GUIDE 1

Setup

Standing H3 up on a machine you control, with a local model.

H3, the Hackathon as a Service Harness
303 Overwatch A.S.B.L

Generated 29 August 2026 from `docs/guides/01-setup.md` in the H3 repository. The markdown is the source; edit that, not this document.

Contents

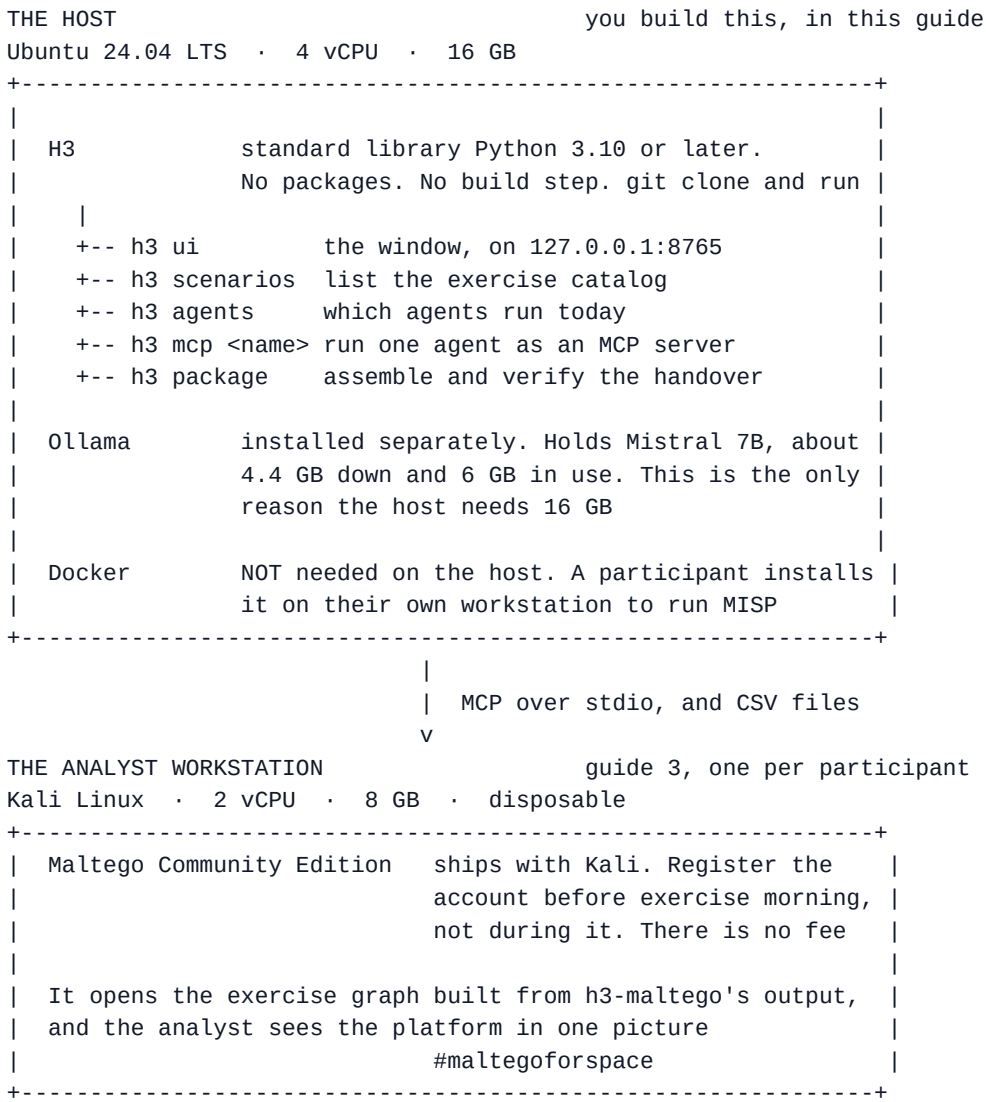
How to stand H3 up on a machine you control, with a local model, so that nothing you type leaves the building.

Allow about an hour. Most of it is downloading a model.

This guide assumes Ubuntu 24.04 LTS, bare metal or a virtual machine. Nesting is not a problem: the host is standard-library Python and runs no containers, so nothing here needs nested virtualization. The analyst tools that do run containers, MISP especially, are stood up on the analyst workstation in guide 3, not on this host.

What you are installing

Two machines with two different jobs, and confusing them is the most common setup mistake. The host runs the exercise. The analyst workstation is where a participant sits. This guide builds the host; the workstation comes in guide 3.



