HOOX · PYNE · DOCUMENTATION

PLATFORM ENGINEERS · RELEASE OPERATORS

DEVOPS MANUAL

CI, Docker, Nuitka, metadata crypto, GCP, security

Open-source Pine Script evaluation framework

2026-08-03 DIN A4 · HTTPS://HOOX.SH/PYNE/DOCS OPEN SOURCE

CONTENTS

OVERVIEW

title: "DevOps" description: "CI matrices, containers, Nuitka LSP binaries, metadata crypto, Cloud Run, and operational invariants for PYNE."

DEVOPS

Abstract

PYNE is not a single binary. It is a **multi-surface product**: a pure-Python library, a Click CLI, a pygls Language Server, a Flask Pro API, a VS Code extension, an AXIS PWA AXIS, and colocated edge workers. DevOps here means keeping those surfaces buildable, testable, and deployable without drifting from the same evaluate contract.

This tab documents the **operational graph**: local loops, GitHub Actions, release tags, Docker images, Nuitka compilation, Fernet-encrypted LSP metadata, GCP Cloud Build/Run, observability hooks, and security controls.

Conceptual model

```
flowchart TB
  subgraph Local
    DEV[make install / hatch]
    TEST[make test lint]
    RUN[make run / run-lsp]
  end
  subgraph CI["GitHub Actions"]
    LINT[lint + mypy]
    PY[test matrix 3.10–3.13]
    PWA[Bun typecheck + coverage]
    VSIX[vscode-extension package]
  end
  subgraph Artifacts
    WHEEL[pynescript wheel]
    LSPBIN[Nuitka pynescript-lsp]
    VSIXART[*.vsix]
    IMG[Docker API image]
  end
  subgraph Deploy
    CR[Cloud Run Pro API]
    REL[GitHub Release + Marketplace]
  end
  DEV --> TEST --> LINT
  LINT --> PY --> PWA
  PY --> WHEEL
  REL --> LSPBIN
  REL --> VSIXART
  IMG --> CR
```

Invariant: CI green on `main` is necessary but not sufficient for a release. Tag-triggered release workflows build platform LSP binaries and VSIX; Cloud Build deploys the API image on its own pipeline.

Interface surface

Concern	Entry	Doc
Local install & loops	`Makefile`, hatch envs	Local development
PR / push CI	`.github/workflows/ci.yml`	CI
Versioned LSP + VSIX	`.github/workflows/release.yml`	Release
Pro API containers	`Dockerfile` (targets api/api-dev/lsp), `docker-bake.hcl`, `docker-compose.yml`	Docker
Compiled LSP	`scripts/build/compile.py`, `ci_build.py`	Nuitka build
Builtin metadata crypto	Fernet key + `.enc`	Metadata crypto
Cloud Run	`cloudbuild.yaml`	GCP
Logs / health	Flask `/`, gunicorn, Cloud Logging	Observability
Auth, CORS, secrets	`backend/middleware/auth.py`	Security

Internals (repo map)

Path	Role
`Makefile`	Human-facing orthography: install, test, lint, build, docker, worker
`pyproject.toml`	Hatch envs (`test` , `lint` , `docs`), optional extras (`lsp` , `data`)
`.github/workflows/`	`ci.yml` , `release.yml` , `axis-nightly.yml`
`scripts/build/`	Nuitka compile + CI build + Fernet metadata stage
`scripts/generate_builtin_metadata.py`	Regenerates LSP `builtin_metadata.json` from live builtins
`Dockerfile` / `docker-bake.hcl`	Multi-target Buildx images (api, api-dev, lsp)
`docker-compose.yml`	Local API (+ optional Redis / LSP profiles)
`cloudbuild.yaml`	Build → GCR push → Cloud Run deploy; optional LSP compile
`vscode-extension/`	Node 22 extension package / vsce

Invariants & edge cases

1. **Two console scripts, two entrypoints.** `pynescript` (Click) ≠ `pynescript-lsp` (pygls). Ops scripts must not conflate them.
2. **Generated artifacts are not hand-edited.** ANTLR/ASDL under `generated/` and `builtin_metadata.json` are code-derived.
3. **Fernet key is gitignored.** Without a stable `CRYPTO_KEY` / `METADATA_KEY` secret, every CI encrypt produces a different `.enc` blob (harmless functionally, bad for reproducibility).
4. **Python matrix vs local default.** CI tests 3.10–3.13; Nuitka release currently pins 3.12.
5. **AXIS nightly is not PR-blocking.** Full Playwright + security gates run on schedule (`axis-nightly.yml`).

Worked examples

```
# Fast local confidence loop
make install
make lint
make test-lsp
make build-check   # import check only, ~30s, no Nuitka compile

# API in Docker — local *dev* stack (api-dev + source mounts)
make docker-up
make docker-smoke

# Production image bake (load local) + optional prod compose overlay
make docker-build
# export ADMIN_TOKEN=… && make docker-prod
```

Failure modes

Symptom	Likely cause	Fix
CI lint red, local green	Different ruff/mypy versions	Match CI install pins in workflow
Metadata decrypt fails in binary	Missing key at runtime	Set `PYNESCRIPT_METADATA_KEY` or embed key at build
Cloud Run 502 after deploy	Image missing deps / gunicorn bind	Confirm `Dockerfile` target `api` ENTRYPOINT + `PORT` / `GUNICORN_BIND`
VSIX empty / missing	`npm ci` / vsce not run	Use `make build-vscode` or release job artifacts
Nuitka Anaconda link error	Static libpython missing	`conda install libpython-static` or keep `--static-libpython=no`

See also

- Contributing
- LSP architecture
- Pro API contract
- pine-worker

LOCAL DEV

title: "Local development" description: "Install, test, lint, and run PYNE surfaces on a developer machine — Make, hatch, Bun, and dual-language loops."

LOCAL DEVELOPMENT

Abstract

A PYNE checkout is a **polyglot workspace**: Python (src-layout package + Flask backend), TypeScript/Bun (AXIS frontend, Cloudflare worker, `pine-worker`), and Node (VS Code extension). Local DevOps is about choosing the right orthography for the surface you are changing without spinning up the entire mesh.

