@push.rocks/smartproxy

a proxy for handling high workloads of proxying, internally using rust for performance.

- readme.md for @push.rocks/smartproxy
- changelog.md for @push.rocks/smartproxy

readme.md for @push.rocks/smartproxy

A high-performance, Rust-powered proxy toolkit for Node.js — unified route-based configuration for SSL/TLS termination, HTTP/HTTPS reverse proxying, WebSocket support, UDP/QUIC/HTTP3, load balancing, custom protocol handlers, and kernel-level NFTables forwarding via `@push.rocks/smartnftables`.

□□ Installation

```
npm install @push.rocks/smartproxy
# or
pnpm add @push.rocks/smartproxy
```

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.

□□ What is SmartProxy?

SmartProxy is a production-ready proxy solution that takes the complexity out of traffic management. Under the hood, all networking — TCP, UDP, TLS, HTTP reverse proxy, QUIC/HTTP3, connection tracking, security enforcement, and NFTables — is handled by a **Rust engine** for maximum performance, while you configure everything through a clean TypeScript API with full type safety.

Whether you're building microservices, deploying edge infrastructure, proxying UDP-based protocols, or need a battle-tested reverse proxy with automatic Let's Encrypt certificates,

SmartProxy has you covered.

⚡ Key Features

Feature	Description
◫ **Rust-Powered Engine**	All networking handled by a high-performance Rust binary via IPC
◫ **Unified Route-Based Config**	Clean match/action patterns for intuitive traffic routing
◫ **Automatic SSL/TLS**	Zero-config HTTPS with Let's Encrypt ACME integration
◫ **Flexible Matching**	Route by port, domain, path, protocol, client IP, TLS version, headers, or custom logic
◫ **High-Performance**	Choose between user-space or kernel-level (NFTables) forwarding
◫ **UDP & QUIC/HTTP3**	First-class UDP transport, datagram handlers, QUIC tunneling, and HTTP/3 support
⚖ **Load Balancing**	Round-robin, least-connections, IP-hash with health checks
◫ **Enterprise Security**	IP filtering, rate limiting, basic auth, JWT auth, connection limits
◫ **WebSocket Support**	First-class WebSocket proxying with ping/pong keep-alive
◫ **Custom Protocols**	Socket and datagram handlers for implementing any protocol in TypeScript
◫ **Live Metrics**	Real-time throughput, connection counts, UDP sessions, and performance data
◫ **Dynamic Management**	Add/remove ports and routes at runtime without restarts
◫ **PROXY Protocol**	Full PROXY protocol v1/v2 support for preserving client information
◫ **Consumer Cert Storage**	Bring your own persistence — SmartProxy never writes certs to disk

◫ Quick Start

Get up and running in 30 seconds:

```
import { SmartProxy, SocketHandlers } from '@push.rocks/smartproxy';

// Create a proxy with automatic HTTPS
```

```
const proxy = new SmartProxy({
  acme: {
    email: 'ssl@yourdomain.com',
    useProduction: true
  },
  routes: [
    // HTTPS route with automatic Let's Encrypt cert
    {
      name: 'https-app',
      match: { ports: 443, domains: 'app.example.com' },
      action: {
        type: 'forward',
        targets: [{ host: 'localhost', port: 3000 }],
        tls: { mode: 'terminate', certificate: 'auto' }
      }
    },
    // HTTP → HTTPS redirect
    {
      name: 'http-redirect',
      match: { ports: 80, domains: 'app.example.com' },
      action: {
        type: 'socket-handler',
        socketHandler: SocketHandlers.httpRedirect('https://{domain}:443{path}', 301)
      }
    }
  ]
});

await proxy.start();
console.log('□□Proxy running with automatic HTTPS!');
```

□□ Core Concepts

□□ Route-Based Architecture

SmartProxy uses a powerful **match/action** pattern that makes routing predictable and maintainable:

```
{
  name: 'api-route',
  match: {
    ports: 443,
    domains: 'api.example.com',
    path: '/v1/*'
  },
  action: {
    type: 'forward',
    targets: [{ host: 'backend', port: 8080 }],
    tls: { mode: 'terminate', certificate: 'auto' }
  }
}
```

Every route consists of:

- **Match** — What traffic to capture (ports, domains, paths, transport, protocol, IPs, headers)
- **Action** — What to do with it (`forward` or `socket-handler`)
- **Security** (optional) — IP allow/block lists, rate limits, authentication
- **Headers** (optional) — Request/response header manipulation with template variables
- **Name/Priority** (optional) — For identification and ordering

□□ TLS Modes

SmartProxy supports three TLS handling modes:

Mode	Description	Use Case
`passthrough`	Forward encrypted traffic as-is (SNI-based routing)	Backend handles TLS
`terminate`	Decrypt at proxy, forward plain HTTP to backend	Standard reverse proxy
`terminate-and-reencrypt`	Decrypt at proxy, re-encrypt to backend. HTTP traffic gets full per-request routing (Host header, path matching) via the HTTP proxy; non-HTTP traffic uses a raw TLS-to-TLS tunnel	Zero-trust / defense-in-depth environments

□□ Common Use Cases

□□ HTTP to HTTPS Redirect

```
import { SmartProxy, SocketHandlers } from '@push.rocks/smartproxy';

const proxy = new SmartProxy({
  routes: [{
    name: 'http-to-https',
    match: { ports: 80, domains: ['example.com', '*.example.com'] },
    action: {
      type: 'socket-handler',
      socketHandler: SocketHandlers.httpRedirect('https://{domain}:443{path}', 301)
    }
  }]
});
```

⚖ Load Balancer with Health Checks

```
import { SmartProxy } from '@push.rocks/smartproxy';

const proxy = new SmartProxy({
  routes: [{
    name: 'load-balancer',
    match: { ports: 443, domains: 'app.example.com' },
    action: {
      type: 'forward',
      targets: [
        { host: 'server1.internal', port: 8080 },
        { host: 'server2.internal', port: 8080 },
        { host: 'server3.internal', port: 8080 }
      ],
      tls: { mode: 'terminate', certificate: 'auto' },
      loadBalancing: {
        algorithm: 'round-robin',
        healthCheck: {
          path: '/health',
          interval: 30000,
          timeout: 5000,
          unhealthyThreshold: 3,
```

```
          healthyThreshold: 2
        }
      }
    }
  }]
});
```

□□ WebSocket Proxy

```
import { SmartProxy } from '@push.rocks/smartproxy';

const proxy = new SmartProxy({
  routes: [{
    name: 'websocket',
    match: { ports: 443, domains: 'ws.example.com', path: '/socket' },
    priority: 100,
    action: {
      type: 'forward',
      targets: [{ host: 'websocket-server', port: 8080 }],
      tls: { mode: 'terminate', certificate: 'auto' },
      websocket: {
        enabled: true,
        pingInterval: 30000,
        pingTimeout: 10000
      }
    }
  }]
});
```

□□ API Gateway with Rate Limiting

```
import { SmartProxy } from '@push.rocks/smartproxy';

const proxy = new SmartProxy({
  routes: [{
    name: 'api-gateway',
    match: { ports: 443, domains: 'api.example.com', path: '/api/*' },
```

```
    priority: 100,
    action: {
      type: 'forward',
      targets: [{ host: 'api-backend', port: 8080 }],
      tls: { mode: 'terminate', certificate: 'auto' }
    },
    headers: {
      response: {
        'Access-Control-Allow-Origin': '*',
        'Access-Control-Allow-Methods': 'GET, POST, PUT, DELETE, OPTIONS',
        'Access-Control-Allow-Headers': 'Content-Type, Authorization',
        'Access-Control-Max-Age': '86400'
      }
    },
    security: {
      rateLimit: {
        enabled: true,
        maxRequests: 100,
        window: 60,
        keyBy: 'ip'
      }
    }
  }]
});
```

□□ Custom Protocol Handler (TCP)

SmartProxy lets you implement any protocol with full socket control. Routes with JavaScript socket handlers are automatically relayed from the Rust engine back to your TypeScript code:

```
import { SmartProxy, SocketHandlers } from '@push.rocks/smartproxy';

const proxy = new SmartProxy({
  routes: [
    // Use pre-built handlers
    {
      name: 'echo-server',
      match: { ports: 7777, domains: 'echo.example.com' },
      action: { type: 'socket-handler', socketHandler: SocketHandlers.echo }
```

```
    },
    // Or create your own custom protocol
    {
      name: 'custom-protocol',
      match: { ports: 9999, domains: 'custom.example.com' },
      action: {
        type: 'socket-handler',
        socketHandler: async (socket) => {
          console.log(`New connection on custom protocol`);
          socket.write('Welcome to my custom protocol!\n');

          socket.on('data', (data) => {
            const command = data.toString().trim();
            switch (command) {
              case 'PING': socket.write('PONG\n'); break;
              case 'TIME': socket.write(`${new Date().toISOString()}\n`); break;
              case 'QUIT': socket.end('Goodbye!\n'); break;
              default: socket.write(`Unknown: ${command}\n`);
            }
          });
        }
      }
    }
  ]
});
```

Pre-built Socket Handlers:

Handler	Description
`SocketHandlers.echo`	Echo server — returns everything sent
`SocketHandlers.proxy(host, port)`	TCP proxy to another server
`SocketHandlers.lineProtocol(handler)`	Line-based text protocol
`SocketHandlers.httpResponse(code, body)`	Simple HTTP response
`SocketHandlers.httpRedirect(url, code)`	HTTP redirect with template variables (`{domain}`, `{path}`, `{port}`, `{clientIp}`)
`SocketHandlers.httpServer(handler)`	Full HTTP request/response handling
`SocketHandlers.httpBlock(status, message)`	HTTP block response
`SocketHandlers.block(message)`	Block with optional message

□□ UDP Datagram Handler

Handle raw UDP datagrams with custom TypeScript logic — perfect for DNS, game servers, IoT protocols, or any UDP-based service:

```
import { SmartProxy } from '@push.rocks/smartproxy';
import type { IRouteConfig, TDatagramHandler, IDatagramInfo } from '@push.rocks/smartproxy';

// Custom UDP echo handler
const udpHandler: TDatagramHandler = (datagram, info, reply) => {
  console.log(`UDP from ${info.sourceIp}:${info.sourcePort} on port ${info.destPort}`);
  reply(datagram); // Echo it back
};

const proxy = new SmartProxy({
  routes: [{
    name: 'udp-echo',
    match: {
      ports: 5353,
      transport: 'udp'        // □□Listen for UDP datagrams
    },
    action: {
      type: 'socket-handler',
      datagramHandler: udpHandler,  // □□Process each datagram
      udp: {
        sessionTimeout: 60000,     // Session idle timeout (ms)
        maxSessionsPerIP: 100,
        maxDatagramSize: 65535
      }
    }
  }]
});

await proxy.start();
```

□□ QUIC / HTTP3 Forwarding

Forward QUIC traffic to backends with optional protocol translation (e.g., receive QUIC, forward as TCP/HTTP1):

```
import { SmartProxy } from '@push.rocks/smartproxy';
import type { IRouteConfig } from '@push.rocks/smartproxy';

const quicRoute: IRouteConfig = {
  name: 'quic-to-backend',
  match: {
    ports: 443,
    transport: 'udp',
    protocol: 'quic'           // □□Match QUIC protocol
  },
  action: {
    type: 'forward',
    targets: [{
      host: 'backend-server',
      port: 8443,
      backendTransport: 'tcp'  // □□Translate QUIC → TCP for backend
    }],
    tls: {
      mode: 'terminate',
      certificate: 'auto'     // □□QUIC requires TLS 1.3
    },
    udp: {
      quic: {
        enableHttp3: true,
        maxIdleTimeout: 30000,
        maxConcurrentBidiStreams: 100,
        altSvcPort: 443,       // Advertise in Alt-Svc header
        altSvcMaxAge: 86400
      }
    }
  }
};

const proxy = new SmartProxy({
  acme: { email: 'ssl@example.com' },
  routes: [quicRoute]
});
```

□□ Best-Effort Backend Protocol (H3 > H2 > H1)

SmartProxy automatically uses the **highest protocol your backend supports** for HTTP requests. The backend protocol is independent of the client protocol — a client using HTTP/1.1 can be forwarded over HTTP/3 to the backend, and vice versa.

```
const route: IRouteConfig = {
  name: 'auto-protocol',
  match: { ports: 443, domains: 'app.example.com' },
  action: {
    type: 'forward',
    targets: [{ host: 'backend', port: 8443 }],
    tls: { mode: 'terminate', certificate: 'auto' },
    options: {
      backendProtocol: 'auto'  // □□Default — best-effort selection
    }
  }
};
```

How protocol discovery works (browser model):

1. First request → TLS ALPN probe detects H2 or H1
2. Backend response inspected for `Alt-Svc: h3=":port"` header
3. If H3 advertised → cached and used for subsequent requests via QUIC
4. Graceful fallback: H3 failure → H2 → H1 with automatic cache invalidation

`backendProtocol`	Behavior
`'auto'` (default)	Best-effort: H3 > H2 > H1 with Alt-Svc discovery
`'http1'`	Always HTTP/1.1
`'http2'`	Always HTTP/2 (hard-fail if unsupported)
`'http3'`	Always HTTP/3 via QUIC (hard-fail if unsupported)

> **Note:** WebSocket upgrades always use HTTP/1.1 to the backend regardless of `backendProtocol`, since there's no performance benefit from H2/H3 Extended CONNECT for tunneled connections, and backend support is rare.

□□ Dual-Stack TCP + UDP Route

Listen on both TCP and UDP with a single route — handle each transport with its own handler:

```
const dualStackRoute: IRouteConfig = {
  name: 'dual-stack-dns',
  match: {
    ports: 53,
    transport: 'all'  // □□Listen on both TCP and UDP
  },
  action: {
    type: 'socket-handler',
    socketHandler: handleTcpDns,      // □□TCP connections
    datagramHandler: handleUdpDns,    // □□UDP datagrams
  }
};
```

⚡ High-Performance NFTables Forwarding

For ultra-low latency on Linux, use kernel-level forwarding via `@push.rocks/smartnftables` (requires root):

```
import { SmartProxy } from '@push.rocks/smartproxy';

const proxy = new SmartProxy({
  routes: [{
    name: 'nftables-fast',
    match: { ports: 443, domains: 'fast.example.com' },
    action: {
      type: 'forward',
      forwardingEngine: 'nftables',
      targets: [{ host: 'backend', port: 8080 }],
      tls: { mode: 'terminate', certificate: 'auto' },
      nftables: {
        protocol: 'tcp',
        preserveSourceIP: true  // Backend sees real client IP
      }
    }
```

```
  }]
});
```

□□ SNI Passthrough (TLS Passthrough)

Forward encrypted traffic to backends without terminating TLS — the proxy routes based on the SNI hostname alone:

```
import { SmartProxy } from '@push.rocks/smartproxy';

const proxy = new SmartProxy({
  routes: [{
    name: 'sni-passthrough',
    match: { ports: 443, domains: 'secure.example.com' },
    action: {
      type: 'forward',
      targets: [{ host: 'backend-that-handles-tls', port: 8443 }],
      tls: { mode: 'passthrough' }
    }
  }]
});
```

□□ Advanced Features

□□ Dynamic Routing

Route traffic based on runtime conditions using function-based host/port resolution:

```
const proxy = new SmartProxy({
  routes: [{
    name: 'dynamic-backend',
    match: {
      ports: 443,
      domains: 'app.example.com'
    },
    action: {
      type: 'forward',
```

```
      targets: [{
        host: (context) => {
          return context.path?.startsWith('/premium')
            ? 'premium-backend'
            : 'standard-backend';
        },
        port: 8080
      }],
      tls: { mode: 'terminate', certificate: 'auto' }
    }
  }]
});
```

> **Note:** Routes with dynamic functions (host/port callbacks) are automatically relayed through the TypeScript socket handler server, since JavaScript functions can't be serialized to Rust.

