Understanding a Test Run and Its Results in Functionize Studio

Studio Basics · https://docs.functionize.com/hc/en-us/articles/43766894416535-Understanding-a-Test-Run-and-Its-Results-in-Functionize-Studio

A test run in Functionize Studio produces more than a pass or a fail. Every step keeps a screenshot of the page as it was, the values it read, and a comparison against the last time the test succeeded. This article explains how to read all of that, and how to run many tests together with an orchestration.

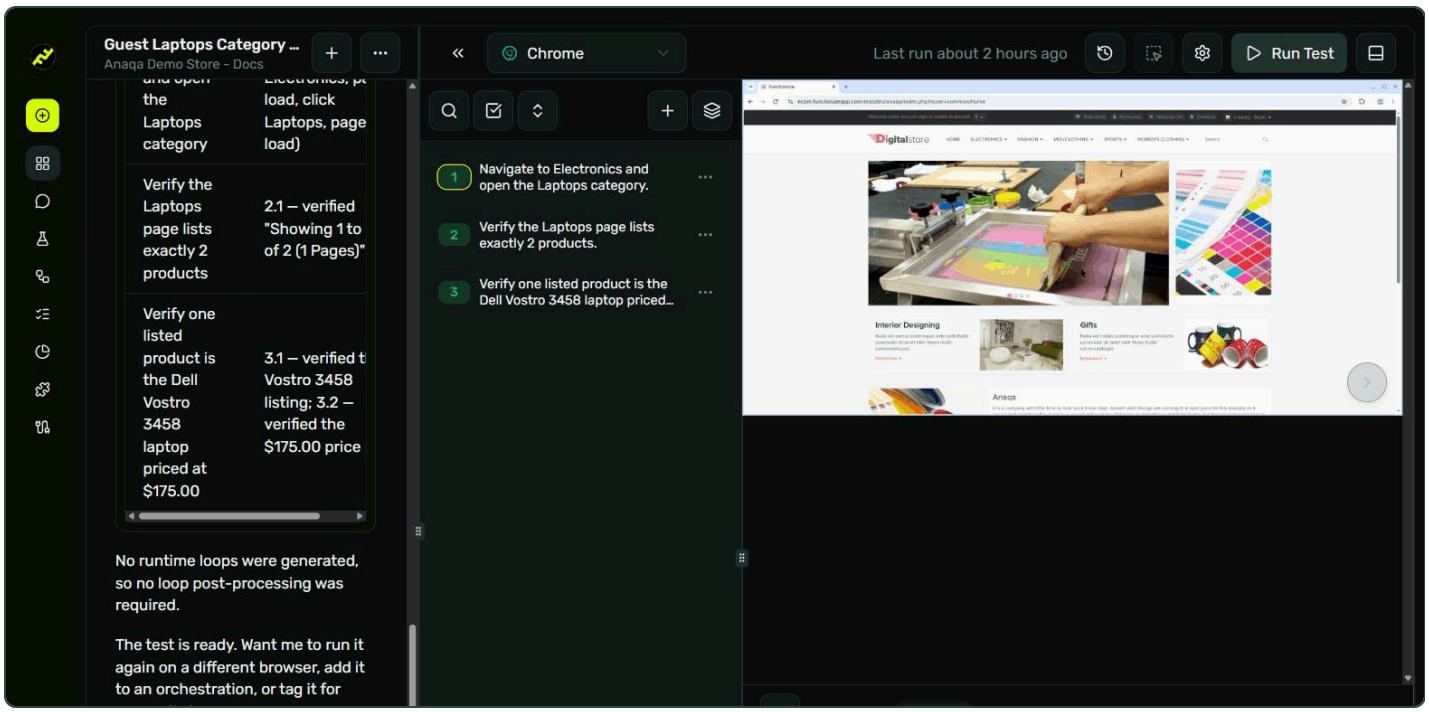
Where a run happens

Runs do not execute in your browser. Studio boots a clean runtime host in the cloud, opens the browser you chose, and drives the test there. You can close the tab and come back — the run carries on and the results wait for you.

