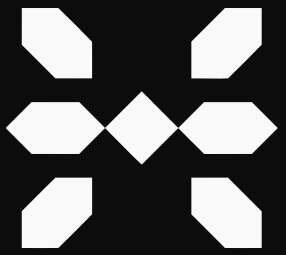
HOOX · AXIS · DOCUMENTATION



EDGE OPERATORS · WORKER AUTHORS

WORKER MANUAL

Cloudflare Worker, Durable Objects, KV, D1, R2, auth

Open charting PWA — own the axes, swap the engine

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

CONTENTS

OVERVIEW

title: "Worker" description: "Cloudflare Worker for AXIS: /api/run proxy, keys, scripts, usage, and Durable Object stream relay."

WORKER

Abstract

The AXIS Cloudflare Worker (`frontend/worker/`) is the **edge data plane** for the PWA: JSON APIs, optional script library (D1), API keys (KV), usage meters, and a WebSocket session relay (Durable Objects). It is **not** a full Pine interpreter in production today.

Honest runtime fact: `POST /api/run` **proxies to** `EXTERNAL_BACKEND` (typically Flask) unless `PYODIDE_IN_WORKER=enabled` and the in-worker path succeeds. In-worker Pyodide is **planned / feature-gated** — see Runtime and `frontend/worker/RUNTIME.md` .

Conceptual model

```
flowchart TB
  PWA[AXIS PWA]
  PWA -->|POST /api/run| W[Worker]
  PWA -->|Bearer /api/scripts| W
  PWA -->|WS /api/stream| W
  W -->|proxy| FLASK[EXTERNAL_BACKEND Flask]
  W -.->|gated| PY[Pyodide in Worker]
  W --> KV[API_KEYS / USAGE KV]
  W --> D1[(D1 scripts)]
  W --> DO[SessionDO]
  DO --> BIN[Binance WS]
```

Infrastructure names (frozen)

Surface	Value
Wrangler / Pages project	**`pynescript-axis`** — **do not rename** lightly
npm package	`axis-worker`
Health JSON `service`	`pynescript-axis-worker`
Product brand	**AXIS**

Custom domains / aliases may say `axis.*`; leave the CF project id stable so KV/D1 IDs and CI stay valid.

Interface surface

Path	Method	Role
`/`, `/health`	GET	Health + feature flags (`scripts`, `d1`, `keys`)
`/api/run`	POST	Run script (proxy / gated Pyodide)
`/api/keys`	GET/POST	Validate / create keys (`X-Admin-Token` for create)
`/api/usage`	GET	Usage stub / KV-backed counters
`/api/scripts`…	CRUD	Script library (Bearer)
`/api/stream`	WS	Session DO relay

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

Local entrypoints

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

Makefile: `make worker-dev`, `make worker-deploy`.

Implementation status

Feature	Status
`/api/run` → `EXTERNAL_BACKEND`	**Works today**
Admin `/api/keys`	Works (KV or shape-check)
Usage KV increment on run	Works when `USAGE` bound
`SessionDO` + `/api/stream`	Implemented; binding often commented until provisioned
`/api/scripts` + D1 schema	Implemented; memory fallback without D1
Pyodide scaffold	`pyodide_runtime.ts` + flag; **wheel pipeline incomplete**
Response cache for `/api/run`	Deferred

Page map

Page	Topic
Runtime	`/api/run`, proxy, Pyodide plan
Durable Objects	Stream fan-out
Data plane	Scripts, D1, drafts
Auth	Bearer keys, admin token
Bindings	wrangler.toml, vars, provision

See also

- Cloudflare deploy
- Engines
- Worker README: `frontend/worker/README.md`

RUNTIME

title: "Worker runtime" description: "POST /api/run execution modes: EXTERNAL_BACKEND proxy today, feature-gated in-worker Pyodide planned."

