readme.md for @push.rocks/smartregistry

> □□ A composable TypeScript library implementing **OCI Distribution Specification v1.1**, **NPM Registry API**, **Maven Repository**, **Cargo/crates.io Registry**, **Composer/Packagist**, **PyPI (Python Package Index)**, and **RubyGems Registry** — everything you need to build a unified container and package registry in one library.

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

□□ Multi-Protocol Support

- **OCI Distribution Spec v1.1**: Full container registry with manifest/blob operations
- **NPM Registry API**: Complete package registry with publish/install/search
- **Maven Repository**: Java/JVM artifact management with POM support
- **Cargo/crates.io Registry**: Rust crate registry with sparse HTTP protocol
- **Composer/Packagist**: PHP package registry with Composer v2 protocol
- **PyPI (Python Package Index)**: Python package registry with PEP 503/691 support
- **RubyGems Registry**: Ruby gem registry with compact index protocol

□□ Unified Architecture

- **Composable Design**: Core infrastructure with protocol plugins — enable only what you need
- **Shared Storage**: Cloud-agnostic S3-compatible backend via @push.rocks/smartbucket with standardized `IS3Descriptor` from @tsclass/tsclass
- **Unified Authentication**: Scope-based permissions across all protocols
- **Path-based Routing**: `/oci/*`, `/npm/*`, `/maven/*`, `/cargo/*`, `/composer/*`, `/pypi/*`, `/rubygems/*`

□□ Authentication & Authorization

- NPM UUID tokens for package operations
- OCI JWT tokens for container operations
- Protocol-specific tokens for Maven, Cargo, Composer, PyPI, and RubyGems
- Unified scope system: `npm:package:foo:write`, `oci:repository:bar:push`
- **Pluggable Auth Provider** (`IAuthProvider`): Integrate LDAP, OAuth, SSO, or any custom auth

□□ Protocol Feature Matrix

Feature	OCI	NPM	Maven	Cargo	Composer	PyPI	RubyGems
Publish/Upload	□	□	□	□	□	□	□
Download	□	□	□	□	□	□	□
Search	—	□	—	□	—	—	—
Version Yank	—	—	—	□	—	—	□
Metadata API	□	□	□	□	□	□	□
Token Auth	□	□	□	□	□	□	□
Checksum Verification	□	□	□	□	—	□	□
Upstream Proxy	□	□	□	□	□	□	□

□□ Upstream Proxy & Caching

- **Multi-Upstream Support**: Configure multiple upstream registries per protocol with priority ordering

- **Scope-Based Routing**: Route specific packages/scopes to different upstreams (e.g., `@company/*` → private registry)
- **S3-Backed Cache**: Persistent caching using existing S3 storage
- **Circuit Breaker**: Automatic failover with configurable thresholds
- **Stale-While-Revalidate**: Serve cached content while refreshing in background
- **Content-Aware TTLs**: Different TTLs for immutable (tarballs) vs mutable (metadata) content

□□ Streaming-First Architecture

- **Web Streams API** (`ReadableStream<Uint8Array>`) — cross-runtime (Node, Deno, Bun)
- **Zero-copy downloads**: Binary artifacts stream directly from S3 to the HTTP response
- **OCI upload streaming**: Chunked blob uploads stored as temp S3 objects, not accumulated in memory
- **Unified response type**: Every `response.body` is a `ReadableStream` — one pattern for all consumers

□□ Enterprise Extensibility

- **Storage Event Hooks** (`IStorageHooks`): Quota tracking, audit logging, virus scanning, cache invalidation
- **Request Actor Context**: Pass user/org info through requests for audit trails and rate limiting

□□ Installation

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

# Using npm
npm install @push.rocks/smartregistry
```

□□ Quick Start

```
import { SmartRegistry, IRegistryConfig } from '@push.rocks/smartregistry';
```

```
const config: IRegistryConfig = {
  storage: {
    accessKey: 'your-s3-key',
    accessSecret: 'your-s3-secret',
    endpoint: 's3.amazonaws.com',
    port: 443,
    useSsl: true,
    region: 'us-east-1',
    bucketName: 'my-registry',
  },
  auth: {
    jwtSecret: 'your-secret-key',
    tokenStore: 'memory',
    npmTokens: { enabled: true },
    ociTokens: {
      enabled: true,
      realm: 'https://auth.example.com/token',
      service: 'my-registry',
    },
  },
  // Enable only the protocols you need
  oci: { enabled: true, basePath: '/oci' },
  npm: { enabled: true, basePath: '/npm' },
  maven: { enabled: true, basePath: '/maven' },
  cargo: { enabled: true, basePath: '/cargo' },
  composer: { enabled: true, basePath: '/composer' },
  pypi: { enabled: true, basePath: '/pypi' },
  rubygems: { enabled: true, basePath: '/rubygems' },
};

const registry = new SmartRegistry(config);
await registry.init();