Each run also starts from a clean state: no leftover session, no cached basket, no cookie from the last run unless the test creates one itself.

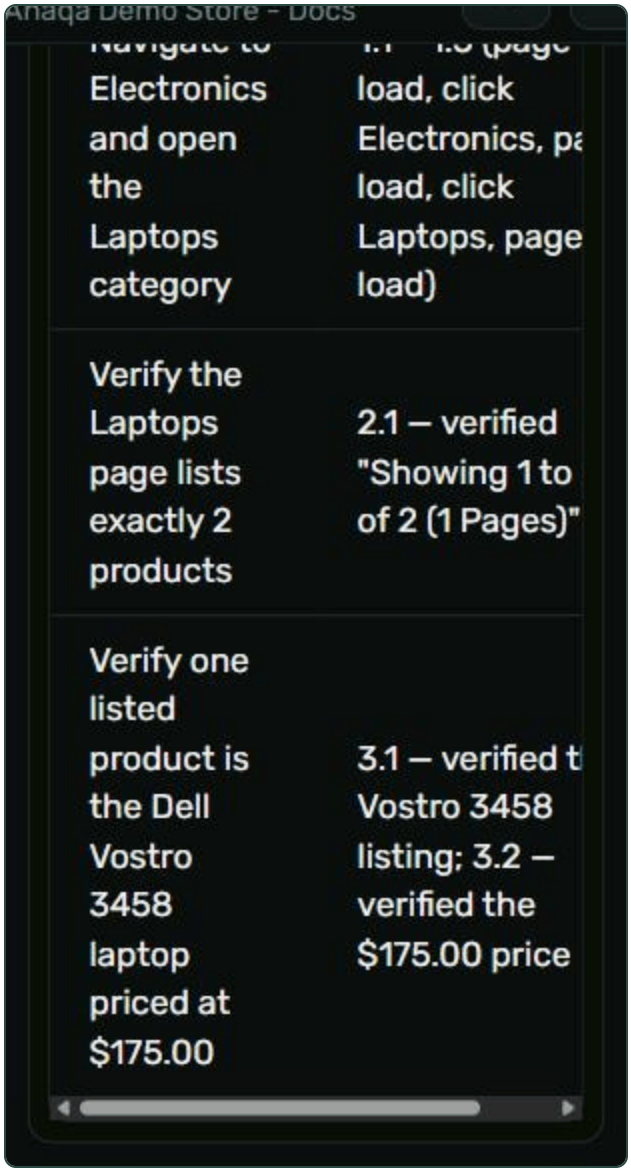
Reading the result

When a run finishes, the editor is your report:



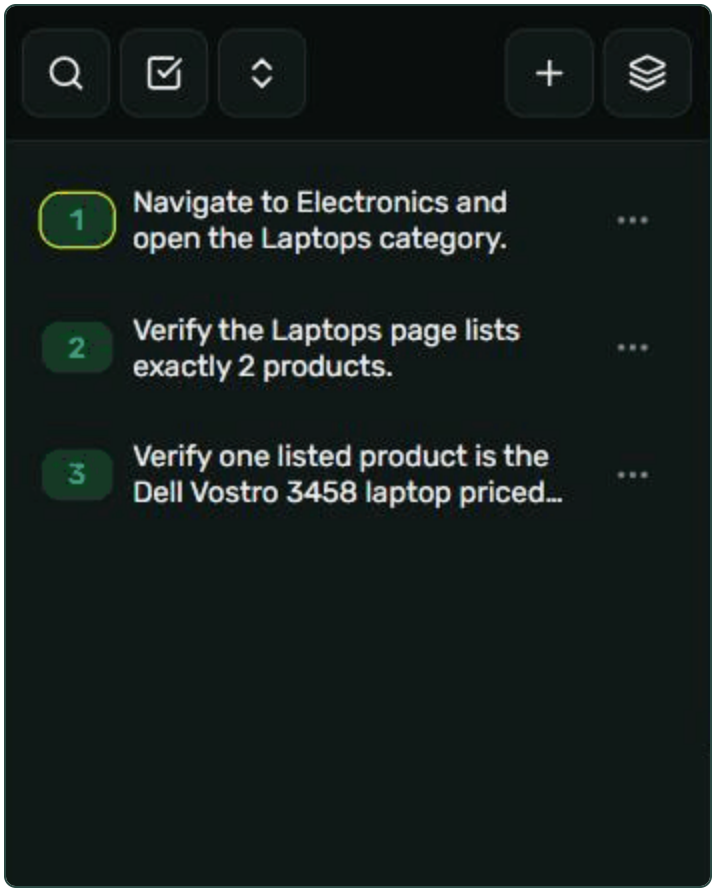
Three panes, three jobs — the agent and its summary on the left, the steps in the middle, the page and its data on the right.

