@lossless.zone/objectstorage

Documentation for @lossless.zone/objectstorage

- readme.md for @lossless.zone/objectstorage
- changelog.md for @lossless.zone/objectstorage

readme.md for @lossless.zone/objectstorage

> □□ S3-compatible object storage server with clustering, erasure coding, and a slick management UI — powered by `smartstorage`.

objectstorage gives you a fully featured, self-hosted S3-compatible storage server with a beautiful web-based management interface — all in a single Docker image. No Java, no bloat, no fuss.

Built on Deno for the backend and `@design.estate/dees-catalog` for a polished UI, it speaks the S3 protocol out of the box while adding powerful management features on top. Scale from a single node to a distributed cluster with erasure coding and multi-drive support.

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.

□ Features

- **Full S3 API compatibility** — Works with any S3 client, SDK, or tool (AWS CLI, boto3, etc.)
- □□ **Cluster mode** — Distribute storage across multiple nodes with QUIC transport, automatic discovery, and quorum writes/reads

- □□ **Erasure coding** — Reed-Solomon erasure coding (default 4+2) for data durability with minimal overhead
- □□ **Multi-drive support** — Stripe data across multiple disks per node with per-drive health monitoring
- □□ **Self-healing** — Background scanner detects and reconstructs missing or corrupt shards automatically
- **Management UI** — Web dashboard for buckets, objects, policies, credentials, cluster config, and storage drives
- **Finder-style object browser** — Column view with file preview, drag-and-drop upload, move/rename, context menus
- **Inline code editing** — Built-in Monaco editor with syntax highlighting and save-back-to-storage
- **PDF viewer** — Render PDFs inline with page navigation, zoom, and thumbnails
- **Named policy management** — Create reusable IAM-style policies, attach them to multiple buckets
- **Credential management** — Add/remove access keys through the UI with live-reload
- **Single Docker image** — Multi-arch (`amd64` + `arm64`), tiny Alpine-based image
- **Fast** — Rust-powered storage engine via `smartstorage`, streaming I/O with zero-copy and backpressure
- **Secure by default** — JWT-based admin auth, S3 SigV4 authentication, bucket policies
- □□ **Dark theme** — Automatic dark mode following your system preference

⚡ Quick Start

Docker (recommended)

```
docker run -d \
  --name objectstorage \
  -p 9000:9000 \
  -p 3000:3000 \
  -v objstdata:/data \
  -e OBJST_ACCESS_KEY=myadminkey \
  -e OBJST_SECRET_KEY=mysupersecret \
  -e OBJST_ADMIN_PASSWORD=myuipassword \
  code.foss.global/lossless.zone/objectstorage:latest
```

Then open **http://localhost:3000** for the management UI and use **http://localhost:9000** as your S3 endpoint.

Deno (development)

```
# Clone and install frontend dependencies
git clone ssh://git@code.foss.global:29419/lossless.zone/objectstorage.git
cd objectstorage
pnpm install
pnpm run build

# Run in ephemeral mode (data stored in .nogit/objstdata)
deno run --allow-all mod.ts server --ephemeral
```

⚙ Configuration

objectstorage is configured through environment variables, CLI flags, or programmatic config. **Environment variables take precedence** over CLI flags.

Server Environment Variables

Variable	Description	Default
`OBJST_PORT`	Storage API port	`9000`
`UI_PORT`	Management UI port	`3000`
`OBJST_STORAGE_DIR`	Data storage directory	`/data`
`OBJST_ACCESS_KEY`	Access key ID	`admin`
`OBJST_SECRET_KEY`	Secret access key	`admin`
`OBJST_ADMIN_PASSWORD`	Admin UI password	`admin`
`OBJST_REGION`	Storage region identifier	`us-east-1`

Cluster Environment Variables

Variable	Description	Default
`OBJST_CLUSTER_ENABLED`	Enable cluster mode (`true`/`false`)	`false`
`OBJST_CLUSTER_NODE_ID`	Unique node identifier	auto-generated
`OBJST_CLUSTER_QUIC_PORT`	QUIC transport port	`4433`

