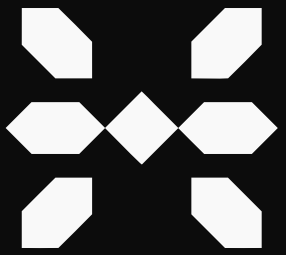
HOOX · AXIS · DOCUMENTATION



PLATFORM ENGINEERS · DEPLOYERS

DEVOPS MANUAL

Local dev, build/serve, Cloudflare, VPS, CORS, CI

Open charting PWA — own the axes, swap the engine

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

CONTENTS

OVERVIEW

COPYRIGHT (C) 2024-2026 JANGO_BLOCKCHAINED
THIS FILE IS PART OF PYNESCRIPT.
PYNESCRIPT IS FREE SOFTWARE: YOU CAN REDISTRIBUTE IT AND/OR MODIFY
IT UNDER THE TERMS OF THE GNU AFFERO GENERAL PUBLIC LICENSE AS PUBLISHED BY
THE FREE SOFTWARE FOUNDATION, EITHER VERSION 3 OF THE LICENSE, OR
(AT YOUR OPTION) ANY LATER VERSION.
PYNESCRIPT IS DISTRIBUTED IN THE HOPE THAT IT WILL BE USEFUL,
BUT WITHOUT ANY WARRANTY; WITHOUT EVEN THE IMPLIED WARRANTY OF
MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. SEE THE
GNU AFFERO GENERAL PUBLIC LICENSE FOR MORE DETAILS.
YOU SHOULD HAVE RECEIVED A COPY OF THE GNU AFFERO GENERAL PUBLIC LICENSE
ALONG WITH PYNESCRIPT. IF NOT, SEE
HTTPS://WWW.GNU.ORG/LICENSES/.
SPDX-LICENSE-IDENTIFIER: AGPL-3.0-ONLY

title: "DevOps" description: "Build, serve, Cloudflare, VPS demo, CORS, and CI for the AXIS PWA and Worker."

DEVOPS

Abstract

AXIS DevOps spans three runnable surfaces:

1. **Vite + Solid PWA** (`frontend/`) — product UI
2. **Static server** (`axis_pwa_server.py` or Vite preview) — production-like assets
3. **Cloudflare Worker** (`frontend/worker/`) — edge API / WS

Python evaluation for desk workflows still often uses the **Flask Pro API** (`make run` on `:5002`). The Worker proxies there when `EXTERNAL_BACKEND` is set.

Conceptual model

```
flowchart LR
  DEV[Vite :3000]
  STATIC[axis_pwa_server :8081]
  FLASK[Flask :5002]
  WRK[Worker :8787]
  CF[(CF Pages + Worker)]
  DEV --> FLASK
  STATIC --> FLASK
  STATIC --> WRK
  WRK --> FLASK
  CF --> FLASK
```

Topology cheat sheet

Mode	Command sketch	Ports
Dev AXIS	`cd frontend && bun run dev`	Vite (often `:3000`)
Dev + Flask	+ `make run`	`:5002`
Static dist	`bun run build` + `python axis_pwa_server.py`	`:8081`
Worker	`cd frontend/worker && npm run dev`	`:8787`
CF prod	Pages `pynescript-axis` + Worker same name family	HTTPS

Page map

Page	Topic
Local dev	Day-to-day loop
Build and serve	Vite build, SPA server, assets
Cloudflare	Pages + Worker deploy
VPS demo	Single-box static + Flask
CORS and origins	Allowed origins matrix
CI and testing	GitHub Actions, coverage, e2e

Frozen names

Surface	Value
CF project	`pynescript-axis`
Health service	`pynescript-axis-worker`

