What a cloud runner is

This page answers: *when you push to a Tiny Tapeout repository, where do the checks actually run, what kind of machine is it, how long does it take, and what does it cost?*

The green ticks and red crosses next to your commits come from programs that GitHub runs for you, on machines you never see. Those machines are called **runners**, and a few facts about them explain a lot of what you'll see in the logs: why every job starts by installing tools, why jobs hand files to each other the way they do, why a push takes minutes to come back, and why a check can pass on your laptop and fail on GitHub.

A new machine for every job

GitHub's documentation defines runners as "the machines that execute jobs in a GitHub Actions workflow", and says that each GitHub-hosted runner "is a new virtual machine (VM) hosted by GitHub" (GitHub-hosted runners). So every job in every workflow gets a new machine that has never seen your repository. The machine runs the job's steps from top to bottom, and GitHub throws it away when the job ends.

You can see this at the top of every job's log. Here are the first lines of the `viewer` job in a `protocol-emulator` run on Nov 15, 2026 (run 43757724629), with the timestamps taken off the front of each line:

```
Current runner version: '2.338.2'
##[group]Runner Image Provisioner
Hosted Compute Agent
Version: 20261103.629
Commit: 15dbd9ccb5dbf34fae002b6ed1a2670f52df37bc
Build Date: 2026-11-03T17:26:59Z
Worker ID: {69e406aa-39b1-443e-812e-16fa60d2a9fb}
Region: centralus
Cloud: Azure
##[endgroup]
##[group]Operating System
Ubuntu
24.04.5
LTS
##[endgroup]
##[group]Runner Image
Image: ubuntu-24.04
Version: 20261111.370.1
```

So that job ran on an Ubuntu 24.04 virtual machine in a Microsoft Azure data centre, built from build `20261111.370.1` of GitHub's `ubuntu-24.04` image. The `##[group]` lines are folds in the log viewer and not part of the output. GitHub rebuilds the image from time to time, and two jobs from the same afternoon can land on different builds of it: the `test` job of a `hart-1` run on the same day reports `Version: 20261111.127.1`.

Three consequences follow from a fresh machine per job, and you'll meet all of them in the Tiny Tapeout workflows:

- **Every job installs its own tools.** The `test` job installs Icarus Verilog and cocotb, the `gds` job installs LibreLane, and the `precheck` job installs the PDK and builds Magic and KLayout. Nothing is left over from your last push.
- **Jobs pass files to each other as artifacts.** The `gds` job finishes by uploading its results as an artifact called `tt_submission`, and the `precheck` and `viewer` jobs start by downloading it, because they run on other machines and can't see the `gds` job's disk.

- **The machine has only what's in your repository.** A file you forgot to commit, a tool you installed on your laptop, or a setting in your shell doesn't exist there. When a check passes for you and fails on the runner, this is the first place to look.

How big it is and how long it takes

A standard Linux runner for a public repository has 4 CPUs, 16 GB of RAM and a 14 GB SSD. For a private repository it has 2 CPUs and 8 GB, with the same 14 GB of disk (runner reference). That's about what a decent laptop has.

These are the times recorded in the team's own runs:

Job	Time	Where it was measured
`test` workflow	20 to 46 s	The team's repositories
`gds`	207 to 230 s	`sentry` (2x2 tiles, SKY130)
`gds`	510 to 573 s	`ttsky-verilog-tinyCGRA` (1x1 tile, SKY130)
`gds`	791 to 1,056 s	`protocol-emulator` (6x2 tiles, IHP)
`precheck`	91 to 315 s	The team's repositories
`gl_test`	35 to 62 s	The team's repositories
`viewer`	6 to 25 s	The team's repositories

Tiny Tapeout's FAQ says that "a typical Tiny Tapeout GDS action often completes in about 5 minutes for a small design", and that "20-25 minutes or more" suggests the design "may be running out of available tile area". Because `precheck`, `gl_test` and `viewer` all wait for `gds`, a push to a digital repository takes somewhere between 5 and 20 minutes to come back fully green. That's long enough that you should plan around it: push, then read the next page or work on something else while you wait, rather than watching the progress bar.

How long a job ran before it failed is a clue to the cause. Very fast red usually means a setup or configuration problem, and slow red usually means a problem in the design itself. `hart-1`'s `gds` job, for example, fails in 54 seconds, because the build stops at the first step that reads `info.yaml`, long before LibreLane starts.

