

@push.rocks/smartbucket

A TypeScript library offering simple and cloud-agnostic object storage with advanced features like bucket creation, file and directory management, and data streaming.

- [readme.md for @push.rocks/smartbucket](#)
- [changelog.md for @push.rocks/smartbucket](#)

readme.md for @push.rocks/smartbucket

A powerful, cloud-agnostic TypeScript library for object storage that makes S3-compatible storage feel like a modern filesystem. Built for developers who demand simplicity, type-safety, and advanced features like real-time bucket watching, metadata management, file locking, intelligent trash handling, and memory-efficient streaming.

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.

Why SmartBucket?

- **Cloud Agnostic** - Write once, run on AWS S3, MinIO, DigitalOcean Spaces, Backblaze B2, Wasabi, Cloudflare R2, or any S3-compatible storage
- **Modern TypeScript** - First-class TypeScript support with complete type definitions and async/await patterns
- **Real-Time Watching** - Monitor bucket changes with polling-based watcher supporting RxJS and EventEmitter patterns
- **Memory Efficient** - Handle millions of files with async generators, RxJS observables, and cursor pagination
- **Smart Trash System** - Recover accidentally deleted files with built-in trash and restore functionality
- **File Locking** - Prevent concurrent modifications with built-in locking mechanisms
- **Rich Metadata** - Attach custom metadata to any file for powerful organization and search
- **Streaming Support** - Efficient handling of large files with Node.js and Web streams
- **Directory-like API** - Intuitive filesystem-like operations on object storage
- **Fail-Fast** - Strict-by-default API catches errors immediately with precise stack traces

Quick Start

```
import { SmartBucket } from '@push.rocks/smartbucket';

// Connect to your storage
const storage = new SmartBucket({
  accessKey: 'your-access-key',
  accessSecret: 'your-secret-key',
  endpoint: 's3.amazonaws.com',
  port: 443,
  useSsl: true
});

// Get or create a bucket
const bucket = await storage.getBucketByName('my-app-data');

// Upload a file
await bucket.fastPut({
  path: 'users/profile.json',
  contents: JSON.stringify({ name: 'Alice', role: 'admin' })
});

// Download it back
const data = await bucket.fastGet({ path: 'users/profile.json' });
console.log(JSON.parse(data.toString()));

// List files efficiently (even with millions of objects!)
for await (const key of bucket.listAllObjects('users/')) {
  console.log('Found:', key);
}

// Watch for changes in real-time
const watcher = bucket.createWatcher({ prefix: 'uploads/', pollIntervalMs: 3000 });
watcher.changeSubject.subscribe((change) => {
  console.log('Change detected:', change.type, change.key);
});
await watcher.start();
```

Install

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

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

Usage

Table of Contents

1. [Getting Started](#)
2. [Working with Buckets](#)
3. [File Operations](#)
4. [Memory-Efficient Listing](#)
5. [Bucket Watching](#)
6. [Directory Management](#)
7. [Streaming Operations](#)
8. [File Locking](#)
9. [Metadata Management](#)
10. [Trash & Recovery](#)
11. [Advanced Features](#)
12. [Cloud Provider Support](#)

Getting Started

First, set up your storage connection:

```
import { SmartBucket } from '@push.rocks/smartbucket';

// Initialize with your cloud storage credentials
const smartBucket = new SmartBucket({
```

```
accessKey: 'your-access-key',
accessSecret: 'your-secret-key',
endpoint: 's3.amazonaws.com', // Or your provider's endpoint
port: 443,
useSsl: true,
region: 'us-east-1' // Optional, defaults to 'us-east-1'
});
```

For MinIO or self-hosted storage:

```
const smartBucket = new SmartBucket({
  accessKey: 'minioadmin',
  accessSecret: 'minioadmin',
  endpoint: 'localhost',
  port: 9000,
  useSsl: false // MinIO often runs without SSL locally
});
```

Working with Buckets

Creating Buckets

```
// Create a new bucket
const myBucket = await smartBucket.createBucket('my-awesome-bucket');
```

Getting Existing Buckets

```
// Get a bucket reference (throws if not found - strict by default!)
const existingBucket = await smartBucket.getBucketByName('existing-bucket');

// Check first, then get (non-throwing approach)
if (await smartBucket.bucketExists('maybe-exists')) {
  const bucket = await smartBucket.getBucketByName('maybe-exists');
}
```

Removing Buckets

```
// Delete a bucket (must be empty)
await smartBucket.removeBucket('old-bucket');
```

File Operations

Upload Files

```
const bucket = await smartBucket.getBucketByName('my-bucket');

// Simple file upload (returns File object)
const file = await bucket.fastPut({
  path: 'documents/report.pdf',
  contents: Buffer.from('Your file content here')
});

// Upload with string content
await bucket.fastPut({
  path: 'notes/todo.txt',
  contents: 'Buy milk\nCall mom\nRule the world'
});

// Upload with overwrite control
const uploadedFile = await bucket.fastPut({
  path: 'images/logo.png',
  contents: imageBuffer,
  overwrite: true // Set to true to replace existing files
});

// Error handling: fastPut throws if file exists and overwrite is false
try {
  await bucket.fastPut({
    path: 'existing-file.txt',
    contents: 'new content'
  });
} catch (error) {
  console.error('Upload failed:', error.message);
}
```

Download Files

```
// Get file as Buffer
const fileContent = await bucket.fastGet({
```

```
    path: 'documents/report.pdf'
  });

// Get file as string
const textContent = fileContent.toString('utf-8');

// Parse JSON files directly
const jsonData = JSON.parse(fileContent.toString());
```

Check File Existence

```
const exists = await bucket.fastExists({
  path: 'documents/report.pdf'
});
```