Conceptual model

```
flowchart LR
  SRC[src/pynescript] --> CLI[pynescript CLI]
  SRC --> LSP[pynescript-lsp]
  SRC --> LIB[import pynescript]
  BE[backend/] --> API[Flask :8080 / :5002]
  FE[frontend/] --> PWA[Bun server :8081]
  PW[pine-worker/] --> BUN[bun test]
  VSC[vscode-extension/] --> VSIX[vsce package]
```

Interface surface

Make targets (primary)

Target	Effect
`make install`	`pip install -e ".[lsp]"`
`make install-bun`	Root + `frontend/worker` Bun deps
`make test`	Frontend Bun tests + full `pytest tests/`
`make test-lsp`	`test_langserver.py` + `test_lsp_features.py`
`make test-backend`	`tests/test_backend.py`
`make lint` / `make fmt`	Ruff check / format on `src/` , `tests/` , `backend/`
`make typecheck`	Root TS + frontend worker `tsc`
`make build`	Nuitka LSP via `scripts/build/compile.py --jobs=4`
`make build-check`	Fast import resolution only (~30s)
`make build-vscode`	Compile + package VSIX
`make run`	`python -m backend.app`
`make run-lsp`	`python -m pynescript.langserver`
`make run-frontend`	AXIS PWA on `:8081` (expects API on `:5002`)
`make docker-build`	`docker buildx bake api` (production image)
`make docker-build-all`	bake api + api-dev + lsp
`make docker-buildx`	multi-platform release bake
`make docker-up` / `docker-run`	compose API on `:5002` (dev target)
`make docker-up-full`	compose with redis profile (not lsp)
`make docker-prod`	gunicorn overlay, no source mounts (needs `ADMIN_TOKEN`)
`make docker-smoke`	curl healthcheck on `:5002`
`make docker-down` / `docker-logs`	stop stack / follow API logs

Hatch equivalents

```
hatch run test:test
hatch run test:test-cov
hatch run lint:style
hatch run lint:typing
hatch run lint:gen-parser   # ANTLR regeneration — see grammar guide
```

See also `CONTRIBUTING.md` and `AGENTS.md` for hatch-first workflows.

Internals

Path	Why it matters locally
`src/pynescript/`	Editable package root (src-layout)
`tests/conftest.py`	Parametrizes `pinescript_filepath` over every `tests/data/builtin_scripts/*.pine` (~hundreds of cases)
`tests/data/library/`	Reference corpus only — not auto-parametrized
`backend/requirements.txt`	Extra deps for Pro API (Flask, numpy, matplotlib, ...)
`pine-worker/`	Colocated TS port; Bun tests independent of Python wheel

Python requirements

- **Requires-Python:** `>=3.10` (`pyproject.toml`)
- **CI matrix:** 3.10–3.13
- **Recommended local:** 3.12 or 3.13 with a virtualenv / hatch env

Optional extras

```
pip install -e ".[lsp]"       # pygls + lsprotocol
pip install -e ".[dev-lsp]"   # + pytest-lsp for e2e LSP
pip install -e ".[data]"      # ccxt for data providers
pip install -r backend/requirements.txt
```

Invariants & edge cases

1. **`from __future__ import annotations` is mandatory** on every new Python file (ruff isort `required-imports`).
2. **Do not edit generated grammar.** Change only `src/pynescript/ast/grammar/antlr4/resource/*.g4`, then regenerate.
3. **No stale backups in `src/`.** Live `builder.py` and `technical.py` are sole sources of truth.
4. **Corpus fixture cost.** Any test using `pinescript_filepath` multiplies across the builtin script corpus. Narrow with `--example-scripts-dir=...` when iterating.
5. **Console scripts are separate.** Installing `.[lsp]` gives both `pynescript` and `pynescript-lsp`; invoking the wrong one is a common first-day footgun.

Worked examples

Library + parse round-trip

```
make install
python -c "from pynescript.ast.helper import parse, unparse; print(unparse(parse('//@version=6\ninc
```

Pro API + AXIS

```
# terminal 1
make run
# terminal 2
make run-frontend
# open http://127.0.0.1:8081 — /run hits backend on :5002 per compose/docs convention
```

LSP without Nuitka

```
make install
make run-lsp
# point editor to stdio server: python -m pynescript.langserver
```

pine-worker only

```
cd pine-worker
bun install
bun test
bun run typecheck
```

Failure modes

Symptom	Cause	Mitigation
`ModuleNotFoundError: pynescript`	Not editable-installed	`make install`
Hundreds of unexpected test failures	Editing shared conftest fixture	Isolate unit tests; avoid corpus fixture
Frontend fails calling API	Backend not on expected port	Start `make run`; check CORS / `ALLOWED_ORIGINS`
`antlr4` not found	CLI not on PATH	Use hatch `lint:gen-parser` or full path to `antlr4`
Ruff `I` import errors	Missing future annotations	Add `from __future__ import annotations`

See also

- CI
- Docker
- Nuitka build

- Contributing
- Installation (end user)

CI

title: "Continuous integration" description: "GitHub Actions matrix for lint, Python tests, backend, VSIX, AXIS PWA, e2e smoke, and nightly gates."

CONTINUOUS INTEGRATION

Abstract

CI is the **public invariant check** for the monorepo-like layout. A single PR can touch the Python evaluator, the Flask API, the VS Code extension, the AXIS PWA, and a Cloudflare Worker typecheck — the workflow graph is deliberately fan-out with concurrency cancellation on newer pushes to the same ref.

Primary definition: `.github/workflows/ci.yml`. Nightly AXIS expansion lives in `axis-nightly.yml`. Release builds are separate (`release.yml`).

Conceptual model

```
flowchart TB
  PUSH[push / PR → main] --> CONC[concurrency: ci-ref]
  CONC --> LINT[lint]
  CONC --> TEST[test matrix 3.10–3.13]
  CONC --> BE[backend-test]
  CONC --> VSC[vscode-ext-test]
  CONC --> PWA[pwa Bun]
  CONC --> E2E[axis-e2e smoke]
  CONC --> CF[cf-worker typecheck]
  NIGHT[schedule 04:00 UTC] --> FULL[axis-nightly full e2e + security]
```

Interface surface

Triggers

Workflow	On	Purpose
`ci.yml`	push/PR to `main`	Required-path confidence
`axis-nightly.yml`	cron `0 4 * * *` + `workflow_dispatch`	Full Playwright + security + coverage gate
`release.yml`	tags `v*` + dispatch	LSP binaries + VSIX + GitHub Release

`ci.yml` jobs

Job	Runtime	What it asserts
lint	Python 3.13	`ruff check src/ tests/` ; `mypy src/ --ignore-missing-imports`
test	Matrix 3.10–3.13	Editable `.[lsp]` ; LSP unit/e2e tests; backend tests; Codecov upload on 3.13 only
backend-test	3.13	Explicit Flask/numpy/matplotlib install + `test_backend.py`
vscode-ext-test	Node 22	`npm ci` → compile → `vsce package` → upload VSIX artifact (7-day)
pwa	Bun 1.3	Root + worker install, dual `tsc` , frontend coverage gate + security tests; upload lcov
axis-e2e	Bun 1.3	Playwright Chromium smoke (`test:e2e:smoke`)
cf-worker	Bun 1.3	`frontend/worker` typecheck only

Concurrency:

```
concurrency:
  group: ci-${{ github.ref }}
  cancel-in-progress: true
```

Nightly (`axis-nightly.yml`)

- Full e2e suite (`bun run test:e2e`) with report artifact on failure
- Coverage gate + security tests (stricter than PR smoke)

Internals

Path	Role
`.github/workflows/ci.yml`	PR/main fan-out
`.github/workflows/axis-nightly.yml`	Scheduled AXIS depth
`.github/workflows/release.yml`	Tag release (see Release)
`frontend/scripts/check-coverage.mjs`	Coverage gate helper for AXIS
`codecov` action	Optional; `fail_ci_if_error: false`

