readme.md for @push.rocks/smartdb

A MongoDB-compatible embedded database server powered by Rust □□⚡ — use the official `mongodb` driver and it just works. No binary downloads, instant startup, zero config.

Install

```
pnpm add @push.rocks/smartdb
# or
npm install @push.rocks/smartdb
```

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

What It Does

`@push.rocks/smartdb` is a **real database server** that speaks the wire protocol used by MongoDB drivers. The core engine is written in Rust for high performance, with a thin TypeScript orchestration layer. Connect with the standard `mongodb` Node.js driver — no mocks, no stubs, no external binaries required.

Why SmartDB?

	SmartDB	External DB Server

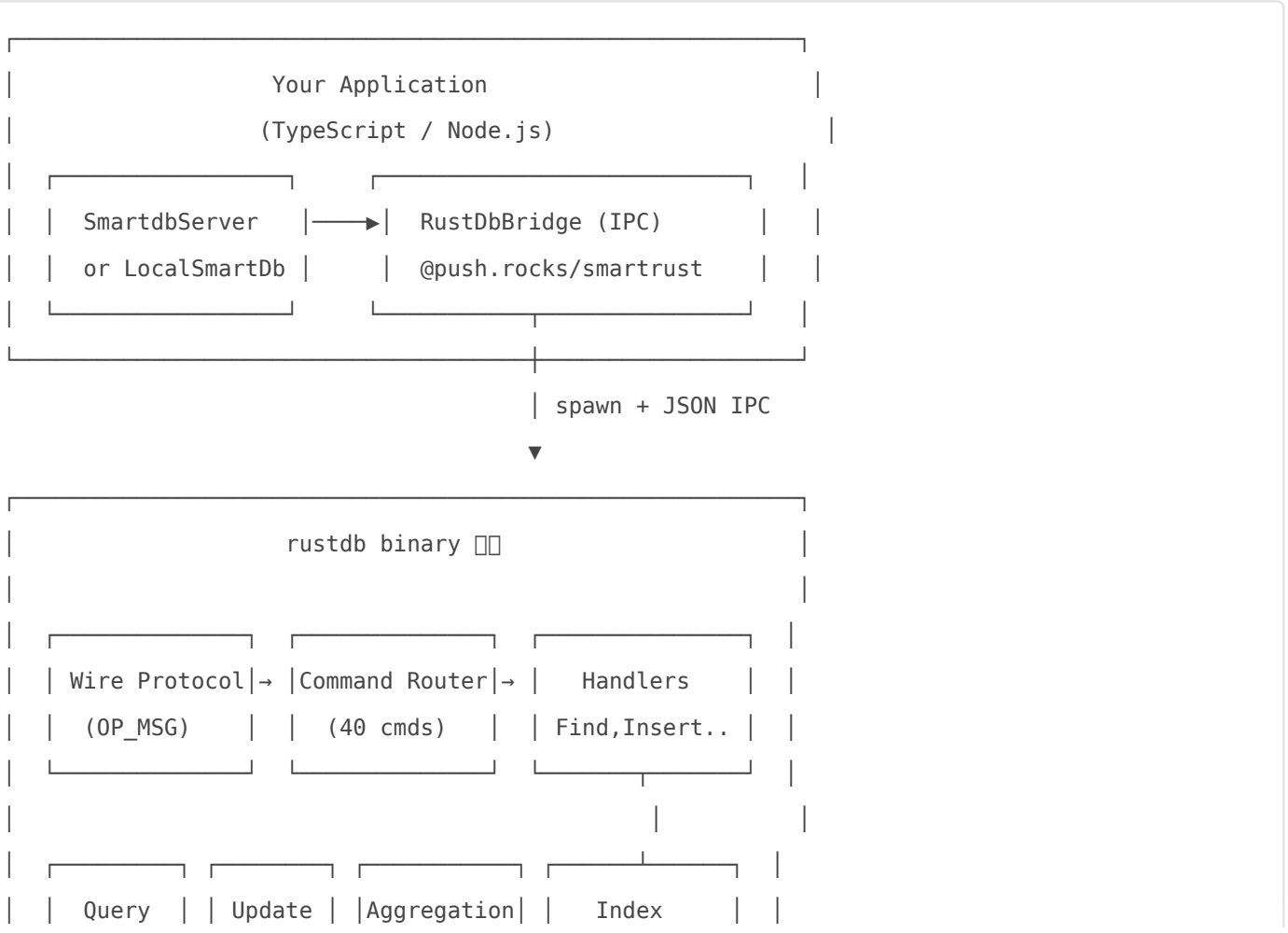
Startup time	~30ms	~2-5s
Binary download	Bundled (~7MB)	~200MB+
Install	`pnpm add`	System package / Docker
Persistence	Memory or file-based	Full disk engine
Perfect for	Unit tests, CI/CD, prototyping, local dev, embedded	Production at scale