WORKER RUNTIME

Abstract

`frontend/worker/src/runtime.ts` implements `handleRun` for **`POST /api/run`**. Execution preference is documented in-file:

1. **`PYODIDE_IN_WORKER=enabled`** → `tryRunInWorker` (`pyodide_runtime.ts`)
2. Else **`EXTERNAL_BACKEND`** set → HTTP proxy to `${EXTERNAL_BACKEND}/run`
3. Else **503** `NO_BACKEND` with a pointer at env vars and `RUNTIME.md`

Production reality: ships as a **proxy** to Flask (or another Python host). In-worker Pyodide is a **scaffold**, not a complete wheel-upload pipeline.

Conceptual model

```
flowchart TD
  REQ[POST /api/run]
  REQ --> VAL{validate body}
  VAL -->|fail| 400
  VAL --> METER[USAGE KV optional]
  METER --> FLAG{PYODIDE_IN_WORKER}
  FLAG -->|enabled| PY[tryRunInWorker]
  PY -->|ok| 200
  PY -->|fail boot| PROXY
  FLAG -->|disabled| PROXY[proxyToExternal]
  PROXY -->|no EXTERNAL_BACKEND| 503
  PROXY -->|fetch| UP[EXTERNAL_BACKEND/run]
```

Interface surface

Request body

```
{
  script: string;           // required non-empty
  data: Bar[];              // required non-empty OHLCV array
  mode?: 'interpret' | 'compile';
}
```

Invalid bodies → `400` `{ status: 'error', code: 'BAD_REQUEST', message }`.

Success / upstream

Proxy returns upstream status and body verbatim (JSON content-type, CORS origin header). Flask Pro API is the usual upstream.

Error codes (Worker-originated)

Code	HTTP	Meaning
`BAD_REQUEST`	400	Validation
`NO_BACKEND`	503	No external backend and Pyodide path off/failed path to proxy
(Pyodide)	200 body error	`tryRunInWorker` may return `{ status: 'error', error }` JSON

Usage metering

If `Authorization: Bearer …` and `USAGE` KV bound:

```
usage:{key} → count++, TTL 30 days
```

Internals

Path	Role
`src/runtime.ts`	validate, meter, dispatch
`src/pyodide_runtime.ts`	gated boot + `run_script` stub path
`RUNTIME.md`	architecture plan for wheels / R2 / cold start
`wrangler.toml` `[vars]`	`EXTERNAL_BACKEND`, `PYODIDE_IN_WORKER`

pyodide_runtime scaffold

When enabled:

- Lazy-loads Pyodide from jsDelivr `v0.26.2` (dev fallback).
- Production intent: R2 wheels + module-level cache (`RUNTIME.md`).
- Calls `run_script(script, bars)` via `runPythonAsync` — requires runtime bootstrap that is **not fully wired** like the browser engine's `pynescript_runtime.py` + micropip install.

Do not enable in production without completing the wheel pipeline and load testing CPU/memory limits.

Constraints (from RUNTIME.md)

Resource	Budget
CPU (paid)	30s limit; typical 500 bars ≪ 1s target
Memory	128 MB; Pyodide ~30 MB + wheel ~5 MB
Cold start	~3s first Pyodide boot (plan)

Roll-out flag

`RUNTIME.md` also mentions `RUNTIME_MODE=in-worker` as product language; **code checks `PYODIDE_IN_WORKER === 'enabled'`**. Prefer the code flag when configuring.

Invariants & edge cases

1. **Body stream** — request body is consumed once; proxy re-serializes the already-parsed object.
2. **CORS** — run responses set `Access-Control-Allow-Origin` from `pickOrigin` (see CORS).
3. **PWA path mismatch** — browser `server` engine posts to `{endpoint}/run`; Worker route is `/api/run`. Align via gateway rewrite or endpoint base path convention.
4. **No response caching yet** — README lists cache for identical `(script, data_hash)` as deferred.