Secrets referenced in CI surface

Secret	Used by	Notes
`VSCE_PAT`	VSIX package / publish	Extension packaging
`METADATA_KEY`	Release / Cloud Build encrypt	Stable Fernet key for metadata
`GITHUB_TOKEN`	Release upload	Default Actions token

``` — CI `test` job currently does not run the full `tests/` corpus on every matrix cell; it prioritizes LSP + backend. Local `make test` is broader.

Invariants & edge cases

1. **Fail-fast is off** on the Python matrix — one version failure does not hide others.
2. **Codecov only on 3.13** to avoid noisy multi-upload; coverage XML path depends on pytest-cov configuration in the job.
3. **PWA install is resilient:** `bun install --frozen-lockfile || bun install` — lockfile drift should still be fixed, not papered forever.
4. **VSIX artifacts are not the Marketplace.** Packaging on every PR proves buildability; publish is release-gated.
5. **Worker typecheck ≠ deploy.** `cf-worker` job does not call `wrangler deploy`.

Worked examples

Reproduce lint locally as CI does

```
pip install ruff flake8 mypy types-python-dateutil antlr4-python3-runtime
ruff check src/ tests/ --output-format=github
mypy src/ --ignore-missing-imports
```

Reproduce Python job slice

```
pip install -e ".[lsp]"
pip install pytest pytest-cov pytest-xdist pytest-asyncio
python -m pytest tests/test_langserver.py tests/test_lsp_features.py -v --tb=short
python -m pytest tests/test_backend.py -v --tb=short
```
```

Reproduce PWA job

```
bun install --frozen-lockfile
cd frontend/worker && bun install --frozen-lockfile && bunx tsc --noEmit
cd ../.. && bunx tsc -p tsconfig.json
cd frontend && bun install && bun run test:coverage:gate && bun run test:security
```

Failure modes

Symptom	Likely cause	Fix
Matrix red only on 3.10	Version-specific typing / API	Guard with `sys.version_info` or drop unsupported API
Playwright smoke flake	Timing / missing deps	Install chromium with deps; re-run; check nightly for full suite
VSIX job fails on `vsce`	Node/engine mismatch	Pin Node 22 as workflow does
Coverage gate red	New untested AXIS paths	Add unit tests or adjust gate intentionally
Cancelled mid-run	Newer push to same branch	Expected concurrency behavior

See also

- Release
- Local development
- Observability
- Security

RELEASE

title: "Release" description: "Tag-driven LSP multi-platform Nuitka binaries, VSIX packaging, GitHub Releases, and Marketplace publish."

RELEASE

Abstract

A PYNE release is a **multi-artifact event**, not a single wheel upload. The primary automation (`.github/workflows/release.yml`) builds platform-native Language Server binaries with Nuitka, packages the VS Code extension, attaches them to a GitHub Release, and optionally publishes to the VS Code Marketplace.

Library wheels may still be built separately via hatch / PyPI workflows; the documented release pipeline optimizes for **editor distribution** (LSP + VSIX).

Conceptual model

```
sequenceDiagram
  participant Dev as Maintainer
  participant GH as GitHub
  participant Build as build-lsp matrix
  participant VSC as build-vscode
  participant Rel as release job
  participant Mkt as Marketplace
  Dev->>GH: git tag vX.Y.Z && push
  GH->>Build: linux / windows / macos
  GH->>VSC: Node 22 vsce package
  Build->>Rel: artifacts
  VSC->>Rel: VSIX
  Rel->>GH: softprops/action-gh-release
  VSC->>Mkt: vsce publish (on tag)
```

Interface surface

Triggers

Event	Behavior
Push tag matching `v*`	Full build + create release + marketplace publish
`workflow_dispatch` with `version` input	Manual build path (release job still gated on tag push condition)

Jobs

Job	Matrix / OS	Output
`build-lsp`	ubuntu-latest, windows-latest, macos-latest	Named artifacts: `pynescript-lsp-linux-x86_64` , `…-windows-x86_64.exe` , `…-macos-arm64`
`build-vscode`	ubuntu-latest	`pynescript-vscode-extension` VSIX
`release`	needs both; only if tag push	GitHub Release body + asset globs
`publish-vscode`	needs VSIX; only if tag push	`vsce publish --pat`

Environment

```
env:
  PYTHON_VERSION: "3.12"
  NUITKA_JOBS: 4
```

Build steps rely on:

```
pip install nuitka==2.0.* cryptography
python scripts/build/compile.py --no-encrypt --check   # metadata stage
python scripts/build/ci_build.py --jobs 4
# CRYPTO_KEY from secrets.METADATA_KEY
```

Local make-side packaging

```
make build          # scripts/build/compile.py --jobs=4
make build-vscode   # npm install && compile && vsce package
```

Artifacts land under `dist/lsp/`, `dist/vsix/`, and `vscode-extension/*.vsix` depending on script path.

Internals

Path	Role
`.github/workflows/release.yml`	Orchestration
`scripts/build/ci_build.py`	CI-oriented Nuitka + metadata encrypt
`scripts/build/compile.py`	Local/full compile options (`--onefile`, `--standalone`, `--check`)
`vscode-extension/package.json`	Extension version / engines
`src/pynescript/__about__.py`	Hatch dynamic version for Python package

Release asset contract (from release body)

Artifact	Install sketch
`pynescript-lsp-linux-x86_64`	`chmod +x` → `/usr/local/bin/pynescript-lsp`
`pynescript-lsp-windows-x86_64.exe`	Place on PATH
`pynescript-lsp-macos-arm64`	Place on PATH (Apple Silicon runners)
`pynescript-*.vsix`	`code --install-extension pynescript-*.vsix`

Invariants & edge cases

1. **`CRYPTO_KEY` must be stable across builds** if you care about byte-identical `builtin_metadata.json.enc`. Supply `secrets.METADATA_KEY`.
2. **Tag name is the version source** for the GitHub Release title (`v` prefix stripped).
3. **Marketplace publish needs `VSCE_PAT`.** Without it, packaging may still succeed while publish fails.
4. **Artifact retention is short** (5 days on build jobs) — releases must attach files immediately.
5. **macOS artifact is ARM64** from `macos-latest` — Intel Mac users may need separate builds if required.
6. **Draft vs prerelease:** workflow currently sets `draft: false`, `prerelease: false` for tag releases.

Worked examples

Cut a release

```
# ensure main is green
git checkout main && git pull
# bump versions in extension / __about__ as needed
git tag -a v0.3.0 -m "v0.3.0"
git push origin v0.3.0
# watch Actions → Build & Release LSP
```

Smoke-test a downloaded binary

```
chmod +x pynescript-lsp-linux-x86_64
./pynescript-lsp-linux-x86_64 --help
# or wire stdio into an editor client
```

Failure modes

Symptom	Cause	Fix
Binary not found after Nuitka	Path layout drift in `ci_build.py`	Inspect `dist/`; fix finder step
Decrypt errors in field	Key mismatch between encrypt and runtime	Align `METADATA_KEY` with embedded key strategy
Empty GitHub Release assets	Artifact download path mismatch	Match `files:` globs to download-artifact layout
Marketplace 401	Bad/expired `VSCE_PAT`	Rotate PAT with Marketplace scopes
Windows `.exe` not marked executable	Finder uses `-type f -executable`	Fallback non-executable find branch in workflow

See also

- Nuitka build
- Metadata crypto
- CI
- VS Code extension

PYPI PUBLISH

title: "PyPI publish" description: "Ship hoox-pyne to PyPI via Trusted Publishing (OIDC). Import package stays pynescript."

