ZABBIX '26
CONFERENCE
GERMANY

WORKSHOP

Practical Use Cases for Integrated JavaScript in Zabbix

Inqbeo · github.com/inqbeo/zabbix-js-workshop

45 minutes · Berlin, 2026

Why does a GUI tool need code?

Zabbix is fully GUI-configurable — and that's the point. Until it isn't.

WHAT THE GUI HANDLES

- Low-Level Discovery, dependent items, calculated items
- JSONPath / XPath / regex preprocessing chains — more capable than people expect
- Trigger expressions, host inventory, actions, escalations
- Out-of-the-box integrations: SNMP, IPMI, JMX, agent
- Monitoring-relevant entities nicely bundled into templates

WHERE THE CEILING SHOWS

- API response shape just doesn't match what JSONPath can express
- A check needs two API calls chained — second URL depends on first
- Notification target needs a custom JSON envelope
- Low-Level Discovery from arbitrary, non-tabular data structures
- Five preprocessing steps for what three lines of code would do

Since 4.4, Zabbix ships a sandboxed JavaScript layer for exactly these cases.

Where JavaScript runs in Zabbix

Server-side. Embedded. Sandboxed. NOT in the browser, NOT on the agent.

✗ **Not in the Zabbix frontend (PHP). Not in the user's browser. Not on the agent.**

ZABBIX SERVER / PROXY PROCESS

DATA COLLECTION

Item Preprocessing — since 4.2

Reshape / validate values per check

Script Item — since 5.2

Full collection logic · up to 10 HttpRequest objects

Browser Item — since 7.0

Headless Chrome via WebDriver · not exercised in this workshop

NOTIFICATION / ACTION

WEBHOOK ENGINE

Media Type — since 4.4

Triggered by alerts. Ready-made for MS Teams, Jira, ServiceNow, OpsGenie, …

Global / Manual Script (type = Webhook) — since 5.4

Triggered manually from a host or problem context menu in the GUI.

Duktape — how it differs from modern JavaScript

ECMAScript 5 with a few sprinkles. Embedded. Synchronous. Not the JS you write at home.

MODERN JS (Node, browser)

```
const codes = data.hourly.weather_code;
const summary = codes
    .filter(c => c >= 60)
    .map(c => `code-${c}`)
    .join(', ');

const fetchOne = async (url) => {
    const r = await fetch(url);
    return r.json();
};
```

DUKTAPE IN ZABBIX

```
var codes = data.hourly.weather_code;
var parts = [];
for (var i = 0; i < codes.length; i++) {
    if (codes[i] >= 60) {
        parts.push('code-' + codes[i]);
    }
}
var summary = parts.join(', ');

// no fetch, no async — synchronous HttpRequest
var r = new HttpRequest();
var body = r.get(url);
```

WHAT IS NOT THERE

let, const (block-scoped), arrow functions, template literals, destructuring, default parameters, spread/rest

async / await / Promises · fetch · require / import · NPM · no callback-style I/O — every call blocks

What you actually have

Just enough vocabulary to read the exercise code.

GLOBALS & STDLIB

```
value
JSON.parse / stringify
Date,  Math
encodeURIComponent
btoa,  atob   (base64)
md5,  sha256
hmac,  sign
```

OBJECTS

```
Zabbix.log(level, msg)
Zabbix.sleep(ms)
new HttpRequest()
  .get / post / put / delete
  .addHeader(...)
  .getStatus()
XML.fromJson / toJson
```

BROWSER ITEM EXTRAS

```
new Browser(opts)
  .navigate(url)
  .findElement(strategy, ...)
  .getScreenshot()  → base64
  .collectPerfEntries()
Element.getText / sendKeys / click
BrowserError
```

Full reference: docs.zabbix.com → Items → JavaScript preprocessing → JavaScript objects

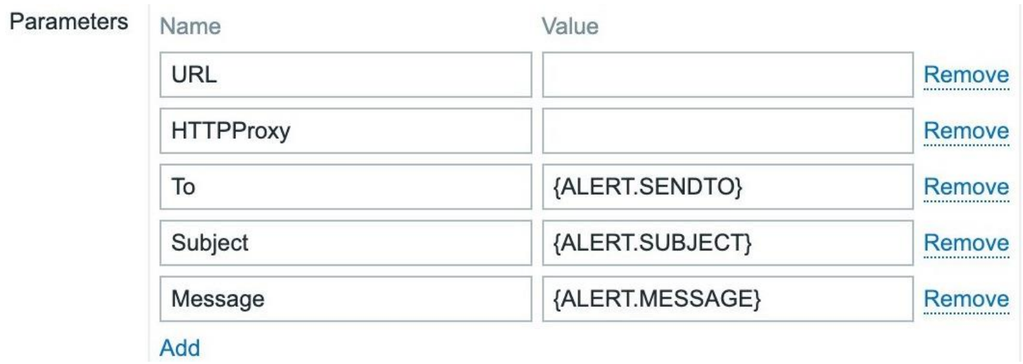
Parameters in, return out

Same convention everywhere: parameters arrive as a JSON string named value.

