FinnSteimle / **PixelCanvas**

Code Issues Pull requests Agents Actions Projects Wiki Security Insights Settings

Watch 0

Fault-tolerant, real-time C++ canvas with Redis, PostgreSQL, and Nginx. Hardened with thread-safe pooling and RAII poison control, maintaining a 100% success rate under high-concurrency load.

0 stars **0** forks **0** watching Branches Activity
Tags

Public repository

... **2** Branches **0** Tags Go to file t Go to file Add file Code

FinnSteimle Merge pull request #1 from FinnSteimle/jwt-fix2 c65c6bb · now

.devcontainer	checkpoint after upgrading to c++20 and ...	4 days ago
init-db	add comments to explain code	5 days ago
nginx	fix jwt and readme	2 minutes ago
src	fix jwt and readme	2 minutes ago
static	fix jwt and readme	2 minutes ago
tests	fix jwt and readme	2 minutes ago
.dockerignore	JWT implementation	2 weeks ago
.env.example	checkpoint after upgrading to c++20 and ...	4 days ago
.gitignore	inital dev-container setup with dependen...	2 weeks ago
CMakeLists.txt	checkpoint after upgrading to c++20 and ...	4 days ago
Dockerfile.backend	fix jwt and readme	2 minutes ago
README.md	fix jwt and readme	2 minutes ago
docker-compose.yml	add backend visualization and change loa...	2 days ago

README

PixelCanvas: Distributed Collaborative Canvas

PixelCanvas is a fault-tolerant, real-time, distributed clone of r/place. It allows multiple users to paint simultaneously on a shared 50x50 grid. The system is designed for high concurrency and fault tolerance.