The summary is worth reading properly:



Take two things from it every time: **how many steps passed out of how many**, and **which browser and version** it ran on. "It passes for me" and "it fails on the schedule" are very often the same test on two different browser builds.

Understanding the numbering

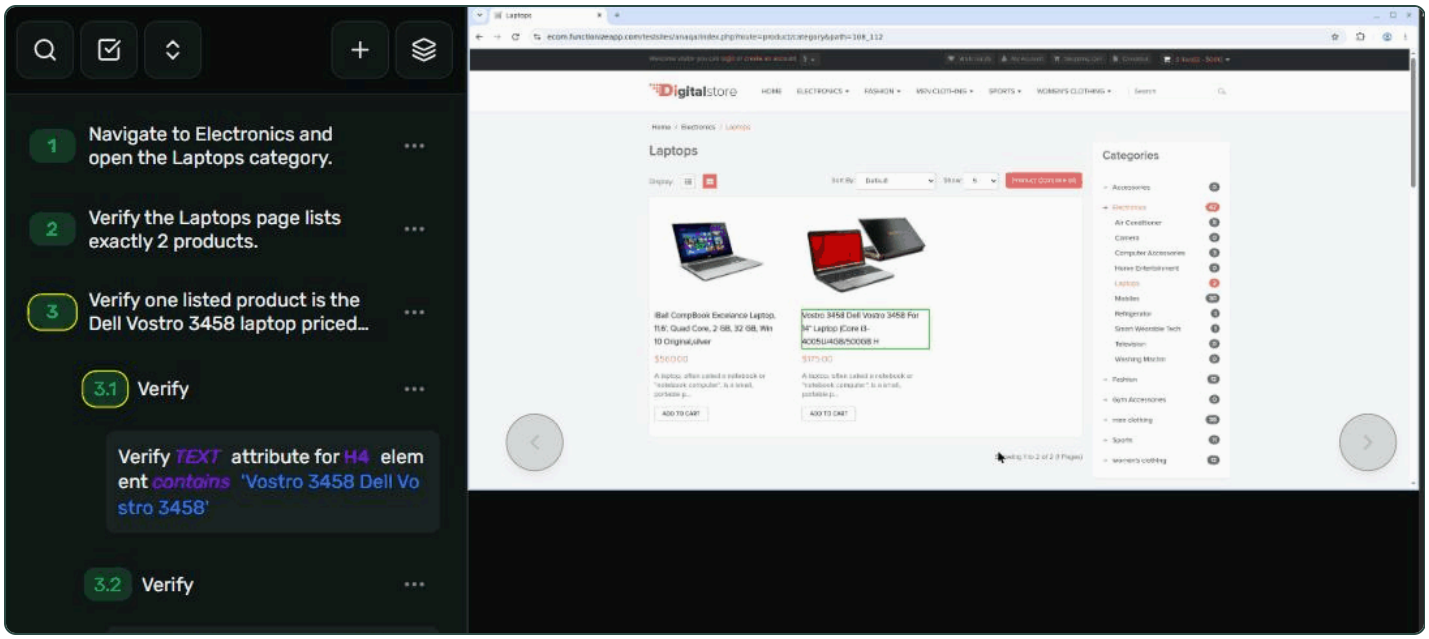


- **Whole numbers** (1, 2, 3) are the instructions *you* wrote.
- **Decimals** (1.1, 1.2 ...) are the actions *Studio generated* to carry them out.