Variable	Description	Default
`OBJST_CLUSTER_SEED_NODES`	Comma-separated seed node addresses	*(empty)*
`OBJST_DRIVE_PATHS`	Comma-separated drive mount paths	storage dir
`OBJST_ERASURE_DATA_SHARDS`	Erasure coding data shards	`4`
`OBJST_ERASURE_PARITY_SHARDS`	Erasure coding parity shards	`2`
`OBJST_ERASURE_CHUNK_SIZE`	Erasure chunk size in bytes	`4194304` (4 MB)
`OBJST_HEARTBEAT_INTERVAL_MS`	Cluster heartbeat interval	`5000`
`OBJST_HEARTBEAT_TIMEOUT_MS`	Cluster heartbeat timeout	`30000`

CLI Flags

```
deno run --allow-all mod.ts server [options]

Server Options:
  --storage-port <port>       Storage API port (default: 9000)
  --ui-port <port>            Management UI port (default: 3000)
  --storage-dir <path>        Storage directory (default: /data)
  --ephemeral                 Use ./.nogit/objstdata for storage (dev mode)

Clustering Options:
  --cluster-enabled           Enable cluster mode
  --cluster-node-id <id>      Unique node identifier
  --cluster-quic-port <port>  QUIC transport port (default: 4433)
  --cluster-seed-nodes <list> Comma-separated seed node addresses
  --drive-paths <list>        Comma-separated drive mount paths
  --erasure-data-shards <n>   Erasure coding data shards (default: 4)
  --erasure-parity-shards <n> Erasure coding parity shards (default: 2)
```

□□ Cluster Mode

objectstorage supports distributed storage across multiple nodes with automatic failover and data redundancy.

How it works

1. **Enable clustering** on each node with `OBJST_CLUSTER_ENABLED=true`
2. **Point nodes at each other** using `OBJST_CLUSTER_SEED_NODES` — nodes discover the full cluster from any seed
3. **Configure drives** per node with `OBJST_DRIVE_PATHS` — each drive is independently managed
4. **Erasure coding** splits objects into data + parity shards across drives and nodes

Example: 3-node cluster

```
# Node 1
docker run -d --name objst-node1 \
  -p 9000:9000 -p 3000:3000 -p 4433:4433/udp \
  -v /mnt/disk1:/drive1 -v /mnt/disk2:/drive2 \
  -e OBJST_CLUSTER_ENABLED=true \
  -e OBJST_CLUSTER_NODE_ID=node-1 \
  -e OBJST_CLUSTER_SEED_NODES=node2:4433,node3:4433 \
  -e OBJST_DRIVE_PATHS=/drive1,/drive2 \
  -e OBJST_ACCESS_KEY=myadminkey \
  -e OBJST_SECRET_KEY=mysupersecret \
  code.foss.global/lossless.zone/objectstorage:latest

# Node 2 and Node 3 — same pattern, different node IDs and seed nodes
```

Erasure coding presets

Config	Data Shards	Parity Shards	Overhead	Fault Tolerance
Default	4	2	50%	2 failures
High durability	6	3	50%	3 failures
Minimal	2	1	50%	1 failure

Inter-node transport

Cluster communication uses **QUIC** (UDP port 4433 by default) with:

- Auto-generated TLS certificates
- Multiplexed streams with flow-control backpressure
- Heartbeat-based failure detection (default: 5s interval, 30s timeout)

Management UI

The web-based management UI is served on the UI port (default: `3000`). Log in with username `admin` and the configured admin password.

Overview

Dashboard showing server status, uptime, storage usage, bucket count, and connection info.

Overview

Buckets

Create/delete buckets. View object counts and sizes. Attach/detach named policies per bucket.

Buckets

Browser

Finder-style column browser for objects. Upload, download, preview, move, rename, and delete files and folders — with syntax-highlighted code preview.

Browser

Right-click any item for quick actions:

Context Menu

Inline Code Editing

Click Edit on any text file to open the built-in Monaco editor with syntax highlighting, language detection, and save-back-to-storage.

Code Editing

PDF Viewer

PDF files render inline with a full-featured viewer — page navigation, zoom, fit-to-page, thumbnails, download, and print.

PDF Viewer

Policies

Create reusable named policies with IAM-style S3 statements. Attach policies to multiple buckets at once.

Policies

Attaching Policies to Buckets

From the Buckets view, click the policy icon on any bucket to see attached and available policies. Attach or detach with one click.

Attach Policy

Access Keys

Add and remove access credentials. Secret keys are masked. Changes take effect immediately — no server restart needed.