// Handle any incoming HTTP request — the router does the rest
const response = await registry.handleRequest({
  method: 'GET',
  path: '/npm/express',
  headers: {},
  query: {},
```

```
});
```

□□ Architecture

Request Flow

```
HTTP Request
    ↓
SmartRegistry (orchestrator)
    ↓
Path-based routing
    ├─→ /oci/*       → OciRegistry
    ├─→ /npm/*       → NpmRegistry
    ├─→ /maven/*     → MavenRegistry
    ├─→ /cargo/*     → CargoRegistry
    ├─→ /composer/*  → ComposerRegistry
    ├─→ /pypi/*      → PypiRegistry
    └─→ /rubygems/*  → RubyGemsRegistry
           ↓
    Shared Storage & Auth
           ↓
    S3-compatible backend
```

Directory Structure

```
ts/
├── core/                   # Shared infrastructure
│   ├── classes.baseregistry.ts
│   ├── classes.registrystorage.ts
│   ├── classes.authmanager.ts
│   └── interfaces.core.ts
├── oci/                    # OCI implementation
├── npm/                    # NPM implementation
├── maven/                  # Maven implementation
├── cargo/                  # Cargo implementation
```

```
├── composer/                # Composer implementation
├── pypi/                    # PyPI implementation
├── rubygems/                # RubyGems implementation
├── upstream/                # Upstream proxy infrastructure
└── classes.smartregistry.ts # Main orchestrator
```

Usage Examples

OCI Registry (Container Images)

```
// Pull a manifest
const response = await registry.handleRequest({
  method: 'GET',
  path: '/oci/library/nginx/manifests/latest',
  headers: { 'Authorization': 'Bearer <token>' },
  query: {},
});

// Push a blob (two-step upload)
const uploadInit = await registry.handleRequest({
  method: 'POST',
  path: '/oci/myapp/blobs/uploads/',
  headers: { 'Authorization': 'Bearer <token>' },
  query: {},
});

const uploadId = uploadInit.headers['Docker-Upload-UUID'];

await registry.handleRequest({
  method: 'PUT',
  path: `/oci/myapp/blobs/uploads/${uploadId}`,
  headers: { 'Authorization': 'Bearer <token>' },
  query: { digest: 'sha256:abc123...' },
  body: blobData,
});
```

□□ NPM Registry

```
// Get package metadata
const metadata = await registry.handleRequest({
  method: 'GET',
  path: '/npm/express',
  headers: {},
  query: {},
});

// Publish a package
const publishResponse = await registry.handleRequest({
  method: 'PUT',
  path: '/npm/my-package',
  headers: { 'Authorization': 'Bearer <npm-token>' },
  query: {},
  body: {
    name: 'my-package',
    versions: { '1.0.0': { /* version metadata */ } },
    'dist-tags': { latest: '1.0.0' },
    _attachments: {
      'my-package-1.0.0.tgz': {
        content_type: 'application/octet-stream',
        data: '<base64-tarball>',
        length: 12345,
      },
    },
  },
});

// Search packages
const search = await registry.handleRequest({
  method: 'GET',
  path: '/npm/-/v1/search',
  headers: {},
  query: { text: 'express', size: '20' },
});
```

□□ Cargo Registry (Rust Crates)

```
// Get registry config (required for Cargo sparse protocol)
const config = await registry.handleRequest({
  method: 'GET',
  path: '/cargo/config.json',
  headers: {},
  query: {},
});

// Publish a crate (binary format: [4 bytes JSON len][JSON][4 bytes crate len][.crate])
const publishResponse = await registry.handleRequest({
  method: 'PUT',
  path: '/cargo/api/v1/crates/new',
  headers: { 'Authorization': '<cargo-token>' },
  query: {},
  body: binaryPublishData,
});

// Yank a version
await registry.handleRequest({
  method: 'DELETE',
  path: '/cargo/api/v1/crates/my-crate/0.1.0/yank',
  headers: { 'Authorization': '<cargo-token>' },
  query: {},
});
```

Using with Cargo CLI:

```
# .cargo/config.toml
[registries.myregistry]
index = "sparse+https://registry.example.com/cargo/"
```

