Getting Started with MyTSDB

This guide will help you get MyTSDB up and running from scratch. All instructions are designed to be copy-paste ready.

Table of Contents

Prerequisites

Required

- **C++17 compatible compiler**
  - GCC 9+ (Linux)
  - Clang 10+ (macOS/Linux)
  - MSVC 2019+ (Windows)
- **CMake 3.15+**
- **Make** (or Ninja)

Optional (but recommended)

- **Intel TBB** (Threading Building Blocks) - For concurrent data structures
- **spdlog** - For logging support
- **Google Test** - For running tests (automatically enabled if found)
- **gRPC & Protobuf** - For RPC support

Installation

macOS (using Homebrew)

```
# Install required dependencies
brew install cmake

# Install optional but recommended dependencies
brew install tbb spdlog googletest grpc protobuf
```

Linux (Ubuntu/Debian)

```
# Update package list
sudo apt-get update

# Install required dependencies
sudo apt-get install -y build-essential cmake

# Install optional but recommended dependencies
sudo apt-get install -y libtbb-dev libspdlog-dev libgtest-dev libgrpc++-
dev protobuf-compiler libprotobuf-dev
```

Linux (Fedora/RHEL/CentOS)

```
# Install required dependencies
sudo dnf install -y gcc-c++ cmake make

# Install optional but recommended dependencies
sudo dnf install -y tbb-devel spdlog-devel gtest-devel grpc-devel
protobuf-devel protobuf-compiler
```

Building the Project

Quick Start (Recommended)

```
# Clone the repository
git clone https://github.com/yourusername/mytsdb.git
cd mytsdb

# Build everything (configure + build)
make

# Run all tests
make test-all
```

That's it! The `make` command will:

1. Configure CMake (create build directory)
2. Build all components
3. Enable tests by default

Step-by-Step Build

If you prefer to build step by step:

```
# 1. Clone the repository
git clone https://github.com/yourusername/mytsdb.git
```

```
cd mytsdb

# 2. Configure CMake (creates build directory)
make configure

# 3. Build all components
make build

# 4. Verify build succeeded
ls -la build/src/libtsdb_lib.*
```

Clean Build

To start fresh:

```
# Clean everything (build directory, cache, etc.)
make clean-all

# Rebuild from scratch
make rebuild
```

Build Options

The Makefile uses these CMake flags by default:

- `-DCMAKE_BUILD_TYPE=Release` - Release build with optimizations
- `-DBUILD_TESTS=ON` - Enable test building
- `-DTSDB_SEMVEC=OFF` - Disable semantic vector components

To customize, edit the `CMAKE_FLAGS` variable in the `Makefile` or run CMake directly:

```
cd build
cmake .. -DCMAKE_BUILD_TYPE=Debug -DBUILD_TESTS=ON
make -j$(nproc)
```

Running Tests

Quick Test Run

```
# Run all tests (453+ tests, takes ~7-8 minutes)
make test-all
```

Test Categories

```
# Unit tests only (357 tests, ~2-3 minutes)
make test-unit

# Integration tests only (177 tests, ~5-6 minutes)
make test-integration

# Specific test suites
make test-core-unit          # Core unit tests (38 tests)
make test-storage-unit       # Storage unit tests (60 tests)
make test-cache-unit         # Cache unit tests (28 tests)
make test-compression-unit   # Compression unit tests (19 tests)
make test-histogram-unit     # Histogram unit tests (22 tests)
```

Background Test Execution

For long-running test suites (useful for overnight runs):

```
# Start tests in background (macOS - prevents hibernation)
make test-background-caffeinate

# Check test status
make test-background-status

# Stop tests if needed
make test-background-stop
```

The test output will be saved to `test_results/background_test_<timestamp>.log`.