Why two machines rather than one. The host optimises for never changing, so the same exercise runs twice and comes up offline from pinned digests. The workstation optimises for having the tools already there, and is thrown away afterwards. Neither wants the other's tradeoff, and running an

exercise from a rolling release host is how an exercise stops being reproducible.

Nothing on the host talks to the internet once Ollama has its model. Step 8 disconnects the machine and proves it.

What actually runs today. All 20 agents serve and all 20 answer, from hash-verified published frameworks read with the standard library. The refusal path is still there: an agent whose corpus was bundled in a format H3 could not read would speak the protocol and refuse every content question rather than answer without the framework in front of it, which would produce something that looks exactly like a citation and is not one. No agent is in that state today.

Run `h3 agents` after step 3 and it will tell you, in a table it derives rather than one somebody wrote down. If the table and this sentence ever disagree, believe the table.

Before you start

Read the sizing table in PLATFORM-REQUIREMENTS. The short version: 4 vCPU and 16 GB of RAM runs H3 with a local model. A participant workstation running Maltego and a MISP instance in Docker wants 8 GB of its own.

Decide who owns this host. It will hold synthetic attack data, so it is an exercise asset from the first day rather than something to be classified later.

Step 1. Prepare the host

Update, set the timezone, create a non-root user, and put your SSH key on it.

```
sudo apt update && sudo apt upgrade -y
sudo timedatectl set-ntp true
```

Verify. `timedatectl` reports the clock synchronised. This matters more than it looks: an exercise is judged against a reporting clock, and a host with the wrong time produces a record nobody can defend.

Step 2. Install Ollama and pull Mistral

```
curl -fsSL https://ollama.com/install.sh | sh
ollama pull mistral
```

About 4.4 GB down and roughly 6 GB in use.

Verify, before H3 is anywhere near it.

```
ollama run mistral "Reply with the single word: ready"
```

If it fails. No GPU is required; Ollama falls back to the processor and is slower, not broken. If the pull stalls, it resumes: run it again. Two things that can fail should never be installed as one step, which is why this is proved on its own.

Step 3. Get H3

```
git clone https://github.com/h4ck32n4u75/h3.git h3
cd h3
```

H3 has no runtime dependencies. It is standard-library Python 3.10 or later, which is why this step is shorter than people expect.

Ubuntu needs one package first, and this is where people stop. A stock Ubuntu 24.04 server image ships Python without `pip`, without `ensurepip` and without `venv`, and it marks the system interpreter `EXTERNALLY-MANAGED` under PEP 668, which means even a working `pip` would refuse to install into it. So the install line below cannot run on a fresh host until you do this:

```
sudo apt install -y python3-venv
```

Then install H3 into a virtual environment, which is what sidesteps PEP 668 and is the habit that will not bite you later:

```
python3 -m venv .venv
source .venv/bin/activate
python3 -m pip install -e .
```

That installs no third-party package. It registers the `h3` entry point and puts the module on the path, which is what lets you run H3 from any directory rather than only from inside the clone. The environment has to be active in every shell that runs `h3`, so if a later step reports `h3: not found`, the answer is almost always `source .venv/bin/activate` in that shell.

Verify.

```
python3 --version
cd /tmp && h3 doctor && cd -
```

Run it from `/tmp` deliberately. The whole point of installing is that `h3` works from a directory that is not the clone, and running the check inside the clone would pass even if that had failed.

`doctor` probes rather than recites: it tells you whether Ollama is actually reachable, which hosted keys are set, whether PyYAML is present, and which Python it is running under. Having done step 2, it should report Ollama reachable with `mistral` pulled, and no keys set, which is the correct default. If you skipped step 2 it will say `ollama NOT reachable`, and that is a true statement about Ollama rather than a problem with H3: every check in step 4 still passes without a model.

If you have no sudo, or would rather not install anything, every command in these guides also works as `python3 -m h3.cli ...` from inside the clone directory, with `export PYTHONPATH=.` set first. This is a real branch and not a consolation prize: it needs no package, no environment and no privileges, and it is the only route on a locked-down host. Pick one convention and stay with it. The guides use the installed `h3` form from here on, because a command that only works from one directory is a command people get wrong at the worst moment; guide 3 shows the module form alongside it so both readers are served.

Step 4. Prove the install with no model at all

```
python3 -m unittest discover -s tests -t .
```

Then run the whole pipeline against the fixture player, which needs no model, no key and no network:

```
python3 -m h3.cli init --project /tmp/demo \
  --platform "Demo Orbital" \
  --statement "A European operator flying a small imaging constellation, commanded from its own telepo
python3 -m h3.cli f01 --project /tmp/demo --adapter echo \
  --decided-by "Your Name"
python3 -m h3.cli validate --project /tmp/demo
```

Verify. The tests pass, and `validate` prints:

```
PASS  no violations, but this record is SAMPLE DATA
```

followed by a note saying how many elements came from the fixture player. That qualification is deliberate and it matters. The `echo` adapter replays a canned satellite decomposition and ignores whatever you typed, so this record is grammatically perfect and describes somebody else's platform. It proves the pipeline works. It is not evidence about anything, and H3 will not let a bare `PASS` suggest otherwise.

