@push.rocks/mongodump

A tool to create and manage dumps of MongoDB databases, supporting data export and import.

- readme.md for @push.rocks/mongodump
- changelog.md for @push.rocks/mongodump

readme.md for @push.rocks/mongodump

A powerful MongoDB backup and restore tool □□

npm version license TypeScript ES Module

What it does □□

`@push.rocks/mongodump` is your go-to solution for creating and managing MongoDB database dumps. Whether you're backing up critical production data, migrating between environments, or implementing disaster recovery strategies, this tool has got your back. It provides a clean, TypeScript-based API for:

- □□ **Full database backups** - Export entire MongoDB databases
- □□ **Collection-level dumps** - Selectively backup specific collections
- □□ **Custom naming** - Transform document names during export
- □□ **Archive support** - Create compressed tar archives of your dumps
- □□ **Async/await patterns** - Modern promise-based workflow
- □□ **TypeScript first** - Full type safety and IntelliSense support

Installation □□

```
# Using npm
npm install @push.rocks/mongodump --save

# Using pnpm (recommended)
pnpm add @push.rocks/mongodump

# Using yarn
yarn add @push.rocks/mongodump
```

Quick Start □□

Get up and running in seconds:

```
import { MongoDump } from '@push.rocks/mongodump';
import type { IMongoDescriptor } from '@tsclass/tsclass';

// Define your MongoDB connection
const mongoDescriptor: IMongoDescriptor = {
  mongoDbName: 'my-database',
  mongoDbUser: 'admin',
  mongoDbPass: 'secure-password',
  mongoDbUrl: 'mongodb+srv://cluster.mongodb.net',
};

// Create a dump of your entire database
const mongoDump = new MongoDump();
const target = await mongoDump.addMongoTargetByMongoDescriptor(mongoDescriptor);
await target.dumpAllCollectionsToDir('./backups/my-backup');

// Done! Your backup is ready □□
await mongoDump.stop();
```

Detailed Usage □□

Setting Up MongoDB Connection

The module uses MongoDB descriptors to establish connections. This approach provides flexibility and security:

```
import { IMongoDescriptor } from '@tsclass/tsclass';
import { MongoDump, MongoDumpTarget } from '@push.rocks/mongodump';

const mongoDescriptor: IMongoDescriptor = {
  mongoDbName: 'production_db',
  mongoDbUser: process.env.MONGO_USER,
```

```
  mongoDbPass: process.env.MONGO_PASS,
  mongoDbUrl: process.env.MONGO_URL,
};
```

Working with MongoDump

The `MongoDump` class is your main entry point for managing database dumps:

```
const mongoDump = new MongoDump();

// Add a MongoDB target
const dumpTarget = await mongoDump.addMongoTargetByMongoDescriptor(mongoDescriptor);

// The target is now ready for dump operations
```

Dumping Collections

Dump All Collections to Directory

Perfect for complete database backups:

```
// Basic dump - uses document _id as filename
await dumpTarget.dumpAllCollectionsToDir('./backups/full-backup');

// Custom naming - use any document field for filenames
await dumpTarget.dumpAllCollectionsToDir(
  './backups/named-backup',
  (doc) => `${doc.username}_${doc.timestamp}`,
  true // Clean directory before dumping
);
```

Dump Specific Collection

When you need granular control:

```
// Get available collections
const collections = await dumpTarget.getCollections();
console.log('Available collections:', collections);
```

```
// Dump a specific collection
const userCollection = collections.find(c => c.collectionName === 'users');
await dumpTarget.dumpCollectionToDir(
  userCollection,
  './backups/users',
  (doc) => doc.email.replace('@', '_at_') // Custom naming
);
```

Archive Support

Create compressed archives for efficient storage and transfer:

```
// Dump to tar archive file
await dumpTarget.dumpCollectionToTarArchiveFile(
  userCollection,
  './backups/users.tar.gz'
);

// Or work with streams for advanced scenarios
const archiveStream = await dumpTarget.dumpAllCollectionsToTarArchiveStream();
// Pipe to S3, network storage, etc.
```

Advanced Patterns

Scheduled Backups

Implement automated backup strategies:

```
import { CronJob } from 'cron';

const backupJob = new CronJob('0 0 * * *', async () => {
  const timestamp = new Date().toISOString().split('T')[0];
  const mongoDump = new MongoDump();
  const target = await mongoDump.addMongoTargetByMongoDescriptor(mongoDescriptor);

  await target.dumpAllCollectionsToDir(
    `./backups/daily/${timestamp}`,
```

```
    null,
    true
  );

  await mongoDump.stop();
  console.log(`✅ Daily backup completed: ${timestamp}`);
});

backupJob.start();
```

Multiple Database Targets

Handle multiple databases simultaneously:

```
const mongoDump = new MongoDump();

// Add multiple targets
const prodTarget = await mongoDump.addMongoTargetByMongoDescriptor(prodDescriptor);
const stagingTarget = await mongoDump.addMongoTargetByMongoDescriptor(stagingDescriptor);

// Dump both in parallel
await Promise.all([
  prodTarget.dumpAllCollectionsToDir('./backups/production'),
  stagingTarget.dumpAllCollectionsToDir('./backups/staging')
]);

await mongoDump.stop();
```

