

From Basics to Synthetic Automation

Unlocking Seamless Web User Journey Verification in Zabbix 7.0

Dimitri Bellini

Zabbix Certified Expert & Presenter | Quadrata



THE EVOLUTION OF WEB MONITORING

Traditional reachability is blind to frontend user experiences

Why Ping & Port Checks Fall Short

- **Ping & HTTP Status Codes Only Validate Infrastructure:** A server returning HTTP 200 may still render an empty page or a broken JavaScript error.
- **Single Page Applications (SPAs):** Modern frameworks (React, Angular, Vue) build the DOM dynamically on the client-side. Web crawlers miss layout bugs.

Synthetic Monitoring Emulates Real Users: Guarantees customer retention by validating transaction flows, database writing, and security logins proactively.

→ **From infrastructure to Business Services**

Our Evolutionary Web Monitoring

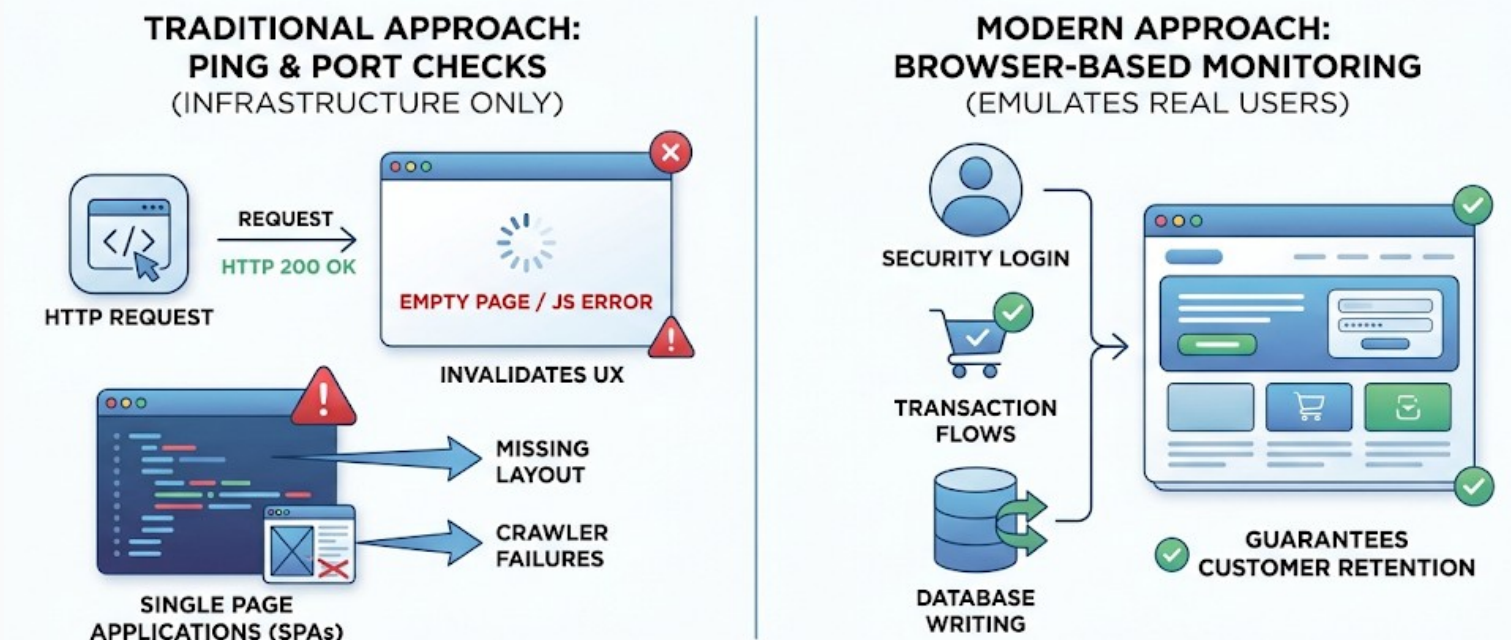
Level 1: HTTP Agent Polling: Lightweight, asynchronous backend API health polling

Level 2: Web Scenarios: Native multi-step sequence execution with cookie tracking