Access Keys

Adding Access Keys

Click "Add Key" to create new access credentials. They're immediately available for API authentication.

Add Access Key

Configuration

View your server's current configuration at a glance — ports, region, storage directory, auth/CORS status, cluster configuration, erasure coding settings, and storage drive paths. The config view also includes an environment variable reference guide for cluster setup.

Configuration

Dark Theme

Full dark theme support — automatically follows your system preference via `prefers-color-scheme`.

Dark Theme

□□ Named Policy System

objectstorage adds a **named policy** abstraction on top of standard S3 bucket policies. Instead of editing raw JSON per bucket, you define reusable policy templates and attach them to any number of buckets.

How it works

1. **Create a named policy** in the Policies view — give it a name, description, and S3 policy statements
2. **Attach it to buckets** — from the Policies view or the Buckets view
3. **objectstorage merges** all attached policy statements into a single S3 policy document and applies it to the bucket automatically

`${bucket}` placeholder

Use `${bucket}` in your policy's `Resource` ARN and it will be replaced with the actual bucket name when applied:

```
[
  {
    "Sid": "PublicRead",
    "Effect": "Allow",
    "Principal": "*",
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::${bucket}/*"
  }
]
```

This lets one policy like "Public Read" work across many buckets without hardcoding names.

Lifecycle

- **Updating a policy** automatically recomputes and re-applies the merged S3 policy on all attached buckets
- **Deleting a policy** detaches it from all buckets and recomputes each
- **Deleting a bucket** cleans up its policy attachments automatically

□□ S3 API Usage

Use any S3-compatible client to interact with the storage. Here are some examples:

AWS CLI

```
# Configure AWS CLI
aws configure set aws_access_key_id admin
aws configure set aws_secret_access_key admin
aws configure set default.region us-east-1

# Create a bucket
aws --endpoint-url http://localhost:9000 s3 mb s3://my-bucket

# Upload a file
aws --endpoint-url http://localhost:9000 s3 cp myfile.txt s3://my-bucket/

# List objects
aws --endpoint-url http://localhost:9000 s3 ls s3://my-bucket/

# Download a file
aws --endpoint-url http://localhost:9000 s3 cp s3://my-bucket/myfile.txt ./downloaded.txt
```

Node.js / TypeScript (AWS SDK v3)

```
import { S3Client, PutObjectCommand, ListObjectsV2Command } from '@aws-sdk/client-s3';

const s3 = new S3Client({
  endpoint: 'http://localhost:9000',
  region: 'us-east-1',
  credentials: {
    accessKeyId: 'admin',
    secretAccessKey: 'admin',
  },
  forcePathStyle: true,
});
```

```
// Upload an object
await s3.send(new PutObjectCommand({
  Bucket: 'my-bucket',
  Key: 'hello.txt',
  Body: 'Hello, S3!',
}));

// List objects in a bucket
const result = await s3.send(new ListObjectsV2Command({
  Bucket: 'my-bucket',
}));
console.log(result.Contents);
```

Python (boto3)

```
import boto3

s3 = boto3.client(
    's3',
    endpoint_url='http://localhost:9000',
    aws_access_key_id='admin',
    aws_secret_access_key='admin',
    region_name='us-east-1',
)

# Upload a file
s3.put_object(Bucket='my-bucket', Key='hello.txt', Body=b'Hello from Python!')

# List objects
response = s3.list_objects_v2(Bucket='my-bucket')
for obj in response.get('Contents', []):
    print(obj['Key'], obj['Size'])
```

□□ Docker

Build

```
# Build for the native platform
pnpm run build:docker

# Or build and run directly
pnpm run start:docker
```

Docker Compose (standalone)

```
services:
  objectstorage:
    image: code.foss.global/lossless.zone/objectstorage:latest
    ports:
      - "9000:9000"  # S3 API
      - "3000:3000"  # Management UI
    volumes:
      - objstdata:/data
    environment:
      OBJST_ACCESS_KEY: myadminkey
      OBJST_SECRET_KEY: mysupersecret
      OBJST_ADMIN_PASSWORD: securepw123

volumes:
  objstdata:
```

