX

Webinar

Extending ZABBIX

all our microphones are muted

ask your questions in Q&A, not in the Chat

use Chat for discussion, networking or applause

1

WHY and HOW

Why to extend?

› Zabbix will evenly distribute checks

› Customized environment

› Specific approach requirement

› Running custom scripts/commands

› Monitoring something that is not available

› out-of-the-box

How to extend?

Zabbix provides multiple different approaches to do that!

Using Zabbix agent checks:
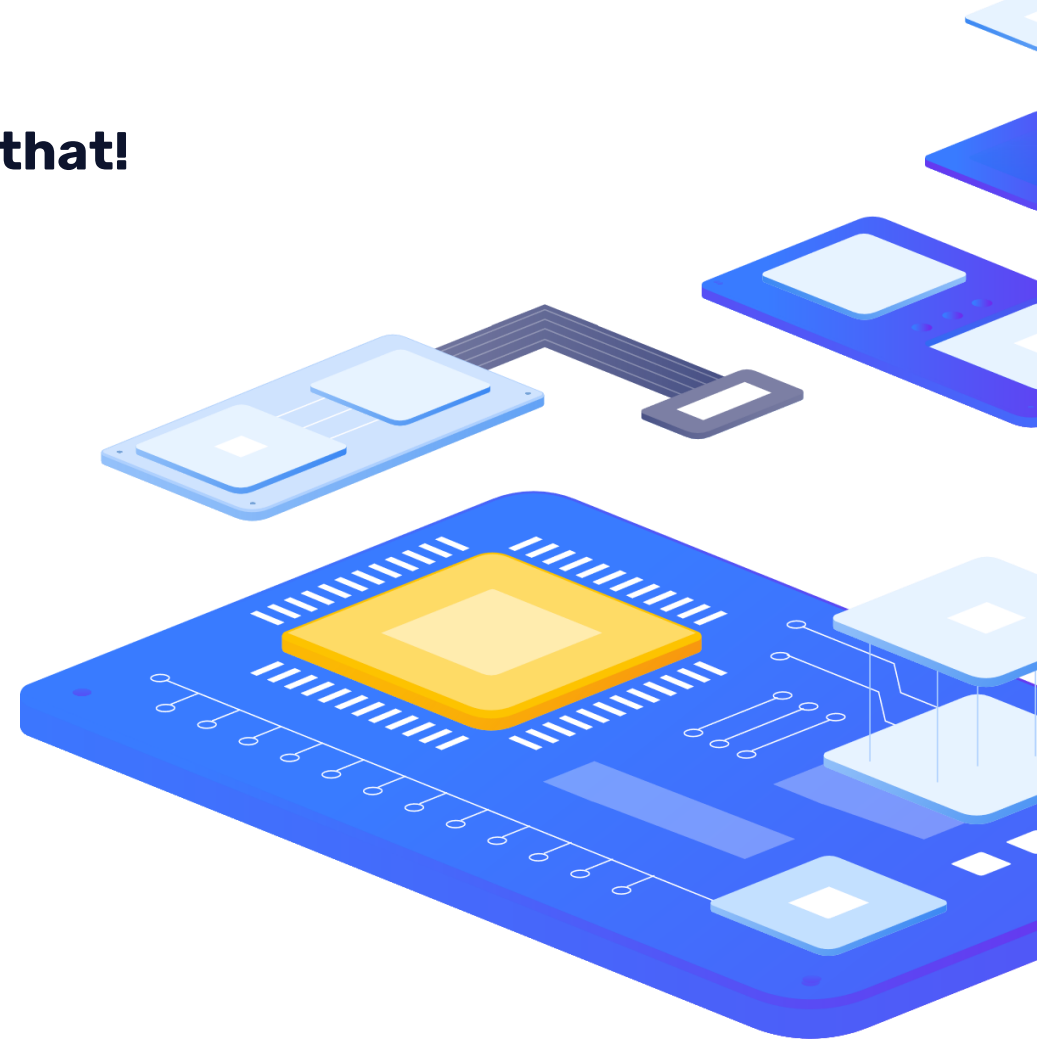
- ❯ Using item key system.run[*]
- ❯ User parameters

Agentless checks

- ❯ External checks
- ❯ Script items

Code-based approach

- ❯ Zabbix API
- ❯ Loadable modules
- ❯ Agent 2 plugins

2
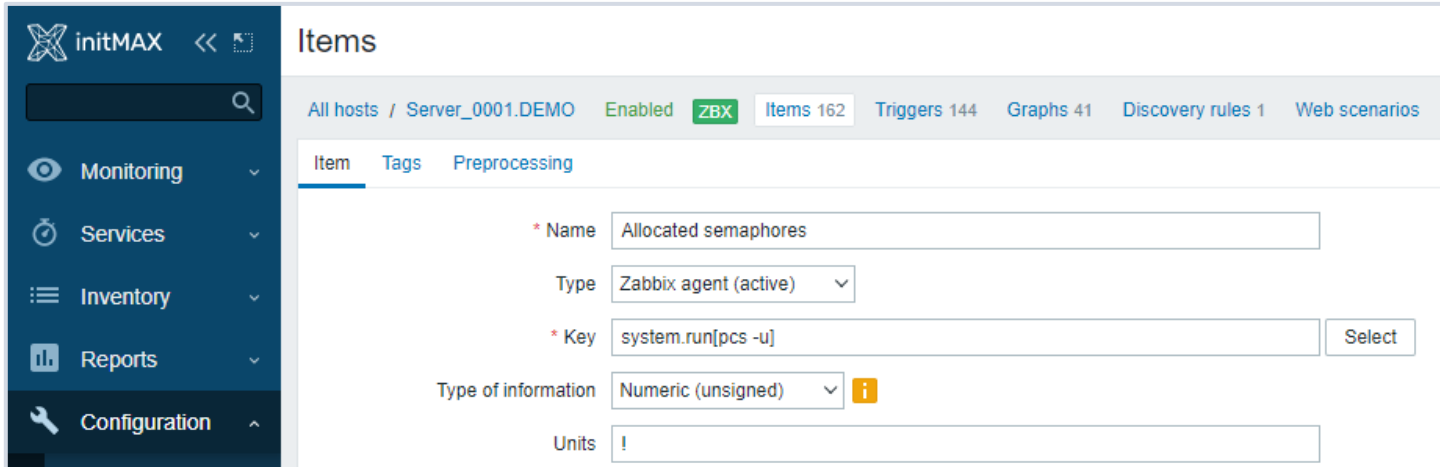
SYSTEM.RUN

SYSTEM.RUN

system.run[command,<mode>]

