Unified Hosting Platform - Product Requirements Document (PRD)

Enterprise-Grade Hosting Control Panel with Billing Automation & Advanced Security

Document Version: 2.0 (Unified Edition)
Date: November 4, 2025
Project Code Name: Hetzner Cloud Commander
Status: Production Ready Specification
License: Proprietary / Open Source Hybrid Model

📋 Table of Contents

---

1. Executive Summary

1.1 Project Vision

The **Unified Hosting Platform** is a next-generation, enterprise-grade hosting control panel that combines:

- **Billing Automation** (WHMCS-inspired features)
- **Control Panel Management** (Enhance Panel + CloudPanel capabilities)
- **Advanced Security** (CloudLinux-inspired PHP hardening, enterprise WAF, firewall)
- **Cloud-Native Architecture** (Built for Hetzner Cloud infrastructure)
- **Modern Tech Stack** (RUST, Go, HTMX for maximum performance)

1.2 Core Differentiators

Feature	Our Solution	Traditional Panels
Performance	RUST core + Go microservices	PHP/Python monoliths
Security	Multi-layer hardening, ML-powered WAF	Basic firewalls
UI/UX	HTMX (no heavy JS frameworks)	jQuery/Angular/React
Automation	n8n + Ansible + Bash + Python	Limited scripting
Cost	Optimized for Hetzner (20-40% savings)	Cloud-agnostic
Architecture	Microservices + Event-driven	Monolithic
Licensing	Unlimited servers, one-time fee	Per-server licensing

1.3 Target Market

Primary:

- Web hosting companies (shared, VPS, cloud, dedicated)
- Managed service providers (MSPs)
- Digital agencies with hosting divisions
- Reseller hosting businesses

Secondary:

- Enterprise IT departments
- SaaS companies needing internal hosting management
- DevOps teams requiring infrastructure automation

1.4 Success Metrics

Technical KPIs:

- API Response Time: < 50ms (p95)
- Panel Load Time: < 500ms
- Server Provisioning: < 60 seconds
- System Resource Usage: < 10% CPU, < 500MB RAM (idle)
- Uptime SLA: 99.95%

Business KPIs:

- Customer Onboarding: < 5 minutes
- Support Ticket Reduction: 40% (via automation)
- Billing Automation: 95% of invoices auto-generated
- Security Incident Rate: < 0.1% of hosted sites

---

2. Technology Stack & Architecture

2.1 Core Technology Stack

Backend Services


yaml

```yaml
Core Engine:
  Language: RUST 1.75+
  Framework: Actix-web 4.x / Axum 0.7+
  Purpose: High-performance core services
  Components:
    - HTTP/HTTPS server
    - Resource monitoring
    - Security engine
    - System provisioning

Microservices:
  Language: Go 1.22+
  Framework: Fiber v2 / Gin
  Purpose: API services and business logic
  Components:
    - Billing service
    - Email service
    - DNS service
    - Backup service
    - Domain service
    - Support ticket service

Database Layer:
  Primary: PostgreSQL 16+
  Cache: Redis 7.2+
  Time-Series: TimescaleDB (for metrics)
  Search: Meilisearch 1.6+
  Message Queue: NATS 2.10+ / RabbitMQ 3.12+

Storage:
  Object Storage: Hetzner Object Storage (S3-compatible)
  Block Storage: Hetzner Volumes
  Local: ext4/xfs on NVMe
```

Frontend Stack


yaml

Frontend:
  Primary: HTMX 1.9+
  Templating: Go templates / Tera (RUST)
  CSS Framework: Tailwind CSS 3.x
  Icons: Lucide Icons
  Charts: Chart.js / Apache ECharts

Philosophy:
  - Server-side rendering (SSR)
  - Progressive enhancement
  - No heavy JavaScript frameworks
  - < 50KB total JS bundle
  - Lazy loading for non-critical resources

Automation Stack


yaml

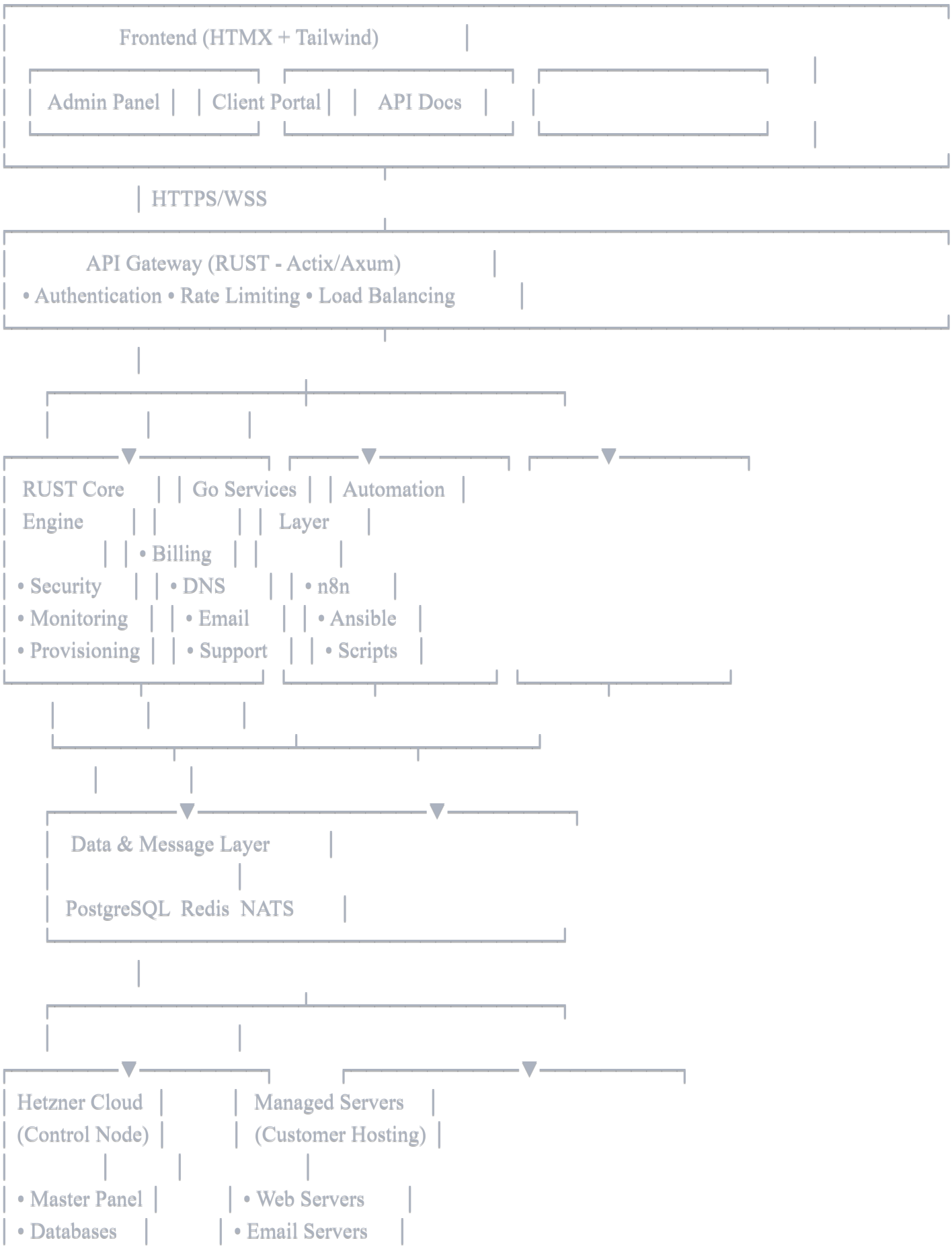
Infrastructure Automation:
  Tool: Ansible 2.15+
  Purpose: Server provisioning, configuration management
  Playbooks:
    - Server initial setup
    - Software installation
    - Security hardening
    - Service deployment
    - Update management

Workflow Automation:
  Tool: n8n 1.x (self-hosted)
  Purpose: Business process automation
  Workflows:
    - Customer onboarding
    - Invoice generation
    - Payment processing
    - Service provisioning
    - Email notifications
    - Webhook handlers

System Scripting:
  Primary: Bash 5.x
  Secondary: Python 3.11+
  Purpose:
    - System maintenance scripts
    - Backup automation
    - Log rotation
    - Resource monitoring
    - Custom integrations

2.2 System Architecture

High-Level Architecture


Frontend (HTMX + Tailwind)
Admin Panel
Client Portal
API Docs
HTTPS/WSS
API Gateway (RUST - Actix/Axum)
• Authentication • Rate Limiting • Load Balancing
RUST Core Engine
Go Services
• Billing
• DNS
• Email
Automation Layer
• n8n
• Ansible
• Scripts
• Security
• Monitoring
• Provisioning
• Support
Data & Message Layer
PostgreSQL Redis NATS
Hetzner Cloud (Control Node)
• Master Panel
• Databases
Managed Servers (Customer Hosting)
• Web Servers
• Email Servers

Microservices Breakdown

yaml

Service: billing-service
  Language: Go
  Port: 8001
  Responsibilities:
    - Invoice management
    - Payment processing
    - Subscription management
    - Credit management
    - Tax calculations
  Database: PostgreSQL
  Cache: Redis

Service: dns-service
  Language: Go
  Port: 8002
  Responsibilities:
    - DNS zone management
    - Record CRUD operations
    - DNSSEC management
    - DNS templates
  Database: PostgreSQL + PowerDNS

Service: email-service
  Language: Go
  Port: 8003
  Responsibilities:
    - Email account management
    - Mailbox quotas
    - Email routing
    - SMTP/IMAP/POP3 management
  Database: PostgreSQL + Dovecot backend

Service: backup-service
  Language: RUST
  Port: 8004
  Responsibilities:
    - Backup scheduling
    - Incremental backups
    - Restoration
    - Retention management
  Storage: Hetzner Object Storage

Service: monitoring-service
  Language: RUST
  Port: 8005
  Responsibilities:
    - Resource metrics collection
    - Alert management
    - Health checks
    - Performance analytics
  Database: TimescaleDB

Service: security-service
  Language: RUST
  Port: 8006
  Responsibilities:
    - WAF rule engine
    - Firewall management
    - Malware scanning
    - Intrusion detection
  Database: Redis (cache) + PostgreSQL

Service: provisioning-service
  Language: RUST
  Port: 8007
  Responsibilities:
    - Server provisioning
    - Service deployment
    - Configuration management
  Integration: Ansible API

Service: support-service
  Language: Go
  Port: 8008
  Responsibilities:
    - Ticket management
    - Knowledge base
    - Live chat
    - Announcements
  Database: PostgreSQL + Meilisearch

2.3 Development Principles

Security First:

- Secure by default configurations
- Principle of least privilege
- Zero-trust architecture
- Regular security audits
- Automated vulnerability scanning

Performance Optimized:

- RUST for CPU-intensive operations
- Go for I/O-bound services
- Connection pooling
- Database query optimization
- Redis caching strategy
- CDN integration

Scalability:

- Horizontal scaling for all services
- Stateless design
- Event-driven architecture
- Message queues for async operations
- Database replication and sharding

Maintainability:

- Comprehensive documentation
- Unit testing (>80% coverage)
- Integration testing
- CI/CD pipelines
- Automated deployments

---

3. System Requirements

3.1 Control Node (Master Panel)

Hetzner Cloud Server Specifications:

yaml

Minimum (Development/Testing):
Instance: CPX21
CPU: 3 vCPU (AMD EPYC)
RAM: 4 GB
Storage: 80 GB NVMe
Network: 20 TB traffic
Monthly Cost: ~€8.50

Recommended (Production - Small):
Instance: CPX31
CPU: 4 vCPU (AMD EPYC)
RAM: 8 GB
Storage: 160 GB NVMe
Network: 20 TB traffic
Monthly Cost: ~€16.50

Recommended (Production - Medium):
Instance: CPX41
CPU: 8 vCPU (AMD EPYC)
RAM: 16 GB
Storage: 240 GB NVMe
Network: 20 TB traffic
Monthly Cost: ~€31.50

Recommended (Production - Large):
Instance: CPX51
CPU: 16 vCPU (AMD EPYC)
RAM: 32 GB
Storage: 360 GB NVMe
Network: 20 TB traffic
Monthly Cost: ~€61.50

Operating System:

- Ubuntu 24.04 LTS (Primary)
- Debian 12 (Secondary)
- Rocky Linux 9 (Optional)

3.2 Managed Hosting Servers

Server Types:

yaml

```yaml
Shared Hosting Server:
  Instance: CPX31 or higher
  CPU: 4+ vCPU
  RAM: 8+ GB
  Storage: 160+ GB NVMe
  Capacity: 50-100 accounts

VPS Hosting Server:
  Instance: CPX41 or higher
  CPU: 8+ vCPU
  RAM: 16+ GB
  Storage: 240+ GB NVMe
  Capacity: 10-25 VPS instances

Dedicated Resources:
  Instance: CCX33 or higher
  CPU: 8 dedicated vCPU
  RAM: 32 GB
  Storage: 240 GB NVMe + Volumes
```

3.3 Additional Infrastructure

Database Server (Separate):

- Hetzner CPX31 or higher
- PostgreSQL 16+ with replication
- Automated backups to Object Storage

Email Server (Separate):

- Hetzner CPX21 or higher
- Postfix + Dovecot
- SpamAssassin + ClamAV

DNS Servers (Separate - Redundant):

- 2x Hetzner CX22 instances
- PowerDNS with PostgreSQL backend
- GeoDNS support

Backup Storage:

- Hetzner Object Storage (S3-compatible)
- Unlimited storage

- €5.18/TB/month

3.4 Network Requirements


yaml

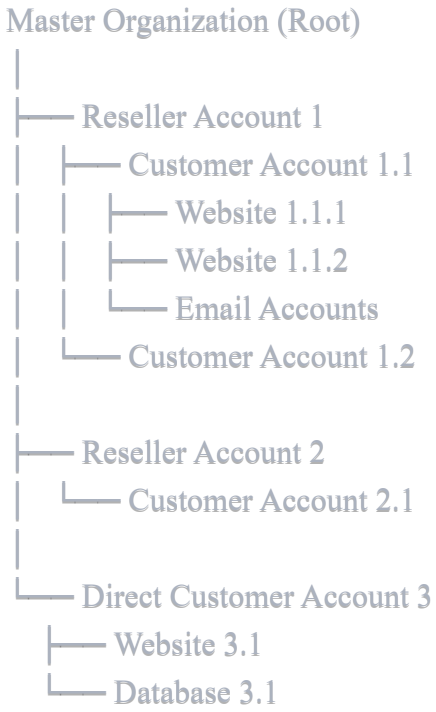
```yaml
Networking:
  IPv4: Public IP per server
  IPv6: /64 subnet per server
  Private Network: Hetzner vSwitch (10 Gbit/s)
  Load Balancer: Hetzner Load Balancer (optional)
  Floating IPs: For HA configurations

Firewall:
  Type: Hetzner Cloud Firewall
  Rules: Managed via Terraform/Ansible
  Default: Deny all, whitelist specific ports

SSL/TLS:
  Certificates: Let's Encrypt (automated)
  Backup: ZeroSSL integration
  Custom: Support for uploaded certificates
```

