readme.md for @push.rocks/smartdaemon

@push.rocks/smartdaemon □□

Turn your Node.js scripts into production-ready system daemons

npm version License: MIT

> Seamlessly manage long-running processes, background services, and system daemons with smart root detection, automatic privilege escalation, and systemd integration.

□□ What is SmartDaemon?

`@push.rocks/smartdaemon` is your Swiss Army knife for turning Node.js applications into bulletproof system services. Whether you're deploying microservices, running scheduled tasks, or managing background workers, SmartDaemon handles the complex stuff so you can focus on your business logic.

□ Key Features

- □□ **Smart Privilege Management**: Automatically detects root status and handles sudo operations seamlessly
- □□ **User/Group Control**: Run services under specific users for enhanced security
- □□ **Systemd Integration**: Full Linux systemd support for production-grade service management

- □□ **Auto-Recovery**: Built-in service restart on failure
- □□ **Declarative Service Definitions**: Simple, clear service configuration
- □□ **Interactive & Non-Interactive Modes**: Support for both passwordless sudo and interactive authentication
- □□ **Complete Lifecycle Management**: Start, stop, enable, disable, and reload services with ease

□□ Installation

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

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

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

□□ Quick Start

Basic Usage

```
import { SmartDaemon } from '@push.rocks/smartdaemon';

// Initialize SmartDaemon
const daemon = new SmartDaemon();

// Create and enable a service
const myService = await daemon.addService({
  name: 'my-api-server',
  description: 'My awesome API server',
  command: 'node server.js',
  workingDir: '/opt/myapp',
  version: '1.0.0'
});
```

```
// Enable and start the service
await myService.enable(); // Creates systemd service file
await myService.start();  // Starts the service
```

Running Services as Specific Users □□

Security best practice: Never run services as root unless absolutely necessary!

```
const webService = await daemon.addService({
  name: 'web-server',
  description: 'Production web server',
  command: 'node app.js',
  workingDir: '/var/www/app',
  version: '2.1.0',
  user: 'www-data',    // Run as www-data user
  group: 'www-data'    // Run under www-data group
});

await webService.enable();
await webService.start();
```

□□ Smart Privilege Escalation

SmartDaemon intelligently handles privilege escalation based on your execution context:

Automatic Root Detection

```
// SmartDaemon automatically detects if running as root
const daemon = new SmartDaemon();

// If running as regular user, sudo will be used automatically
// If running as root, commands execute directly
```

With Sudo Password

If you're not running as root and don't have passwordless sudo configured:

```
const daemon = new SmartDaemon({
  sudoPassword: 'your-sudo-password'  // Will be used for privilege escalation
});

// All privileged operations will now use the provided password
const service = await daemon.addService({
  name: 'privileged-service',
  description: 'Service requiring root privileges',
  command: 'node admin-task.js',
  workingDir: '/opt/admin',
  version: '1.0.0'
});

await service.enable(); // Automatically uses sudo with password
```

Passwordless Sudo (Recommended for Production)

For production environments, configure passwordless sudo for specific systemctl commands:

1. Create a sudoers file: `/etc/sudoers.d/smartdaemon`
2. Add the following content:

```
# Allow smartdaemon to manage services without password
yourusername ALL=(ALL) NOPASSWD: /usr/bin/systemctl start smartdaemon_*
yourusername ALL=(ALL) NOPASSWD: /usr/bin/systemctl stop smartdaemon_*
yourusername ALL=(ALL) NOPASSWD: /usr/bin/systemctl enable smartdaemon_*
yourusername ALL=(ALL) NOPASSWD: /usr/bin/systemctl disable smartdaemon_*
yourusername ALL=(ALL) NOPASSWD: /usr/bin/systemctl daemon-reload
```

□□ Complete API

SmartDaemon Class

```
interface ISmartDaemonOptions {
  sudoPassword?: string;  // Optional sudo password for privilege escalation
}

class SmartDaemon {
  constructor(options?: ISmartDaemonOptions);

  // Add a new service or update existing one
  async addService(options: ISmartDaemonServiceOptions): Promise<SmartDaemonService>;

  // Direct access to managers (advanced usage)
  systemdManager: SmartDaemonSystemdManager;
  templateManager: SmartDaemonTemplateManager;
}
```

SmartDaemonService Class

```
interface ISmartDaemonServiceOptions {
  name: string;           // Unique service identifier
  description: string;    // Human-readable description
  command: string;        // Command to execute
  workingDir: string;     // Working directory (absolute path)
  version: string;        // Service version
  user?: string;          // User to run service as (optional)
  group?: string;         // Group to run service as (optional)
}

class SmartDaemonService {
  // Lifecycle management
  async enable(): Promise<void>;   // Install and enable service
  async disable(): Promise<void>;  // Disable service
  async start(): Promise<void>;    // Start service
  async stop(): Promise<void>;     // Stop service

  // Service management
  async save(): Promise<void>;     // Save service configuration
  async delete(): Promise<void>;   // Remove service
  async reload(): Promise<void>;   // Reload systemd daemon
```

```
}
```

