Andreas Lehr

WE MANAGE GMBH



NUREMBERG · NOV 17–19 2026

Open Source End-To-End Monitoring

CELEBRATING 20 YEARS OF MONITORING EXCELLENCE!

HELLO, OSMC

Hi, I'm Andy



WE MANAGE

DevOps & SRE as a service. We run web platforms for e-commerce, SaaS and content sites, 24/7 on EU infrastructure, and we get paged when they break.

- 25+ years of Linux and web operations. It all started with Quake servers and LAN parties.
- Newsletter **allesnurgecloud.com** (German).
- English edition coming: **uptime-funk.com**
- Podcast **Happy Bootstrapping**

LinkedIn

FLASHBACK · OSMC 2024

Customer Centric Monitoring

Six pillars, seen from the user. Today we zoom into the outside-in ones.

Watch the 2024 talk

01 · TODAY

Uptime & certificates

02 · TODAY

Page speed & Web Vitals

03 · TODAY

Synthetic browser tests

04

Structured logs & events

05

Error tracking

06

APM

PART 1

What is end-to-end monitoring?

CHANGE THE PERSPECTIVE

From inside out to outside in



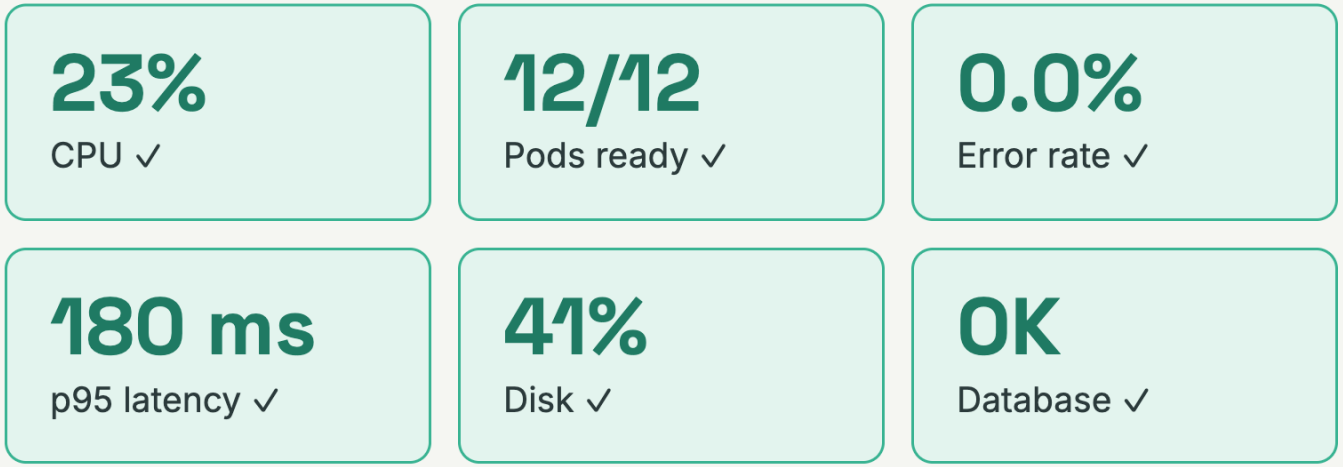
End-to-end checks start here, where the customer is.

Metrics, logs and traces live here. All green does not mean it works.

EVERY ON-CALL ENGINEER KNOWS THIS

All green. And still broken.

Grafana · prod · last 6h



CUSTOMER · 08:14

> **“Our checkout has been broken since 3 a.m. Nobody can pay.”**

DEFINITION

An end-to-end check is a scripted visitor

It shows your platform the way your users see it: from the outside, from where they are.

Is it up?

Status, expected text, certificate

Does the flow work?

Login, checkout, a chain of API calls

How fast does it feel?

Response time per step, Web Vitals

From where?

Several locations: target down or our network down?

Synthetic = lab data on a schedule. It complements real user monitoring, it doesn’t replace it.

WHY

Why you need them

No traffic needed

Find the broken login at 3 a.m., before the first customer does.

Deploys break flows, not pods

Every container healthy, checkout still broken.

Third parties break you

Certificates, CDNs, payment providers, external includes.

Proof from the customer's side

SLAs and status pages based on what users actually get.