› command: command that should be executed, i.e. bash or PowerShell

› mode: wait/nowait

  › **wait** – wait till end of execution (default), usually used for data gathering
  › **nowait** - do not wait end of execution, can be used for scheduled command execution

› Be careful with nowait option, Zabbix can create multiple processes
  if execution of the command takes a lot of time

SYSTEM.RUN examples
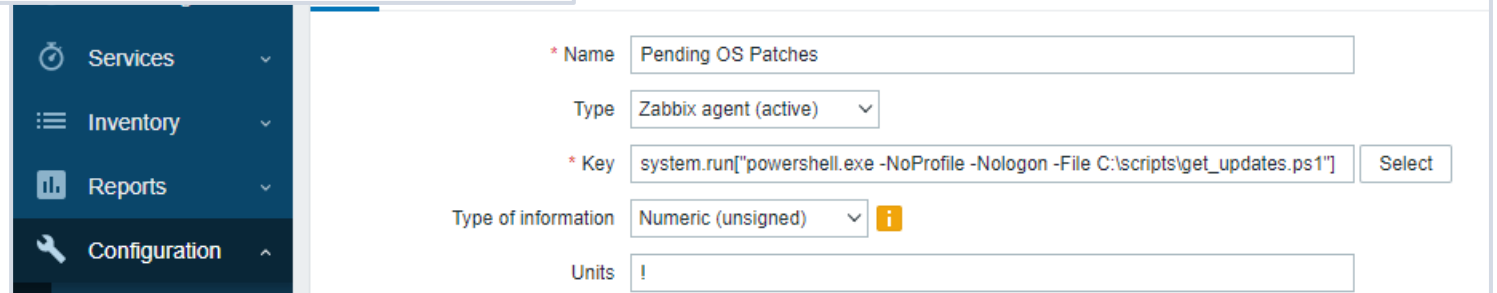
system.run[ipcs -u]

system.run["powershell.exe -NoProfile -Nologo -File C:\scr\get_updates.ps1"]

SYSTEM.RUN examples

Allow remote command execution in the agent configuration:

```
# vi /etc/zabbix/zabbix_agentd.conf (or zabbix_agent2.conf)
```

```
### Option: AllowKey
AllowKey=system.run[ipcs -u]
```

With Zabbix agents before 5.0, you need to add

```
### Option: EnableRemoteCommands - Deprecated
EnableRemoteCommands=1
```

(This will allow any remote command execution)

Restart the agent:

```
# systemctl restart zabbix-agent (zabbix-agent2)
```

Configuring SYSTEM.RUN

Even though it is possible to allow any remote command execution with newer Zabbix agents by adding to agent configuration:

```
### Option: AllowKey
AllowKey=system.run[*]
```

It can raise serious security concerns and a better approach is to allow only approved commands to be executed.
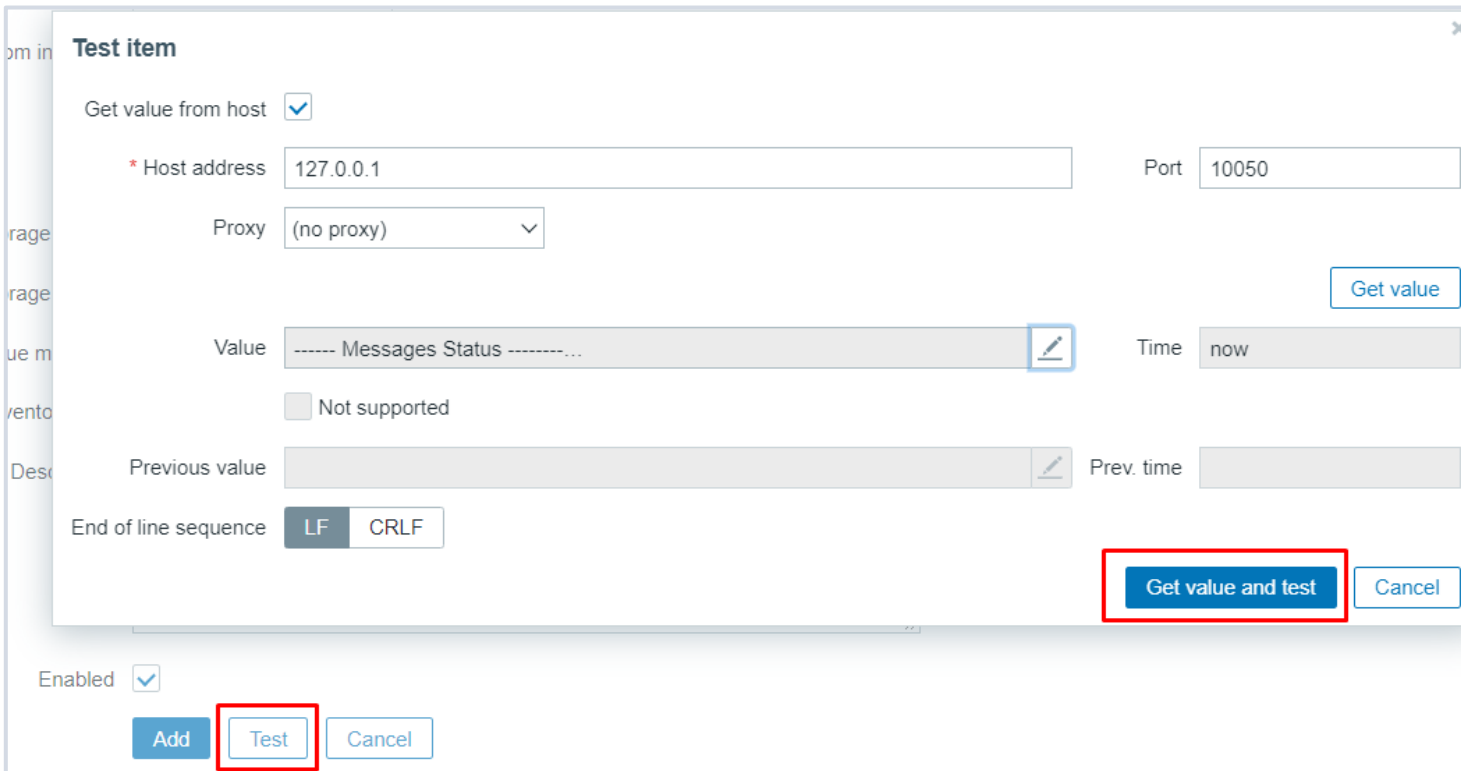
Is there a difference?

```
### Option: AllowKey
AllowKey=system.run[*]
AllowKey=system.run[*,*]
```