□□ Protocol-Specific Routing

Restrict routes to specific application-layer protocols. When `protocol` is set, the Rust engine detects the protocol after connection (or after TLS termination) and only matches routes that accept that protocol:

```
// HTTP-only route (rejects raw TCP connections)
const httpOnlyRoute: IRouteConfig = {
  name: 'http-api',
  match: {
    ports: 443,
    domains: 'api.example.com',
    protocol: 'http',   // Only match HTTP/1.1, HTTP/2, and WebSocket upgrades
  },
  action: {
    type: 'forward',
    targets: [{ host: 'api-backend', port: 8080 }],
    tls: { mode: 'terminate', certificate: 'auto' }
  }
};
```

```
// Raw TCP route (rejects HTTP traffic)
const tcpOnlyRoute: IRouteConfig = {
  name: 'database-proxy',
  match: {
    ports: 5432,
    protocol: 'tcp',    // Only match non-HTTP TCP streams
  },
  action: {
    type: 'forward',
    targets: [{ host: 'db-server', port: 5432 }]
  }
};
```

> **Note:** Omitting `protocol` (the default) matches any protocol. For TLS routes, protocol detection happens *after* TLS termination — during the initial SNI-based route match, `protocol` is not yet known and the route is allowed to match. The protocol restriction is enforced after the proxy peeks at the decrypted data.

Security Controls

Comprehensive per-route security options:

```
{
  name: 'secure-api',
  match: { ports: 443, domains: 'api.example.com' },
  action: {
    type: 'forward',
    targets: [{ host: 'api-backend', port: 8080 }],
    tls: { mode: 'terminate', certificate: 'auto' }
  },
  security: {
    // IP-based access control
    ipAllowList: ['10.0.0.0/8', '192.168.*'],
    ipBlockList: ['192.168.1.100'],

    // Connection limits
    maxConnections: 1000,
```

```
    // Rate limiting
    rateLimit: {
      enabled: true,
      maxRequests: 100,
      window: 60
    },

    // Authentication
    basicAuth: { users: [{ username: 'admin', password: 'secret' }] },
    jwtAuth: { secret: 'your-jwt-secret', algorithm: 'HS256' }
  }
}
```

Security options are configured directly on each route's `security` property — no separate helpers needed.

□□ Runtime Management

Control your proxy without restarts:

```
// Dynamic port management
await proxy.addListeningPort(8443);
await proxy.removeListeningPort(8080);
const ports = await proxy.getListeningPorts();

// Update routes on the fly (atomic, mutex-locked)
await proxy.updateRoutes([...newRoutes]);

// Get real-time metrics
const metrics = proxy.getMetrics();
console.log(`Active connections: ${metrics.connections.active()}`);
console.log(`Bytes in: ${metrics.totals.bytesIn()}`);
console.log(`Requests/sec: ${metrics.requests.perSecond()}`);
console.log(`Throughput in: ${metrics.throughput.instant().in} bytes/sec`);

// UDP metrics
console.log(`UDP sessions: ${metrics.udp.activeSessions()}`);
console.log(`Datagrams in: ${metrics.udp.datagramsIn()}`);
```

```
// Get detailed statistics from the Rust engine
const stats = await proxy.getStatistics();

// Certificate management
await proxy.provisionCertificate('my-route-name');
await proxy.renewCertificate('my-route-name');
const certStatus = await proxy.getCertificateStatus('my-route-name');

// NFTables status
const nftStatus = await proxy.getNfTablesStatus();
```

□□ Header Manipulation

Transform requests and responses with template variables:

```
{
  action: {
    type: 'forward',
    targets: [{ host: 'backend', port: 8080 }]
  },
  headers: {
    request: {
      'X-Real-IP': '{clientIp}',
      'X-Request-ID': '{uuid}',
      'X-Forwarded-Proto': 'https'
    },
    response: {
      'Strict-Transport-Security': 'max-age=31536000',
      'X-Frame-Options': 'DENY'
    }
  }
}
```

□□ PROXY Protocol Support

Preserve original client information through proxy chains:

```
const proxy = new SmartProxy({
  // Accept PROXY protocol from trusted load balancers
  acceptProxyProtocol: true,
  proxyIPs: ['10.0.0.1', '10.0.0.2'],

  // Forward PROXY protocol to backends
  sendProxyProtocol: true,

  routes: [...]
});
```

□□ Custom Certificate Provisioning

Supply your own certificates or integrate with external certificate providers:

```
const proxy = new SmartProxy({
  certProvisionFunction: async (domain: string) => {
    // Return 'http01' to let the built-in ACME handle it
    if (domain.endsWith('.example.com')) return 'http01';

    // Or return a static certificate object
    return {
      publicKey: myPemCert,
      privateKey: myPemKey,
    };
  },
  certProvisionFallbackToAcme: true,  // Fall back to ACME if callback fails
  routes: [...]
});
```

□□ Consumer-Managed Certificate Storage

SmartProxy **never writes certificates to disk**. Instead, you own all persistence through the `certStore` interface. This gives you full control — store certs in a database, cloud KMS, encrypted vault, or wherever makes sense for your infrastructure:

```
const proxy = new SmartProxy({
  routes: [...],
```

```
  certProvisionFunction: async (domain) => myAcme.provision(domain),

  // Your persistence layer — SmartProxy calls these hooks
  certStore: {
    // Called once on startup to pre-load persisted certs
    loadAll: async () => {
      const certs = await myDb.getAllCerts();
      return certs.map(c => ({
        domain: c.domain,
        publicKey: c.certPem,
        privateKey: c.keyPem,
        ca: c.caPem,      // optional
      }));
    },

    // Called after each successful cert provision
    save: async (domain, publicKey, privateKey, ca) => {
      await myDb.upsertCert({ domain, certPem: publicKey, keyPem: privateKey, caPem: ca });
    },

    // Optional: called when a cert should be removed
    remove: async (domain) => {
      await myDb.deleteCert(domain);
    },
  },
});
```

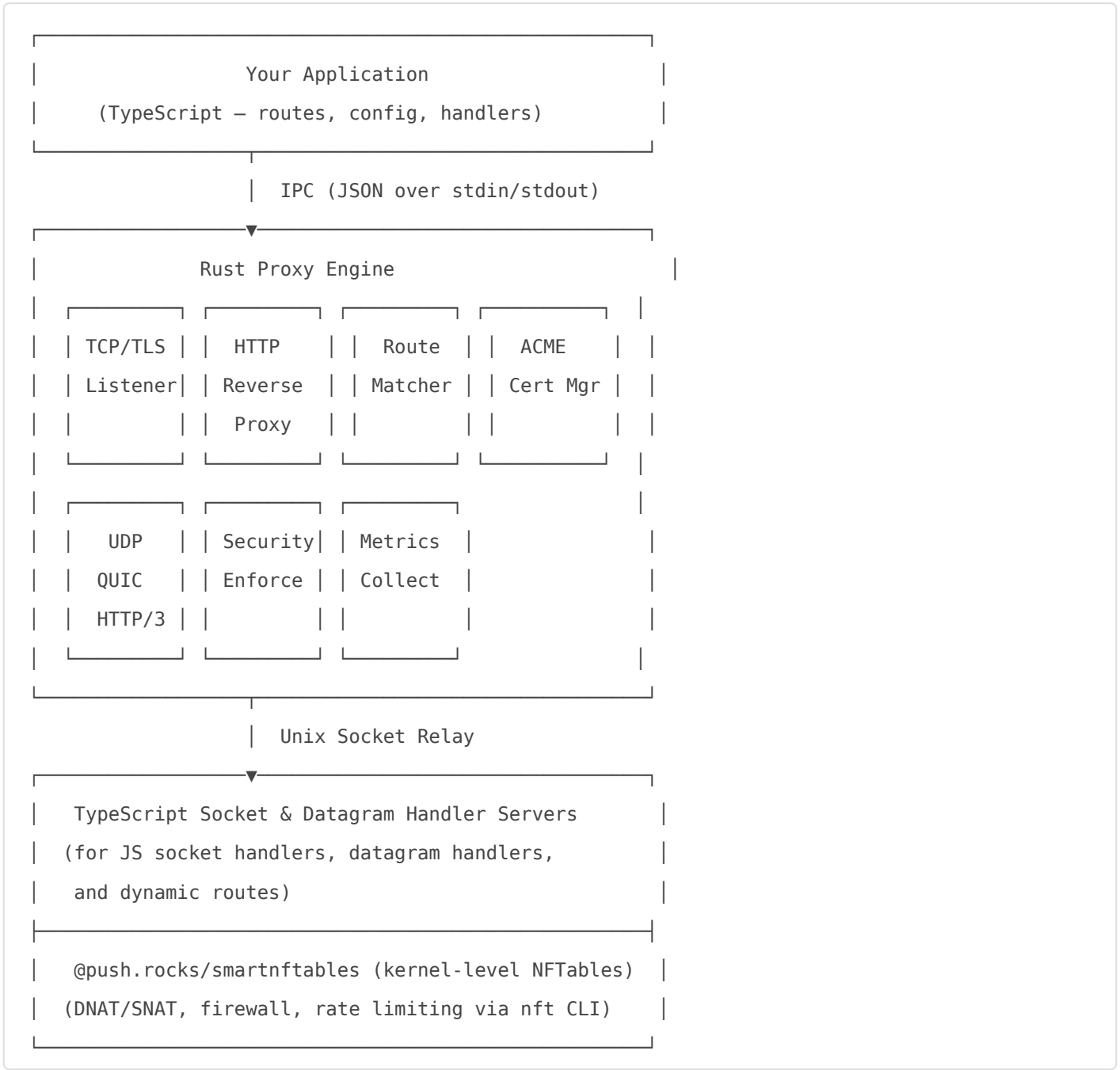
Startup flow:

1. Rust engine starts
2. Default self-signed `*` fallback cert is loaded (unless `disableDefaultCert: true`)
3. `certStore.loadAll()` is called → all returned certs are loaded into the Rust TLS stack
4. `certProvisionFunction` runs for any remaining `certificate: 'auto'` routes (skipping domains already loaded from the store)
5. After each successful provision, `certStore.save()` is called

This means your second startup is instant — no re-provisioning needed for domains that already have valid certs in your store.

□□ Architecture

SmartProxy uses a hybrid **Rust + TypeScript** architecture:



- **Rust Engine** handles all networking: TCP, UDP, TLS, QUIC, HTTP proxying, connection management, security, and metrics
- **TypeScript** provides the npm API, configuration types, validation, and handler callbacks
- **NFTables** managed by `@push.rocks/smartnftables` — kernel-level DNAT/SNAT forwarding, firewall rules, and rate limiting via the `nft` CLI
- **IPC** — The TypeScript wrapper uses JSON commands/events over stdin/stdout to communicate with the Rust binary

- **Socket/Datagram Relay** — Unix domain socket servers for routes requiring TypeScript-side handling (socket handlers, datagram handlers, dynamic host/port functions)

□□ Route Configuration Reference

Match Criteria

```
interface IRouteMatch {
  ports: TPortRange;                    // Required — port(s) to listen on
  transport?: 'tcp' | 'udp' | 'all';    // Transport protocol (default: 'tcp')
  domains?: string | string[];          // 'example.com', '*.example.com'
  path?: string;                        // '/api/*', '/users/:id'
  clientIp?: string[];                  // ['10.0.0.0/8', '192.168.*']
  tlsVersion?: string[];                // ['TLSv1.2', 'TLSv1.3']
  headers?: Record<string, string | RegExp>;  // Match by HTTP headers
  protocol?: 'http' | 'tcp' | 'udp' | 'quic' | 'http3';  // Application-layer protocol
}

// Port range supports single numbers, arrays, and ranges
type TPortRange = number | Array<number | { from: number; to: number }>;
```

Action Types

Type	Description
`forward`	Proxy to one or more backend targets (with optional TLS, WebSocket, load balancing, UDP/QUIC)
`socket-handler`	Custom socket/datagram handling function in TypeScript

Target Options

```
interface IRouteTarget {
  host: string | string[] | ((context: IRouteContext) => string | string[]);
  port: number | 'preserve' | ((context: IRouteContext) => number);
  tls?: IRouteTls;                  // Per-target TLS override
  priority?: number;                // Target priority
```

```
  match?: ITargetMatch;             // Sub-match within a route (by port, path, headers,
method)
  websocket?: IRouteWebSocket;
  loadBalancing?: IRouteLoadBalancing;
  sendProxyProtocol?: boolean;
  headers?: IRouteHeaders;
  advanced?: IRouteAdvanced;
  backendTransport?: 'tcp' | 'udp'; // Backend transport (e.g., receive QUIC, forward as TCP)
}
```

TLS Options

```
interface IRouteTls {
  mode: 'passthrough' | 'terminate' | 'terminate-and-reencrypt';
  certificate?: 'auto' | {
    key: string;
    cert: string;
    ca?: string;
    keyFile?: string;
    certFile?: string;
  };
  acme?: {
    email: string;
    useProduction?: boolean;
    challengePort?: number;
    renewBeforeDays?: number;
  };
  versions?: string[];
  ciphers?: string;
  honorCipherOrder?: boolean;
  sessionTimeout?: number;
}
```

WebSocket Options

```
interface IRouteWebSocket {
  enabled: boolean;
```

```
  pingInterval?: number;     // ms between pings
  pingTimeout?: number;      // ms to wait for pong
  maxPayloadSize?: number;   // Maximum frame payload
  subprotocols?: string[];   // Allowed subprotocols
  allowedOrigins?: string[]; // CORS origins
}
```

Load Balancing Options

```
interface IRouteLoadBalancing {
  algorithm: 'round-robin' | 'least-connections' | 'ip-hash';
  healthCheck?: {
    path: string;
    interval: number;        // ms
    timeout: number;         // ms
    unhealthyThreshold: number;
    healthyThreshold: number;
  };
}
```