PYPI PUBLISH

Abstract

The Python distribution name on PyPI is `hoox-pyne`. After install, the import path and CLIs stay `pynescript` / `pynescript-lsp`. Automation lives in `.github/workflows/publish.yml` and uses Trusted Publishing (OIDC) against the GitHub environment `pypi`.

> **Why not `pynescript` on PyPI?** That name is already owned by elbakramer/pynescript. Publishing under

> `hoox-pyne` avoids a name collision while keeping the code package id.

Install surface

```
pip install hoox-pyne
pip install "hoox-pyne[lsp]"
pip install "hoox-pyne[pro]"

python -c "import pynescript; print(pynescript.__version__)"
pynescript --help
pynescript-lsp   # stdio language server
```

One-time registration (do this before the first tag)

1. PyPI pending trusted publisher

1. Sign in at pypi.org (2FA required).
2. Account → **Publishing** → **Add a new pending publisher**.
3. Fill:

Field	Value
PyPI project name	`hoox-pyne`
Owner	`jango-blockchained`
Repository	`pyne`
Workflow name	`publish.yml`
Environment name	`pypi`

4. Save. The project is created automatically on the **first successful upload**.

2. GitHub environment

Repo **Settings** → **Environments** → **New environment** → **`pypi`**.

Optional: required reviewers, wait timer, or branch policy limited to tags / `main`.

```
# CLI equivalent (no protection rules):
gh api -X PUT repos/jango-blockchained/pyne/environments/pypi
```

3. Private repo note

`pyne` may be private. Trusted Publishing still works; the workflow only needs `id-token: write` on the publish job (already set).

Cut a release

```
# 1. Version + changelog on main
#    src/pynescript/__about__.py  →  e.g. 0.3.0
#    CHANGELOG.md

# 2. Local smoke (no upload)
pip install build twine
rm -rf dist/
python -m build
twine check dist/*
# expect: hoox_pyne-0.3.0-*.whl  and  hoox_pyne-0.3.0.tar.gz

# 3. Push main, then tag
git tag -a v0.3.0 -m "v0.3.0"
git push origin v0.3.0

# 4. Watch Actions → Publish
```

Dry-run without upload: **Actions → Publish → Run workflow →** `dry_run=true`.

Workflow map

Job	When	What
`build`	tag `v*` or dispatch	`python -m build` + `twine check` + artifact
`publish-pypi`	tag `v*` **or** dispatch with `dry_run=false`	OIDC upload via `pypa/gh-action-pypi-publish`

Failure modes

Symptom	Cause	Fix
403 / no trusted publisher	Pending publisher not added or wrong workflow/env name	Re-check PyPI pending publisher fields exactly
Environment not found	Missing GitHub `pypi` env	Create environment
Wrong wheel name `pynescript-*`	Old `pyproject.toml` name	Ensure `name = "hoox-pyne"`
Tag publish skipped	`dry_run` dispatch only	Use a real `v*` tag or `dry_run=false`

See also

- Release — LSP binaries + VSIX
- CI
- Root `CONTRIBUTING.md`

DOCKER

title: "Docker" description: "Buildx multi-target images, Compose profiles, and Cloud Run production contract for the PYNE Pro API."

DOCKER

Abstract

Containers package the **Pro API** (Flask + evaluator) for reproducible local runs and Cloud Run deploys. A single multi-stage `Dockerfile` exposes named targets; **Docker Buildx Bake** (`docker-bake.hcl`) drives local loads and multi-platform release builds. Compose wires volume mounts and optional Redis/LSP profiles for desk development, with a production overlay for gunicorn.

Conceptual model

```
flowchart LR
  subgraph Bake
    B[docker buildx bake]
  end
  subgraph Targets
    T1[api gunicorn]
    T2[api-dev Flask]
    T3[lsp stdio]
  end
  subgraph Runtime
    CR[Cloud Run :8080]
    CMP[compose :5002]
  end
  B --> T1
  B --> T2
  B --> T3
  T1 --> CR
  T2 --> CMP
  COMPOSE[docker-compose.yml] --> T2
  PROD[docker-compose.prod.yml] --> T1
```

Interface surface

Dockerfile targets

Target	Process model	Audience
`api`	`entrypoint-api.sh` → gunicorn (`0.0.0.0:$PORT`)	Production / Cloud Run / compose prod
`api-dev`	`python -m backend.app` with `HOST=0.0.0.0`	Local compose (source mounts)
`lsp`	`python -m pynescript.langserver` (stdio)	Compose profile `lsp`

All targets: Python 3.12-slim, non-root `appuser` (uid 1000), `API_KEY_STORE=/data/api_keys.json`, `MPLBACKEND=Agg`, healthcheck via `curl` (HTTP targets).

Buildx Bake (`docker-bake.hcl`)

```
# Local load (host platform)
docker buildx bake api                # production image → pynescript-api:latest
docker buildx bake                    # default group: api + api-dev
docker buildx bake all                # api + api-dev + lsp

# Multi-platform (amd64 + arm64). Set REGISTRY to push:
REGISTRY=gcr.io/$PROJECT/pynescript TAG=$COMMIT_SHA docker buildx bake release
```

Make wrappers:

```
make docker-build       # bake api
make docker-build-all   # bake all targets
make docker-buildx      # bake release (multi-platform)
```

One-time multi-platform builder (if needed):

```
docker buildx create --use --name pynescript
```

Compose

```
cp .env.example .env   # optional overrides

make docker-up         # api-dev on host :5002
make docker-up-full    # + redis profile (api + redis)
make docker-prod       # requires ADMIN_TOKEN; gunicorn, no source mounts
make docker-smoke      # curl http://127.0.0.1:5002/
make docker-down

# stdio LSP (not a long-running compose service)
docker compose --profile lsp run --rm lsp
```

Service	Profile	Notes
`api`	default	Port `${API_PORT:-5002}:8080`, ro mounts of `src/` + `backend/`, volume `api_data` → `/data`
`redis`	`redis`	redis:7-alpine, AOF, `${REDIS_PORT:-6379}`
`lsp`	`lsp`	stdio; use `docker compose run --rm lsp` (not detached `up`)

Network name: `pynescript-dev`.

Production overlay (`docker-compose.prod.yml`): target `api`, **`volumes: !override`** so only `api_data:/data` remains (dev source binds are dropped), SQLite key store on `/data`, mem/cpu caps, requires `ADMIN_TOKEN`.