Test if your SYSTEM can RUN the command

Starting with 5.0 you can test your newly added system.run items right from the frontend using the test button:

Test if your SYSTEM can RUN the command

Or you can use zabbix_get for the same purpose:

```
# zabbix_get –s <agent-IP> -k system.run['ipcs –u']
```

```
------ Messages Status --------
allocated queues = 0
used headers = 0
used space = 0 bytes


------ Shared Memory Status --------
……
```

Test if your SYSTEM can RUN the command

If during testing, you see errors like:

⚠️ Incorrect value for field "key_": incorrect syntax near ""$8'}"]".

Consider checking your key, if the command has quotes:

```
system.run["ps -ef | grep zabbix | awk {'print $2" "$8'}"]
```

Consider escaping them to execute the command successfully:

```
system.run["ps -ef | grep zabbix | awk {'print $2\" \"$8'}"]
```

3

User Parameters

User Parameters

Another way of executing command, not predefined in Zabbix

› Shell commands

› Custom scripts

Syntax: UserParameter=key,[<command>]

› **key** - The keys that will be used in the item, any unique key can be specified

› **command** - Command that will be executed when the key is requested

All commands are executed under the same OS user under which Zabbix agent is running. Make sure this user will have enough permissions to execute the command specified
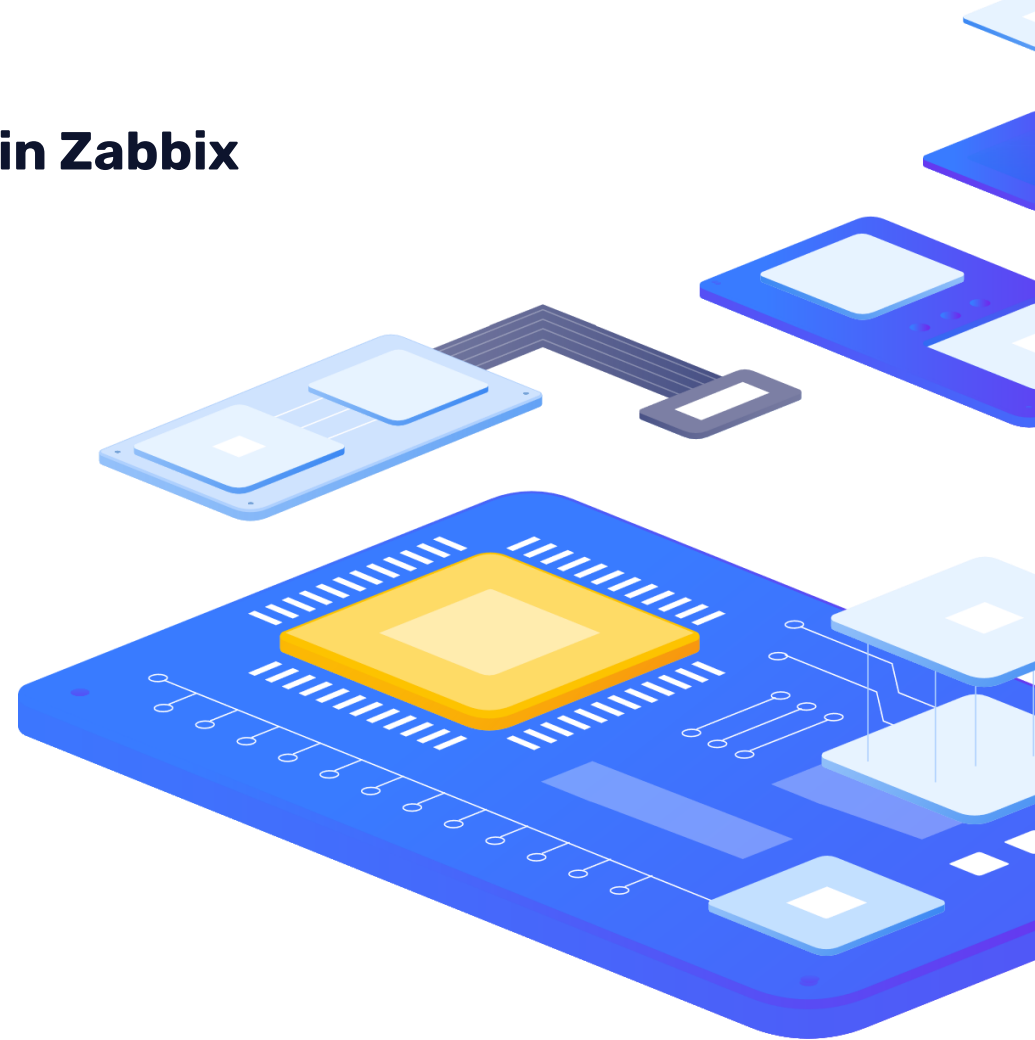
User Parameters

User parameters must be configured for every agent where they will be used:

> Directly in zabbix_agentd.conf or zabbix_agent2.conf files

> Included in a .conf file in the zabbix_agentd.d/zabbix_agent2.d directory (recommended)

```
### Option: UserParameter
#       User-defined parameter to monitor. There can be several user-defined parameters.
#
UserParameter=
```