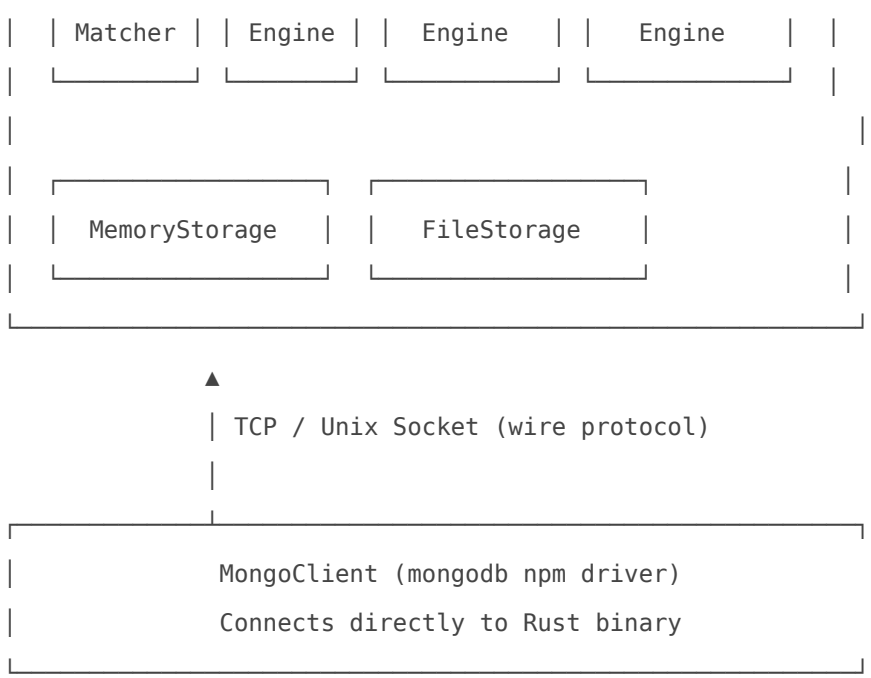
Two Ways to Use It

- □□ `SmartdbServer` — Full control. Configure port, host, storage backend, Unix sockets. Great for test fixtures or custom setups.
- □□ `LocalSmartDb` — Zero-config convenience. Give it a folder path, get a persistent database over a Unix socket. Done.

Architecture: TypeScript + Rust □□

SmartDB uses the same **sidecar binary** pattern as @push.rocks/smartproxy:





The TypeScript layer handles **lifecycle only** (start/stop/configure via IPC). All database operations flow directly from the `MongoClient` to the Rust binary over TCP or Unix sockets — **zero per-query IPC overhead**.

Quick Start

Option 1: LocalSmartDb (Zero Config) □□

The fastest way to get a persistent local database:

```
import { LocalSmartDb } from '@push.rocks/smartdb';
import { MongoClient } from 'mongodb';

// Point it at a folder — that's it
const db = new LocalSmartDb({ folderPath: './my-data' });
const { connectionUri } = await db.start();

// Connect with the standard driver
const client = new MongoClient(connectionUri, { directConnection: true });
await client.connect();

// Use it like any wire-protocol-compatible database
```

```
const users = client.db('myapp').collection('users');
await users.insertOne({ name: 'Alice', email: 'alice@example.com' });
const user = await users.findOne({ name: 'Alice' });
console.log(user); // { _id: ObjectId(...), name: 'Alice', email: 'alice@example.com' }

// Data persists to disk automatically — survives restarts!
await client.close();
await db.stop();
```

Option 2: SmartdbServer (Full Control) □□

```
import { SmartdbServer } from '@push.rocks/smartdb';
import { MongoClient } from 'mongodb';

// TCP mode
const server = new SmartdbServer({ port: 27017 });
await server.start();

const client = new MongoClient('mongodb://127.0.0.1:27017');
await client.connect();

const db = client.db('myapp');
await db.collection('users').insertOne({ name: 'Alice', age: 30 });
const user = await db.collection('users').findOne({ name: 'Alice' });

await client.close();
await server.stop();
```

API Reference

SmartdbServer

The core server class. Manages the Rust database engine and exposes connection details.

