readme.md for @push.rocks/smartnftables

A TypeScript module for managing Linux nftables rules with a high-level, type-safe API. Handles NAT (DNAT/SNAT/masquerade), firewall rules, IP sets, and rate limiting — all from clean, declarative TypeScript.

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.

Install

```
pnpm install @push.rocks/smartnftables
# or
npm install @push.rocks/smartnftables
```

> ⚠ **Requires root privileges** to actually apply nftables rules to the kernel. Without root, rules are tracked in memory but not applied (a warning is logged). Great for development/testing!

Quick Start

```
import { SmartNftables } from '@push.rocks/smartnftables';
```

```
const nft = new SmartNftables();
await nft.initialize();

// Port forward 8080 → 192.168.1.100:80
await nft.nat.addPortForwarding('web', {
  sourcePort: 8080,
  targetHost: '192.168.1.100',
  targetPort: 80,
});

// Block a suspicious IP
await nft.firewall.blockIP('10.0.0.99');

// Rate limit HTTP to 100 req/s per IP
await nft.rateLimit.addRateLimit('http-limit', {
  port: 80,
  protocol: 'tcp',
  rate: '100/second',
  perSourceIP: true,
});

// Clean up everything when done
await nft.cleanup();
```

Architecture □□

The library is organized around a **facade pattern** with specialized sub-managers:

```
SmartNftables (main facade)
├── nat          → NatManager       (DNAT, SNAT, masquerade)
├── firewall     → FirewallManager  (filter rules, IP sets, stateful tracking)
└── rateLimit    → RateLimitManager (packet/connection rate limiting)
```

All rules are tracked in **rule groups** identified by string IDs, so you can add, inspect, and remove them programmatically.

API Reference

`SmartNftables` — Main Facade

```
const nft = new SmartNftables({
  tableName: 'smartnftables', // nftables table name (default: 'smartnftables')
  family: 'ip',               // 'ip' | 'ip6' | 'inet' (default: 'ip')
  dryRun: false,               // generate commands without executing (default: false)
});
```

Method	Description
`initialize()`	Create the nftables table and NAT chains. Idempotent.
`cleanup()`	Delete the entire table and clear all tracking.
`status()`	Get an `INftStatus` report of the current managed state.
`applyRuleGroup(id, commands)`	Apply and track a group of raw nft commands.
`removeRuleGroup(id)`	Remove a tracked rule group.
`getRuleGroup(id)`	Retrieve a tracked rule group by ID.

□□ NAT — `nft.nat`

Port Forwarding (DNAT)

```
await nft.nat.addPortForwarding('my-service', {
  sourcePort: 443,
  targetHost: '10.0.0.5',
  targetPort: 8443,
  protocol: 'tcp',           // 'tcp' | 'udp' | 'both' (default: 'tcp')
  preserveSourceIP: false,    // skip masquerade if true (default: false)
});

await nft.nat.removePortForwarding('my-service');
```

Port Range Forwarding

Map a range of ports to another host:

```
// Forward ports 3000-3010 → 10.0.0.5:3000-3010
await nft.nat.addPortRange('dev-ports', 3000, 3010, '10.0.0.5', 3000, 'tcp');
```

```
await nft.nat.removePortRange('dev-ports');
```

SNAT (Source NAT)

```
await nft.nat.addSnat('egress', {
  sourceAddress: '203.0.113.1',
  targetPort: 80,
  protocol: 'tcp',
});
```

Masquerade

```
await nft.nat.addMasquerade('outbound', {
  targetPort: 443,
  protocol: 'tcp',
});
```

□□ Firewall — `nft.firewall`

Basic Rules

```
await nft.firewall.addRule('allow-ssh', {
  direction: 'input',         // 'input' | 'output' | 'forward'
  action: 'accept',           // 'accept' | 'drop' | 'reject'
  sourceIP: '10.0.0.0/24',
  destPort: 22,
  protocol: 'tcp',
  comment: 'Allow SSH from trusted network',
});

await nft.firewall.removeRule('allow-ssh');
```

Block an IP

```
await nft.firewall.blockIP('10.0.0.99');
await nft.firewall.blockIP('192.168.0.0/16', { direction: 'forward' });
```

Allow Only Specific IPs on a Port

```
// Only these IPs can reach port 3306 — everything else is dropped
await nft.firewall.allowOnlyIPs('db-access', ['10.0.0.1', '10.0.0.2'], 3306, 'tcp');
```

Stateful Connection Tracking

```
// Allow established/related, drop invalid — on the input chain
await nft.firewall.enableStatefulTracking('input');
```