UserParameter can be simple or flexible:

> Simple: UserParameter=mysql.qps,mysqladmin status | cut -f9 -d":"

> Flexible: UserParameter=mysql.ping[*], mysqladmin -u$1 -p$2 ping | grep -c alive

User Parameters

After defined in the configuration file, UserParameter can be added in the frontend:

Simple: UserParameter=mysql.qps,mysqladmin status | cut -f9 -d":"

* Name	MySQL queries per second
Type	Zabbix agent
* Key	mysql.qps Select

Flexible: UserParameter=mysql.ping[*], mysqladmin -u$1 -p$2 ping | grep -c alive

* Name	MySQL queries per second
Type	Zabbix agent
* Key	mysql.ping[zabbix,{$MYSQL.PASSWORD}] Select

User Parameters

Multiple user parameters can be defined in each agent:

› Multiple include files can be specified with different sets of parameters

› All keys per agent must be unique or Zabbix agent will exit with error:

ERROR: cannot add user parameter "mysql.status,mysqladmin status : key "mysql.status" already exists

Directory from which UserParameter will be executed can be specified:

```
### Option: UserParameterDir
#	When executing UserParameter commands the agent will change the working directory to the one
#	specified in the UserParameterDir option.
UserParameterDir=
```

The return value of the command is standard output together with standard error

› Environment may not be preserved on some Unix systems

User Parameters

Some symbols can not be passed as arguments by default:

› \ ' " ` * ? [ ] { } ~ $! & ; ( ) < > | # @

› newline characters are not allowed

```
### Option: UnsafeUserParameters
#       Allow all characters to be passed in arguments to user-defined parameters.
#       The following characters and newline characters are not allowed:
#       \ ' " ` * ? [ ] { } ~ $ ! & ; ( ) < > | # @
# Range: 0-1
UnsafeUserParameters=1
```

If not set:

⚠ Special characters "\, ', ", `, *, ?, [,], {, }, ~, $, !, &, ;, (,), <, >, |, #, @, 0x0a" are not allowed in the parameters. ✕

User Parameters

To reload list of user parameters:

❯ Restart Zabbix agent to re-read entire configuration

```
systemctl restart zabbix-agent
```

Use a specific Zabbix agent runtime command

❯ Works both for Zabbix agent and Zabbix agent 2

❯ Only user parameters will be updated, other configuration changes ignored

❯ Not supported for zabbix_agentd on OpenBSD, NetBSD and Windows

```
zabbix_agentd –R userparameter_reload
zabbix_agent2 –R userparameter_reload
```

3

External checks

External checks

script[<parameter1>,<parameter2>,...]

› **script:** name of a shell script or a binary.

› **parameter(s):** optional command line parameters.

› Can be passed without parameters - use key script[] or script

› Script must be in the directory defined as the location for external scripts in Zabbix server/proxy configuration file

› Executed by Zabbix server/proxy under zabbix user

› Do not overuse external checks! As each script requires starting a fork process by Zabbix server, running many scripts can decrease Zabbix performance.
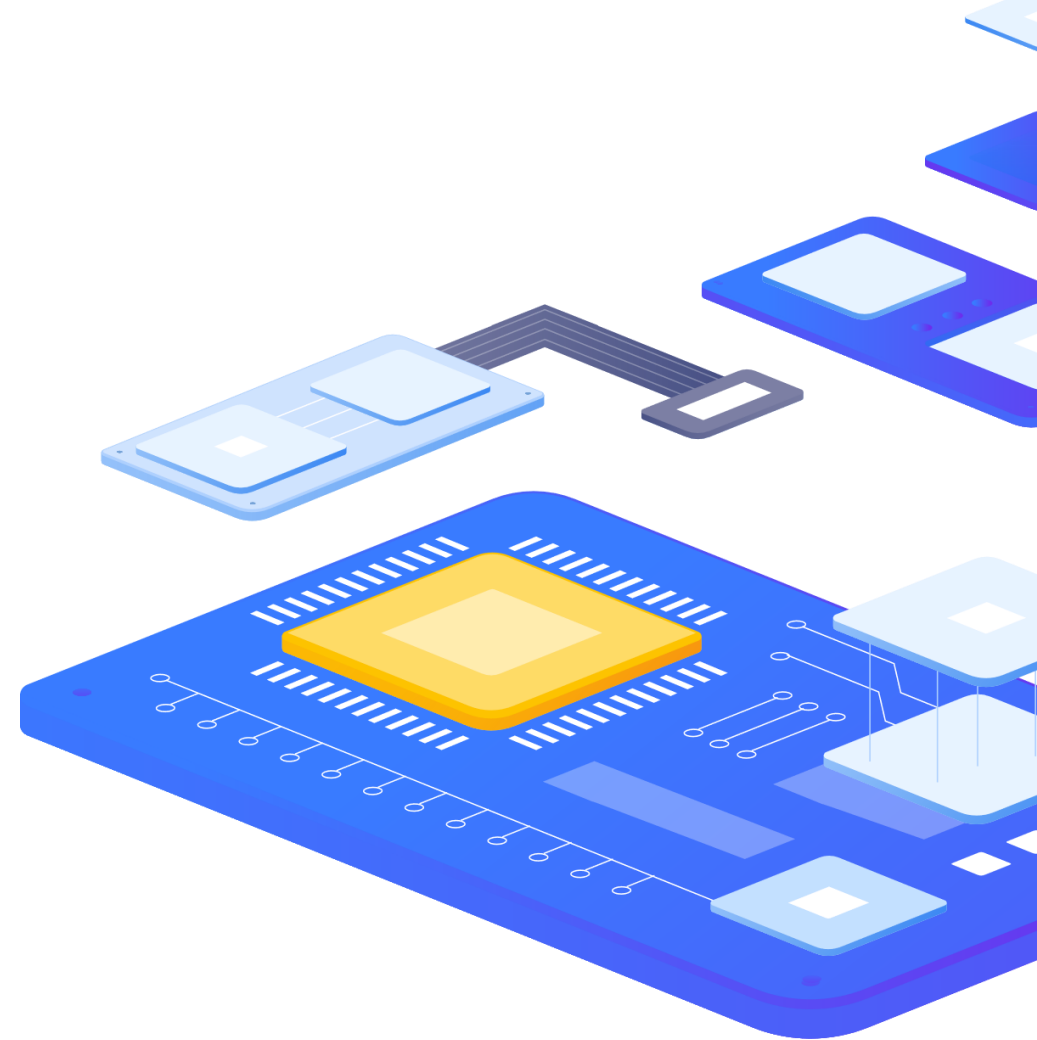
External checks

