@serve.zone/spark

The systemd service that monitors and keeps a node online

- readme.md for @serve.zone/spark
- changelog.md for @serve.zone/spark

readme.md for @serve.zone/spark

> **A powerful Deno-powered server management tool for the modern infrastructure**

Spark is a comprehensive tool for maintaining and configuring servers at the OS level, with deep Docker integration and advanced task scheduling capabilities. Built for the serve.zone infrastructure, Spark serves as the backbone for @serve.zone/cloudly cluster management, handling everything from daemon orchestration to container lifecycle management.

Features

- **Standalone Binary** - No runtime dependencies, just download and run
- **Docker Integration** - Native support for Docker services, stacks, secrets, and networks
- ⚙ **Daemon Management** - Systemd integration for reliable service operation
- **Task Scheduling** - Cron-like task scheduling for automation
- **Auto-Updates** - Self-updating capabilities for zero-downtime deployments
- **Secure Secrets** - Docker secrets management for sensitive data
- **Comprehensive Logging** - Built-in logging with multiple severity levels
- **Mode Support** - Cloudly and CoreFlow node operation modes

Installation

Quick Install (Recommended)

Install the latest version via our installation script:

```
curl -sSL https://code.foss.global/serve.zone/spark/raw/branch/master/install.sh | sudo bash
```

npm Install

Install via npm (automatically downloads the correct binary for your platform):

```
npm install -g @serve.zone/spark
```

Specific Version

```
curl -sSL https://code.foss.global/serve.zone/spark/raw/branch/master/install.sh | sudo bash -s -- --version v1.2.4
```

Manual Installation

Download the binary for your platform from the releases page and make it executable:

```
# Example for Linux x64
wget https://code.foss.global/serve.zone/spark/releases/download/v1.2.4/spark-linux-x64
chmod +x spark-linux-x64
sudo mv spark-linux-x64 /usr/local/bin/spark
```

Supported Platforms

- □□ Linux (x86_64, ARM64)
- □□ macOS (Intel, Apple Silicon)
- □□ Windows (x86_64)

□□ Quick Start

Install as System Daemon

Set up Spark to run as a systemd service:

```
sudo spark installdaemon
```

This command:

- Creates a systemd service unit
- Enables automatic startup on boot
- Starts the Spark daemon immediately

Configure Operation Mode

Spark supports different operation modes for various use cases:

```
# For Cloudly cluster management
sudo spark asdaemon --mode cloudly

# For CoreFlow node management
sudo spark asdaemon --mode coreflow-node
```

View Logs

Monitor Spark daemon activity in real-time:

```
sudo spark logs
```

□□ CLI Reference

Core Commands

`spark installdaemon`

Installs Spark as a system daemon service. This sets up a systemd unit that automatically starts on boot.

```
sudo spark installdaemon
```

`spark updatedaemon`

Updates the daemon service configuration to the current Spark version.

```
sudo spark updatedaemon
```

`spark asdaemon [--mode MODE]`

Runs Spark in daemon mode. Requires a mode to be specified (either via `--mode` flag or from saved configuration).

```
sudo spark asdaemon --mode cloudly
```

Available modes:

- `cloudly` - Manages Cloudly services
- `coreflow-node` - Manages CoreFlow node services

`spark logs`

Displays real-time logs from the Spark daemon service.

```
sudo spark logs
```

`spark prune`

Performs a complete cleanup of Docker resources and restarts services. Use with caution!

```
sudo spark prune
```

This command:

1. Stops the Spark daemon
2. Removes all Docker stacks
3. Removes all Docker services
4. Removes all Docker secrets
5. Removes specified Docker networks
6. Prunes the Docker system
7. Restarts Docker
8. Restarts the Spark daemon

Advanced Usage

Check Version

```
spark --version
```

Get Help

```
spark help
```

□□ Programmatic Usage

While Spark is primarily designed as a CLI tool and daemon, you can also use it as a library in your Deno projects.

