**Run Ingestion Server – Windows Setup, Troubleshooting & Optimization Report**

This report documents **exactly what was done** to set up, troubleshoot, optimize, and verify the stripped-down **Run Ingestion Server** on Windows using Git Bash / PowerShell and Docker.  
It is structured to clearly align each action with the intended requirement.

**1) Understanding the Requirements**

**Our Goal:**  
Improve performance and scalability **without breaking core features**.

**Requirements kept functional:**

* POST /runs — must still accept batches of runs.
* GET /runs/{id} — must retrieve a single run by ID.
* All runs in a batch must still be uploaded to **MinIO/S3**.
* Must demonstrate measurable performance improvement (benchmarks).

**Allowed changes:**

* Adjusting request/response formats.
* Optimizing database queries and S3 access.
* Introducing caching.
* Restructuring schema/logic for efficiency.

**2) Setting Up the Development Environment**

**2.1 Project Setup**

* Cloned project (or unzipped if provided).
* Ensured **first commit was the unmodified starter code**.

git clone https://github.com/Manoja444/LangSmith---Manoja.git

cd LangSmith---Manoja

**2.2 Dependency Installation (Windows-safe)**

* Problem: memray dependency does **not support Windows** → install failed.
* Solution: Installed only runtime dependencies.

poetry install --only main

* Alternative permanent fix in pyproject.toml:

toml

memray = { version = "^1.17", markers = "sys\_platform != 'win32'"

**2.3 Database & MinIO Services**

* **Original issue:** Port **9002** was already in use → container failed to start.
* **Fix:** Changed MinIO ports to **9100** (API) and **9101** (Console) in docker-compose-db.yaml

ports:

- "9004:9006”

- "9005:9007"

* Brought up containers:

docker compose -f docker-compose-db.yaml up -d

* Configured .env / settings to match new ports:

S3\_ENDPOINT\_URL=http://localhost:8000

S3\_ACCESS\_KEY=minioadmin1

S3\_SECRET\_KEY=minioadmin1

* Created MinIO bucket from console:http://localhost:8000

**2.4 Database Migrations**

poetry run alembic upgrade head

**2.5 Running the Server (Windows)**

* **Issue:** uvicorn not recognized in Git Bash.
* **Fix:** Used Python module form.

poetry run python -m uvicorn ls\_py\_handler.main:app --host 0.0.0.0 --port 8000 –reload

**2.6 Quick API Verification**

POST request via postman with request body payload below.

The api url : http://localhost:8000/runs

[

  {

    "trace\_id": "944ce838-b5c5-4628-8f23-089fbda8b9e3",

    "name": "Weather Query",

    "inputs": {

      "query": "What is the weather in San Francisco?"

    },

    "outputs": {

      "response": "It is currently 65°F and sunny in San Francisco."

    },

    "metadata": {

      "model": "gpt-4",

      "temperature": 0.7,

      "tokens": 42

    }

  }

]

The response from postman

[

    {

        "id": "ddbffe26-6039-4710-8783-400c26fa023a",

        "trace\_id": "944ce838-b5c5-4628-8f23-089fbda8b9e3",

        "name": "Weather Query",

        "inputs": {

            "query": "What is the weather in San Francisco?"

        },

        "outputs": {

            "response": "It is currently 65°F and sunny in San Francisco."

        },

        "metadata": {

            "model": "gpt-4",

            "temperature": 0.7,

            "tokens": 42

        }

    }

]

A screenshot of a computer

AI-generated content may be incorrect.

**3) Establishing the Baseline**

* Benchmarked before any changes (or closest possible on Windows without memray).
* Saved results for later comparison:
  + Requests/sec
  + Latency
  + Memory usage

**4) Exploring Current Implementation**

* Reviewed ls\_py\_handler/main.py (FastAPI endpoints).
* Identified:
  + **Large JSON serialization** of entire batch.
  + **Multiple DB inserts per run** (slow for big batches).
  + **Multiple S3 GET calls** when retrieving runs.

**5) Optimization Opportunities (Planned)**

* **Serialization:** Switch to NDJSON (one run per line, track byte offsets).
* **DB:** Use bulk insert with asyncpg.copy\_records\_to\_table; add index on trace\_id.
* **S3:** Fetch run data with a single **byte-range GET**.
* **Caching:** Implement small in-memory TTL cache for frequently accessed runs.

**6) Implementing Optimizations (What We Did)**

**6.1 Schema Changes**

Added byte range storage in DB:

Sql

ALTER TABLE runs

ADD COLUMN IF NOT EXISTS s3\_key text,

ADD COLUMN IF NOT EXISTS byte\_start bigint,

ADD COLUMN IF NOT EXISTS byte\_end bigint;

CREATE INDEX IF NOT EXISTS runs\_trace\_id\_idx ON runs(trace\_id);

**6.2 POST /runs Changes**

* Converted incoming runs → NDJSON buffer (one serialization pass).
* Uploaded **single object** to MinIO.
* Stored (id, trace\_id, name, s3\_key, byte\_start, byte\_end) for each run via **bulk insert**.

**6.3 GET /runs/{id} Changes**

* Queried DB for byte offsets + S3 key.
* Performed **one byte-range GET** from MinIO.
* Parsed JSON, normalized IDs, cached for **30 seconds**.

**6.4 Windows-specific Fixes**

* Skipped memray.
* Changed MinIO ports to avoid conflicts.
* Used python -m uvicorn to start server.

**7) Testing**

After each change:

poetry run pytest

Benchmarked to confirm performance improvement.

**8) Measuring Improvements**

Target table for results:

| **Scenario** | **Requests/sec** | **p95 Latency (ms)** | **Peak RSS (MB)** |
| --- | --- | --- | --- |
| Baseline POST |  |  |  |
| Optimized POST (NDJSON + bulk) |  |  |  |
| Baseline GET |  |  |  |
| Optimized GET (range) |  |  |  |
| Optimized GET (cache hit) |  |  |  |

**9) Preparing Final Submission**

* Optimized code merged into main branch.
* README updated with:
  + What was changed.
  + Why it’s faster.
  + How to reproduce.
* Git history kept clean:
  + Commit 1 = template.
  + Later commits = focused changes (schema, POST, GET, cache).

**Appendix — Key Troubleshooting Commands**

**Find & kill process using a port (Windows):**

powershell

netstat -aon | findstr :9002

taskkill /PID <PID> /F

**Check Docker container ports:**

bash

docker ps --format "table {{.Names}}\t{{.Ports}}"

Bottom of Form

A screen shot of a computer

AI-generated content may be incorrect.