Worked examples

Local proxy to Flask

```
# wrangler.toml [vars]
EXTERNAL_BACKEND = "http://127.0.0.1:5002"
PYODIDE_IN_WORKER = "disabled"
```

```
curl -sS -X POST http://127.0.0.1:8787/api/run \
  -H 'content-type: application/json' \
  -d '{"script":"//@version=5\nindicator(\"t\")\nplot(close)","data":[{"time":1,"open":1,"high":1,'
```

Force 503 for missing backend

Unset `EXTERNAL_BACKEND`, keep Pyodide disabled → expect `NO_BACKEND` message referencing `RUNTIME.md`.

Failure modes

Symptom	Fix
503 `NO_BACKEND`	Set `EXTERNAL_BACKEND` or finish Pyodide path
Proxy 502/connection refused	Flask not listening; wrong URL from Worker (use tunnel/public URL in CF, not localhost)
Pyodide enabled but errors	Wheel missing; CDN blocked; incomplete `run_script` bootstrap

See also

- Bindings
- Engines
- Auth (Bearer on run for metering)

DURABLE OBJECTS

title: "Durable Objects" description: "SessionDO WebSocket relay: one Binance upstream per session, fan-out to browser clients."

DURABLE OBJECTS

Abstract

`SessionDO` (`frontend/worker/src/durable-objects/session.ts`) is a **per-session WebSocket relay**. Browsers connect to the Worker at `/api/stream`; the Worker pins a Durable Object by session name and the DO opens **one** upstream Binance kline socket, broadcasting payloads to connected clients.

Goal: survive browser per-origin WebSocket connection limits and share upstream subscriptions.

Conceptual model

```
flowchart LR
  B1[Browser tab A]
  B2[Browser tab B]
  W[Worker /api/stream]
  DO[SessionDO]
  UP[Binance kline WS]
  B1 --> W
  B2 --> W
  W -->|idFromName session| DO
  DO --> UP
  DO -->|broadcast| B1
  DO -->|broadcast| B2
```

Interface surface

Client → Worker

```
wss://<worker>/api/stream?session=<name>&symbol=BTCUSDT&interval=1m
```

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

- Requires `env.SESSIONS` binding; else `503` `NO_DO`.
- `idFromName(session || 'default')`.
- Rewrites request to DO path `/ws?…` and returns upgrade response.

DO HTTP surface

Path	Role
`/ws`	Only path; requires `Upgrade: websocket` (else 426)

Query: `symbol` (default `BTCUSDT`), `interval` (default `1m`).

Client → DO messages (JSON)

Message	Effect
`{ "action": "subscribe", "symbol", "interval" }`	Rebind upstream if changed
`{ "action": "ping" }`	`{ action: "pong", t }`

DO → client

- Raw Binance kline JSON (same as direct venue WS).
- Status: `{ type: 'status', state: 'open'|'closed', url? }`
- Errors: `{ type: 'error', message }`

Example plugin

`frontend/public/plugins/example-cf-do-stream.js` maps kline fields to `Bar` and builds the Worker URL from config `endpoint`.

Internals

Topic	Detail
Upstream URL	`wss://stream.binance.com:9443/ws/{symbol}@kline_{interval}`
Client lifecycle	`acceptWebSocket`; on last client close → `closeUpstream`
Hibernation	Comment intent: hibernate when empty; implementation uses explicit close
Fan-out	`broadcast` to all `clients`

wrangler binding (often commented until ready)

```
# [[durable_objects.bindings]]
# name = "SESSIONS"
# class_name = "SessionDO"

# [[migrations]]
# tag = "v1"
# new_sqlite_classes = ["SessionDO"]
```

`SessionDO` is re-exported from `src/index.ts` for the Workers runtime class registry.

Invariants & edge cases

1. **Session key** — example plugin uses `symbol@interval` as session name so like-minded tabs share one DO.
2. **No auth on stream today** — treat as public relay of public market data; do not put secrets in query strings.
3. **Upstream hard-coded Binance** — not a generic multiprovider DO yet.
4. **Binding optional in local** — without `SESSIONS`, stream API fails closed with `NO_DO`.