---

4. Core Platform Features

4.1 Multi-Tenancy Architecture

Organizational Hierarchy:



Features:

- **Unlimited Depth:** Support for multi-level reseller hierarchy
- **Isolated Resources:** Each organization has separate resource pools
- **Branded Portals:** White-label client portals per reseller
- **Permission Inheritance:** Granular permission system
- **Impersonation:** Admin can impersonate any lower-level user
- **Resource Quotas:** Per-organization limits (disk, bandwidth, accounts)

4.2 User Management System

User Types:

yaml

Super Admin:
Access Level: Full system access
Permissions:
- Manage all organizations
- System configuration
- Billing management
- Security settings
- API access management
- Server management

Reseller:
Access Level: Organization-specific
Permissions:
- Create customers
- Manage packages
- View invoices
- Custom branding
- White-label portal
- API access (limited)

Customer (End User):
Access Level: Account-specific
Permissions:
- Manage websites
- Email accounts
- Databases
- DNS records
- File manager
- Support tickets
- Billing (view/pay)

Technical User:
Access Level: Service-specific
Permissions:
- SSH/SFTP access
- Cron jobs
- Application deployment
- Database access
- Log viewing

Authentication:

- **Primary:** Password + 2FA (TOTP)
- **SSO:** SAML 2.0 / OAuth 2.0 support
- **API Keys:** For programmatic access
- **WebAuthn:** FIDO2 hardware key support
- **Session Management:** JWT-based with Redis storage

4.3 Dashboard & UI

Admin Dashboard:


yaml

```
Widgets:
  - System Status Overview
  - Resource Usage (CPU, RAM, Disk)
  - Revenue Metrics (MRR, ARR, Churn)
  - Active Services Count
  - Recent Activities Log
  - Security Alerts
  - Pending Tasks
  - Support Ticket Queue
  - Server Health Map
  - Quick Actions Panel

Features:
  - Customizable widget layout
  - Real-time updates via WebSockets
  - Dark mode / Light mode
  - Responsive design (mobile-first)
  - Keyboard shortcuts
  - Search everything (Cmd+K)
```

Client Portal:


yaml

Sections:
- Dashboard (Service overview)
- Services (Hosting, Domains, VPS)
- Billing (Invoices, Payments, Credits)
- Support (Tickets, Knowledge Base)
- Account Settings
- Security (2FA, API keys)

Features:
- Single-page application feel (HTMX)
- Instant search
- Contextual help
- Multi-language support
- Accessible (WCAG 2.1 AA)

4.4 Notification System

Channels:


yaml

Email:
  Engine: Go email-service + n8n
  Features:
    - HTML templates
    - Localization
    - Attachment support
    - Bounce handling
    - Queue management

SMS:
  Providers:
    - Twilio
    - Vonage (Nexmo)
    - MessageBird
  Use Cases:
    - 2FA codes
    - Critical alerts
    - Payment reminders

Push Notifications:
  Web Push API
  Use Cases:
    - Service status changes
    - Support ticket updates
    - System alerts

Webhooks:
  n8n integration
  Supports: Slack, Discord, Microsoft Teams

In-App:
  Real-time notifications bell
  Persistent notification center
  Mark as read/unread

---

5. Billing & Automation System

5.1 Core Billing Features

Invoice Management:


yaml

```
Automated Invoice Generation:
  Frequency: Daily, Weekly, Monthly, Quarterly, Annual
  Generation Time: Configurable (e.g., 1st of month)
  Due Date: Configurable grace period

Invoice Features:
  - Line item breakdown
  - Tax calculations (VAT, GST, Sales Tax)
  - Proration support
  - Credits application
  - Multiple currencies (40+)
  - PDF generation (LaTeX or wkhtmltopdf)
  - Custom invoice numbers
  - Company branding

Recurring Billing:
  - Automatic charge attempts
  - Failed payment handling
  - Retry logic (3 attempts over 7 days)
  - Dunning management
  - Auto-suspension after X days
  - Cancellation grace period
```

Payment Processing:


yaml

Payment Gateways:
  Primary:
    - Stripe (Cards, SEPA, iDEAL, Bancontact)
    - PayPal (Express Checkout, Subscriptions)

  Secondary:
    - Mollie (European payments)
    - GoCardless (Direct Debit)
    - Authorize.net
    - Braintree

  Cryptocurrency:
    - BTCPay Server integration
    - Supported: BTC, ETH, LTC, XMR

Payment Methods:
  - Credit/Debit Cards
  - Bank Transfer (Manual reconciliation)
  - SEPA Direct Debit
  - PayPal
  - Cryptocurrency
  - Credits/Prepaid Balance
  - Purchase Orders (B2B)

Automation:
  - Automatic payment capture
  - Invoice marking as paid
  - Service activation
  - Receipt generation
  - Accounting export (QuickBooks, Xero)

Product & Service Management:


yaml

Product Types:

1. Hosting Services:
   - Shared Hosting
   - Reseller Hosting
   - VPS Hosting
   - Dedicated Servers
   - Cloud Hosting

2. Domain Names:
   - Registration
   - Transfer
   - Renewal
   - Premium domains

3. SSL Certificates:
   - Let's Encrypt (Free)
   - Paid SSL (Comodo, DigiCert)
   - Wildcard SSL
   - EV SSL

4. Email Services:
   - Business Email
   - Email Archiving
   - Email Security (SpamExperts)

5. Software Licenses:
   - cPanel/Plesk licenses
   - Softaculous
   - Custom software keys

6. Additional Services:
   - Managed backups
   - CDN services
   - Managed security
   - Priority support
   - Server management

Pricing Models:

- Fixed price (one-time, recurring)
- Tiered pricing (based on resources)
- Usage-based (bandwidth, storage)

- Pay-as-you-go
- Custom pricing per customer

Setup Fees:
- One-time setup charges
- Migration fees
- Custom configuration fees

5.2 Package System

Hosting Package Configuration:


yaml

Package Structure:
  Name: Package identifier
  Type: Shared | Reseller | VPS | Dedicated

Resources:
  Disk Space: GB (e.g., 10 GB, Unlimited)
  Bandwidth: GB/month (e.g., 100 GB, Unlimited)
  Domains: Count (1, 5, 10, Unlimited)
  Subdomains: Count (Unlimited default)
  Email Accounts: Count (10, 50, Unlimited)
  Databases: Count (1, 5, 10, Unlimited)
  FTP Accounts: Count (1, 5, Unlimited)

Features:
  - PHP Version Selection: []string
  - SSL Certificates: bool
  - SSH Access: bool
  - Cron Jobs: int
  - Backup Frequency: string
  - Staging Environment: bool
  - Git Integration: bool
  - WP-CLI Access: bool
  - Node.js Support: bool
  - Python Support: bool

Security:
  - ModSecurity: bool
  - Firewall Rules: bool
  - Malware Scanning: bool
  - DDoS Protection: bool
  - Isolated PHP: bool

Performance:
  - CPU Cores: int
  - RAM: GB
  - I/O Priority: Low | Medium | High
  - Process Limit: int
  - Entry Processes: int (concurrent PHP processes)
  - NPROC: int (process limit)
  - PMEM: MB (PHP memory limit)

- IOPS: int (disk I/O operations)
- IOBW: MB/s (I/O bandwidth)

Package Management:


yaml

Operations:
- Create new package
- Clone existing package
- Edit package resources
- Enable/disable features
- Set pricing (one-time, recurring)
- Configure upgrade paths
- Set trial periods
- Configure auto-provisioning

Visibility:
- Public (shown on website)
- Hidden (manual assignment only)
- Reseller-specific
- Private (invitation-only)

Upgrade/Downgrade:
- Automatic resource adjustment
- Proration calculation
- Migration scheduling
- Data preservation
- Rollback capability

5.3 Domain Management

Domain Registrar Integration:


yaml

Supported Registrars:
- WHMCS Module Compatible:
  - Namecheap
  - ResellerClub
  - Enom
  - OpenSRS
  - CentralNic
  - GoDaddy Reseller
  - Internet.bs
  - Hexonet

- API Integration (Custom):
  - Cloudflare Registrar
  - Google Domains
  - Name.com

Features:
Registration:
  - Real-time availability check
  - Bulk domain search
  - Premium domain support
  - IDN (Internationalized Domain Names)
  - Multiple TLDs (500+)
  - WHOIS privacy protection
  - Auto-renewal configuration

Transfer:
  - EPP code management
  - Transfer status tracking
  - Auto-renewal after transfer
  - WHOIS update

Management:
  - DNS management (built-in)
  - Nameserver updates
  - Contact information updates
  - Auth/EPP code retrieval
  - Domain locking/unlocking
  - WHOIS privacy toggle
  - DNSSEC management

Renewal:
- Automatic renewal reminders
- Grace period handling
- Redemption period support
- Bulk renewal discounts

Pricing:
- Registrar cost + markup (fixed or %)
- Multi-year discount tiers
- Transfer pricing
- Renewal pricing (can differ from registration)
- Premium domain surcharges

5.4 Support Ticket System

Ticket Management:

yaml

Ticket Lifecycle:
States: Open → Assigned → In Progress → Waiting Reply → Resolved → Closed

Ticket Properties:
- Unique ID
- Subject
- Department (Billing, Technical, Sales)
- Priority (Low, Normal, High, Critical)
- Status
- Assigned To (Staff member)
- Customer
- Service (Related to)
- Created Date
- Last Updated
- SLA Timer

Features:
- Rich text editor
- File attachments (max 10 MB per file)
- Internal notes (staff-only)
- Email notifications
- Ticket merging
- Ticket splitting
- Canned responses (templates)
- Auto-assignment rules
- Tag system
- Search & filtering
- Export to PDF

SLA Management:
Priority Levels:
Critical:
First Response: 15 minutes
Resolution Time: 1 hour
High:
First Response: 1 hour
Resolution Time: 4 hours
Normal:
First Response: 4 hours
Resolution Time: 24 hours
Low:

First Response: 12 hours
Resolution Time: 48 hours

SLA Tracking:
- Timer paused when waiting for customer reply
- Escalation when SLA breach approaching
- Automatic notifications
- SLA performance reports

Knowledge Base:

yaml

```
Structure:
  Categories:
    - Getting Started
    - Billing & Account
    - Hosting Management
    - Email Setup
    - Domain Management
    - Security & Backup
    - Troubleshooting
    - API Documentation

  Articles:
    - Title
    - Content (Markdown support)
    - Category
    - Tags
    - Visibility (Public, Customer-only, Reseller-only)
    - Author
    - Created/Updated dates
    - View count
    - Helpful votes

  Features:
    - Full-text search (Meilisearch)
    - Related articles suggestion
    - Breadcrumb navigation
    - Table of contents (auto-generated)
    - Code syntax highlighting
    - Embedded videos
    - Helpful/Not Helpful voting
    - Comment system (optional)
    - Multi-language support
```

Live Chat (Optional):

yaml

Integration:
- Chatwoot (Open source)
- Tawk.to
- Crisp
- Intercom
- Custom WebSocket implementation

Features:
- Real-time messaging
- Typing indicators
- File sharing
- Chat history
- Canned responses
- Routing to departments
- Offline message capture
- Mobile app support

5.5 Client Relationship Management

Customer Profiles:


yaml

Information Captured:

Basic:

- Full Name
- Company Name (optional)
- Email (primary + additional)
- Phone numbers (mobile, work, home)
- Address (billing & service)
- Country, State, Postal Code
- Preferred Language
- Time Zone

Business:

- Company Registration Number
- VAT/Tax ID
- Industry
- Company Size
- Website

Account:

- Customer ID
- Account Status (Active, Suspended, Cancelled)
- Registration Date
- Last Login
- Total Spent
- Lifetime Value (LTV)
- Credit Balance
- Payment Method on File

Marketing:

- Marketing Consent (GDPR compliant)
- Communication Preferences (Email, SMS, Phone)
- Tags (VIP, High-Risk, Beta Tester, etc.)
- Custom Fields (unlimited)

Activity Tracking:

- Login history
- Service provisioning
- Invoice history
- Payment history
- Support ticket history

- Configuration changes
- API usage logs

Reseller Management:


yaml

Reseller-Specific Features:

Branding:

- Custom logo
- Color scheme
- Company name
- Support email/phone
- Terms of Service URL
- Privacy Policy URL

Commission Structure:

- Percentage-based
- Fixed amount per service
- Tiered commissions (volume-based)
- Monthly payout
- Payout history

Resource Allocation:

- Disk space pool
- Bandwidth pool
- Maximum accounts
- Server assignment

Pricing Control:

- Override retail pricing
- Set custom pricing per package
- Coupon creation
- Discount management

Customer Management:

- View all customers
- Create customers
- Suspend/unsuspend
- Terminate services
- Reset passwords

Reporting:

- Revenue reports
- Service usage

- Customer growth
- Support ticket metrics

---

6. Hosting Control Panel

6.1 Website Management

Site Types & Application Support:

yaml

Application Types:

1. WordPress:
   - One-click installer
   - WP-CLI integration
   - Automatic updates (core, plugins, themes)
   - Staging environment
   - Malware scanner (WordFence API)
   - Performance optimization (Object cache, OPcache)
   - Multisite support

2. Generic PHP:
   - PHP 7.4, 8.0, 8.1, 8.2, 8.3, 8.4
   - Composer installed
   - Multiple PHP-FPM pools
   - Per-directory PHP version
   - Framework templates:
     - Laravel 10/11
     - Symfony 6/7
     - CodeIgniter 4
     - CakePHP 5
     - Yii 2
     - Slim 4

3. Node.js:
   - NVM for version management
   - Node.js 16, 18, 20, 22
   - npm, yarn, pnpm support
   - PM2 process manager
   - Environment variable management
   - Framework support:
     - Express.js
     - NestJS
     - Next.js
     - Nuxt.js
     - AdonisJS

4. Python:
   - Python 3.9, 3.10, 3.11, 3.12
   - Virtual environments (venv)
   - pip package management
   - WSGI/ASGI support

- Framework support:
  - Django 4/5
  - Flask 3
  - FastAPI
  - Tornado

5. Static Sites:
   - HTML/CSS/JavaScript
   - JAMstack deployments
   - Static site generator support:
     - Hugo
     - Jekyll
     - Gatsby
     - Eleventy (11ty)

6. Reverse Proxy:
   - Proxy to internal services
   - WebSocket support
   - SSL termination
   - Custom headers
   - Load balancing