Backend Protocol Options

```
// Set on action.options
{
  action: {
    type: 'forward',
    targets: [...],
    options: {
      backendProtocol: 'auto' | 'http1' | 'http2' | 'http3'
    }
  }
}
```

Value	Backend Behavior
`'auto'`	Best-effort: discovers H3 via Alt-Svc, probes H2 via ALPN, falls back to H1
`'http1'`	Always HTTP/1.1 (no ALPN probe)

Value	Backend Behavior
`'http2'`	Always HTTP/2 (hard-fail if handshake fails)
`'http3'`	Always HTTP/3 over QUIC (3s connect timeout, hard-fail if unreachable)

UDP & QUIC Options

```
interface IRouteUdp {
  sessionTimeout?: number;       // Idle timeout per UDP session (ms, default: 60000)
  maxSessionsPerIP?: number;     // Max concurrent sessions per IP (default: 1000)
  maxDatagramSize?: number;      // Max datagram size in bytes (default: 65535)
  quic?: IRouteQuic;
}

interface IRouteQuic {
  maxIdleTimeout?: number;               // QUIC idle timeout (ms, default: 30000)
  maxConcurrentBidiStreams?: number;      // Max bidi streams (default: 100)
  maxConcurrentUniStreams?: number;       // Max uni streams (default: 100)
  enableHttp3?: boolean;                 // Enable HTTP/3 (default: false)
  altSvcPort?: number;                   // Port for Alt-Svc header
  altSvcMaxAge?: number;                 // Alt-Svc max age in seconds (default: 86400)
  initialCongestionWindow?: number;      // Initial congestion window (bytes)
}
```

□□ Exports Reference

```
import {
  // Core
  SmartProxy,                    // Main proxy class
  SocketHandlers,                // Pre-built socket handlers (echo, proxy, block,
httpRedirect, httpServer, etc.)

  // Route Utilities
  mergeRouteConfigs,             // Deep-merge two route configs
  findMatchingRoutes,            // Find routes matching criteria
  findBestMatchingRoute,         // Find best matching route
  cloneRoute,                    // Deep-clone a route
```

```
  generateRouteId,                // Generate deterministic route ID
  RouteValidator,                 // Validate route configurations
} from '@push.rocks/smartproxy';
```

All routes are configured as plain `IRouteConfig` objects with `match` and `action` properties — see the examples throughout this document.

□□ API Documentation

SmartProxy Class

```
class SmartProxy extends EventEmitter {
  constructor(options: ISmartProxyOptions);

  // Lifecycle
  start(): Promise<void>;
  stop(): Promise<void>;

  // Route Management (atomic, mutex-locked)
  updateRoutes(routes: IRouteConfig[]): Promise<void>;

  // Port Management
  addListeningPort(port: number): Promise<void>;
  removeListeningPort(port: number): Promise<void>;
  getListeningPorts(): Promise<number[]>;

  // Monitoring & Metrics
  getMetrics(): IMetrics;              // Sync — returns cached metrics adapter
  getStatistics(): Promise<any>;       // Async — queries Rust engine

  // Certificate Management
  provisionCertificate(routeName: string): Promise<void>;
  renewCertificate(routeName: string): Promise<void>;
  getCertificateStatus(routeName: string): Promise<any>;
  getEligibleDomainsForCertificates(): string[];

  // NFTables (managed by @push.rocks/smartnftables)
```

```
  getNfTablesStatus(): INftStatus | null;

  // Events
  on(event: 'error', handler: (err: Error) => void): this;
  on(event: 'certificate-issued', handler: (ev: ICertificateIssuedEvent) => void): this;
  on(event: 'certificate-failed', handler: (ev: ICertificateFailedEvent) => void): this;
}
```

Configuration Options

```
interface ISmartProxyOptions {
  routes: IRouteConfig[];                   // Required: array of route configs

  // ACME/Let's Encrypt
  acme?: {
    email: string;                          // Contact email for Let's Encrypt
    useProduction?: boolean;                // Use production servers (default: false)
    port?: number;                          // HTTP-01 challenge port (default: 80)
    renewThresholdDays?: number;            // Days before expiry to renew (default: 30)
    autoRenew?: boolean;                    // Enable auto-renewal (default: true)
    renewCheckIntervalHours?: number;       // Renewal check interval (default: 24)
  };

  // Custom certificate provisioning
  certProvisionFunction?: (domain: string) => Promise<ICert | 'http01'>;
  certProvisionFallbackToAcme?: boolean;    // Fall back to ACME on failure (default: true)
  certProvisionTimeout?: number;            // Timeout per provision call (ms)
  certProvisionConcurrency?: number;        // Max concurrent provisions

  // Consumer-managed certificate persistence (see "Consumer-Managed Certificate Storage")
  certStore?: ISmartProxyCertStore;

  // Self-signed fallback
  disableDefaultCert?: boolean;             // Disable '*' self-signed fallback (default:
false)

  // Rust binary path override
  rustBinaryPath?: string;                  // Custom path to the Rust proxy binary
```

```
// Global defaults
defaults?: {
  target?: { host: string; port: number };
  security?: { ipAllowList?: string[]; ipBlockList?: string[]; maxConnections?: number };
};

// PROXY protocol
proxyIPs?: string[];                      // Trusted proxy IPs
acceptProxyProtocol?: boolean;            // Accept PROXY protocol headers
sendProxyProtocol?: boolean;              // Send PROXY protocol to targets

// Timeouts
connectionTimeout?: number;               // Backend connection timeout (default: 60s)
initialDataTimeout?: number;              // Initial data/SNI timeout (default: 60s)
socketTimeout?: number;                   // Socket inactivity timeout (default: 60s)
maxConnectionLifetime?: number;           // Max connection lifetime (default: 1h)
inactivityTimeout?: number;               // Inactivity timeout (default: 75s)
gracefulShutdownTimeout?: number;         // Shutdown grace period (default: 30s)

// Connection limits
maxConnectionsPerIP?: number;             // Per-IP connection limit (default: 100)
connectionRateLimitPerMinute?: number;    // Per-IP rate limit (default: 300/min)

// Keep-alive
keepAliveTreatment?: 'standard' | 'extended' | 'immortal';
keepAliveInactivityMultiplier?: number;   // (default: 4)
extendedKeepAliveLifetime?: number;       // (default: 1h)

// Metrics
metrics?: {
  enabled?: boolean;
  sampleIntervalMs?: number;
  retentionSeconds?: number;
};

// Behavior
enableDetailedLogging?: boolean;          // Verbose connection logging
enableTlsDebugLogging?: boolean;          // TLS handshake debug logging
```

```
}
```

ISmartProxyCertStore Interface

```
interface ISmartProxyCertStore {
  /** Called once on startup to pre-load persisted certs */
  loadAll: () => Promise<Array<{
    domain: string;
    publicKey: string;
    privateKey: string;
    ca?: string;
  }>>;

  /** Called after each successful cert provision */
  save: (domain: string, publicKey: string, privateKey: string, ca?: string) => Promise<void>;

  /** Optional: remove a cert from storage */
  remove?: (domain: string) => Promise<void>;
}
```

IMetrics Interface

The `getMetrics()` method returns a cached metrics adapter that polls the Rust engine:

```
const metrics = proxy.getMetrics();

// Connection metrics
metrics.connections.active();              // Current active connections
metrics.connections.total();               // Total connections since start
metrics.connections.byRoute();             // Map<routeName, activeCount>
metrics.connections.byIP();                // Map<ip, activeCount>
metrics.connections.topIPs(10);            // Top N IPs by connection count

// Throughput (bytes/sec)
metrics.throughput.instant();              // { in: number, out: number }
metrics.throughput.recent();               // Recent average
metrics.throughput.average();              // Overall average
```

```
metrics.throughput.byRoute();             // Map<routeName, { in, out }>

// Request rates
metrics.requests.perSecond();             // Requests per second
metrics.requests.perMinute();             // Requests per minute
metrics.requests.total();                 // Total requests

// UDP metrics
metrics.udp.activeSessions();             // Current active UDP sessions
metrics.udp.totalSessions();             // Total UDP sessions since start
metrics.udp.datagramsIn();               // Datagrams received
metrics.udp.datagramsOut();              // Datagrams sent

// Cumulative totals
metrics.totals.bytesIn();                 // Total bytes received
metrics.totals.bytesOut();                // Total bytes sent
metrics.totals.connections();             // Total connections

// Backend metrics
metrics.backends.byBackend();             // Map<backend, IBackendMetrics>
metrics.backends.protocols();             // Map<backend, protocol>
metrics.backends.topByErrors(10);         // Top N error-prone backends

// Percentiles
metrics.percentiles.connectionDuration();  // { p50, p95, p99 }
metrics.percentiles.bytesTransferred();    // { in: { p50, p95, p99 }, out: { p50, p95, p99 }
}
```

□□ Troubleshooting

Certificate Issues

- □ Ensure domain DNS points to your server
- □ Port 80 must be accessible for ACME HTTP-01 challenges
- □ Check DNS propagation with `dig` or `nslookup`
- □ Verify the email in ACME configuration is valid
- □ Use `getCertificateStatus('route-name')` to check cert state

Connection Problems

- □ Check route priorities (higher number = matched first)
- □ Verify security rules aren't blocking legitimate traffic
- □ Test with `curl -v` for detailed connection output
- □ Enable debug logging with `enableDetailedLogging: true`

Rust Binary Not Found

SmartProxy searches for the Rust binary in this order:

1. `rustBinaryPath` option in `ISmartProxyOptions`
2. `SMARTPROXY_RUST_BINARY` environment variable
3. Platform-specific npm package (`@push.rocks/smartproxy-linux-x64`, etc.)
4. `dist_rust/rustproxy` relative to the package root (built by `tsrust`)
5. Local dev build (`./rust/target/release/rustproxy`)
6. System PATH (`rustproxy`)

QUIC / HTTP3 Caveats

- **GREASE frames are disabled.** The underlying h3 crate sends GREASE frames by default to test protocol extensibility. However, some HTTP/3 clients and servers don't properly ignore unknown frame types, causing 400/500 errors or stream hangs (h3#206). SmartProxy disables GREASE on both the server side (for incoming H3 requests) and the client side (for H3 backend connections) to maximize compatibility.
- **HTTP/3 is pre-release.** The h3 ecosystem (h3 0.0.8, h3-quinn 0.0.10, quinn 0.11) is still pre-1.0. Expect rough edges.

Performance Tuning

- □ Use NFTables forwarding for high-traffic routes (Linux only)
- □ Enable connection keep-alive where appropriate
- □ Use `getMetrics()` and `getStatistics()` to identify bottlenecks
- □ Adjust `maxConnectionsPerIP` and `connectionRateLimitPerMinute` based on your workload
- □ Use `passthrough` TLS mode when backend can handle TLS directly

□□ Best Practices

1. □□ **Use Helper Functions** — They provide sensible defaults and prevent common mistakes
2. □□ **Set Route Priorities** — More specific routes should have higher priority values
3. □□ **Enable Security** — Always use IP filtering and rate limiting for public-facing services
4. □□ **Monitor Metrics** — Use the built-in metrics to catch issues early
5. □□ **Certificate Monitoring** — Set up alerts before certificates expire
6. □□ **Graceful Shutdown** — Always call `proxy.stop()` for clean connection termination
7. □ **Validate Routes** — Use `RouteValidator.validateRoutes()` to catch config errors before deployment
8. □□ **Atomic Updates** — Use `updateRoutes()` for hot-reloading routes (mutex-locked, no downtime)
9. □□ **Use Socket Handlers** — For protocols beyond HTTP, implement custom socket handlers instead of fighting the proxy model
10. □□ **Use `certStore`** — Persist certs in your own storage to avoid re-provisioning on every restart

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.

changelog.md for @push.rocks/smartproxy

2026-03-27 - 27.1.0 - feat(rustproxy-passthrough)

add selective connection recycling for route, security, and certificate updates

- introduce a shared connection registry to track active TCP and QUIC connections by route, source IP, and domain
- recycle only affected connections when route actions or security rules change instead of broadly invalidating traffic
- gracefully recycle existing connections when TLS certificates change for a domain
- apply route-level IP security checks to QUIC connections and share route cancellation state with UDP listeners

2026-03-26 - 27.0.0 - BREAKING CHANGE(smart-proxy)

remove route helper APIs and standardize route configuration on plain route objects

- Removes TypeScript route helper exports and related Rust config helpers in favor of defining routes directly with match and action properties.
- Updates documentation and tests to use plain IRouteConfig objects and SocketHandlers imports instead of helper factory functions.
- Moves socket handlers to a top-level utils export and keeps direct socket-handler route configuration as the supported pattern.

2026-03-26 - 26.3.0 - feat(nftables)

move NFTables forwarding management from the Rust engine to @push.rocks/smartnftables

- add @push.rocks/smartnftables as a runtime dependency and export it via the plugin layer
- remove the internal rustproxy-nftables crate along with Rust-side NFTables rule application and status management
- apply and clean up NFTables port-forwarding rules in the TypeScript SmartProxy lifecycle and route update flow
- change getNfTablesStatus to return local smartnftables status instead of querying the Rust bridge
- update README documentation to describe NFTables support as provided through @push.rocks/smartnftables

2026-03-26 - 26.2.4 - fix(rustproxy-http)

improve HTTP/3 connection reuse and clean up stale proxy state

- Reuse pooled HTTP/3 SendRequest handles to skip repeated SETTINGS handshakes and reduce request overhead on QUIC pool hits
- Add periodic cleanup for per-route rate limiters and orphaned backend metrics to prevent unbounded memory growth after traffic or backend errors stop
- Enforce HTTP max connection lifetime alongside idle timeouts and apply configured lifetime values from the TCP listener
- Reduce HTTP/3 body copying by using owned Bytes paths for request and response streaming, and replace the custom response body adapter with a stream-based implementation
- Harden auxiliary proxy components by capping datagram handler buffer growth and removing duplicate RustProxy exit listeners

2026-03-25 - 26.2.3 - fix(repo)

no changes to commit

2026-03-25 - 26.2.2 - fix(proxy)

improve connection cleanup and route validation handling

- add timeouts for HTTP/1 upstream connection drivers to prevent lingering tasks
- ensure QUIC relay sessions cancel and abort background tasks on drop
- avoid registering unnamed routes as duplicates and label unnamed catch-all conflicts clearly
- fix offset mapping route helper to forward only remaining route options without overriding derived values
- update project config filename and toolchain versions for the current build setup

2026-03-23 - 26.2.1 - fix(rustproxy-http)

include the upstream request URL when caching H3 Alt-Svc discoveries

- Tracks the request path that triggered Alt-Svc discovery in connection activity state
- Adds request URL context to Alt-Svc debug logging and protocol cache insertion reasons for better traceability

2026-03-23 - 26.2.0 - feat(protocol-cache)

add sliding TTL re-probing and eviction for backend protocol detection

- extend protocol cache entries and metrics with last accessed and last probed timestamps
- trigger periodic ALPN re-probes for cached H1/H2 entries while keeping active entries alive with a sliding 1 day TTL
- log protocol transitions with reasons and evict cache entries when all protocol fallback attempts fail

2026-03-22 - 26.1.0 - feat(rustproxy-http)

add protocol failure suppression, h3 fallback escalation, and protocol cache metrics exposure

- introduces escalating cooldowns for failed H2/H3 protocol detection to prevent repeated upgrades to unstable backends
- adds within-request escalation to cached HTTP/3 when TCP or TLS backend connections fail in auto-detect mode
- exposes detected protocol cache entries and suppression state through Rust metrics and the TypeScript metrics adapter

2026-03-21 - 26.0.0 - BREAKING CHANGE(ts-api,rustproxy)

remove deprecated TypeScript protocol and utility exports while hardening QUIC, HTTP/3, WebSocket, and rate limiter cleanup paths

- Removes large parts of the public TypeScript surface including detection, TLS, router, websocket, proxy/common protocol, and multiple core utility exports and files.
- Adds parent-child cancellation handling for HTTP/3 and QUIC stream forwarding to stop orphaned tasks and close idle or overlong streams.
- Improves cleanup reliability with RAII guards for WebSocket upstream tracking and QUIC connection metrics, plus periodic cleanup for rate limiter and proxy address maps.
- Cleans backend metrics state when active backend connections drop to zero and tracks passthrough backend sockets for shutdown cleanup.