□□ Real-World Examples

Microservice Deployment

```
import { SmartDaemon } from '@push.rocks/smartdaemon';

async function deployMicroservices() {
  const daemon = new SmartDaemon();

  // API Gateway
  const apiGateway = await daemon.addService({
    name: 'api-gateway',
    description: 'API Gateway Service',
    command: 'node --max-old-space-size=2048 gateway.js',
    workingDir: '/opt/services/gateway',
    version: '3.2.1',
    user: 'apiuser',
    group: 'apigroup'
  });

  // Auth Service
  const authService = await daemon.addService({
    name: 'auth-service',
    description: 'Authentication Service',
    command: 'node auth-service.js',
    workingDir: '/opt/services/auth',
    version: '2.1.0',
    user: 'authuser',
    group: 'authgroup'
  });

  // Database Sync Worker
  const dbWorker = await daemon.addService({
    name: 'db-sync-worker',
    description: 'Database Synchronization Worker',
```

```
    command: 'node workers/db-sync.js',
    workingDir: '/opt/services/workers',
    version: '1.5.3',
    user: 'dbworker',
    group: 'dbgroup'
  });

  // Enable and start all services
  const services = [apiGateway, authService, dbWorker];

  for (const service of services) {
    await service.enable();
    await service.start();
    console.log(`□ Service ${service.name} is running`);
  }
}

deployMicroservices();
```

Scheduled Task Runner

```
async function setupScheduledTasks() {
  const daemon = new SmartDaemon();

  // Backup service that runs every night
  const backupService = await daemon.addService({
    name: 'nightly-backup',
    description: 'Nightly database backup service',
    command: 'node --require dotenv/config backup-runner.js',
    workingDir: '/opt/backup',
    version: '1.0.0',
    user: 'backup',
    group: 'backup'
  });

  await backupService.enable();
  await backupService.start();
}
```

Development vs Production Setup

```
import { SmartDaemon } from '@push.rocks/smartdaemon';

async function setupService(environment: 'development' | 'production') {
  let daemonOptions = {};

  if (environment === 'development') {
    // In development, you might use sudo password
    daemonOptions = {
      sudoPassword: process.env.SUDO_PASSWORD
    };
  }
  // In production, rely on passwordless sudo configuration

  const daemon = new SmartDaemon(daemonOptions);

  const service = await daemon.addService({
    name: `app-${environment}`,
    description: `Application ${environment} server`,
    command: environment === 'production'
      ? 'node --production app.js'
      : 'node --inspect app.js',
    workingDir: '/opt/application',
    version: '1.0.0',
    user: environment === 'production' ? 'appuser' : process.env.USER,
    group: environment === 'production' ? 'appgroup' : process.env.USER
  });

  await service.enable();
  await service.start();
}
```

□□ Advanced Usage

Direct SystemdManager Access

For advanced users who need fine-grained control:

```
const daemon = new SmartDaemon();

// Access the SystemdManager directly
const systemdManager = daemon.systemdManager;

// Check if running on a compatible system
const canRun = await systemdManager.checkElegibility();

// Get all existing SmartDaemon services
const existingServices = await systemdManager.getServices();

// Reload systemd daemon
await systemdManager.reload();
```

Custom Service Templates

Access the template manager for custom service file generation:

```
const daemon = new SmartDaemon();
const template = daemon.templateManager.generateUnitFileForService(myService);
console.log(template); // View generated systemd unit file
```

□□ System Requirements

- **Operating System**: Linux with systemd (Ubuntu 16.04+, Debian 9+, CentOS 7+, RHEL 7+, Fedora, Arch, etc.)
- **Node.js**: Version 14.x or higher
- **Permissions**: Either root access or properly configured sudo

⚙ Generated Systemd Service Files

SmartDaemon generates professional systemd unit files with:

- Automatic restart on failure
- Network target dependencies
- Proper working directory configuration
- User/Group assignment
- Resource limits configuration
- Syslog integration for logging
- Environment variable support

Example generated service file:

```
[Unit]
Description=My API Server
Requires=network.target
After=network.target

[Service]
Type=simple
User=www-data
Group=www-data
Environment=NODE_OPTIONS="--max_old_space_size=100"
ExecStart=/bin/bash -c "cd /opt/myapp && node server.js"
WorkingDirectory=/opt/myapp
Restart=always
RestartSec=10
LimitNOFILE=infinity
LimitCORE=infinity
StandardOutput=syslog
StandardError=syslog

[Install]
WantedBy=multi-user.target
```

□□ Troubleshooting

Permission Denied Errors

If you encounter "Interactive authentication required" errors:

1. **Option 1**: Run your application with sudo

   ```
   sudo node your-app.js
   ```

2. **Option 2**: Configure passwordless sudo (recommended for production)
3. **Option 3**: Provide sudo password programmatically

   ```
   const daemon = new SmartDaemon({ sudoPassword: 'your-password' });
   ```

Service Not Found

If systemctl can't find your service:

- Check that the service was enabled: `await service.enable()`
- Verify the service name: Services are prefixed with `smartdaemon_`
- Reload systemd: `await service.reload()`

Service Fails to Start

Check logs using:

```
journalctl -u smartdaemon_your-service-name -f
```

□□ Security Best Practices

1. **Never run services as root** unless absolutely necessary
2. **Use dedicated service users** with minimal permissions
3. **Configure passwordless sudo** for production instead of storing passwords
4. **Limit sudo permissions** to only the specific systemctl commands needed
5. **Regular security audits** of your service configurations

□□ Support

- □□ Email: hello@task.vc
- □□ Issues: GitHub Issues
- □□ Documentation: https://code.foss.global/push.rocks/smartdaemon

License and Legal Information

This repository contains open-source code that is licensed under the MIT License. A copy of the MIT License can be found in the license file within this repository.

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 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 #2
Created 2026-03-28 11:10:29 UTC by foss.global Team
Updated 2026-03-28 11:39:18 UTC by foss.global Team