Import from Deno

```
import { Spark } from 'https://code.foss.global/serve.zone/spark/raw/branch/master/mod.ts';
```

Basic Usage

```
import { Spark } from './mod.ts';

// Create a Spark instance
const spark = new Spark();

// Start the daemon programmatically
await spark.daemonStart();
```

Task Scheduling

Schedule automated tasks using the built-in task manager:

```
import { Spark } from './mod.ts';

const spark = new Spark();

// Define a custom task
const backupTask = {
  name: 'daily-backup',
  taskFunction: async () => {
    console.log('Running backup...');
    // Your backup logic here
  },
};
```

```
// Schedule it to run daily at 2 AM
spark.sparkTaskManager.taskmanager.addAndScheduleTask(
  backupTask,
  '0 2 * * *'
);
```

Service Management

Manage Docker services programmatically:

```
import { Spark } from './mod.ts';

const spark = new Spark();

// Add a service to manage
spark.sparkUpdateManager.services.push({
  name: 'my-app',
  image: 'code.foss.global/myorg/myapp',
  url: 'myapp',
  environment: 'production',
  port: '3000',
  secretJson: {
    API_KEY: 'secret-value',
    DATABASE_URL: 'postgresql://...',
  },
});

// Start managing services
await spark.sparkUpdateManager.start();
```

Configuration Management

Access and modify Spark's configuration:

```
import { Spark } from './mod.ts';

const spark = new Spark();
```

```
// Write configuration
await spark.sparkConfig.kvStore.writeKey('mode', 'cloudly');

// Read configuration
const mode = await spark.sparkConfig.kvStore.readKey('mode');
console.log(`Current mode: ${mode}`);
```

Logging

Use Spark's built-in logger for consistent logging:

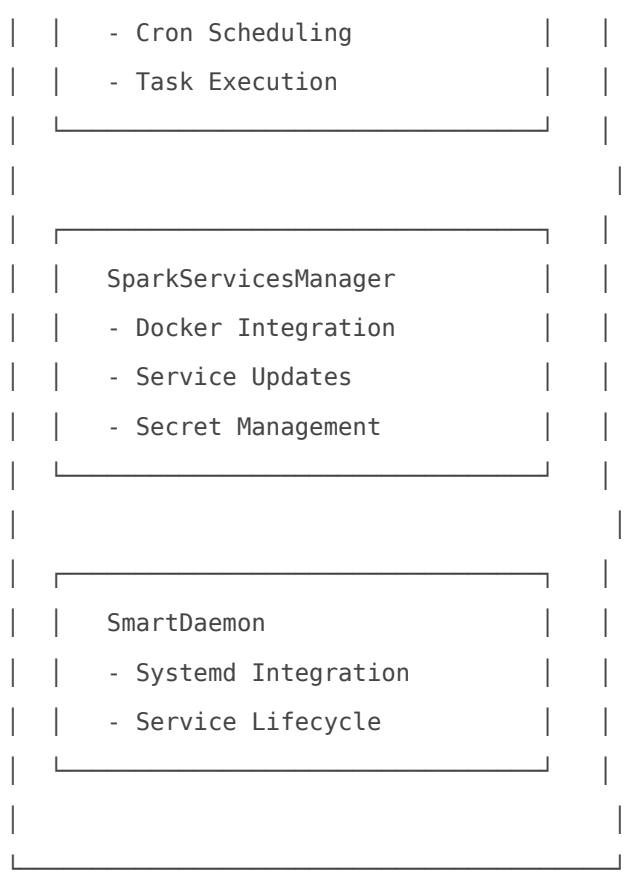
```
import { logger } from './ts/spark.logging.ts';

// Log at different levels
logger.log('info', 'Application starting...');
logger.log('ok', 'Service deployed successfully');
logger.log('warn', 'High memory usage detected');
logger.log('error', 'Failed to connect to database');
logger.log('success', 'Backup completed');
```