Level 3: Script Items: Secure server-side JavaScript to chain and transform APIs

Level 4: Browser Items: Executing real headless Chrome via W3C WebDriver

Level 5: Visual Automation: *Quadrate's interactive recorder (No-Code)*



LEVEL 1: ASYNCHRONOUS HTTP AGENT

HTTP agents execute lightweight API health checks



High-Performance Backend Polling

- Asynchronous Execution Engine: Handles up to 1,000 concurrent web connections per pre-forked poller, completely agentless
- Persistent Connection Pooling: Leverage Zabbix 7.0 cURL features to optimize keep-alive handshakes, reducing server overhead
- XML & JSON Extractors: Extract telemetry directly in preprocessing using structured **JSONPath** or **XML XPath** rules
- REST API Health Polling: Querying a status endpoint like Elasticsearch “http://localhost:9200/?pretty” returns structured cluster variables

GENERIC REST API PREPROCESSING

```
// HTTP Agent configuration details
Type: HTTP agent
Key: elasticsearch.status
URL: http://localhost:9200/?pretty
RequestMethod: GET

// Master JSON Response returned to Zabbix:
{
  "name" : "node-master-70",
  "cluster_name" : "elasticsearch",
  "version" : {
    "number" : "6.1.3",
    "build_hash" : "af51318"
  },
  "tagline" : "You know, for search"
}

// Preprocessing Step configured on dependent item:
Type: JSONPath
Parameters: $.version.number ==> Output: '6.1.3'
```

Item Tags Preprocessing

Name HTTP agent item

Type HTTP agent

Key http_value_search

Type of information Numeric (unsigned)

URL http://localhost:9200/str/values/_search

Query fields

Request type POST

Request body type Raw data JSON data XML data

Request body

Sequential Web Scenarios preserve cookies across multi-step flows

Session-State Transaction Testing

Multi-Step HTTP Sequences: Chains requests sequentially (e.g. GET login page, POST credentials, check dashboard layout)

Automatic Cookie Preservation: Cookies from the first request are natively saved and transmitted to subsequent steps in the thread (via libcurl)

Flexible Page Redirections: Automatically processes and follows up to ten sequential HTTP redirects

HTML String Validation: Evaluates content presence in realtime (e.g. throwing a trigger error if HTML does not match the regex `Welcome.\*admin`)

Web Scenario Validation Loop

- Step 1: HTTP GET /login**
Extract form headers & load static CSS assets
- Step 2: HTTP POST /auth**
Transmit variables {user} and {password}. Preserve session cookies.
- Step 3: HTTP GET /dashboard**
Verify status code == 200 and search body string `Welcome.\*admin`
- Telemetry Gathered:**
Download Speed, Response Time, Response Code per Step.

ScenarioSteps 5TagsAuthentication

* Steps

Name	Timeout	URL	Required	Status codes	Action
1: First page	15s	http://127.0.0.1/zabbix/index.php	Zabbix SIA	200	Remove
2: Login	15s	http://127.0.0.1/zabbix/index.php		200	Remove
3: Login check	15s	http://127.0.0.1/zabbix/zabbix.php?... Administrati...		200	Remove
4: Logout	15s	http://127.0.0.1/zabbix/index.php		200	Remove
5: Logout check	15s	http://127.0.0.1/zabbix/index.php	Username	200	Remove
Add					

Details of web scenario: Frontend check

Step	Speed	Response time	Response code	Status
First page	31.18 KBps	64.81ms	200	OK
Log in	44.64 KBps	258.36ms	200	OK
Log in Check	74.16 KBps	155.53ms	200	OK
Log out	19.03 KBps	106.17ms	200	OK
Log out check	42.13 KBps	47.96ms	200	OK
TOTAL		632.82ms		OK

Fromnow-30m

To

now

Apply

Last 2 days

Last 7 days

Last 30 days

Last 3 months

Last 6 months

Last 1 year

Last 2 years

Yesterday

Day before yesterday

This day last week

Previous week

Previous month

Previous year

Today

Today so far

This week

This week so far

This month

This month so far

This year

This year so far

Last 5 minutes

Last 15 minutes

Last 30 minutes

Last 1 hour

Last 3 hours

Last 6 hours

Last 12 hours

Last 1 day

Script items execute JavaScript to perform custom API chaining

Programmable API Integration

Secure JavaScript Sandbox: Runs directly in the Zabbix server/proxy engine, using the Duktape JS environment