```
cargo publish --registry=myregistry
cargo install --registry=myregistry my-crate
```

□□ Composer Registry (PHP Packages)

```
// Get repository root
const packagesJson = await registry.handleRequest({
  method: 'GET',
  path: '/composer/packages.json',
  headers: {},
  query: {},
});

// Upload a package (ZIP with composer.json inside)
const uploadResponse = await registry.handleRequest({
  method: 'PUT',
  path: '/composer/packages/vendor/package',
  headers: { 'Authorization': 'Bearer <composer-token>' },
  query: {},
  body: zipBuffer,
});
```

Using with Composer CLI:

```
{
  "repositories": [
    { "type": "composer", "url": "https://registry.example.com/composer" }
  ]
}
```

```
composer require vendor/package
```

□□ PyPI Registry (Python Packages)

```
// Get package index (PEP 503 HTML)
const htmlIndex = await registry.handleRequest({
  method: 'GET',
  path: '/simple/requests/',
  headers: { 'Accept': 'text/html' },
  query: {},
});

// Get package index (PEP 691 JSON)
const jsonIndex = await registry.handleRequest({
```

```
  method: 'GET',
  path: '/simple/requests/',
  headers: { 'Accept': 'application/vnd.pypi.simple.v1+json' },
  query: {},
});

// Upload a package
const upload = await registry.handleRequest({
  method: 'POST',
  path: '/pypi/',
  headers: {
    'Authorization': 'Bearer <pypi-token>',
    'Content-Type': 'multipart/form-data',
  },
  query: {},
  body: {
    ':action': 'file_upload',
    protocol_version: '1',
    name: 'my-package',
    version: '1.0.0',
    filetype: 'bdist_wheel',
    content: wheelData,
    filename: 'my_package-1.0.0-py3-none-any.whl',
  },
});
```

Using with pip:

```
pip install --index-url https://registry.example.com/simple/ my-package
python -m twine upload --repository-url https://registry.example.com/pypi/ dist/*
```

□□ RubyGems Registry

```
// Upload a gem
const uploadGem = await registry.handleRequest({
  method: 'POST',
  path: '/rubygems/api/v1/gems',
  headers: { 'Authorization': '<rubygems-api-key>' },
  query: {},
```

```
  body: gemBuffer,
});

// Get compact index
const versions = await registry.handleRequest({
  method: 'GET',
  path: '/rubygems/versions',
  headers: {},
  query: {},
});
```

Using with Bundler:

```
# Gemfile
source 'https://registry.example.com/rubygems' do
  gem 'my-gem'
end
```

```
gem push my-gem-1.0.0.gem --host https://registry.example.com/rubygems
bundle install
```

□□ Authentication

```
const authManager = registry.getAuthManager();

// Authenticate user
const userId = await authManager.authenticate({ username: 'user', password: 'pass' });

// Create protocol-specific tokens
const npmToken = await authManager.createNpmToken(userId, false);
const ociToken = await authManager.createOciToken(userId, ['oci:repository:myapp:push'],
3600);
const pypiToken = await authManager.createPypiToken(userId, false);
const cargoToken = await authManager.createCargoToken(userId, false);
const composerToken = await authManager.createComposerToken(userId, false);
const rubygemsToken = await authManager.createRubyGemsToken(userId, false);

// Validate and check permissions
const token = await authManager.validateToken(npmToken, 'npm');
```

```
const canWrite = await authManager.authorize(token, 'npm:package:my-package', 'write');
```

□□ Upstream Proxy Configuration

```
import { SmartRegistry, StaticUpstreamProvider } from '@push.rocks/smartregistry';

const upstreamProvider = new StaticUpstreamProvider({
  npm: {
    enabled: true,
    upstreams: [
      {
        id: 'company-private',
        url: 'https://npm.internal.company.com',
        priority: 1,
        enabled: true,
        scopeRules: [{ pattern: '@company/*', action: 'include' }],
        auth: { type: 'bearer', token: process.env.NPM_PRIVATE_TOKEN },
      },
      {
        id: 'npmjs',
        url: 'https://registry.npmjs.org',
        priority: 10,
        enabled: true,
        scopeRules: [{ pattern: '@company/*', action: 'exclude' }],
      },
    ],
    cache: { enabled: true, staleWhileRevalidate: true },
  },
  oci: {
    enabled: true,
    upstreams: [
      { id: 'dockerhub', url: 'https://registry-1.docker.io', priority: 1, enabled: true },
    ],
  },
});

const registry = new SmartRegistry({
  storage: { /* S3 config */ },
```

```
  auth: { /* Auth config */ },
  upstreamProvider,
  npm: { enabled: true, basePath: '/npm' },
  oci: { enabled: true, basePath: '/oci' },
});
```

Custom Auth Provider