Docker Compose (3-node cluster)

```
services:
  node1:
    image: code.foss.global/lossless.zone/objectstorage:latest
    ports:
      - "9001:9000"
      - "3001:3000"
      - "4433:4433/udp"
    volumes:
```

```
    - node1-drive1:/drive1
    - node1-drive2:/drive2
  environment:
    OBJST_CLUSTER_ENABLED: "true"
    OBJST_CLUSTER_NODE_ID: node-1
    OBJST_CLUSTER_QUIC_PORT: "4433"
    OBJST_CLUSTER_SEED_NODES: node2:4433,node3:4433
    OBJST_DRIVE_PATHS: /drive1,/drive2
    OBJST_ACCESS_KEY: myadminkey
    OBJST_SECRET_KEY: mysupersecret

node2:
  image: code.foss.global/lossless.zone/objectstorage:latest
  ports:
    - "9002:9000"
    - "3002:3000"
    - "4434:4433/udp"
  volumes:
    - node2-drive1:/drive1
    - node2-drive2:/drive2
  environment:
    OBJST_CLUSTER_ENABLED: "true"
    OBJST_CLUSTER_NODE_ID: node-2
    OBJST_CLUSTER_QUIC_PORT: "4433"
    OBJST_CLUSTER_SEED_NODES: node1:4433,node3:4433
    OBJST_DRIVE_PATHS: /drive1,/drive2
    OBJST_ACCESS_KEY: myadminkey
    OBJST_SECRET_KEY: mysupersecret

node3:
  image: code.foss.global/lossless.zone/objectstorage:latest
  ports:
    - "9003:9000"
    - "3003:3000"
    - "4435:4433/udp"
  volumes:
    - node3-drive1:/drive1
    - node3-drive2:/drive2
  environment:
```

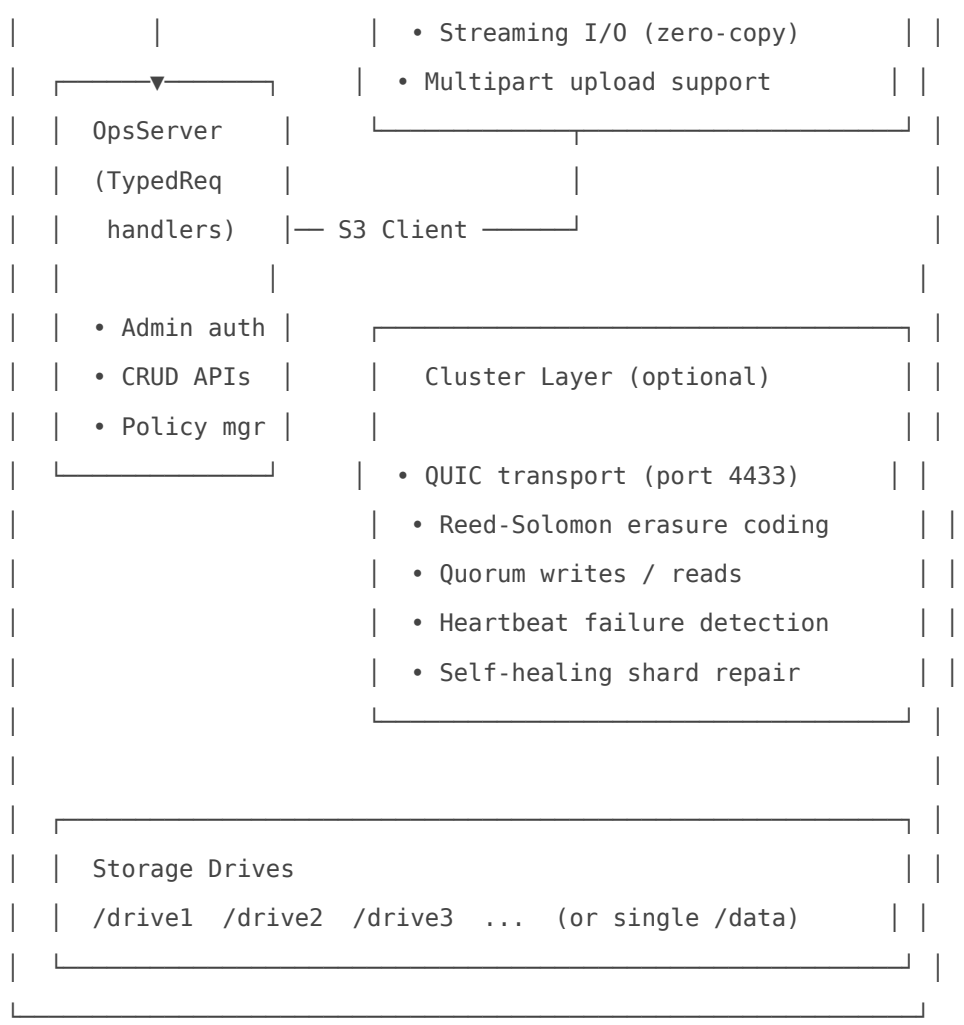
```
      OBJST_CLUSTER_ENABLED: "true"
      OBJST_CLUSTER_NODE_ID: node-3
      OBJST_CLUSTER_QUIC_PORT: "4433"
      OBJST_CLUSTER_SEED_NODES: node1:4433,node2:4433
      OBJST_DRIVE_PATHS: /drive1,/drive2
      OBJST_ACCESS_KEY: myadminkey
      OBJST_SECRET_KEY: mysupersecret

volumes:
  node1-drive1:
  node1-drive2:
  node2-drive1:
  node2-drive2:
  node3-drive1:
  node3-drive2:
```