2026-03-20 - 25.17.10 - fix(rustproxy-http)

reuse the shared HTTP proxy service for HTTP/3 request handling

- Refactors H3ProxyService to delegate requests to the shared HttpProxyService instead of maintaining separate routing and backend forwarding logic.

- Aligns HTTP/3 with the TCP/HTTP path for route matching, connection pooling, and ALPN-based upstream protocol detection.
- Generalizes request handling and filters to accept boxed/generic HTTP bodies so both HTTP/3 and existing HTTP paths share the same proxy pipeline.
- Updates the HTTP/3 integration route matcher to allow transport matching across shared HTTP and QUIC handling.

2026-03-20 - 25.17.9 - fix(rustproxy-http)

correct HTTP/3 host extraction and avoid protocol filtering during UDP route lookup

- Use the URI host or strip the port from the Host header so HTTP/3 requests match routes consistently with TCP/HTTP handling.
- Remove protocol filtering from HTTP/3 route lookup because QUIC transport already constrains routing to UDP and protocol validation happens earlier.

2026-03-20 - 25.17.8 - fix(rustproxy)

use SNI-based certificate resolution for QUIC TLS connections

- Replaces static first-certificate selection with the shared CertResolver used by the TCP/TLS path.
- Ensures QUIC connections can present the correct certificate per requested domain.
- Keeps HTTP/3 ALPN configuration while improving multi-domain TLS handling.

2026-03-20 - 25.17.7 - fix(readme)

document QUIC and HTTP/3 compatibility caveats

- Add notes explaining that GREASE frames are disabled on both server and client HTTP/3 paths to avoid interoperability issues
- Document that the current HTTP/3 stack depends on pre-1.0 h3 ecosystem components and may still have rough edges

2026-03-20 - 25.17.6 - fix(rustproxy-http)

disable HTTP/3 GREASE for client and server connections

- Switch the HTTP/3 server connection setup to use the builder API with send_grease(false)
- Switch the HTTP/3 client handshake to use the builder API with send_grease(false) to improve compatibility

2026-03-20 - 25.17.5 - fix(rustproxy)

add HTTP/3 integration test for QUIC response stream FIN handling

- adds an integration test covering HTTP/3 proxying over QUIC with TLS termination
- verifies response bodies fully arrive and the client receives stream termination instead of hanging
- adds test-only dependencies for quinn, h3, h3-quinn, rustls, bytes, and http

2026-03-20 - 25.17.4 - fix(rustproxy-http)

prevent HTTP/3 response body streaming from hanging on backend completion

- extract and track Content-Length before consuming the response body
- stop the HTTP/3 body loop when the stream reports end-of-stream or the expected byte count has been sent
- add a per-frame idle timeout to avoid indefinite waits on stalled or close-delimited backend bodies

2026-03-20 - 25.17.3 - fix(repository)

no changes detected

2026-03-20 - 25.17.2 - fix(rustproxy-http)

enable TLS connections for HTTP/3 upstream requests when backend re-encryption or TLS is configured

- Pass backend TLS client configuration into the HTTP/3 request handler.
- Detect TLS-required upstream targets using route and target TLS settings before connecting.
- Build backend request URIs with the correct http or https scheme to match the upstream connection.

2026-03-20 - 25.17.1 - fix(rustproxy-routing)

allow QUIC UDP TLS connections without SNI to match domain-restricted routes

- Exempts UDP transport from the no-SNI rejection logic because QUIC encrypts the TLS ClientHello and SNI is unavailable at accept time
- Adds regression tests to confirm QUIC route matching succeeds without SNI while TCP TLS without SNI remains rejected

2026-03-19 - 25.17.0 - feat(rustproxy-passthrough)

add PROXY protocol v2 client IP handling for UDP and QUIC listeners

- propagate trusted proxy IP configuration into UDP and QUIC listener managers
- extract and preserve real client addresses from PROXY protocol v2 headers for HTTP/3 and QUIC stream handling
- apply rate limiting, session limits, routing, and metrics using the resolved client IP while preserving correct proxy return-path routing

2026-03-19 - 25.16.3 - fix(rustproxy)

upgrade fallback UDP listeners to QUIC when TLS certificates become available

- Rebuild and apply QUIC TLS configuration during route and certificate updates instead of only when adding new UDP ports.
- Add logic to drain UDP sessions, stop raw fallback listeners, and start QUIC endpoints on existing ports once TLS is available.
- Retry QUIC endpoint creation during upgrade and fall back to rebinding raw UDP if the upgrade cannot complete.

2026-03-19 - 25.16.2 - fix(rustproxy-http)

cache backend Alt-Svc only from original upstream responses during protocol auto-detection

- Moves Alt-Svc discovery into streaming response construction so it reads backend headers before response filters inject client-facing Alt-Svc values
- Stores the protocol cache key in connection activity during auto-detect mode and clears it after HTTP/3 connection failure to avoid re-caching failed H3 routes
- Prevents fallback requests from reintroducing stale or self-injected Alt-Svc entries that could cause repeated H3 retry loops

2026-03-19 - 25.16.1 - fix(http-proxy)

avoid repeated HTTP/3 recaching after QUIC fallback and document backend protocol selection

- Suppress Alt-Svc HTTP/3 recaching after a failed QUIC backend connection to prevent repeated H3 timeout fallback loops
- Force an ALPN probe on TCP fallback so auto detection correctly reselects HTTP/2 or HTTP/1.1 after H3 connection failure
- Add README documentation for best-effort backendProtocol selection and supported protocol modes

2026-03-19 - 25.16.0 - feat(quic,http3)

add HTTP/3 proxy handling and hot-reload QUIC TLS configuration

- initialize and wire H3ProxyService into QUIC listeners so HTTP/3 requests are handled instead of being kept as placeholder connections
- add backend HTTP/3 support with protocol caching that stores Alt-Svc advertised H3 ports for auto-detection
- hot-swap TLS certificates across active QUIC endpoints and require terminating TLS for QUIC route validation
- document QUIC route setup with required TLS and ACME configuration

2026-03-19 - 25.15.0 - feat(readme)

document UDP, QUIC, and HTTP/3 support in the README

- Adds README examples for UDP datagram handlers, QUIC/HTTP3 forwarding, and dual-stack TCP/UDP routes
- Expands configuration and API reference sections to cover transport matching, UDP/QUIC options, backend transport selection, and UDP metrics
- Updates architecture and feature descriptions to reflect UDP, QUIC, HTTP/3, and datagram handler capabilities

2026-03-19 - 25.14.1 - fix(deps)

update build and runtime dependencies and align route validation test expectations

- split the test preparation step into a dedicated test:before script while keeping test execution separate
- bump development tooling and runtime package versions in package.json
- adjust the route validation test to match the current generic handler error message

2026-03-19 - 25.14.0 - feat(udp,http3)

add UDP datagram handler relay support and stream HTTP/3 request bodies to backends

- establish a persistent Unix socket relay for UDP datagram handlers and process handler replies back to clients
- update route validation and smart proxy route reload logic to support datagramHandler routes
- record UDP, QUIC, and HTTP/3 byte metrics more accurately, including request bytes in and UDP session cleanup connection tracking
- add integration tests for UDP forwarding, datagram handlers, and UDP metrics

2026-03-19 - 25.13.0 - feat(smart-proxy)

add UDP transport support with QUIC/HTTP3 routing and datagram handler relay

- adds UDP listener and session tracking infrastructure in the Rust proxy, including UDP metrics and hot-reload support for transport-specific ports
- introduces QUIC and HTTP/3 support in routing and HTTP handling, including Alt-Svc advertisement and QUIC TLS configuration
- extends route configuration types in Rust and TypeScript with transport, UDP, QUIC, backend transport, and mixed port range support
- adds a TypeScript datagram handler relay server and bridge command so UDP socket-handler routes can dispatch datagrams to application callbacks
- updates nftables rule generation so protocol=all creates both TCP and UDP rules

2026-03-19 - 25.12.0 - feat(proxy-protocol)

add PROXY protocol v2 support to the Rust passthrough listener and streamline TypeScript proxy protocol exports

- detect and parse PROXY protocol v2 headers in the Rust TCP listener, including TCP and UDP address families
- add Rust v2 header generation, incomplete-header handling, and broader parser test coverage
- remove deprecated TypeScript proxy protocol parser exports and tests, leaving shared type definitions only

2026-03-17 - 25.11.24 - fix(rustproxy-http)

improve async static file serving, websocket handshake buffering, and shared metric metadata handling

- convert static file serving to async filesystem operations and await directory/file checks
- preserve and forward bytes read past the WebSocket handshake header terminator to avoid dropping buffered upstream data
- reuse Arc values for route and source identifiers across counting bodies and metric reporting
- standardize backend key propagation across H1/H2 forwarding, retry, and fallback paths for consistent logging and metrics

2026-03-17 - 25.11.23 - fix(rustproxy-http,rustproxy-metrics)

reduce per-frame metrics overhead by batching body byte accounting

- Buffer HTTP body byte counts and flush them every 64 KB, at end of stream, and on drop to keep totals accurate while preserving throughput sampling.
- Skip zero-value counter updates in metrics collection to avoid unnecessary atomic and DashMap operations for the unused direction.

2026-03-17 - 25.11.22 - fix(rustproxy-http)

reuse healthy HTTP/2 upstream connections after requests with bodies

- Registers successful HTTP/2 connections in the pool regardless of whether the proxied request included a body
- Continues to avoid pooling upstream connections that returned 502 Bad Gateway responses

2026-03-17 - 25.11.21 - fix(rustproxy-http)

reuse pooled HTTP/2 connections for requests with and without bodies

- remove the bodyless-request restriction from HTTP/2 pool checkout
- always return successful HTTP/2 senders to the connection pool after requests

2026-03-17 - 25.11.20 - fix(rustproxy-http)

avoid downgrading cached backend protocol on H2 stream errors

- Treat HTTP/2 stream-level failures as retryable request errors instead of evidence that the backend only supports HTTP/1.1
- Keep protocol cache entries unchanged after successful H2 handshakes so future requests continue using HTTP/2

- Lower log severity for this fallback path from warning to debug while still recording backend H2 failure metrics

2026-03-16 - 25.11.19 - fix(rustproxy-http)

avoid reusing pooled HTTP/2 connections for requests with bodies to prevent upload flow-control stalls

- Limit HTTP/2 pool checkout to bodyless requests such as GET, HEAD, and DELETE
- Skip re-registering HTTP/2 connections in the pool after requests that send a body
- Prevent stalled uploads caused by depleted connection-level flow control windows on reused HTTP/2 connections

2026-03-16 - 25.11.18 - fix(repo)

no changes to commit

2026-03-16 - 25.11.17 - fix(rustproxy-http)

prevent stale HTTP/2 connection drivers from evicting newer pooled connections

- add generation IDs to pooled HTTP/2 senders so pool removal only affects the matching connection
- update HTTP/2 proxy and retry paths to register generation-tagged connections and skip eviction before registration completes

2026-03-16 - 25.11.16 - fix(repo)

no changes to commit

2026-03-16 - 25.11.15 - fix(rustproxy-http)

implement vectored write support for backend streams

- Add poll_write_vectored forwarding for both plain and TLS backend stream variants
- Expose is_write_vectored so the proxy can correctly report vectored write capability

2026-03-16 - 25.11.14 - fix(rustproxy-http)

forward vectored write support in ShutdownOnDrop AsyncWrite wrapper

- Implements poll_write_vectored by delegating to the wrapped writer
- Exposes is_write_vectored so the wrapper preserves underlying AsyncWrite capabilities

2026-03-16 - 25.11.13 - fix(rustproxy-http)

remove hot-path debug logging from HTTP/1 connection pool hits

- Stops emitting debug logs when reusing HTTP/1 idle connections in the connection pool.
- Keeps pool hit behavior unchanged while reducing overhead on a frequently executed path.

2026-03-16 - 25.11.12 - fix(rustproxy-http)

remove connection pool hit logging and keep logging limited to actual failures

- Removes debug and warning logs for HTTP/2 connection pool hits and age checks.
- Keeps pool behavior unchanged while reducing noisy per-request logging in the Rust HTTP proxy layer.

2026-03-16 - 25.11.11 - fix(rustproxy-http)

improve HTTP/2 proxy error logging with warning-level connection failures and debug error details

- Adds debug-formatted error fields to HTTP/2 handshake, retry, fallback, and request failure logs
- Promotes upstream HTTP/2 connection error logs from debug to warn to improve operational visibility

2026-03-16 - 25.11.10 - fix(rustproxy-http)

validate pooled HTTP/2 connections asynchronously before reuse and evict stale senders

- Add an async ready() check with a 500ms timeout before reusing pooled HTTP/2 senders to catch GOAWAY/RST states before forwarding requests
- Return connection age from the HTTP/2 pool checkout path and log warnings for older pooled connections
- Evict pooled HTTP/2 senders when they are closed, exceed max age, fail readiness validation, or time out during readiness checks

2026-03-16 - 25.11.9 - fix(rustproxy-routing)

reduce hot-path allocations in routing, metrics, and proxy protocol handling

- skip HTTP header map construction unless a route on the current port uses header matching

- reuse computed client IP strings during HTTP route matching to avoid redundant allocations
- optimize per-route and per-IP metric updates with get-first lookups to avoid unnecessary String creation on existing entries
- replace heap-allocated PROXY protocol peek and discard buffers with stack-allocated buffers in the TCP listener
- improve domain matcher case-insensitive wildcard checks while preserving glob fallback behavior

2026-03-16 - 25.11.8 - fix(rustproxy-http)

prevent premature idle timeouts during streamed HTTP responses and ensure TLS close_notify is sent on dropped connections

- track active streaming response bodies so the HTTP idle watchdog does not close connections mid-transfer
- add a ShutdownOnDrop wrapper for TLS-terminated HTTP connections to send shutdown on drop and avoid improperly terminated TLS sessions
- apply the shutdown wrapper in passthrough TLS terminate and terminate+reencrypt HTTP handling

2026-03-16 - 25.11.7 - fix(rustproxy)

prevent TLS route reload certificate mismatches and tighten passthrough connection handling

- Load updated TLS configs before swapping the route manager so newly visible routes always have their certificates available.
- Add timeouts when peeking initial decrypted data after TLS handshake to avoid leaked idle connections.
- Raise dropped, blocked, unmatched, and errored passthrough connection events from debug to warn for better operational visibility.