THE LADDER

Four levels, from cheap to deep

LEVEL 1

HTTP

Status, text on the page, certificate

LEVEL 2

API

One request, assert the JSON

LEVEL 3

Multi-step API

Token → capture an ID → next request, polling, cleanup

LEVEL 4

Browser

Real Chrome: clicks, steps, Web Vitals

Each level up: more insight per run, more cost per run. That's why they run at different intervals.

LANDSCAPE

Open source tools today

Tool	HTTP	Multi-step API	Browser	Good to know
Uptime Kuma	Yes	–	Page load + screenshot	UI first, huge community
Gatus	Yes	Suites (alpha)	–	YAML, Prometheus metrics
Blackbox exporter	Yes	–	–	The Prometheus classic
OneUptime	Yes	Via script	Playwright script	Full platform, Apache 2.0
Upright (37signals)	Yes	Via Playwright	Playwright probes	Rails engine, Prometheus, MIT
Robot Framework + Robotmk	Yes	Keywords	Browser library (Playwright)	Framework is OSS, Robotmk needs Checkmk paid editions
Grafana SM agent + k6	Yes	MultiHTTP	k6 browser	Agent is OSS, control plane is Grafana Cloud

OUR CONSTRAINTS

We didn’t want another tool. We wanted glue.

Our stack, nothing new

telegraf → VictoriaMetrics → Grafana. No extra SaaS, no second alerting, no new login.

Checks as code

Today in Ansible group_vars, later in the customers’ own Git repos. Renovate keeps versions current.

Several locations

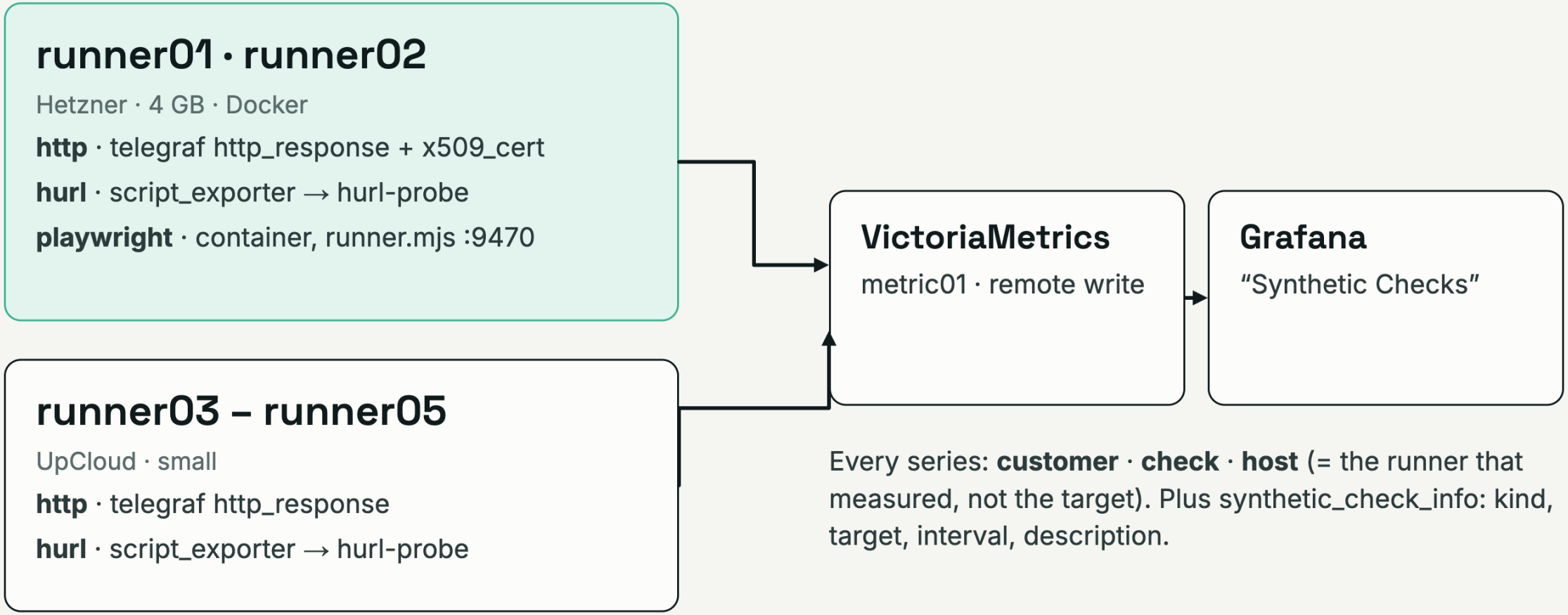
Tell “the target is down” from “our network is down”.