Image Details

- **Base**: `alpine:edge` with Deno runtime
- **Architectures**: `linux/amd64`, `linux/arm64`
- **Size**: ~150 MB compressed
- **Init system**: `tini` for proper signal handling
- **Exposed ports**: `9000` (S3), `3000` (UI), `4433` (QUIC cluster transport)
- **Volume**: `/data` — all bucket data and config persisted here

□□ Architecture

```
┌──────────────────────────────────────────────────────────┐
│                     objectstorage                        │
│                                                          │
│  ┌─────────────────┐     ┌──────────────────────────────┐ │
│  │  Management     │     │   Storage Engine (smartstorage)  │ │
│  │  UI (port       │     │   Rust binary via ruststorage    │ │
│  │   3000)         │     │   (port 9000)                │ │
│  │                 │     │                              │ │
│  │  dees-catalog   │     │  • S3 API (path-style routing)  │ │
│  │  SPA bundle     │     │  • SigV4 authentication         │ │
│  └────────┬────────┘     │  • Bucket policies              │ │
```



Tech Stack

Layer	Technology
Storage Engine	`@push.rocks/smartstorage` (Rust binary via `ruststorage`)
Cluster Transport	QUIC via `quinn` (auto-TLS, multiplexed streams, backpressure)
Erasure Coding	Reed-Solomon (configurable data + parity shards)
Runtime	Deno
Management API	`@api.global/typedrequest` + `@api.global/typedserver`
Auth	JWT via `@push.rocks/smartjwt`, S3 SigV4
Frontend	`@design.estate/dees-element` (LitElement) + `@design.estate/dees-catalog`
Frontend Build	esbuild via `@git.zone/tsbundle`
Docker	Multi-stage (Node.js build → Alpine + Deno runtime)

□□ Development

```
# Install dependencies
pnpm install

# Watch mode — auto-rebuilds frontend + restarts backend
pnpm run watch

# Build frontend bundle only
pnpm run build

# Type check backend
deno check mod.ts

# Run tests
pnpm test

# Run in development mode
deno run --allow-all mod.ts server --ephemeral
```

License and Legal Information

Trademarks

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 @lossless.zone/objectstorage

2026-03-24 - 1.8.1 - fix(build)

migrate build tool config to .smartconfig.json and bump tooling dependencies

- Move tsbundle, tsdocker, and tswatch configuration from npmextra.json to .smartconfig.json
- Update @git.zone/tsbundle, @git.zone/tsdocker, and @git.zone/tswatch devDependencies
- Bump @aws-sdk/client-s3 import to ^3.1016.0

2026-03-24 - 1.8.0 - feat(docs,web)

document cluster and erasure coding support and align UI storage provider naming

- expand the README with cluster mode, erasure coding, multi-drive configuration, and multi-node Docker Compose examples
- update web storage integration to use the renamed storage data provider interface and browser component
- bump runtime, UI, and tooling dependencies to newer compatible versions

2026-03-22 - 1.7.1 - fix(repo)

no changes to commit

2026-03-22 - 1.7.0 - feat(cluster)

add cluster configuration support across server, CLI, and admin UI

- add environment variable and CLI support for cluster, QUIC, seed node, drive, erasure coding, and heartbeat settings
- pass cluster-aware configuration into smartstorage and expose the QUIC port in Docker
- extend config APIs and admin UI to display cluster, erasure coding, and storage drive configuration
- upgrade @push.rocks/smartstorage to ^6.3.1 to support the new cluster capabilities

2026-03-21 - 1.6.0 - feat(scripts)

add release script for committing and pushing docker images

- Adds a new npm release script to automate git commit and Docker image push steps.

