Distributed KV Store (C++) — Complete Guide

1) What This Project Is

You get a minimal Redis-like key-value server written in C++17 with a simple text protocol over TCP. It supports PING, GET, SET, DEL, and durable append-only file (AOF) persistence. A tiny CLI client is included.
The codebase is intentionally compact and dependency-light so you can read, run, and extend it offline.

2) System Requirements & Setup

Operating System: Linux (Ubuntu 22.04+ recommended). Toolchain: g++, cmake, make.
Install once:

```
sudo apt update
sudo apt install -y build-essential cmake git
```

Project layout:

```
kvstore-cpp/
  CMakeLists.txt
  config.yaml
  include/ (headers)
  src/ (sources)
  scripts/ (helper scripts)
  data/ (created at runtime; holds appendonly.aof)
```

3) Build Instructions (No Internet Needed)

Run the provided script:

```
cd kvstore-cpp/scripts
./build.sh
```

This creates `build/kvstore` (server) and `build/kvclient` (client).
If you prefer manual steps:

```
cd kvstore-cpp
mkdir -p build && cd build
cmake ..
make -j
```

4) Run the Server & Send Commands

Start server (reads config.yaml; creates data/ as needed):

```
cd kvstore-cpp
./scripts/run_server.sh
```

From another terminal, connect using the client:

```
echo -e "PING\nSET foo bar\nGET foo\n" | ./build/kvclient 127.0.0.1 6380
```

Expected responses:

```
PONG
OK
$ bar
```

You can also use `nc 127.0.0.1 6380` and type commands manually, one per line.

5) Configuration (config.yaml)

mode: reserved for future (leader/follower); leave as "leader".
host: bind address (default 0.0.0.0). port: TCP port (default 6380).
aof_path: path to append-only file for durability. max_clients: soft limit.
To use a different config file at runtime:

```
CONFIG_FILE=/path/to/alt.yaml ./build/kvstore
```

6) Text Protocol Reference

Each command is a single line terminated by '\n'. Supported commands:

```
PING
```

Distributed KV Store (C++) — Complete Guide

-> PONG
SET <key> <value...>
All text after <key> becomes the value (spaces preserved). -> OK
GET <key>
-> $ <value> (if exists) | (nil) if missing
DEL <key>
-> OK if key existed and was deleted; (nil) if it did not exist.
Error handling:
Unknown command or wrong arity -> 'ERR unknown/arity'

7) Persistence Model (AOF)

On every mutating command (SET/DEL), the server appends the operation to the file at aof_path (default data/appendonly.aof).
On startup, the server replays the file to reconstruct the in-memory map.
You can inspect the AOF with any text editor. To reset the database, stop the server and remove the AOF:

```
rm -f data/appendonly.aof
```

Then start the server again.

8) Using the CLI Client Interactively

The included `kvclient` lets you type commands and see responses.
Start it:

```
./build/kvclient 127.0.0.1 6380
```

Then type:

```
PING
SET a 1
GET a
DEL a
GET a
```

Press Ctrl+D to exit. This is useful for demos and manual testing.

9) Quick Demo Script (End-to-End)

For a one-shot demo, run:

```
./scripts/demo.sh
```

It builds the project, starts the server in the background, pipes a sequence of commands to the client,
prints responses, then shuts the server down. Use this to sanity-check your environment.

10) Simple Load Testing (Manual)

Open 3 terminals and run the following in parallel to simulate load:
Terminal A: start the server.
Terminal B: while true; do echo -e "SET k1 v1\nGET k1" | ./build/kvclient 127.0.0.1 6380 > /dev/null; done
Terminal C: while true; do echo -e "SET k2 v2\nGET k2" | ./build/kvclient 127.0.0.1 6380 > /dev/null; done
Watch server stdout for stability. You can add timestamps to measure latency if desired.

11) Logs & Troubleshooting

Symptom: 'listen failed' -> Port in use or insufficient permissions. Change 'port' in config.yaml.
Symptom: No response -> Firewall or SELinux may block. Test locally (127.0.0.1). Ensure newline at the end of each command.
Symptom: Data lost after restart -> Verify that commands appear in data/appendonly.aof and that the path is correct.
Symptom: Build errors -> Ensure g++ >= 9 and CMake >= 3.16; re-run ./scripts/build.sh to recreate

Distributed KV Store (C++) — Complete Guide

the build folder.

12) Extending the Server (Roadmap)

Easy extensions to showcase depth:
- LRU cache with a size limit; evict on SET when memory threshold is exceeded.
- Expiring keys: store TTL with each entry and lazily expire on GET.
- Binary protocol: reduce parsing overhead and support length-prefixed values.
- Sharding: route keys using consistent hashing across multiple processes.
- Replication: implement a leader that streams write operations to followers.
- Snapshotting: periodically write an RDB-style snapshot to speed cold start.

13) File Overview & Key Functions

src/server.cpp
  - load_config: minimal parser for config.yaml
  - create_listener, accept_loop, client_loop: networking and concurrency
  - handle_command: parses and executes GET/SET/DEL/PING; appends to AOF
src/store.cpp
  - thread-safe map with std::mutex guarding GET/SET/DEL
src/persistence.cpp
  - append: writes each mutation to AOF
  - replay: reconstructs in-memory state at startup
src/client.cpp
  - tiny interactive client for manual testing

14) Security Notes (Local Dev)

This server is for learning purposes. It does not implement authentication, TLS, or access control. Do not expose it to the internet. For remote tests, use SSH port forwarding or run behind a firewall/VPN.

15) One-Page Quickstart (Copy/Paste)

```
sudo apt update && sudo apt install -y build-essential cmake git
# unzip the project, then:
cd kvstore-cpp/scripts && ./build.sh
cd .. && ./scripts/run_server.sh   # server on 0.0.0.0:6380
# new terminal:
echo -e "PING\nSET foo bar\nGET foo\n" | ./build/kvclient 127.0.0.1 6380
# check durability:
pkill kvstore || true
./scripts/run_server.sh
echo -e "GET foo\n" | ./build/kvclient 127.0.0.1 6380  # -> $ bar
```