Failure modes

Symptom	Cause
503 `NO_DO`	Binding commented / not deployed
426 expected websocket	Non-upgrade GET
No bars	Upstream block or wrong interval string
Stale symbol	Client did not send `subscribe` after change

See also

- Streams
- Bindings
- Plugin examples

DATA PLANE

title: "Worker data plane" description: "Script library API: /api/scripts, D1 schema, drafts, revisions, and in-memory fallback."

WORKER DATA PLANE

Abstract

The Worker **data plane** for user scripts is `frontend/worker/src/scripts.ts` — REST under `/api/scripts` with Bearer auth (auth). Persistence prefers **D1** (`env.DB`); without D1, per-isolate **in-memory** maps support local `wrangler dev`.

The PWA **cloud** storage plugin is the primary client.

Conceptual model

```
flowchart TB
  CLOUD[cloud storage plugin]
  CLOUD -->|Bearer| SCR[handleScripts]
  SCR --> AUTH[requireApiKey → userId]
  AUTH --> D1[(D1 scripts)]
  AUTH --> MEM[(memScripts Map)]
```

Interface surface

All routes require valid API key unless noted.

Path	Methods	Behavior
`/api/scripts`	GET	List meta for `userId`
`/api/scripts`	POST	Create script
`/api/scripts/:id`	GET	Read full document
`/api/scripts/:id`	PUT	Upsert; honors `If-Match` revision
`/api/scripts/:id`	DELETE	Remove
`/api/scripts/_draft`	GET/PUT	Per-user draft buffer

Document fields

Field	Notes
`id`	Client or server assigned
`name`, `description`, `path`	Meta
`content`	Pine source
`revision`	Opaque string (`rev_…`) for concurrency
`created_at` / `updated_at`	ms epochs (API may camelCase in JSON)

Concurrency

PUT with `If-Match: <revision>` rejects mismatched revisions (409 path covered in security tests). Fresh writes mint `newRevision()`.

Internals

D1 schema (`schemas/scripts.sql`)

```
CREATE TABLE scripts (
  user_id TEXT NOT NULL,
  id TEXT NOT NULL,
  name TEXT NOT NULL,
  description TEXT,
  path TEXT,
  content TEXT NOT NULL,
  revision TEXT NOT NULL,
  created_at INTEGER NOT NULL,
  updated_at INTEGER NOT NULL,
  PRIMARY KEY (user_id, id)
);

CREATE TABLE script_drafts (
  user_id TEXT PRIMARY KEY,
  content TEXT NOT NULL,
  name TEXT,
  updated_at INTEGER NOT NULL
);
```

Apply:

```
wrangler d1 execute pynescript --remote --file=schemas/scripts.sql
wrangler d1 execute pynescript --local  --file=schemas/scripts.sql
```

Binding name in `wrangler.toml` **must** be `DB` (code uses `env.DB`). Database name can be `pynescript`.

Partitioning

`userId = SHA-256(api_key).hex.slice(0, 32)` — raw keys are not stored as D1 partition ids.

Health feature flags

```
{
  "status": "healthy",
  "service": "pynescript-axis-worker",
  "features": { "scripts": true, "d1": true/false, "keys": true/false }
}
```

Optional R2

`BUNDLES` R2 is reserved for indicator / pynescript wheels (`RUNTIME.md`). Not required for scripts CRUD.

Invariants & edge cases

1. **Memory store is not multi-isolate durable** — production needs D1.
2. **Drafts ≠ commits** — drafts are crash buffers; library list is separate.
3. **CORS** — scripts handler sets allow headers including `If-Match`.
4. **Cloud plugin dual-write** — browser still saves drafts to local storage.

Failure modes

Code / symptom	Meaning
401	Missing/invalid Bearer
404	Unknown script id for user
409	Revision conflict
Empty list after deploy	Schema not applied / wrong DB id

See also

- Storage plugins
- Auth
- Bindings

AUTH

title: "Worker auth" description: "API keys (pn_...), admin token, Bearer extraction, ALLOW_OPEN_KEYS, and KV validation."