□□ Architecture

Core Components

```
┌─────────────────────────────────────────┐
│           Spark Instance                │
├─────────────────────────────────────────┤
│                                         │
│  ┌──────────────────────────────┐   │
│  │   SparkConfig                │   │
│  │   - KV Store                 │   │
│  │   - Mode Configuration       │   │
│  └──────────────────────────────┘   │
│                                        │
│  ┌──────────────────────────────┐   │
│  │   SparkTaskManager           │   │
```



Key Classes

- `Spark` - Main orchestrator class that coordinates all components
- `SparkConfig` - Handles configuration storage and retrieval
- `SparkTaskManager` - Manages scheduled tasks and automation
- `SparkServicesManager` - Manages Docker services and updates
- `SparkInfo` - Provides project and version information

□□ Update Management

Spark includes self-updating capabilities:

```
import { Spark } from './mod.ts';

const spark = new Spark();

// Check for and apply updates
await spark.sparkUpdateManager.updateServices();
```

The update manager:

- Pulls latest Docker images
- Manages service rollouts
- Handles zero-downtime deployments
- Manages Docker secrets securely

□□ Docker Integration

Service Definition

```
const serviceDefinition = {
  name: 'api-server',
  image: 'code.foss.global/myorg/api',
  url: 'api',
  environment: 'production',
  port: '8080',
  secretJson: {
    JWT_SECRET: 'your-jwt-secret',
    DB_PASSWORD: 'your-db-password',
  },
};

spark.sparkUpdateManager.services.push(serviceDefinition);
```

Stack Management

Spark manages Docker stacks for complex multi-service deployments:

```
# View running stacks
docker stack ls

# Spark manages these automatically
```

□□ Development

Prerequisites

- Deno v2.x or later

Running from Source

```
# Clone the repository
git clone https://code.foss.global/serve.zone/spark.git
cd spark

# Run directly
deno run --allow-all mod.ts

# Run tests
deno test --allow-all test/

# Type check
deno check mod.ts

# Format code
deno fmt

# Lint
deno lint
```

Building Binaries

Compile for all supported platforms:

```
bash scripts/compile-all.sh
```

Binaries will be output to `dist/binaries/`.

Compile for Specific Platform

```
# Linux x64
deno compile --allow-all --output spark-linux-x64 --target x86_64-unknown-linux-gnu mod.ts

# macOS ARM64
deno compile --allow-all --output spark-macos-arm64 --target aarch64-apple-darwin mod.ts
```

□□ Security

Permissions

Spark requires the following Deno permissions:

- `--allow-net` - API communication, Docker socket access
- `--allow-read` - Configuration files, project files
- `--allow-write` - Logs, configuration updates
- `--allow-run` - systemctl, Docker commands
- `--allow-env` - Environment variables
- `--allow-sys` - System information

Secrets Management

Always use Docker secrets for sensitive data:

```
const serviceWithSecrets = {
  name: 'secure-app',
  image: 'myapp:latest',
  secretJson: {
    API_KEY: Deno.env.get('API_KEY')!,
    DB_PASSWORD: Deno.env.get('DB_PASSWORD')!,
  },
};
```

□□ Troubleshooting

Service Won't Start

Check the daemon status:

```
sudo systemctl status smartdaemon_spark
```

View recent logs:

```
sudo journalctl -u smartdaemon_spark -n 100
```

Docker Issues

Verify Docker is running:

```
sudo systemctl status docker
```

Check Docker socket permissions:

```
sudo ls -la /var/run/docker.sock
```

Configuration Issues

Check current mode:

```
# Run spark programmatically
deno run --allow-all -e "
import { Spark } from './mod.ts';
const s = new Spark();
const mode = await s.sparkConfig.kvStore.readKey('mode');
console.log('Mode:', mode);
"
```

□□ Examples

Automated System Maintenance

```
import { Spark } from './mod.ts';
```

```
const spark = new Spark();

// Schedule weekly system updates
const updateTask = {
  name: 'system-update',
  taskFunction: async () => {
    const shell = new Deno.Command('bash', {
      args: ['-c', 'apt-get update && apt-get upgrade -y'],
    });
    await shell.output();
  },
};

// Every Sunday at 3 AM
spark.sparkTaskManager.taskmanager.addAndScheduleTask(
  updateTask,
  '0 3 * * 0'
);

await spark.daemonStart();
```

