readme.md for @push.rocks/qenv

Smart Environment Variable Management for Node.js

> Never hardcode secrets again. Load environment variables from multiple sources with ease and confidence.

Features

 Multi-source Loading - Automatically loads from environment variables, config files, and Docker secrets
 Type-Safe - Full TypeScript support with comprehensive type definitions
 Flexible Formats - Supports `.yml`, `.yaml`, and `.json` configuration files
 Docker Ready - Built-in support for Docker secrets and secret.json files
 Async & Sync - Both synchronous and asynchronous variable retrieval
 Strict Mode - Optional strict mode that throws errors for missing variables
 Base64 Objects - Handle complex configuration objects with automatic encoding/decoding
 Dynamic Resolution - Support for async functions as environment variable sources

Installation

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

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

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

□□ Quick Start

```
import { Qenv } from '@push.rocks/qenv';

// Create a new Qenv instance
const qenv = new Qenv('./', './', true);

// Access environment variables
const dbHost = await qenv.getEnvVarOnDemand('DB_HOST');
const apiKey = await qenv.getEnvVarOnDemand('API_KEY');

// Use strict mode to ensure variables exist
const criticalVar = await qenv.getEnvVarOnDemandStrict('CRITICAL_CONFIG');
// Throws error if CRITICAL_CONFIG is not set!
```

□□ Configuration

Setting Up Your Environment Files

1. Define Required Variables (`qenv.yml`)

Create a `qenv.yml` file to specify which environment variables your application needs:

```
required:
  - DB_HOST
  - DB_USER
  - DB_PASSWORD
  - API_KEY
  - LOG_LEVEL
```

2. Provide Values (`env.yml` or `env.json`)

For local development, create an `env.yml` or `env.json` file:

env.yml:

```
DB_HOST: localhost
DB_USER: developer
DB_PASSWORD: supersecret123
API_KEY: dev-key-12345
LOG_LEVEL: debug
```

env.json:

```
{
  "DB_HOST": "localhost",
  "DB_USER": "developer",
  "DB_PASSWORD": "supersecret123",
  "API_KEY": "dev-key-12345",
  "LOG_LEVEL": "debug"
}
```

> □□ **Pro Tip:** Add `env.yml` and `env.json` to your `.gitignore` to keep secrets out of version control!

□□ Advanced Usage

Loading Priority

Qenv loads variables in this order (first found wins):

1. **Process environment variables** - Already set in `process.env`
2. **Configuration files** - From `env.yml` or `env.json`
3. **Docker secrets** - From `/run/secrets/`
4. **Docker secret JSON** - From `/run/secrets/secret.json`

Handling Complex Objects

Store and retrieve complex configuration objects:

```
# In env.yml
DATABASE_CONFIG:
```

```
  database:
    host: localhost
    port: 5432
    options:
      ssl: true
      poolSize: 10

// Qenv automatically handles base64 encoding
const dbConfig = await qenv.getEnvVarOnDemandAsObject('DATABASE_CONFIG');
console.log(dbConfig.database.options.poolSize); // 10
```

Dynamic Environment Variables

Load variables from external sources dynamically:

```
const qenv = new Qenv();

// Define an async function to fetch configuration
const fetchFromVault = async () => {
  const response = await fetch('https://vault.example.com/api/secret');
  const data = await response.json();
  return data.secret;
};

// Use the function as an environment variable source
const secret = await qenv.getEnvVarOnDemand(fetchFromVault);
```

Working with Docker

Qenv seamlessly integrates with Docker secrets:

```
# docker-compose.yml
version: '3.7'
services:
  app:
    image: your-app
    secrets:
      - db_password
```

```
      - api_key

secrets:
  db_password:
    external: true
  api_key:
    external: true
```

Your application automatically reads from `/run/secrets/`:

```
const qenv = new Qenv();
// Automatically loads from /run/secrets/db_password
const dbPassword = await qenv.getEnvVarOnDemand('db_password');
```

Handling Missing Variables

Control how your application handles missing environment variables:

```
// Fail fast (default behavior)
const qenvStrict = new Qenv('./', './', true);
// Application exits if required variables are missing

// Graceful handling
const qenvRelaxed = new Qenv('./', './', false);
// Application continues, you handle missing variables

// Check what's missing
if (qenvRelaxed.missingEnvVars.length > 0) {
  console.warn('Missing variables:', qenvRelaxed.missingEnvVars);
  // Implement fallback logic
}
```