Environment variables

Variable	Default	Meaning
`PORT`	`8080`	Listen port inside the container
`HOST`	`0.0.0.0`	Bind address (dev Flask runner)
`API_PORT`	`5002`	Host port published to container `8080`
`FLASK_ENV`	`production` / `development`	App mode by target
`STORE_BACKEND`	`json` (prod overlay: `sqlite`)	`json` | `sqlite` | `redis`
`API_KEY_STORE`	`/data/api_keys.json`	JSON key store path (`STORE_BACKEND=json`)
`API_KEY_STORE_SQLITE`	`/data/api_keys.db`	SQLite path (`STORE_BACKEND=sqlite`)
`ALLOWED_ORIGINS`	see `.env.example`	CORS allow-list
`ADMIN_TOKEN`	unset	Required for `POST /auth/create_key` (`X-Admin-Token`) and prod compose
`REDIS_URL`	empty in prod; compose dev default `redis://redis:6379/0`	Required when `STORE_BACKEND=redis`
`GUNICORN_WORKERS` / `GUNICORN_THREADS` / `GUNICORN_TIMEOUT`	`2` / `4` / `60`	Prod entrypoint knobs
`GUNICORN_BIND`	`0.0.0.0:${PORT}`	Optional gunicorn bind override

Cloud Build image tags

From `cloudbuild.yaml` (builds **`--target api`**):

```
gcr.io/$PROJECT_ID/pynescript/pynescript-pro-api:$COMMIT_SHA
gcr.io/$PROJECT_ID/pynescript/pynescript-pro-api:latest
```

Internals

Build stages

1. **base** — slim image, `curl` + freetype/png for matplotlib Agg, `appuser`, `/data`
2. **builder** — `build-essential`, pip cache mounts, install `backend/requirements.txt` then `pip install ".[lsp]"` into `/install` (requires `pyproject.toml`, `README.md`, `LICENSE`, `NOTICE`, `src/`, `backend/`)
3. **runtime** — copy prefix + sources, `PYTHONPATH=/app/src:/app` so compose mounts override site-packages
4. **api / api-dev / lsp** — final CMDs and healthchecks

`.dockerignore`

Keeps context small by excluding `tests/` (corpus), `docs/`, `brand/`, node modules, compiler `*.nbc` / `*.nbi` caches, venvs, and IDE cruft.

Compose healthchecks

API: `curl -fsS http://127.0.0.1:8080/`. Redis: `redis-cli ping`.

Invariants & edge cases

1. **Host port 5002 vs container 8080.** AXIS local docs assume API on 5002 via compose; Cloud Run uses 8080.
2. **Read-only mounts in compose** mean dependency changes need image rebuild; source edits are visible via `PYTHONPATH` precedence.
3. **Production target is immutable** — no live mounts; use the prod overlay or Cloud Run.
4. **Non-root `appuser`** cannot write arbitrary paths; keep `API_KEY_STORE` under `/data`.
5. **Cloud Build always builds `--target api`** (gunicorn). Do not point it at `api-dev`.
6. **LSP is stdio** — not an HTTP service; use `docker compose run --rm lsp` rather than expecting a published port.

Worked examples

Production-like local API (bake)

```
docker buildx bake api
docker run --rm -p 8080:8080 \
  -e FLASK_ENV=production \
  -e ALLOWED_ORIGINS=http://localhost:8081 \
  pynescript-api:latest
curl -s http://127.0.0.1:8080/ | jq .
```

Compose with Redis

```
docker compose --profile redis up --build
```

Optional LSP container

```
docker compose --profile lsp run --rm lsp
```

Production overlay

```
docker compose -f docker-compose.yml -f docker-compose.prod.yml up --build -d
```

Failure modes

Symptom	Cause	Fix
Healthcheck fails	process not listening / curl missing	Use image targets from this Dockerfile; check `docker compose logs api`
Connection refused on host port	Flask bound to `127.0.0.1`	Dev target sets `HOST=0.0.0.0`; do not override to localhost
Permission denied writing keys	Running as `appuser` outside `/data`	Set `API_KEY_STORE=/data/...` and mount `api_data`
Import errors after mount	PYTHONPATH / stale site-packages	`PYTHONPATH=/app/src:/app`; rebuild after dependency changes
OOM under load	Small memory cap	Raise `API_MEM_LIMIT` / Cloud Run memory; reduce concurrency
CORS failures from PWA	Origin not allowed	Set `ALLOWED_ORIGINS`
Huge build context	Missing `.dockerignore`	Ensure `.dockerignore` is present (corpus/docs excluded)

See also

- GCP
- Security
- Pro API lifecycle
- Local development

NUITKA BUILD

title: "Nuitka LSP build" description: "Compile pynescript-lsp with Nuitka — onefile vs standalone, CI build script, Anaconda quirks, and artifact layout."

NUITKA LSP BUILD

Abstract

The Language Server can run as pure Python (`python -m pynescript.langserver`) or as a **Nuitka-compiled binary** for editor distribution. Compilation freezes the pygls server, providers, and (when configured) encrypted builtin metadata into a deployable artifact under `dist/`.

Primary tools: `scripts/build/compile.py` (local) and `scripts/build/ci_build.py` (CI/Cloud Build).

Conceptual model

```
flowchart TB
  META[generate_builtin_metadata.py] --> JSON[builtin_metadata.json]
  JSON --> ENC[Fernet encrypt → .enc + .sha256]
  JSON --> PLAIN[plaintext for dev]
  SRC[src/pynescript/langserver] --> NUITKA[nuitka --onefile|--standalone]
  ENC --> NUITKA
  NUITKA --> BIN[dist/…/pynescript-lsp]
  BIN --> VSIX[optional VSIX bundle]
```

Interface surface

Commands

```
# Prerequisites
pip install nuitka cryptography

# Full onefile build (default path via Make)
make build
# ≡ python scripts/build/compile.py --jobs=4

# Fast import check only (~30s)
make build-check
# ≡ python scripts/build/compile.py --check

# Explicit modes
python scripts/build/compile.py --standalone
python scripts/build/compile.py --onefile --jobs 8
python scripts/build/ci_build.py --jobs=4
```

Build modes (approx. wall time)

Mode	Time	Notes
`--check`	~30s	Import resolution / no full compile
`--standalone`	5–15 min	Directory distribution, faster iteration
`--onefile`	10–30 min	Self-extracting single binary
CI 8 cores	5–15 min	High parallelism

Expected layout

```
dist/
├── lsp/
│   └── pynescript-lsp          # onefile (path may vary by script)
├── vsix/
│   └── pynescript-lsp.vsix     # optional bundle
└── pynescript-lsp              # ci_build final move target (script-dependent)
```

Also see `scripts/build/README.md`.

Internals

Key Nuitka flags (`compile.py`)

- `--static-libpython=no` — default; critical for many Linux/Anaconda envs
- `--python-flag=no_site,no_docstrings`
- `--include-data-dir=…/providers=pynescript/langserver/providers`
- `--lto=auto` , `--jobs=N`
- `--onefile` or `--standalone`
- `--follow-imports` on local compile path

Entry module

Compilation targets `src/pynescript/langserver` (package entry), product name `pynescript-lsp` .

Metadata stage

Before compile, scripts may:

1. Run `scripts/generate_builtin_metadata.py` if plaintext metadata missing
2. Fernet-encrypt to `builtin_metadata.json.enc`
3. Write truncated SHA256 sidecar
4. Write `scripts/build/.metadata.key` (gitignored) or consume `CRYPTO_KEY` env in CI

Anaconda

If static libpython is unavailable:

```
conda install libpython-static
# or rely on --static-libpython=no (already the default in compile.py)
```

Invariants & edge cases

1. **C compiler required** (gcc/clang/MSVC). Containers need `build-essential` or equivalent if compiling inside Docker.
2. **Python 3.10+**; release workflow pins **3.12**.
3. **Providers data dir must ship** — completion/hover collapse without metadata.
4. **Onefile startup cost** includes extract-to-temp; standalone is snappier for local soak tests.
5. **Do not commit** `scripts/build/.metadata.key` .