See also

- Worker
- Testing reference
- AXIS hub

LOCAL DEV

COPYRIGHT (C) 2024-2026 JANGO_BLOCKCHAINED
THIS FILE IS PART OF PYNESCRIPT.
PYNESCRIPT IS FREE SOFTWARE: YOU CAN REDISTRIBUTE IT AND/OR MODIFY
IT UNDER THE TERMS OF THE GNU AFFERO GENERAL PUBLIC LICENSE AS PUBLISHED BY
THE FREE SOFTWARE FOUNDATION, EITHER VERSION 3 OF THE LICENSE, OR
(AT YOUR OPTION) ANY LATER VERSION.
PYNESCRIPT IS DISTRIBUTED IN THE HOPE THAT IT WILL BE USEFUL,
BUT WITHOUT ANY WARRANTY; WITHOUT EVEN THE IMPLIED WARRANTY OF
MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. SEE THE
GNU AFFERO GENERAL PUBLIC LICENSE FOR MORE DETAILS.
YOU SHOULD HAVE RECEIVED A COPY OF THE GNU AFFERO GENERAL PUBLIC LICENSE
ALONG WITH PYNESCRIPT. IF NOT, SEE
HTTPS://WWW.GNU.ORG/LICENSES/.
SPDX-LICENSE-IDENTIFIER: AGPL-3.0-ONLY

title: "Local development" description: "Day-to-day AXIS loop: Vite, Flask, Worker wrangler, endpoints, and plugin examples."

LOCAL DEVELOPMENT

Abstract

Local AXIS development is a **multi-process** topology. AXIS is a Solid app under Vite; evaluation usually hits Flask; optional cloud/storage/stream features hit the Worker.

Conceptual model

```
flowchart TB
  subgraph Browser
    UI[Vite HMR app]
  end
  UI -->|engine server| FLASK[Flask :5002]
  UI -->|cloud / run proxy| WRK[wrangler :8787]
  WRK --> FLASK
  UI -->|optional direct| VENUE[Venue REST/WS]
```

Interface surface — commands

Frontend (preferred product path)

```
cd frontend
bun install
bun run dev          # Vite + Solid → typically http://127.0.0.1:3000
```

Flask Pro API (engine backend)

From repo root:

```
make run             # python -m backend.app → :5002
# or hatch / documented backend entry
```

Set PWA endpoint to `http://127.0.0.1:5002` for the **server** engine.

Worker

```
cd frontend/worker
npm install          # or bun
npm run dev          # http://127.0.0.1:8787
```

Point cloud storage / Worker endpoint at `http://127.0.0.1:8787`. For `/api/run` proxy, set Worker `EXTERNAL_BACKEND=http://127.0.0.1:5002` (works on local wrangler; not on CF edge).

Makefile helpers

Target	Intent
`make run-frontend`	Historical Bun static server (`frontend/server.ts`) on `:8081` — **legacy shell path**
`make worker-dev`	Wrangler dev
`make worker-install`	Install worker deps
`make test-frontend`	Bun unit tests under `frontend/tests/`

For the **Solid** product, prefer `bun run dev` / `bun run build` over the legacy `server.ts` unless you are debugging the old tree (legacy shell).

Internals — useful paths

Path	Role
`frontend/src/index.tsx`	App entry
`frontend/vite.config.*`	Vite / Solid plugin
`frontend/src/engines/catalog.ts`	Default engine endpoint
`frontend/worker/wrangler.toml`	Local vars
`frontend/public/plugins/`	Example modules served at `/plugins/`

Worked loop

1. Terminal A: `make run` (Flask).
2. Terminal B: `cd frontend && bun run dev`.
3. Open Vite URL; set engine **Server-Side**, endpoint `http://127.0.0.1:5002`.
4. Load `mock-walk` if venues are blocked.
5. Optional Terminal C: Worker with `ALLOW_OPEN_KEYS=1` for cloud library experiments.

Load an example plugin

With Vite or static server:

```
http://127.0.0.1:3000/plugins/example-coingecko-source.js
```

Manager → Plugins → Load from URL.

Invariants & edge cases

1. **CORS** — Flask must allow the Vite origin; Worker `pickOrigin` allows `:3000` and `:8081`.
2. **Pyodide local** — needs vendor wheels under `public/` (copied to dist on build).
3. **Two endpoints** — topbar endpoint for engine vs cloud storage plugin config can differ (Flask vs Worker).
4. **Legacy Makefile message** still mentions AXIS and `:8081` Bun server.