SCRIPT ITEM · parameters as you define them


```
// In the Script field:
var p = JSON.parse(value);
var conf  = p.myconf;        // "myvalue"
var other = p.myotherconf;   // "myothervalue"
return JSON.stringify({ result: conf });
```

WEBHOOK · parameters from {ALERT.*} macros




```
// In the Script field:
var p = JSON.parse(value);
var url     = p.URL;
var subject = p.Subject;     // {ALERT.SUBJECT} substituted
var msg     = p.Message;     // {ALERT.MESSAGE} substituted
return 'OK';
```

THE CONTRACT

value is always a string. For Script items and webhooks it is JSON — JSON.parse it. For preprocessing it is the raw item value.

return whatever you want Zabbix to see. Items want a stringified value (use JSON.stringify for structured data).

throw 'message' marks the item / webhook as failed; the message lands in the alerter or item error column.

Macros in JavaScript

Two macro families. Two integration points. Don't conflate them.

USER MACROS · {$NAME}

Resolved into your source by server / proxy BEFORE the script runs. Just write them inline.

```
// {$THRESHOLD} substituted as text:
var threshold = '{$THRESHOLD}';
if (!isNaN(threshold) && Number(value) > threshold) {
    return threshold;
}
return value;
```

Substituted as raw text — quotes, spaces, or special chars in the macro value can break your script. Always wrap string macros in quotes; cast numerics with Number().

BUILT-IN MACROS · {HOST.HOST}, {ITEM.LASTVALUE}, ...

NOT substituted into the source. Pass them through the parameter list instead.

```
// In the Parameters list:
//   Host = {HOST.HOST}
//   Last = {ITEM.LASTVALUE}

var p = JSON.parse(value);
var url = 'https://api/' + p.Host;
```

Works in: Script items · Browser items · Webhooks (this is what {ALERT.*} on the previous slide was).

Does NOT work in: Item Preprocessing source.

BOTTOM LINE

Use {$USER_MACROS} as compile-time constants — anywhere any script runs.

Use {BUILT.IN} macros only as parameter values — and parse them out of value in your code.

Recommendation: Develop outside Zabbix

The dev workflow Zabbix doesn't quite give you.

(a) THE DEV TOOL: zabbix_js

```
zabbix_js -s script.js -i fixture.json
```

Keep your script in a .js file. Keep its input (the value JSON) in a .json file. Iterate at the shell.

Both files live in git — diffable, reviewable, with history.

CI runs zabbix_js on every commit against a folder of fixtures and diffs the output.

Deploy with the Zabbix API: template.import, mediatype.update, script.update.

GET IT

Linux: Zabbix repo → apt install zabbix-js / dnf install zabbix-js

Windows: no native build — install via WSL with the Linux package above.

macOS: no native build — install in a Linux VM or container with the package.

(b) THE GOTCHA: SCRIPTS LIVE IN THE DB

No file. No git. No diff between revisions.

The audit log records that a change happened — not what the 200-line webhook looked like yesterday.

Zabbix.log() writes to /var/log/zabbix/zabbix_server.log. No log view in the GUI.

DebugLevel ≥ 3 already adds a lot of unrelated noise. DebugLevel 4 makes Zabbix.log(4, ...) visible — and grows the log fast.

So: develop on a separate test Zabbix at DebugLevel 4. Production stays at 3.

The scenario

Why we are about to write JavaScript for the next half hour.

ACME LOGISTICS

A small DACH warehouse operator. Sites in Berlin, Hamburg, Munich, Vienna, Zurich.

Outdoor staging areas. Perishable goods. SLA penalties when icy roads delay deliveries.

Operations wants outdoor weather monitored per warehouse city — and the duty manager pinged when conditions go sideways.

ONE HOST PER CITY

Each warehouse is a Zabbix host. The city name is a host macro.

WEATHER AS ITEMS

Temperature, wind, humidity, 6-hour rain forecast. Trigger when bad.

PUSH WHEN IT MATTERS

On problem, the duty manager gets a notification on their phone or laptop.

Exercise preparations

Three things before we start. Takes about a minute.

1. GET YOUR NUMBER

Each participant gets a slip from the trainer

01 – 40

The number is yours for the next 30 minutes. Credentials are on the slip.