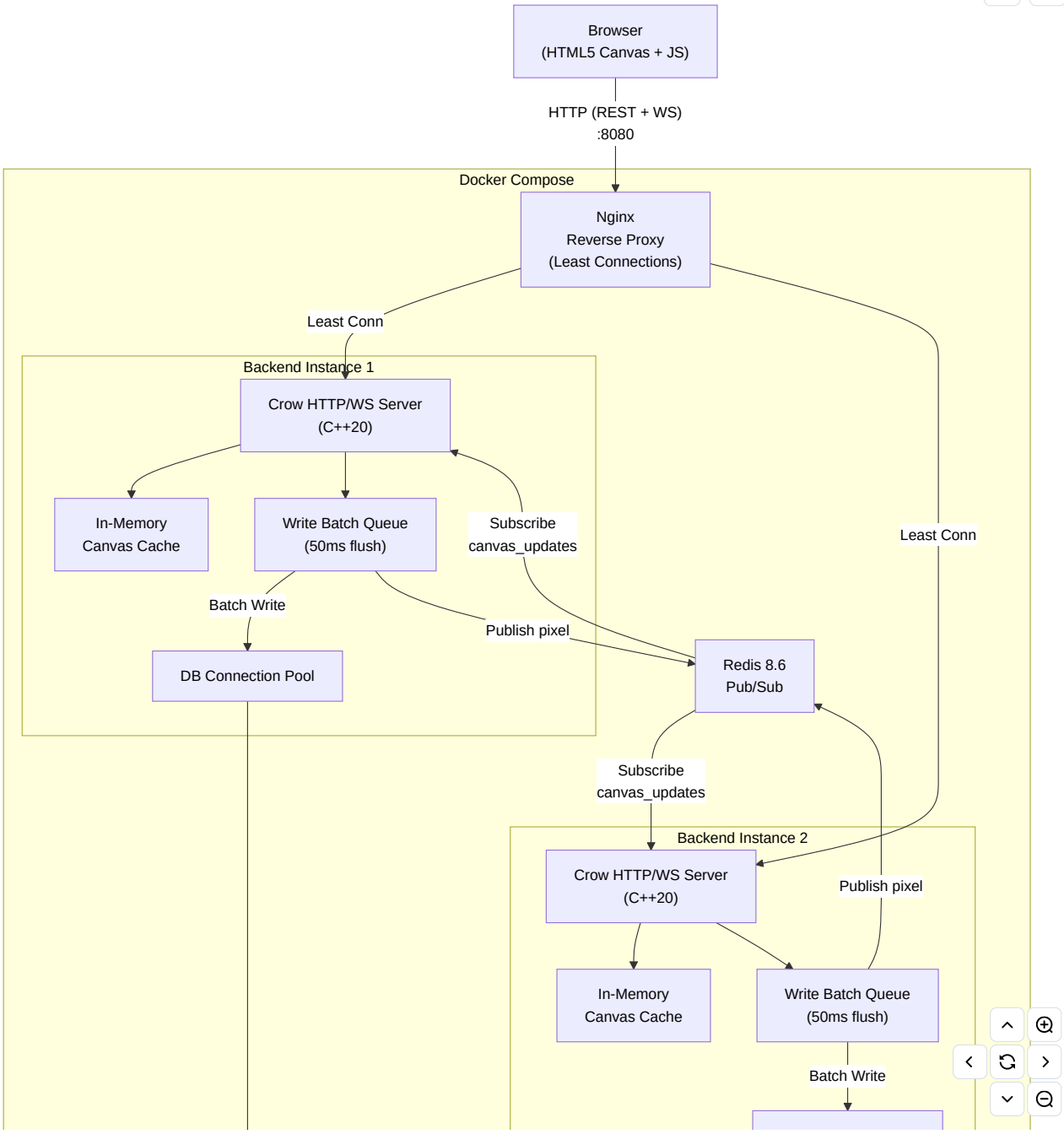
Architecture & Scalability

The system uses a low-latency, "read-optimized" architecture designed to handle thousands of concurrent users:

- **In-Memory Cache:** The backend serves the 50x50 canvas directly from memory, eliminating database scans for the read path.
- **Write Batching:** Pixel updates are queued and persisted to the PostgreSQL database in batches every 50ms to reduce I/O overhead.
- **Real-Time Synchronization:** Redis Pub/Sub synchronizes state across multiple backend instances behind a least-connections Nginx load balancer.
- **Fault Tolerance:** Automatic database connection pooling with a 2-second timeout and health checks ensures resilience during

node failures or restarts.

System Architecture



Auth Flow (Access + Refresh Tokens)

Performance Benchmarks

- **Extreme Scalability:** 1,000 concurrent Virtual Users (VUs) with a 99.98% success rate.
- **Throughput:** 101,449+ pixel updates per second broadcasted via Redis with 1,000 concurrent VUs.
- **Low Latency:** 2.07ms - 5.12ms average request duration at baseline load (100 VUs).

- **Efficiency:** 2.2GB of real-time JSON traffic handled under high load (500 VUs).

Features

- **Distributed Backends:** Scalable C++ instances behind Nginx.
- **JWT Security:** Access/refresh token authentication with automatic token renewal. Passwords hashed with Argon2 (libsodium).
- **Persistent Storage:** PostgreSQL stores user accounts and the canvas state.
- **Docker-Ready:** Deploys with a single command via Docker Compose.

Technical Stack

- **Frontend:** HTML5 Canvas, Vanilla JS, CSS
- **Backend:** C++20 (Crow Microframework)
- **Database:** PostgreSQL 18
- **Messaging:** Redis 8.6
- **Auth:** Argon2 (libsodium), JWT (jwt-cpp)
- **Orchestration:** Docker, Nginx

Prerequisites

- **Docker** and **Docker Compose**

Environment Variables

Variable	Description
`JWT_SECRET`	Secret key for signing JWT tokens
`POSTGRES_USER`	PostgreSQL username
`POSTGRES_PASSWORD`	PostgreSQL password
`POSTGRES_DB`	PostgreSQL database name
`DB_HOST`	PostgreSQL hostname (use `db` for Docker Compose)

Deployment

1. **Configure Environment:** `cp .env.example .env` and fill in the variables above.
2. **Run with Docker Compose:** `docker-compose up --build -d`
3. **Access App:** `http://localhost:8080`

API Reference

Method	Endpoint	Auth	Description
`POST`	`/register`	None	Create a new user. Body: `{ "username", "password" }`
`POST`	`/login`	None	Authenticate and receive tokens. Returns `{ "access_token", "refresh_token" }`
`POST`	`/refresh`	None	Exchange a refresh token for a new access token. Body: `{ "refresh_token" }`
`GET`	`/canvas`	Bearer (access)	Retrieve the full 50x50 canvas state as JSON
`WS`	`/ws?token=<access_token>`	Query param	Real-time pixel updates. Send: `{ "x", "y", "color" }`

Testing

Load Testing

To execute the load test using **k6**:

```
k6 run tests/loadtest.js
```

The test simulates a complete user lifecycle: registration, login, canvas retrieval, and real-time drawing via WebSockets.

Failover Demo

Stop a backend instance: `docker stop pixelcanvas-backend1-1` . The system will continue to function as Nginx reroutes traffic and clients automatically reconnect.

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Contributors 1

FinnSteimle

Languages

C++ 59.9% **JavaScript** 20.5% **CSS** 6.0% **CMake** 6.0% **HTML** 5.3% **Dockerfile** 2.3%

Suggested workflows

Based on your tech stack

C/C++ with Make Configure

Build and test a C/C++ project using Make.

Node.js Configure

Build and test a Node.js project with npm.

Jekyll using Docker image Configure

Package a Jekyll site using the jekyll/builder Docker image.

More workflows Dismiss suggestions