Dynamic HTTP Client Chaining: Enables complex multi-step data collection where dynamic values from Step 1 are parsed and passed to subsequent requests

User Macro Support: Accesses host and global user macros (e.g. `{ $WEBSITE.DOMAIN }`) securely within the script parameters

API Token Handling: Ideal for authenticating with modern REST APIs that require dynamic token extraction (OAuth2 flow) before querying telemetry

Item

Tags

Preprocessing

* Name

Data collector script

Type

Script

* Key

script.data.collector

Select

Type of information

Text

Parameters

Name	Value	Action
host	{HOST.CONN}	Remove
endpoint	{ \$ENDPOINT }	Remove
Add		

* Script

var request = new HttpRequest();...

OAUTH2 TOKEN EXTRACTION AND CHAINING

```
// Level 3 Script Item: Dynamic OAuth Chaining
var request = new HttpRequest();
var response;
var token;

// Step 1: Request access token from Auth server
response = request.get("https://api.example.com/token");
if (request.getStatus() != 200) {
    throw 'Authentication failed: ' + response;
}
token = JSON.parse(response).access_token;

// Step 2: Query API injecting the dynamic token
request.addHeader('Authorization: Bearer ' + token);
response = request.get("https://api.example.com/data");

return response; // Outputs raw JSON payload
```

LEVEL 4: REAL BROWSER SYNTHETIC TESTING



Browser items simulate actual human-browser interactions

Full Browser Engine Driving

True SPA Rendering: Opens an actual headless Chrome or Firefox instance to render dynamic HTML and client-side JavaScript perfectly

Mimics Real User Behavior: Programmatically clicks buttons, navigates links, enters text fields, and validates form submissions

Visual Verification: Captures and stores Base64 screenshots directly in the database as binary item history, showing what failed visually

W3C WebDriver Protocol: Uses standard JSON wire protocol to drive browsers via a separate Selenium Standalone container or ChromeDriver

Item

Tags

Preprocessing

*

Name

Type

Browser

*

Key

Select

Type of information

Numeric (unsigned)

Parameters

Name

Value

Action

Remove

Add

*

Script

var browser = new Browser(Browser.chromeOptions());...

Core Synthetic Capabilities

Screen Dimension Setting: Render tests at precise viewports like 1920x1080 to check responsive design

Base64 PNG Screenshots: Visual binary data attached directly to test metrics

Performance Timings: Collects navigation and resource load timings for every step natively

W3C Locator Strategies: Interacts with web elements using XPath, CSS Selectors, Class Name, or Link Text

JavaScript templates handle complex login flows and screenshot errors

Script Architecture

W3C Locator strategies: Find DOM elements programmatically using browser engines like CSS selectors or XPath

Input Simulation: Send credentials to name and password input boxes natively

Performance Profiling: `collectPerfEntries()` tracks asset download speed and page weight dynamically

Try-Catch Error Screenshot: Captures and attaches base64 image *only* if elements fail to load, protecting database storage space from bloat

STANDARD ZABBIX 7.0 BROWSER SCRIPT

```
var browser, result;
browser = new Browser(Browser.chromeOptions());
try {
  var params = JSON.parse(value);
  browser.setScreenSize(1920, 1080);
  browser.navigate(params.webURL);

  // Locate DOM inputs & send credentials
  var el = browser.findElement("xpath", "//input[@id='name']");
  el.sendKeys(params.username);

  el = browser.findElement("xpath", "//input[@id='password']");
  el.sendKeys(params.password);

  // Click submit button
  browser.findElement("xpath", "//button[@id='enter']").click();
  browser.collectPerfEntries("dashboard_load");

  result = browser.getResult();
} catch (err) {
  browser.setError(err.message);
  result = browser.getResult();
  result.error.screenshot = browser.getScreenshot();
} finally {
  return JSON.stringify(result);
}
```