Delete Files

```
// Permanent deletion
await bucket.fastRemove({
  path: 'old-file.txt'
});
```

Copy & Move Files

```
// Copy file within bucket
await bucket.fastCopy({
  sourcePath: 'original/file.txt',
  destinationPath: 'backup/file-copy.txt'
});

// Move file (copy + delete original)
await bucket.fastMove({
  sourcePath: 'temp/draft.txt',
  destinationPath: 'final/document.txt'
});

// Copy to different bucket
const targetBucket = await smartBucket.getBucketByName('backup-bucket');
await bucket.fastCopy({
```

```
sourcePath: 'important/data.json',
destinationPath: 'archived/data.json',
targetBucket: targetBucket
});
```

Memory-Efficient Listing

SmartBucket provides three powerful patterns for listing objects, optimized for handling **millions of files** efficiently:

Async Generators (Recommended)

Memory-efficient streaming using native JavaScript async iteration:

```
// List all objects with prefix - streams one at a time!
for await (const key of bucket.listAllObjects('documents/')) {
  console.log('Found:', key);

  // Process each file individually (memory efficient!)
  const content = await bucket.fastGet({ path: key });
  processFile(content);

  // Early termination support
  if (shouldStop()) break;
}

// Find objects matching glob patterns
for await (const key of bucket.findByGlob('**/*.json')) {
  console.log('JSON file:', key);
}

// Complex glob patterns
for await (const key of bucket.findByGlob('npm/packages/*/index.json')) {
  console.log('Package index:', key);
}

for await (const key of bucket.findByGlob('images/*.{jpg,png,gif}')) {
  console.log('Image:', key);
}
```

Why use async generators?

- Processes one item at a time (constant memory usage)
- Supports early termination with `break`
- Native JavaScript - no dependencies
- Perfect for large buckets with millions of objects

RxJS Observables

Perfect for reactive pipelines and complex data transformations:

```
import { filter, take, map } from 'rxjs/operators';

// Stream keys as Observable with powerful operators
bucket.listAllObjectsObservable('logs/')
  .pipe(
    filter(key => key.endsWith('.log')),
    take(100),
    map(key => ({ key, timestamp: Date.now() }))
  )
  .subscribe({
    next: (item) => console.log('Log file:', item.key),
    error: (err) => console.error('Error:', err),
    complete: () => console.log('Listing complete')
  });

// Combine with other observables
import { merge } from 'rxjs';

const logs$ = bucket.listAllObjectsObservable('logs/');
const backups$ = bucket.listAllObjectsObservable('backups/');

merge(logs$, backups$)
  .pipe(filter(key => key.includes('2024')))
  .subscribe(key => console.log('2024 file:', key));
```

Cursor Pattern

Explicit pagination control for UI and resumable operations:

```

// Create cursor with custom page size
const cursor = bucket.createCursor('uploads/', { pageSize: 100 });

// Fetch pages manually
while (cursor.hasMore()) {
  const page = await cursor.next();
  console.log(`Page has ${page.keys.length} items`);

  for (const key of page.keys) {
    console.log(` - ${key}`);
  }

  if (page.done) break;
}

// Save and restore cursor state (perfect for resumable operations!)
const token = cursor.getToken();
// Store token in database or session...

// ... later, in a different request ...
const newCursor = bucket.createCursor('uploads/', { pageSize: 100 });
newCursor.setToken(token); // Resume from saved position!
const nextPage = await newCursor.next();

// Reset cursor to start over
cursor.reset();

```

Convenience Methods

```

// Collect all keys into array (WARNING: loads everything into memory!)
const allKeys = await bucket.listAllObjectsArray('images/');
console.log(`Found ${allKeys.length} images`);

```

Performance Comparison:

Method	Memory Usage	Best For	Supports Early Exit
Async Generator	O(1) - constant	Most use cases, large datasets	Yes
Observable	O(1) - constant	Reactive pipelines, RxJS apps	Yes

Method	Memory Usage	Best For	Supports Early Exit
Cursor	O(pageSize)	UI pagination, resumable ops	Yes
Array	O(n) - grows with results	Small datasets (<10k items)	No

Bucket Watching

Monitor your storage bucket for changes in real-time with the powerful `BucketWatcher`:

```
// Create a watcher for a specific prefix
const watcher = bucket.createWatcher({
  prefix: 'uploads/',          // Watch files with this prefix
  pollIntervalMs: 3000,       // Check every 3 seconds
  includeInitial: false,      // Don't emit existing files on start
});

// RxJS Observable pattern (recommended for reactive apps)
watcher.changeSubject.subscribe((change) => {
  if (change.type === 'add') {
    console.log('New file:', change.key);
  } else if (change.type === 'modify') {
    console.log('Modified:', change.key);
  } else if (change.type === 'delete') {
    console.log('Deleted:', change.key);
  }
});

// EventEmitter pattern (classic Node.js style)
watcher.on('change', (change) => {
  console.log(`${change.type}: ${change.key}`);
});

watcher.on('error', (err) => {
  console.error('Watcher error:', err);
});

// Start watching
await watcher.start();
```

```

// Wait until watcher is ready (initial state built)
await watcher.readyDeferred.promise;
console.log('Watcher is now monitoring the bucket');

// ... your application runs ...

// Stop watching when done
await watcher.stop();
// Or use the alias:
await watcher.close();

```

Watcher Options

```

interface IBucketWatcherOptions {
  prefix?: string;           // Filter objects by prefix (default: '' = all)
  pollIntervalMs?: number;  // Polling interval in ms (default: 5000)
  bufferTimeMs?: number;    // Buffer events before emitting (for batching)
  includeInitial?: boolean; // Emit existing files as 'add' on start (default: false)
  pageSize?: number;        // Objects per page when listing (default: 1000)
}