Constructor Options (`ISmartdbServerOptions`)

```
import { SmartdbServer } from '@push.rocks/smartdb';

// TCP mode (default)
const server = new SmartdbServer({
  port: 27017,              // Default: 27017
  host: '127.0.0.1',        // Default: 127.0.0.1
  storage: 'memory',        // 'memory' or 'file' (default: 'memory')
  storagePath: './data',     // Required when storage is 'file'
});

// Unix socket mode — no port conflicts!
const server = new SmartdbServer({
  socketPath: '/tmp/smartdb.sock',
  storage: 'file',
  storagePath: './data',
});

// Memory storage with periodic persistence
const server = new SmartdbServer({
  storage: 'memory',
  persistPath: './data/snapshot.json',
  persistIntervalMs: 30000, // Save every 30s
});
```

Methods & Properties

Method / Property	Type	Description
`start()`	`Promise<void>`	Start the server (spawns Rust binary)
`stop()`	`Promise<void>`	Stop the server and clean up
`getConnectionUri()`	`string`	Get the `mongodb://` connection URI
`running`	`boolean`	Whether the server is currently running
`port`	`number`	Configured port (TCP mode)
`host`	`string`	Configured host (TCP mode)
`socketPath`	`string | undefined`	Socket path (socket mode)

LocalSmartDb

Zero-config wrapper around SmartdbServer. Uses Unix sockets and file-based persistence.

Constructor Options (`ILocalSmartDbOptions`)

```
import { LocalSmartDb } from '@push.rocks/smartdb';

const db = new LocalSmartDb({
  folderPath: './data',                 // Required: data storage directory
  socketPath: '/tmp/custom.sock',       // Optional: custom socket (default: auto-generated)
});
```

Methods & Properties

Method / Property	Type	Description
`start()`	`Promise<ILocalSmartDbConnectionInfo>`	Start and return connection info
`stop()`	`Promise<void>`	Stop the server
`getConnectionInfo()`	`ILocalSmartDbConnectionInfo`	Get current connection info
`getConnectionUri()`	`string`	Get the connection URI
`getServer()`	`SmartdbServer`	Access the underlying server
`running`	`boolean`	Whether the server is running

Connection Info (`ILocalSmartDbConnectionInfo`)

```
interface ILocalSmartDbConnectionInfo {
  socketPath: string;      // e.g., /tmp/smartdb-abc123.sock
  connectionUri: string;   // e.g., mongodb://%2Ftmp%2Fsmartdb-abc123.sock
}
```

Supported Operations

SmartDB supports the core operations through the wire protocol. Use the standard `mongodb` driver — these all work:

CRUD

```
// Insert
await collection.insertOne({ name: 'Bob' });
await collection.insertMany([{ a: 1 }, { a: 2 }]);

// Find
const doc = await collection.findOne({ name: 'Bob' });
const docs = await collection.find({ age: { $gte: 18 } }).toArray();

// Update
await collection.updateOne({ name: 'Bob' }, { $set: { age: 25 } });
await collection.updateMany({ active: false }, { $set: { archived: true } });

// Delete
await collection.deleteOne({ name: 'Bob' });
await collection.deleteMany({ archived: true });

// Replace
await collection.replaceOne({ _id: id }, { name: 'New Bob', age: 30 });

// Find and Modify
await collection.findOneAndUpdate({ name: 'Bob' }, { $inc: { visits: 1 } }, { returnDocument:
'after' });
await collection.findOneAndDelete({ expired: true });
await collection.findOneAndReplace({ _id: id }, { name: 'Replaced' }, { returnDocument:
'after' });
```

Query Operators

```
// Comparison
{ age: { $eq: 25 } }     { age: { $ne: 25 } }
{ age: { $gt: 18 } }     { age: { $lt: 65 } }
{ age: { $gte: 18 } }    { age: { $lte: 65 } }
{ status: { $in: ['active', 'pending'] } }
{ status: { $nin: ['deleted'] } }

// Logical
{ $and: [{ age: { $gte: 18 } }, { active: true }] }
{ $or: [{ status: 'active' }, { admin: true }] }
```

```
{ $not: { status: 'deleted' } }

// Element
{ email: { $exists: true } }
{ type: { $type: 'string' } }

// Array
{ tags: { $all: ['mongodb', 'database'] } }
{ scores: { $elemMatch: { $gte: 80, $lt: 90 } } }
{ tags: { $size: 3 } }

// Regex
{ name: { $regex: /^Al/i } }
```

Update Operators

```
{ $set: { name: 'New Name' } }
{ $unset: { tempField: '' } }
{ $inc: { count: 1 } }
{ $mul: { price: 1.1 } }
{ $min: { low: 50 } }        { $max: { high: 100 } }
{ $push: { tags: 'new' } }   { $pull: { tags: 'old' } }
{ $addToSet: { tags: 'unique' } }
{ $pop: { queue: 1 } }       // Remove last
{ $pop: { queue: -1 } }      // Remove first
{ $rename: { old: 'new' } }
{ $currentDate: { lastModified: true } }
```