Asynchronous pollers orchestrate browser actions over W3C WebDriver

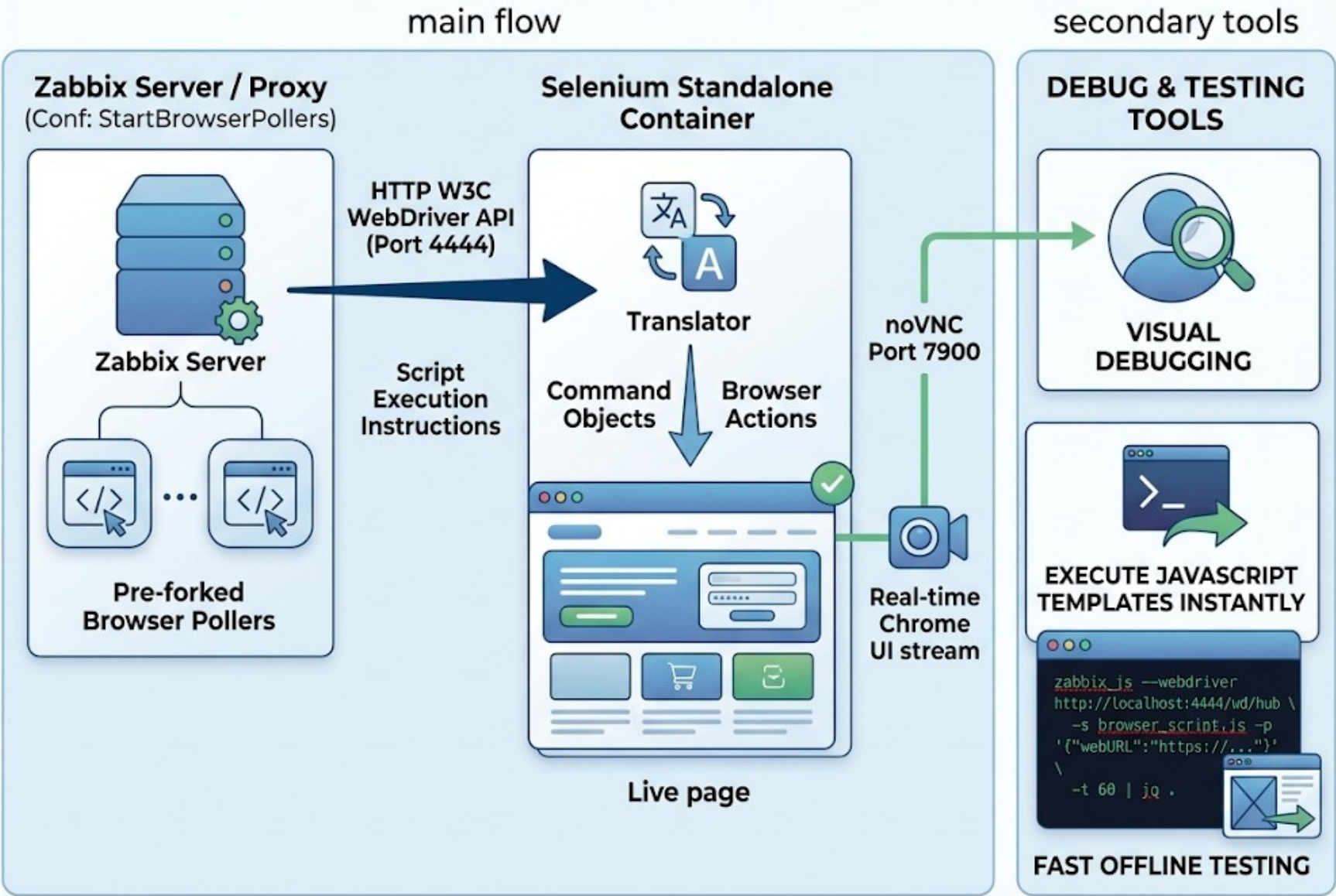
Distributed Orchestration Flow

- Browser Item Pollers:** pre-forked server/proxy processes configured using `StartBrowserPollers` in `zabbix\_server.conf`
- WebDriver Communication:** Server sends instructions to the Selenium Standalone container (port 4444) over W3C HTTP API
- Visual Debug Interface:** Connect to port 7900 on the Selenium container via noVNC to observe active browser execution in real-time
- Fast Offline Testing:** Execute and debug JavaScript templates instantly on the command-line with the `zabbix\_js` tool:

```
zabbix_js --webdriver http://localhost:4444/wd/hub \
-s browser_script.js -p '{"webURL":"https://..."}' \
-t 60 | jq .
```

