Pet Rock — A Scalable but Still Useless Web Service

by Mario Burger · AI assistance used (ChatGPT, Claude)

A horizontally-scalable demo web app built around the concept of caring for a virtual pet rock. The actual game is deliberately pointless — the real purpose is to showcase WebSockets, real-time pub/sub fan-out across stateless API replicas, JWT authentication with refresh-token rotation, and load balancing with Traefik.

Quick Start

Requires Docker with Docker Compose.

1. Create a `.env` file in the project root:

```
JWT_SECRET=change-me-to-a-long-random-string
```

2. Start the full stack:

```
docker compose up --build --scale api=3
```

Change `api=3` to any number to demonstrate horizontal scaling.

3. Open the app:

URL	What it is
http://localhost/register	Create your rock
http://localhost	App (redirects to login if not authenticated)
http://localhost:8080	Traefik dashboard

Successful startup is confirmed when each `api-n` container logs `INFO: Application startup complete`.

Architecture

```
Browser → Traefik (lb) → API instances (×N, stateless) → PostgreSQL + Redis
                ↘ Frontend (Nginx, static)
                                    ↑
                              Worker (background loop)
```

All services run in Docker Compose. Traefik routes `/api/*` to the API pool and everything else to the frontend. The API is fully stateless — any instance can serve any request. Redis pub/sub is used to fan out WebSocket events across instances.

Services

Service	Technology	Role
lb	Traefik v3	Reverse proxy, load balancer
api	FastAPI · Uvicorn · Python	REST API + WebSocket endpoints, horizontally scalable
worker	Python (same image as API)	Background loop: dust simulation, leaderboard refresh, telemetry heartbeat
db	PostgreSQL 18	Primary persistent store
redis	Redis 8	Cache, pub/sub message bus for WebSocket fan-out
frontend	React 19 · TypeScript · Vite · Nginx	SPA served as static files
migrate	Alembic	Runs DB migrations on startup before API/worker start

Real-time pattern

WebSocket connections are spread across multiple `api` instances by Traefik. When one instance receives a rock action, it publishes an event to a Redis channel. All instances subscribe to that channel and push the update to their connected clients. This means a user connected to `api-1` instantly sees actions performed by a user on `api-2`.

Implemented in `backend/app/api/`: `shrine_ws.py`, `leaderboard_ws.py`, `rockscape_ws.py`, `telemetry_ws.py`.

WebSocket clients fall back to REST polling automatically when the connection drops and re-upgrade once it's restored. The connection state is surfaced in the header indicator (green = WebSocket live, yellow = polling fallback, blue = REST-only page, red = error).

Authentication

JWT-based with short-lived access tokens (default: 1 minute) and rotating refresh tokens (default: 7 days).

- Access tokens are validated on every protected API request via `Authorization: Bearer <token>`
- On expiry, the frontend proactively refreshes the access token before sending the next request — no 401 visible in the console under normal conditions
- Refresh tokens are stored in the `RefreshToken` table (not stateless) and rotated on every use
- Concurrent refresh calls are serialized client-side to prevent race conditions during token rotation

Auth logic: `backend/app/auth.py`, routes: `backend/app/api/auth.py`.

Data model

Model	Description
User	Account with hashed password (Argon2)
Rock	One per user — dust_level (0–100), mood, last_simulated_at
RockAction	Records each polish/admire event
ShrineFeed	Public activity feed derived from actions and offers
LeaderboardDaily	Aggregated daily scores, rebuilt every 60 s by the worker
RefreshToken	Stored refresh tokens for rotation and revocation

ORM: SQLAlchemy 2 (async). Migrations: Alembic (backend/alembic/).

Worker

backend/app/worker.py runs an infinite loop that:

1. Accumulates 1 dust/minute per rock (capped at 100)
2. Rebuilds LeaderboardDaily every 60 s
3. Publishes heartbeat telemetry to Redis so the Telemetry dashboard can show live node status

---

Pages

Route	Auth	Connection	Description
/register	Public	REST	Create account + rock
/login	Public	REST	Sign in
/	Required	REST	Your rock — polish, admire, rename, see dust level
/rockscape	Required	WebSocket	Parallax scene showing all recently active rocks
/shrine	Required	WebSocket	Live public activity feed + post offerings
/leaderboard	Required	WebSocket	Daily score rankings, updated by worker every 60 s
/telemetry	Required	WebSocket	Live service health dashboard + architecture diagram with per-replica status

---

Development

Environment variables

Variable	Required	Default	Description
JWT_SECRET	Yes	—	Secret for signing JWTs — set this

			in .env
DATABASE_URL	No	postgresql+psycopg://petrock:petrock@db:5432/petrock	PostgreSQL connection string
REDIS_URL	No	redis://redis:6379/0	Redis connection string
ACCESS_TOKEN_MINUTES	No	1	Access token lifetime
REFRESH_TOKEN_DAYS	No	7	Refresh token lifetime

Database migrations

```
# Inside the api container, or with backend env vars set locally:
alembic upgrade head

# Create a new migration after changing models:
alembic revision --autogenerate -m "description"
```

Frontend (outside Docker, with hot reload)

```
cd frontend
npm install
npm run dev      # Vite dev server on :5173
npm run build    # Production build
```

Backend (outside Docker)

```
cd backend
pip install -r requirements.txt
uvicorn app.main:app --reload    # API server on :8000
python -m app.worker             # Background worker
```

Scaling

The API is stateless by design. Scale it up at any time:

```
docker compose up --scale api=5
```

Watch the `X-Served-By` header change on each request. All instances share the same PostgreSQL database and Redis bus, so real-time events and leaderboard data are consistent across replicas.

Load Tests

Run with hey against 2 API replicas on WSL2 (Intel Core i7-10750H @ 2.60 GHz · 32 GB RAM · 12 logical CPUs):

```
hey -z 30s -c 100 -H "Authorization: Bearer <access_token>" <endpoint>
```

These tests hit the frontend static files served by Nginx behind Traefik:

```
/shrine       13,055 req/s  (avg 7.7 ms, slowest 57.2 ms)
/register     12,729 req/s  (avg 7.9 ms, slowest 62.2 ms)
/rockscape    12,528 req/s  (avg 8.0 ms, slowest 66.0 ms)
/leaderboard  12,789 req/s  (avg 7.8 ms, slowest 71.2 ms)
/telemetry    12,572 req/s  (avg 8.0 ms, slowest 63.3 ms)
```