Web Server Configuration:

yaml

Primary Web Server: NGINX 1.26+
Features:
- HTTP/2 and HTTP/3 (QUIC) support
- SSL/TLS 1.3
- Brotli compression
- GZip compression
- Request buffering
- Rate limiting
- GeoIP filtering
- Custom error pages
- Security headers (HSTS, CSP, X-Frame-Options)

PHP-FPM Configuration:
Per-Site Pools:
- Isolated Unix user
- Dedicated socket
- Resource limits (memory, processes)
- OPcache settings
- Environment variables
- PHP.ini overrides

Performance Tuning:
- pm.max_children (auto-calculated based on RAM)
- pm.start_servers
- pm.min_spare_servers
- pm.max_spare_servers
- pm.max_requests (prevent memory leaks)
- request_terminate_timeout

Caching Layer (Optional):
Varnish Cache 7.x:
- Full-page caching
- ESI support
- Grace mode (serve stale on backend failure)
- VCL customization
- Cache invalidation (PURGE, BAN)
- Cache hit rate monitoring

Redis Object Cache:
- WordPress object caching
- Session storage

- PHP OPcache + Redis APCu
- LRU eviction policy

Site Creation Workflow:

yaml

Process:

1. User Input:
   - Domain name
   - Site type (WordPress, PHP, Node.js, etc.)
   - PHP version (if applicable)
   - SSL certificate (Let's Encrypt or custom)
   - Advanced options

2. Automated Steps:
   - Create Unix user (isolated)
   - Setup directory structure:

```
/home/{username}/
  ├── htdocs/          (web root)
  ├── logs/            (access, error logs)
  ├── tmp/             (temporary files)
  ├── private/         (above web root)
  └── backups/         (local backup cache)
```

   - Configure NGINX vhost
   - Create PHP-FPM pool (if PHP)
   - Setup SSL certificate (Let's Encrypt auto-issue)
   - Create DNS records (if using platform DNS)
   - Configure firewall rules
   - Setup log rotation
   - Create initial backups

3. Post-Provisioning:
   - Send credentials (SFTP, SSH, database if created)
   - Setup monitoring
   - Schedule first backup
   - Enable security features

Execution Time: < 60 seconds

6.2 Domain Management (Per-Site)

yaml

Domain Configuration:

Primary Domain:

- Main domain for the site
- SSL certificate attached
- WWW redirection (configurable)

Aliases (Add-on Domains):

- Unlimited aliases per site
- Point to same document root
- Separate SSL certificates
- Independent DNS records

Subdomains:

- Unlimited subdomains
- Can point to different document roots
- Wildcard subdomain support
- SSL via Let's Encrypt (wildcard)

Redirects:

- 301 (Permanent) / 302 (Temporary)
- Domain to domain
- Path-based redirects
- Regex support
- HTTPS enforcement

DNS Management:

If using platform DNS:

- A Records
- AAAA Records (IPv6)
- CNAME Records
- MX Records (email routing)
- TXT Records (SPF, DKIM, DMARC, verification)
- SRV Records
- CAA Records (SSL issuance control)
- NS Records (subdomain delegation)
- TTL configuration per record
- DNSSEC support
- GeoDNS (route based on visitor location)

6.3 File Management

yaml

File Manager (Web-Based):

Features:

- Browse directory tree
- Upload files (drag & drop, chunked upload for large files)
- Download files/folders (as ZIP)
- Create/edit files (syntax highlighting)
- Create/delete/rename folders
- Move/copy files
- Change permissions (chmod)
- Change ownership (chown) - admin only
- Extract archives (ZIP, TAR, GZ, BZ2)
- Create archives
- Search files
- Preview images
- View file metadata
- Symlink creation
- Code editor with syntax highlighting (CodeMirror)

Security:

- Chrooted to user's home directory
- File upload virus scanning
- File type restrictions
- Maximum file size limits
- Path traversal protection

SFTP/SSH Access:

Protocol: SSH2 / SFTP

Port: 22 (configurable)

Authentication:

- Password (required on first setup)
- SSH key (recommended)
- 2FA via Google Authenticator (optional)

Chroot:

- Isolated to user's home directory
- No system directory access
- jailkit or similar implementation

SSH Features:

- Bash shell
- WP-CLI (WordPress sites)

- Composer (PHP sites)
- npm/yarn (Node.js sites)
- pip (Python sites)
- Git (for deployment)
- Custom binaries (approved list)

FTP/FTPS (Optional):
Protocol: FTP, FTPS, SFTP
Server: ProFTPD or Pure-FTPd
Features:
- Multiple FTP accounts per site
- Per-account directory restrictions
- Bandwidth throttling
- Connection limits
- Anonymous FTP (disabled by default)

6.4 Database Management

yaml

Supported Database Engines:

1. MySQL/MariaDB:
   - MySQL 8.0 (primary)
   - MariaDB 10.11 (alternative)
   - Per-site database creation
   - Database user management
   - Remote access (whitelist IP)
   - Import/Export (SQL dumps)
   - phpMyAdmin access

2. PostgreSQL 16:
   - Full SQL support
   - Extension support (PostGIS, pg_vector)
   - pgAdmin access (web-based)
   - Replication support

3. Redis:
   - Object caching
   - Session storage
   - Separate Redis instance per site (isolated)
   - Redis Commander (web UI)

4. MongoDB (Optional):
   - NoSQL support
   - Mongo Express (web UI)

Database Operations:

Create:
- Auto-generate database name (prefix_suffix)
- Auto-generate username & password (secure)
- Encoding: UTF8MB4 (MySQL) / UTF8 (PostgreSQL)
- Collation: utf8mb4_unicode_ci (MySQL)

Management:
- Change password
- Add/remove users
- Grant/revoke privileges
- Repair tables (MySQL)
- Optimize tables (MySQL)
- Check tables (MySQL)

```
  Backup:
    - Automatic daily backups
    - On-demand backups
    - Export to SQL file
    - Point-in-time recovery (PostgreSQL)

  Monitoring:
    - Connection count
    - Query statistics
    - Slow query log
    - Table sizes
    - Index usage

phpMyAdmin Configuration:
  Version: Latest
  Features:
    - Web-based SQL interface
    - Database browsing
    - Query execution
    - Import/Export
    - User management
    - Search & replace
    - Visual query builder
  Security:
    - IP whitelisting
    - 2FA enforced for admin
    - Separate login credentials
    - Session timeout
```

---

7. Security Framework

7.1 PHP Hardening System (CloudLinux-Inspired)

HardenedPHP Engine (RUST Implementation):

yaml

Core Features:
  PHP Version Selector:
    - Support: PHP 7.4 - 8.4
    - Per-site version selection
    - Per-directory version override (.user.ini)
    - Automatic security patching for EOL versions
    - Custom PHP builds with security patches

  Disabled Functions (Security):
    Default Disabled:
      - exec, passthru, shell_exec, system
      - proc_open, popen, curl_exec
      - curl_multi_exec, parse_ini_file
      - show_source, eval (optional)

    Configurable:
      - Allow list per site
      - Required for specific applications
      - Audit log for function usage

  Resource Limits:
    Per-Site Limits:
      memory_limit: 128M - 512M (configurable)
      max_execution_time: 30 - 300 seconds
      max_input_time: 60 seconds
      upload_max_filesize: 2M - 100M
      post_max_size: 8M - 100M
      max_input_vars: 1000 - 5000

    Process Limits:
      pm.max_children: Auto-calculated
      request_terminate_timeout: 300s
      rlimit_files: 1024 (open file limit)
      rlimit_core: 0 (no core dumps)

  Security Features:
    open_basedir: Chroot to site directory
    allow_url_fopen: Disabled (default)
    allow_url_include: Disabled (always)
    expose_php: Off (hide PHP version)
    display_errors: Off (production)

```
    log_errors: On
    error_reporting: E_ALL & ~E_DEPRECATED
    session.cookie_httponly: On
    session.cookie_secure: On (HTTPS sites)
    session.cookie_samesite: Strict

  PHP Extension Management:
    Core Extensions: Always available
    Optional Extensions:
      - Selectable per site via control panel
      - Extensions: gd, imagick, redis, memcached,
                    mongodb, intl, soap, xmlrpc, zip,
                    ioncube, sourceguardian
      - Version-specific availability

  Vulnerability Patching:
    Automated Patching:
      - CVE monitoring (NVD database)
      - Automatic patch application
      - Backport security fixes to EOL versions
      - Notification to admins
      - Rollback capability

    Patch Sources:
      - Official PHP security updates
      - Remi repository patches
      - Custom patches (reviewed & tested)
```

Account Isolation (CageFS-Like):

yaml

Implementation: Linux Namespaces + cgroups v2

Features:
  Filesystem Isolation:
    - Separate filesystem view per site
    - No visibility into other accounts
    - Read-only system directories
    - Symlink attack prevention
    - Hard link restrictions

  Process Isolation:
    - PID namespace isolation
    - Cannot see other users' processes
    - Process limit enforcement (cgroups)
    - CPU quota (cgroups cpu.max)
    - Memory limits (cgroups memory.max)

  Network Isolation:
    - Per-site network namespace (optional)
    - Outbound connection limits
    - Port binding restrictions
    - Rate limiting per site

  Temporary Filesystem:
    - Separate /tmp per site
    - tmpfs with size limit
    - Automatic cleanup on site deletion

  Environment Isolation:
    - Separate environment variables
    - No access to global environment
    - Secure variable storage (encrypted)

Resource Limits (cgroups v2):
  CPU:
    cpu.weight: 100 - 10000 (priority)
    cpu.max: "50000 100000" (50% of one core)

  Memory:
    memory.max: 256MB - 4GB
    memory.high: 80% of max (throttle before OOM)

memory.swap.max: 0 (no swap usage)

I/O:
io.weight: 100 - 10000 (priority)
io.max: IOPS and bandwidth limits

Process:
pids.max: 100 - 500 processes

Network:
bandwidth: TX/RX limits (tc qdisc)
connection limit: per-site iptables rules

7.2 Web Application Firewall (WAF)

ModSecurity 3.x + OWASP CRS 4.x:

yaml

WAF Engine: ModSecurity 3 (libmodsecurity)
Integration: NGINX connector
Rule Set: OWASP Core Rule Set (CRS) 4.6+

Protection Categories:

1. OWASP Top 10:
   - A01: Broken Access Control
   - A02: Cryptographic Failures
   - A03: Injection (SQL, Command, LDAP)
   - A04: Insecure Design
   - A05: Security Misconfiguration
   - A06: Vulnerable and Outdated Components
   - A07: Identification and Authentication Failures
   - A08: Software and Data Integrity Failures
   - A09: Security Logging and Monitoring Failures
   - A10: Server-Side Request Forgery (SSRF)

2. Attack Types:
   - SQL Injection (SQLi)
   - Cross-Site Scripting (XSS)
   - Cross-Site Request Forgery (CSRF)
   - Remote File Inclusion (RFI)
   - Local File Inclusion (LFI)
   - Remote Code Execution (RCE)
   - XML External Entity (XXE)
   - Server-Side Template Injection (SSTI)
   - Path Traversal
   - Session Fixation

3. Protocol Validation:
   - HTTP Protocol violations
   - Malformed requests
   - Invalid HTTP methods
   - Oversized headers/bodies
   - Null byte injection

4. Scanner Detection:
   - Nikto, Nmap, SQLMap
   - Burp Suite, OWASP ZAP
   - Dirbuster, Wfuzz
   - Security scanner signatures

Rule Configuration:
  Paranoia Level: 1-4 (default: 2)
    Level 1: Basic protection, low false positives
    Level 2: Moderate protection (recommended)
    Level 3: High protection, may have false positives
    Level 4: Extreme protection, tuning required

  Anomaly Scoring:
    Inbound Threshold: 5 (default)
    Outbound Threshold: 4 (default)
    Scoring: Each rule adds points, threshold triggers block

  Actions:
    - Detect Only (log but don't block)
    - Block (403 Forbidden)
    - Redirect (to custom page)
    - CAPTCHA Challenge
    - Rate Limit (temporary block)

Custom Rule Engine:
  Rule Creation:
    - Regex-based patterns
    - IP address rules
    - User-agent rules
    - Geolocation rules
    - Custom variable matching

  Rule Management:
    - Per-site custom rules
    - Global rules
    - Rule priorities
    - Rule enable/disable
    - False positive whitelisting

  Rule Templates:
    - WordPress hardening
    - Joomla protection
    - Drupal security
    - E-commerce (WooCommerce, PrestaShop)
    - API protection

Machine Learning Enhancement:
  Implementation: RUST-based ML engine
  Features:
    - Behavioral analysis
    - Anomaly detection
    - Zero-day exploit detection
    - Adaptive rule tuning
    - False positive reduction

  Training Data:
    - Attack patterns from live traffic
    - Threat intelligence feeds
    - Community shared data
    - Vendor provided datasets

7.3 Enterprise Firewall

Multi-Layer Firewall System:


yaml

Layer 1: Cloud Firewall (Hetzner)
  Type: Stateful packet filter
  Location: Before server (cloud level)
  Rules:
    - Default deny all inbound
    - Allow specific ports:
      - 22 (SSH) - restricted IPs
      - 80 (HTTP)
      - 443 (HTTPS)
      - 25, 465, 587 (SMTP)
      - 110, 995 (POP3)
      - 143, 993 (IMAP)
      - 53 (DNS)
      - Custom application ports
    - Outbound: Allow all (can be restricted)

  Features:
    - Managed via Hetzner API
    - Terraform integration
    - Rate limiting (connection/s)
    - GeoIP filtering
    - DDoS mitigation (automatic)

Layer 2: Host Firewall (nftables)
  Implementation: RUST service managing nftables

  Default Rules:
    Input Chain:
      - Allow established, related connections
      - Allow localhost
      - Allow ICMP (ping) - rate limited
      - Drop invalid packets
      - Log and drop suspicious packets

    Forward Chain:
      - Drop all (server is not a router)

    Output Chain:
      - Allow all established connections
      - Allow DNS (53)
      - Allow HTTP/HTTPS (80, 443)

- Log unknown outbound connections

Advanced Features:
  Connection Tracking:
    - Max connections per IP: 100
    - SYN flood protection
    - Connection state tracking
    - Timeout: 300 seconds

  Rate Limiting:
    - HTTP: 100 requests/second per IP
    - SSH: 3 attempts/minute per IP
    - SMTP: 20 emails/minute per user
    - General: 1000 packets/second per IP

  Port Knocking:
    - Secret port sequence to open SSH
    - Time-limited access
    - Configurable sequences

  Fail2Ban Integration:
    - Automatic IP banning
    - Jail configurations:
      - SSH (sshd)
      - Web server (nginx-http-auth, nginx-noscript)
      - Email (postfix, dovecot)
      - FTP (pure-ftpd, proftpd)
      - ModSecurity (waf)

    Ban Policy:
      - SSH: 3 failures, ban 1 hour
      - HTTP Auth: 5 failures, ban 10 minutes
      - ModSecurity: 10 violations, ban 1 hour
      - Recidive: Permanent ban after 3 incidents