2. OPEN YOUR LAB

Empty Zabbix 7.4, fresh per participant

`student-NN.zabbix-workshop.inqbeo.de`

Replace NN with your number. HTTPS, browser-only — no SSH needed.

3. OPEN THE REPO

Step-by-step instructions live here

`github.com/inqbeo/zabbix-js-workshop`

Per-exercise README with click-by-click instructions, starter and solution code, fixtures for zabbix_js.

Three exercises

One JS engine. Three places where the declarative GUI runs out.

01 PREPROCESSING RUNS OUT

Item Preprocessing

Declarative chains can't shape this value. One JS block flattens the API, translates the WMO code, derives a warning flag.

10 minutes

02 ONE CALL ISN'T ENOUGH

Script Item

The second URL depends on what the first returned. Geocode the city, then fetch its forecast — chained in one item.

10 minutes

03 NO MEDIA TYPE FOR THIS

Webhook Media Type

Zabbix ships 41 media types — ntfy isn't one. Write the webhook yourself: alert in, HTTP POST out, watch it fire.

10 minutes

NOTIFICATION TARGET: ntfy

Open-source notification service. Self-hostable.

You POST a plain-text message with HTTP headers for title, priority, tags. The body is the message.

Anyone subscribed to that topic — via a browser tab, mobile app, or curl — receives it instantly.

```
curl -d "Berlin: rain warning"  \
     -H "Title: Acme Warehouse Berlin" \
     -H "Priority: 4"                  \
     -H "Tags: warning,cloud_rain"     \
     https://ntfy.zabbix-workshop.inqbeo.de/student-04
```

ZABBIX

01

EXERCISE 1

JavaScript Preprocessing

Preprocessing is declarative — JSONPath, regex, multiplier. When that's not enough, JS is the escape hatch.

WHAT YOU'LL DO

- Flatten Open-Meteo's nested response into the seven top-level fields the dependents expect
- Translate the numeric WMO weather code (e.g. 61 → "Light rain") with a lookup table
- Aggregate next-6h max temperature and total precipitation across hourly arrays
- Derive a rain_warning boolean from the precipitation total

STARTER & STEP-BY-STEP

github.com/inqbeo/zabbix-js-workshop/exercise-1/

ZABBIX

02

EXERCISE 2

Script Item

When the second URL depends on what the first returned. An HTTP item can't express that — a Script item can.

WHAT YOU'LL DO

- Read the city name from the parameter table (delivered as JSON in value)
- GET the geocoding API → { latitude, longitude }
- GET the forecast API for those coordinates
- Return one combined JSON document — throw on errors so Zabbix marks unsupported

STARTER & STEP-BY-STEP

github.com/inqbeo/zabbix-js-workshop/exercise-2/

ZABBIX

03

EXERCISE 3

Webhook → ntfy

Zabbix ships 41 media types — none of them ntfy. Build your own; click Test and watch it fire.

WHAT YOU'LL DO

- Define the parameter table — alert subject, message, severity, recovery flag
- POST to ntfy with Title, Priority, and Tags HTTP headers
- Map Zabbix severity to ntfy priority and emoji tags
- Test from the dialog; wire it to a real action so triggers fire live notifications

STARTER & STEP-BY-STEP

github.com/inqbeo/zabbix-js-workshop/exercise-3/

What we did NOT cover

None of these are advanced — just things we didn't have time for. Examples in the repo.

Browser items

Real Chrome via WebDriver. Element / Browser objects. Base-64 screenshots into Binary items.

The XML object

Parse / build / XPath. Useful for SOAP-era APIs and old vendor exports.

Webhook tags & ticket integrations

Return `{tags: {ticket_id: ...}}` so Zabbix associates the event and updates on recovery.

setHttpAuth, setProxy, customRequest

The rest of the HttpRequest API surface.

Calling Zabbix's own API

Manual host scripts and event actions can call `/api_jsonrpc.php` on their own server. One click → tag events, update problems, reconfigure hosts. Automation from inside Zabbix.

LLD via JavaScript

Turn arbitrary input into `{data: [{"{#NAME}": ...}]}` for Low-Level Discovery, when the data shape is too irregular for declarative LLD.

LIMITS TO REMEMBER

10 s execution timeout · 64 MB heap · 10 HttpRequest objects per run · Zabbix.log() debug output capped at 8 MB per run

TAKE AWAY

JavaScript gives Zabbix real superpowers.

Once you know the gotchas, it's easy.

And — it's a lot of fun.

WORKSHOP REPO

github.com/inqbeo/zabbix-js-workshop

README per exercise with step-by-step instructions, starter and solution code, fixtures for zabbix_js, and these slides as PDF at the repo root.

INQBEO

inqbeo.de

Open-source consulting & training: Kubernetes, monitoring, GitOps, DevOps.

Questions ?