Failure modes

Issue	Fix
Engine not ready	Flask down / wrong port
CORS errors	Align origins; see CORS
Plugin 404	File only under `src/plugins/` not `public/plugins/`
Worker scripts 401	Set open keys or mint `pn_` key

See also

- Build and serve

- Worker local
- Plugin examples

BUILD AND SERVE

COPYRIGHT (C) 2024-2026 JANGO_BLOCKCHAINED
THIS FILE IS PART OF PYNESCRIPT.
PYNESCRIPT IS FREE SOFTWARE: YOU CAN REDISTRIBUTE IT AND/OR MODIFY
IT UNDER THE TERMS OF THE GNU AFFERO GENERAL PUBLIC LICENSE AS PUBLISHED BY
THE FREE SOFTWARE FOUNDATION, EITHER VERSION 3 OF THE LICENSE, OR
(AT YOUR OPTION) ANY LATER VERSION.
PYNESCRIPT IS DISTRIBUTED IN THE HOPE THAT IT WILL BE USEFUL,
BUT WITHOUT ANY WARRANTY; WITHOUT EVEN THE IMPLIED WARRANTY OF
MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. SEE THE
GNU AFFERO GENERAL PUBLIC LICENSE FOR MORE DETAILS.
YOU SHOULD HAVE RECEIVED A COPY OF THE GNU AFFERO GENERAL PUBLIC LICENSE
ALONG WITH PYNESCRIPT. IF NOT, SEE
HTTPS://WWW.GNU.ORG/LICENSES/.
SPDX-LICENSE-IDENTIFIER: AGPL-3.0-ONLY

title: "Build and serve" description: "Vite production build, dist layout, axis_pwa_server.py SPA rules, and asset caching."

BUILD AND SERVE

Abstract

Production AXIS assets are a **static site** produced by Vite (`bun run build` → `frontend/dist`). Serve them with any static host, Cloudflare Pages, or the repo's `axis_pwa_server.py` SPA-aware server.

Critical requirement: **real binary assets** for Pyodide (`.wasm`, `.whl`, `.zip`) must not be rewritten to `index.html`.

Conceptual model

```
flowchart LR
  SRC[src/** Solid]
  PUB[public/**]
  SRC --> VITE[vite build]
  PUB --> VITE
  VITE --> DIST[dist/]
  DIST --> PAGES[CF Pages]
  DIST --> PWA[axis_pwa_server.py]
  DIST --> PREV[vite preview]
```

Interface surface

package.json scripts

```
"dev": "vite",
"build": "vite build",
"preview": "vite preview",
"test": "bun test tests/ worker/tests/",
"test:coverage:gate": "bun run test:coverage && bun scripts/check-coverage.mjs 95",
"test:security": "bun test tests/security/",
"test:e2e:smoke": "playwright test --grep @smoke",
"test:all": "bun run test:coverage:gate && bun run test:security"
```

axis_pwa_server.py

```
cd frontend
bun run build
python axis_pwa_server.py
# HOST / PORT env; default 0.0.0.0:8081 → dist/
```

Behavior (`frontend/axis_pwa_server.py`):

Rule	Detail
Root	Serves `frontend/dist`
Cache	`/assets/*` → `immutable` long cache; else `no-cache`
SPA fallback	Unknown paths → `/index.html`
Never rewrite	`/assets/` , `/plugins/` , `/vendor/` , `/pyodide/` , or extensions including `.js/.css/.whl/.wasm/.zip/.py/.json…`

This SPA exception list exists because Pyodide + micropip die with opaque `BadZipFile` when HTML is returned for a wheel URL. The engine's `assertZipAsset` detects that class of failure early.