  IP Reputation:
    - Threat intelligence feeds
    - Known bad IP lists (AbuseIPDB, Spamhaus)
    - Automatic blocking
    - Whitelist for false positives

Layer 3: Application Firewall (Per-Site)
  Type: NGINX-level restrictions

  Features:
    - IP allow/deny lists
    - User-agent filtering
    - Referrer checking
    - Geographic restrictions (GeoIP2)
    - Request method filtering (GET, POST, etc.)
    - Custom header requirements
    - API rate limiting (per endpoint)
    - Bot protection (CAPTCHA challenges)

GeoIP Filtering:
  Database: MaxMind GeoIP2
  Actions:
    - Allow list (specific countries)
    - Deny list (block countries)
    - Redirect (to localized site)
    - Custom response codes

  Use Cases:
    - Comply with data residency laws
    - Block high-risk countries
    - Reduce attack surface
    - Content localization

DDoS Protection:
  Layer 3/4 (Network/Transport):
    - Hetzner's DDoS protection (automatic)
    - SYN flood mitigation (SYN cookies)
    - UDP flood mitigation
    - ICMP flood mitigation
    - Amplification attack protection

  Layer 7 (Application):
    - HTTP flood protection
    - Slowloris attack mitigation
    - RUDY (R-U-Dead-Yet) protection
    - Challenge-response (JavaScript, CAPTCHA)
    - Rate limiting per endpoint

- Connection limits per IP

Mitigation Strategies:
- Traffic profiling (baseline vs. attack)
- Automatic scaling (spin up instances)
- Blackhole routing (severe attacks)
- Anycast routing (distribute load)

7.4 Brute Force Protection

Multi-Stage Protection:


yaml

Stage 1: Rate Limiting
SSH:
- 3 login attempts per minute
- 10 attempts per hour
- Source IP tracking

Web Applications:
- Login forms: 5 attempts per 15 minutes
- Password reset: 3 requests per hour
- Registration: 2 attempts per hour

API:
- Authentication endpoints: 10 requests per minute
- Token refresh: 20 requests per hour

Stage 2: Progressive Delays
Delay Calculation: delay = 2^(attempts - 1) seconds

Example:
Attempt 1: 0 seconds delay
Attempt 2: 2 seconds delay
Attempt 3: 4 seconds delay
Attempt 4: 8 seconds delay
Attempt 5: 16 seconds delay
Attempt 6+: 32 seconds delay (max)

Stage 3: CAPTCHA Challenges
Trigger: After 3 failed attempts
Type: hCaptcha or reCAPTCHA v3
Bypass: Trusted IPs (whitelist)

CAPTCHA Levels:
- Easy: After 3 failures
- Medium: After 5 failures
- Hard: After 10 failures

Stage 4: Account Lockout
Temporary Lockout:
- Duration: 15 minutes after 5 failures
- Extended: 1 hour after 10 failures
- Notification: Email to account owner

Permanent Lockout:
- After 20 consecutive failures
- Admin intervention required
- IP blacklist + account flag

Stage 5: IP Blocking (Fail2Ban)
Ban Triggers:
- SSH: 3 failures in 10 minutes → ban 1 hour
- HTTP Auth: 5 failures in 10 minutes → ban 10 minutes
- Web Login: 10 failures in 30 minutes → ban 1 hour
- Repeat Offender: 3 bans in 24 hours → ban 24 hours

Permanent Ban:
- 5 ban events in 7 days
- Added to global blacklist
- Manual removal required

Credential Stuffing Detection:
Indicators:
- Multiple usernames from single IP
- Multiple IPs trying same username
- Unusual login patterns (time, geo)
- User-agent anomalies

Response:
- CAPTCHA challenge
- Temporary IP block
- Email notification
- Security alert (dashboard)

Distributed Attack Mitigation:
Detection:
- 100+ IPs attempting same account
- 1000+ login attempts across accounts
- Geographic distribution pattern

Response:
- Enable global rate limiting
- Require CAPTCHA for all logins
- Notify security team

- Consider DDoS attack

Two-Factor Authentication (2FA):
Methods:
- TOTP (Google Authenticator, Authy)
- SMS (Twilio, Vonage)
- Email (backup method)
- Backup codes (one-time use)
- Hardware keys (WebAuthn/FIDO2)

Enforcement:
- Optional (default)
- Required for admins
- Required for API access
- Grace period: 14 days after enabling

Recovery:
- Backup codes (10 codes, single use)
- Email recovery link
- Admin reset (with verification)

7.5 Malware & Antivirus Protection

Multi-Engine Scanner:


yaml

Scanning Engines:

1. ClamAV:
   - Open source antivirus
   - Virus database updates: Hourly
   - Detection: Signature-based
   - File types: All

2. Linux Malware Detect (LMD):
   - Focuses on malware targeting Linux
   - Detects: Backdoors, rootkits, exploits
   - Updates: Daily

3. AI/ML-Based Scanner (Custom - RUST):
   - Heuristic analysis
   - Behavioral detection
   - Zero-day malware detection
   - Training: Continuous (threat feeds)

Scan Types:

Real-Time Scanning:
- File upload scanning (web forms, FTP, SFTP)
- Email attachment scanning (all inbound)
- Quarantine on detection
- Notification: Admin + customer

Scheduled Scans:
- Daily: Quick scan (changed files only)
- Weekly: Full scan (all files)
- Monthly: Deep scan (including archives)
- Custom: Per-site schedules

On-Demand Scans:
- Manual trigger via control panel
- API-triggered scans
- Post-migration scans
- Pre-restore scans

Detection & Response:

Malware Signatures Detected:
- Webshells: c99, r57, WSO, b374k, etc.
- Backdoors: FilesMan, WSO Shell, Adminer

- Exploits: TimThumb, WordPress plugin vulns
- Phishing kits
- Cryptocurrency miners
- SEO spam injections

Actions:
Automatic:
- Quarantine file (move to /quarantine/)
- Disable site (optional, configurable)
- Block uploads from infected account
- Notify account owner
- Notify admin
- Create incident ticket

Manual Options:
- Delete infected file
- Attempt automatic cleanup
- Restore from backup (pre-infection)
- Whitelist (false positive)
- Request manual review

Cleanup Process:
Automated Cleanup:
- Remove malicious code (pattern-based)
- Restore modified core files (WordPress)
- Reset file permissions
- Reset account passwords
- Revoke compromised API keys

Manual Cleanup:
- Security expert review
- Deep forensic analysis
- Vulnerability assessment
- Remediation report
- Prevention recommendations

Behavioral Monitoring:
Process Monitoring:
- Unusual CPU usage (crypto miners)
- Excessive network traffic
- Unusual outbound connections

- Privilege escalation attempts
- Rootkit detection

File Integrity Monitoring:
- Core file modification detection
- Permission changes
- Ownership changes
- Suspicious file creation (in tmp, wp-content/uploads)
- Hidden files (dotfiles in web root)

Web Traffic Analysis:
- High 404 error rates (scanning)
- POST to unusual locations
- Large POST requests (shell uploads)
- Requests to suspicious files (.php in uploads)
- User-agent anomalies

Integration with WAF:
- Share threat intelligence
- Block IPs uploading malware
- Add WAF rules for detected attack patterns
- Correlate web attacks with malware infections

Reporting:
Scan Reports:
- Files scanned: count
- Threats detected: count
- False positives: count
- Quarantined files: list
- Cleaned files: list
- Scan duration
- Next scheduled scan

Incident Reports:
- Infection timeline
- Entry point analysis
- Affected files
- Data breach assessment
- Remediation steps taken
- Prevention recommendations

7.6 SSL/TLS Management

yaml

Certificate Types:
1. Let's Encrypt (Free):
- Domain Validated (DV)
- Automatic issuance
- 90-day validity
- Auto-renewal (60 days before expiry)
- Wildcard support (DNS-01 challenge)

2. ZeroSSL (Free - Backup):
- Domain Validated (DV)
- 90-day validity
- API-based issuance
- Fallback when Let's Encrypt fails

3. Custom/Paid SSL:
- Upload custom certificates
- Organization Validated (OV)
- Extended Validation (EV)
- Code signing certificates

ACME Protocol Support:
Challenges:
- HTTP-01: File-based verification (most common)
- DNS-01: TXT record verification (wildcard certs)
- TLS-ALPN-01: TLS handshake verification

Client: acme.sh (Bash) or lego (Go)

Automation:
- Automatic domain verification
- Certificate installation
- NGINX reload (zero-downtime)
- Renewal 30 days before expiry
- Fallback to ZeroSSL on failure
- Email notification on issues

TLS Configuration:
Protocols:
- TLS 1.2 (minimum)
- TLS 1.3 (preferred)
- TLS 1.0/1.1 (disabled - deprecated)

- SSLv2/SSLv3 (disabled - insecure)

Cipher Suites (Modern):
- TLS_AES_256_GCM_SHA384 (TLS 1.3)
- TLS_CHACHA20_POLY1305_SHA256 (TLS 1.3)
- TLS_AES_128_GCM_SHA256 (TLS 1.3)
- ECDHE-RSA-AES128-GCM-SHA256 (TLS 1.2)
- ECDHE-RSA-AES256-GCM-SHA384 (TLS 1.2)

Features:
- OCSP Stapling (enabled)
- Perfect Forward Secrecy (PFS)
- HSTS (HTTP Strict Transport Security)
  - max-age: 31536000 (1 year)
  - includeSubDomains
  - preload
- Session resumption (TLS tickets)
- 0-RTT (TLS 1.3) - optional

SSL Rating:
Target: A+ on SSL Labs
Tests:
- Certificate validity
- Protocol support
- Cipher strength
- Key exchange
- HSTS implementation
- Certificate transparency

7.7 Security Monitoring & Logging

yaml

Log Collection:
  System Logs:
    - /var/log/auth.log (SSH, login attempts)
    - /var/log/syslog (system messages)
    - /var/log/kern.log (kernel messages)

  Application Logs:
    - NGINX access/error logs
    - PHP-FPM logs
    - Database logs (MySQL, PostgreSQL)
    - Email logs (Postfix, Dovecot)

  Security Logs:
    - ModSecurity audit logs
    - Fail2Ban logs
    - Firewall logs (nftables)
    - Malware scan logs
    - Intrusion detection logs

Log Aggregation:
  Tool: Vector + Loki + Grafana

  Pipeline:
    1. Vector (log collector):
      - Collects from all services
      - Parses structured logs
      - Filters sensitive data
      - Enriches with metadata

    2. Loki (log storage):
      - Time-series log database
      - Label-based indexing
      - Efficient compression
      - Retention: 90 days (configurable)

    3. Grafana (visualization):
      - Log exploration
      - Custom dashboards
      - Alerting rules
      - Correlation with metrics

SIEM Features:
  Correlation Engine:
    - Correlate logs across services
    - Detect attack patterns
    - Track lateral movement
    - Identify compromised accounts

  Use Cases:
    - Brute force detection
    - Unusual login patterns
    - Data exfiltration attempts
    - Privilege escalation
    - Malware propagation

Alerting:
  Alert Channels:
    - Email (admin, security team)
    - SMS (critical alerts)
    - Slack/Discord webhooks
    - PagerDuty integration
    - In-app notifications

  Alert Types:
    Critical (Immediate):
      - Active malware infection
      - Root access compromise
      - DDoS attack in progress
      - Mass data deletion

    High (15 minutes):
      - Brute force attack (ongoing)
      - WAF blocking spike
      - Unusual outbound traffic
      - Failed backup

    Medium (1 hour):
      - High 404 rate (scanning)
      - SSL certificate expiring soon
      - Disk space warning
      - High CPU/memory usage

Low (Daily digest):
- Failed login attempts (low count)
- Software updates available
- Scheduled maintenance reminder

Security Dashboard:

Metrics:
- Security score (0-100)
- Blocked attacks (24h, 7d, 30d)
- Top attacked sites
- Top attack sources (IPs, countries)
- WAF rule hits
- Malware detections
- Failed logins
- SSL certificate status
- Firewall rule hits
- Vulnerability count (unpatched)

Visualizations:
- Attack map (geographic)
- Timeline (attacks over time)
- Threat heatmap (site x attack type)
- Top attackers list
- Blocked IP list

Compliance & Auditing:

Audit Logs:
- All admin actions
- Configuration changes
- User creation/deletion
- Service provisioning
- Billing events
- Data access (for PII)

Compliance Reports:
- PCI DSS compliance status
- GDPR data access logs
- HIPAA audit logs
- SOC 2 evidence
- ISO 27001 checklist

```
Data Retention:
  - Security logs: 1 year
  - Audit logs: 7 years
  - Access logs: 90 days
  - Compliance logs: Per requirement
```

---

8. Multi-Server Management

8.1 Server Roles & Distribution

Role-Based Architecture (Inspired by Enhance Panel):

yaml

Server Roles:

1. Control Node (Master):

   Purpose: Central management, control panel UI, billing

   Services:

   - Web interface (HTMX frontend)
   - API gateway (RUST)
   - PostgreSQL (primary database)
   - Redis (session, cache)
   - n8n (workflow automation)
   - NATS (message broker)

   Requirements:

   - Hetzner CPX31 minimum
   - Private network access
   - Floating IP (high availability)

2. Application Server:

   Purpose: Host customer websites and applications

   Services:

   - NGINX web server
   - PHP-FPM (7.4-8.4)
   - Node.js runtime
   - Python runtime
   - Git
   - Varnish Cache (optional)

   Count: 1-N (scale as needed)

   Load Balancing: Hetzner Load Balancer or internal NGINX

3. Database Server:

   Purpose: Host customer databases

   Services:

   - MySQL/MariaDB
   - PostgreSQL
   - Redis instances (per customer)
   - phpMyAdmin
   - pgAdmin

   Count: 1-N (replication supported)

   High Availability: Master-slave replication

4. Email Server:
   Purpose: Email hosting for customer domains
   Services:
     - Postfix (SMTP)
     - Dovecot (IMAP/POP3)
     - SpamAssassin (spam filtering)
     - ClamAV (virus scanning)
     - Roundcube (webmail)
     - DKIM/DMARC/SPF

   Count: 1-N (redundancy recommended)
   Clustering: MX records point to multiple servers

5. DNS Server:
   Purpose: Authoritative DNS for customer domains
   Services:
     - PowerDNS
     - PostgreSQL backend
     - DNSSEC support
     - GeoDNS (optional)

   Count: 2+ (minimum for redundancy)
   Location: Different geographic regions

6. Backup Server:
   Purpose: Incremental backup storage
   Services:
     - Restic backup server
     - Hetzner Object Storage sync
     - Backup scheduler
     - Retention manager

   Count: 1+ (multi-region recommended)
   Storage: Hetzner Storage Box or Object Storage

7. Monitoring Server:
   Purpose: Central monitoring and alerting
   Services:
     - Prometheus
     - Grafana
     - Alertmanager