```

Buffered Events

For high-frequency change environments, buffer events to reduce processing overhead:

```

const watcher = bucket.createWatcher({
  prefix: 'high-traffic/',
  pollIntervalMs: 1000,
  bufferTimeMs: 2000, // Collect events for 2 seconds before emitting
});

// Receive batched events as arrays
watcher.changeSubject.subscribe((changes) => {
  if (Array.isArray(changes)) {
    console.log(`Batch of ${changes.length} changes:`);
    changes.forEach(c => console.log(` - ${c.type}: ${c.key}`));
  }
});

```

```
await watcher.start();
```

Change Event Structure

```
interface IStorageChangeEvent {  
  type: 'add' | 'modify' | 'delete';  
  key: string;           // Object key (path)  
  bucket: string;       // Bucket name  
  size?: number;        // File size (not present for deletes)  
  etag?: string;        // ETag hash (not present for deletes)  
  lastModified?: Date;  // Last modified date (not present for deletes)  
}
```

Watch Use Cases

- **Sync systems** - Detect changes to trigger synchronization
- **Analytics** - Track file uploads/modifications in real-time
- **Notifications** - Alert users when their files are ready
- **Processing pipelines** - Trigger workflows on new file uploads
- **Backup systems** - Detect changes for incremental backups
- **Audit logs** - Track all bucket activity

Directory Management

SmartBucket provides powerful directory-like operations for organizing your files:

```
// Get base directory  
const baseDir = await bucket.getBaseDirectory();  
  
// List directories and files  
const directories = await baseDir.listDirectories();  
const files = await baseDir.listFiles();  
  
console.log(`Found ${directories.length} directories`);  
console.log(`Found ${files.length} files`);  
  
// Navigate subdirectories  
const subDir = await baseDir.getSubDirectoryByName('projects/2024');  
  
// Create nested file
```

```
await subDir.fastPut({
  path: 'report.pdf',
  contents: reportBuffer
});

// Get directory tree structure
const tree = await subDir.getTreeArray();

// Get directory path
console.log('Base path:', subDir.getBasePath()); // "projects/2024/"

// Create empty file as placeholder
await subDir.createEmptyFile('placeholder.txt');

// Check existence
const fileExists = await subDir.fileExists({ path: 'report.pdf' });
const dirExists = await baseDir.directoryExists('projects');
```

Streaming Operations

Handle large files efficiently with streaming support:

Download Streams

```
// Node.js stream (for file I/O, HTTP responses, etc.)
const nodeStream = await bucket.fastGetStream(
  { path: 'large-video.mp4' },
  'nodestream'
);

// Pipe to file
import * as fs from 'node:fs';
nodeStream.pipe(fs.createWriteStream('local-video.mp4'));

// Pipe to HTTP response
app.get('/download', async (req, res) => {
  const stream = await bucket.fastGetStream(
    { path: 'file.pdf' },
    'nodestream'
  );
```

```
);
res.setHeader('Content-Type', 'application/pdf');
stream.pipe(res);
});

// Web stream (for modern browser/Deno environments)
const webStream = await bucket.fastGetStream(
  { path: 'large-file.zip' },
  'webstream'
);
```

Upload Streams

```
import * as fs from 'node:fs';

// Stream upload from file
const readStream = fs.createReadStream('big-data.csv');
await bucket.fastPutStream({
  path: 'uploads/big-data.csv',
  readableStream: readStream,
  nativeMetadata: {
    'content-type': 'text/csv',
    'x-custom-header': 'my-value'
  }
});
```

Reactive Streams with RxJS

```
// Get file as ReplaySubject for reactive programming
const replaySubject = await bucket.fastGetReplaySubject({
  path: 'data/sensor-readings.json'
});

// Multiple subscribers can consume the same data
replaySubject.subscribe({
  next: (chunk) => processChunk(chunk),
  complete: () => console.log('Stream complete')
});
```

```
replaySubject.subscribe({
  next: (chunk) => logChunk(chunk)
});
```

File Locking

Prevent concurrent modifications with built-in file locking:

```
const baseDir = await bucket.getBaseDirectory();
const file = await baseDir.getFile({ path: 'important-config.json' });

// Lock file for 10 minutes
await file.lock({ timeoutMillis: 600000 });

// Check lock status via metadata
const metadata = await file.getMetaData();
const isLocked = await metadata.checkLocked();

// Get lock info
const lockInfo = await metadata.getLockInfo();
console.log(`Lock expires: ${new Date(lockInfo.expires)}`);

// Unlock when done
await file.unlock();

// Force unlock (even if locked by another process)
await file.unlock({ force: true });
```

Lock use cases:

- Prevent concurrent writes during critical updates
- Protect configuration files during deployment
- Coordinate distributed workers
- Ensure data consistency

Metadata Management

Attach and manage rich metadata for your files:

```
const baseDir = await bucket.getBaseDirectory();
const file = await baseDir.getFile({ path: 'document.pdf' });

// Get metadata handler
const metadata = await file.getMetaData();

// Store custom metadata (can be any JSON-serializable value)
await metadata.storeCustomMetaData({
  key: 'author',
  value: 'John Doe'
});

await metadata.storeCustomMetaData({
  key: 'tags',
  value: ['important', 'quarterly-report', '2024']
});

await metadata.storeCustomMetaData({
  key: 'workflow',
  value: { status: 'approved', approvedBy: 'jane@company.com' }
});

// Retrieve metadata
const author = await metadata.getCustomMetaData({ key: 'author' });

// Delete metadata
await metadata.deleteCustomMetaData({ key: 'workflow' });

// Check if file has any metadata
const hasMetadata = await file.hasMetaData();

// Get file type detection
const fileType = await metadata.getFileType({ useFileExtension: true });

// Get file type from magic bytes (more accurate)
const detectedType = await metadata.getFileType({ useMagicBytes: true });

// Get file size
const size = await metadata.getSizeInBytes();
```