Five written instructions became ten actions in the test above. When something fails, the decimal tells you which action broke and the whole number tells you which of your intentions it belonged to. That pair is most of a diagnosis.

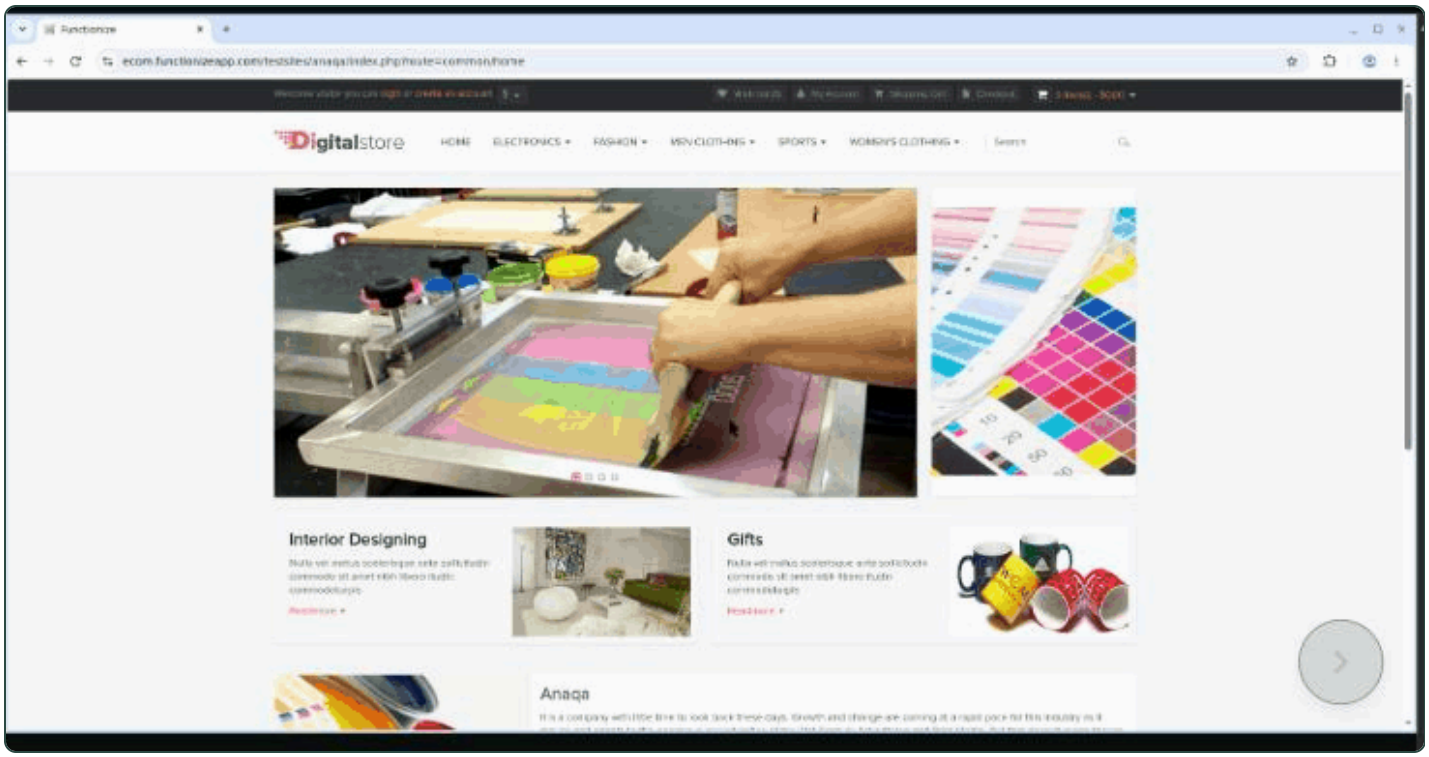
Opening a single step

Click a step to expand it and see the action it performed:



The action describes its target rather than naming a selector path, which is why the step survives the page being restructured around it.

