SPEC-10: REST API Hosting on Cloudflare Workers (Free Tier)

1. Background

Hearthlink CORE and Custom GPTs need a secure REST API endpoint to invoke commands remotely. Leveraging Cloudflare Workers' free tier ensures global low-latency execution, free TLS, and built-in KV for secrets—all at zero infrastructure cost.

Goals:

- Expose `/v1/tasks`, `/v1/status`, and `/v1/health` endpoints.
- Secure with JWT auth stored in Workers KV.
- Minimal cold-start latency (<100 ms).
- High availability with 100k requests/day free quota.

2. Requirements (MoSCoW)

Must-have

- Cloudflare Worker script handling HTTP routing.
- KV namespace for JWT secret (`JWT_SECRET`) and rate counters.
- JWT validation middleware for `Authorization: Bearer <token>`.
- `/v1/tasks` POST: invoke Hearthlink commands with body schema `{ command: string, flags?: string[] }`.
- `/v1/status` GET: return Worker health and KV connectivity.
- `/v1/health` GET: simple 200 OK for uptime check.

Should-have

- Rate-limiting logic in Worker (KV-based counter per token).
- Input JSON schema validation with error responses.
- CORS policy restricting origins to `https://platform.openai.com`.
- **Web dashboard showing usage metrics** (via Workers Analytics API), integrated into Sentry UI dashboards.
- **Websocket support for real-time logs**, feeding live events into Sentry.

Could-have

- Web dashboard showing usage metrics (via Workers Analytics API).
- Websocket support for real-time logs.

Won't-have (v1)

- Complex orchestration (no durable objects).

• File uploads/downloads.

3. Architecture & Diagrams

3.1 Component Diagram

```
@startuml cfComponents
title Cloudflare Workers REST API Components
actor CustomGPT
node "Cloudflare Worker" {
  component Router
  component JWTMiddleware
  component RateLimiter
  component HearthlinkClient
}
database KV as "Workers KV"
CustomGPT --> Router: HTTP Request
Router --> JWTMiddleware: Validate Token
JWTMiddleware --> KV: Fetch JWT_SECRET
Router --> RateLimiter: Check Limit
RateLimiter --> KV: Increment Counter
Router --> HearthlinkClient: invoke command
@enduml
```

3.2 Sequence Diagram

```
@startuml cfSequence
title Request Flow for /v1/tasks
actor CustomGPT
participant Worker
participant JWTMW as JWTMiddleware
participant RateLM as RateLimiter
participant KV
participant HL as HearthlinkAPI
CustomGPT -> Worker: POST /v1/tasks + Body + Auth
Worker -> JWTMW: validate(token)
JWTMW -> KV: get("JWT_SECRET")
JWTMW --> Worker: valid?
Worker -> RateLM: check(token)
RateLM -> KV: increment(token)
RateLM --> Worker: allowed?
Worker -> HV: HL.invoke(command, flags)
HL --> Worker: result
Worker --> CustomGPT: 200 JSON
@enduml
```

4. Implementation

4.1 Wrangler Configuration

```
name = "hearthlink-api"
type = "javascript"

[env.production]
  account_id = "<CF_ACCOUNT_ID>"
  workers_dev = true

[[kv_namespaces]]
binding = "JWT_SECRETS"
id = "<KV_NAMESPACE_ID>"
```

4.2 Worker Script (`index.js`)

```
import jwt from 'jsonwebtoken'

addEventListener('fetch', event => {
  event.respondWith(handleRequest(event.request))
})

async function handleRequest(req) {
  const url = new URL(req.url)
  if (url.pathname === '/v1/health') return new Response('OK', {status:200})
  if (url.pathname === '/v1/status') return statusHandler()
  if (url.pathname === '/v1/tasks' && req.method === 'POST') return
tasksHandler(req)
  return new Response('Not Found', {status:404})
}

// JWT Middleware
async function verifyJWT(request) {
  const auth = request.headers.get('Authorization') || ''
  const token = auth.split(' ')[1]
  const secret = await JWT_SECRETS.get('JWT_SECRET')
  return jwt.verify(token, secret)
}

// Rate Limiter
async function checkRate(token) {
  const key = `rate_${token}`
  const count = await JWT_SECRETS.get(key) || 0
  if (count > 1000) throw new Error('Rate limit exceeded')
  await JWT_SECRETS.put(key, parseInt(count)+1, {expirationTtl:86400})
}

// Handlers...
```

5. Security & Configuration

- **JWT_SECRET** stored in KV via `wrangler kv:key put JWT_SECRETS JWT_SECRET <secret>`
- **CORS**: add `Access-Control-Allow-Origin: https://platform.openai.com` header.
- **Secrets Rotation**: rotate JWT_SECRET monthly; invalidate old tokens.

6. Testing & Deployment

1. **Unit Tests**: mock `JWT_SECRETS` with Wrangler's KV mock plugin.
2. **Integration**: `wrangler dev` locally; call endpoints via curl.
3. **Deploy**: `wrangler publish --env production`.
4. **Monitor**: use Cloudflare dashboard for request counts and errors.

7. Documentation

- Add `docs/cloudflare-workers.md` with full setup and API spec.
- Include example curl snippets, error codes, and response schemas.
- Draft a quickstart for Custom GPT developers to configure their `openapi.yaml` for the endpoints.

This plan ensures a fully functional, secure, zero-cost REST API for Hearthlink, ready for Custom GPT integrations.