2026-03-16 - 25.11.6 - fix(rustproxy-http,rustproxy-passthrough)

improve upstream connection cleanup and graceful tunnel shutdown

- Evict pooled HTTP/2 connections when their driver exits and shorten the maximum pooled H2 age to reduce reuse of stale upstream connections.
- Strip hop-by-hop headers from backend responses before forwarding to HTTP/2 clients to avoid invalid H2 response handling.
- Replace immediate task aborts in WebSocket and TCP tunnel watchdogs with cancellation-driven graceful shutdown plus timed fallback aborts.
- Use non-blocking semaphore acquisition in the TCP listener so connection limits do not stall the accept loop for the entire port.

2026-03-16 - 25.11.5 - fix(repo)

no changes to commit

2026-03-15 - 25.11.4 - fix(rustproxy-http)

report streamed HTTP and WebSocket bytes per chunk for real-time throughput metrics

- Update CountingBody to record bytes immediately on each data frame instead of aggregating until completion or drop
- Record WebSocket tunnel traffic inside both copy loops and remove the final aggregate byte report to keep throughput metrics current

2026-03-15 - 25.11.3 - fix(repo)

no changes to commit

2026-03-15 - 25.11.2 - fix(rustproxy-http)

avoid reusing HTTP/1 senders during streaming responses and relax HTTP/2 keep-alive timeouts

- Stop returning HTTP/1 senders to the connection pool before upstream response bodies finish streaming to prevent unsafe reuse on active connections.
- Increase HTTP/2 keep-alive timeout from 5 seconds to 30 seconds in proxy connection builders to better support longer-lived backend streams.
- Improves reliability for large streaming payloads and backend fallback request handling.

2026-03-15 - 25.11.1 - fix(rustproxy-http)

keep connection idle tracking alive during streaming and tune HTTP/2 connection lifetimes

- Propagate connection activity tracking through HTTP/1, HTTP/2, and WebSocket forwarding so active request and response body streams do not trigger the idle watchdog.
- Update CountingBody to refresh connection activity timestamps while data frames are polled during uploads and downloads.
- Increase pooled HTTP/2 max age and set explicit HTTP/2 connection window sizes to improve long-lived streaming behavior.

2026-03-15 - 25.11.0 - feat(rustproxy-http)

add HTTP/2 Extended CONNECT WebSocket proxy support

- Enable HTTP/2 CONNECT protocol support on the Hyper auto connection builder
- Detect WebSocket requests for both HTTP/1 Upgrade and HTTP/2 Extended CONNECT flows

- Translate HTTP/2 WebSocket requests to an HTTP/1.1 backend handshake and return RFC-compliant client responses

2026-03-12 - 25.10.7 - fix(rustproxy-http)

remove Host header from HTTP/2 upstream requests while preserving it for HTTP/1 retries

- strips the Host header before sending HTTP/2 upstream requests so :authority from the URI is used instead
- avoids 400 responses from nginx caused by sending both Host and :authority headers
- keeps a cloned header set for bodyless request retries so HTTP/1 fallback still retains the Host header

2026-03-12 - 25.10.6 - fix(rustproxy-http)

use the requested domain as HTTP/2 authority instead of the backend host and port

- build HTTP/2 absolute URIs from the client-facing domain so the :authority pseudo-header matches the Host header
- remove backend port from generated HTTP/2 request URIs and fall back to the upstream host only when no domain is available
- apply the authority handling consistently across pooled, inline, and generic upstream request paths

2026-03-12 - 25.10.5 - fix(rustproxy-http)

configure HTTP/2 client builders with a Tokio timer for keep-alive handling

- Adds TokioTimer to all HTTP/2 client builder instances in proxy_service.

- Ensures configured HTTP/2 keep-alive interval and timeout settings have the required timer runtime support.

2026-03-12 - 25.10.4 - fix(rustproxy-http)

stabilize upstream HTTP/2 forwarding and fallback behavior

- Remove hop-by-hop headers before forwarding requests to HTTP/2 backends to comply with RFC 9113.
- Use ALPN-enabled TLS configuration whenever HTTP/2 is possible, including explicit H2 connections and retries.
- Add HTTP/2 handshake timeouts, tuned connection settings, and fallback to HTTP/1 when H2 negotiation times out or fails.
- Register pooled HTTP/2 senders only after a successful first request to avoid reusing broken connections.
- Build absolute URIs for HTTP/2 upstream requests so pseudo-headers such as scheme and authority are derived correctly.

2026-03-12 - 25.10.3 - fix(rustproxy-http)

include request domain in backend proxy error and protocol detection logs

- Adds domain context to backend TCP/TLS connect, handshake, request failure, retry, and fallback log entries in the Rust HTTP proxy service.
- Propagates the resolved host/domain through H1, H2, pooled, and fallback forwarding paths so backend-level diagnostics can be correlated with the original request domain.

2026-03-12 - 25.10.2 - fix(repo)

no code changes to release

2026-03-12 - 25.10.1 - fix(repo)

no changes to commit

2026-03-12 - 25.10.0 - feat(metrics)

add per-backend connection, error, protocol, and pool metrics with stale backend pruning

- tracks backend connection lifecycle, connect timing, protocol detection, pool hit/miss rates, handshake/request errors, and h2 fallback failures in Rust metrics
- exposes backend metrics through the TypeScript metrics adapter with backend listings, protocol lookup, and top error summaries
- prunes backend metrics for backends no longer referenced by active routes, including preserved-port targets expanded across listening ports

2026-03-11 - 25.9.3 - fix(rustproxy-http)

Evict stale HTTP/2 pooled senders and retry bodyless requests with fresh backend connections to avoid 502s

- Introduce MAX_H2_AGE (120s) and evict HTTP/2 senders older than this or closed
- Check MAX_H2_AGE on checkout and during background eviction to prevent reuse of stale h2 connections
- Add connection_pool.remove_h2() to explicitly remove dead H2 senders from the pool
- When a pooled H2 request returns a 502 and the original request had an empty body, retry using a fresh H2 connection (retry_h2_with_fresh_connection)
- On H2 auto-detect failures, retry as HTTP/1.1 for bodyless requests via forward_h1_empty_body; return 502 for requests with bodies
- Evict dead H2 senders on backend request failures in reconnect_backend so subsequent attempts create fresh connections

2026-03-08 - 25.9.2 - fix(protocol-cache)

Include requested_host in protocol detection cache key to avoid cache oscillation when multiple frontend domains share the same backend

- Add ProtocolCacheKey.requested_host: Option<String> to distinguish cache entries by incoming request Host/:authority
- Update protocol cache lookups/inserts in proxy_service to populate requested_host
- Enhance debug logging to show requested_host on cache hits
- Fixes repeated ALPN probing / cache oscillation when different frontend domains share a backend with differing HTTP/2 support

2026-03-03 - 25.9.1 - fix(rustproxy)

Cancel connections for routes removed/disabled by adding per-route cancellation tokens and make RouteManager swappable (ArcSwap) for runtime updates

- Add per-route CancellationToken map (DashMap) to TcpListenerManager and call token.cancel() when routes are removed (invalidate_removed_routes)
- Propagate Arc<ArcSwap<RouteManager>> into HttpProxyService and passthrough listener so the route manager can be hot-swapped without restarting listeners
- Use per-route child cancellation tokens in accept/connection handling and forwarders to terminate existing connections when a route is removed
- Prune HTTP proxy caches and retain/cleanup per-route tokens when routes are active/removed
- Update test.test.sni-requirement.node.ts to allocate unique free ports via findFreePorts to avoid port conflicts during tests

2026-03-03 - 25.9.0 - feat(rustproxy-http)

add HTTP/2 auto-detection via ALPN with TTL-backed protocol cache and h1-only/h2 ALPN client configs

- Add protocol_cache module: bounded, TTL-based cache (5min TTL), max entries (4096), background cleanup task and clear() to discard stale detections.
- Introduce BackendProtocol::Auto and expose 'auto' in TypeScript route types to allow ALPN-based protocol auto-detection.
- Add build_tls_acceptor_h1_only() to create a TLS acceptor that advertises only http/1.1 (used for backends/tests that speak plain HTTP/1.1).
- Add shared_backend_tls_config_alpn() and default_backend_tls_config_with_alpn() to provide client TLS configs advertising h2+http/1.1 for auto-detection.
- Wire backend_tls_config_alpn and protocol_cache into proxy_service, tcp_listener and passthrough paths; add set_backend_tls_config_alpn() and prune protocol_cache on route updates.
- Update passthrough tests to use h1-only acceptor to avoid false HTTP/2 detection when backends speak plain HTTP/1.1.
- Include reconnection/fallback handling and ensure ALPN-enabled client config is used for auto-detection mode.

2026-02-26 - 25.8.5 - fix(release)

bump patch version (no source changes)

- No changes detected in git diff
- Current version: 25.8.4
- Recommend patch bump to 25.8.5 to record release without code changes

2026-02-26 - 25.8.4 - fix(proxy)

adjust default proxy timeouts and keep-alive behavior to shorter, more consistent values

- Increase connection timeout default from 30,000ms to 60,000ms (30s -> 60s).
- Reduce socket timeout default from 3,600,000ms to 60,000ms (1h -> 60s).
- Reduce max connection lifetime default from 86,400,000ms to 3,600,000ms (24h -> 1h).
- Change inactivity timeout default from 14,400,000ms to 75,000ms (4h -> 75s).
- Update keep-alive defaults: keepAliveTreatment 'extended' -> 'standard', keepAliveInactivityMultiplier 6 -> 4, extendedKeepAliveLifetime 604800000 -> 3,600,000ms (7d -> 1h).
- Apply these consistent default values across Rust crates (rustproxy-config, rustproxy-passthrough) and the TypeScript smart-proxy implementation.
- Update unit test expectations to match the new defaults.

2026-02-26 - 25.8.3 - fix(smartproxy)

no code or dependency changes detected; no version bump required

- No files changed in the provided diff (No changes).
- package.json version remains 25.8.2.
- No dependency or source updates detected; skip release.

2026-02-26 - 25.8.2 - fix(connection)

improve connection handling and timeouts

- Flush logs on process beforeExit and avoid calling process.exit in SIGINT/SIGTERM handlers to preserve host graceful shutdown
- Store protocol entries with a createdAt timestamp in ProtocolDetector and remove stale entries older than 30s to prevent leaked state from abandoned handshakes or port scanners
- Add backend connect timeout (30s) and idle timeouts (5 minutes) for dynamic forwards; destroy sockets on timeout and emit logs for timeout events

2026-02-25 - 25.8.1 - fix(allocator)

switch global allocator from tikv-jemallocator to mimalloc

- Replaced tikv-jemallocator with mimalloc in rust/Cargo.toml workspace dependencies.
- Updated rust/crates/rustproxy/Cargo.toml to use mimalloc as a workspace dependency.
- Updated rust/Cargo.lock: added mimalloc and libmimalloc-sys entries and removed tikv-jemallocator and tikv-jemalloc-sys entries.
- Changed the global allocator in crates/rustproxy/src/main.rs from tikv_jemallocator::Jemalloc to mimalloc::MiMalloc.
- Impact: runtime memory allocator is changed which may affect memory usage and performance; no public API changes but recommend testing memory/performance in deployments.

2026-02-24 - 25.8.0 - feat(rustproxy)

use tikv-jemallocator as the global allocator to reduce glibc fragmentation and slow RSS growth; add allocator dependency and enable it in rustproxy, update lockfile, and run tsrust before tests

- Added tikv-jemallocator dependency to rust/Cargo.toml and rust/crates/rustproxy/Cargo.toml
- Enabled tikv_jemallocator as the global allocator in rust/crates/rustproxy/src/main.rs
- Updated rust/Cargo.lock with tikv-jemallocator and tikv-jemalloc-sys entries
- Modified package.json test script to run tsrust before tstest

2026-02-24 - 25.7.10 - fix(rustproxy)

Use cooperative cancellation for background tasks, prune stale caches and metric entries, and switch tests to dynamic port allocation to avoid port conflicts

- Introduce tokio_util::sync::CancellationToken to coordinate graceful shutdown of sampling and renewal tasks; await handles on stop and reset the token so the proxy can be restarted.
- Add safety Drop impls (RustProxy, TcpListenerManager) as a last-resort abort path when stop() is not called.
- MetricsCollector: avoid creating per-IP metric entries when the IP has no active connections; prune orphaned per-IP metric maps during sampling; add tests covering late record_bytes races and pruning behavior.
- Passthrough/ConnectionTracker: remove per-connection record/zombie-scanner complexity, add cleanup_stale_timestamps to prune rate-limit timestamp entries, and add an RAII ConnectionTrackerGuard to guarantee connection_closed is invoked.
- HTTP proxy improvements: add prune_stale_routes and reset_round_robin to clear caches (rate limiters, regex cache, round-robin counters) on route updates.
- Tests: add test/helpers/port-allocator.ts and update many tests to use findFreePorts/assertPortsFree (dynamic ports + post-test port assertions) to avoid flakiness and port collisions in CI.

2026-02-21 - 25.7.9 - fix(tests)

use high non-privileged ports in tests to avoid conflicts and CI failures

- Updated multiple test files to use high-range, non-privileged ports instead of well-known or conflicting ports.
- Files changed: test/test.acme-http01-challenge.ts, test/test.connection-forwarding.ts, test/test.forwarding-regression.ts, test/test.http-port8080-forwarding.ts, test/test.port-mapping.ts, test/test.smartproxy.ts, test/test.socket-handler.ts.
- Notable port remappings: 8080/8081 -> 47730/47731 (and other proxy ports like 47710), 8443 -> 47711, 7001/7002 -> 47712/47713, 9090 -> 47721, 8181/8182 -> 47732/47733, 9999 -> 47780, TEST_PORT_START/PROXY_PORT_START -> 47750/48750, and TEST_SERVER_PORT/PROXY_PORT -> 47770/47771.