Multi-Service Deployment

```
import { Spark } from './mod.ts';

const spark = new Spark();

// Add multiple services
const services = [
  {
    name: 'frontend',
    image: 'code.foss.global/myorg/frontend',
    url: 'frontend',
    port: '80',
    environment: 'production',
  },
  {
    name: 'backend',
```

```
    image: 'code.foss.global/myorg/backend',
    url: 'backend',
    port: '3000',
    environment: 'production',
    secretJson: {
      DATABASE_URL: Deno.env.get('DATABASE_URL')!,
    },
  },
  {
    name: 'worker',
    image: 'code.foss.global/myorg/worker',
    url: 'worker',
    environment: 'production',
  },
];

services.forEach(svc => spark.sparkUpdateManager.services.push(svc));

await spark.daemonStart();
```

□□ Support

For issues, questions, or contributions:

- □□ Report Issues
- □□ View Source

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.

changelog.md for @serve.zone/spark

2026-02-04 - 1.2.5 - fix(deps)

update Docker API usage to DockerHost facade, bump dependencies, and adjust tests/docs

- Bumped @serve.zone/api and @serve.zone/interfaces to ^5.3.0 and @apiclient.xyz/docker to ^5.1.0 in deno.json
- Replaced plugins.docker.* calls with dockerHost facade methods (getServiceByName, getSecretByName, createImageFromRegistry → createImageFromRegistry, createSecret, createService) in updatemanager and added conditional secret removal to avoid errors
- Updated a number of @push.rocks package versions in deno.json
- Adjusted test/test.ts to use the Deno.test object form and disabled sanitizeResources/sanitizeOps to accommodate smartdaemon signal listeners
- Bumped install example version in readme to v1.2.4 and cleaned up project Serena metadata files

2025-10-23 - 1.2.4 - fix(deno)

Update deno configuration and add local Claude settings

- deno.json: add exports pointing to ./mod.ts
- deno.json: enable nodeModulesDir = "auto" to improve compatibility with Node-style dependencies
- Add .claude/settings.local.json for local environment & tooling configuration

2025-10-23 - 1.2.3 - fix(package)

Add .claude local settings with development permissions

- Add .claude/settings.local.json to define local development permissions for Claude tooling
- No runtime or library code modified; this is a local configuration file

- Patch bump recommended

2024-12-20 - 1.2.2 - fix(core)

Refactored configuration management classes and improved service update handling

- Replaced SparkLocalConfig with SparkConfig for configuration management.
- Improved service handling and update check logic.
- Consolidated service definition and update logic for better maintainability.

2024-12-19 - 1.2.1 - fix(taskmanager)

Remove checkinSlackTask from SparkTaskManager for streamlined task management

- checkinSlackTask has been removed from the task manager class.
- Removal of the slack check-in task allows the system to focus on essential update tasks.

2024-12-18 - 1.2.0 - feat(core)

Initial commit of the Spark project with core functionalities for server management and integration with Docker.

- Add core functionalities for server maintenance and configuration.
- Integrate Docker for advanced task scheduling and service management.
- Provide CLI commands for daemon management and task execution.

2024-12-18 - 1.1.0 - feat(core)

Update package dependencies and improve API integration.

- Updated devDependencies and dependencies in package.json.
- Integrated new package @serve.zone/api.
- Updated identityArg in SparkLocalConfig for userHomeDir kvStore.

2024-06-13 - 1.0.85 to 1.0.90 - Core Updates

Routine updates and fixes to core functionality.

- Updated core component throughout versions for enhanced stability
- Incremental improvements applied on versions 1.0.85 to 1.0.90

2024-05-08 - 1.0.82 to 1.0.85 - Core Enhancements

Consistent updates made to improve core operations.

- Updates focused on core functionality for improved performance
- Series of updates applied from versions 1.0.82 to 1.0.85