Metadata use cases:

- Track file ownership and authorship
- Add tags and categories for search
- Store processing status or workflow state
- Enable rich querying and filtering
- Maintain audit trails

Trash & Recovery

SmartBucket includes an intelligent trash system for safe file deletion and recovery:

```
const baseDir = await bucket.getBaseDirectory();
const file = await baseDir.getFile({ path: 'important-data.xlsx' });

// Move to trash instead of permanent deletion
await file.delete({ mode: 'trash' });

// Permanent deletion (use with caution!)
await file.delete({ mode: 'permanent' });

// Access trash
const trash = await bucket.getTrash();
const trashDir = await trash.getTrashDir();
const trashedFiles = await trashDir.listFiles();

// Restore from trash
const trashedFile = await baseDir.getFile({
  path: 'important-data.xlsx',
  getFromTrash: true
});

await trashedFile.restore({ useOriginalPath: true });

// Or restore to a different location
await trashedFile.restore({
  toPath: 'recovered/important-data.xlsx'
});
```

Trash features:

- Recover accidentally deleted files
- Preserves original path in metadata
- Tracks deletion timestamp
- List and inspect trashed files

Advanced Features

File Statistics

```
// Get detailed file statistics
const stats = await bucket.fastStat({ path: 'document.pdf' });
console.log(`Size: ${stats.ContentLength} bytes`);
console.log(`Last modified: ${stats.LastModified}`);
console.log(`ETag: ${stats.ETag}`);
console.log(`Content type: ${stats.ContentType}`);
```

Magic Bytes Detection

Detect file types by examining the first bytes (useful for validation):

```
// Read first bytes for file type detection
const magicBytes = await bucket.getMagicBytes({
  path: 'mystery-file',
  length: 16
});

// Or from a File object
const baseDir = await bucket.getBaseDirectory();
const file = await baseDir.getFile({ path: 'image.jpg' });
const magic = await file.getMagicBytes({ length: 4 });

// Check file signatures
if (magic[0] === 0xFF && magic[1] === 0xD8) {
  console.log('This is a JPEG image');
} else if (magic[0] === 0x89 && magic[1] === 0x50) {
  console.log('This is a PNG image');
}
```

JSON Data Operations

```
const baseDir = await bucket.getBaseDirectory();
const file = await baseDir.getFile({ path: 'config.json' });

// Read JSON data
const config = await file.getJsonData();

// Update JSON data
config.version = '2.0';
config.updated = new Date().toISOString();
config.features.push('newFeature');

await file.writeJsonData(config);
```

Directory & File Type Detection

```
// Check if path is a directory
const isDir = await bucket.isDirectory({ path: 'uploads/' });

// Check if path is a file
const isFile = await bucket.isFile({ path: 'uploads/document.pdf' });
```

Clean Bucket Contents

```
// Remove all files and directories (use with caution!)
await bucket.cleanAllContents();
```

Cloud Provider Support

SmartBucket works seamlessly with all major S3-compatible providers:

Provider	Status	Notes
AWS S3	Supported	Native S3 API
MinIO	Supported	Self-hosted, perfect for development
DigitalOcean Spaces	Supported	Cost-effective S3-compatible
Backblaze B2	Supported	Very affordable storage
Wasabi	Supported	High-performance hot storage
Google Cloud Storage	Supported	Via S3-compatible API

Provider	Status	Notes
Cloudflare R2	Supported	Zero egress fees
Any S3-compatible	Supported	Works with any S3-compatible provider

Configuration examples:

```
// AWS S3
const awsStorage = new SmartBucket({
  accessKey: process.env.AWS_ACCESS_KEY_ID,
  accessSecret: process.env.AWS_SECRET_ACCESS_KEY,
  endpoint: 's3.amazonaws.com',
  region: 'us-east-1',
  useSsl: true
});

// MinIO (local development)
const minioStorage = new SmartBucket({
  accessKey: 'minioadmin',
  accessSecret: 'minioadmin',
  endpoint: 'localhost',
  port: 9000,
  useSsl: false
});

// DigitalOcean Spaces
const doStorage = new SmartBucket({
  accessKey: process.env.DO_SPACES_KEY,
  accessSecret: process.env.DO_SPACES_SECRET,
  endpoint: 'nyc3.digitaloceanspaces.com',
  region: 'nyc3',
  useSsl: true
});

// Backblaze B2
const b2Storage = new SmartBucket({
  accessKey: process.env.B2_KEY_ID,
  accessSecret: process.env.B2_APPLICATION_KEY,
  endpoint: 's3.us-west-002.backblazeb2.com',
  region: 'us-west-002',
```

```
    useSsl: true
  });

// Cloudflare R2
const r2Storage = new SmartBucket({
  accessKey: process.env.R2_ACCESS_KEY_ID,
  accessSecret: process.env.R2_SECRET_ACCESS_KEY,
  endpoint: `${process.env.R2_ACCOUNT_ID}.r2.cloudflarestorage.com`,
  region: 'auto',
  useSsl: true
});
```

Advanced Configuration

```
// Environment-based configuration with @push.rocks/qenv
import { Qenv } from '@push.rocks/qenv';

const qenv = new Qenv('./', './.nogit/');

const smartBucket = new SmartBucket({
  accessKey: await qenv.getEnvVarOnDemand('S3_ACCESS_KEY'),
  accessSecret: await qenv.getEnvVarOnDemand('S3_SECRET'),
  endpoint: await qenv.getEnvVarOnDemand('S3_ENDPOINT'),
  port: parseInt(await qenv.getEnvVarOnDemand('S3_PORT')),
  useSsl: await qenv.getEnvVarOnDemand('S3_USE_SSL') === 'true',
  region: await qenv.getEnvVarOnDemand('S3_REGION')
});
```