Reuse what devs write

Playwright specs from the dev team run unchanged.

ARCHITECTURE

Five runners, one stack



Every series: **customer** · **check** · **host** (= the runner that measured, not the target). Plus synthetic_check_info: kind, target, interval, description.

PART 2

From simple to complex, in code

WHY LEVEL 1 NEEDS A MATCH

200 OK. Technically.

shop.example.com/black-friday



Sorry, our servers are a bit overwhelmed.

Meet one of our office dogs while you wait.

`HTTP 200 OK`

The status check is happy.
The customer isn't.

Assert the content, not just the status code.

LEVEL 1 · HTTP

Is it up, and does it say the right thing?

group_vars/statuswerk_runner

```yaml
synthetic_checks:
  demo:
    http:
      - name: netways-website
        description: netways.de answers with its claim
        url: https://netways.de/
        status: 200
        match: "Die Open Source Experten"
```

- Rendered into telegraf http_response: status, body match, headers, basic auth
- x509_cert: certificate expiry
- Every minute, from all five runners

LEVEL 2 · API

One request, assert the JSON

hurl · plain text HTTP

```
GET https://statuswerk.eu/api/v1/monitors
Authorization: Bearer {{token}}
HTTP 200
[Asserts]
jsonpath "$.data" count > 0
```

WHAT'S HURL?

A command line tool that runs **HTTP requests written in plain text**. Chains requests, captures values, asserts on headers and body. REST, GraphQL, SOAP or plain HTML.

- One binary, no Node, no spec file
- Runs on all five runners
- Exit code → script_success

LEVEL 3 · MULTI-STEP API

Capture, then chain

```
step 1
GET https://statuswerk.eu/api/v1/monitors
Authorization: Bearer {{token}}
HTTP 200
[Captures]
monitor_id: jsonpath "$.data[0].id"
```

Capture the first monitor's ID

```
step 2
GET https://statuswerk.eu/api/v1/monitors/{{monitor_id}}
Authorization: Bearer {{token}}
HTTP 200
[Asserts]
jsonpath "$.data.id" == {{monitor_id}}
```

Reuse it, and assert we got the same one back

Lives in group_vars as content: !unsafe | so Jinja keeps its hands off hurl's {{ }}.

WHAT WE ADDED: HURL-PROBE

“Flow is red” isn’t enough. Which request?

```
GET /metrics · Prometheus text format · scraped by telegraf → VictoriaMetrics
script_success{script="statuswerk__api-monitors"} 1
synthetic_hurl_step_duration_seconds{index="01",step="GET /api/v1/monitors"} 0.41
synthetic_hurl_step_duration_seconds{index="02",step="GET /api/v1/monitors/{{monitor_id}}"} 0.24
```

INDEX LABEL

Keeps the flow order. Without it, tables and stacked graphs sort alphabetically.

STEP FROM SOURCE

Method and path as written, {{monitor_id}} stays a placeholder. No new series per run.

MISSING STEP

A request without an answer is an error. A missing last step was never reached.

LEVEL 3 · MULTI-STEP WITH LOGIC

When a flow needs logic: Playwright, no browser

statuswerk incident-flow (simplified)

```
test('incident flow', async ({ request }) => { // request only: no browser
  try {
    await test.step('create', async () => { /* POST, notify_subscribers: false */ });
    await test.step('public page shows it', async () => {
      await expect.poll(async () => {
        const r = await request.get(incident.public_url, { headers: { Accept: 'text/html' } });
        return r.ok() && (await r.text()).includes(title);
      }).toBe(true);
    });
    await test.step('update', async () => { /* … */ });
    await test.step('resolve', async () => { /* … */ });
  } finally {
    await test.step('delete', async () => { /* … */ }); // always clean up
  }
});
```

4 s · 0 Chromium processes

own test status page

notify_subscribers: false

every 30 min: API quota

TWO WAYS FOR API CHECKS

hurl or Playwright?

hurl: declarative

- Requests, captures, asserts in one text file
- Waits by retrying one request until asserts pass
- No if/else, no cleanup when a step fails
- One Rust binary, starts instantly
- Runs on all five runners

Playwright: code

- Loops, branches, shared helpers
- expect.poll: wait for any condition
- try/finally: cleanup even on failure
- API and browser in the same test
- Node + container: two runners

Linear chain → hurl.
Logic, waiting across requests, cleanup → Playwright.