Error Handling

Implement robust error handling:

```
try {
  const mongoDump = new MongoDump();
  const target = await mongoDump.addMongoTargetByMongoDescriptor(mongoDescriptor);

  await target.dumpAllCollectionsToDir('./backups/safe-backup');
  console.log('✅ Backup successful');

} catch (error) {
```

```
  console.error('□ Backup failed:', error);
  // Implement notification/retry logic

} finally {
  await mongoDump.stop();
}
```

Clean Shutdown

Always ensure proper cleanup of database connections:

```
// Closes all open MongoDB connections
await mongoDump.stop();
```

API Reference □□

MongoDump

The main class for managing dump operations.

Methods:

- `addMongoTargetByMongoDescriptor(descriptor: IMongoDescriptor)`: Create a new dump target
- `stop()`: Close all database connections

MongoDumpTarget

Handles operations for a specific MongoDB database.

Methods:

- `getCollections()`: List all collections in the database
- `dumpCollectionToDir(collection, directory, nameTransform?)`: Dump single collection
- `dumpAllCollectionsToDir(directory, nameTransform?, cleanDir?)`: Dump all collections
- `dumpCollectionToTarArchiveFile(collection, filepath)`: Create tar archive
- `dumpAllCollectionsToTarArchiveStream()`: Get tar archive stream

Best Practices □□

1. **Environment Variables**: Store credentials in environment variables, never hardcode them
2. **Error Handling**: Always wrap dump operations in try-catch blocks
3. **Connection Cleanup**: Always call `stop()` when done
4. **Backup Verification**: Periodically verify your backups can be restored
5. **Storage Rotation**: Implement backup rotation to manage disk space
6. **Monitoring**: Add logging and monitoring for production backup jobs

Why Choose @push.rocks/mongodump? □□

- **Type Safety**: Full TypeScript support with comprehensive type definitions
- **Modern Async**: Built on promises and async/await patterns
- **Flexible Export**: Multiple export formats and naming options
- **Production Ready**: Battle-tested in production environments
- **Clean API**: Intuitive, well-documented interface
- **Active Maintenance**: Regular updates and security patches

License and Legal Information

Trademarks

Company Information

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

For any legal inquiries or if you require 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.

changelog.md for @push.rocks/mongodump

2025-08-18 - 1.1.0 - feat(MongoDumpTarget)

Implement core MongoDumpTarget methods and update documentation & project configs

- Implemented MongoDumpTarget.createAndInit(), init(), getCollections(), dumpCollectionToDir() and dumpAllCollectionsToDir() with connection handling, collection enumeration and file output.
- Added safe directory handling (ensureDir / ensureEmptyDir) and document-to-file writing using smartfile.memory.toFs and smartpath transformation.
- Enhanced README with detailed usage examples, API reference and best practices.
- Added local project configuration files (.claude/settings.local.json and .serena/project.yml) to support tooling and project metadata.

2025-08-18 - 1.0.10 - fix(mongodump.plugins)

Bump @types/node to ^22.0.0 and use runtime import for @tsclass/tsclass in plugins

- Bumped dev dependency @types/node from ^17.0.40 to ^22.0.0 in package.json
- Changed import of @tsclass/tsclass from a type-only import to a runtime import in ts/mongodump.plugins.ts so tsclass is exported and available at runtime

2025-08-18 - 1.0.9 - fix(dependencies)

Update dependencies, normalize package scopes to @push.rocks, bump mongodb to v6, adjust tests and add pnpm metadata

- Bump mongodb dependency from ^4.6.0 to ^6.18.0 (major upgrade — potential breaking changes)
- Normalize package scopes from @pushrocks/* to @push.rocks/* across code and dependencies
- Replace @gitzone dev tooling with @git.zone equivalents and add @git.zone/tsrun
- Change test script to use tstest with --verbose and update test imports to @git.zone/tstest; adjust sample data import paths
- Add packageManager field for pnpm@10.14.0 and add pnpm-workspace.yaml (onlyBuiltDependencies)
- Remove .gitlab-ci.yml (CI configuration removed)
- Update ts/00_commitinfo_data and mongodump.plugins to use @push.rocks naming
- Minor test harness change: export default tap.start() instead of tap.start()

2024-05-29 - 1.0.8 - release

Finalized 1.0.8 with packaging, configuration, and organization updates.

- Update package description.
- Update TypeScript configuration (tsconfig).
- Update npmextra.json githost entries to correct package hosting information.
- Switch to new organization scheme for the project.

2022-06-06 - 1.0.1 → 1.0.7 - maintenance

Series of maintenance releases and core fixes across multiple minor version bumps.

- Multiple "fix(core): update" changes applied across versions 1.0.1 through 1.0.7.
- Several version-only commits and routine package bumps. No notable user-facing API changes.