Environment Topology

- Zabbix Server / Proxy**
Pre-forked Browser Pollers handle script scheduling and parse execution results.
- WebDriver Port 4444**
Selenium Standalone Docker container translates W3C command objects to browser actions.
- noVNC Port 7900**
Stream active Chrome UI rendering in real-time to debug layout issues visually.



Developing custom browser scenarios requires heavy scripting overhead

1. Writing & Debugging Code

Creating custom transaction tests requires highly specialized JavaScript development. Operations teams struggle to maintain complex try-catch blocks and performance profile loops, slowing down template deployment significantly.

2. Brittle Element Selectors

Locating web elements relies on static XPath, CSS classes, or ID paths. Every minor change in a website's frontend layout instantly breaks these selectors, rendering items unsupported and triggering false alerts.

3. Exception & Timeout Friction

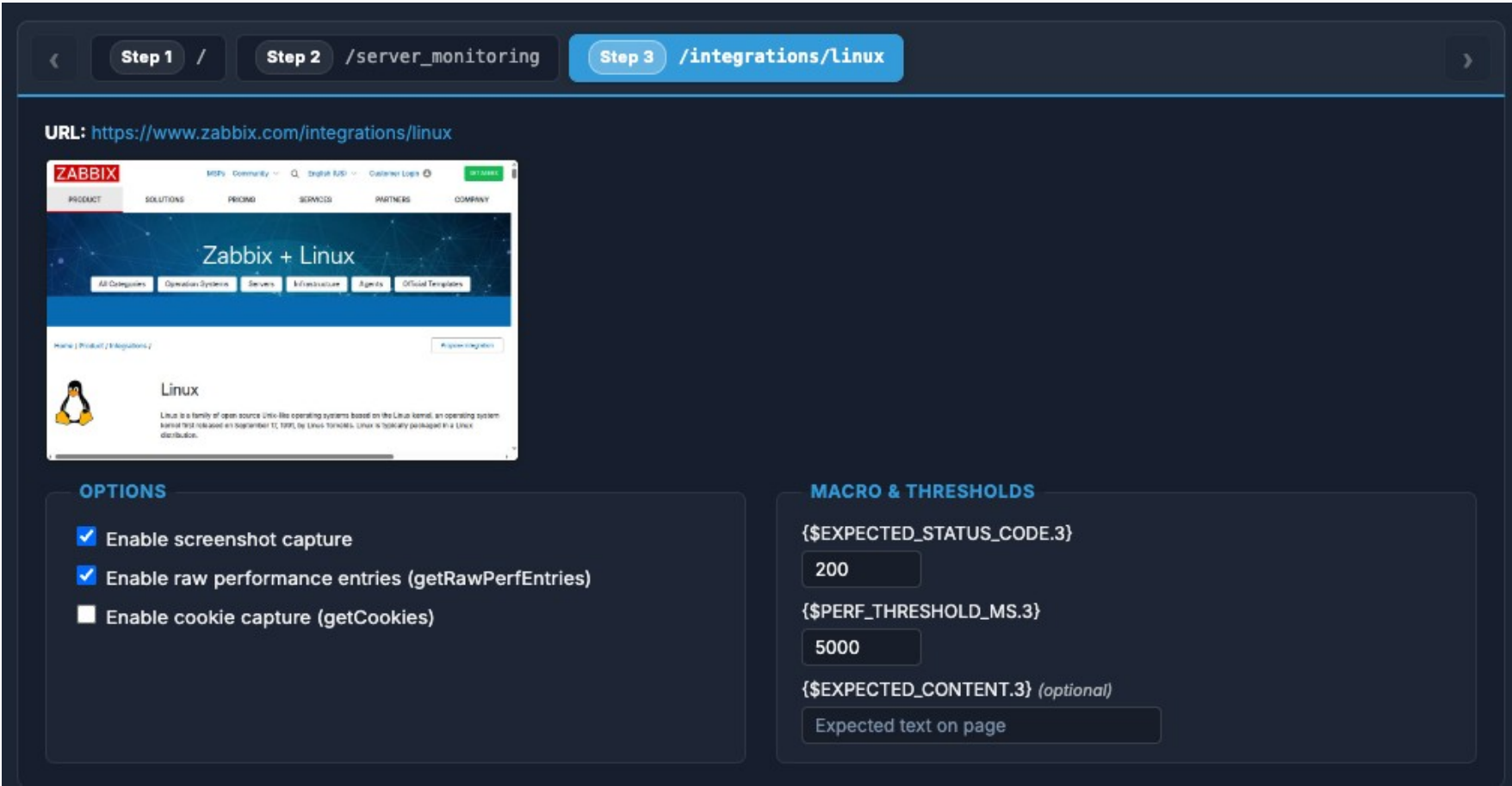
Handling race conditions, slow dynamic elements, and loading states requires complex waiting loops. Debugging timeouts and handling dynamic cookies requires heavy manual logging, creating massive operational overhead.