Create and copy the script to folder defined in ExternalScripts parameter:

```
### Option: ExternalScripts
#       Full path to location of external scripts.
ExternalScripts=/usr/lib/zabbix/externalscripts
```

Make sure the script is executable:

```
# chmod +x check_oracle.sh
```

Test script from the frontend or from command line under zabbix user, i.e.:

```
# su -s /bin/bash -c ./check_oracle.sh zabbix
```

If necessary, add permissions on the command to allow that user to execute it. Only commands in the specified directory are available for execution.

External checks

Create the item in the frontend:

Item	Tags	Preprocessing

* Name	Oracle status
Type	External check ▾
* Key	check_oracle.sh["{HOST.CONN}","{$MYSQL.USER}","{$MYSQL.PWD}"]	Select

Zabbix server or proxy will execute:

```
# ./check_oracle.sh 192.0.0.1 DBuser DBpassword
```

❯ The return value of the check is standard output together with standard error (the full output with trimmed trailing whitespace is returned since Zabbix 2.0).

External checks – Unsupported status

The item will change the status to unsupported if:

> Zabbix server lacks the necessary permissions to execute the script

> Script is not found

> Timeout has been reached while executing the script

> Exit code is not 0

A text (character, log or text type of information) item will not become unsupported in case of standard error output.

4

Script items

Script Items

Item key: any unique key that will be used to identify the item.

› Can be used to collect data by executing a user-defined JavaScript code with the ability to retrieve data over HTTP/HTTPS

› Optional list of parameters (pairs of name and value) and timeout can be specified.

› Executed by Zabbix server or Zabbix proxy

Zabbix uses Duktape, an embedded Javascript engine based on ECMAScript E5/E5.1
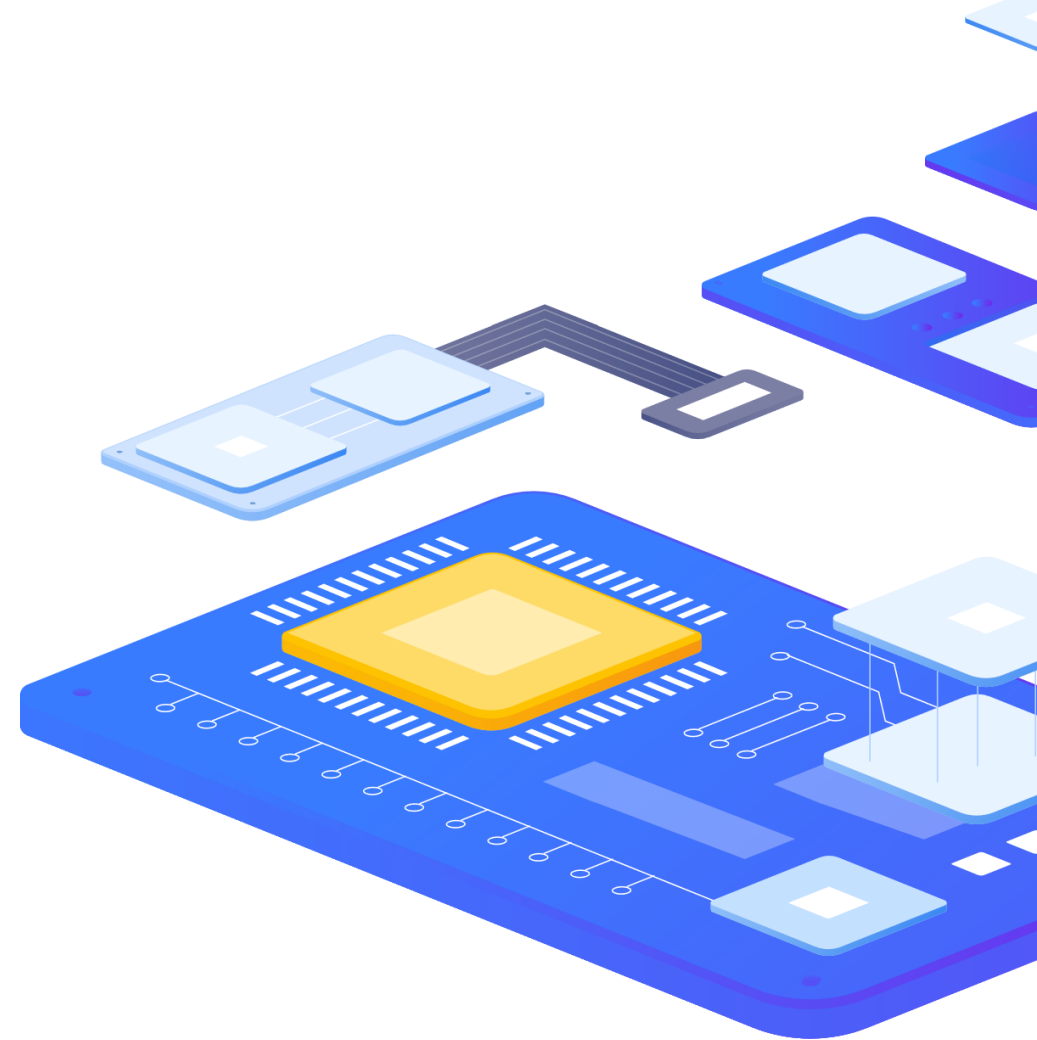
Script Items

Fields you will need to configure:

› Key - enter a unique key that will be used to identify the item.

› Parameters – specify the variables to be passed to the script as the attribute and value pairs. Built-in macros and user macros are supported.

