Pipeline Runner: Technical Architecture & Design Document

Table of Contents

---

1. Project Overview

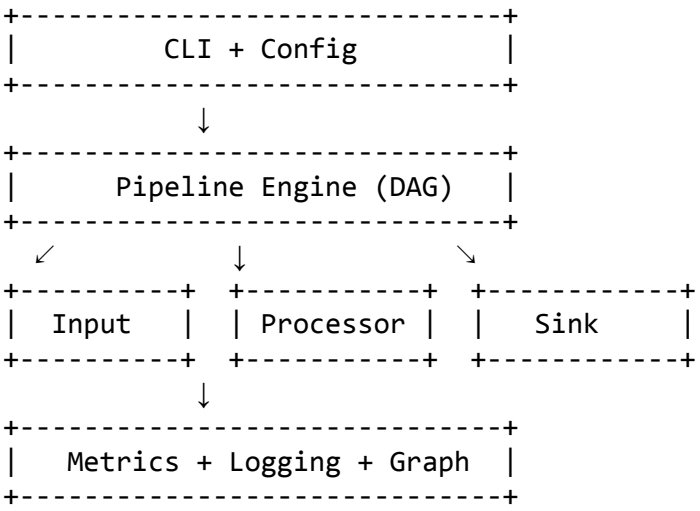
Pipeline Runner is a real-time, multithreaded, DAG-configurable signal processing application written in C++17. It ingests streaming data (e.g., simulated sine wave), processes it through a flexible graph of stages (FFT, filters, sinks), and delivers output to files or monitoring systems — all defined via JSON configuration.

2. Key Features

- Stream-based data ingestion
- DAG-based configurable pipelines
- Real-time FFT and filtering
- Multiple output sinks
- Logging and metrics
- Graceful shutdown and thread-safe queues
- Containerized via Docker

3. Architecture

High-Level Layers



- **Input:** Stream generator (e.g., sine wave)
- **Processor:** FFT, filters
- **Sinks:** File sink, logger sink
- **Engine:** Executes the DAG per config
- **CLI:** Parses options and launches threads
- **Monitoring:** Logs, metrics, .dot graph

4. Execution Pipeline: DAG Design

- The pipeline is modeled as a Directed Acyclic Graph (DAG).
- Each node has a name, type (FFT, Filter, Sink), and may have multiple next connections.
- Execution begins at the `start_node` and traverses the DAG recursively.
- Supports both linear and branched pipelines.

5. Core Components & Algorithms

- **Sine Wave Generator:** Generates samples using `value = amplitude * sin(2π * frequency * t)` and adds a timestamp.
- **FFT (using FFTW):** Transforms time-domain input to frequency-domain using `fftw_plan_r2r_1d`.
- **Filter:** Removes values outside `[min, max]`.
- **Sinks:**
  - FileSink: writes data to text files
  - LoggerSink: logs data for debugging (reuses FileSink)

6. Libraries Used

Library	Purpose
nlohmann/json	JSON config parsing
spdlog	High-performance logging
fftw3	Fast Fourier Transform
fmt	Formatting (used by spdlog)
CLI11	CLI interface/arg parsing
std::thread	Multithreading
std::atomic	Thread-safe counters/flags
filesystem	Output directory mgmt

7. Codebase Structure

```
pipeline_runner/
├── include/
│   ├── Data.h
│   ├── BlockingQueue.h
│   ├── OutputSink.h
│   ├── Logger.h
│   ├── PipelineEngine.h
│   └── Metrics.h
├── src/
│   ├── main.cpp
│   ├── SineWaveGenerator.cpp
│   ├── FFT.cpp
│   ├── Filter.cpp
│   ├── FileSink.cpp
│   └── PipelineEngine.cpp
├── configs/
│   └── pipeline.json
├── build/
│   └── output/     ← runtime logs, output, metrics
├── Dockerfile
├── README.md
├── CMakeLists.txt
└── .gitignore
```

8. File Responsibilities

- **main.cpp:** CLI interface, loads config, starts threads, graceful shutdown
- **PipelineEngine.h/cpp:** Parses DAG config, maps node names, executes pipeline
- **SineWaveGenerator.cpp:** Generates sine wave samples
- **FFT.cpp:** Performs FFT
- **Filter.cpp:** Applies min/max filter
- **FileSink.cpp:** Writes output values to text file, logs to pipeline.log
- **Logger.h:** Initializes and routes spdlog to output/pipeline.log

- **Metrics.h:** Tracks counts, writes to metrics.json
- **BlockingQueue.h:** Thread-safe FIFO queue

9. Monitoring & Observability

- `metrics.json`: shows counts of processed batches
- `pipeline_graph.dot`: DOT file for Graphviz
- `pipeline.log`: runtime logs with thread IDs
- CLI shows live progress

10. Containerization with Docker

- Fully containerized using the Dockerfile.
- **Build:**

```
docker build -t pipeline-runner .
```

- **Run:**

```
docker run --rm -v $(pwd)/build/output:/app/output pipeline-runner
```

11. Configuration Format (pipeline.json)

```
{
  "start_node": "fft",
  "input": { "frequency": 2.0, "amplitude": 1.0, "rate": 100 },
  "pipeline": [
    { "name": "fft", "type": "FFT", "next": ["filter1", "filter2"] },
    { "name": "filter1", "type": "Filter", "min": 0.1, "max": 1.0, "next":
["file_sink"] },
    { "name": "filter2", "type": "Filter", "min": 0.0, "max": 0.5, "next":
["logger_sink"] },
    { "name": "file_sink", "type": "FileSink", "path": "output.txt" },
    { "name": "logger_sink", "type": "FileSink", "path": "debug.log" }
  ]
}
```

12. Conclusion

Pipeline Runner is a scalable, modular, and highly configurable real-time data processing system built in C++. It uses a clean, DAG-based architecture to support flexible and parallel transformations of input data streams.

Key engineering strengths: - Thread-safe design - Minimal latency via in-memory queues - Modular pipeline configuration - Strong logging and observability - Dockerized for easy deployment

This system demonstrates how C++ can be used to build robust data pipelines with modern architecture patterns like DAGs, multithreading, and containerization.