2026-03-21 - 1.5.1 - fix(project)

no changes to commit

2026-03-21 - 1.5.0 - feat(test)

add end-to-end test coverage for container lifecycle, auth, buckets, objects, policies, credentials, status, and S3 compatibility

- adds a reusable test helper for creating isolated ObjectStorageContainer instances and logging in as admin
- introduces a test script in package.json to run the new Deno test suite
- covers core management API flows including authentication, bucket operations, object operations, named policies, credentials, and server status/config
- verifies S3 SDK interoperability including bucket and object operations plus credential rejection cases

2026-03-15 - 1.4.2 - fix(license)

add missing license file

- Adds a repository license file.

2026-03-15 - 1.4.1 - fix(readme)

refresh README with clearer feature documentation, UI screenshots, and client usage examples

- Adds newly documented UI capabilities such as inline code editing, PDF viewing, dark theme, and configuration overview
- Expands S3 usage examples with AWS CLI, Node.js/TypeScript, and Python boto3 snippets
- Improves README structure, section headings, linked dependency references, and screenshot coverage

2026-03-14 - 1.5.0 - feat(core)

rebrand from s3container to objectstorage

- Rename project from @lossless.zone/s3container to @lossless.zone/objectstorage
- Replace @push.rocks/smarts3 dependency with @push.rocks/smartstorage
- Change all environment variable prefixes from S3_ to OBJST_
- Rename S3Container class to ObjectStorageContainer
- Update all web component prefixes from s3c- to objst-
- Update UI labels, CLI flags, documentation, and Docker configuration

2026-03-14 - 1.4.0 - feat(docs)

expand management UI documentation with screenshots and feature walkthroughs

- Reorganizes the Management UI section into dedicated Overview, Buckets, Browser, Policies, Access Keys, and Dark Theme sections
- Adds screenshots for core UI views and workflows such as inline code editing, PDF viewing, attaching policies, and adding access keys
- Clarifies browser capabilities including syntax-highlighted code preview and save-back-to-S3 editing

2026-03-14 - 1.3.0 - feat(web-router)

add URL-based view routing and synchronize navigation state across the web UI

- introduce a dedicated app router that maps valid views to browser paths and handles initial route resolution
- update app shell and bucket navigation to use router-based view changes instead of directly mutating UI state
- refresh policy and bucket management modals by updating modal content through the created modal instance

2026-03-14 - 1.2.0 - feat(readme)

add comprehensive project documentation for setup, configuration, UI, and S3 usage

- Adds an initial README covering Docker and Deno quick start flows
- Documents environment variables, CLI flags, management UI views, and named policy behavior
- Includes S3 client usage examples, Docker deployment guidance, architecture overview, and development instructions

2026-03-14 - 1.1.3 - fix(package)

update package metadata

- Adjusts package.json with a single-line metadata change.

2026-03-14 - 1.1.2 - fix(package)

update package metadata

- Adjusts package.json with a single-line metadata change.

2026-03-14 - 1.1.1 - fix(package)

update package metadata

- Adjusts package.json with a single-line metadata change.

2026-03-14 - 1.1.0 - feat(policies)

add named policy management with bucket attachments in the ops API and web UI

- introduces a PolicyManager that persists named policies and bucket attachments, then recomputes and applies merged bucket policies automatically
- adds typed ops server endpoints and shared request/data interfaces for creating, updating, deleting, listing, and attaching policies
- adds a dedicated Policies view in the web UI and replaces raw bucket policy JSON editing with attach/detach management for named policies
- cleans up policy attachments when buckets are deleted

2026-03-14 - 1.0.0 - initial release

Initial release of s3container, a MinIO replacement built on smarts3 with a dees-catalog-based management UI.

- Added an S3-compatible storage server powered by `@push.rocks/smarts3`
- Added a management UI with `dees-s3-browser` for object browsing
- Added bucket management with policy support
- Added credential management for access key / secret key pairs
- Added Docker containerization with a multi-stage build using Node.js and Deno