IP Sets

Create named sets and match against them:

```
// Create a set of blocked IPs
await nft.firewall.createIPSet({
  name: 'blocklist',
  type: 'ipv4_addr',
  elements: ['10.0.0.50', '10.0.0.51'],
});

// Dynamically add/remove elements
await nft.firewall.addToIPSet('blocklist', ['10.0.0.52']);
await nft.firewall.removeFromIPSet('blocklist', ['10.0.0.50']);

// Clean up
await nft.firewall.deleteIPSet('blocklist');
```

You can also build set-matching rules directly with the low-level builder:

```
import { buildIPSetMatchRule } from '@push.rocks/smartnftables';

const rule = buildIPSetMatchRule('smartnftables', 'ip', {
  setName: 'blocklist',
  direction: 'input',
  matchField: 'saddr',
  action: 'drop',
});
```

Rate Limiting — `nft.rateLimit`

Packet Rate Limiting

```
// Global: drop packets over 1000/second on port 80
await nft.rateLimit.addRateLimit('http-global', {
  port: 80,
  protocol: 'tcp',
  rate: '1000/second',
  burst: 50,
  action: 'drop',
});

// Per-IP: each source IP gets its own 100/second limit
await nft.rateLimit.addRateLimit('http-per-ip', {
  port: 80,
  protocol: 'tcp',
  rate: '100/second',
  perSourceIP: true,
});

await nft.rateLimit.removeRateLimit('http-per-ip');
```

Connection Rate Limiting

Limit the rate of **new connections** (uses `ct state new`):

```
await nft.rateLimit.addConnectionRateLimit('ssh-connrate', {
  port: 22,
  protocol: 'tcp',
  rate: '5/second',
  perSourceIP: true,
});

await nft.rateLimit.removeConnectionRateLimit('ssh-connrate');
```

□□ Low-Level Rule Builders

For advanced use cases, you can generate raw nft command strings without applying them:

```
import {
  buildDnatRules,
  buildSnatRule,
  buildMasqueradeRule,
  buildFirewallRule,
  buildRateLimitRule,
  buildPerIpRateLimitRule,
  buildConnectionRateRule,
  buildIPSetCreate,
  buildIPSetAddElements,
  buildIPSetRemoveElements,
  buildIPSetDelete,
  buildIPSetMatchRule,
  buildTableSetup,
  buildFilterChains,
  buildTableCleanup,
} from '@push.rocks/smartnftables';

const commands = buildDnatRules('mytable', 'ip', {
  sourcePort: 8080,
  targetHost: '10.0.0.5',
  targetPort: 80,
});
// → ['nft add rule ip mytable prerouting tcp dport 8080 dnat to 10.0.0.5:80',
//    'nft add rule ip mytable postrouting tcp dport 80 masquerade']
```

Dry Run Mode □□

Generate commands without touching the kernel — perfect for testing, debugging, or CI:

```
const nft = new SmartNftables({ dryRun: true });
await nft.initialize();
await nft.nat.addPortForwarding('test', {
  sourcePort: 80,
  targetHost: '10.0.0.1',
  targetPort: 8080,
});
```

```
console.log(nft.status());
// Rules tracked in memory, nothing executed
```

Status Reporting □□

```
const status = nft.status();
// {
//   initialized: true,
//   tableName: 'smartnftables',
//   family: 'ip',
//   isRoot: true,
//   activeGroups: 3,
//   groups: {
//     'nat:web': { ruleCount: 2, createdAt: 1711411200000 },
//     'fw:block-10_0_0_99': { ruleCount: 1, createdAt: 1711411200100 },
//     'ratelimit:http-limit': { ruleCount: 1, createdAt: 1711411200200 },
//   }
// }
```

Types

All interfaces and types are fully exported for use in your own code:

Type	Description
`INftDnatRule`	DNAT port forwarding rule config
`INftSnatRule`	Source NAT rule config
`INftMasqueradeRule`	Masquerade rule config
`INftFirewallRule`	Firewall filter rule config
`INftIPSetConfig`	IP set creation config
`INftRateLimitRule`	Rate limiting rule config
`INftConnectionRateRule`	New-connection rate limit config
`ISmartNftablesOptions`	Constructor options
`INftStatus`	Status report shape
`TNftProtocol`	`'tcp' | 'udp' | 'both'`

Type	Description
`TNftFamily`	`'ip' | 'ip6' | 'inet'`
`TFirewallAction`	`'accept' | 'drop' | 'reject'`
`TCtState`	`'new' | 'established' | 'related' | 'invalid'`

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 11:13:49 UTC by foss.global Team
Updated 2026-03-28 12:20:35 UTC by foss.global Team