Testing

```
# Run all tests
pnpm test

# Run specific test file
pnpm tstest test/test.watcher.node.ts --verbose

# Run tests with log file
```

Error Handling Best Practices

SmartBucket uses a **strict-by-default** approach - methods throw errors instead of returning null:

```
// Check existence first
if (await bucket.fastExists({ path: 'file.txt' })) {
  const content = await bucket.fastGet({ path: 'file.txt' });
  process(content);
}

// Try/catch for expected failures
try {
  const file = await bucket.fastGet({ path: 'might-not-exist.txt' });
  process(file);
} catch (error) {
  console.log('File not found, using default');
  useDefault();
}

// Explicit overwrite control
try {
  await bucket.fastPut({
    path: 'existing-file.txt',
    contents: 'new data',
    overwrite: false // Explicitly fail if exists
  });
} catch (error) {
  console.log('File already exists');
}
```

Best Practices

1. **Always use strict mode** for critical operations to catch errors early
2. **Check existence first** with `fastExists()`, `bucketExists()`, etc. before operations
3. **Implement proper error handling** for network and permission issues
4. **Use streaming** for large files (>100MB) to optimize memory usage
5. **Leverage metadata** for organizing and searching files

6. **Enable trash mode** for important data to prevent accidental loss
7. **Lock files** during critical operations to prevent race conditions
8. **Use async generators** for listing large buckets to avoid memory issues
9. **Set explicit overwrite flags** to prevent accidental file overwrites
10. **Use the watcher** for real-time synchronization and event-driven architectures

Performance Tips

- **Listing:** Use async generators or cursors for buckets with >10,000 objects
- **Uploads:** Use streams for files >100MB
- **Downloads:** Use streams for files you'll process incrementally
- **Metadata:** Cache metadata when reading frequently
- **Locking:** Keep lock durations as short as possible
- **Glob patterns:** Be specific to reduce objects scanned
- **Watching:** Use appropriate `pollIntervalMs` based on your change frequency

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.

changelog.md for @push.rocks/smartbucket

2026-03-14 - 4.5.1 - fix(readme)

refresh documentation wording and simplify README formatting

- removes emoji-heavy headings and inline log examples for a cleaner presentation
- clarifies S3-compatible storage wording in the project description
- streamlines testing and error handling sections without changing library functionality

2026-03-14 - 4.5.0 - feat(storage)

generalize S3 client and watcher interfaces to storage-oriented naming with backward compatibility

- rename SmartBucket internals from s3Client to storageClient and normalize configuration using storage descriptor types
- update watcher and interface types from S3-specific names to storage-oriented equivalents while keeping deprecated aliases
- refresh tests and documentation comments to reflect S3-compatible object storage terminology
- bump build, test, AWS SDK, and related dependency versions

2026-01-25 - 4.4.1 - fix(tests)

add explicit 'as string' type assertions to environment variable retrievals in tests

- Add 'as string' assertions for S3_ACCESSKEY, S3_SECRETKEY, S3_ENDPOINT, and S3_BUCKET to satisfy TypeScript typings in tests
- Cast S3_PORT to string before parseInt to ensure correct input type and avoid compiler errors
- Changes are localized to test/test.listing.node+deno.ts

2026-01-25 - 4.4.0 - feat(watcher)

add polling-based BucketWatcher to detect add/modify/delete events and expose RxJS Observable and EventEmitter APIs

- Add new BucketWatcher implementation (ts/classes.watcher.ts) with polling, buffering and change detection by ETag/Size/LastModified
- Expose createWatcher(...) on Bucket (ts/classes.bucket.ts) and export watcher and interfaces from index (ts/index.ts)
- Add watcher-related interfaces (IS3ChangeEvent, IS3ObjectState, IBucketWatcherOptions) to ts/interfaces.ts
- Comprehensive README/docs additions and examples for BucketWatcher, buffering, options and use cases (readme.md, readme.hints.md)
- Add unit/integration tests for watcher behavior (test/test.watcher.node.ts)
- Non-breaking API additions only — ready for a minor version bump from 4.3.1 to 4.4.0

2026-01-24 - 4.3.1 - fix(bucket)

propagate S3 client errors instead of silently logging them; update build script, bump dev/dependencies, and refresh npmextra configuration

- Remove .catch(...) wrappers around s3Client.send so errors are no longer swallowed and will propagate to callers
- Update build script to use 'tsbuild tsfolders --allowimplicitany'
- Bump devDependencies: @git.zone/tsbuild to ^4.1.2, @git.zone/tsrun to ^2.0.1, @git.zone/tstest to ^3.1.6
- Bump dependency @aws-sdk/client-s3 to ^3.975.0
- Adjust npmextra.json structure (@git.zone/cli, @git.zone/tsdoc), add release registries and @ship.zone/szci entry
- Remove pnpm-workspace.yaml onlyBuiltDependencies configuration

2025-11-20 - 4.3.0 - feat(listing)

Add memory-efficient listing APIs: async generator, RxJS observable, and cursor pagination; export ListCursor and Minimatch; add minimatch dependency; bump to 4.2.0

- Added memory-efficient listing methods on Bucket: listAllObjects (async generator), listAllObjectsObservable (RxJS Observable), createCursor (returns ListCursor) and listAllObjectsArray (convenience array collector).