WORKER AUTH

Abstract

AXIS Worker auth is **API-key based**, not session cookies. Keys are created by admins (`X-Admin-Token`), validated via KV when bound, and used as `Authorization: Bearer` on script library and optionally on `/api/run` (usage metering).

Core helpers: `frontend/worker/src/auth.ts`. Key CRUD: `frontend/worker/src/keys.ts`.

Conceptual model

```
flowchart TD
  REQ[Request]
  REQ --> BEARER[extractBearer]
  BEARER --> KV{API_KEYS bound?}
  KV -->|yes| LOOKUP[get key:…]
  LOOKUP -->|miss| 401
  LOOKUP -->|hit| CTX[AuthContext]
  KV -->|no| OPEN{ALLOW_OPEN_KEYS?}
  OPEN -->|yes| CTX
  OPEN -->|no| SHAPE{pn_ + 48 hex?}
  SHAPE -->|yes| CTX
  SHAPE -->|no| 401
```

Interface surface

AuthContext

```
{ key: string; userId: string; tier: string }
```

`userId` is a truncated SHA-256 of the key (stable partition id).

extractBearer

1. `Authorization: Bearer <token>`
2. Else query `?key=`

requireApiKey

Returns either `{ ok: true, ctx }` or `{ ok: false, status, code, message }`.

Code	HTTP	When
`NO_KEY`	401	Empty
`INVALID_KEY`	401	Unknown in KV / malformed

Admin keys API (`/api/keys`)

Action	Auth	Behavior
create (`POST` or `?action=create`)	`X-Admin-Token` === `env.ADMIN_TOKEN`	Mint `pn_` + 48 hex, store `key:{key}` in KV (1y TTL) with tier
validate (`GET` or `?action=validate`)	Bearer or `?key=`	Tier + created_at if known

Tiers: `free` | `hobby` | `pro` | `team` | `enterprise`.

Without `ADMIN_TOKEN` set, **create always fails** (`isAdmin` false).

Without KV on validate: accepts well-formed `pn_[a-f0-9]{48}` as hobby (dev).

ALLOW_OPEN_KEYS

`wrangler.toml` defaults local demos to `"1"`:

> accept any non-empty Bearer key for `/api/scripts`

Never leave open in production. Prefer bound `API_KEYS` + real admin-minted keys.

Internals

Path	Role
`src/auth.ts`	Bearer, hash, requireApiKey
`src/keys.ts`	Admin create / validate
`src/runtime.ts`	Optional usage increment by Bearer
`src/scripts.ts`	requireApiKey gate

Key format

```
'pn_' + 24 random bytes as hex  // 48 hex chars
```

Invariants & edge cases

1. **Admin token is a shared secret** — rotate via wrangler secrets in production, not committed vars.
2. **KV record shape** — JSON `{ key, tier, createdAt }` under `key:${apiKey}`.
3. **No OAuth / cookies** — PWA stores the key in plugin config (`storage:cloud`).
4. **Stream DO unauthenticated** — public market data only.

Worked examples

Create a key (admin)

```
curl -sS -X POST 'http://127.0.0.1:8787/api/keys?action=create' \
  -H "X-Admin-Token: $ADMIN_TOKEN" \
  -H 'content-type: application/json' \
  -d '{"tier":"hobby"}'
```

Validate

```
curl -sS 'http://127.0.0.1:8787/api/keys?action=validate' \
  -H "Authorization: Bearer pn_…"
```

Failure modes

Symptom	Cause
403 create	Wrong/missing admin token
401 scripts	Open keys off + no KV + bad shape
Cross-user leakage tests fail	Partition hash regression

See also

- Data plane
- Bindings
- Security tests

BINDINGS

title: "Worker bindings" description: "wrangler.toml vars, KV, D1, R2, Durable Objects — provision once, paste IDs, frozen project name."

