Technical brief · for endpoint engineers

The cost of scripted remediation

Why scripts became the default, the three costs every script carries, the fair objection, and eight questions to take into a design review. For the endpoint engineer who has written remediation scripts and kept them alive.

A scheduled script — spooler-check.ps1

```
# Runs every 5 minutes, on every device
$since = (Get-Date).AddMinutes(-5)
$hit = Get-WinEvent -FilterHashtable @{
  LogName = 'System'; Id = 7031; StartTime = $since
} -ErrorAction SilentlyContinue
if ($hit) { Restart-Service -Name Spooler -Force }
```

Runs every 5 minutes, whether or not anything broke.

A Nanoheal fix — event-triggered

When	Windows raises **Event 7031**
Run	**Print spooler recovery** · approved v3
Then	✓ **Checks the spooler stayed up**

Nothing runs until the device reports the symptom.

Nobody chose scripted remediation for its elegance. They chose it because the platform offered little else: a way to run code on an endpoint, so every fix became a program. The

review cycle, the packaging, the version matrix and the breakage are not lapses in discipline. They are what code costs.

Scripts work. But each one costs you in detection, payload weight and drift, and all three rise with coverage, which is the thing you bought automation for.

01 COST ONE

Detection

You pay to rebuild what the OS already knows.

02 COST TWO

Payload weight

Every script rides along on every endpoint, for good.

03 COST THREE

Drift

Code remembers an estate that has since moved on.

Cost one: detection

Detection is free from the OS. Scripts pay for it again, on every machine.

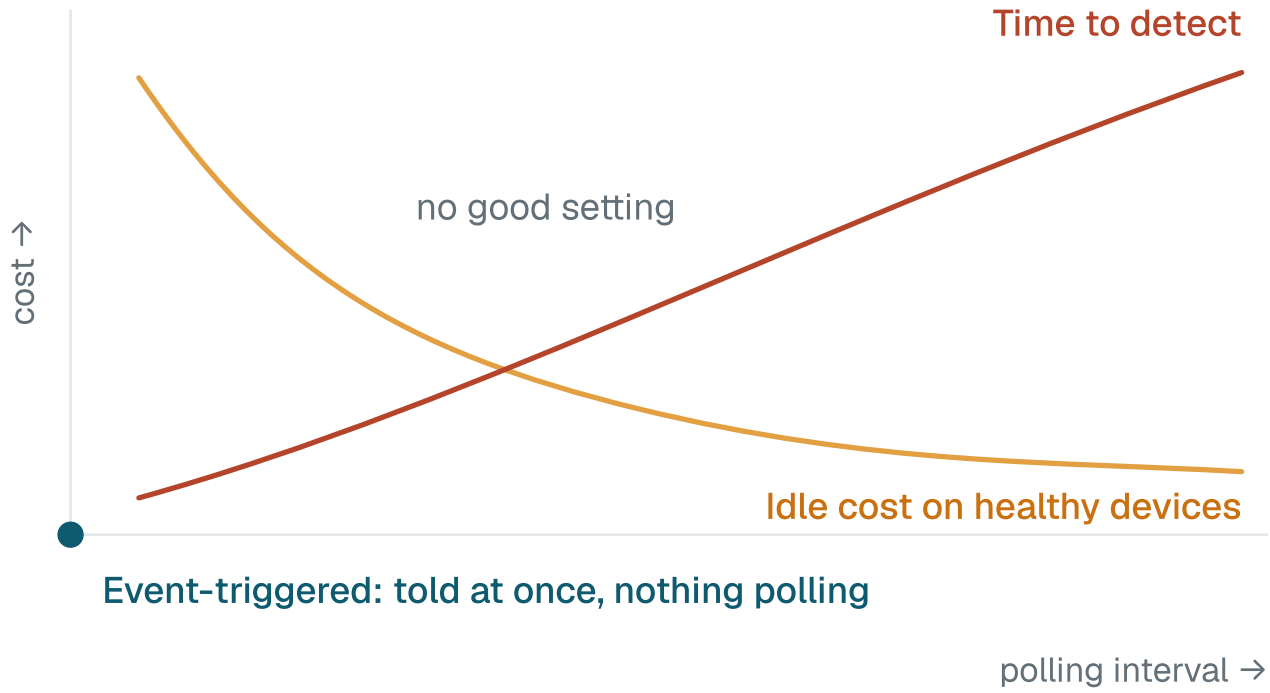
Take a print spooler that has died. Windows already knows: Event 7031 lands in the system log the instant it happens, unasked and free.

A script cannot be told that. Something has to be awake to notice: a scheduled task re-reading the log, a resident process that never exits, an agent sweeping on a timer. Whichever you pick, you have built a second detector for a fact already published, and it runs on every machine for as long as the script exists.

> **A polling interval is a dial with no good setting: fast detection or a light footprint, never both.**

Tight means CPU, memory and battery spent on the healthy majority. Loose means the ticket is open before the automation wakes. Nanoheal never makes that choice: the OS event is the trigger, with no polling, no background probes, and detection latency is milliseconds.