Expected public assets

Path (public → dist)	Purpose
`/plugins/*.js`	Example dynamic plugins
`/vendor/*.whl`	pynescript + antlr wheels
`/pyodide/v0.26.2/**`	Self-hosted Pyodide
`/pyodide/pynescript_runtime.py`	Browser run bridge

Internals

Path	Role
`frontend/package.json`	Build scripts
`frontend/axis_pwa_server.py`	Threading HTTP SPA server
`frontend/src/engines/catalog.ts`	Asset URL expectations
`frontend/LEGACY.md`	Product path vs old shell

Makefile note

`make pages-deploy` may historically deploy the wrong tree — **prefer**:

```
cd frontend && bun run build
wrangler pages deploy dist --project-name=pynescript-axis
```

Worked example — static desk demo

```
cd frontend
bun install
bun run build
PORT=8081 python axis_pwa_server.py
# another terminal
make run   # Flask :5002
```

Open `http://127.0.0.1:8081`, engine endpoint `http://127.0.0.1:5002`.

Invariants & edge cases

1. **Service workers / offline** — product path uses Vite PWA assets; legacy root `sw.js` is not the shipping SW.
2. **Relative pyodide index** — resolves against `location.origin`.
3. **Do not hand-edit dist** — regenerate from build.

Failure modes

Symptom	Cause
Blank routes on refresh	Server lacks SPA fallback
Pyodide HTML wheel error	Fallback rewriting `.whl` or missing vendor
Missing plugins	`public/plugins` not copied (check Vite publicDir)

See also

- Local dev
- Cloudflare
- Engines

CLOUDFLARE

COPYRIGHT (C) 2024-2026 JANGO_BLOCKCHAINED
THIS FILE IS PART OF PYNESCRIPT.
PYNESCRIPT IS FREE SOFTWARE: YOU CAN REDISTRIBUTE IT AND/OR MODIFY
IT UNDER THE TERMS OF THE GNU AFFERO GENERAL PUBLIC LICENSE AS PUBLISHED BY
THE FREE SOFTWARE FOUNDATION, EITHER VERSION 3 OF THE LICENSE, OR
(AT YOUR OPTION) ANY LATER VERSION.
PYNESCRIPT IS DISTRIBUTED IN THE HOPE THAT IT WILL BE USEFUL,
BUT WITHOUT ANY WARRANTY; WITHOUT EVEN THE IMPLIED WARRANTY OF
MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. SEE THE
GNU AFFERO GENERAL PUBLIC LICENSE FOR MORE DETAILS.
YOU SHOULD HAVE RECEIVED A COPY OF THE GNU AFFERO GENERAL PUBLIC LICENSE
ALONG WITH PYNESCRIPT. IF NOT, SEE
HTTPS://WWW.GNU.ORG/LICENSES/.
SPDX-LICENSE-IDENTIFIER: AGPL-3.0-ONLY

title: "Cloudflare deployment" description: "Pages + Worker for AXIS: frozen project pynescript-axis, bindings, deploy order, EXTERNAL_BACKEND reality."

CLOUDFLARE DEPLOYMENT

Abstract

Production AXIS on Cloudflare is a **split deploy**:

Piece	Role
Pages	Static PWA (`frontend/dist`)
Worker	JSON API + WebSocket (`frontend/worker`)

Project id: `pynescript-axis` — frozen infrastructure name (dashboard, wrangler, CI). Do not rename without a full binding/domain migration.

Evaluation truth: Worker `/api/run` **proxies to** `EXTERNAL_BACKEND` unless the gated Pyodide path is fully implemented and enabled. You still need a Python host (VPS Flask, container, etc.) for real Pine fidelity today.

