Hono Cloudflare Workers Project Structure

Full Cloudflare stack with D1 database, KV cache, R2 storage, and Queues for background jobs.

#hono #typescript #cloudflare #workers #d1 #kv #r2

PNG PDF Copy Prompt

Project Directory

```
myproject/
  src/
    index.ts  Worker entry
    app.ts  Hono app
    routes/
      index.ts
      users.ts
      files.ts  R2 uploads
    services/
      user.service.ts
      storage.service.ts  R2 operations
      cache.service.ts  KV operations
    db/
      index.ts
      schema.ts
    queues/  Queue consumers
      email.queue.ts
      webhook.queue.ts
    middleware/
      bindings.ts  Type-safe env
      auth.ts
    types/
      env.d.ts  Bindings types
  migrations/
    .gitkeep
  wrangler.toml  All bindings co…
  drizzle.config.ts
  package.json
  tsconfig.json
  .dev.vars  Local secrets
  .gitignore
```

Why This Structure?

Full Cloudflare platform integration: D1 for SQL, KV for caching, R2 for files, Queues for async work. All bindings typed in `env.d.ts`. Hono provides the routing layer with Cloudflare-native middleware.

Key Directories

`src/queues/` - Queue consumer handlers for background jobs

`src/services/storage.service.ts` - R2 bucket operations

`src/services/cache.service.ts` - KV namespace operations

`src/types/env.d.ts` - Typed bindings for D1, KV, R2, Queues

Getting Started

1. `npm create hono@latest -- --template cloudflare-workers`
2. `wrangler d1 create mydb`
3. `wrangler kv:namespace create cache`
4. `wrangler r2 bucket create files`
5. Update `wrangler.toml` with binding names

Cloudflare Bindings

D1 - `env.DB` - SQLite database

KV - `env.CACHE` - Key-value store

R2 - `env.FILES` - Object storage

Queues - `env.EMAIL_QUEUE` - Async jobs

Secrets - `env.API_KEY` - Encrypted secrets

When To Use This

- Building on Cloudflare's full platform
- Need database, cache, and file storage
- Background job processing required
- Global edge deployment priority
- Cloudflare ecosystem investment

Trade-offs

Vendor lock-in - Deeply tied to Cloudflare services

D1 limitations - SQLite constraints vs full Postgres

Cold starts - Large apps may have slower cold starts