2026-02-19 - 25.7.8 - fix(no-changes)

no changes detected; nothing to release

- Current package version: 25.7.7
- Git diff: no changes
- No files modified; no release necessary

2026-02-19 - 25.7.7 - fix(proxy)

restrict PROXY protocol parsing to configured trusted proxy IPs and parse PROXY headers before metrics/fast-path so client IPs reflect the real source

- Add proxy_ips: Vecstd::net::IpAddr to ConnectionConfig with a default empty Vec
- Populate proxy_ips from options.proxy_ips strings in rust/crates/rustproxy/src/lib.rs, parsing each to IpAddr
- Only peek for and parse PROXY v1 headers when the remote IP is contained in proxy_ips (prevents untrusted clients from injecting PROXY headers)
- Move PROXY protocol parsing earlier so metrics and fast-path logic use the effective (real client) IP after PROXY parsing
- If proxy_ips is empty, behavior remains unchanged (no PROXY parsing)

2026-02-19 - 25.7.6 - fix(throughput)

add tests for per-IP connection tracking and throughput history; assert per-IP eviction after connection close to prevent memory leak

- Adds runtime assertions for per-IP TCP connection tracking (m.connections.byIP) while a connection is active
- Adds checks for throughput history (m.throughput.history) to ensure history length and timestamps are recorded
- Asserts that per-IP tracking data is evicted after connection close (byIP.size === 0) to verify memory leak fix
- Reorders test checks so per-IP and history metrics are validated during the active connection and totals are validated after close

2026-02-19 - 25.7.5 - fix(rustproxy)

prune stale per-route metrics, add per-route rate limiter caching and regex cache, and improve connection tracking cleanup to prevent memory growth

- Prune per-route metrics for routes removed from configuration via MetricsCollector::retain_routes invoked during route table updates
- Introduce per-route shared RateLimiter instances (DashMap) with a request-count-triggered periodic cleanup to avoid stale limiters
- Cache compiled URL-rewrite regexes (regex_cache) to avoid recompiling patterns on every request and insert compiled regex on first use
- Improve upstream connection tracking to remove zero-count entries and guard against underflow, preventing unbounded DashMap growth
- Evict per-IP metrics and timestamps when the last connection for an IP closes so per-IP DashMap entries are fully freed
- Add unit tests validating connection tracking cleanup, per-IP eviction, and route-metrics retention behavior

2026-02-19 - 25.7.4 - fix(smart-proxy)

include proxy IPs in smart proxy configuration

- Add proxyIps: this.settings.proxyIPs to proxy options in ts/proxies/smart-proxy/smart-proxy.ts
- Ensures proxy IPs from settings are passed into the proxy implementation (enables proxy IP filtering/whitelisting)

2026-02-16 - 25.7.3 - fix(metrics)

centralize connection-closed reporting via ConnectionGuard and remove duplicate explicit metrics.connection_closed calls

- Removed numerous explicit metrics.connection_closed calls from rust/crates/rustproxy-http/src/proxy_service.rs so connection teardown and byte counting are handled by the connection guard / counting body instead of ad-hoc calls.
- Simplified ConnectionGuard in rust/crates/rustproxy-passthrough/src/tcp_listener.rs: removed the disarm flag and disarm() method so Drop always reports connection_closed.
- Stopped disarming the TCP-level guard when handing connections off to HTTP proxy paths (HTTP/WebSocket/streaming flows) to avoid missing or double-reporting metrics.
- Fixes incorrect/duplicate connection-closed metric emission and ensures consistent byte/connection accounting during streaming and WebSocket upgrades.

2026-02-16 - 25.7.2 - fix(rustproxy-http)

preserve original Host header when proxying and add X-Forwarded-* headers; add TLS WebSocket echo backend helper and integration test for terminate-and-reencrypt websocket

- Preserve the client's original Host header instead of replacing it with backend host:port when proxying requests.
- Add standard reverse-proxy headers: X-Forwarded-For (appends client IP), X-Forwarded-Host, and X-Forwarded-Proto for upstream requests.

- Ensure raw TCP/HTTP upstream requests copy original headers and skip X-Forwarded-* (which are added explicitly).
- Add start_tls_ws_echo_backend test helper to start a TLS WebSocket echo backend for tests.
- Add integration test test_terminate_and_reencrypt_websocket to verify WS upgrade through terminate-and-reencrypt TLS path.
- Rename unused parameter upstream to _upstream in proxy_service functions to avoid warnings.

2026-02-16 - 25.7.1 - fix(proxy)

use TLS to backends for terminate-and-reencrypt routes

- Set upstream.use_tls = true when a route's TLS mode is TerminateAndReencrypt so the proxy re-encrypts to backend servers.
- Add start_tls_http_backend test helper and update integration tests to run TLS-enabled backend servers validating re-encryption behavior.
- Make the selected upstream mutable to allow toggling the use_tls flag during request handling.

2026-02-16 - 25.7.0 - feat(routes)

add protocol-based route matching and ensure terminate-and-reencrypt routes HTTP through the full HTTP proxy; update docs and tests

- Introduce a new 'protocol' match field for routes (supports 'http' and 'tcp') and preserve it through cloning/merging.
- Add Rust integration test verifying terminate-and-reencrypt decrypts TLS and routes HTTP traffic via the HTTP proxy (per-request Host/path routing) instead of raw tunneling.
- Add TypeScript unit tests covering protocol field validation, preservation, interaction with terminate-and-reencrypt, cloning, merging, and matching behavior.
- Update README with a Protocol-Specific Routing section and clarify terminate-and-reencrypt behavior (HTTP routed via HTTP proxy; non-HTTP uses raw TLS-to-TLS tunnel).
- Example config: include health check thresholds (unhealthyThreshold and healthyThreshold) in the sample healthCheck settings.

2026-02-16 - 25.6.0 - feat(rustproxy)

add protocol-based routing and backend TLS re-encryption support

- Introduce optional route_match.protocol ("http" | "tcp") in Rust and TypeScript route types to allow protocol-restricted routing.
- RouteManager: respect protocol field during matching and treat TLS connections without SNI as not matching domain-restricted routes (except wildcard-only routes).
- HTTP proxy: add BackendStream abstraction to unify plain TCP and tokio-rustls TLS backend streams, and support connecting to upstreams over TLS (upstream.use_tls) with an InsecureBackendVerifier for internal/self-signed backends.
- WebSocket and HTTP forwarding updated to use BackendStream so upstream TLS is handled transparently.
- Passthrough listener: perform post-termination protocol detection for TerminateAndReencrypt; route HTTP flows into HttpProxyService and handle non-HTTP as TLS-to-TLS tunnel.
- Add tests for protocol matching, TLS/no-SNI behavior, and other routing edge cases.
- Add rustls and tokio-rustls dependencies (Cargo.toml/Cargo.lock updates).

2026-02-16 - 25.5.0 - feat(tls)

add shared TLS acceptor with SNI resolver and session resumption; prefer shared acceptor and fall back to per-connection when routes specify custom TLS versions

- Add CertResolver that pre-parses PEM certs/keys into CertifiedKey instances for SNI-based lookup and cheap runtime resolution
- Introduce build_shared_tls_acceptor to create a shared ServerConfig with session cache (4096) and Ticketer for session ticket resumption
- Add ArcSwap<Option<TlsAcceptor>> shared_tls_acceptor to tcp_listener for hot-reloadable, pre-built acceptor and update accept loop/handlers to use it
- set_tls_configs now attempts to build and store the shared TLS acceptor, falling back to per-connection acceptors on failure; raw PEM configs are still retained for route-level fallbacks
- Add get_tls_acceptor helper: prefer shared acceptor for performance and session resumption, but build per-connection acceptor when a route requests custom TLS versions

2026-02-16 - 25.4.0 - feat(rustproxy)

support dynamically loaded TLS certificates via loadCertificate IPC and include them in listener TLS configs for rebuilds and hot-swap

- Adds loaded_certs: HashMap<String, TlsCertConfig> to RustProxy to store certificates loaded at runtime
- Merge loaded_certs into tls_configs in rebuild and listener hot-swap paths so dynamically loaded certs are served immediately
- Persist loaded certificates on loadCertificate so future rebuilds include them

2026-02-15 - 25.3.1 - fix(plugins)

remove unused dependencies and simplify plugin exports

- Removed multiple dependencies from package.json to reduce dependency footprint: @push.rocks/lik, @push.rocks/smartacme, @push.rocks/smartdelay, @push.rocks/smartfile, @push.rocks/smartnetwork, @push.rocks/smartpromise, @push.rocks/smartrequest, @push.rocks/smartrx, @push.rocks/smartstring, @push.rocks/taskbuffer, @types/minimatch, @types/ws, pretty-ms, ws
- ts/plugins.ts: stopped importing/exporting node:https and many push.rocks and third-party modules; plugins now only re-export core node modules (without https), tsclass, smartcrypto, smartlog (+ destination-local), smartrust, and minimatch
- Intended effect: trim surface area and remove unused/optional integrations; patch-level change (no feature/API additions)

2026-02-14 - 25.3.0 - feat(smart-proxy)

add background concurrent certificate provisioning with per-domain timeouts and concurrency control

- Add ISmartProxyOptions settings: certProvisionTimeout (ms) and certProvisionConcurrency (default 4)

- Run certProvisionFunction as fire-and-forget background tasks (stores promise on start/route-update and awaited on stop)
- Provision certificates in parallel with a concurrency limit using a new ConcurrencySemaphore utility
- Introduce per-domain timeout handling (default 300000ms) via withTimeout and surface timeout errors as certificate-failed events
- Refactor provisioning into provisionSingleDomain to isolate domain handling, ACME fallback preserved
- Run provisioning outside route update mutex so route updates are not blocked by slow provisioning

2026-02-14 - 25.2.2 - fix(smart-proxy)

start metrics polling before certificate provisioning to avoid blocking metrics collection

- Start metrics polling immediately after Rust engine startup so metrics are available without waiting for certificate provisioning.
- Run certProvisionFunction after startup because ACME/DNS-01 provisioning can hang or be slow and must not block observability.
- Code change in ts/proxies/smart-proxy/smart-proxy.ts: metricsAdapter.startPolling() moved to run before provisionCertificatesViaCallback().

2026-02-14 - 25.2.1 - fix(smartproxy)

no changes detected in git diff

- The provided diff contains no file changes; no code or documentation updates to release.

2026-02-14 - 25.2.0 - feat(metrics)

add per-IP and HTTP-request metrics, propagate source IP through proxy paths, and expose new metrics to the TS adapter

- Add per-IP tracking and IpMetrics in MetricsCollector (active/total connections, bytes, throughput).
- Add HTTP request counters and tracking (total_http_requests, http_requests_per_sec, recent counters and tests).
- Include throughput history (ThroughputSample serialization, retention and snapshotting) and expose history in snapshots.
- Propagate source IP through HTTP and passthrough code paths: CountingBody.record_bytes and MetricsCollector methods now accept source_ip; connection_opened/closed calls include source IP.
- Introduce ForwardMetricsCtx to carry metrics context (collector, route_id, source_ip) into passthrough forwarding routines; update ConnectionGuard to include source_ip.
- TypeScript adapter (rust-metrics-adapter.ts) updated to return per-IP counts, top IPs, per-IP throughput, throughput history mapping, and HTTP request rates/total where available.
- Numerous unit tests added for per-IP tracking, HTTP request tracking, throughput history and ThroughputTracker.history behavior.

2026-02-13 - 25.1.0 - feat(metrics)

add real-time throughput sampling and byte-counting metrics

- Add CountingBody wrapper to count HTTP request and response bytes and report them to MetricsCollector.
- Implement lock-free hot-path byte recording and a cold-path sampling API (sample_all) in MetricsCollector with throughput history and configurable retention (default 3600s).
- Spawn a background sampling task in RustProxy (configurable sample_interval_ms) and tear it down on stop so throughput trackers are regularly sampled.
- Instrument passthrough TCP forwarding and socket-relay paths to record per-chunk bytes (lock-free) so long-lived connections contribute to throughput measurements.
- Wrap HTTP request/response bodies with CountingBody in proxy_service to capture bytes_in/bytes_out and report on body completion; connection_closed handling updated accordingly.
- Expose recent throughput metrics to the TypeScript adapter (throughputRecentIn/Out) and pass metrics settings from the TS SmartProxy into Rust.
- Add http-body dependency and update Cargo.toml/Cargo.lock entries for the new body wrapper usage.
- Add unit tests for MetricsCollector throughput tracking and a new end-to-end throughput test (test.throughput.ts).
- Update test certificates (assets/certs cert.pem and key.pem) used by TLS tests.

2026-02-13 - 25.0.0 - BREAKING CHANGE(certs)

accept a second eventComms argument in certProvisionFunction, add cert provisioning event types, and emit certificate lifecycle events

- Breaking API change: certProvisionFunction signature changed from (domain: string) => Promise<TSmartProxyCertProvisionObject> to (domain: string, eventComms: ICertProvisionEventComms) => Promise<TSmartProxyCertProvisionObject>. Custom provisioners must accept (or safely ignore) the new second argument.
- New types added and exported: ICertProvisionEventComms, ICertificateIssuedEvent, ICertificateFailedEvent.
- smart-proxy now constructs an eventComms channel that allows provisioners to log/warn/error and set expiry date and source for the issued event.
- Emits 'certificate-issued' (domain, expiryDate, source, isRenewal?) on successful provisioning and 'certificate-failed' (domain, error, source) on failures.
- Updated public exports to include the new types so they are available to consumers.
- Removed readme.byte-counting-audit.md (documentation file deleted).

2026-02-13 - 24.0.1 - fix(proxy)

improve proxy robustness: add connect timeouts, graceful shutdown, WebSocket watchdog, and metrics guard

- Add tokio-util CancellationToken to HTTP handlers to support graceful shutdown (stop accepting new requests while letting in-flight requests finish).
- Introduce configurable upstream connect timeout (DEFAULT_CONNECT_TIMEOUT) and return 504 Gateway Timeout on connect timeouts to avoid hanging connections.
- Add WebSocket watchdog with inactivity and max-lifetime checks, activity tracking via AtomicU64, and cancellation-driven tunnel aborts.
- Add ConnectionGuard RAII in passthrough listener to ensure metrics.connection_closed() is called on all exit paths and disarm the guard when handing off to the HTTP proxy.
- Expose HttpProxyService::with_connect_timeout and wire connection timeout from ConnectionConfig into listeners.
- Add tokio-util workspace dependency (CancellationToken) and related code changes across rustproxy-http and rustproxy-passthrough.