- Loki (logs)
- Vector (log aggregation)

Count: 1 (high availability setup recommended)

Server Provisioning:

Automated via Ansible:

1. Launch Hetzner Cloud instance (API call)
2. Assign to private network
3. Configure firewall rules
4. Install base OS packages
5. Install role-specific software
6. Configure services
7. Integrate with control node
8. Add to monitoring
9. Enable backups
10. Health check & validation

Time to Deploy: 5-10 minutes per server

Server Management:

Operations:

- Add new server (auto-provision)
- Remove server (graceful drain)
- Update software (rolling updates)
- Reboot server (scheduled maintenance)
- Scale resources (resize instance)
- Migrate sites (between servers)
- Load balancing configuration
- Failover testing

Monitoring:

- CPU, RAM, Disk, Network usage
- Service health checks
- SSL certificate expiry
- Backup status
- Security alerts
- Performance metrics (response time, throughput)

8.2 Load Balancing

yaml

Implementation: Hetzner Load Balancer + NGINX

Hetzner Load Balancer:
  Type: Layer 4 (TCP) or Layer 7 (HTTP/HTTPS)
  Algorithm: Round Robin, Least Connections, Weighted

  Features:
    - SSL termination
    - Health checks (HTTP, TCP)
    - Sticky sessions (cookie-based)
    - WebSocket support
    - Backend server management

  Configuration:
    Health Check:
      Protocol: HTTP
      Path: /health
      Port: 80/443
      Interval: 15 seconds
      Timeout: 10 seconds
      Unhealthy threshold: 3 failures
      Healthy threshold: 3 successes

    Sticky Sessions:
      Type: Cookie
      Cookie Name: HETZNER_LB
      Lifetime: 3600 seconds

NGINX Load Balancing:
  Configuration:
    upstream app_servers {
        least_conn;

        server app1.example.com:443 max_fails=3 fail_timeout=30s weight=5;
        server app2.example.com:443 max_fails=3 fail_timeout=30s weight=5;
        server app3.example.com:443 max_fails=3 fail_timeout=30s weight=1 backup;

        keepalive 32;
    }

    server {

```
    listen 443 ssl http2;

    location / {
        proxy_pass https://app_servers;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        proxy_connect_timeout 5s;
        proxy_send_timeout 60s;
        proxy_read_timeout 60s;
    }
}
```

Algorithms:
- Round Robin (default)
- Least Connections (least_conn)
- IP Hash (ip_hash) - for sticky sessions
- Random
- Weighted

High Availability:

Active-Active:
- Multiple load balancers
- Floating IP failover
- Health check monitoring

Active-Passive:
- Primary load balancer
- Standby ready to take over
- Keepalived for IP failover
- VRRP protocol

Session Persistence:

Methods:
- Cookie-based (application)
- IP hash (load balancer)
- Consistent hashing

Use Cases:

- Shopping carts
- User sessions
- WebSocket connections

8.3 Multi-Region Deployment

yaml

Geographic Regions:
  Primary: Hetzner Falkenstein (Germany)
  Secondary: Hetzner Helsinki (Finland)
  Tertiary: Hetzner Ashburn (USA)

Deployment Strategy:
  1. Active-Active (Multi-Master):
    - Websites replicated across regions
    - Database replication (async)
    - CDN for static assets
    - GeoDNS routing to nearest region
    - Use case: Global audience

  2. Active-Passive (DR):
    - Primary region handles all traffic
    - Secondary region is hot standby
    - Automatic failover on primary failure
    - Use case: Disaster recovery

  3. Regional Isolation:
    - Each region independent
    - Specific customers per region
    - Data residency compliance
    - Use case: Data sovereignty

Data Replication:
  Database:
    Method: PostgreSQL logical replication
    Direction: Primary → Replicas
    Lag: < 1 second (async) / 0 seconds (sync)

  Files:
    Method: Rsync + Lsyncd (real-time)
    Direction: Bidirectional or Primary → Replicas
    Frequency: Real-time or hourly

  Object Storage:
    Method: S3 cross-region replication
    Hetzner Object Storage supports multi-region

Failover Process:

Automatic Failover:

1. Health check detects primary failure
2. GeoDNS switches to secondary region
3. Load balancer updates backend pool
4. Database promotes replica to master
5. Notification sent to admins

Recovery:

1. Fix primary region issue
2. Sync data from secondary (if needed)
3. Switch back to primary (manual)
4. Validate data consistency

Monitoring:

- Cross-region latency
- Replication lag
- Data consistency checks
- Failover readiness tests (monthly)

---

9. Database Management

Detailed in Section 6.4 - Expanding here with multi-server aspects:

9.1 Database Server Configuration

yaml

```
Architecture: Dedicated database server(s)

Master-Slave Replication:
  Master Server:
    - Handles all writes
    - Primary for customer databases
    - Replicates to slaves

  Slave Servers:
    - Read replicas
    - Backup purposes
    - Failover targets
    - Analytics/reporting queries

  Replication:
    - Asynchronous (default)
    - Semi-synchronous (optional)
    - Lag monitoring: < 5 seconds target

Database Clustering (Future):
  - Galera Cluster (MySQL/MariaDB)
  - Patroni + etcd (PostgreSQL)
  - Multi-master writes
  - Automatic failover

Connection Pooling:
  Tool: PgBouncer (PostgreSQL) / ProxySQL (MySQL)
  Benefits:
    - Reduced connection overhead
    - Connection reuse
    - Query routing
    - Read/write split
```

9.2 Database Security

yaml

Access Control:
- No root remote access
- Database-specific users
- Least privilege principle
- IP whitelisting (application servers only)

Encryption:
- TLS for connections (enforced)
- Data-at-rest encryption (LUKS)
- Backup encryption (AES-256)

Audit Logging:
- All DDL operations logged
- Failed authentication attempts
- Privileged operations
- Query logging (slow queries)

---

10. Email Services

10.1 Email Server Components


yaml

SMTP Server: Postfix 3.8+
  Features:
    - TLS encryption (mandatory)
    - SMTP AUTH (SASL)
    - Relay restrictions
    - Sender verification
    - Recipient validation
    - Spam filtering (SpamAssassin)
    - Virus scanning (ClamAV)
    - DKIM signing
    - SPF checking
    - DMARC enforcement
    - Greylisting (optional)
    - Rate limiting

  Configuration:
    - Per-domain virtual mailboxes
    - PostgreSQL backend
    - Maildir format
    - Quotas enforced
    - Aliases support
    - Catch-all accounts

IMAP/POP3 Server: Dovecot 2.3+
  Features:
    - TLS encryption (mandatory)
    - Authentication (PAM, PostgreSQL)
    - Quota enforcement
    - Sieve filtering (server-side rules)
    - Push notifications (IMAP IDLE)
    - Full-text search (Solr optional)
    - Master/slave replication

  Protocols:
    - IMAP (port 143 + TLS, port 993 SSL)
    - POP3 (port 110 + TLS, port 995 SSL)
    - ManageSieve (port 4190)

Webmail: Roundcube 1.6+
  Features:
    - Modern responsive UI

- HTML email support
- Address book
- Calendar integration
- File attachments
- Multiple identities
- Filters/rules
- Plugins (calendar, carddav, etc.)

Access: webmail.yourdomain.com

Spam Filtering: SpamAssassin 4+
Features:
- Bayesian filtering
- DNS-based blocklists (DNSBLs)
- Header analysis
- Body analysis
- Auto-learning
- Custom rules
- Per-user preferences

Thresholds:
- Reject: Score > 15
- Mark as spam: Score > 5
- Allow: Score < 5

Antivirus: ClamAV
- Scans all attachments
- Quarantine infected emails
- Notify sender/recipient
- Daily signature updates

Authentication:
SPF (Sender Policy Framework):
- TXT record in DNS
- Specifies authorized sending IPs
- Prevents spoofing

DKIM (DomainKeys Identified Mail):
- Cryptographic signature
- Private key on server
- Public key in DNS (TXT record)

- Validates email integrity

DMARC (Domain-based Message Authentication):
- Policy published in DNS
- Specifies SPF/DKIM handling
- Reporting (aggregate, forensic)
- Policies: none, quarantine, reject

10.2 Email Account Management

yaml

Account Operations:

Create:

- Email address (local@domain.com)
- Password (strong, auto-generated)
- Quota (default 1 GB)
- Aliases
- Forwarding rules

Modify:

- Change password
- Update quota
- Add/remove aliases
- Configure forwarding
- Enable/disable account
- Set autoresponder (vacation)

Delete:

- Archive emails (before deletion)
- Remove from database
- Clean up disk space

Quota Management:

- Per-account limits
- Warning at 80% usage
- Reject emails at 100% usage
- Quota increase requests
- Usage statistics

Autoresponder (Vacation):

- Start/end date
- Custom message
- Reply frequency (once per sender per day)
- Subject line
- Exclude mailing lists

Email Forwarding:

- Forward to external address
- Forward and keep copy
- Multiple forwards
- Conditional forwarding (filter-based)

Mailing Lists:
  Tool: Mailman 3 (optional)
  Features:
    - Subscriber management
    - Moderation
    - Archives
    - Digests
    - Custom reply-to

---

11. DNS Management

11.1 DNS Infrastructure


yaml

DNS Server: PowerDNS Authoritative 4.8+
Backend: PostgreSQL
Clustering: Master-slave DNS servers

Features:
  - Authoritative DNS (not recursive)
  - DNSSEC support
  - GeoDNS (location-based routing)
  - Dynamic DNS (API-driven)
  - ALIAS records (CNAME for root)
  - CAA records (SSL control)
  - DNS templates
  - Bulk operations
  - Import/export zones

API: PowerDNS HTTP API
  - RESTful interface
  - Authentication (API keys)
  - Zone management
  - Record CRUD operations
  - DNSSEC operations

11.2 DNS Record Types

yaml

Supported Records:
A: IPv4 address
AAAA: IPv6 address
CNAME: Canonical name (alias)
MX: Mail exchanger
TXT: Text (SPF, DKIM, DMARC, verification)
NS: Nameserver
SOA: Start of authority
SRV: Service locator
CAA: Certificate authority authorization
PTR: Pointer (reverse DNS)
ALIAS: Virtual CNAME for root domain

Record Management:
Operations:
- Add record
- Edit record
- Delete record
- Bulk import (BIND format)
- Bulk export

Validation:
- Syntax checking
- Duplicate prevention
- TTL validation
- DNSSEC consistency

DNS Templates:
Predefined Templates:
- Default (A, MX, CNAME www)
- Email-only (MX, SPF, DKIM, DMARC)
- Hosting (A, AAAA, MX, TXT)
- Subdomain (CNAME)
- CDN (CNAME to CDN)

Custom Templates:
- User-defined record sets
- Variable substitution
- Apply to multiple domains

11.3 DNSSEC


yaml

```
DNSSEC Implementation:
  Key Types:
    - KSK (Key Signing Key): RSA 2048-bit or ECDSA P-256
    - ZSK (Zone Signing Key): RSA 1024-bit or ECDSA P-256

  Algorithm: 13 (ECDSA P-256) or 8 (RSA SHA-256)

  Operations:
    - Enable DNSSEC (automated)
    - Generate keys
    - Sign zone
    - DS record retrieval (for registrar)
    - Key rollover (ZSK: 90 days, KSK: 1 year)
    - Signature refresh (before expiry)

  Validation:
    - DNSSEC validation tests
    - DNSViz integration
    - Automatic error detection
    - Email alerts on validation failures
```

11.4 GeoDNS (Optional)


yaml

Purpose: Route users to nearest server

Implementation: PowerDNS GeoIP backend

Configuration:
  Continents:
    - NA (North America)
    - SA (South America)
    - EU (Europe)
    - AS (Asia)
    - AF (Africa)
    - OC (Oceania)

  Countries: Specific country codes
  Regions: State/province level
  Cities: City-level precision (expensive)

Use Cases:
  - CDN routing
  - Multi-region deployments
  - Latency optimization
  - Compliance (data residency)

Example:
  Query from Germany → Returns EU server IP
  Query from USA → Returns NA server IP
  Query from Australia → Returns OC server IP

---

12. Backup & Disaster Recovery

12.1 Backup Strategy

yaml

Backup Types:
  Full Backup:
    - Complete snapshot
    - Frequency: Weekly
    - Retention: 4 weeks

  Incremental Backup:
    - Changes since last backup
    - Frequency: Daily
    - Retention: 7 days
    - Space-efficient (deduplication)

  Real-Time Backup:
    - Continuous file syncing
    - Tool: Lsyncd
    - Use case: Critical data

Backup Scope:
  Website Files:
    - htdocs/ (web root)
    - private/ (above web root)
    - Excluded: cache/, tmp/

  Databases:
    - mysqldump (MySQL/MariaDB)
    - pg_dump (PostgreSQL)
    - Compressed (gzip)

  Email:
    - Maildir folders
    - Email configuration

  Configuration:
    - Site vhosts
    - PHP-FPM pools
    - Database credentials
    - Cron jobs

Backup Storage:
  Primary: Hetzner Object Storage (S3-compatible)
  Secondary: Local backup cache (for fast restores)

```
  Tertiary: Offsite backup (optional - AWS S3, Backblaze B2)

Backup Encryption:
  Algorithm: AES-256-GCM
  Key Management: Vault (HashiCorp) or encrypted keystore
  Encryption: At rest (always)
  Decryption: On restore only
```

12.2 Backup Implementation

Tool: Restic

yaml

Restic Features:
- Incremental backups
- Deduplication (chunk-level)
- Encryption (built-in)
- Compression
- Integrity checks (verification)
- Snapshots with tags
- Fast restore
- S3-compatible storage
- Low memory footprint
- Written in Go (fast)

Backup Process:
1. Schedule: Cron job (daily 2 AM)
2. Pre-backup: Database dumps
3. Backup: Restic snapshot creation
4. Upload: To Hetzner Object Storage
5. Verification: Integrity check
6. Cleanup: Remove old snapshots (retention policy)
7. Notification: Email report (success/failure)

Retention Policy:
Keep last 7 daily backups
Keep last 4 weekly backups
Keep last 12 monthly backups
Total retention: ~1 year

Backup Monitoring:
- Backup success/failure alerts
- Backup size trending
- Storage usage
- Time to complete
- Data change rate

12.3 Restore Procedures

yaml

Restore Types:

Full Site Restore:
- All files + database
- Use case: Complete site recovery
- Downtime: 5-15 minutes

Partial Restore:
- Specific files/folders
- Use case: Accidental deletion
- Downtime: None (restore to temp location)

Database Restore:
- Specific database
- Use case: Corrupted database
- Downtime: 2-5 minutes

Point-in-Time Restore:
- Restore to specific date/time
- Use case: Recover from hack/malware
- Downtime: 5-15 minutes

Restore Process:

1. Select backup snapshot (date/time)
2. Choose restore type (full/partial)
3. Select destination:
   - Original location (overwrite)
   - Temporary location (review first)
   - Different site (migration)
4. Execute restore:
   - Download from Object Storage
   - Decrypt and decompress
   - Extract to destination
   - Restore database (if included)
   - Fix permissions/ownership
5. Verify:
   - File integrity check
   - Test site functionality
   - Database consistency
6. Notification: Email confirmation

Restore Time:

- Small site (< 1 GB): 2-5 minutes
- Medium site (1-10 GB): 5-15 minutes
- Large site (> 10 GB): 15-60 minutes

Disaster Recovery Testing:
- Quarterly restore tests
- Verify backup integrity
- Test restore procedures
- Document any issues
- Update DR plan

12.4 Disaster Recovery Plan

yaml

Scenarios:

1. Server Hardware Failure:
   - Provision new server (Hetzner API)
   - Restore from backups
   - Update DNS (if IP changed)
   - RTO: 1-2 hours

2. Data Center Outage:
   - Failover to secondary region
   - GeoDNS automatic routing
   - RTO: 5-15 minutes

3. Ransomware/Crypto Malware:
   - Isolate infected server
   - Restore from clean backup (before infection)
   - Scan restored files
   - Investigate infection source
   - RTO: 2-4 hours

4. Database Corruption:
   - Promote replica to master
   - Or restore from backup
   - RTO: 15-30 minutes

5. Accidental Data Deletion:
   - Restore specific files from backup
   - RTO: 5-15 minutes

Recovery Time Objectives (RTO):
- Critical Services: 15 minutes
- High Priority: 1 hour
- Medium Priority: 4 hours
- Low Priority: 24 hours

Recovery Point Objectives (RPO):
- Databases: 1 hour (incremental backups)
- Files: 24 hours (daily backups)
- Configuration: 24 hours

DR Testing Schedule:
- Monthly: Restore random site (partial)

- Quarterly: Full disaster recovery drill
- Annually: Multi-region failover test

---

13. Monitoring & Observability

13.1 Monitoring Stack

yaml

Components:
  Prometheus:
    - Metrics collection
    - Time-series database
    - Alerting rules
    - Retention: 30 days

  Grafana:
    - Visualization
    - Dashboards
    - Custom queries (PromQL)
    - Multi-datasource support

  Loki:
    - Log aggregation
    - Time-series log database
    - Grafana integration
    - Retention: 90 days

  Vector:
    - Log collection
    - Log parsing
    - Log routing
    - Performance: Written in RUST

  Alertmanager:
    - Alert routing
    - Deduplication
    - Silencing
    - Notification channels

Exporters:
  Node Exporter:
    - CPU, memory, disk, network metrics
    - Systemd service metrics
    - Hardware sensors

  NGINX Exporter:
    - HTTP requests/responses
    - Connections
    - Response times

PostgreSQL Exporter:
- Connection count
- Query performance
- Database sizes
- Replication lag

Redis Exporter:
- Memory usage
- Key count
- Hit/miss ratio

PHP-FPM Exporter:
- Process count
- Request queue
- Slow requests

Custom Exporters (RUST/Go):
- Application-specific metrics
- Business metrics (signups, revenue)
- Security metrics (attacks blocked)

13.2 Monitoring Dashboards

yaml

Infrastructure Dashboard:

Panels:

- Server health (up/down status)
- CPU usage per server
- Memory usage per server
- Disk usage per server
- Network traffic (inbound/outbound)
- Load average
- Swap usage
- I/O wait
- Open file descriptors
- Network connections

Application Dashboard:

Panels:

- HTTP request rate
- HTTP error rate (4xx, 5xx)
- Response time (p50, p95, p99)
- Active PHP-FPM processes
- Database connections
- Cache hit ratio (Redis, OPcache)
- Queue lengths
- Slow queries count

Security Dashboard:

Panels:

- WAF blocks (per hour)
- Failed login attempts
- Firewall drops
- Malware detections
- DDoS events
- SSL cert expiry countdown
- Suspicious IP list
- Brute force attempts

Business Dashboard:

Panels:

- Active customers
- New signups (daily, weekly, monthly)
- Revenue (MRR, ARR)
- Churn rate

- Average revenue per user (ARPU)
- Support ticket count
- Uptime percentage
- Customer satisfaction score

13.3 Alerting Rules

yaml

Alert Severity Levels:

Critical (P1):

- Service down
- Database unreachable
- DDoS attack detected
- Root compromise suspected
- All backups failed

Notification: SMS + Email + Slack + PagerDuty
Response Time: Immediate

High (P2):

- High CPU usage (>90% for 5 min)
- High memory usage (>95%)
- Disk space low (<10%)
- SSL cert expiring in 7 days
- Backup failed (1 instance)
- High error rate (>5%)

Notification: Email + Slack
Response Time: 15 minutes

Medium (P3):

- High load average (>10)
- Slow queries detected
- High PHP-FPM queue
- Moderate disk space (<20%)
- SSL cert expiring in 30 days

Notification: Email
Response Time: 1 hour

Low (P4):

- Software updates available
- Non-critical service restart
- Scheduled maintenance reminder

Notification: Daily digest email
Response Time: Best effort

Alert Examples:

```yaml
HighCPUUsage:
  Condition: avg(cpu_usage) > 90 for 5 minutes
  Action: Notify admins, investigate processes

DiskSpaceLow:
  Condition: disk_free < 10%
  Action: Notify admins, cleanup suggestions

WebsiteDown:
  Condition: http_status != 200 for 3 checks
  Action: Notify customer + admin, check logs

SSLExpiring:
  Condition: ssl_days_until_expiry < 7
  Action: Trigger renewal, notify if fails

HighErrorRate:
  Condition: error_rate > 5% for 10 minutes
  Action: Notify admins, check error logs
```

13.4 Health Checks


yaml

Service Health Endpoints:
  /health:
    - Simple UP/DOWN status
    - Response: 200 OK or 503 Service Unavailable

  /health/detailed:
    - Component-wise status
    - Response: JSON with details

```
{
  "status": "healthy",
  "components": {
    "database": "healthy",
    "redis": "healthy",
    "storage": "healthy",
    "api": "healthy"
  },
  "uptime": 123456,
  "version": "1.0.0"
}
```

Health Check Frequency:
  Internal: 10 seconds
  External (monitoring service): 1 minute
  Load balancer: 15 seconds

Checks Performed:
  - Process running
  - Port responding
  - HTTP response code
  - Response time < threshold
  - Database query (simple SELECT 1)
  - Redis ping
  - Disk space > minimum
  - Memory available

---

14. API & Integration Framework

14.1 RESTful API

yaml

API Design: REST + OpenAPI 3.1 specification

Base URL: https://api.yourplatform.com/v1/

Authentication:
  Methods:
    - API Key (X-API-Key header)
    - OAuth 2.0 (client credentials)
    - JWT tokens (user sessions)

  Scopes:
    - read: Read-only access
    - write: Create/update resources
    - delete: Delete resources
    - admin: Full access

Rate Limiting:
  Tiers:
    Free: 60 requests/hour
    Basic: 1000 requests/hour
    Pro: 10000 requests/hour
    Enterprise: Unlimited (fair use)

  Headers:
    X-RateLimit-Limit: Total allowed requests
    X-RateLimit-Remaining: Remaining requests
    X-RateLimit-Reset: Unix timestamp of reset

  Exceeded: 429 Too Many Requests

API Endpoints (Examples):
  Customer Management:
    GET /customers - List customers
    GET /customers/{id} - Get customer details
    POST /customers - Create customer
    PUT /customers/{id} - Update customer
    DELETE /customers/{id} - Delete customer

  Service Management:
    GET /services - List services
    GET /services/{id} - Get service details

POST /services - Create service
PUT /services/{id} - Update service
DELETE /services/{id} - Delete service

Billing:
GET /invoices - List invoices
GET /invoices/{id} - Get invoice details
POST /invoices - Create invoice
POST /invoices/{id}/pay - Record payment

Domain Management:
GET /domains - List domains
GET /domains/{id} - Get domain details
POST /domains - Register domain
PUT /domains/{id} - Update domain
POST /domains/{id}/renew - Renew domain

DNS Management:
GET /domains/{id}/dns - List DNS records
GET /dns/{id} - Get DNS record
POST /domains/{id}/dns - Create DNS record
PUT /dns/{id} - Update DNS record
DELETE /dns/{id} - Delete DNS record

Hosting Management:
GET /sites - List hosting sites
GET /sites/{id} - Get site details
POST /sites - Create site
PUT /sites/{id} - Update site
DELETE /sites/{id} - Delete site

Database Management:
GET /sites/{id}/databases - List databases
POST /sites/{id}/databases - Create database
GET /databases/{id} - Get database details
DELETE /databases/{id} - Delete database

Backup Management:
GET /sites/{id}/backups - List backups
POST /sites/{id}/backups - Create backup
POST /backups/{id}/restore - Restore backup

```
Response Format:
  Success (200 OK):
    {
      "success": true,
      "data": { ... },
      "meta": {
        "pagination": {
          "total": 100,
          "page": 1,
          "per_page": 20
        }
      }
    }

  Error (4xx/5xx):
    {
      "success": false,
      "error": {
        "code": "INVALID_INPUT",
        "message": "Invalid email address",
        "details": { ... }
      }
    }

API Documentation:
  - OpenAPI/Swagger UI
  - Interactive testing
  - Code examples (cURL, Python, Go, PHP, JavaScript)
  - Postman collection
```

14.2 Webhooks

yaml

Webhook Events:
  Customer:
    - customer.created
    - customer.updated
    - customer.deleted

  Service:
    - service.created
    - service.activated
    - service.suspended
    - service.cancelled

  Billing:
    - invoice.created
    - invoice.paid
    - invoice.overdue
    - payment.received
    - payment.failed

  Domain:
    - domain.registered
    - domain.transferred
    - domain.renewed
    - domain.expired

  Site:
    - site.created
    - site.deleted
    - backup.completed
    - backup.failed

  Security:
    - malware.detected
    - attack.blocked
    - ssl.expiring
    - ssl.renewed

Webhook Configuration:
  Setup:
    - URL endpoint (customer provides)
    - Events to subscribe

- Secret for signature verification
- Retry policy
- Active/inactive toggle

Payload:
POST to configured URL
Content-Type: application/json
Headers:
X-Webhook-Signature: HMAC-SHA256(secret, payload)
X-Webhook-Event: event_name
X-Webhook-ID: unique_id
X-Webhook-Timestamp: unix_timestamp

Body:

```
{
  "event": "service.created",
  "data": { ... },
  "timestamp": 1699999999
}
```

Retry Policy:
- Retry on failure (non-200 response)
- Exponential backoff (1s, 2s, 4s, 8s, 16s)
- Max 5 retries
- Mark as failed after max retries
- Manual retry available

Webhook Security:
Signature Verification:
- Calculate HMAC-SHA256 of payload + secret
- Compare with X-Webhook-Signature header
- Reject if mismatch

IP Whitelisting:
- Configure allowed IP ranges
- Reject requests from other IPs

HTTPS Only:
- Enforce HTTPS URLs
- Reject HTTP webhooks

14.3 CLI Tool

yaml

CLI Name: hcc (Hetzner Cloud Commander)

Installation:
curl -sSL https://get.yourplatform.com/hcc.sh | bash

Or download binary for your OS:
- Linux (amd64, arm64)
- macOS (amd64, arm64)
- Windows (amd64)

Configuration:
hcc config set api-key YOUR_API_KEY
hcc config set api-url https://api.yourplatform.com/v1

Commands:
Authentication:
hcc login - Interactive login
hcc logout - Clear credentials

Customer Management:
hcc customer list
hcc customer get <id>
hcc customer create --name "John Doe" --email "john@example.com"
hcc customer update <id> --name "Jane Doe"
hcc customer delete <id>

Service Management:
hcc service list
hcc service get <id>
hcc service create --package shared-basic --domain example.com
hcc service suspend <id>
hcc service unsuspend <id>
hcc service cancel <id>

Site Management:
hcc site list
hcc site create --domain example.com --type wordpress
hcc site delete <id>
hcc site backup <id>
hcc site restore <id> --backup <backup_id>

```
Database Management:
  hcc db list --site <site_id>
  hcc db create --site <site_id> --name mydb
  hcc db delete <id>

DNS Management:
  hcc dns list --domain example.com
  hcc dns add --domain example.com --type A --name @ --value 1.2.3.4
  hcc dns update <id> --value 5.6.7.8
  hcc dns delete <id>

Server Management (Admin):
  hcc server list
  hcc server add --role application --region fsn1
  hcc server remove <id>

Output Format:
  - Table (default)
  - JSON (--output json)
  - YAML (--output yaml)

Features:
  - Tab completion (bash, zsh, fish)
  - Colored output
  - Progress indicators
  - Error handling with helpful messages
  - Confirmation prompts for destructive actions
```

---

15. Automation Systems

15.1 n8n Workflow Automation

yaml

n8n Deployment:
  Type: Self-hosted
  Version: Latest (1.x)
  Database: PostgreSQL
  Access: https://n8n.yourplatform.com (internal)

Predefined Workflows:
  1. Customer Onboarding:
    Trigger: New customer created (webhook)
    Steps:
      - Send welcome email
      - Create default services (if applicable)
      - Add to CRM (optional)
      - Schedule follow-up email (3 days)

  2. Invoice Generation:
    Trigger: Schedule (1st of month, 00:00)
    Steps:
      - Query active services
      - Calculate charges
      - Generate invoices
      - Email invoices to customers
      - Update accounting system

  3. Payment Processing:
    Trigger: Payment gateway webhook (Stripe, PayPal)
    Steps:
      - Validate payment
      - Mark invoice as paid
      - Send receipt email
      - Activate/extend service
      - Log transaction

  4. Service Provisioning:
    Trigger: Service created + paid
    Steps:
      - Allocate server
      - Create hosting account (via API)
      - Setup DNS records
      - Issue SSL certificate
      - Send credentials email

- Create backup schedule

5. Expiration Reminder:
   Trigger: Schedule (daily check)
   Steps:
   - Query services expiring in 7, 3, 1 days
   - Send reminder emails
   - For unpaid: Send overdue notice
   - For 30 days overdue: Suspend service

6. Malware Detection Response:
   Trigger: Malware scan webhook (detection)
   Steps:
   - Quarantine infected files
   - Disable site (optional)
   - Send alert email to customer
   - Create support ticket
   - Schedule cleanup task

7. SSL Certificate Renewal:
   Trigger: Schedule (daily check)
   Steps:
   - Query certs expiring in 30 days
   - Request renewal (Let's Encrypt)
   - Install new certificate
   - Verify installation
   - Send notification on success/failure

8. Backup Verification:
   Trigger: Backup completed (webhook)
   Steps:
   - Verify backup integrity
   - Check backup size (anomaly detection)
   - Send notification on failure
   - Update backup status in database

9. Support Ticket Routing:
   Trigger: New support ticket (webhook)
   Steps:
   - Analyze ticket content (keywords)
   - Assign to department

- Set priority based on keywords
- Send acknowledgment email
- Notify assigned staff

10. Affiliate Commission:
Trigger: Payment received (webhook)
Steps:
- Check if customer referred
- Calculate commission (%)
- Add to affiliate balance
- Send commission report email
- Threshold met: Generate payout

Custom Workflows:
- User can create via n8n UI
- Access control (admin only)
- Version control (Git integration)
- Testing environment

Integration Nodes:
- HTTP Request
- Webhook
- PostgreSQL
- Redis
- Stripe
- PayPal
- Email (SMTP)
- SMS (Twilio)
- Slack
- Discord
- Google Sheets
- Custom nodes (RUST/Go services)

15.2 Ansible Automation

yaml

Ansible Usage:
  Purpose: Server provisioning, configuration management, deployment
  Version: Ansible 2.15+
  Control Node: Management server (not target servers)
  Inventory: Dynamic (generated from control panel database)

Playbooks:
  1. server_provision.yml:
    Purpose: Initial server setup
    Tasks:
      - Create Hetzner Cloud instance (via API)
      - Wait for SSH availability
      - Update system packages
      - Configure hostname
      - Setup timezone
      - Create admin user
      - Configure SSH (disable root, key-only)
      - Setup firewall (nftables)
      - Install base packages
      - Configure logging
      - Setup monitoring agent
      - Join private network
      - Register with control panel

    Variables:
      - server_role: application | database | email | dns
      - server_region: fsn1 | hel1 | ash
      - server_type: cpx21 | cpx31 | cpx41
      - ssh_public_key: ...