Conceptual model

```
flowchart TB
  USER[Browser]
  USER --> PAGES[CF Pages dist]
  USER --> WRK[CF Worker]
  WRK -->|EXTERNAL_BACKEND| VPS[Flask / PYNE host]
  WRK --> KV[KV]
  WRK --> D1[D1]
  WRK --> DO[SessionDO]
```

Deploy procedure

1. Provision bindings (once)

See Worker bindings:

- KV `API_KEYS`, `USAGE`
- D1 `pynescript` + schema
- Optional R2, Durable Objects

2. Configure secrets / vars

Production checklist:

Setting	Production guidance
`EXTERNAL_BACKEND`	Public HTTPS URL of Flask/PYNE API
`ALLOWED_ORIGIN`	Exact Pages origin (e.g. `https://…pages.dev` or custom)
`ADMIN_TOKEN`	Secret, strong
`ALLOW_OPEN_KEYS`	`"0"` or unset
`PYODIDE_IN_WORKER`	`"disabled"` until wheel pipeline ready

3. Deploy Worker

```
cd frontend/worker
npm install
npx wrangler deploy
```

4. Build & deploy Pages

```
cd frontend
bun install
bun run build
npx wrangler pages deploy dist --project-name=pynescript-axis
```

Worker package also defines:

```
"deploy:pages": "wrangler pages deploy ../dist --project-name=pynescript-axis"
```

5. Point the PWA

- Engine endpoint: Worker URL (if path-mapped to `/run`) **or** Flask URL directly
- Cloud storage endpoint: Worker origin
- Ensure CORS allows Pages origin

Internals

Path	Role
`frontend/worker/wrangler.toml`	Worker name + bindings
`frontend/worker/README.md`	Ops narrative
`frontend/worker/package.json`	deploy scripts
Makefile `worker-deploy` / `pages-deploy`	Convenience (verify Pages path)

Health check

```
curl -sS https://<worker>/health
# service: pynescript-axis-worker
```

Invariants & edge cases

1. **Pages ≠ Worker host** — CORS required unless same-origin routing via custom domain routes.
2. **Localhost EXTERNAL_BACKEND** is useless from the edge.
3. **DO migrations** apply on deploy when uncommented.
4. **Observability** enabled in wrangler for log tails (`wrangler tail`).

Failure modes

Issue	Fix
503 NO_BACKEND	Set reachable EXTERNAL_BACKEND
CORS blocked	ALLOWED_ORIGIN mismatch
SPA HTML for wheels	Pages must serve `/vendor` and `/pyodide` as static files
Stream NO_DO	Enable SessionDO bindings

See also

- Worker
- CORS
- VPS demo (backend co-host)

VPS DEMO

COPYRIGHT (C) 2024-2026 JANGO_BLOCKCHAINED
THIS FILE IS PART OF PYNESCRIPT.
PYNESCRIPT IS FREE SOFTWARE: YOU CAN REDISTRIBUTE IT AND/OR MODIFY
IT UNDER THE TERMS OF THE GNU AFFERO GENERAL PUBLIC LICENSE AS PUBLISHED BY
THE FREE SOFTWARE FOUNDATION, EITHER VERSION 3 OF THE LICENSE, OR
(AT YOUR OPTION) ANY LATER VERSION.
PYNESCRIPT IS DISTRIBUTED IN THE HOPE THAT IT WILL BE USEFUL,
BUT WITHOUT ANY WARRANTY; WITHOUT EVEN THE IMPLIED WARRANTY OF
MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. SEE THE
GNU AFFERO GENERAL PUBLIC LICENSE FOR MORE DETAILS.
YOU SHOULD HAVE RECEIVED A COPY OF THE GNU AFFERO GENERAL PUBLIC LICENSE
ALONG WITH PYNESCRIPT. IF NOT, SEE
HTTPS://WWW.GNU.ORG/LICENSES/.
SPDX-LICENSE-IDENTIFIER: AGPL-3.0-ONLY

title: "VPS demo topology" description: "Single-box demo: static AXIS dist + Flask Pro API (+ optional reverse proxy) without Cloudflare."

VPS DEMO TOPOLOGY

Abstract

A **VPS demo** is the lowest-surprise production-like setup for Pine evaluation: one machine runs Flask (PYNE) and serves the static AXIS build. Optional nginx/caddy terminates TLS and reverse-proxies `/run` same-origin to avoid CORS.

No Worker required. Cloud storage and DO streams will not work without the Worker.