Worked examples

Local onefile with explicit jobs

```
python scripts/build/compile.py --jobs "$(nproc)"
find dist -name 'pynescript-lsp*' -type f
```

CI-shaped build

```
python scripts/build/ci_build.py --jobs=4
```

Verify without compile

```
python scripts/build/compile.py --check
```

Failure modes

Symptom	Cause	Fix
Link error on libpython	Anaconda static lib missing	Install static lib or keep `--static-libpython=no`
Binary missing builtins	Data dir not included	Confirm `--include-data-dir` providers path
Huge binary	Full stdlib pull-in	Review Nuitka plugins / unused deps
CI timeout	Low cores / onefile	Raise jobs; use larger runner; prefer standalone intermediate
Decrypt fail at runtime	Key not embedded / env unset	See Metadata crypto

See also

- Metadata crypto
- Release
- LSP builtin metadata

METADATA CRYPTO

title: "Metadata encryption" description: "Fernet-encrypted LSP builtin_metadata.json — key lifecycle, CI secrets, integrity hashes, and runtime decrypt."

METADATA ENCRYPTION

Abstract

LSP completion, hover, and signature help depend on a large **builtin metadata** document derived from the live evaluator surface. In open development the JSON is plaintext and git-tracked. For **compiled Nuitka binaries**, the build pipeline can ship an encrypted blob so the binary does not leave a trivially scrapable metadata file on disk.

Mechanism: **Fernet** (symmetric, `cryptography` package) + optional SHA256 integrity sidecar.

Conceptual model

```
flowchart LR
  GEN[generate_builtin_metadata.py] --> PLAIN[providers/builtin_metadata.json]
  PLAIN --> FERNET[Fernet.encrypt]
  KEY[.metadata.key or CRYPTO_KEY] --> FERNET
  FERNET --> ENC[builtin_metadata.json.enc]
  PLAIN --> SHA[builtin_metadata.json.sha256]
  ENC --> LOAD[metadata_decrypt.load_encrypted_metadata]
  KEY --> LOAD
  SHA --> LOAD
  PLAIN --> DEV[get_metadata_cached prefers plaintext]
```

Interface surface

Files

Path	Tracked?	Role
`src/pynescript/langserver/providers/builtin_metadata.json`	yes	Plaintext for dev / hatch
`…/builtin_metadata.json.enc`	often yes	Encrypted payload for binaries
`…/builtin_metadata.json.sha256`	yes	Truncated SHA256 of plaintext
`scripts/build/.metadata.key`	**no** (gitignored)	Local Fernet key material
`scripts/generate_builtin_metadata.py`	yes	Regenerates JSON from code

Environment variables

Variable	Context	Meaning
`CRYPTO_KEY`	Build scripts / CI / Cloud Build	Key bytes for encrypt stage
`PYNESCRIPT_METADATA_KEY`	Runtime decrypt fallback	Env-supplied key if file missing
GitHub `secrets.METADATA_KEY`	Actions	Stable secret → `CRYPTO_KEY`
Cloud Build `_METADATA_KEY`	`cloudbuild.yaml` substitution	`CRYPTO_KEY=${_METADATA_KEY}`

Code entrypoints

- Encrypt stage: `scripts/build/compile.py` (`encrypt_metadata`), `scripts/build/ci_build.py` (`stage_metadata`)
- Decrypt: `src/pynescript/langserver/providers/metadata_decrypt.py`
- Loader preference: plaintext if present, else encrypted (`get_metadata_cached`)

Internals

Generation (always code-derived)

```
python scripts/generate_builtin_metadata.py
```

Do **not** hand-edit the JSON for permanent feature work — re-run the generator after adding builtins.

Encrypt (local sketch)

```
from cryptography.fernet import Fernet
key = Fernet.generate_key()  # or load stable secret
fernet = Fernet(key)
encrypted = fernet.encrypt(plaintext_bytes)
```

Build scripts chmod the key file to `0o600` when writing `.metadata.key`.

Decrypt integrity check

After Fernet decrypt, if a `.sha256` sidecar exists, the loader compares `sha256(plaintext)[:16]` and raises on mismatch.

Dev vs binary

Mode	Behavior
Editable install / repo checkout	Prefer plaintext JSON
Nuitka onefile	Encrypted data dir + key resolution via MEIPASS / env
Missing both	`FileNotFoundError` directing to compile script

Invariants & edge cases

1. **Stable key ⇒ reproducible `.enc`.** Random key every CI run yields a different ciphertext even if plaintext is identical — not a functional bug, but pollutes diffs and caches.
2. **Plaintext remains in git** for open-source transparency of the *language surface*. Encryption protects the *bundled binary layout*, not the secret of the API catalog.
3. **Truncated hash is integrity, not secrecy.** It detects bitrot / wrong plaintext, not attackers with the key.
4. **Never commit `.metadata.key`.** Rotate if leaked; regenerate `.enc` with the new key for binary builds.
5. **After adding builtins:** regenerate metadata, re-encrypt with the **same** CI key, commit updated JSON (and `.enc` if tracked).

Worked examples

Regenerate + local encrypt

```
python scripts/generate_builtin_metadata.py
pip install cryptography
python scripts/build/compile.py --check   # stages metadata paths as configured
```

CI snippet

```
- name: Build LSP binary
  run: python scripts/build/ci_build.py --jobs 4
  env:
    CRYPTO_KEY: ${{ secrets.METADATA_KEY }}
```

Runtime with env key

```
pynescript-lsp
```

Failure modes

Symptom	Cause	Fix
`No metadata decryption key found`	Missing file + env	Supply `PYNESCRIPT_METADATA_KEY` or rebuild with key
`Metadata integrity check failed`	Stale hash or wrong ciphertext	Re-encrypt from current plaintext; commit both
Completion empty in binary only	Data dir not included in Nuitka	Fix `--include-data-dir`
Divergent `.enc` every CI	Unset `CRYPTO_KEY`	Configure `METADATA_KEY` secret

See also

- Nuitka build
- LSP builtin metadata
- Release
- Security

GCP

title: "GCP & Cloud Run" description: "Cloud Build pipeline, Cloud Run Pro API sizing, cost model, and substitution secrets for PYNE deployments."

GCP & CLOUD RUN

Abstract

Production-shaped hosting for the Pro API targets **Google Cloud Run**, fed by **Cloud Build** (`cloudbuild.yaml`). The pipeline builds a container, pushes multi-tags to Container Registry, deploys a managed service, and can optionally compile the LSP binary in the same build graph.

This page codifies the as-checked-in config and the cost envelope from `docs/gcp_cost_estimate.md` — treat dollar figures as planning estimates, not invoices.