    Execution Time: 5-10 minutes

  2. application_server.yml:
    Purpose: Configure application server role
    Tasks:
      - Install NGINX
      - Install PHP-FPM (multiple versions)
      - Install Node.js (via nvm)
      - Install Python
      - Install Git
      - Install Composer

- Install WP-CLI
- Configure PHP-FPM pools (templates)
- Configure NGINX (base config)
- Setup log rotation
- Install Varnish Cache (optional)
- Configure Redis (local instance)
- Setup monitoring exporters

Execution Time: 10-15 minutes

3. security_hardening.yml:
Purpose: Apply security best practices
Tasks:
- Configure nftables (default deny)
- Install Fail2Ban
- Configure ModSecurity + OWASP CRS
- Setup ClamAV
- Configure automatic security updates
- Disable unnecessary services
- Configure audit logging (auditd)
- Set file permissions
- Configure SELinux/AppArmor
- Setup AIDE (file integrity monitoring)
- Configure sysctl security parameters

Execution Time: 5-10 minutes

4. ssl_certificate.yml:
Purpose: Issue and install SSL certificate
Tasks:
- Verify domain DNS
- Request Let's Encrypt certificate
- Install certificate
- Configure NGINX SSL
- Test SSL configuration
- Setup auto-renewal cron

Execution Time: 1-2 minutes

5. site_deploy.yml:
Purpose: Deploy a new website

Tasks:
- Create Unix user
- Setup directory structure
- Create NGINX vhost
- Create PHP-FPM pool
- Setup SSL certificate
- Configure DNS (if using platform DNS)
- Create database (if required)
- Setup backup schedule
- Configure monitoring

Execution Time: 1-2 minutes

6. system_update.yml:
Purpose: Update software packages
Tasks:
- Update package lists
- Upgrade packages (security only or all)
- Reboot if kernel updated (optional)
- Verify services after reboot

Rolling Update:
- One server at a time
- Wait for health check before next
- Automatic rollback on failure

Execution Time: 5-30 minutes per server

7. backup_restore.yml:
Purpose: Restore from backup
Tasks:
- Download backup from Object Storage
- Extract backup
- Restore files
- Restore database
- Fix permissions
- Verify restoration

Execution Time: 5-30 minutes (depends on size)

Ansible Roles:

```
  - common: Base configuration
  - security: Hardening tasks
  - nginx: Web server setup
  - php: PHP-FPM configuration
  - mysql: Database server
  - postgresql: Database server
  - postfix: Email server (SMTP)
  - dovecot: Email server (IMAP/POP3)
  - powerdns: DNS server
  - monitoring: Prometheus exporters

Dynamic Inventory:
  Source: PostgreSQL database (server list)
  Groups:
    - control_nodes
    - application_servers
    - database_servers
    - email_servers
    - dns_servers
    - backup_servers

  Script: inventory.py (queries database)

Secrets Management:
  Tool: Ansible Vault
  Encrypted Variables:
    - Database passwords
    - API keys
    - SSL private keys
    - Encryption keys

  Vault Password: Stored securely (env var or file)
```

15.3 Bash Shell Scripting

yaml

Scripts Location: /opt/platform/scripts/

System Maintenance Scripts:
  /opt/platform/scripts/cleanup.sh:
    Purpose: Clean up temporary files, old logs
    Schedule: Daily (via cron)
    Tasks:
      - Delete files in /tmp older than 7 days
      - Rotate logs
      - Clean package manager cache
      - Clean old backups (local cache)
      - Vacuum databases

  /opt/platform/scripts/health_check.sh:
    Purpose: System health verification
    Schedule: Every 5 minutes
    Tasks:
      - Check disk space
      - Check memory usage
      - Check process count
      - Check service status
      - Alert if thresholds exceeded

  /opt/platform/scripts/backup.sh:
    Purpose: Execute backup jobs
    Schedule: Daily (2 AM)
    Tasks:
      - Iterate through accounts
      - Dump databases
      - Create Restic snapshot
      - Upload to Object Storage
      - Verify backup
      - Send report

  /opt/platform/scripts/ssl_renew.sh:
    Purpose: Renew expiring SSL certificates
    Schedule: Daily
    Tasks:
      - Query certs expiring in 30 days
      - Request renewal (Let's Encrypt)
      - Install renewed certificates

- Reload NGINX
- Send notification

/opt/platform/scripts/security_scan.sh:
Purpose: Security scanning
Schedule: Weekly
Tasks:
- Run ClamAV scan
- Run malware detect (LMD)
- Check file integrity (AIDE)
- Check for rootkits (rkhunter)
- Generate security report

Site Management Scripts:
/opt/platform/scripts/site_create.sh:
Purpose: Create new hosting site
Called by: Provisioning service (Go)
Parameters: domain, user, php_version, site_type
Tasks:
- Create Unix user
- Setup directories
- Create NGINX vhost
- Create PHP-FPM pool
- Setup SSL (Let's Encrypt)
- Reload services

/opt/platform/scripts/site_delete.sh:
Purpose: Delete hosting site
Called by: Provisioning service (Go)
Parameters: domain, user
Tasks:
- Backup before deletion (optional)
- Remove NGINX vhost
- Remove PHP-FPM pool
- Delete Unix user (and files)
- Reload services

Database Scripts:
/opt/platform/scripts/db_create.sh:
Purpose: Create MySQL database
Parameters: db_name, db_user, db_pass

Tasks:
- Create database
- Create user
- Grant privileges
- Flush privileges

/opt/platform/scripts/db_backup.sh:
Purpose: Backup all databases
Schedule: Daily (1 AM)
Tasks:
- Iterate through databases
- mysqldump each database
- Compress with gzip
- Move to backup directory
- Delete dumps older than 7 days

Email Scripts:
/opt/platform/scripts/email_create.sh:
Purpose: Create email account
Parameters: email, password, quota
Tasks:
- Add to virtual mailbox table
- Create Maildir
- Set permissions
- Set quota

/opt/platform/scripts/email_quota.sh:
Purpose: Check email quotas
Schedule: Daily
Tasks:
- Calculate mailbox sizes
- Update database
- Send warnings at 80%, 90%, 100%

Monitoring Scripts:
/opt/platform/scripts/monitor_websites.sh:
Purpose: Check website availability
Schedule: Every 5 minutes
Tasks:
- Iterate through active sites
- HTTP GET request

```
    - Check response code, time
    - Alert if down

  /opt/platform/scripts/monitor_resources.sh:
    Purpose: Resource usage monitoring
    Schedule: Every minute
    Tasks:
      - Collect CPU, RAM, Disk, Network
      - Send to monitoring service
      - Alert on thresholds

Utility Scripts:
  /opt/platform/scripts/migrate_site.sh:
    Purpose: Migrate site between servers
    Parameters: domain, from_server, to_server
    Tasks:
      - Backup on source server
      - Transfer backup to destination
      - Restore on destination
      - Update DNS (if needed)
      - Verify migration
      - Delete from source (optional)
```

15.4 Python Automation Scripts

yaml

Scripts Location: /opt/platform/python/

Invoice Generation (invoice_generator.py):
  Purpose: Generate monthly invoices
  Schedule: Cron (1st of month)
  Libraries: psycopg2, jinja2, pdfkit
  Tasks:
    - Query active services
    - Calculate charges (base + usage)
    - Apply discounts/coupons
    - Generate PDF invoices
    - Send via email
    - Update database

Report Generator (reports.py):
  Purpose: Generate analytics reports
  Schedule: Weekly/Monthly
  Libraries: pandas, matplotlib, seaborn
  Tasks:
    - Query database for metrics
    - Analyze data (trends, growth)
    - Generate charts (usage, revenue)
    - Create PDF report
    - Email to admins

Data Migration (migrate_data.py):
  Purpose: Migrate data between systems
  Use Case: Import from cPanel, Plesk, etc.
  Libraries: paramiko, mysql-connector
  Tasks:
    - Connect to source system
    - Extract accounts, databases, emails
    - Transform data (normalize)
    - Load into platform
    - Verify migration

Billing Reconciliation (reconcile.py):
  Purpose: Reconcile payments with bank statements
  Schedule: Daily
  Libraries: pandas, csv
  Tasks:

- Import bank statement (CSV)
- Match with payment records
- Identify discrepancies
- Generate reconciliation report

Email Campaign (email_campaign.py):
Purpose: Send bulk emails (announcements, offers)
Libraries: smtplib, email, jinja2
Tasks:
- Query recipient list
- Render email templates
- Send in batches (rate limit)
- Track open/click rates (optional)
- Unsubscribe handling

Provisioning Automation (provision.py):
Purpose: Complex provisioning workflows
Libraries: requests, paramiko
Tasks:
- API calls to create resources
- SSH commands for configuration
- Wait for services to start
- Health checks
- Rollback on failure

Security Scanner (security_scanner.py):
Purpose: Scan for vulnerabilities
Schedule: Weekly
Libraries: requests, beautifulsoup4, lxml
Tasks:
- Crawl websites
- Check for common vulnerabilities
- SQL injection tests
- XSS tests
- Outdated software detection
- Generate report

API Client (api_client.py):
Purpose: Interact with third-party APIs
Libraries: requests, json
Examples:

- Domain registrar API
- Payment gateway API
- SMS gateway API
- Threat intelligence feeds
- Currency exchange rates

---

16. Performance Optimization

16.1 Caching Strategy

yaml

Caching Layers:

1. Varnish Cache (Optional):
   - Full-page caching
   - HTTP accelerator
   - Sits in front of NGINX
   - Cache hit: Serve directly (bypass NGINX/PHP)
   - Cache miss: Forward to NGINX

   Configuration:
   - Cache static assets (images, CSS, JS)
   - Cache HTML (with rules)
   - Respect Cache-Control headers
   - Purge on content update
   - Grace mode (serve stale on backend failure)

   Performance: 100-250x faster than dynamic generation

2. Redis Object Cache:
   - WordPress object cache
   - Session storage (PHP sessions)
   - Application cache

   Configuration:
   - Separate Redis instance per customer (isolation)
   - LRU eviction policy
   - Maxmemory limit: 256MB per instance

   Performance: Sub-millisecond response time

3. OPcache (PHP):
   - Opcodes caching
   - Eliminates PHP compilation
   - In-memory cache

   Configuration:

```
opcache.enable=1
opcache.memory_consumption=128
opcache.interned_strings_buffer=8
opcache.max_accelerated_files=10000
opcache.revalidate_freq=60
opcache.validate_timestamps=1
```

Performance: 3-5x faster PHP execution

4. Browser Cache:
- Cache-Control headers
- ETags
- Expires headers

Configuration (NGINX):
Static assets: max-age=31536000 (1 year)
HTML: max-age=3600 (1 hour)
API responses: no-cache (verify each time)

Content Delivery Network (CDN):
Integration: Cloudflare, BunnyCDN, StackPath

Benefits:
- Edge caching (global)
- DDoS protection
- SSL termination
- Image optimization
- Bandwidth savings

Setup:
- DNS points to CDN
- CDN origin: Platform servers
- Cache rules configuration
- Purge API integration

16.2 Database Optimization

yaml

Query Optimization:
- Proper indexing (analyze slow queries)
- Avoid SELECT * (select needed columns)
- Use EXPLAIN for query plans
- Optimize JOINs
- Use LIMIT for large result sets
- Batch INSERT/UPDATE operations

Connection Pooling:
Tool: PgBouncer (PostgreSQL) / ProxySQL (MySQL)

Benefits:
- Reuse connections
- Reduce overhead
- Handle more concurrent clients

Configuration:
pool_mode: transaction
default_pool_size: 20
max_client_conn: 100
max_db_connections: 50

Database Replication:
Master-Slave:
- Master: All writes
- Slaves: Read replicas
- Asynchronous replication

Read/Write Splitting:
- Application routes reads to slaves
- Writes go to master only
- Load balanced reads (multiple slaves)

Partitioning:
Strategy: Time-based partitioning
Use Case: Large tables (logs, metrics)

Example:
- Table: access_logs
- Partition by: month (access_logs_2025_01, _02, etc.)
- Old partitions: Archived or dropped

Vacuum & Analyze:
PostgreSQL:
- Autovacuum enabled
- Manual VACUUM ANALYZE on large operations
- REINDEX periodically

MySQL:
- OPTIMIZE TABLE regularly
- ANALYZE TABLE after bulk operations

Query Caching:
- Application-level caching (Redis)
- Prepared statements (reduce parsing)
- Materialized views (pre-computed results)

16.3 Web Server Optimization


yaml

NGINX Configuration:

Worker Processes:

worker_processes auto; (one per CPU core)

worker_connections 2048;

Keepalive:

keepalive_timeout 65;

keepalive_requests 100;

Buffers:

client_body_buffer_size 128k;

client_max_body_size 100m;

client_header_buffer_size 1k;

large_client_header_buffers 4 8k;

Timeouts:

client_body_timeout 12;

client_header_timeout 12;

send_timeout 10;

Compression:

gzip on;

gzip_comp_level 5;

gzip_types text/plain text/css application/json application/javascript;

gzip_vary on;

brotli on;

brotli_comp_level 6;

brotli_types text/plain text/css application/json application/javascript;