Strict Mode for Critical Variables

Use the new strict getter when you absolutely need a variable:

```
try {
  // This will throw if TOKEN is not set
```

```
  const token = await qenv.getEnvVarOnDemandStrict('TOKEN');

  // You can also check multiple fallback names
  const db = await qenv.getEnvVarOnDemandStrict(['DATABASE_URL', 'DB_CONNECTION']);
} catch (error) {
  console.error('Critical configuration missing:', error.message);
  process.exit(1);
}
```

□□ CI/CD Integration

GitHub Actions

```
name: Deploy
on: [push]
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Deploy with secrets
        env:
          API_KEY: ${{ secrets.API_KEY }}
          DB_PASSWORD: ${{ secrets.DB_PASSWORD }}
        run: |
          npm install
          npm run deploy
```

GitLab CI

```
deploy:
  stage: deploy
  script:
    - npm install
    - npm run deploy
```

```
  variables:
    API_KEY: $CI_API_KEY
    DB_PASSWORD: $CI_DB_PASSWORD
```

□□ Testing

For testing, create a separate `test/assets/env.yml`:

```
import { Qenv } from '@push.rocks/qenv';

describe('MyApp', () => {
  let qenv: Qenv;

  beforeEach(() => {
    qenv = new Qenv('./test/assets', './test/assets', false);
  });

  it('should load test configuration', async () => {
    const testVar = await qenv.getEnvVarOnDemand('TEST_VAR');
    expect(testVar).toBe('test-value');
  });
});
```

□□ Debugging

Enable detailed logging to troubleshoot environment variable loading:

```
const qenv = new Qenv();

// Check what's loaded
console.log('Required vars:', qenv.requiredEnvVars);
console.log('Available vars:', qenv.availableEnvVars);
console.log('Missing vars:', qenv.missingEnvVars);

// Access the logger
qenv.logger.log('info', 'Custom log message');
```

□□ Real-World Example

Here's how you might use qenv in a production Node.js application:

```
import { Qenv } from '@push.rocks/qenv';
import { createServer } from './server';
import { connectDatabase } from './database';

async function bootstrap() {
  // Initialize environment
  const qenv = new Qenv();

  // Load critical configuration
  const config = {
    port: await qenv.getEnvVarOnDemand('PORT') || '3000',
    dbUrl: await qenv.getEnvVarOnDemandStrict('DATABASE_URL'),
    apiKey: await qenv.getEnvVarOnDemandStrict('API_KEY'),
    logLevel: await qenv.getEnvVarOnDemand('LOG_LEVEL') || 'info',
    features: await qenv.getEnvVarOnDemandAsObject('FEATURE_FLAGS')
  };

  // Connect to database
  await connectDatabase(config.dbUrl);

  // Start server
  const server = createServer(config);
  server.listen(config.port, () => {
    console.log(`□□Server running on port ${config.port}`);
  });
}

bootstrap().catch(error => {
  console.error('Failed to start application:', error);
  process.exit(1);
});
```

□□ API Reference

Class: `Qenv`

Constructor

```
new Qenv(
  qenvFileBasePathArg?: string,  // Path to qenv.yml (default: process.cwd())
  envFileBasePathArg?: string,   // Path to env.yml/json (default: same as qenv)
  failOnMissing?: boolean        // Exit on missing vars (default: true)
)
```

Methods

Method	Description	Returns
`getEnvVarOnDemand(name)`	Get environment variable value	`Promise<string | undefined>`
`getEnvVarOnDemandStrict(name)`	Get variable or throw error	`Promise<string>`
`getEnvVarOnDemandSync(name)`	Synchronously get variable	`string | undefined`
`getEnvVarOnDemandAsObject(name)`	Get variable as decoded object	`Promise<any>`

Properties

Property	Type	Description
`requiredEnvVars`	`string[]`	List of required variable names
`availableEnvVars`	`string[]`	List of found variable names
`missingEnvVars`	`string[]`	List of missing variable names
`keyValueObject`	`object`	All loaded variables as key-value pairs

License and Legal Information

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 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, and any usage must be approved in writing by Task Venture Capital GmbH.

Company Information

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

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

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

Revision #3
Created 2026-03-28 11:09:56 UTC by foss.global Team
Updated 2026-03-28 12:16:33 UTC by foss.global Team