The polling trade-off



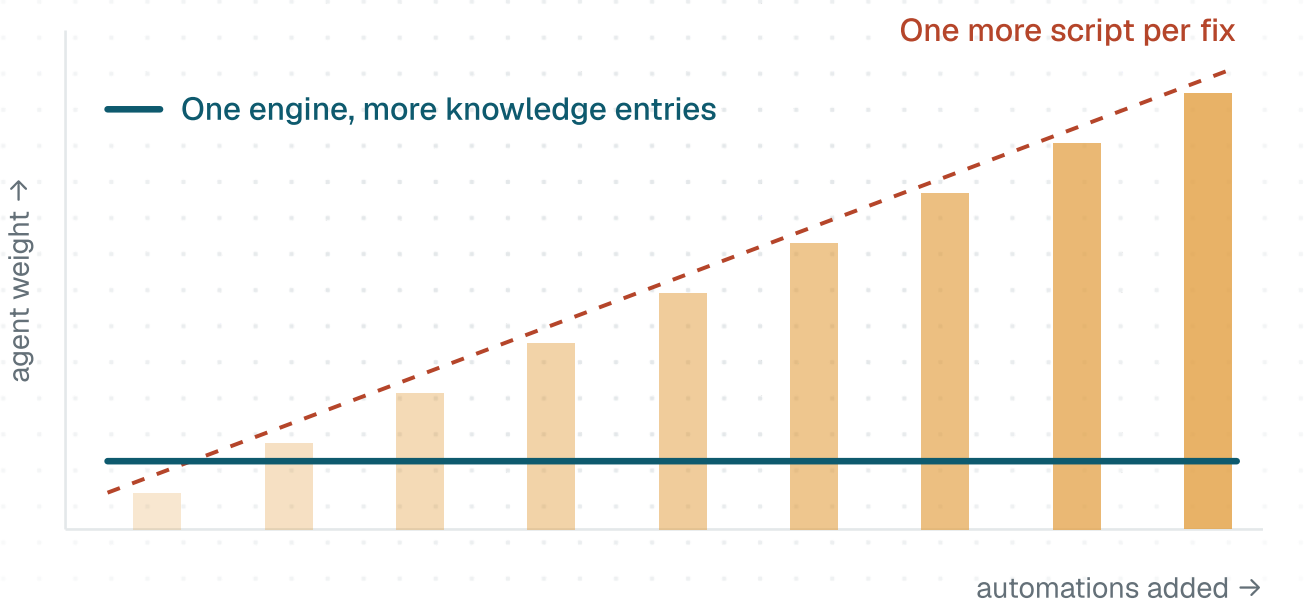
Cost two: payload weight

Every script is a permanent passenger on every endpoint.

One script costs nothing. A few hundred, each carrying its own detection, logging, error handling and schedule, add up to an agent that has quietly become a workload in its own right. VDI feels it first, multiplied by session density. Laptops feel it as battery life.

Most IT leaders have seen the result without seeing the cause: coverage that plateaus, not for lack of ideas, but because nobody wants to add more weight to the fleet. Nanoheal's execution layer is a fixed, signed set of execution capabilities that ships once. The next automation adds a knowledge entry measured in kilobytes, not another program, and the agent footprint stays flat as automations are added.

Weight on every endpoint



Cost three: drift

A script is a snapshot of an estate that has since moved on.

A script freezes the environment as it stood when it was written: this path, this key, this service name, this build. Each is a dependency on something outside your control. A feature update moves the path. A GPO change withdraws the permission. A vendor renames the service.

And the failure is quiet: the tickets drift back, and weeks later a review finds a regression nobody can explain.

Keeping *what* apart from *how* is the defence. The knowledge entry says “repair the spooler’s queue state”, and nothing more. The engine owns how that is done on each platform and build, and is updated centrally as one component instead of hundreds of scripts. And because each run checks its own result, a capability that stops working surfaces as a failed check, not as tickets creeping back.

A script encodes how

```
Stop-Service Spooler
Remove-Item 'C:\Windows\System32\spool\PRINTERS\*'
Set-ItemProperty 'HKLM:\...\Print' ...
Start-Service Spooler
```

A feature update, a GPO change or a renamed service, and it fails quietly.

A Nanoheal fix states what

Repair the spooler’s queue state.

✓ Today’s build

✓ After the next feature update

✓ Next year’s build

Every run checks its own result.

Questions to ask

One dead spooler. Eight questions to ask of any fix.

Ask of any way you automate a fix

How does it find out something broke?

Nanoheal

The OS raises the event, and that event is the trigger. Nothing polls.

What does it cost when nothing is wrong?

Nanoheal

Nothing. No CPU, memory or battery is spent waiting.

What ships to the device for each fix?

Nanoheal

A knowledge entry, measured in kilobytes, not executable code.

Does the agent grow as you add fixes?

Nanoheal

No. It stays the same size at fix 200 as at fix 1.

How long until the first fix runs?

Nanoheal

Day one, from the automation library.

How do you know it worked?

Nanoheal

Each run checks that the symptom is gone.

How are changes approved?

Nanoheal

Staged, approved and versioned publishing.

Who can build one?

Nanoheal

Anyone in IT, in the visual automation builder. Knowledge, not code.

The fair objection

A fixed set of actions can’t do everything. That is the trade, and it is a good one.

The set covers what endpoint remediation actually consists of, across the OS, software, network and policy. The long tail of real tickets is the same capabilities with different settings, rarely new computation.

The trade is explicit. You give up running arbitrary code on the endpoint. In return, nobody has to keep arbitrary code alive on the endpoint.

The ceiling on coverage is a price, not a limit.
Change the price.

---

nanoheal

The operating system for the digital workplace. See it on your own tickets: nanoheal.com/demo/