LEVEL 4 · BROWSER

A real visitor path, step by step

files/synthetic/demo/netways-website.spec.ts

```ts
import { test, expect } from '../../fixtures';

test('netways.de visitor path', async ({ page }) => {
  await test.step('home', async () => {
    const response = await page.goto('https://netways.de/');
    expect(response?.status()).toBe(200);
    await expect(page.getByRole('heading', { name: /unsere leistungen/i })).toBeVisible();
  });
  await test.step('trainings', async () => {
    await page.getByRole('link', { name: /^Schulungen/ }).filter({ visible: true }).first().click();
    await expect(page).toHaveURL(/\/schulungen\/$/);
  });
  await test.step('icinga training', async () => { /* … */ });
});
```

home 3.53 s | trainings 1.83 s | icinga training 2.31 s real averages, 16 Nov

LEVEL 4 · FOR FREE

Web Vitals for every spec, for free

fixtures.ts

```ts
export const test = base.extend({
  page: async ({ page }, use, testInfo) => {
    const vitals = {};
    await page.exposeBinding('__wmVital', (_s, m) => { (vitals[m.page] ??= {})[m.metric] = m.value; });
    await page.addInitScript({ content: initScript }); // injects Google's web-vitals
    await use(page);
    await testInfo.attach('web-vitals', { body: JSON.stringify(vitals), contentType: 'application/json' });
  },
});
```

LCP FCP TTFB CLS INP TBT if no interaction

The spec author writes a normal test. Because it imports test from our fixtures.ts, every page gets Google's web-vitals injected and the values land in the result. Lab data, not field data.

LEARNING · BROWSER

A cookie banner broke our check

The issue: the consent banner shows only to browsers that don't look headless. Our runner uses the Desktop Chrome profile, so it saw the banner, and the banner covered the page. Timeout after 90 s.

Locally we never saw it. Only the screenshot in the failure trace did.

Traces and screenshots on failure are mandatory. Test locally with the runner's profile.

```
fix: page.addLocatorHandler

// Click the consent banner away, in any step
await page.addLocatorHandler(
  page.getByText('Weiter ohne Einwilligung',
    { exact: true }).first(),
  async (button) => { await button.click(); });
```

Bonus idea: assert the banner *does* show up. Consent is a GDPR requirement, worth a check of its own.

LEVEL 4 · GETTING STARTED

Let AI write the check

RECORD

Playwright codegen

```
npx playwright codegen
--device="Desktop Chrome"
https://osmc.de/
```

Click through the site, get a spec. Same profile as the runner.

AGENTS

Planner → Generator → Healer

```
npx playwright init-agents
--loop=claude
```

Explores the site, writes a test plan, generates specs, repairs broken tests. Built into Playwright since 1.56.

APIS

Keploy · Schemathesis

Keploy (Apache 2.0): tests from recorded traffic or an OpenAPI spec.

Schemathesis (MIT): fuzzes edge cases from your OpenAPI spec.

AI writes the check. Deterministic code runs it.

RULE OF THUMB

01 **Cheap checks: often and everywhere.**

02 **Expensive checks: less often, for depth.**

http every minute × 5 · hurl every 5 min × 5 · browser every 10 min × 2

WHAT WE BUILT

playwright-synthetic-exporter

WHAT IT DOES

- runner.mjs, Node stdlib only: interval per check
- Playwright JSON reporter → /metrics
- Result, steps and Web Vitals as Prometheus metrics
- Trace and screenshot on failure

DESIGN CHOICES

- One run at a time: a browser takes ~1 GB, two kill a 4 GB box
- A scrape never starts a run, it reads the last result
- No retries: the second opinion is another location
- Official Playwright image, digest pinned