- New ListCursor class (ts/classes.listcursor.ts) providing page-based iteration: next(), hasMore(), reset(), getToken()/setToken().
- Added glob matching helper findByGlob(pattern) using minimatch (exported via plugins.Minimatch).
- Exported ListCursor from ts/index.ts and exported Minimatch via ts/plugins.ts.
- Added minimatch dependency in package.json and bumped package version to 4.2.0; increased test timeout to 120s.
- Updated tests to read S3_SECRETKEY, S3_PORT and to assert bucket name from env (test/test.node+deno.ts, test/test.trash.node+deno.ts).
- No breaking changes: new APIs are additive and existing behavior preserved.

2025-11-20 - 4.2.0 - feat(listing)

Add memory-efficient listing with async generators, RxJS observables, and cursor pagination for huge buckets

New Memory-Efficient Listing Methods:

Async Generator (Recommended for most use cases):

- `Bucket.listAllObjects(prefix?)` - Stream object keys one at a time using `for await...of`
- `Bucket.findByGlob(pattern)` - Find objects matching glob patterns (e.g., `**/*.json`, `npm/packages/*/index.json`)
- Memory efficient, supports early termination, composable

RxJS Observable (For complex reactive pipelines):

- `Bucket.listAllObjectsObservable(prefix?)` - Emit keys as Observable for use with RxJS operators (filter, map, take, etc.)
- Perfect for complex data transformations and reactive architectures

Cursor Pattern (For manual pagination control):

- `Bucket.createCursor(prefix?, options?)` - Create cursor for explicit page-by-page iteration
- `ListCursor.next()` - Fetch next page of results
- `ListCursor.hasMore()` - Check if more results available
- `ListCursor.reset()` - Reset to beginning
- `ListCursor.getToken()` / `ListCursor.setToken()` - Save/restore pagination state
- Ideal for UI pagination and resumable operations

Convenience Methods:

- `Bucket.listAllObjectsArray(prefix?)` - Collect all keys into array (WARNING: loads all into memory)

Benefits:

- Memory-efficient streaming for buckets with millions of objects
- Three patterns for different use cases (generators, observables, cursors)
- Support for early termination and incremental processing
- Glob pattern matching with minimatch
- Full TypeScript support with proper types
- Zero breaking changes - all new methods

Dependencies:

- Added `minimatch` for glob pattern support

Files Changed:

- `ts/classes.bucket.ts` - Added all listing methods
- `ts/classes.listcursor.ts` - NEW: Cursor implementation
- `ts/plugins.ts` - Export Minimatch
- `ts/index.ts` - Export ListCursor
- `test/test.listing.node+deno.ts` - NEW: Comprehensive listing tests
- `package.json` - Added minimatch dependency

2025-11-20 - 4.1.0 - feat(core)

Add S3 endpoint normalization, directory pagination, improved metadata checks, trash support, and related tests

- Add `normalizeS3Descriptor` helper to sanitize and normalize various S3 endpoint formats and emit warnings for mismatches (`helpers.ts`).
- Use normalized endpoint and credentials when constructing S3 client in `SmartBucket` (`classes.smartbucket.ts`).
- Implement paginated listing helper `listObjectsV2AllPages` in `Directory` and use it for `listFiles` and `listDirectories` to aggregate `Contents` and `CommonPrefixes` across pages (`classes.directory.ts`).
- Improve `MetaData.hasMetaData` to catch `NotFound` errors and return `false` instead of throwing (`classes.metadata.ts`).
- Export metadata and trash modules from index (`ts/index.ts`) and add a `Trash` class with utilities for trashed files and key encoding (`classes.trash.ts`).
- Enhance Bucket operations: `fastCopy` now preserves or replaces native metadata correctly, `cleanAllContents` supports paginated deletion, and improved `fastExists` error handling (`classes.bucket.ts`).
- Fix `Directory.getSubDirectoryByName` to construct new `Directory` instances with the correct parent directory reference.

- Add tests covering metadata absence and pagination behavior (test/test.local.node+deno.ts).

2025-11-20 - 4.0.1 - fix(plugings)

Use explicit node: imports for native path and stream modules in ts/plugings.ts

- Replaced imports of 'path' and 'stream' with 'node:path' and 'node:stream' in ts/plugings.ts.
- Ensures correct ESM resolution of Node built-ins when package.json type is 'module' and avoids accidental conflicts with userland packages.

2025-11-20 - 4.0.0 - BREAKING CHANGE(core)

Make API strict-by-default: remove *Strict variants, throw on not-found/exists conflicts, add explicit exists() methods, update docs/tests and bump deps

- Breaking: Core API methods are strict by default and now throw errors instead of returning null when targets are missing or already exist (e.g. getBucketByName, getFile, getSubDirectoryByName, fastPut, fastPutStream).
- Removed *Strict variants: fastPutStrict, getBucketByNameStrict, getFileStrict, getSubDirectoryByNameStrict — use the base methods which are now strict.
- Added explicit existence checks: bucketExists (SmartBucket), fileExists (Directory/fileExists), directoryExists (Directory.directoryExists), and fastExists (Bucket.fastExists) to allow non-throwing checks before operations.
- Return type updates: fastPut now returns Promise<File> (no null), getBucketByName/getFile/getSubDirectoryByName now return the respective objects or throw.
- Improved error messages to guide callers (e.g. suggest setting overwrite:true on fastPut when object exists).
- Updated README, changelog and tests to reflect the new strict semantics and usage patterns.
- Developer/runtime dependency bumps: @git.zone/tsbuild, @git.zone/tsrun, @git.zone/tstest, @aws-sdk/client-s3, @push.rocks/smartstring, @tsclass/tsclass (version bumps recorded in package.json).
- Major version bump to 4.0.0 to reflect breaking API changes.