Conceptual model

```
flowchart LR
  U[Users]
  U --> NGX[nginx/caddy :443]
  NGX -->|/| STATIC[dist]
  NGX -->|/run| FLASK[Flask :5002]
```

Interface surface

Processes

Process	Command	Port
Build (CI or once)	`cd frontend && bun run build`	—
Static	`PORT=8081 python axis_pwa_server.py`	8081
API	`make run` / `python -m backend.app`	5002

Same-origin proxy sketch (Caddy)

```
axis.example.com {
  handle /run* {
    reverse_proxy 127.0.0.1:5002
  }
  handle {
    reverse_proxy 127.0.0.1:8081
  }
}
```

Then PWA endpoint = `https://axis.example.com` and the **server** engine posts to `/run` on the same host.

Minimal without TLS

- Static `:8081`, Flask `:5002`
- Set endpoint `http://VPS_IP:5002`
- Open firewall; accept CORS from static origin (configure Flask accordingly)

Internals

Concern	Notes
Pyodide offline	Ship full `dist` including `vendor` + `pyodide`
Process supervisor	systemd units for flask + axis_pwa_server
Updates	Rebuild dist artifact; restart static only
Secrets	Flask admin / pro keys separate from Worker

Worked example — systemd sketch

```
# /etc/systemd/system/axis-pwa.service
[Service]
WorkingDirectory=/opt/axis/frontend
Environment=PORT=8081
ExecStart=/usr/bin/python3 axis_pwa_server.py
Restart=on-failure
```

Pair with an existing `backend` unit for Flask.

Invariants & edge cases

1. **CPU/RAM** — Pyodide is browser-side; VPS load is Flask evaluate concurrency.
2. **Do not expose open admin tokens**.
3. **Disk** — dist + wheels are tens of MB.
4. Hybrid: VPS Flask as `EXTERNAL_BACKEND` for a CF Worker still works if VPS has a public URL.

Failure modes

Issue	Fix
Mixed content	HTTPS page calling HTTP Flask blocked
CORS	Prefer same-origin proxy
502 on /run	Flask crashed; check journalctl

See also

- Build and serve
- Cloudflare
- CORS

CORS AND ORIGINS

COPYRIGHT (C) 2024-2026 JANGO_BLOCKCHAINED
THIS FILE IS PART OF PYNESCRIPT.
PYNESCRIPT IS FREE SOFTWARE: YOU CAN REDISTRIBUTE IT AND/OR MODIFY
IT UNDER THE TERMS OF THE GNU AFFERO GENERAL PUBLIC LICENSE AS PUBLISHED BY
THE FREE SOFTWARE FOUNDATION, EITHER VERSION 3 OF THE LICENSE, OR
(AT YOUR OPTION) ANY LATER VERSION.
PYNESCRIPT IS DISTRIBUTED IN THE HOPE THAT IT WILL BE USEFUL,
BUT WITHOUT ANY WARRANTY; WITHOUT EVEN THE IMPLIED WARRANTY OF
MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. SEE THE
GNU AFFERO GENERAL PUBLIC LICENSE FOR MORE DETAILS.
YOU SHOULD HAVE RECEIVED A COPY OF THE GNU AFFERO GENERAL PUBLIC LICENSE
ALONG WITH PYNESCRIPT. IF NOT, SEE
HTTPS://WWW.GNU.ORG/LICENSES/.
SPDX-LICENSE-IDENTIFIER: AGPL-3.0-ONLY

title: "CORS and origins" description: "How AXIS Worker pickOrigin works, local allowlist, ALLOWED_ORIGIN, and Flask/browser constraints."

CORS AND ORIGINS

Abstract

AXIS is a browser product that talks to **cross-origin** backends (Flask, Worker, venues). CORS misconfiguration is the #1 local-deploy footgun after missing wheels.

The Worker implements an explicit origin picker; Flask and venue APIs have their own rules.