› Script - JavaScript code. This code must provide the logic for returning the metric value. May perform HTTP GET, POST, PUT and DELETE requests and has control over HTTP headers and request body.

› Timeout - JavaScript execution timeout (1-60s, default 3s).

Note: parameters are passed as JSON string, which you can parse to an object and use in the script

Script Items

Create a script type item:

Item	Tags	Preprocessing

* Name	Get Zabbix release notes
Type	Script
* Key	get.release.notes

Select

Type of information	Text

Parameters

Name	Value	Action
url	{$DOMAIN}	Remove
subpage	/release_notes	Remove

Add

* Script	`var json = value;…`

Script Items

> With script like:

```
var obj = JSON.parse(value);
var url = obj.url;
var subpage = obj.subpage;
var request = new HttpRequest();
return request.get(url + subpage);
```

> To get the content of Zabbix release page and make use of parameters.

> Note: multiple HTTP requests can be made too:

```
var request = new HttpRequest();
return request.get("https://www.zabbix.com") + request.get("https://www.zabbix.com/release_notes");
```

5

Zabbix API
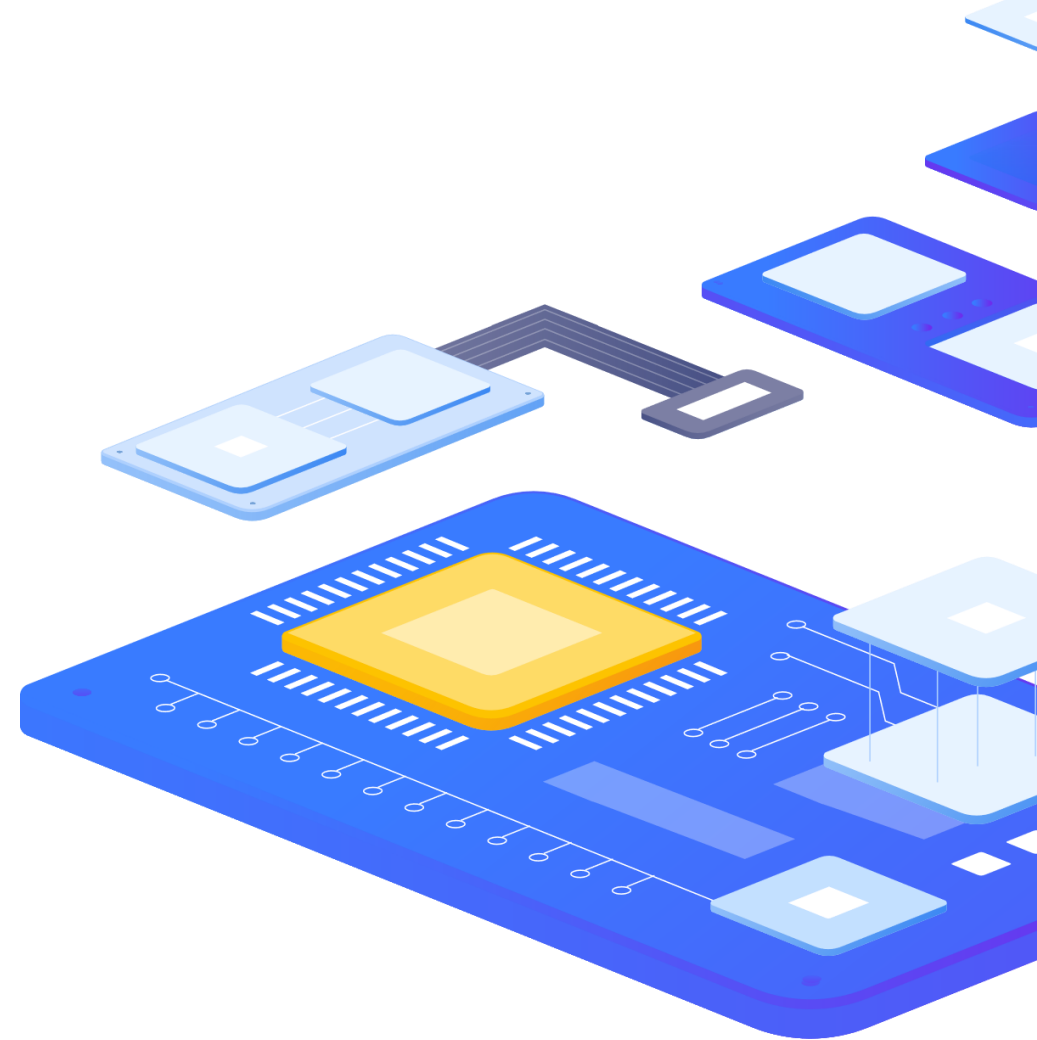
Zabbix API

Allows you to:

› Programmatically retrieve and modify the configuration of Zabbix

› Access historical data

› Create new applications to work with Zabbix

› Integrate Zabbix with third party software

› Automate routine tasks

› Control User Permissions from the frontend

initMAX

Zabbix API

Zabbix API connects to the Zabbix frontend:

› Address that will be used - http://zabbix.example.com/zabbix/api_jsonrpc.php.

› Protocol used – JSON-RPC 2.0

› Consists of multiple separate method, like host.create, history.get, etc.

› Each method described in the documentation with examples

› Communication is encoded using the JSON format.

Zabbix API – Login method

Before getting any data, you will need to login:

```
{
    "jsonrpc": "2.0",
    "method": "user.login",
    "params": {
        "user": "zabbix_api",
        "password": "afHhYTTQhsBX"
    },
    "id": 1,
    "auth": null
}
```

And get the authentication token:

```
{
    "jsonrpc": "2.0",
    "result": "0424bd59b80767b54e4191e7",
    "id": 1
}
```

Zabbix API – API Token

Or event simpler:

generate API token in the Zabbix frontend:

≡ API tokens ˅

* Name   API Token
* User   Admin (Zabbix Administrator) ✖   Select
Description
Set expiration date and time   ☑
* Expires at   2022-12-31 00:00:00
Enabled   ☑

Add   Cancel

≡ API tokens ˅

✓   API token added

Name: API Token

User: Admin (Zabbix Administrator)

Auth token: 3b0adba9054ce303a7cce850fab9a7e4c326e002c787b2dc825260ae3172c219 ℹ️ Copy to clipboard

✖
Make sure to copy the auth token as you won't be able to view it after the page is closed.