Test Results

Test results are displayed in the terminal. For detailed output:

```
# Run tests with verbose output
cd build
ctest --output-on-failure -V
```

Using the Library

Basic Example

Create a file `example.cpp`:

```
#include "tsdb/storage/storage_impl.h"
#include "tsdb/core/types.h"
#include <iostream>
```

```cpp
using namespace tsdb;

int main() {
    // Configure storage
    core::StorageConfig config = core::StorageConfig::Default();
    config.data_dir = "./tsdb_data";

    // Create storage instance
    storage::StorageImpl storage(config);

    // Initialize
    auto init_result = storage.init(config);
    if (!init_result.ok()) {
        std::cerr << "Init failed: " << init_result.error() << std::endl;
        return 1;
    }

    // Create time series
    core::Labels labels;
    labels.add("__name__", "cpu_usage");
    labels.add("host", "server1");
    core::TimeSeries series(labels);
    series.add_sample(std::time(nullptr) * 1000, 0.75);

    // Write
    auto write_result = storage.write(series);
    if (!write_result.ok()) {
        std::cerr << "Write failed: " << write_result.error() <<
std::endl;
        return 1;
    }

    // Read
    auto read_result = storage.read(labels, 0, std::time(nullptr) * 1000);
    if (read_result.ok()) {
        const auto& samples = read_result.value().samples();
        std::cout << "Read " << samples.size() << " samples" << std::endl;
        for (const auto& sample : samples) {
            std::cout << "  Time: " << sample.timestamp()
                      << ", Value: " << sample.value() << std::endl;
        }
    }

    // Close
    storage.close();
    return 0;
}
```

Compile and Run

```
```

```
# Build your example
cd build
g++ -std=c++17 -I../include -L./src -ltsdb_lib ../example.cpp -o example

# Run
./example
```

Or use CMake (recommended):

```
# Add to examples/CMakeLists.txt or create your own CMakeLists.txt
# Then build with: make
```

Project Structure

```
mytsdb/
├── include/tsdb/          # Public headers
│   ├── core/              # Core types and interfaces
│   ├── storage/           # Storage implementation
│   ├── query/             # Query engine
│   └── histogram/         # Histogram support
├── src/tsdb/              # Implementation source files
├── test/                  # Test suite
│   ├── unit/              # Unit tests
│   └── integration/       # Integration tests
├── examples/              # Example code
├── docs/                  # Documentation
├── CMakeLists.txt         # CMake configuration
├── Makefile               # Convenience Makefile
└── README.md              # Project overview
```

Available Makefile Targets

Build Targets

```
make                    # Configure and build (default)
make configure          # Run CMake configuration only
make build              # Build all components
make rebuild            # Clean everything and rebuild
make install            # Install the library
```

Clean Targets

```
make clean              # Clean build artifacts (keep configuration)
make clean-all          # Complete clean (remove build dir and cache)
make test-clean         # Clean test results only
```

Test Targets

```
make test-all                   # All tests (453+ tests)
make test-unit                  # Unit tests (357 tests)
make test-integration           # Integration tests (177 tests)
make test-core-unit             # Core unit tests
make test-storage-unit          # Storage unit tests
make test-cache-unit            # Cache unit tests
make test-compression-unit      # Compression unit tests
make test-histogram-unit        # Histogram unit tests
make test-background            # Run tests in background
make test-background-status     # Check background test status
make test-background-stop       # Stop background tests
```

Help

```
make help               # Show all available targets
```

Troubleshooting

Build Issues

Problem: CMake can't find dependencies (TBB, spdlog, etc.)

Solution: Install missing dependencies (see Installation section). The project will build without optional dependencies, but some features will be disabled.

Problem: Build fails with "undefined reference" errors

Solution: Make sure you're linking against the built library:

```
# Rebuild from scratch
make clean-all
make rebuild
```

Problem: Tests fail to build

Solution: Install Google Test:

```
# macOS
brew install googletest

# Linux (Ubuntu/Debian)
sudo apt-get install -y libgtest-dev
```

Runtime Issues

Problem: Library not found at runtime

Solution: Add build directory to library path:

```
# Linux
export LD_LIBRARY_PATH=$PWD/build/src:$LD_LIBRARY_PATH

# macOS
export DYLD_LIBRARY_PATH=$PWD/build/src:$DYLD_LIBRARY_PATH
```

Problem: Tests timeout

Solution: Some tests may take longer on slower systems. Check test timeouts in `test/CMakeLists.txt` or run tests individually:

```
make test-core-unit  # Faster test suite
```

Performance Issues

Problem: Build is slow

Solution: Use parallel builds (default) or increase parallelism:

```
make build -j8  # Use 8 parallel jobs
```

Next Steps

- Read the Architecture Overview
- Check out Example Code
- Review API Documentation (TODO)
- See Test Status for current test results

Getting Help

- **Issues:** GitHub Issues
- **Documentation:** See docs/ directory
- **Test Status:** See docs/planning/FAILING_TESTS_FIX_PLAN.md

---

Happy coding! 🚀