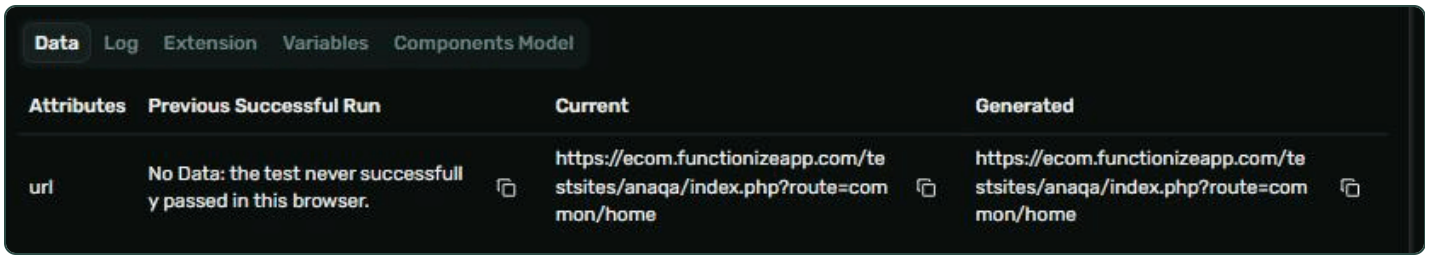
The right-hand pane shows the page at that exact moment:



This is a real screenshot from the run, not a reconstruction. If you ever doubt that a step did what you think it did, this settles it.

The Data tab is the one to learn

Underneath the screenshot, **Data** lists the attributes the step read, in three columns:



Previous Successful Run	What this value was the last time the test passed.
Current	What it is in this run.
Generated	What it was when the test was first created.

Reading those three together answers most failures:

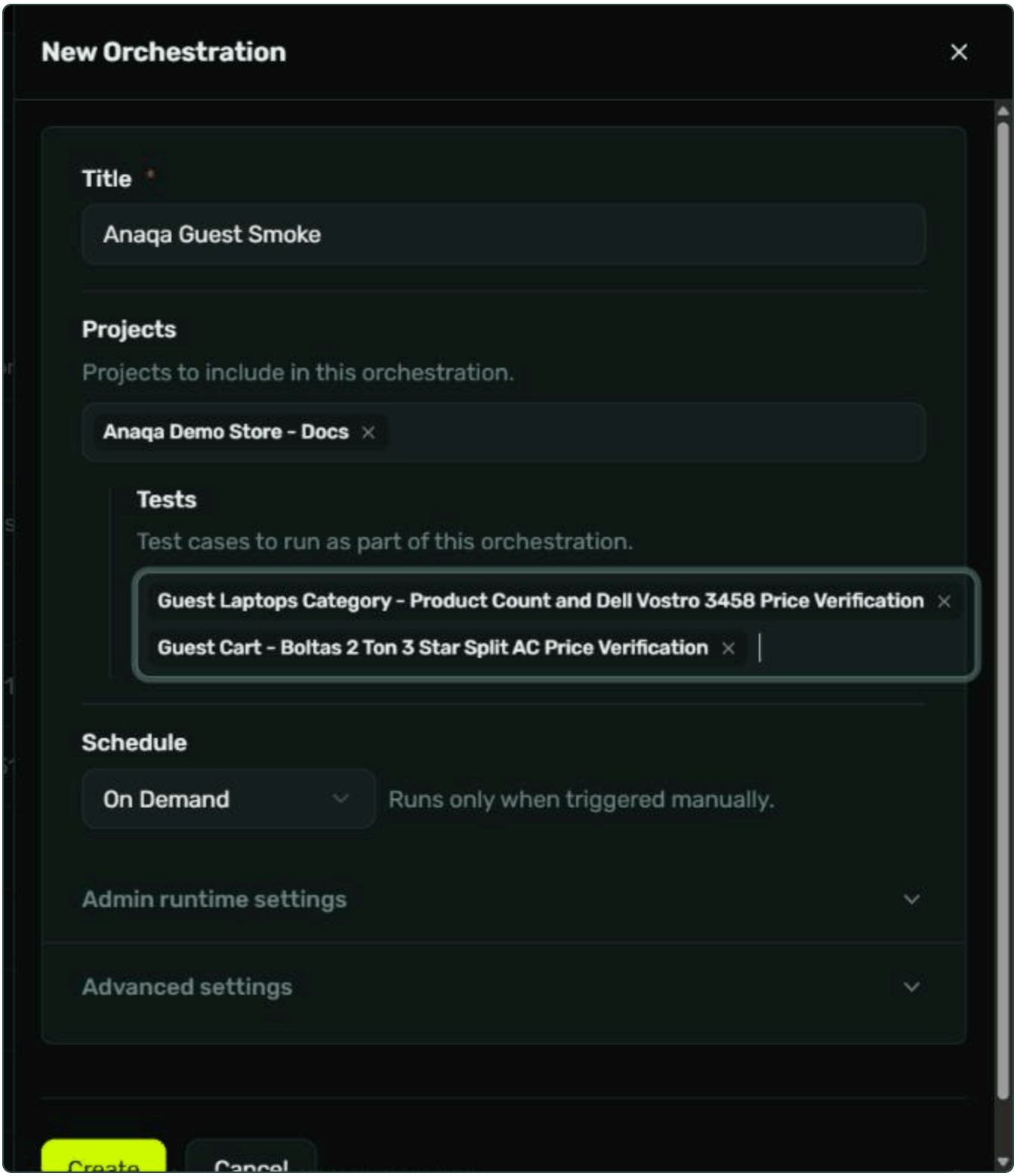
- **Current differs from Previous Successful Run** — your application changed, and the difference is the change.
- **They match but the step still failed** — the application is fine and the expectation moved.
- **Previous Successful Run says there is no data** — this test has never passed in this browser. Worth knowing before hunting a regression that never existed.