Conceptual model

```
flowchart TD
  ORIGIN[Browser Origin header]
  ORIGIN --> W[Worker pickOrigin]
  W -->|localhost:3000/8081| ECHO[Echo request origin]
  W -->|else| ALLOWED[env.ALLOWED_ORIGIN]
  ORIGIN --> F[Flask CORS middleware]
  ORIGIN --> V[Venue APIs]
```

Worker behavior

From `frontend/worker/src/index.ts`:

Always echoed (local-dev regex):

- Any `http://` or `https://` origin on **`localhost`** or **`127.0.0.1`**, optional port (e.g. Vite `:3000`, `axis_pwa_server` `:8081`, Playwright ephemeral ports)

Not allowed (and not needed):

- `http://0.0.0.0:…` — browsers do not use `0.0.0.0` as a page origin in normal use; binding the server with `HOST=0.0.0.0` only means "listen on all interfaces," not a CORS origin.

Otherwise: `env.ALLOWED_ORIGIN` or default `https://pynescript.ai` (exact match).

Localhost/127 allowlisting is **safe for CORS**: only pages actually served from those hosts present that `Origin`. A public site cannot forge `Origin: http://localhost:3000` in a real browser. **CORS headers applied:**

```
Access-Control-Allow-Origin: <picked>
Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS
Access-Control-Allow-Headers: Content-Type, Authorization, X-Admin-Token, If-Match
Access-Control-Max-Age: 86400
Vary: Origin
```

`OPTIONS` → `204` preflight.

Scripts handler duplicates allow headers for consistency.

Production implication

If Pages is `https://pynescript-axis.pages.dev` (example), set:

```
ALLOWED_ORIGIN = "https://pynescript-axis.pages.dev"
```

or your custom domain. **Exact string match** — no wildcard logic in code.

Flask / server engine

The **server** engine posts from the browser to the configured endpoint. Flask Pro API must:

1. Answer `OPTIONS` preflight if cross-origin.
2. Reflect or allow the PWA origin.
3. Accept `Content-Type: application/json`.

Same-origin reverse proxy (VPS demo) eliminates CORS for `/run`.

Venue sources/streams

Public Binance/OKX/etc. REST must send CORS headers usable by browsers. If blocked:

1. Use offline `mock-walk` / `mock-poll`.
2. Add a **same-origin proxy** on Worker/Flask (not a general open proxy — scope carefully).
3. Prefer DO relay only for WS fan-out (WS is not CORS in the XHR sense, but still network-reachable).

Dynamic plugins

- **Module import** of plugin JS: needs CORS on the **script origin** (same-origin `/plugins/` is safe).
- **fetch inside plugin**: subject to third-party CORS independently.

Invariants & edge cases

1. **Credentials** — Worker CORS does not set `Allow-Credentials: true`; use Bearer headers, not cookies.
2. **Multiple prod domains** — code picks a single `ALLOWED_ORIGIN`; multi-domain needs code change or edge rewrite.
3. **Vite vs 8081** — both covered for Worker; Flask config must match whichever you use.
4. **Health checks** from curl omit Origin — still work; browser calls need correct ACAO.

Failure modes

Browser console	Meaning
blocked by CORS policy	Origin not allowlisted
preflight 404	Server lacks OPTIONS
No ACAO on error body	Some error paths forgot headers (Worker try/catch mostly covered)

See also

- Worker bindings
- Local dev
- Dynamic loader

CI AND TESTING

COPYRIGHT (C) 2024-2026 JANGO_BLOCKCHAINED
THIS FILE IS PART OF PYNESCRIPT.
PYNESCRIPT IS FREE SOFTWARE: YOU CAN REDISTRIBUTE IT AND/OR MODIFY
IT UNDER THE TERMS OF THE GNU AFFERO GENERAL PUBLIC LICENSE AS PUBLISHED BY
THE FREE SOFTWARE FOUNDATION, EITHER VERSION 3 OF THE LICENSE, OR
(AT YOUR OPTION) ANY LATER VERSION.
PYNESCRIPT IS DISTRIBUTED IN THE HOPE THAT IT WILL BE USEFUL,
BUT WITHOUT ANY WARRANTY; WITHOUT EVEN THE IMPLIED WARRANTY OF
MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. SEE THE
GNU AFFERO GENERAL PUBLIC LICENSE FOR MORE DETAILS.
YOU SHOULD HAVE RECEIVED A COPY OF THE GNU AFFERO GENERAL PUBLIC LICENSE
ALONG WITH PYNESCRIPT. IF NOT, SEE
HTTPS://WWW.GNU.ORG/LICENSES/.
SPDX-LICENSE-IDENTIFIER: AGPL-3.0-ONLY