Expires at: 2022-12-31 00:00:00

Description:

Enabled: ☑

Close

Zabbix API – Method Call

Now you can configure your Zabbix and get needed reports trough API, like finding hosts where inventory field os contains Centos:

```json
{
"jsonrpc": "2.0",
"method": "host.get",
"params": {
"searchInventory": {
"os": "Centos"
},
"selectInventory": ["os","type"],
"output": ["hostid","name"]
},
"auth": "<PUT AUTHENTICATION TOKEN HERE>",
"id": 1
}
```

Zabbix API

It is possible to use any utility to POST
the JSON-RPC data to Zabbix API:

GUI based utilities:

Service URL

🌐 http://company.com/zabbix/api_jsonrpc.php

Request JSON String

```
{
    "jsonrpc": "2.0",
    "method": "user.login",
    "params": {
        "user": "zabbix_api",
        "password": "afHhYTTQhsBX"
    },
    "id": 1,
    "auth": null
}
```

Call Method ⓘ Uses the HTTP POST method when submitting data.

Command line utilities:

```
curl -s -X POST -H 'Content-Type: application/json-rpc' -d '
{"jsonrpc":"2.0","method":"user.login","params":
{"user":"zabbix_api","password":"afHhYTTQhsBX"},
"id":1,"auth":null}
' http://www.initmax.cz/zabbix/api_jsonrpc.php
```

Zabbix API

Or even by using various programming or scripting languages :

❯ Use programming language you are familiar with

❯ Control workflow using built-in operands

❯ Some programming languages have Zabbix API plugins

6

Loadable Modules
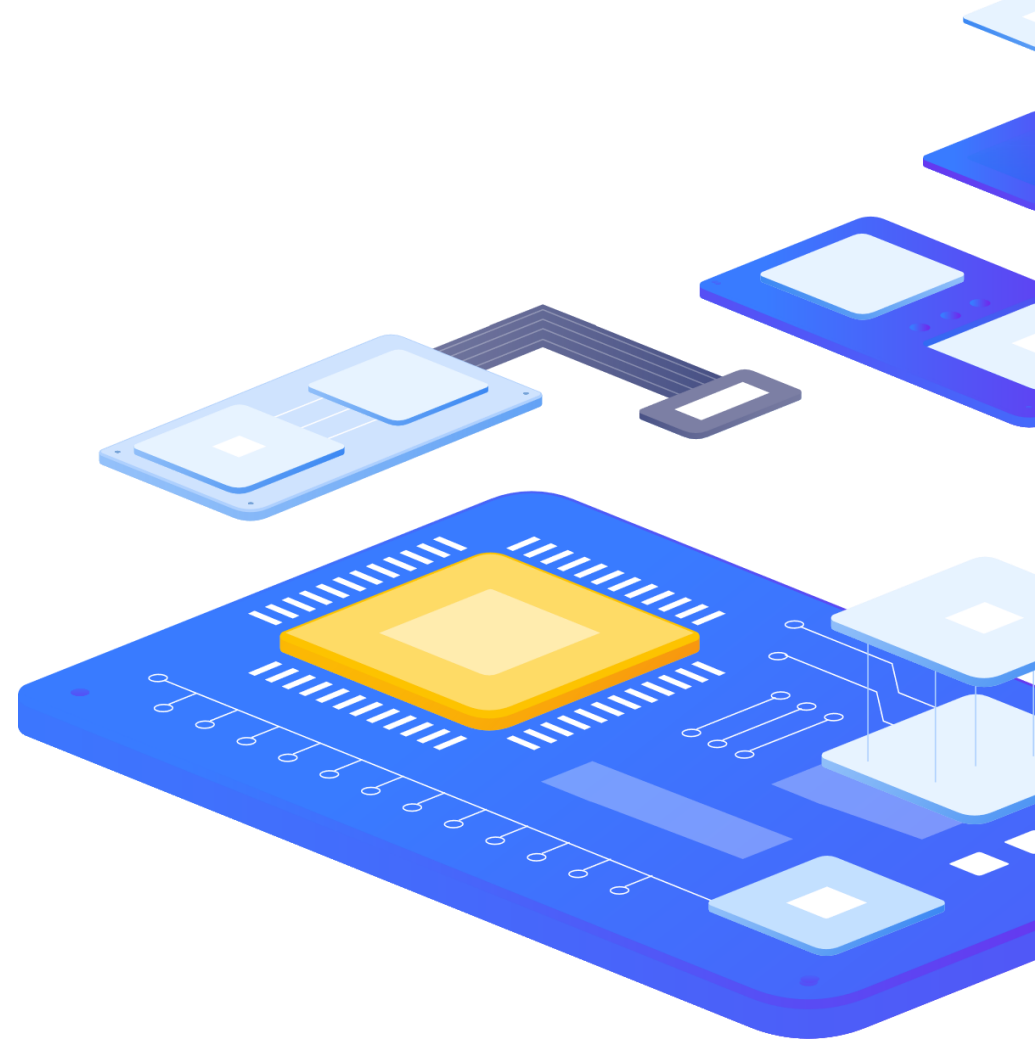
Loadable Modules

Ability to implement any logic in C language:

› Is a shared library used by Zabbix daemon and loaded on startup

› Typically, modules will have .so extension

› Can be built into Zabbix server, agent or proxy

› perform ~10 times faster than user parameters, commands or scripts

Can be used only within Unix platforms. Won't work for agents in a Windows environment

Loadable Modules

There are currently six functions in the Zabbix module API:

One is mandatory:

› zbx_module_api_version() - returns the API version implemented by this module

Five other are optional:

› zbx_module_init() - performs the initialization for the module

› zbx_module_item_list() - returns a list of item keys supported by the module

› zbx_module_item_timeout()- specifies the timeout for items implemented by the module

› zbx_module_history_write_cbs() - returns functions to write history data of different types

› zbx_module_uninit() - performs the necessary uninitialization such as freeing allocated resources, closing file descriptors, etc.