```
import { SmartRegistry, IAuthProvider, IAuthToken, TRegistryProtocol } from
'@push.rocks/smartregistry';

class LdapAuthProvider implements IAuthProvider {
  async init() { /* connect to LDAP */ }

  async authenticate(credentials) {
    const result = await this.ldapClient.bind(credentials.username, credentials.password);
    return result.success ? credentials.username : null;
  }

  async validateToken(token: string, protocol?: TRegistryProtocol): Promise<IAuthToken | null>
{
    const session = await this.sessionStore.get(token);
    return session ? { userId: session.userId, scopes: session.scopes } : null;
  }

  async createToken(userId: string, protocol: TRegistryProtocol, options?) {
    const token = crypto.randomUUID();
    await this.sessionStore.set(token, { userId, protocol, ...options });
    return token;
  }

  async revokeToken(token: string) { await this.sessionStore.delete(token); }

  async authorize(token: IAuthToken | null, resource: string, action: string) {
    if (!token) return action === 'read';
    return this.checkPermissions(token.userId, resource, action);
  }
}
```

```
const registry = new SmartRegistry({
  ...config,
  authProvider: new LdapAuthProvider(),
});
```

□□ Storage Hooks (Quota & Audit)

```
import { SmartRegistry, IStorageHooks, IStorageHookContext } from '@push.rocks/smartregistry';

const storageHooks: IStorageHooks = {
  async beforePut(ctx: IStorageHookContext) {
    if (ctx.actor?.orgId) {
      const usage = await getStorageUsage(ctx.actor.orgId);
      const quota = await getQuota(ctx.actor.orgId);
      if (usage + (ctx.metadata?.size || 0) > quota) {
        return { allowed: false, reason: 'Storage quota exceeded' };
      }
    }
    return { allowed: true };
  },

  async afterPut(ctx: IStorageHookContext) {
    await auditLog.write({
      action: 'storage.put',
      key: ctx.key,
      protocol: ctx.protocol,
      actor: ctx.actor,
      timestamp: ctx.timestamp,
    });
  },

  async beforeDelete(ctx: IStorageHookContext) {
    if (await isProtectedPackage(ctx.key)) {
      return { allowed: false, reason: 'Cannot delete protected package' };
    }
    return { allowed: true };
  },
```

```
};

const registry = new SmartRegistry({ ...config, storageHooks });
```

□□ Request Actor Context

```
// Pass actor information for audit/quota tracking
const response = await registry.handleRequest({
  method: 'PUT',
  path: '/npm/my-package',
  headers: { 'Authorization': 'Bearer <token>' },
  query: {},
  body: packageData,
  actor: {
    userId: 'user123',
    tokenId: 'token-abc',
    ip: req.ip,
    userAgent: req.headers['user-agent'],
    orgId: 'org-456',
  },
});
```

⚙ Configuration

Storage Configuration

Extends `IS3Descriptor` from `@tsclass/tsclass`:

```
storage: {
  accessKey: string;        // S3 access key
  accessSecret: string;     // S3 secret key
  endpoint: string;         // S3 endpoint (e.g., 's3.amazonaws.com')
  port?: number;            // Default: 443
  useSsl?: boolean;         // Default: true
  region?: string;          // AWS region
  bucketName: string;       // Bucket name for registry storage
```

```
}
```

Authentication Configuration

```
auth: {
  jwtSecret: string;
  tokenStore: 'memory' | 'redis' | 'database';
  npmTokens: { enabled: boolean; defaultReadonly?: boolean };
  ociTokens: { enabled: boolean; realm: string; service: string };
  pypiTokens: { enabled: boolean };
  rubygemsTokens: { enabled: boolean };
}
```

Protocol Configuration

Each protocol accepts:

```
{
  enabled: boolean;
  basePath: string;         // URL prefix, e.g. '/npm'
  registryUrl?: string;     // Public-facing base URL (used in generated metadata links)
  features?: Record<string, boolean>;
}
```

The `registryUrl` is important when the registry is served behind a reverse proxy or on a non-default port. For example, if your server is at `https://registry.example.com`, set `registryUrl: 'https://registry.example.com/npm'` for the NPM protocol so that generated metadata URLs point to the correct host.

□□ API Reference

Core Classes

SmartRegistry

Main orchestrator — routes requests to the appropriate protocol handler.

Method	Description
`init()`	Initialize the registry and all enabled protocols
`handleRequest(context)`	Route and handle an HTTP request
`getStorage()`	Get the shared `RegistryStorage` instance
`getAuthManager()`	Get the shared `AuthManager` instance
`getRegistry(protocol)`	Get a specific protocol handler by name
`isInitialized()`	Check if the registry has been initialized
`destroy()`	Clean up resources

Protocol Endpoints

OCI Registry

Method	Path	Description
`GET`	`/{name}/manifests/{ref}`	Get manifest by tag or digest
`PUT`	`/{name}/manifests/{ref}`	Push manifest
`GET`	`/{name}/blobs/{digest}`	Get blob
`POST`	`/{name}/blobs/uploads/`	Initiate blob upload
`PUT`	`/{name}/blobs/uploads/{uuid}`	Complete blob upload
`GET`	`/{name}/tags/list`	List tags
`GET`	`/{name}/referrers/{digest}`	Get referrers (OCI 1.1)

NPM Registry

Method	Path	Description
`GET`	`/{package}`	Get package metadata (packument)
`PUT`	`/{package}`	Publish package
`GET`	`/{package}/-/{tarball}`	Download tarball
`GET`	`/-/v1/search?text=...`	Search packages
`PUT`	`/-/user/org.couchdb.user:{user}`	Login
`GET/POST/DELETE`	`/-/npm/v1/tokens`	Token management
`PUT`	`/-/package/{pkg}/dist-tags/{tag}`	Manage dist-tags

Maven Repository

Method	Path	Description
`PUT`	`/{group}/{artifact}/{version}/{file}`	Upload artifact
`GET`	`/{group}/{artifact}/{version}/{file}`	Download artifact
`GET`	`/{group}/{artifact}/maven-metadata.xml`	Get metadata

Cargo Registry

Method	Path	Description
`GET`	`/config.json`	Registry configuration
`GET`	`/{p1}/{p2}/{name}`	Sparse index entry
`PUT`	`/api/v1/crates/new`	Publish crate (binary format)
`GET`	`/api/v1/crates/{crate}/{version}/download`	Download .crate
`DELETE`	`/api/v1/crates/{crate}/{version}/yank`	Yank version
`PUT`	`/api/v1/crates/{crate}/{version}/unyank`	Unyank version
`GET`	`/api/v1/crates?q=...`	Search crates

Composer Registry

Method	Path	Description
`GET`	`/packages.json`	Repository metadata
`GET`	`/p2/{vendor}/{package}.json`	Package version metadata
`GET`	`/packages/list.json`	List all packages
`GET`	`/dists/{vendor}/{package}/{ref}.zip`	Download package ZIP
`PUT`	`/packages/{vendor}/{package}`	Upload package
`DELETE`	`/packages/{vendor}/{package}[/{version}]`	Delete package/version

PyPI Registry

Method	Path	Description
`GET`	`/simple/`	List all packages (PEP 503/691)
`GET`	`/simple/{package}/`	List package files
`POST`	`/`	Upload package (multipart)
`GET`	`/pypi/{package}/json`	Package metadata API

Method	Path	Description
`GET`	`/pypi/{package}/{version}/json`	Version metadata
`GET`	`/packages/{package}/{filename}`	Download file

RubyGems Registry

Method	Path	Description
`GET`	`/versions`	Master versions file (compact index)
`GET`	`/info/{gem}`	Gem info file
`GET`	`/names`	List all gem names
`POST`	`/api/v1/gems`	Upload .gem file
`DELETE`	`/api/v1/gems/yank`	Yank version
`PUT`	`/api/v1/gems/unyank`	Unyank version
`GET`	`/api/v1/versions/{gem}.json`	Version metadata
`GET`	`/gems/{gem}-{version}.gem`	Download .gem file

□□ Scope Format

Unified scope format across all protocols:

```
{protocol}:{type}:{name}:{action}

Examples:
  npm:package:express:read          # Read express package
  npm:package:*:write               # Write any package
  oci:repository:nginx:pull         # Pull nginx image
  oci:repository:*:push             # Push any image
  cargo:crate:serde:write           # Write serde crate
  composer:package:vendor/pkg:read  # Read Composer package
  pypi:package:requests:read        # Read PyPI package
  rubygems:gem:rails:write          # Write RubyGems gem
  {protocol}:*:*:*                  # Full access for a protocol
```

□□ Storage Structure

