Can this monitor ever return PASS? The absence test for alerts you already trust

A twenty-minute test you run on five of your own alerts. Each one gets a verdict — CAN PASS or CANNOT PASS — and at least one of them will not survive the fourth question.

TOOL · FRANKLIN J BRYANT IV · 2026-11-10 · 9 MIN READ

The short answer. An alert is only a control if there is a reachable state of the world that makes it fire. Most alerts are never tested against that condition, so nobody knows whether the green light is information or decoration. This is the test, and it takes twenty minutes if you actually do it.

Who this is for

The person who owns the dashboards and gets asked, in a meeting, whether the system is healthy. Not the person who built them. You do not need infrastructure access to run this, and you do not need to write any code.

It is not for teams with no monitoring at all. Those teams have a different problem and this test will not help them yet.

The reframe

The usual question about monitoring is *is it working?* That question cannot be answered by looking at a green light, because a green light is exactly what a broken monitor shows.

The question that can be answered is narrower and much less comfortable: **can this alert ever return FAIL?**

Read that literally. For the alert to fire, some condition has to occur. If you cannot name the condition, or if you name it and then discover nothing in the system can produce it, the alert cannot fail. It will report green forever, in the language of success, and it will do that whether the system is healthy or dead.

This is the failure mode that costs the most, because it does not degrade — it inverts. The worse the system gets, the more confidently it reports. We have run this on our own estate and found five in one reporting cycle. Two of them are worked through below.

The test

Print it. Work it on paper. A test you can only take on a screen is a test you can click through without reading.

Step 1 — pick five

Not the five you are proud of. The five you would be embarrassed to explain out loud to the person who signs the invoice.

1. One on a money path — invoicing, payments, order intake, lead capture. Anything where a silent stop costs cash before a human notices.
2. One client-facing uptime check. The one a client could test before you could.
3. One backup or restore check. This is the category with the most unexamined green in it.

4. One you would be embarrassed to explain. You already know which one. It is usually the oldest.
5. One anything. If nothing comes to mind, use the oldest alert you still run — age is a good proxy for written once and never re-checked.

Write the five names down and stop. Do not evaluate yet. Evaluating early is how people talk themselves out of the finding.

Step 2 — write the fire condition

For each alert, one sentence: the exact state of the world that makes it fire.

The word *exact* is doing the work.

> Fires when `GET /health` returns a non-200 for three consecutive probes from outside the network.

That is a fire condition. This is not:

> Alerts if the API is down.

"Down" is a conclusion, not an observation. You cannot test a conclusion.

Step 3 — interrogate it

Five questions per alert. Write the answers down; the ones you cannot answer are the finding.

#	Question	What a bad answer looks like
1	What writes the thing this alert reads — and does it write before or after the work happens?	"The job logs that it started" — a line written before the work is evidence of intent, not of completion
2	If that writer died silently right now, would this alert still pass?	"I think so" — then it can pass while dead, and it is not a probe
3	Is this reading the thing itself, or a proxy for the thing?	A log's modification time is a proxy for freshness. An exit code is a proxy for correctness. A process name is a proxy for a service
4	Has it ever fired? If never — is that because nothing has broken, or because it cannot?	"It has never gone off, so we must be solid" is the sentence that precedes every bad month
5	What is the last *positive* evidence that this check works, with a date?	If the answer is "we tested it when we set it up," the check is unverified and you are trusting its author's optimism

Question 2 is the one that decides most cases. Question 5 is the one that stings.

Step 4 — the verdict

Fill this in. One row per alert.

#	Alert	Fire condition (Step 2)	Ever fired? When?	Last proof it works (date)	Verdict
1					
2					
3					
4					
5					

CAN PASS — you named a reachable condition, and you can point at a time it fired, or at a deliberate test you ran.

CANNOT PASS — no reachable condition produces a failure signal. The alert cannot tell a working system from a dead one.

Scoring is not a percentage. One CANNOT PASS is a liability, not a control, and it is sitting in front of somebody who is making decisions from it.

Step 5 — the two we failed

The monitor that passed for 28 days. A sync job pushed documents into a search index; a canary checked that the sync was fresh. The index stopped receiving documents on 9 October. The canary printed `PASS` on every run until 6 November. Four defects had to line up and they did: the sync wrote its log line *before* contacting the server, so failure produced the same artifact as success; the canary judged freshness by the log's modification time; the shell health check was `if [ $? -ne 0 ]` applied to an assignment, which returns 0 on success and on failure, so the check could not fail; and one probe treated a Cloudflare login page as healthy because it returned HTTP 302. Four proxies, all satisfied, none of them the fact.

The socket that was not a service. A docs server held port 9623 for 52 days with every HTTP probe timing out. A listening socket is not a service. A process name in a process list is not a listener answering requests. In the same window, an internal task board had been failing on a bad command since 2 November and had raised nothing at all — a control that fails silently is worse than no control, because its absence is mistaken for calm.

What to do with a CANNOT PASS

1. **Delete it or fix it, today.** An alert nobody trusts trains everybody to ignore the channel. That cost is already being paid.
2. **Rewrite it against the fact, not the proxy.** Not "did the job log a line" but "does a known document come back when I search for it." A retrieval either returns the document or it does not. A count returns a number, and numbers can be lied to by ordering.
3. **Write the expected result before you run the test.** If you do not know what failure looks like in advance, you will rationalise whatever you see.
4. **Test it by breaking the thing on purpose**, in a window you choose, with someone watching. An untested check is a hypothesis.

What this page does not do

It does not tell you what to monitor, and it will not find a monitor you forgot you had — you have to bring the list. It says nothing about coverage: five alerts that all pass is not proof you are covered, it is proof you tested five alerts. And it is a self-test, which means the honest answers are yours to give.

If you run it and two of your five come back CANNOT PASS, that is normal. Ours did.

The Negative Probe Audit is this test run across your whole estate — every monitor, dashboard and alert, each one given a verdict and a replacement probe specified where the current one cannot fail. Seven to ten days, and it ends with the list of which greens are real.

IF YOU WANT THIS BUILT

We do this for a living: the audit, the build, and the person who answers when something breaks. The next step is a short call and it costs nothing.

prospyr305.com/book-with-franklin | (561) 990-2428 | scan for this page

Prospyr 305 · a subsidiary of Prospyr Inc · West Palm Beach, Florida. Free to read, print and share. General information, not advice for your specific situation.