WORKER BINDINGS

Abstract

Bindings and vars for the AXIS Worker are declared in `frontend/worker/wrangler.toml`. Many production bindings ship **commented** so clone-and-dev does not require Cloudflare resources; `wrangler dev` synthesizes local substitutes where possible.

Project name is frozen: `name = "pynescript-axis"`.

Conceptual model

```
flowchart TB
  W[Worker isolate]
  W --> V[vars]
  W --> KV1[API_KEYS KV]
  W --> KV2[USAGE KV]
  W --> D1[(DB D1)]
  W --> R2[BUNDLES R2]
  W --> DO[SESSIONS DO]
```

Interface surface — Env

From `src/index.ts`:

```
interface Env {
  API_KEYS?: KVNamespace;
  USAGE?: KVNamespace;
  DB?: D1Database;
  BUNDLES?: R2Bucket;
  SESSIONS?: DurableObjectNamespace;

  EXTERNAL_BACKEND?: string;
  ALLOWED_ORIGIN?: string;
  ADMIN_TOKEN?: string;
  PYODIDE_IN_WORKER?: string;
  ALLOW_OPEN_KEYS?: string;
}
```

Vars (`[vars]`)

Var	Default (repo)	Role
`EXTERNAL_BACKEND`	`""`	Flask (or other) base URL for `/api/run` proxy
`ALLOWED_ORIGIN`	`https://pynescript.ai`	CORS allow when Origin not local
`ADMIN_TOKEN`	`""`	Shared secret for key minting
`PYODIDE_IN_WORKER`	`"disabled"`	Gate in-worker Python
`ALLOW_OPEN_KEYS`	`"1"`	Local demos; set `"0"` in prod

Prefer **secrets** for `ADMIN_TOKEN` and production backends:

```
wrangler secret put ADMIN_TOKEN
wrangler secret put EXTERNAL_BACKEND
```

Provision recipes

KV

```
wrangler kv namespace create API_KEYS
wrangler kv namespace create USAGE
# paste ids into wrangler.toml [[kv_namespaces]]
```

D1

```
wrangler d1 create pynescript
# [[d1_databases]] binding = "DB"  (name MUST be DB)
wrangler d1 execute pynescript --remote --file=schemas/scripts.sql
```

Repo may already contain a `database_id` for the project's D1 — treat IDs as environment-specific when forking.

R2 (optional)

```
wrangler r2 bucket create indicator-bundles
# binding BUNDLES
```

Durable Objects

Uncomment bindings + migrations in `wrangler.toml`, then:

```
wrangler deploy
```

Class: `SessionDO` exported from `src/index.ts`.

Other wrangler settings

Setting	Value
`main`	`src/index.ts`
`compatibility_date`	`2024-11-01`
`compatibility_flags`	`nodejs_compat`
`[observability]`	`enabled = true`

Static PWA assets are **not** served by this Worker in the documented split: **Pages** serves `dist/`, Worker serves API/WS. Project name for Pages deploy remains `pynescript-axis`.

Deploy commands

```
cd frontend/worker
wrangler deploy

# Pages (from frontend/)
bun run build
wrangler pages deploy dist --project-name=pynescript-axis
# or npm run deploy:pages from worker package
```

Makefile: `make worker-deploy`, `make pages-deploy` (note: some Makefile targets still point at legacy tree — prefer `dist/` from Vite build; see build and serve).

Invariants & edge cases

1. **Do not rename** CF project without migrating all binding IDs, custom domains, and CI.
2. **Local wrangler** auto-provisions mock KV/D1; production needs real IDs.
3. **`EXTERNAL_BACKEND=localhost` on CF** cannot reach your laptop — use a public tunnel or co-located VPS.
4. **Health** reports which optional bindings are present.

Failure modes

Deploy / runtime issue	Fix
Deploy fails placeholder KV	Uncomment only after create, or leave commented
Scripts empty	Apply SQL schema
Stream 503	Enable DO binding + migration
CORS fails prod origin	Set `ALLOWED_ORIGIN` to exact Pages origin

See also

- Cloudflare DevOps
- Runtime
- CORS