Aggregation Pipeline

```
const results = await collection.aggregate([
  { $match: { status: 'active' } },
  { $group: { _id: '$category', total: { $sum: '$amount' } } },
  { $sort: { total: -1 } },
  { $limit: 10 },
  { $project: { category: '$_id', total: 1, _id: 0 } },
]).toArray();
```

Supported stages: `$match`, `$project`, `$group`, `$sort`, `$limit`, `$skip`, `$unwind`, `$lookup`, `$addFields`, `$count`, `$facet`, `$replaceRoot`, `$set`, `$unionWith`, `$out`, `$merge`

Group accumulators: `$sum`, `$avg`, `$min`, `$max`, `$first`, `$last`, `$push`, `$addToSet`, `$count`

Indexes

```
await collection.createIndex({ email: 1 }, { unique: true });
await collection.createIndex({ name: 1, age: -1 });    // compound
await collection.createIndex({ field: 1 }, { sparse: true });
const indexes = await collection.listIndexes().toArray();
await collection.dropIndex('email_1');
await collection.dropIndexes();  // drop all except _id
```

Database & Admin

```
await db.listCollections().toArray();
await db.createCollection('new');
await db.dropCollection('old');
await db.dropDatabase();
await db.stats();

const admin = client.db().admin();
await admin.listDatabases();
await admin.ping();
await admin.serverStatus();
```

Bulk Operations

```
const result = await collection.bulkWrite([
  { insertOne: { document: { name: 'Bulk1' } } },
  { updateOne: { filter: { name: 'X' }, update: { $set: { bulk: true } } } },
  { deleteOne: { filter: { name: 'Expired' } } },
]);
```

Count & Distinct

```
const count = await collection.countDocuments({ status: 'active' });
const estimated = await collection.estimatedDocumentCount();
const names = await collection.distinct('name');
```

Wire Protocol Commands

Category	Commands
Handshake	`hello`, `isMaster`, `ismaster`
CRUD	`find`, `insert`, `update`, `delete`, `findAndModify`, `getMore`, `killCursors`
Aggregation	`aggregate`, `count`, `distinct`
Indexes	`createIndexes`, `dropIndexes`, `listIndexes`
Sessions	`startSession`, `endSessions`
Transactions	`commitTransaction`, `abortTransaction`
Admin	`ping`, `listDatabases`, `listCollections`, `drop`, `dropDatabase`, `create`, `serverStatus`, `buildInfo`, `dbStats`, `collStats`, `connectionStatus`, `currentOp`, `renameCollection`

Compatible with wire protocol versions 0–21 (driver versions 3.6 through 7.0).

Rust Crate Architecture □□

The Rust engine is organized as a Cargo workspace with 8 focused crates:

Crate	Purpose
`rustdb`	Binary entry point: TCP/Unix listener, management IPC, CLI
`rustdb-config`	Server configuration types (serde, camelCase JSON)
`rustdb-wire`	Wire protocol parser/encoder (OP_MSG, OP_QUERY, OP_REPLY)
`rustdb-query`	Query matcher, update engine, aggregation, sort, projection
`rustdb-storage`	Storage backends (memory, file) + WAL + OpLog
`rustdb-index`	B-tree/hash indexes, query planner (IXSCAN/COLLSCAN)

Crate	Purpose
`rustdb-txn`	Transaction + session management with snapshot isolation
`rustdb-commands`	40 command handlers wiring everything together

Cross-compiled for `linux_amd64` and `linux_arm64` via @git.zone/tsrust.

Testing Example

```
import { expect, tap } from '@git.zone/tstest/tapbundle';
import { SmartdbServer } from '@push.rocks/smartdb';
import { MongoClient } from 'mongodb';

let server: SmartdbServer;
let client: MongoClient;

tap.test('setup', async () => {
  server = new SmartdbServer({ port: 27117 });
  await server.start();
  client = new MongoClient('mongodb://127.0.0.1:27117', { directConnection: true });
  await client.connect();
});

tap.test('should insert and find', async () => {
  const col = client.db('test').collection('items');
  await col.insertOne({ name: 'Widget', price: 9.99 });
  const item = await col.findOne({ name: 'Widget' });
  expect(item?.price).toEqual(9.99);
});

tap.test('teardown', async () => {
  await client.close();
  await server.stop();
});

export default tap.start();
```

License and Legal Information

This repository contains open-source code licensed under the MIT License. A copy of the license can be found in the license file.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH or third parties, and are not included within the scope of the MIT license granted herein.

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #3
Created 2026-03-28 11:13:49 UTC by foss.global Team
Updated 2026-03-28 12:20:35 UTC by foss.global Team