Visual Recorder Inside Zabbix

- Integrated VNC Interface:** Renders a live visual browser session directly inside the native Zabbix administration interface.
- True User Emulation:** Type target URLs, navigate, click buttons, fill login forms naturally. No development tools required.
- Fully Compliant Zabbix Module:** Integrates natively into Zabbix's extension system. Does not use any proprietary external agents or third-party platforms.
- Accessible to Operations Teams:** Empowers non-developers and system administrators to configure complex synthetic tests in seconds.

Visual Recording Pipeline

1. Start Session ==> Type Target Webpage URL
2. Record Clicks ==> Visual Embedded VNC Browser opens
3. Live Interact ==> Click Buttons, Input credentials, Type text
4. Auto-Generate ==> Clicks and inputs are logged instantly



Visual interactions compile directly into standard Zabbix 7.0 templates

How the Automation Engine Works

Coordinate & Element Tracking: Instantly translates visual clicks into stable XPath and CSS selectors, ignoring dynamic ID noise.

Standard JavaScript Compiler: Writes, optimized JavaScript code fully compliant with Zabbix 7.0's native Browser item objects.

Zabbix Template Exporter: Packages the compiled JavaScript into standard, native Zabbix templates (YAML).

Triggers and Triggers Included: Automatically creates accompanying items, preprocess JSONPath steps, screenshot storage triggers, and dashboard widgets.

GENERATED YAML ZABBIX TEMPLATE

```
// Generated native Zabbix Template extract:
zabbix_export:
  version: '7.0'
  templates:
    - template: 'Dynamic_Checkout_Scenario'
      items:
        - name: 'Synthetic Browser Item'
          type: BROWSER
          key: browser.checkout.test
          script: |
            var browser = new Browser(Browser.chromeOptions());
            try {
              browser.navigate("https://site.com");
              browser.findElement("xpath", "//button").click();
              ...
            } catch (err) { ... }
      triggers:
        - expression: 'last(/browser.checkout.test/error) <> 0'
```

Now it's time for a DEMO...

Key Takeaways

Zabbix 7.0 is a Synthetic Powerhouse: The new **HTTP Agent**, **Script**, and **Browser** items represent an extraordinary leap forward in native web synthetic testing capabilities.

Multi-Layered Strategy: Match your monitoring layers to your architectural needs—polling APIs with Level 1 Agents, and transaction rendering with Level 4 Browser Items.

Quadrata Visual Recorder Speeds Deployment by 10x: Low-code visual recordings completely eliminate manual scripting overhead, democratizing synthetic testing for the whole team.

Get Involved!

Join the Live Demo: Visit the **Quadrata Booth** in the exhibition hall for a live interactive VNC recording demonstration.

Italian Zabbix Community: Join our Telegram group **@ZabbixItalia** to discuss synthetic testing with 1,000+ local engineers.

Questions & Answers: The floor is now open!