Now ask the harness what it actually has:

```
python3 -m h3.cli agents
```

Verify. A table of 20 agents, one per row, each marked `reachable` or `no corpus`, ending with a count per state. This table is derived from the code every time you run it, so it cannot drift from what is installed.

Read the two counts separately and do not add them together. `reachable` means the agent answers. `no corpus` means it runs and refuses, which is a different and honest thing. Prove one of each speaks the protocol:

```
printf '%s\n' '{"jsonrpc":"2.0","id":1,"method":"initialize","params":{}}' \
  '{"jsonrpc":"2.0","id":2,"method":"tools/list","params":{}}' \
  | python3 -m h3.cli mcp maltego --project /tmp/demo
```

And one that refuses, so you see what the other 15 do:

```
printf '%s\n' \
  '{"jsonrpc":"2.0","id":1,"method":"tools/call","params":{"name":"lookup","arguments":{"query":"later
  | python3 -m h3.cli mcp attack --project /tmp/demo
```

It answers with a refusal that names the framework, the inventory file and what would fix it. That is the agent working correctly, not failing.

The whole roster can be checked at once, by the agent whose job that is:

```
printf '%s\n' '{"jsonrpc":"2.0","id":1,"method":"tools/call","params":{"name":"health","arguments":{}}
  | python3 -m h3.cli mcp federation --project /tmp/demo
```

It builds every one of the 22 and asks each for its tool list over the real transport, rather than reporting what a registry claims.

Verify. Two JSON lines back. The first names `h3-maltego` and protocol version `2024-11-05`. The second lists its tools, each tagged `[SAFE | class ORG]` so a client can tell you what a call will do before it happens.

Why this step exists. When something misbehaves later, you already know whether the harness or the model is at fault. Skipping this costs an evening the first time it matters. The `agents` table matters for a second reason: it is how you check a claim in our documentation against the software in front of you.

Step 5. Open the window

```
python3 -m h3.cli ui --project /tmp/demo
```

On a machine with a desktop, that opens a browser. On a headless host, forward the port from your own machine instead:

```
ssh -L 8765:127.0.0.1:8765 user@your-host
```

then open `http://127.0.0.1:8765/` locally.

Do not bind H3 to a public interface. The port forward keeps the loopback binding intact and exposes nothing on the lab network. That is the point of it.

Verify. The window loads, and the header shows the project path.

Step 6. Connect H3 to the local model

The Exercise tab selects a prepared exercise rather than generating one, so it does not ask for a model. Two places in the window do: the **AI WorkBench** tab, whose model picker offers `ollama` and the hosted providers, and the **Platform** tab's Configuration view, which shows which adapters this machine can reach.

The check that proves the model is wired up is on the command line:

```
python3 -m h3.cli f01 --project /tmp/demo --adapter ollama --model mistral \
  --decided-by "Your Name"
```

Verify. Proposals appear, some are accepted, some may be rejected with a validator code and a plain sentence. Rejections at this stage are the system working, not failing.

Step 7, optional. Add a hosted key

H3 needs no key. If you want one for the harder functions or for speed on a machine with no GPU:

```
export H3_ANTHROPIC_KEY=...
```

`H3_OPENAI_KEY`, `H3_GEMINI_KEY` and `H3_MISTRAL_KEY` work the same way.

Now prove the boundary before you use it.

```
python3 -m h3.cli f01 --project /tmp/demo --adapter anthropic --layer PCE \
  --decided-by "Your Name"
```

This must refuse. Your platform description is class ORG and no hosted adapter is permitted to see it by default. Only then, if you accept the egress:

```
python3 -m h3.cli f01 --project /tmp/demo --adapter anthropic --allow-org-egress \
  --decided-by "Your Name"
```

That flag is per command. There is no setting that turns the boundary off, because a boundary you can forget you disabled is not a boundary.

Step 8. Go offline and prove it

Pre-stage everything while you still have a network, then disconnect the host and run Step 4 and Step 6 again.

A claim about offline operation that has not been tested with the cable out is a hope. Test it now, while the install is fresh, rather than on the morning of an exercise.

Next

Guide 2, Platform validation, proves the platform is sound before you trust it with real work. Do not skip it: it is the only stage that tests whether H3 refuses what it claims to refuse.