OPEN SOURCE · MIT

playwright-synthetic-exporter

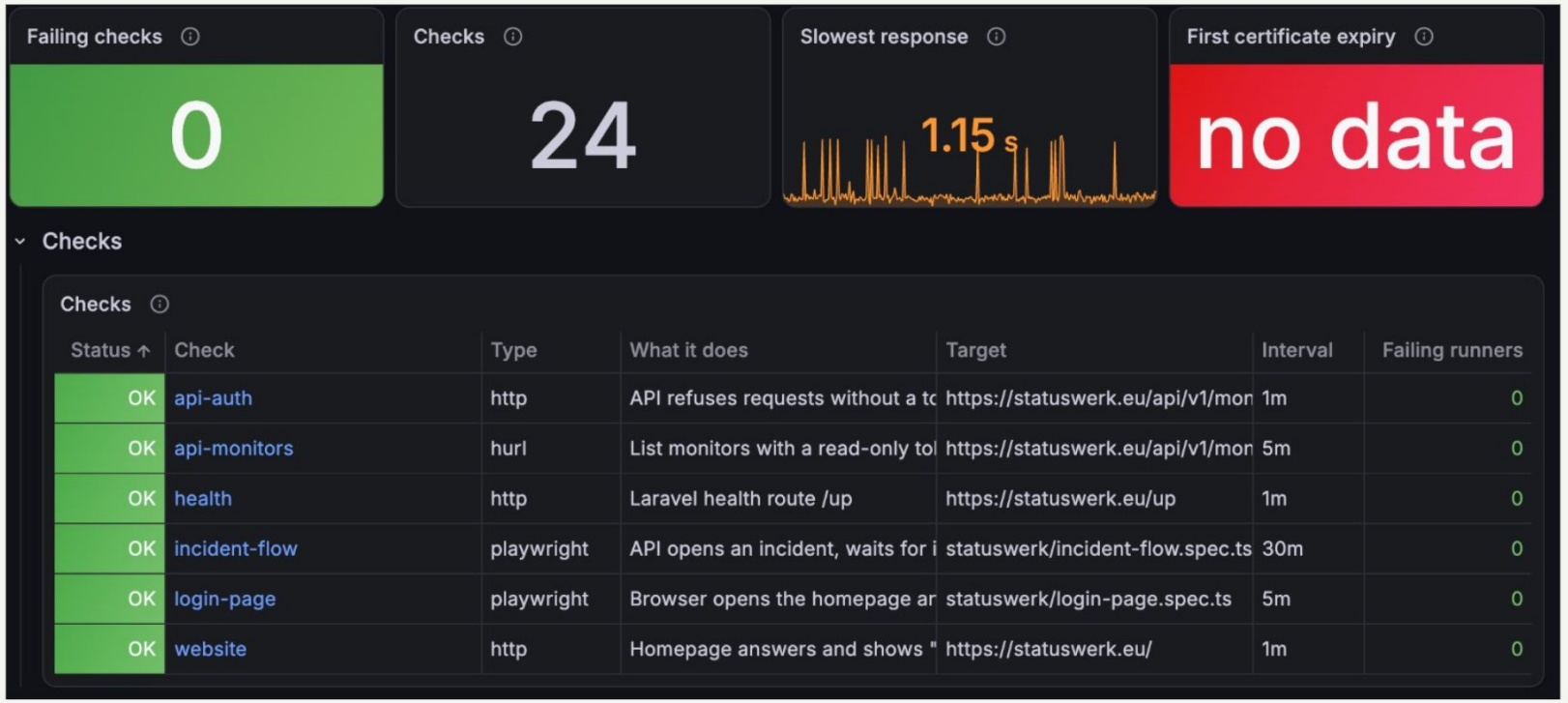
github.com/we-manage/playwright-synthetic-exporter

- Runner, fixtures with Web Vitals, trace on failure
- hurl/: hurl-probe, example, CI
- Docker Compose with Prometheus and Grafana
- Alert rules and dashboard

Next: alert only when ≥2 runners see red · traces & screenshots to S3 · mobile profiles · console errors as metrics

DASHBOARD

One dashboard for all three kinds



Tiles: failing checks, slowest response, next certificate expiry

Check list from synthetic_check_info: what it does, target, interval

DASHBOARD

Every flow, step by step



One row per flow: total per runner, steps of the last run, steps stacked over time. The gap to the dashed total is browser start and hooks.

Web Vitals per page: lab data from the runners, against Google's thresholds.

TAKEAWAYS

01 Measure from outside, from more than one place.

02 Cheap checks often, expensive ones less often.

03 Reuse what you have: your stack, your devs' Playwright specs, plain-text hurl.

Questions?

Andreas Lehr · We Manage GmbH

we-manage.de

uptime-funk.com · allesnurgecloud.com · linkedin.com/in/andreaslehr



LinkedIn