Loadable Modules

Zabbix agent, server and proxy support two parameters to deal with modules:

LoadModulePath – full path to the location of loadable modules

LoadModule – module names to load at startup, which contain:

> Module name for modules included in the LoadModulePath

> Module name with a full path starting with / (LoadModulePath is ignored)

```
LoadModulePath=/usr/local/lib/zabbix/agent/
LoadModule=mysql.so
LoadModule=apache.so
LoadModule=/home/myuser/mymodule.so
```

Zabbix component will fail to start if:

> The module file is missing

> In case of bad permissions (must be readable by Zabbix user)

> If a shared library is not a Zabbix module

7

GO Plugins

GO Plugins

Zabbix Agent 2 gives Zabbix more capabilities for data collection on the "GO"

› Less complicated than C Loadable modules, so creating plugins is much more accessible.

› Agent interacts with plugins through a two-tier task queue:

› Each plugin has a task queue;

› Scheduler has an active plugin queue.

› Ensures better concurrency.

GO plugins are available exclusively for Zabbix Agent 2
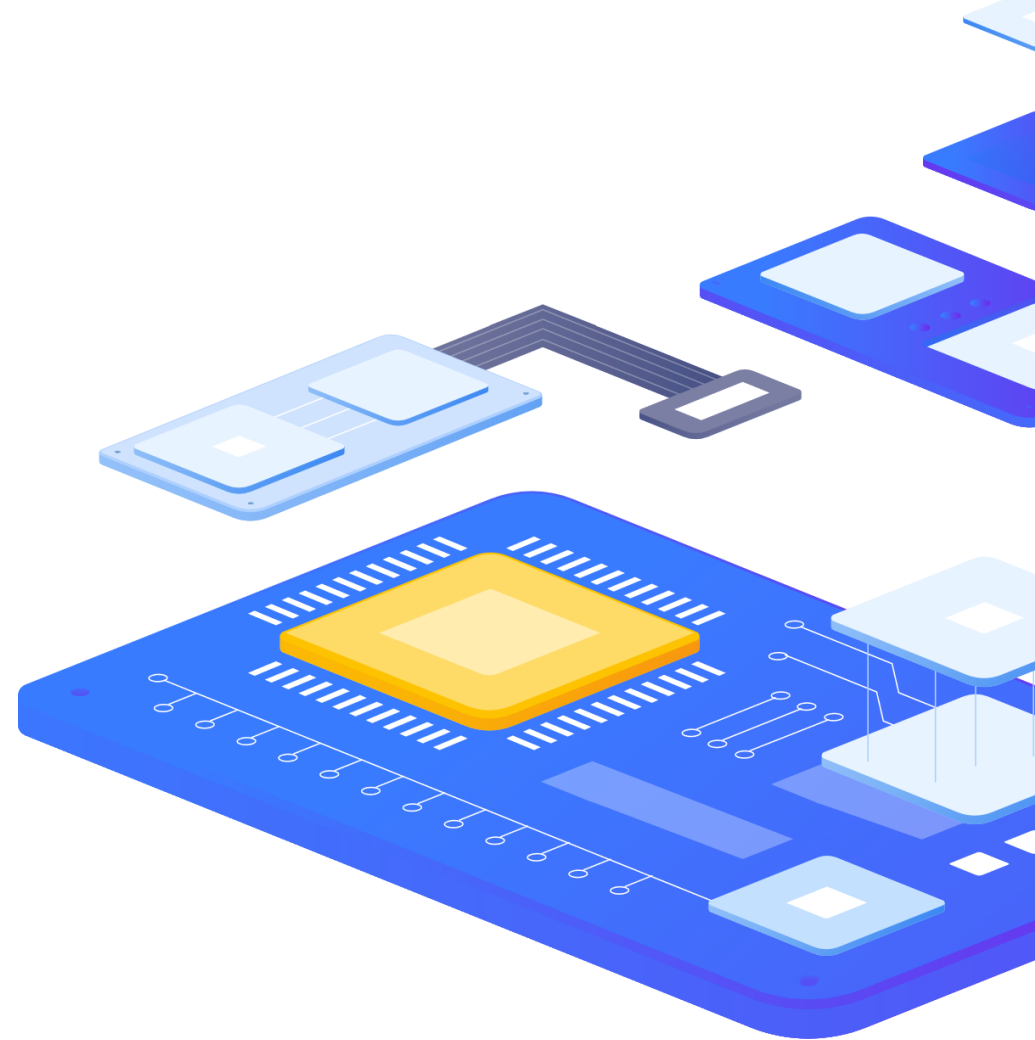
GO Plugins

For Zabbix Agent 2 five interfaces are available:

› **Exporter** - A very simple interface that polls metrics and returns a value, several values, an error, or nothing at all

› **Watcher** - With Watcher you can implement a metric polling process without using Scheduler. This may be useful for plugins that use trapping

› **Collector** - is used for plugins that need to collect data regularly. However, it can't return data, so you'll need Exporter for that.

› **Runner** - provides a way to perform initialization when a plugin is activated (the Start() function) and deinitialization when it is stopped (the Stop() function).

› **Configurator** - serves for configuring plugins.

GO Plugins

A plugin is simply a Go package with one or several interfaces that define its logic:

```
package packageName
import "zabbix.com/pkg/plugin"
type Plugin struct {
    plugin.Base
}
var impl Plugin
func (p *Plugin) Export(key string, params []string, ctx plugin.ContextProvider) (res interface{}, err error) {
    // Write your code here
    return
}
func init() {
    plugin.RegisterMetrics(&impl, "PluginName", "key", "Description.")
}
```

8

Questions?

CONTACT US:

Phone:	〉	+420 800 244 442
Web:	〉	https://www.initmax.cz
Email:	〉	tomas.hermanek@initmax.cz
LinkedIn:	〉	https://www.linkedin.com/company/initmax
Twitter:	〉	https://twitter.com/initmax
Tomáš Heřmánek:	〉	+420 732 447 184

initMAX