The remaining tabs are **Log** (the raw execution log for that step), **Extension**, **Variables** (values carried between steps) and **Components Model**.

Running many tests together: orchestrations

An orchestration is a named group of tests that run as one job, on demand or on a schedule. It is how a handful of tests becomes a regression suite.

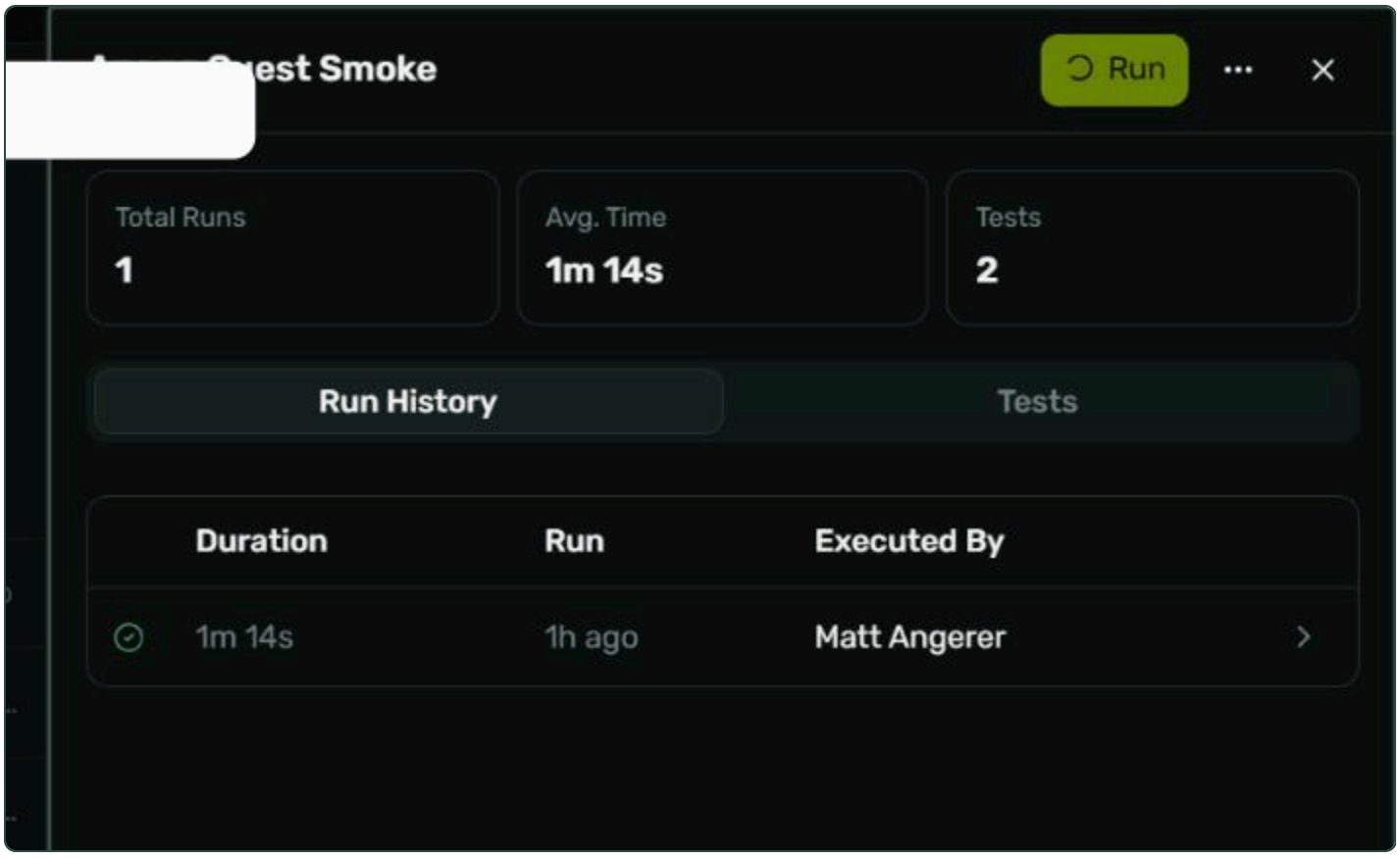
From **Orchestrations** in the sidebar, choose **New Orchestration**:



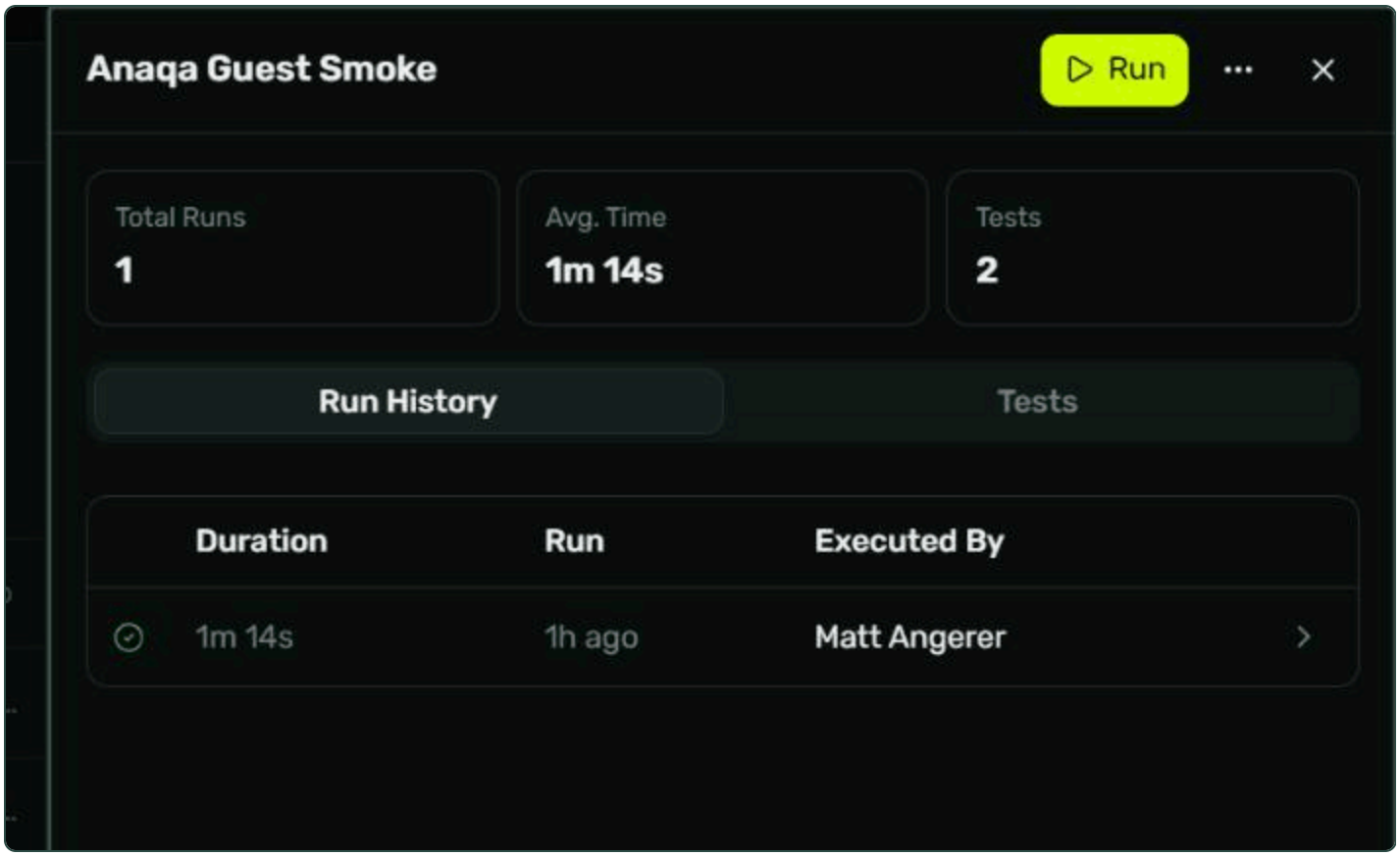
Four things to fill in:

- **Title.** Name it after what it proves, not when it runs — "Guest smoke" ages better than "Nightly 2am", which becomes a lie the first time the schedule changes.
- **Projects.** Pick the project first. The **Tests** field stays locked until you do, and will tell you so.
- **Tests.** Add the tests to include. They run as one job.
- **Schedule. On Demand**, **Hourly**, **Every 4 Hours**, **Every 12 Hours**, **Daily** or **Monthly**.

Click **Run** to start it immediately; the button becomes **Stop** while the job is in flight:



When it finishes, the summary fills in and the run appears in the history with its result, duration and who triggered it:



The two tests above ran as one job in 1 minute 14 seconds. A history entry covers the whole job, and you can still open any individual test to see its steps and evidence.

If a run you have just triggered does not appear in the history straight away, reload the page.

Choosing a schedule

Two things worth saying plainly. Frequent schedules cost runtime, so an hourly suite of fifty tests is a real commitment — start daily and tighten it only if the feedback is genuinely too slow. And a scheduled suite nobody reads is worse than no suite, because it manufactures confidence. Decide who looks at the result before you set the schedule, not after.

When a step fails

A failed step is red, and the three places that explain it are the ones above: the step's screenshot, the **Data** comparison, and the agent on the left, which has the run in front of it and can be asked directly.

For a worked example, see Diagnosing a Failed Test in Functionize Studio.

The Log and Extension tabs

Two of the five footer tabs get overlooked, and one of them answers a question people ask constantly.

Log — how confident Studio was

Select an action and open **Log**. It shows the execution method Studio used, whether it succeeded, and — the useful part — the **evaluator scores** behind that decision:

```
executionMethodName: pageInitMethod
successful: true
evaluators scores: evaluatorName: empty, score: 100
```

Those scores are Studio telling you how strongly the element it acted on matched what it was looking for. That is exactly the evidence you want when the question is "*did self-healing pick the right thing?*" — a step that passed on a weak match is worth looking at before it becomes a false green.

> **Where this matters most**
> Reviewing healing events. Self-Healing and Timing makes the point that a heal which binds to the wrong element turns a real defect into a passing test. The Log tab is where you check.

Extension — what your own code returned

If the selected action called an extension, this tab shows the response. If it did not, it says *No extension response for this action*, which is itself useful: it tells you the extension was not reached rather than that it returned nothing. See Extensions: Running Your Own Code Inside a Test.