What it costs

"Use of the standard GitHub-hosted runners is free and unlimited on public repositories" (runner reference). That's the main reason this course has you make your repository public.

Private repositories draw on an allowance. On the Free plan it's stated as $12 a month, which is the same as the 2,000 minutes a month that GitHub's pricing page still quotes, at the rate of $0.006 a minute for the smallest Linux runner. Usage is rounded up to the whole minute for each job (Actions billing, runner pricing). The rounding matters more than you'd think for a Tiny Tapeout repository, because a push to a digital one starts six jobs. In the `protocol-emulator` run above, the four jobs of the `gds` workflow took 13 min 33 s, 3 min 38 s, 1 min 2 s and 9 s, which round up to 14 + 4 + 2 + 1 = 21 minutes. Add a minute each for `docs` and `test`, and one push costs 23 minutes, so a private repository like it would use up the month's allowance in about 87 pushes, and probably fewer, since a private repository's runner has half the CPUs and the jobs would take longer. In a public repository the same pushes cost nothing.

A few limits apply either way. A single job can run for at most 6 hours, a Free account can run 20 jobs at once, and a run can be re-run at most 50 times (limits). You'll work in your own GitHub account, so your jobs queue against your own limit and not against anyone else's.

Logs and artifacts don't last forever. By default GitHub keeps them for 90 days, and History as evidence shows what that looks like when you need an old log and it's gone.

Your tools and the runner's tools

The runner installs its tools fresh on every run, at whatever versions the workflow asks for, and those needn't match the tools on your laptop or in the workbench. In a `hart-1` test run on Nov 15, 2026, the `test` job installed Icarus Verilog `12.0-2build2` from Ubuntu's packages and `cocotb-2.1.0` from the repository's `test/requirements.txt`. The `gl_test` job installs a different Icarus, Tiny Tapeout's own 13.0 package, and at the digital template's current tag (`ttsky26d`) the `gds` job installs LibreLane 3.0.14. The `precheck` job, in both templates, builds its own Magic and KLayout and installs the PDK at a fixed commit. A difference in any of these can make a check pass in one place and fail in the other.

For that reason IC102 pins the tools you run locally to the versions Tiny Tapeout's workflows use, so that your local runs and the runner's agree. Your tools and the runner's tools lists them for your version of the course, and says where the two still differ.

You might wonder whether you could run the workflows themselves on your own machine and skip the wait. There is a community tool for that, `act`, which "reads in your GitHub Actions from `.github/workflows/`" and runs them in Docker containers (nektos/act). It isn't GitHub's, though, and its own documentation says that "Default runners are intentionally incomplete". Among the features it lists as unsupported are step summaries, `job.permissions` and OIDC (not supported), and the `viewer` job needs the last two, so it can't work under `act`. In this course you run the individual tools locally to get quick answers, and treat the runner as the place where a check counts.

Check yourself

1. The `precheck` job needs the GDS file the `gds` job just built. Why does it download an artifact instead of reading the file from the `gds` job's disk?

Answer: A1, in Answers at the end.

2. A push to your repository goes red in under a minute. Is the problem more likely in your design or in your configuration, and why?

Answer: A2, in Answers at the end.

3. A friend keeps their Tiny Tapeout repository private on a Free account and pushes about five times a day. Using the `protocol-emulator` numbers on this page, roughly how long will the month's allowance last?

Answer: A3, in Answers at the end.

4. Your cocotb test passes in the workbench and fails on the runner. Name two things that could differ between the two machines.

Answer: A4, in Answers at the end.

Next: Reading a workflow file.

Answers

A1. Check yourself

Because the two jobs run on different machines. Each job gets a new virtual machine that's thrown away when the job ends, so the only way to hand a file from one job to the next is to upload it as an artifact (`tt_submission`) and download it in the later job.

A2. Check yourself

In your configuration. Building a layout takes several minutes, so a job that fails within a minute stopped before the build started, usually at a step that reads `info.yaml` or another setup file, as in `hart-1`'s 54-second `gds` failure.

A3. Check yourself

At about 23 billed minutes a push, five pushes a day use about 115 minutes a day, so 2,000 minutes last about 17 days. Made public, the same repository would cost nothing.

A4. Check yourself

Any two of: the simulator version (the runner's `test` job installed Icarus 12.0 from Ubuntu's packages on Nov 15, 2026), the cocotb version (whatever `test/requirements.txt` pins, which was 2.1.0 in the digital template), a file you have locally but never committed, or a setting in your own shell that the runner doesn't have.