2025-11-20 - 4.0.0 - BREAKING:

Strict by default + exists methods

Complete API overhaul: all methods throw by default, removed all *Strict variants, added dedicated exists methods

Breaking Changes:

Putters (Write Operations):

- `fastPut`: Return type `Promise<File | null>` → `Promise<File>`, throws when file exists and `overwrite` is false
- `fastPutStream`: Now throws when file exists and `overwrite` is false (previously returned silently)
- `fastPutStrict`: **Removed** - use `fastPut` directly

Getters (Read Operations):

- `getBucketByName`: Return type `Promise<Bucket | null>` → `Promise<Bucket>`, throws when bucket not found
- `getBucketByNameStrict`: **Removed** - use `getBucketByName` directly
- `getFile`: Return type `Promise<File | null>` → `Promise<File>`, throws when file not found
- `getFileStrict`: **Removed** - use `getFile` directly
- `getSubDirectoryByName`: Return type `Promise<Directory | null>` → `Promise<Directory>`, throws when directory not found
- `getSubDirectoryByNameStrict`: **Removed** - use `getSubDirectoryByName` directly

New Methods (Existence Checks):

- `bucket.fastExists({ path })` - Already existed
- `directory.fileExists({ path })` - **NEW** - Check if file exists
- `directory.directoryExists(name)` - **NEW** - Check if subdirectory exists
- `smartBucket.bucketExists(name)` - **NEW** - Check if bucket exists

Benefits:

- **Simpler API**: Removed 4 redundant *Strict methods
- **Type-safe**: No nullable returns - `Promise<T>` not `Promise<T | null>`
- **Fail-fast**: Errors throw immediately with precise stack traces
- **Consistent**: All methods behave the same way
- **Explicit**: Use `exists()` to check, then `get()` to retrieve
- **Better debugging**: Error location is always precise

Migration Guide:

```
// =====  
// Pattern 1: Check then Get (Recommended)  
// =====  
  
// Before (v3.x):  
const bucket = await smartBucket.getBucketByName('my-bucket');  
if (bucket) {  
  // use bucket  
}  
  
// After (v4.0):  
if (await smartBucket.bucketExists('my-bucket')) {  
  const bucket = await smartBucket.getBucketByName('my-bucket'); // guaranteed to exist  
  // use bucket  
}  
  
// =====  
// Pattern 2: Try/Catch  
// =====  
  
// Before (v3.x):  
const file = await directory.getFile({ path: 'file.txt' });  
if (!file) {  
  // Handle not found  
}  
  
// After (v4.0):  
try {  
  const file = await directory.getFile({ path: 'file.txt' });  
  // use file  
} catch (error) {  
  // Handle not found  
}  
  
// =====  
// Pattern 3: Remove *Strict calls  
// =====
```

```
// Before (v3.x):
const file = await directory.getFileStrict({ path: 'file.txt' });

// After (v4.0):
const file = await directory.getFile({ path: 'file.txt' }); // already strict

// =====
// Pattern 4: Write Operations
// =====

// Before (v3.x):
const file = await bucket.fastPutStrict({ path: 'file.txt', contents: 'data' });

// After (v4.0):
const file = await bucket.fastPut({ path: 'file.txt', contents: 'data' }); // already strict
```

2025-08-18 - 3.3.10 - fix(helper)

Normalize and robustly parse S3 endpoint configuration; use normalized descriptor in SmartBucket and update dev tooling

- Add normalizeS3Descriptor to ts/helpers.ts: robust endpoint parsing, coercion of useSsl/port, sanitization, warnings for dropped URL parts, and canonical endpoint URL output.
- Update SmartBucket (ts/classes.smartbucket.ts) to use the normalized endpoint, region, credentials and forcePathStyle from normalizeS3Descriptor.
- Adjust dev tooling: bump @git.zone/tsbuild -> ^2.6.7, @git.zone/tstest -> ^2.3.4, @push.rocks/qenv -> ^6.1.3 and update test script to run tstest with --verbose --logfile --timeout 60.
- Add .claude/settings.local.json containing local assistant/CI permission settings (local config only).

2025-08-15 - 3.3.9 - fix(docs)

Revise README with detailed usage examples and add local Claude settings

- Revamped README: reorganized content, added emojis and clearer headings for install, getting started, bucket/file/directory operations, streaming, metadata, trash/recovery, locking, and advanced configuration.
- Added many concrete code examples for SmartBucket, Bucket, Directory, File, streaming (node/web), RxJS replay subjects, metadata handling, trash workflow, file locking, magic-bytes detection, JSON operations, and cleaning bucket contents.
- Included testing instructions (pnpm test) and a Best Practices section with recommendations for strict mode, streaming, metadata, trash usage, and locking.
- Added `.claude/settings.local.json` to include local Claude configuration and tool permissions.
- No source code or public API changes; documentation and local tooling config only.

2025-08-15 - 3.3.8 - fix(tests)

Update tests to use `@git.zone/tstest`, upgrade dependencies, remove GitLab CI and add local CI/workspace config

- Tests: replace imports from `@push.rocks/tapbundle` with `@git.zone/tstest/tapbundle` and switch `tap.start()` to export default `tap.start()`
- Dependencies: bump `@aws-sdk/client-s3` and several `@push.rocks` packages; upgrade `@tsclass/tsclass` to a newer major
- DevDependencies: upgrade `@git.zone/tsbuild`, `@git.zone/tstest`, `@push.rocks/qenv`, and `@push.rocks/tapbundle`
- CI/config: remove `.gitlab-ci.yml`, add `.claude/settings.local.json`
- Workspace: add `pnpm-workspace.yml` and `packageManager` field in `package.json`

2024-12-02 - 3.3.7 - fix(package)

Update author field in `package.json`

- Corrected the author field from 'Lossless GmbH' to 'Task Venture Capital GmbH' in the `package.json` file.