title: "CI and testing" description: "GitHub Actions for AXIS: unit coverage gate, Playwright smoke, worker typecheck, nightly full e2e."

CI AND TESTING

Abstract

AXIS quality gates live primarily under `frontend/` (Bun tests) and `frontend/worker/tests/`, orchestrated by GitHub Actions (`.github/workflows/ci.yml`, `axis-nightly.yml`). Deep test conventions are documented in Testing reference and `frontend/TESTING.md`.

Conceptual model

```
flowchart TB
  PR[Pull request]
  PR --> UNIT[Bun unit + worker tests]
  PR --> GATE[Coverage gate 95%]
  PR --> E2E[Playwright @smoke]
  PR --> WTC[Worker tsc]
  NIGHT[Nightly cron]
  NIGHT --> FULL[Playwright all]
  NIGHT --> SEC[Security + coverage]
```

Interface surface — local commands

```
cd frontend
bun run test                 # tests/ + worker/tests/
bun run test:unit
bun run test:coverage:gate   # lcov + scripts/check-coverage.mjs 95
bun run test:security
bun run test:e2e:smoke
bun run test:e2e:critical
bun run test:e2e
bun run test:all             # gate + security
```

Worker:

```
cd frontend/worker
npm run typecheck
```

Repo Makefile:

```
make test-frontend
make worker-typecheck
```

CI workflows

ci.yml (excerpted jobs)

Job (names vary)	What
Frontend tests / coverage	Bun install, coverage, artifact `axis-coverage-lcov`
`axis-e2e`	Playwright `@smoke` on PR
Worker typecheck	`frontend/worker` tsc
Related	Python suite remains for pynescript core

axis-nightly.yml

Job	What
Playwright full	all e2e specs
Security + coverage gate	`test:coverage:gate` + `test:security`

Schedule: `0 4 * * *` UTC + `workflow_dispatch`.

Internals

Path	Role
`frontend/TESTING.md`	Authoritative testing guide
`frontend/scripts/check-coverage.mjs`	Scoped line gate
`frontend/playwright.config.ts`	webServer build+preview `:4173`
`frontend/e2e/`	Specs with `@smoke` / `@critical` tags
`frontend/worker/tests/`	auth, keys, scripts, runtime
`.github/workflows/ci.yml`	PR gates
`.github/workflows/axis-nightly.yml`	Nightly

Coverage policy (summary)

- **Gate:** 95% lines on a **scoped** core (plugins, storage minus idb, store, results, sources, streams, data, selected chart helpers, worker auth/keys/runtime/scripts).
- **Excluded from gate:** heavy chart UI, pyodide boot, legacy JS, Solid `.tsx` surfaces (unit-tested elsewhere or deferred).

E2E design

- Builds production preview; baseURL `http://127.0.0.1:4173`.
- Mocks `/run` and Binance where possible so smoke does not need live Pro API.
- Selectors use `data-testid` (`axis-btn-load`, `axis-manager`, …).

Invariants & edge cases

1. **No real network in unit tests** — mock `fetch` / WebSocket.
2. **Import `./setup` first** when touching store/plugins.
3. **CI Bun version** pinned in workflows (e.g. 1.3).
4. **Playwright browsers** installed with `--with-deps` on CI.

Failure modes

CI red	Likely
Coverage gate	New untested code in scoped paths
Smoke timeout	Build slow; webServer 180s
Worker tsc	Env typing / DO exports
Flaky e2e	Missing testid; race without mock

See also

- Testing reference
- Local dev