2026-02-13 - 24.0.0 - BREAKING CHANGE(smart-proxy)

move certificate persistence to an in-memory store and introduce consumer-managed certStore API; add default self-signed fallback cert and change ACME account handling

- Cert persistence removed from Rust side: CertStore is now an in-memory cache (no filesystem reads/writes). Rust no longer persists or loads certs from disk.
- ACME account credentials are no longer persisted by the library; AcmeClient uses ephemeral accounts only and account persistence APIs were removed.
- TypeScript API changes: removed certificateStore option and added ISmartProxyCertStore + certStore option for consumer-provided persistence (loadAll, save, optional remove).
- Default self-signed fallback certificate added (generateDefaultCertificate) and loaded as '*' unless disableDefaultCert is set.
- SmartProxy now pre-loads certificates from consumer certStore on startup and persists certificates by calling certStore.save() after provisioning.
- provisionCertificatesViaCallback signature changed to accept preloaded domains (prevents re-provisioning), and ACME fallback behavior adjusted with clearer logging.
- Rust cert manager methods made infallible for cache-only operations (load_static/store no longer return errors for cache insertions); removed store-backed load_all/remove/base_dir APIs.
- TCP listener tls_configs concurrency improved: switched to ArcSwap<HashMap<...>> so accept loops see hot-reloads immediately.
- Removed dependencies related to filesystem cert persistence from the tls crate (serde_json, tempfile) and corresponding Cargo.lock changes and test updates.

2026-02-13 - 23.1.6 - fix(smart-proxy)

disable built-in Rust ACME when a certProvisionFunction is provided and improve certificate provisioning flow

- Pass an optional ACME override into buildRustConfig so Rust ACME can be disabled per-run
- Disable Rust ACME when certProvisionFunction is configured to avoid provisioning race conditions
- Normalize routing glob patterns into concrete domain identifiers for certificate provisioning (expand leading-star globs and warn on unsupported patterns)

- Deduplicate domains during provisioning to avoid repeated attempts
- When the callback returns 'http01', explicitly trigger Rust ACME for the route via bridge.provisionCertificate and log success/failure

2026-02-13 - 23.1.5 - fix(smart-proxy)

provision certificates for wildcard domains instead of skipping them

- Removed early continue that skipped domains containing '*' in the domain loop
- Now calls provisionFn for wildcard domains so certificate provisioning can proceed for wildcard hosts
- Fixes cases where wildcard domains never had certificates requested

2026-02-12 - 23.1.4 - fix(tests)

make tests more robust and bump small dependencies

- Bump dependencies: @push.rocks/smartrust ^1.2.1 and minimatch ^10.2.0
- Replace hardcoded ports with named constants (ECHO_PORT, PROXY_PORT, PROXY_PORT_1/2) to avoid collisions between tests
- Add server 'error' handlers and reject listen promises on server errors to prevent silent hangs
- Reduce test timeouts and intervals (shorter test durations, more frequent pings) to speed up test runs
- Ensure proxy is stopped between tests and remove forced process.exit; export tap.start() consistently
- Adjust assertions to match the new shorter ping/response counts

2026-02-12 - 23.1.3 - fix(rustproxy)

install default rustls crypto provider early; detect and skip raw fast-path for HTTP connections and return proper HTTP 502 when no route matches

- Install ring-based rustls crypto provider at startup to prevent panics from instant-acme/hyper-rustls calling ClientConfig::builder() before TLS listeners are initialized
- Add a non-blocking 10ms peek to detect HTTP traffic in the TCP passthrough fast-path to avoid misrouting HTTP and ensure HTTP proxy handles CORS, errors, and request-level routing
- Skip the fast-path and fall back to the HTTP proxy when HTTP is detected (with a debug log)
- When no route matches for detected HTTP connections, send an HTTP 502 Bad Gateway response and close the connection instead of silently dropping it

2026-02-11 - 23.1.2 - fix(core)

use node: scoped builtin imports and add route unit tests

- Replaced bare Node built-in imports (events, fs, http, https, net, path, tls, url, http2, buffer, crypto) with 'node:' specifiers for ESM/bundler compatibility (files updated include ts/plugins.ts, ts/core/models/socket-types.ts, ts/core/utils/enhanced-connection-pool.ts, ts/core/utils/socket-tracker.ts, ts/protocols/common/fragment-handler.ts, ts/protocols/tls/sni/client-hello-parser.ts, ts/protocols/tls/sni/sni-extraction.ts, ts/protocols/websocket/utils.ts, ts/tls/sni/sni-handler.ts).
- Added new unit tests (test/test.bun.ts and test/test.deno.ts) covering route helpers, validators, matching, merging and cloning to improve test coverage.

2026-02-11 - 23.1.1 - fix(rust-proxy)

increase rust proxy bridge maxPayloadSize to 100 MB and bump dependencies

- Set maxPayloadSize to 100 * 1024 * 1024 (100 MB) in ts/proxies/smart-proxy/rust-proxy-bridge.ts to support large route configs
- Bump devDependency @types/node from ^25.2.2 to ^25.2.3
- Bump dependency @push.rocks/smartrust from ^1.1.1 to ^1.2.0

2026-02-10 - 23.1.0 - feat(rust-bridge)

integrate tsrust to build and locate cross-compiled Rust binaries; refactor rust-proxy bridge to use typed IPC and streamline process handling; add @push.rocks/smartrust and update build/dev dependencies

- Add tsrust to the build script and include dist_rust candidates when locating the Rust binary (enables cross-compiled artifacts produced by tsrust).
- Remove the old rust-binary-locator and refactor rust-proxy-bridge to use explicit, typed IPC command definitions and improved process spawn/cleanup logic.
- Introduce @push.rocks/smartrust for type-safe JSON IPC and export it via plugins; update README with expanded metrics documentation and change initialDataTimeout default from 60s to 120s.
- Add rust/.cargo/config.toml with aarch64 linker configuration to support cross-compilation for arm64.
- Bump several devDependencies and runtime dependencies (e.g. @git.zone/tsbuild, @git.zone/tstest, @push.rocks/smartserve, @push.rocks/taskbuffer, ws, minimatch, etc.).
- Update runtime message guiding local builds to use 'pnpm build' (tsrust) instead of direct cargo invocation.

2026-02-09 - 23.0.0 - BREAKING CHANGE(proxies/nftables-proxy)

remove nftables-proxy implementation, models, and utilities from the repository

- Deleted nftables-proxy module files under ts/proxies/nftables-proxy (index, models, utils, command executor, validators, etc.)
- Removed nftables-proxy exports from ts/index.ts and ts/proxies/index.ts
- Updated smart-proxy types to drop dependency on nftables proxy models
- Breaking change: any consumers importing nftables-proxy will no longer find those exports; update imports or install/use the extracted/alternative package if applicable

2026-02-09 - 22.6.0 - feat(smart-proxy)

add socket-handler relay, fast-path port-only forwarding, metrics and bridge improvements, and various TS/Rust integration fixes

- Add Unix-domain socket relay for socket-handler routes so Rust can hand off matched connections to TypeScript handlers (metadata JSON + initial bytes, relay implementation in Rust and SocketHandlerServer in TS).
- Implement fast-path port-only forwarding in the TCP accept/handler path to forward simple non-TLS, port-only routes immediately without peeking at client data (improves server-speaks-first protocol handling).
- Use ArcSwap for route manager hot-reload visibility in accept loops and share socket_handler_relay via Arc<RwLock> so listeners see relay path updates immediately.
- Enhance SNI/HTTP parsing: add extract_http_path and extract_http_host to aid domain/path matching from initial data.
- Improve RustProxy shutdown/kill handling: remove listeners, reject pending requests, destroy stdio pipes and unref process to avoid leaking handles.
- Enhance Rust <-> TS metrics bridge and adapter: add immediate poll(), map Rust JSON fields to IMetrics (per-route active/throughput/totals), and use safer polling/unref timers.
- SocketHandlerServer enhancements: track active sockets, destroy on stop, pause/resume to prevent data loss, support async socketHandler callbacks and dynamic function-based target forwarding (resolve host/port functions and forward).
- TypeScript smart-proxy lifecycle tweaks: only set bridge relay after Rust starts, guard unexpected-exit emission when intentionally stopping, stop polling and remove listeners on stop, add stopping flag.
- Misc: README and API ergonomics updates (nft proxy option renames and config comments), various test updates to use stable http.request helper, adjust timeouts/metrics sampling and assertions, and multiple small bugfixes in listeners, timeouts and TLS typings.

2026-02-09 - 22.5.0 - feat(rustproxy)

introduce a Rust-powered proxy engine and workspace with core crates for proxy functionality, ACME/TLS support, passthrough and HTTP proxies, metrics, nftables integration, routing/security, management IPC, tests, and README updates

- Add Rust workspace and multiple crates: rustproxy, rustproxy-config, rustproxy-routing, rustproxy-tls, rustproxy-passthrough, rustproxy-http, rustproxy-nftables, rustproxy-metrics, rustproxy-security
- Implement ACME integration (instant-acme) and an HTTP-01 challenge server with certificate lifecycle management
- Add TLS management: cert store, cert manager, SNI resolver, TLS acceptor/connector and certificate hot-swap support
- Implement TCP/TLS passthrough engine with ClientHello SNI parsing, PROXY v1 support, connection tracking and bidirectional forwarder

- Add Hyper-based HTTP proxy components: request/response filtering, CORS, auth, header templating and upstream selection with load balancing
- Introduce metrics (throughput tracker, metrics collector) and log deduplication utilities
- Implement nftables manager and rule builder (safe no-op behavior when not running as root)
- Add route types, validation, helpers, route manager and matchers (domain/path/header/ip)
- Provide management IPC (JSON over stdin/stdout) for TypeScript wrapper control (start/stop/add/remove ports, load certificates, etc.)
- Include extensive unit and integration tests, test helpers, and an example Rust config.json
- Update README to document the Rust-powered engine, new features and rustBinaryPath lookup

2026-01-31 - 22.4.2 - fix(tests)

shorten long-lived connection test timeouts and update certificate metadata timestamps

- Reduced test timeouts from 65–70s to 60s and shortened internal waits from ~61–65s to 55s to ensure tests complete within CI runner limits (files changed: test/test.long-lived-connections.ts, test/test.websocket-keepalive.node.ts).
- Updated log message to reflect the new 55s wait.
- Bumped certificate metadata timestamps in certs/static-route/meta.json (issueDate, savedAt, expiryDate).

2026-01-30 - 22.4.1 - fix(smartproxy)

improve certificate manager mocking in tests, enhance IPv6 validation, and record initial bytes for connection metrics

- Add createMockCertManager and update tests to fully mock createCertificateManager to avoid real ACME calls and make provisioning deterministic
- Record initial data chunk bytes in route-connection-handler and report them to metricsCollector.recordBytes to improve metrics accuracy
- Improve IPv6 validation regex to accept IPv6-mapped IPv4 addresses (::ffff:x.x.x.x)
- Add/set missing mock methods used in tests (setRoutes, generateConnectionId, trackConnectionByRoute, validateAndTrackIP) and small test adjustments (route names, port changes)

- Make test robustness improvements: wait loops for connection cleanup, increase websocket keepalive timeout, and other minor test fixes/whitespace cleanups
- Update certificate meta timestamps (test fixtures)

2026-01-30 - 22.4.0 - feat(smart-proxy)

calculate when SNI is required for TLS routing and allow session tickets for single-target passthrough routes; add tests, docs, and npm metadata updates

- Add calculateSniRequirement() and isWildcardOnly() to determine when SNI is required for routing decisions
- Use the new calculation to allow TLS session tickets for single-route passthrough or wildcard-only domains and block them when SNI is required
- Replace previous heuristic in route-connection-handler with the new SNI-based logic
- Add comprehensive unit tests (test/test.sni-requirement.node.ts) covering multiple SNI scenarios
- Update readme.hints.md with Smart SNI Requirement documentation and adjust troubleshooting guidance
- Update npmextra.json keys, add release registries and adjust tsdoc/CI metadata

2026-01-30 - 22.3.0 - feat(docs)

update README with installation, improved feature table, expanded quick-start, ACME/email example, API options interface, and clarified licensing/trademark text

- Added Installation section with npm/pnpm commands
- Reformatted features into a markdown table for clarity
- Expanded Quick Start example and updated ACME email placeholder
- Added an ISmartProxyOptions interface example showing acme/defaults/behavior options
- Clarified license file path and expanded trademark/legal wording
- Minor editorial and formatting improvements throughout the README

2026-01-30 - 22.2.0 - feat(proxies)

introduce nftables command executor and utilities, default certificate provider, expanded route/socket helper modules, and security improvements

- Added NftCommandExecutor with retry, temp-file support, sync execution, availability and conntrack checks.
- Refactored NfTablesProxy to use executor/utils (normalizePortSpec, validators, port normalizer, IP family filtering) and removed inline command/validation code.
- Introduced DefaultCertificateProvider to replace the deprecated CertificateManager; HttpProxy now uses DefaultCertificateProvider (CertificateManager exported as deprecated alias for compatibility).
- Added extensive route helper modules (http, https, api, load-balancer, nftables, dynamic, websocket, security, socket handlers) to simplify route creation and provide reusable patterns.
- Enhanced SecurityManagers: centralized security utilities (normalizeIP, isIPAuthorized, parseBasicAuthHeader, cleanup helpers), added validateAndTrackIP and JWT token verification, better IP normalization and rate tracking.
- Added many utility modules under ts/proxies/nftables-proxy/utils (command executor, port spec normalizer, rule validator) and exposed them via barrel export.

2025-12-09 - 22.1.1 - fix(tests)

Normalize route configurations in tests to use name (remove id) and standardize route names

- Removed deprecated id properties from route configurations in multiple tests and rely on the name property instead
- Standardized route.name values to kebab-case / lowercase (examples: 'tcp-forward', 'tls-passthrough', 'domain-a', 'domain-b', 'test-forward', 'nftables-test', 'regular-test', 'forward-test', 'test-forward', 'tls-test')
- Added explicit names for inner and outer proxies in proxy-chain-cleanup test ('inner-backend', 'outer-frontend')
- Updated certificate metadata timestamps in certs/static-route/meta.json

2025-12-09 - 22.1.0 - feat(smart-proxy)

Improve connection/rate-limit atomicity, SNI parsing, HttpProxy & ACME orchestration, and routing utilities

- Fix race conditions for per-IP connection limits by introducing atomic validate-and-track flow (SecurityManager.validateAndTrackIP) and propagating connectionId for atomic tracking.

