TruVerifAI

PANEL REVIEW · SANDBOX SETUP GUIDE

From zero to a blocked risky change in ~15 minutes

For the engineering team running the evaluation: stand up the disposable sandbox, arm the gates, and test against real code.

What you'll do: stand up a disposable machine in your own tenancy, install your coding agent on it, arm the review gates, and watch a real risky change get blocked in about a minute. Then point it at your own code. Delete the machine when done and nothing remains. **Budget ~15 minutes.** Start from the template repo at github.com/TruVerifAI/panel-review-sandbox; this guide expands the repo's README.

Two accounts are involved, both used on the sandbox machine:

- **Your coding-agent account** (for example your Anthropic/Claude Code login), used to sign the agent in.
- **Your TruVerifAI account**, used by `npx @truverifai/init` to arm the gates and to grant evaluation credits. Same login later unlocks BYOK / BYOM / persistence on truverif.ai.

1 Open the machine

You need a Linux machine with **Node 18+, Python 3, and git**. Pick one path.

Path A: GitHub Codespaces (one click, nothing to provision). Open the template repo, click **"Use this template" ▸ Create a new repository** to make your own private copy, then in your copy click **Code ▸ Codespaces ▸ Create codespace on main**. A browser VS Code opens in about a minute; Node, Python, and git are already installed, and a welcome banner prints the steps. Everything below runs in its terminal.

Path B: your own AWS EC2 (or any VM / Docker host). Any box with Node + Python + git works. On a fresh EC2 instance: launch Amazon Linux 2023 or Ubuntu 22.04+ (a `t3.small`, 2 vCPU / 2 GB, is plenty; open only SSH port 22 to your own IP; no inbound web ports are needed), SSH in, then:

```
# Ubuntu:
sudo apt update && sudo apt install -y git python3 curl
curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -
sudo apt install -y nodejs

# Amazon Linux 2023:
sudo dnf install -y git python3 nodejs20
```

Get your copy of the sandbox: use "Use this template" on GitHub as in Path A to create your own repo, then clone it (recommended, so commits are yours), or clone the template directly to try it read-only:

```
git clone https://github.com/<your-org>/<your-sandbox-copy>.git
cd <your-sandbox-copy>
npm install
```

Confirm the toolchain: `node -v` (18+), `python3 --version` (3.x), `git --version`.

2 Install your coding agent and sign it in

`init` arms the gates on **whatever agents are already installed** on the machine (it detects them by looking for their CLI on the PATH). So install the agent you'll use here, on the sandbox machine, and sign in with **your own agent account**, before you run `init`. For example, Claude Code:

```
npm install -g @anthropic-ai/claude-code
claude   # launches the agent; sign in with your company Claude account when prompted
```

Any of the certified agents work the same way: install it on the machine and sign in with your account: **Claude Code, Codex CLI, Cursor CLI, GitHub Copilot CLI, Gemini CLI, Antigravity.** (You don't have to install all of them, just the one your team uses.)

3 Arm the review gates

```
npx @truverifai/init
```

`init` detects the agent(s) from Step 2, then signs you in to **TruVerifAI** by **device flow**: it prints a short code and a `truverif.ai` URL and waits. Open that URL in **your own browser** (on any device), enter the code, and approve. No browser opens on the sandbox machine; you approve it the way you sign a TV into a streaming app. Your first TruVerifAI login is granted **free evaluation credits** automatically.

It then installs the **write gate and commit gate** on each detected agent, plus a **git pre-commit gate** for the repo. Confirm everything is live (it fires a synthetic gate to prove it, not just checks that files exist):

```
npx @truverifai/init doctor
```

Green rows mean armed. If you install another agent later, just re-run `npx @truverifai/init` to arm it too. For the full per-agent setup instructions (and to confirm which surfaces are connected to your account), see **truverif.ai/settings/mcp**.

4 Watch a gate block a risky change

The repo ships **throwaway demo code** in `demo/` (a small, non-proprietary trading core that does not run, so it is safe to break). The repo also marks that code as **custom floors** in `.truverifai/risk.json`, so the gate treats any change to it as must-review. One command stages a change that warrants panel review:

```
npm run demo:risky-change
```

This deterministically edits `demo/tradecore.py`, widening the position-size cap `MAX_POS_PCT` from `0.08` to `1.00` (it removes the 8% notional cap, exactly the kind of change that must never ship unreviewed). Now commit it:

```
git add -A && git commit -m "widen position cap"
```

The **commit gate blocks it**, cites the `trade_core` / `risk_limits` floor from `.truverifai/risk.json`, and routes it to a four-model review. That is the product in one motion. To undo the demo edit, revert the file with git (`git checkout -- demo/tradecore.py`).

Then work through **THINGS-TO-TRY.md** in the repo: remove a stop-loss and commit (blocked), edit a comment and commit (it sails through; the gate isn't just blocking everything), and run the review the gate points you to, apply the findings, and commit clean.

5 Bring your own code and set up custom floors

The real test is your own code. Drop **any non-critical repo of yours** onto the machine and build features on it with your agent for a day. You get real verdicts on code you understand, and it never leaves your tenancy.

To make the gate treat *your* important files as must-review, set up custom floors:

1. **Scaffold and draft.** Run `npx @truverifai/init floors init`. It creates an inert `.truverifai/risk.json` and prints a prompt you can hand your agent to draft real floors for the repo. (Or just ask your agent directly: "set up custom floors for this repo.")
2. **Validate coverage.** Run `npx @truverifai/init floors check --preview`. It runs your repo's files through the matcher and shows exactly which files each floor covers, so you can confirm you fenced the right code before relying on it.
3. **See it fire.** Change a floored file and commit; the gate blocks and cites your own floor by name.

6 Keep your data where you want it (BYOK / BYOM / persistence)

Signed in to truverif.ai with the **same TruVerifAI account** you used in Step 3:

- **Content persistence.** ON by default (a dashboard audit trail of your eval). Turn it **off** at **truverif.ai/settings/mcp** and review content is no longer stored on our side (only usage/billing metadata is). Your choice.
- **BYOK (bring your own keys).** Run the review on your own model-provider keys, so your provider bills you directly and we charge only the platform fee. Set it up and see the walkthrough at **truverif.ai/byok**.
- **BYOM (bring your own models).** Run the whole panel on **your own models in your own AWS Bedrock or GCP Vertex tenancy**. Review content is then processed by models you already operate and never reaches our accounts. Best if you already run inference. Details at **truverif.ai/byom**.

Regardless of setting, only the **diff you submit for review** ever leaves the machine; the local gate checks send hashes and labels, never your source. Full data-flow: **SECURITY.md** in the repo (and the companion Security & Data-Flow brochure).

7 When you're done, tear it down

```
npx @truverifai/init uninstall
```

Removes the gates and **revokes your API key server-side** (so a leaked copy is dead), then delete the Codespace or terminate the EC2 instance. Uninstall is the machine ceasing to exist.

Cost and credits

Compute only: often **$0** on the Codespaces free tier, and single-digit to low-tens of dollars for a multi-day eval on Codespaces or a small EC2 box. The review **credits are on us** for the evaluation: your first login gets 50 free, and if you need more, email us the login you used and we top you up.

> The entire client is open source (MIT, zero runtime dependencies): github.com/TruVerifAI/init. Read every line before you trust it.