```
bucket/
├── oci/
│   ├── blobs/sha256/{hash}
│   ├── manifests/{repository}/{digest}
│   └── tags/{repository}/tags.json
├── npm/
│   └── packages/{name}/
│       ├── index.json              # Packument
│       └── {name}-{ver}.tgz       # Tarball
├── maven/
│   ├── artifacts/{group}/{artifact}/{version}/
│   └── metadata/{group}/{artifact}/maven-metadata.xml
├── cargo/
│   ├── config.json
│   ├── index/{p1}/{p2}/{name}     # Sparse index
│   └── crates/{name}/{name}-{ver}.crate
├── composer/
│   └── packages/{vendor}/{package}/
│       ├── metadata.json
│       └── {reference}.zip
├── pypi/
│   ├── simple/index.html
│   ├── simple/{package}/index.html
│   ├── packages/{package}/{filename}
│   └── metadata/{package}/metadata.json
└── rubygems/
    ├── versions
    ├── info/{gemname}
    ├── names
    └── gems/{gemname}-{version}.gem
```

□□ Streaming Architecture

All responses from `SmartRegistry.handleRequest()` use the **Web Streams API**. The `body` field on `IResponse` is always a `ReadableStream<Uint8Array>` — whether the content is a 2GB container image layer or a tiny JSON metadata response.

How It Works

- **Binary downloads** (blobs, tarballs, .crate, .zip, .whl, .gem) stream directly from S3 to the response — zero buffering in memory
- **JSON/metadata responses** are automatically wrapped into a `ReadableStream` at the API boundary
- **OCI chunked uploads** store each PATCH chunk as a temp S3 object instead of accumulating in memory, then stream-assemble during the final PUT with incremental SHA-256 verification

Stream Helpers

```
import { streamToBuffer, streamToJson, toReadableStream } from '@push.rocks/smartregistry';

// Consume a stream into a Buffer
const buffer = await streamToBuffer(response.body);

// Consume a stream into parsed JSON
const data = await streamToJson(response.body);

// Create a ReadableStream from any data type
const stream = toReadableStream({ hello: 'world' });
```

Consuming in Node.js HTTP Servers

Since Node.js `http.ServerResponse` uses Node streams, bridge with `Readable.fromWeb()`:

```
import { Readable } from 'stream';

if (response.body) {
  Readable.fromWeb(response.body).pipe(res);
} else {
  res.end();
}
```

□□ Integration with Express

```
import express from 'express';
import { Readable } from 'stream';
import { SmartRegistry } from '@push.rocks/smartregistry';

const app = express();
const registry = new SmartRegistry(config);
await registry.init();

app.all('*', async (req, res) => {
  const response = await registry.handleRequest({
    method: req.method,
    path: req.path,
    headers: req.headers as Record<string, string>,
    query: req.query as Record<string, string>,
    body: req.body,
  });

  res.status(response.status);
  for (const [key, value] of Object.entries(response.headers)) {
    res.setHeader(key, value);
  }

  if (response.body) {
    // All response bodies are ReadableStream<Uint8Array> — pipe to HTTP response
    Readable.fromWeb(response.body).pipe(res);
  } else {
    res.end();
  }
});

app.listen(5000);
```

□□ Testing with smartstorage

smartregistry works seamlessly with @push.rocks/smartstorage, a local S3-compatible server for testing — no cloud credentials needed.

```
import { SmartStorage } from '@push.rocks/smartstorage';
import { SmartRegistry } from '@push.rocks/smartregistry';

// Start local S3 server
const s3Server = await SmartStorage.createAndStart({
  server: { port: 3456, silent: true },
  storage: { cleanSlate: true },
});

// Get S3 descriptor from the running server
const s3Descriptor = await s3Server.getStorageDescriptor();

const registry = new SmartRegistry({
  storage: { ...s3Descriptor, bucketName: 'my-test-registry' },
  auth: { jwtSecret: 'test', tokenStore: 'memory', npmTokens: { enabled: true } },
  npm: { enabled: true, basePath: '/npm' },
  oci: { enabled: true, basePath: '/oci' },
});
await registry.init();

// ... run your tests ...
await s3Server.stop();
```

□□ Development

```
pnpm install       # Install dependencies
pnpm run build     # Build
pnpm test          # Run all tests
```

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 11:13:40 UTC by foss.global Team
Updated 2026-03-28 12:20:26 UTC by foss.global Team