Conceptual model

```
flowchart LR
  GIT[push main] --> CB[Cloud Build]
  CB --> DOCKER["docker build --target api"]
  DOCKER --> GCR[gcr.io project/repo/service]
  GCR --> RUN[Cloud Run deploy]
  CB --> LSP[optional Nuitka ci_build]
  RUN --> SQL[(Cloud SQL optional)]
  RUN --> REDIS[(Memorystore optional)]
  RUN --> GCS[Cloud Storage charts]
```

Interface surface

Substitutions (`cloudbuild.yaml`)

Substitution	Default	Meaning
`_SERVICE_NAME`	`pynescript-pro-api`	Cloud Run service
`_REGION`	`us-central1`	Deploy region
`_REPOSITORY`	`pynescript`	Image path segment
`_METADATA_KEY`	(secret/subst)	Fernet key → `CRYPTO_KEY` for LSP stage

Build steps

1. **build** — `docker build` with BuildKit; tags `$COMMIT_SHA` + `latest` ; cache-from `latest`
2. **push** — `docker push --all-tags`
3. **deploy** — `gcloud run deploy` with managed platform flags
4. **build-lsp** — `python:3.12-slim` image runs `scripts/build/ci_build.py --jobs=4` with `CRYPTO_KEY=${_METADATA_KEY}`

Cloud Run flags (as configured)

Flag	Value	Rationale
`--allow-unauthenticated`	on	Public HTTP API (auth via API keys at app layer)
`--min-instances`	0	Scale to zero
`--max-instances`	10	Cap spend / blast radius
`--memory`	512Mi	Evaluator headroom
`--cpu`	1	Single vCPU per instance
`--concurrency`	10	Parallel requests per instance
`--timeout`	60s	Aligns with gunicorn timeout
`--set-env-vars`	`FLASK_ENV=production`	App mode

Machine / budget knobs

```
options:
  machineType: E2_HIGHCPU_8
  logging: CLOUD_LOGGING_ONLY
timeout: 1200s
```

Internals

Path	Role
`cloudbuild.yaml`	Pipeline definition
`Dockerfile` target `api`	Production image recipe used by Cloud Build
`backend/app.py`	Health `/`, CORS, body size limit
`docs/gcp_cost_estimate.md`	Scenario costs (April 2026 draft)

Cost envelope (planning)

Scale	Rough monthly	Notes
Free / hobby	~$0	Within Cloud Run free tier if tiny
MVP (~10 pro users)	~$50	Cloud Run + micro SQL
Growth (~50)	~$120	Add Redis caching
Scale (~200)	~$350	Dominated by evaluator CPU-sec

Dominant cost driver: **script execution CPU time** (~2–5 CPU-sec per typical backtest over 1k bars). Memory ~200–400MB per concurrent Python worker.

Free-tier leverage (GCP)

- Cloud Run: 2M requests, 400K CPU-sec, 200K GB-sec / month
- Cloud Build: 120 build-minutes / day
- Artifact Registry / GCS small free slices

Invariants & edge cases

1. **App-layer auth still required** when Cloud Run allows unauthenticated invoke — see Security.
2. **Scale-to-zero cold starts** add latency; min-instances=1 trades money for snappiness.
3. **LSP build step does not deploy the binary** to Cloud Run; it is a release-adjacent compile in the same YAML.
4. **512Mi may be tight** for large OHLCV payloads — watch OOM kills; raise memory before raising concurrency.
5. **Cost estimate doc** also lists Railway / Render / Fly / self-host alternatives if GCP is overkill.

Worked examples

Manual deploy sketch (operator)

```
gcloud builds submit --config cloudbuild.yaml \
  --substitutions=_METADATA_KEY="$METADATA_KEY"
```

Smoke health after deploy

```
curl -s "https://SERVICE-URL/"
# expect JSON status healthy, service pynescript-pro-api
```

Failure modes

Symptom	Cause	Fix
Deploy timeout	Nuitka step + docker on 20 min budget	Split LSP compile to release pipeline
429 / queue	max-instances or concurrency	Raise caps carefully; add queue
High bill	No cache; chatty clients	Redis TTL; Cloud Tasks for backtests
Auth bypass illusion	Unauthenticated Cloud Run	Enforce API keys in Flask
Image pull errors	Wrong project / registry perms	IAM on GCR + Cloud Run service agent

See also

- Docker
- Observability
- Security
- Pro API

OBSERVABILITY

title: "Observability" description: "Health endpoints, gunicorn process model, Cloud Logging, coverage artifacts, and operational signals for PYNE."

OBSERVABILITY

Abstract

PYNE's production surface today optimizes for **simple, hostable signals** rather than a full OpenTelemetry mesh: HTTP health checks, process managers (gunicorn), cloud platform logs, CI artifacts (coverage, Playwright reports), and application-level API key usage counters. Treat this page as the map of *what exists in-repo*, not a claim of enterprise APM parity.

Conceptual model

```
flowchart TB
  CLIENT[Client] --> LB[Cloud Run / local]
  LB --> GUNI[gunicorn workers]
  GUNI --> APP[Flask backend.app]
  APP --> HEALTH["GET /"]
  APP --> RUN["POST /run …"]
  APP --> KEYS[APIKeyStore counters]
  GUNI --> LOGS[stdout / Cloud Logging]
  CI[GitHub Actions] --> ART[coverage lcov / playwright report]
```

Interface surface

Health

`backend/app.py` exposes:

```
GET /
→ { "status": "healthy", "service": "pynescript-pro-api", … }
```

Used by:

- `Dockerfile` target `api` `HEALTHCHECK` (`curl -fsS http://127.0.0.1:8080/`)
- `docker-compose.yml` api healthcheck
- Load balancers / Cloud Run startup-ish probes (platform-dependent)

Process model (production)

```
gunicorn --bind :8080 --workers 2 --threads 4 --timeout 60 backend.app:app
```

Knob	Effect on signals
workers	Process isolation; multiplies memory
threads	Concurrent requests within worker
timeout 60s	Aligns with Cloud Run `--timeout=60s` — long evals die predictably

Logs

Environment	Where logs go
Local `make run`	Process stdout/stderr
Docker	Container logs (`docker logs`, compose)
Cloud Build	`logging: CLOUD_LOGGING_ONLY`
Cloud Run	Cloud Logging (request + container)

Application modules use standard library / Flask logging patterns; there is no mandatory structured-log schema enforced repo-wide. `` ` `` if a future branch introduces OpenTelemetry exporters.

CI / quality signals

Signal	Source
Ruff / mypy	`ci.yml` lint job
Pytest results	LSP + backend jobs
Codecov (optional)	Python 3.13 matrix cell
AXIS coverage lcov	`axis-coverage-lcov` artifact
Playwright report	Uploaded on nightly/e2e failure
Security tests	`bun run test:security` (PWA job + nightly)

Usage metering (app-layer)

`backend/middleware/auth.py` tracks per-key:

- `calls_used` / `calls_limit` / tier
- `last_used` timestamps

These are **business metrics** for rate limits, not Prometheus time series — export them if you need dashboards.