2024-12-02 - 3.3.6 - fix(package)

Fix license field in `package.json` to reflect MIT licensing

2024-11-25 - 3.3.5 - fix(test)

Refactor trash test to improve metadata validation

- Added new checks in trash tests to ensure metadata files are correctly moved to trash.
- Validated the presence and integrity of metadata within trashed files.

2024-11-25 - 3.3.4 - fix(core)

Minor refactoring and cleanup of TypeScript source files for improved readability and maintainability.

2024-11-24 - 3.3.3 - fix(documentation)

Improved documentation accuracy and consistency

- Updated the project description to reflect the cloud-agnostic nature and advanced capabilities
- Enhanced the README with detailed explanations and code examples for advanced features like trash management
- Clarified the handling and importance of metadata using the MetaData utility

2024-11-24 - 3.3.2 - fix(documentation)

Updated keywords and description for clarity and consistency.

- Modified keywords and description in package.json and npmextra.json.
- Enhanced readme.md file structure and examples

2024-11-24 - 3.3.1 - fix(File)

Fixed issue with file restore metadata operations.

- Corrected the order of operations in the file restore function to ensure custom metadata is appropriately deleted after moving the file.

2024-11-24 - 3.3.0 - feat(core)

Enhanced directory handling and file restoration from trash

- Refined getSubDirectoryByName to handle file paths treated as directories.
- Introduced file restoration function from trash to original or specified paths.

2024-11-24 - 3.2.2 - fix(core)

Refactor Bucket class for improved error handling

- Ensured safe access using non-null assertions when finding a bucket.
- Enhanced fastPut method by adding fastPutStrict for safer operations.
- Added explicit error handling and type checking in fastExists method.

2024-11-24 - 3.2.1 - fix(metadata)

Fix metadata handling for deleted files

- Ensured metadata is correctly stored and managed when files are deleted into the trash.

2024-11-24 - 3.2.0 - feat(bucket)

Enhanced SmartBucket with trash management and metadata handling

- Added functionality to move files to a trash directory.
- Introduced methods to handle file metadata more robustly.
- Implemented a method to clean all contents from a bucket.
- Enhanced directory retrieval to handle non-existent directories with options.
- Improved handling of file paths and metadata within the storage system.

2024-11-18 - 3.1.0 - feat(file)

Added functionality to retrieve magic bytes from files and detect file types using magic bytes.

- Introduced method `getMagicBytes` in `File` and `Bucket` classes to retrieve a specific number of bytes from a file.
- Enhanced file type detection by utilizing magic bytes in `MetaData` class.
- Updated dependencies for better performance and compatibility.

2024-11-18 - 3.0.24 - fix(metadata)

Fix metadata handling to address type assertion and data retrieval.

- Fixed type assertion issues in `MetaData` class properties with type non-null assertions.
- Corrected the handling of JSON data retrieval in `MetaData.storeCustomMetaData` function.

2024-10-16 - 3.0.23 - fix(dependencies)

Update package dependencies for improved functionality and security.

- Updated `@aws-sdk/client-s3` to version `^3.670.0` for enhanced S3 client capabilities.
- Updated `@push.rocks/smartstream` to version `^3.2.4`.
- Updated the dev dependency `@push.rocks/tapbundle` to version `^5.3.0`.

2024-07-28 - 3.0.22 - fix(dependencies)

Update dependencies and improve bucket retrieval logging

- Updated `@aws-sdk/client-s3` to `^3.620.0`

- Updated @git.zone/tsbuild to ^2.1.84
- Updated @git.zone/tsrun to ^1.2.49
- Updated @push.rocks/smartpromise to ^4.0.4
- Updated @tsclass/tsclass to ^4.1.2
- Added a log for when a bucket is not found by name in getBucketByName method

2024-07-04 - 3.0.21 - fix(test)

Update endpoint configuration in tests to use environment variable

- Modified `qenv.yml` to include `S3_ENDPOINT` as a required environment variable.
- Updated test files to fetch `S3_ENDPOINT` from environment instead of hardcoding.

2024-06-19 - 3.0.20 - Fix and Stability Updates

Improved overall stability and consistency.

2024-06-18 - 3.0.18 - Delete Functions Consistency

Ensured more consistency between delete methods and trash behavior.

2024-06-17 - 3.0.17 to 3.0.16 - Fix and Update

Routine updates and fixes performed.

2024-06-11 - 3.0.15 to 3.0.14 - Fix and Update

Routine updates and fixes performed.

2024-06-10 - 3.0.13 - Trash Feature Completion

Finished work on trash feature.

2024-06-09 - 3.0.12 - Fix and Update

Routine updates and fixes performed.

2024-06-08 - 3.0.11 to 3.0.10 - Fix and Update

Routine updates and fixes performed.

2024-06-03 - 3.0.10 - Fix and Update

Routine updates and fixes performed.

2024-05-29 - 3.0.9 - Update

Description

Updated project description.

2024-05-27 - 3.0.8 to 3.0.6 - Pathing and Core Updates

Routine updates and fixes performed.

- S3 paths' pathing differences now correctly handled with a reducePath method.

2024-05-21 - 3.0.5 to 3.0.4 - Fix and Update

Routine updates and fixes performed.

2024-05-17 - 3.0.3 to 3.0.2 - Fix and Update

Routine updates and fixes performed.

2024-05-17 - 3.0.0 - Major Release

Introduced breaking changes in core and significant improvements.

2024-05-05 - 2.0.5 - Breaking Changes

Introduced breaking changes in core functionality.

2024-04-14 - 2.0.4 - TSConfig Update

Updated TypeScript configuration.

2024-01-01 - 2.0.2 - Organization Scheme Update

Switched to the new organizational scheme.