- Add connection-manager createConnection options (connectionId, skipIpTracking) and avoid double-tracking IPs when validated atomically.
- RouteConnectionHandler now generates connection IDs earlier and uses atomic IP validation to prevent concurrent connection bypasses; cleans up IP tracking on global-limit rejects.
- Enhanced TLS SNI extraction and ClientHello parsing: robust fragmented ClientHello handling, PSK-based SNI extraction for TLS 1.3 resumption, tab-reactivation heuristics and improved logging (new client-hello-parser and sni-extraction modules).
- HttpProxy integration improvements: HttpProxyBridge initialized/synced from SmartProxy, forwardToHttpProxy forwards initial data and preserves client IP via CLIENT_IP header, robust handling of client disconnects during setup.
- Certificate manager (SmartCertManager) improvements: better ACME initialization sequence (deferred provisioning until ports are bound), improved challenge route add/remove handling, custom certificate provisioning hook, expiry handling fallback behavior and safer error messages for port conflicts.
- Route/port orchestration refactor (RouteOrchestrator): port usage mapping, safer add/remove port sequences, NFTables route lifecycle updates and certificate manager recreation on route changes.
- PortManager now refcounts ports and reuses existing listeners instead of rebinding; provides helpers to add/remove/update multiple ports and improved error handling for EADDRINUSE.
- Connection cleanup, inactivity and zombie detection hardened: batched cleanup queue, optimized inactivity checks, half-zombie detection and safer shutdown workflows.
- Metrics, routing helpers and validators: SharedRouteManager exposes expandPortRange/getListeningPorts, route helpers add convenience HTTPS/redirect/loadbalancer builders, route-validator domain rules relaxed to allow 'localhost', '*' and IPs, and tests updated accordingly.
- Tests updated to reflect behavioral changes (connection limit checks adapted to detect closed/ reset connections, HttpProxy integration test skipped in unit suite to avoid complex TLS setup).

2025-12-09 - 22.0.0 - BREAKING CHANGE(smart-proxy/utils/route-validator)

Consolidate and refactor route validators; move to class-based API and update usages

Replaced legacy route-validators.ts with a unified route-validator.ts that provides a class-based RouteValidator plus the previous functional API (isValidPort, isValidDomain, validateRouteMatch, validateRouteAction, validateRouteConfig, validateRoutes, hasRequiredPropertiesForAction,

assertValidRoute) for backwards compatibility. Updated utils exports and all imports/tests to reference the new module. Also switched static file loading in certificate manager to use SmartFileFactory.nodeFs(), and added @push.rocks/smartserve to devDependencies.

- Rename and consolidate validator module: route-validators.ts removed; route-validator.ts added with RouteValidator class and duplicated functional API for compatibility.
- Updated exports in ts/proxies/smart-proxy/utils/index.ts and all internal imports/tests to reference './route-validator.js' instead of './route-validators.js'.
- Certificate manager now uses plugins.smartfile.SmartFileFactory.nodeFs() to load key/cert files (safer factory usage instead of direct static calls).
- Added @push.rocks/smartserve to devDependencies in package.json.
- Because the validator filename and some import paths changed, this is a breaking change for consumers importing the old module path.

2025-08-19 - 21.1.7 - fix(route-validator)

Relax domain validation to accept 'localhost', prefix wildcards (e.g. *example.com) and IP literals; add comprehensive domain validation tests

- Allow 'localhost' as a valid domain pattern in route validation
- Support prefix wildcard patterns like '*example.com*' *in addition to* '.example.com'
- Accept IPv4 and IPv6 literal addresses in domain validation
- Add test coverage: new test/test.domain-validation.ts with many real-world and edge-case patterns

2025-08-19 - 21.1.6 - fix(ip-utils)

Fix IP wildcard/shorthand handling and add validation test

- Support shorthand IPv4 wildcard patterns (e.g. '10.', '*192.168.*') by expanding them to full 4-octet patterns before matching
- Normalize and expand patterns in IpUtils.isGlobIPMatch and SharedSecurityManager IP checks to ensure consistent minimatch comparisons
- Relax route validator wildcard checks to accept 1-4 octet wildcard specifications for IPv4 patterns
- Add test harness test-ip-validation.ts to exercise common wildcard/shorthand IP patterns

2025-08-19 - 21.1.5 - fix(core)

Prepare patch release: documentation, tests and stability fixes (metrics, ACME, connection cleanup)

- Byte counting and throughput: per-route and per-IP throughput trackers with per-second sampling; removed double-counting and improved sampling buffers for accurate rates
- HttpProxy and forwarding: Ensure metricsCollector.recordBytes() is called in forwarding paths so throughput is recorded reliably
- ACME / Certificate Manager: support for custom certProvisionFunction with configurable fallback to ACME (http01) and improved challenge route lifecycle
- Connection lifecycle and cleanup: improved lifecycle component timer/listener cleanup, better cleanup queue batching and zombie/half-zombie detection
- Various utilities and stability improvements: enhanced IP utils, path/domain matching improvements, safer socket handling and more robust fragment/ClientHello handling
- Tests and docs: many test files and readme.hints.md updated with byte-counting audit, connection cleanup and ACME guidance

2025-08-14 - 21.1.4 - fix(security)

Critical security and stability fixes

- Fixed critical socket.emit override vulnerability that was breaking TLS connections
- Implemented comprehensive socket cleanup with new socket tracker utility
- Improved code organization by extracting RouteOrchestrator from SmartProxy
- Fixed IPv6 loopback detection for proper IPv6 support
- Added memory bounds to prevent unbounded collection growth
- Fixed certificate manager race conditions with proper synchronization
- Unreferenced long-lived timers to prevent process hanging
- Enhanced route validation for socket-handler actions
- Fixed header parsing when extractFullHeaders option is enabled

2025-07-22 - 21.1.1 - fix(detection)

Fix SNI detection in TLS detector

- Restored proper TLS detector implementation with ClientHello parsing
- Fixed imports to use new protocols module locations
- Added missing detectWithContext method for fragmented detection

- Fixed method names to match BufferAccumulator interface
- Removed unused import readUInt24BE

2025-07-21 - 21.1.0 - feat(protocols)

Refactor protocol utilities into centralized protocols module

- Moved TLS utilities from `ts/tls/` to `ts/protocols/tls/`
- Created centralized protocol modules for HTTP, WebSocket, Proxy, and TLS
- Core utilities now delegate to protocol modules for parsing and utilities
- Maintains backward compatibility through re-exports in original locations
- Improves code organization and separation of concerns

2025-07-22 - 21.0.0 - BREAKING_CHANGE(forwarding)

Remove legacy forwarding module

- Removed the `forwarding` namespace export from main index
- Removed TForwardingType and all forwarding handlers
- Consolidated route helper functions into route-helpers.ts
- All functionality is now available through the route-based system
- MIGRATION: Replace `import { forwarding } from '@push.rocks/smartproxy'` with direct imports of route helpers

2025-07-21 - 20.0.2 - fix(docs)

Update documentation to improve clarity

- Enhanced readme with clearer breaking change warning for v20.0.0
- Fixed example email address from ssl@bleu.de to ssl@example.com
- Added load balancing and failover features to feature list
- Improved documentation structure and examples

2025-07-20 - 20.0.1 - BREAKING_CHANGE(routing)

Refactor route configuration to support multiple targets

- Changed route action configuration from single `target` to `targets` array
- Enables load balancing and failover capabilities with multiple upstream targets
- Updated all test files to use new `targets` array syntax
- Automatic certificate metadata refresh

2025-06-01 - 19.5.19 - fix(smartproxy)

Fix connection handling and improve route matching edge cases

- Enhanced cleanup logic to prevent connection accumulation under rapid retry scenarios
- Improved matching for wildcard domains and path parameters in the route configuration
- Minor refactoring in async utilities and internal socket handling for better performance
- Updated test suites and documentation for clearer configuration examples

2025-05-29 - 19.5.3 - fix(smartproxy)

Fix route security configuration location and improve ACME timing tests and socket mock implementations

- Move route security from action.security to the top-level route.security to correctly enforce IP allow/block lists (addresses failing in test.route-security.ts)
- Update readme.problems.md to document the routing security configuration issue with proper instructions
- Adjust certificate metadata in certs/static-route/meta.json with updated timestamps
- Update test.acme-timing.ts to export default tap.start() instead of tap.start() to ensure proper parsing

- Improve socket simulation and event handling mocks in test.http-fix-verification.ts and test.http-forwarding-fix.ts to more reliably mimic net.Socket behavior
- Minor adjustments in multiple test files to ensure proper port binding, race condition handling and route lookups (e.g. getRoutesForPort implementation)

2025-05-29 - 19.5.2 - fix(test)

Fix ACME challenge route creation and HTTP request parsing in tests

- Replaced the legacy ACME email 'test@example.com' with 'test@acmetest.local' to avoid forbidden domain issues.
- Mocked the CertificateManager in test/test.acme-route-creation to simulate immediate ACME challenge route addition.
- Adjusted updateRoutes callback to capture and verify challenge route creation.
- Enhanced the HTTP request parsing in socket handler by capturing and asserting parsed request details (method, path, headers).

2025-05-29 - 19.5.1 - fix(socket-handler)

Fix socket handler race condition by differentiating between async and sync handlers. Now, async socket handlers complete their setup before initial data is emitted, ensuring that no data is lost. Documentation and tests have been updated to reflect this change.

- Added detailed explanation in readme.hints.md about the race condition issue, root cause, and solution implementation.
- Provided a code snippet that checks if the socket handler returns a Promise and waits for its resolution before emitting initial data.
- Updated tests (test.socket-handler-race.ts, test.socket-handler.simple.ts, test.socket-handler.ts) to verify correct behavior of async handlers.

2025-05-28 - 19.5.0 - feat(socket-handler)

Add socket-handler support for custom socket handling in SmartProxy

- Introduce new action type 'socket-handler' in IRouteAction to allow users to provide a custom socket handler function.
- Update the RouteConnectionHandler to detect 'socket-handler' actions and invoke the handler with the raw socket, giving full control to the user.
- Provide optional context (such as route configuration, client IP, and port) to the socket handler if needed.
- Add helper functions in route-helpers for creating socket handler routes and common patterns like echo, proxy, and line-based protocols.
- Include a detailed implementation plan and usage examples in readme.plan.md.

2025-05-28 - 19.4.3 - fix(smartproxy)

Improve port binding intelligence and ACME challenge route management; update route configuration tests and dependency versions.

- Bumped dev dependency versions in package.json (tsbuild from ^2.5.1 to ^2.6.4, tstest from ^1.9.0 to ^2.3.1, @types/node updated, smartfile from ^11.2.0 to ^11.2.5, smartlog from ^3.1.7 to ^3.1.8)
- Removed readme.plan.md containing legacy development plan information
- Normalized route configuration properties across tests (using 'ports' and 'domains' instead of legacy 'port' or 'domain')
- Enhanced PortManager with reference counting and smarter port conflict detection to avoid redundant bindings
- Refined ACME challenge route integration to merge with existing port bindings and improve error handling
- Adjusted test expectations (e.g. using toEqual instead of toBe, and improved timeout handling) to align with current API changes

2025-05-20 - 19.4.2 - fix(dependencies)

Update dependency versions: upgrade @types/node to ^22.15.20 and @push.rocks/smartlog to ^3.1.7 in package.json

- Bump @types/node from ^22.15.19 to ^22.15.20
- Bump @push.rocks/smartlog from ^3.1.3 to ^3.1.7

2025-05-20 - 19.4.1 - fix(smartproxy)

Bump @push.rocks/smartlog to ^3.1.3 and improve ACME port binding behavior in SmartProxy

- Updated package.json to use @push.rocks/smartlog version ^3.1.3
- Enhanced tests (test.http-port8080-simple.ts) to verify improved port binding intelligence for ACME challenge routes
- Ensured that existing port listeners are reused and not re-bound when updating routes

2025-05-20 - 19.4.0 - feat(certificate-manager, smart-proxy)

Improve port binding intelligence for ACME challenges

- Reordered SmartProxy initialization flow to bind ports before initializing the certificate manager
- Enhanced port binding logic to better handle ACME challenge routes
- Improved error detection and reporting for port binding conflicts
- Added better diagnostics for ACME challenge port issues
- Made route updates more intelligent with detailed port tracking
- Fixed race condition where ACME routes were added before port 80 was bound
- Added special handling for ACME port conflicts with improved error messages

2025-05-20 - 19.3.14 - fix(certificate-manager, smart-proxy)

Add error handling around logger calls in route update callback

- Added try/catch blocks around logger calls in certificate-manager.ts
- Added try/catch blocks around logger calls in smart-proxy.ts related to route updates
- Provided fallback to console.log when logger fails
- Ensured core route update functionality continues to work even if logging fails

2025-05-20 - 19.3.13 - fix(port-manager, certificate-manager)

Improve port binding and ACME challenge route integration in SmartProxy

- Added reference counting in PortManager so that routes sharing the same port reuse the existing binding.
- Enhanced error handling to distinguish internal port conflicts from external ones, with more descriptive messages.
- Adjusted ACME challenge route addition to merge with existing port bindings when port is already in use.
- Refactored updateRoutes to release orphaned ports and bind only new required ports, minimizing rebinding operations.
- Improved certificate-manager logic to provide clearer error notifications when ACME port conflicts occur.

2025-05-19 - 19.3.12 - fix(tests)

Update test mocks to include provisionAllCertificates methods in certificate manager stubs and related objects.

- Added async provisionAllCertificates functions to several test mocks (e.g. in test.port80-management.node.ts, test.route-callback-simple.ts, test.route-update-callback.node.ts, and test.simple-acme-mock.ts) to simulate ACME certificate provisioning.
- Enhanced logging and port-add history debugging for ACME challenge port addition.

2025-05-19 - 19.3.11 - fix(logger)

Replace raw console logging calls with structured logger usage across certificate management, connection handling, and route processing for improved observability.

- Replaced console.log, console.warn, and console.error in SmartCertManager with logger.log for more consistent logging.
- Updated ConnectionManager and RouteConnectionHandler to log detailed connection events using a structured logger.
- Enhanced logging statements with contextual metadata such as connection IDs, remote IPs, target information, and component identifiers.
- Standardized log output across proxy modules to aid in debugging and monitoring.

2025-05-19 - 19.3.10 - fix(certificate-manager, smart-proxy)

Fix race condition in ACME certificate provisioning and refactor certificate manager initialization to defer provisioning until after port listeners are active

- Removed superfluous provisionCertificatesAfterPortsReady method
- Made provisionAllCertificates public so that SmartProxy.start() calls it after ports are listening
- Updated SmartProxy.start() to wait for port setup (via PortManager) before triggering certificate provisioning
- Improved ACME HTTP-01 challenge timing so that port 80 (or configured ACME port) is guaranteed to be ready
- Updated documentation (changelog and Acme timing docs) and tests to reflect the change

2025-05-19 - 19.3.10 - refactor(certificate-manager, smart-proxy)

Simplify certificate provisioning code by removing unnecessary wrapper method

- Removed superfluous SmartCertManager.provisionCertificatesAfterPortsReady() method
- Made SmartCertManager.provisionAllCertificates() public instead

- Updated SmartProxy.start() to call provisionAllCertificates() directly
- Updated documentation and tests to reflect the change
- No functional changes, just code simplification

2025-05-19 - 19.3.9 - fix(certificate-manager, smart-proxy)

Fix ACME certificate provisioning timing to ensure ports are listening first

- Fixed race condition where certificate provisioning would start before ports were listening
- Modified SmartCertManager.initialize() to defer certificate provisioning
- Added SmartCertManager.provisionCertificatesAfterPortsReady() for delayed provisioning
- Updated SmartProxy.start() to call certificate provisioning after ports are ready