Internals

Path	Role
`backend/app.py`	Health route, CORS, `MAX_CONTENT_LENGTH`
`backend/middleware/auth.py`	Key store, rate limit counters
`backend/services/backtest.py`	Backtest metrics objects for API responses
`Dockerfile` (`api` target)	HEALTHCHECK + gunicorn entrypoint
`cloudbuild.yaml`	Cloud Logging-only build logs
`logs/` (repo)	Local combined/error log files if generated by tools — not the production contract

Invariants & edge cases

1. **Health ≠ readiness of evaluator warm caches.** A 200 on `/` does not mean the first `/run` will be fast (cold import / JIT / numba).
2. **Timeouts double-bind.** Gunicorn 60s and Cloud Run 60s — the tighter effective limit is still 60s.
3. **Scale-to-zero hides idle metrics.** No traffic ⇒ no samples; use synthetic uptime checks if SLOs matter.
4. **Coverage artifacts expire** (7–14 days retention on Actions uploads).
5. **Do not scrape secrets from logs.** API keys and admin tokens must never be logged at info level.

Worked examples

Local health loop

```
make run
curl -s http://127.0.0.1:5002/ | python -m json.tool
```

Docker health

```
docker inspect --format='{{json .State.Health}}' CONTAINER
```

Tail Cloud Run (gcloud)

```
gcloud run services logs read pynescript-pro-api --region us-central1 --limit 50
```

Failure modes

Symptom	Cause	Fix
Health green, `/run` 500	Exception in evaluator path	Inspect request logs; reproduce with payload
Worker kills	Soft timeout	Optimize script / raise timeout consistently
Missing CI artifact	Path wrong / tests passed	`if-no-files-found: ignore` hides absence
Rate limit false positives	Shared key / limit too low	Adjust tier limits; multi-key
Silent deploy failure	Build logging-only + unread	Check Cloud Build history UI

See also

- GCP
- Docker
- Security
- API lifecycle

SECURITY

title: "Security" description: "API keys, CORS, body limits, secrets hygiene, Fernet metadata keys, and threat-aware defaults for PYNE ops."

SECURITY

Abstract

PYNE security is layered: **transport/hosting** (Cloud Run, containers), **application** (API keys, admin tokens, CORS, body size caps), **build secrets** (Fernet metadata keys, VSCE PAT), and **supply chain** (CI dependency installs, extension packaging). This page documents controls that exist in code and workflows — not a full threat model certification.

Conceptual model

```
flowchart TB
  NET[Internet] --> CR[Cloud Run allow-unauthenticated]
  CR --> CORS[flask-cors allowlist]
  CORS --> SIZE[MAX_CONTENT_LENGTH 5MB]
  SIZE --> AUTH[require_api_key]
  AUTH --> STORE[APIKeyStore hash + tiers]
  AUTH --> RUN[Evaluator /run]
  ADMIN[X-Admin-Token] --> KEYMGMT[key minting routes]
  BUILD[CI secrets] --> META[METADATA_KEY]
  BUILD --> VSCE[VSCE_PAT]
```

Interface surface

HTTP application controls (`backend/app.py`)

Control	Default / mechanism
Body size	`MAX_CONTENT_LENGTH = 5 * 1024 * 1024` (5 MiB) — DoS/OOM mitigation
CORS	`ALLOWED_ORIGINS` env (comma-separated); default includes product hosts + localhost regex
Methods	GET, POST only via flask-cors config
Headers	`Content-Type` , `Authorization` , `X-Admin-Token`
Credentials	`supports_credentials=False`

API keys (`backend/middleware/auth.py`)

- Keys stored as **hashes**, not raw secrets (store path via `API_KEY_STORE`)
- Tiers: `free` , `hobby` (5k), `pro` (50k), `team` (200k), `enterprise` (∞)
- Rate limit by monthly call counters (`calls_used` / `calls_limit`)
- Decorators: `require_api_key` , `require_admin_token`

Build / release secrets

Secret	Purpose
`METADATA_KEY` / `_METADATA_KEY`	Stable Fernet key for LSP metadata encrypt
`VSCE_PAT`	VS Code Marketplace / vsce package
`GITHUB_TOKEN`	Release asset upload
`PYNESCRIPT_METADATA_KEY`	Runtime decrypt env fallback

Container hardening (`Dockerfile` target `api`)

- Multi-stage build (no compiler toolchain in final image)
- Non-root `appuser` (uid 1000)
- Production env flags + gunicorn entrypoint
- Healthcheck without privileged ops

- Intended mutable path `/data` (key store); prod compose uses SQLite there for multi-worker

Admin minting

- `POST /auth/create_key` requires `ADMIN_TOKEN` env **and** matching `X-Admin-Token` header (constant-time compare)
- Unset `ADMIN_TOKEN` → 403 (fail closed, not open access)

Key store backends (`STORE_BACKEND`)

Value	Use
`json` (default)	Single-process / tests; file may hold raw keys
`sqlite`	Multi-worker single host (shared volume); **hash-only**
`redis`	Multi-replica; needs `REDIS_URL`; **hash-only**

Internals

Path	Role
`backend/app.py`	CORS, body limit, blueprint registration
`backend/middleware/auth.py`	Key store selection, tiers, decorators, admin gate
`backend/middleware/key_store_redis.py` / `key_store_sqlite.py`	Hash-only shared stores
`backend/middleware/schemas.py`	Request validation schemas
`src/pynescript/langserver/providers/metadata_decrypt.py`	Encrypted metadata load
`.github/workflows/*.yml`	Secret consumption
`frontend` security tests	`bun run test:security` in CI

Audit notes encoded in source

Comments in `app.py` reference a **2026-07-05 audit** (findings S1 body size, S7/S8 CORS). Prefer reading the code over assuming defaults never change.

Invariants & edge cases

1. **Cloud Run `--allow-unauthenticated` does not mean open data.** It means IAM does not gate invoke; the Flask app must.
2. **Free tier with `calls_limit=0` is special-cased** as unlimited remaining in some helpers — verify tier semantics before production billing.
3. **Localhost CORS regex** permits any port on localhost/127.0.0.1 — fine for desk, wrong for prod `ALLOWED_ORIGINS`.
4. **Fernet protects binary metadata packaging**, not user script confidentiality.
5. **Admin token routes are high privilege** — rotate and store separately from user API keys.
6. **Never log raw API keys.** Hash lookups only.

Worked examples

Harden CORS for a single origin

```
python -m backend.app
```

Point key store at a volume

```
# ensure process user can write /data
```

Rotate metadata key (ops procedure sketch)

1. Generate new Fernet key; store as secret `METADATA_KEY`
2. Rebuild LSP with `CRYPTO_KEY` set
3. Ship new binary; retire old key after client upgrade window
4. Do not leave dual-key decrypt unless you implement it

Failure modes

Symptom	Cause	Fix
Browser CORS error	Origin not listed	Set `ALLOWED_ORIGINS`
413 Payload Too Large	Body > 5MB	Reduce OHLCV batch; raise limit knowingly
401/403 on `/run`	Missing/invalid key	Check `Authorization` header scheme
Rate limited	Tier exhausted	Upgrade tier / wait reset
Secret in fork PR logs	Misconfigured workflow echo	Never `echo` secrets; use masked env
Root-owned files in volume	Old image ran as root	Align UID with `appuser`

See also

- Auth and keys (API)
- Metadata crypto
- Docker
- GCP