HTTP/2 & HTTP/3:

listen 443 ssl http2;

listen 443 quic reuseport;

http3 on;

Static File Handling:

- sendfile on;
- tcp_nopush on;
- tcp_nodelay on;
- Open file cache (reduces syscalls)

PHP-FPM Optimization:
  Process Manager:
    pm = dynamic
    pm.max_children = auto-calculated (RAM-based)
    pm.start_servers = 25% of max_children
    pm.min_spare_servers = 25% of max_children
    pm.max_spare_servers = 75% of max_children
    pm.max_requests = 500 (recycle after X requests)

  Memory:
    memory_limit = 256M (per request)

  Timeout:
    max_execution_time = 60
    request_terminate_timeout = 60

Asset Optimization:
  Images:
    - WebP format (smaller, better quality)
    - Lazy loading (defer offscreen images)
    - Responsive images (srcset)
    - Image CDN (optimization on-the-fly)

  CSS/JavaScript:
    - Minification (remove whitespace, comments)
    - Concatenation (fewer HTTP requests)
    - Defer non-critical JS
    - Async loading
    - Critical CSS inlining

  Fonts:
    - WOFF2 format (best compression)
    - font-display: swap (avoid FOIT)
    - Subset fonts (only needed characters)

16.4 Application-Level Optimization

yaml

RUST Core Services:
  Optimizations:
    - Zero-copy operations
    - Memory pooling
    - Async I/O (Tokio runtime)
    - Minimal allocations
    - Profile-guided optimization (PGO)

  Benchmarks:
    - Request handling: < 1ms (p99)
    - Memory usage: < 10MB per service
    - CPU usage: < 5% (idle)

Go Microservices:
  Optimizations:
    - Goroutine pooling
    - Sync.Pool for reusable objects
    - Efficient JSON parsing (jsoniter)
    - Database connection pooling
    - Context-based timeouts

  Benchmarks:
    - API response: < 50ms (p95)
    - Memory: < 50MB per service
    - Concurrent requests: 10,000+

Frontend (HTMX):
  Optimizations:
    - Server-side rendering (fast TTFB)
    - Minimal JavaScript (< 50KB total)
    - CSS purging (remove unused)
    - HTTP/2 push (critical resources)
    - Service worker (offline capability)

  Metrics:
    - Lighthouse Score: 95+ (all metrics)
    - First Contentful Paint: < 1s
    - Time to Interactive: < 2s
    - Total page size: < 500KB

---

17. Development Roadmap

17.1 Phase 1: Foundation (Months 1-4)

yaml

Month 1: Core Infrastructure
- Setup development environment
- Hetzner Cloud account & networking
- Git repository structure
- CI/CD pipeline (GitHub Actions / GitLab CI)
- Database schema design
- API specification (OpenAPI)

Month 2: Authentication & Core Services
- User authentication system (JWT)
- RBAC (Role-Based Access Control)
- API gateway (RUST - Actix/Axum)
- Session management (Redis)
- 2FA implementation (TOTP)

Month 3: Billing Foundation
- Customer management (Go service)
- Invoice generation
- Payment gateway integration (Stripe)
- Product/package management
- Basic reporting

Month 4: Control Panel Core
- Admin dashboard (HTMX frontend)
- Server management
- Site provisioning (basic)
- Database creation
- File manager (basic)

Deliverables:
- ✅ Functional admin panel
- ✅ User can create accounts
- ✅ Basic billing system
- ✅ Simple hosting provisioning

Testing:
- Unit tests (>70% coverage)
- Integration tests

- Manual testing
- Alpha release (internal)

17.2 Phase 2: Enhanced Features (Months 5-8)

yaml

Month 5: Email & DNS
- Email server setup (Postfix, Dovecot)
- Email account management
- Webmail (Roundcube)
- DNS server (PowerDNS)
- DNS management interface
- SPF/DKIM/DMARC automation

Month 6: Advanced Hosting
- Multi-application support (WordPress, PHP, Node.js, Python)
- SSL automation (Let's Encrypt)
- Staging environments
- Git integration
- WP-CLI integration
- One-click installers

Month 7: Support System
- Ticket system (Go service)
- Knowledge base
- Email piping to tickets
- SLA tracking
- Canned responses

Month 8: Client Portal
- Client dashboard (HTMX)
- Service management
- Billing history
- Support ticket interface
- Account settings
- 2FA setup

Deliverables:
- ✅ Complete hosting stack
- ✅ Email services functional
- ✅ DNS management
- ✅ Client self-service portal
- ✅ Support ticket system

Testing:
- Beta release (selected customers)
- Load testing

- Security audit
- Bug fixes

17.3 Phase 3: Security Implementation (Months 9-12)

yaml

Month 9: Web Application Firewall
- ModSecurity integration
- OWASP CRS 4.x
- Custom rule engine
- WAF dashboard
- Threat intelligence feeds

Month 10: PHP Hardening & Isolation
- PHP version selector
- Disabled functions management
- Resource limits (cgroups v2)
- Namespace isolation
- Automatic security patching

Month 11: Firewall & IDS
- nftables configuration
- Fail2Ban integration
- GeoIP filtering
- DDoS mitigation
- Intrusion detection
- Security alerts

Month 12: Malware Scanning
- ClamAV integration
- Linux Malware Detect (LMD)
- Custom ML-based scanner (RUST)
- Real-time scanning
- Scheduled scans
- Quarantine & cleanup

Deliverables:
- ✅ Enterprise-grade security
- ✅ Hardened PHP environment
- ✅ Multi-layer firewall
- ✅ Malware protection
- ✅ Automated threat response

Testing:
- Penetration testing
- Security audit (3rd party)

- Vulnerability scanning
- Compliance review (PCI DSS)

17.4 Phase 4: Advanced Automation (Months 13-16)

yaml

Month 13: n8n Workflow Automation
- n8n deployment
- Predefined workflows (10+)
- Webhook integrations
- Custom nodes (if needed)
- Workflow marketplace

Month 14: Ansible Automation
- Server provisioning playbooks
- Configuration management
- Security hardening playbooks
- Deployment automation
- Rolling updates

Month 15: Monitoring & Observability
- Prometheus + Grafana
- Loki for logs
- Vector for log collection
- Custom dashboards (10+)
- Alerting rules (50+)
- PagerDuty integration

Month 16: API & CLI
- Complete RESTful API
- Webhook system
- CLI tool (hcc)
- API documentation (Swagger UI)
- SDKs (Python, Go, PHP, JavaScript)

Deliverables:
- ✅ Comprehensive automation
- ✅ Full observability
- ✅ Developer-friendly API
- ✅ CLI tool
- ✅ Advanced workflows

Testing:
- API load testing
- Automation testing

- Documentation review
- User acceptance testing (UAT)

17.5 Phase 5: Scaling & Optimization (Months 17-20)

yaml

Month 17: Multi-Server Management
- Server role system
- Load balancing (Hetzner LB + NGINX)
- Multi-region support
- Server migration tools
- Failover automation

Month 18: Performance Optimization
- Varnish Cache integration
- Database optimization
- Query caching
- CDN integration (Cloudflare)
- Asset optimization

Month 19: Reseller System
- Reseller hierarchy
- White-label portals
- Commission system
- Resource allocation
- Custom pricing

Month 20: Finalization
- Bug fixes
- Performance tuning
- Documentation (user & developer)
- Video tutorials
- Marketing materials
- Production deployment

Deliverables:
- ✅ Production-ready platform
- ✅ Scalable architecture
- ✅ Complete documentation
- ✅ Marketing materials
- ✅ 1.0 Release

Testing:
- Load testing (1000+ sites)
- Disaster recovery testing
- Multi-region failover

- Final security audit
- Performance benchmarking

17.6 Post-Launch (Month 21+)


yaml

Continuous Improvement:
- Bug fixes (ongoing)
- Security patches (weekly)
- Feature requests (community-driven)
- Performance optimization
- Scalability improvements

Roadmap Items:
- Kubernetes support
- Container orchestration
- AI-powered threat detection
- Predictive scaling
- Advanced analytics
- Mobile app (iOS, Android)
- Marketplace (templates, plugins)
- Third-party integrations (>50)
- Multi-cloud support (AWS, GCP, Azure)
- Edge computing features

---

18. Appendices

Appendix A: Technology Comparison


yaml

RUST vs Go vs PHP:

Performance:

RUST: Highest (no GC, zero-cost abstractions)

Go: High (efficient GC, compiled)

PHP: Medium (interpreted, slower)

Memory Safety:

RUST: Best (ownership system, no null)

Go: Good (GC prevents most issues)

PHP: Poor (manual memory management)

Concurrency:

RUST: Async/await, threads

Go: Goroutines (lightweight threads)

PHP: Limited (forks, async via ext)

Use Cases in Platform:

RUST: Core engine, security, monitoring

Go: Microservices, APIs, business logic

PHP: Legacy support only (not used)

HTMX vs React/Vue:

Bundle Size:

HTMX: ~14KB

React: ~40KB (without React DOM)

Vue: ~33KB

Complexity:

HTMX: Low (HTML attributes)

React: High (JSX, state, hooks)

Vue: Medium (templates, reactivity)

Server-Side:

HTMX: Yes (requires SSR)

React: Optional (SSR with Next.js)

Vue: Optional (SSR with Nuxt.js)

Why HTMX:

- Minimal JavaScript
- Server-side rendering (fast TTFB)
- Progressive enhancement

- Accessibility (works without JS)
- Reduced complexity

Hetzner vs AWS/GCP/Azure:
Cost:
Hetzner: Lowest (40-60% cheaper)
AWS: Highest
GCP: High
Azure: High

Performance:
Hetzner: High (AMD EPYC, NVMe)
AWS: High
GCP: High
Azure: Medium-High

Network:
Hetzner: 20TB included
AWS: Pay per GB (expensive)
GCP: Pay per GB
Azure: Limited free, then pay

API:
Hetzner: Full-featured, RESTful
AWS: Comprehensive
GCP: Comprehensive
Azure: Comprehensive

Why Hetzner:
- Cost-effective
- European data centers (GDPR)
- Fast network (20 Gbit/s)
- Generous traffic allowance
- Solid API
- Good reputation

Appendix B: Security Compliance

yaml

PCI DSS Compliance:

Requirements:

1. Install and maintain firewall (✓ nftables)
2. Unique passwords (✓ enforced)
3. Protect cardholder data (✓ encrypted)
4. Encrypt transmission (✓ TLS 1.3)
5. Use antivirus (✓ ClamAV)
6. Secure systems (✓ hardened)
7. Restrict access (✓ RBAC)
8. Unique IDs (✓ per user)
9. Restrict physical access (✓ Hetzner DC)
10. Track access (✓ audit logs)
11. Test security (✓ regular audits)
12. Security policy (✓ documented)

GDPR Compliance:

Requirements:

- Lawful processing (✓ consent)
- Data minimization (✓ only necessary data)
- Accuracy (✓ user can update)
- Storage limitation (✓ retention policy)
- Security (✓ encryption, access control)
- Right to access (✓ API endpoint)
- Right to erasure (✓ delete account)
- Data portability (✓ export feature)
- Breach notification (✓ <72 hours)
- DPO appointed (✓ if required)

SOC 2 Type II:

Trust Service Criteria:

- Security (✓ covered)
- Availability (✓ 99.95% SLA)
- Processing Integrity (✓ accurate)
- Confidentiality (✓ encrypted)
- Privacy (✓ GDPR compliant)

Audit: Annual 3rd-party audit

ISO/IEC 27001:

Information Security Management System (ISMS)

- Risk assessment (✓ annual)

- Security controls (✓ implemented)
- Incident management (✓ process)
- Business continuity (✓ DR plan)
- Compliance (✓ ongoing)

Appendix C: Resource Estimation


yaml

Server Sizing (Hetzner):
  Control Node:
    Small (< 100 customers):
      Instance: CPX31
      vCPU: 4
      RAM: 8 GB
      Disk: 160 GB
      Cost: €16.50/month

    Medium (100-500 customers):
      Instance: CPX41
      vCPU: 8
      RAM: 16 GB
      Disk: 240 GB
      Cost: €31.50/month

    Large (500-2000 customers):
      Instance: CPX51
      vCPU: 16
      RAM: 32 GB
      Disk: 360 GB
      Cost: €61.50/month

  Application Server (per 50 hosting accounts):
    Instance: CPX31
    vCPU: 4
    RAM: 8 GB
    Disk: 160 GB
    Cost: €16.50/month

  Database Server (per 500 databases):
    Instance: CPX31
    vCPU: 4
    RAM: 8 GB
    Disk: 240 GB (SSD)
    Cost: €16.50/month + storage

  Email Server (per 1000 accounts):
    Instance: CPX21
    vCPU: 3
    RAM: 4 GB

Disk: 160 GB
Cost: €8.50/month

DNS Server (redundant pair):
Instance: CX22 (2x)
vCPU: 2 (each)
RAM: 4 GB (each)
Disk: 40 GB (each)
Cost: €5.83/month x 2 = €11.66/month

Cost Example (500 Customers):
Control Node: €31.50
Application Servers (10): €165.00
Database Servers (1): €16.50
Email Server (1): €8.50
DNS Servers (2): €11.66
Backup Storage (5TB): €25.90
Load Balancer: €5.39
Floating IPs (5): €5.00
Total: ~€270/month

Revenue Potential:
500 customers @ €10/month avg = €5,000/month
Gross Profit: €5,000 - €270 = €4,730/month
Profit Margin: 94.6%

Appendix D: Glossary

yaml

Terms:

- API: Application Programming Interface
- CDN: Content Delivery Network
- CI/CD: Continuous Integration/Continuous Deployment
- CLI: Command Line Interface
- CNAME: Canonical Name (DNS record)
- CRUD: Create, Read, Update, Delete
- CSP: Content Security Policy
- CSRF: Cross-Site Request Forgery
- DDoS: Distributed Denial of Service
- DKIM: DomainKeys Identified Mail
- DMARC: Domain-based Message Authentication, Reporting & Conformance
- DNS: Domain Name System
- DNSSEC: DNS Security Extensions
- DR: Disaster Recovery
- EPP: Extensible Provisioning Protocol
- GDPR: General Data Protection Regulation
- GeoIP: Geographic IP location
- HSTS: HTTP Strict Transport Security
- HTMX: High power tools for HTML
- HTTPS: HTTP Secure
- IMAP: Internet Message Access Protocol
- IDS: Intrusion Detection System
- IPS: Intrusion Prevention System
- JWT: JSON Web Token
- LMD: Linux Malware Detect
- MFA: Multi-Factor Authentication
- MRR: Monthly Recurring Revenue
- MX: Mail Exchanger (DNS record)
- NVMe: Non-Volatile Memory Express
- OPcache: PHP opcode cache
- OWASP: Open Web Application Security Project
- PCI DSS: Payment Card Industry Data Security Standard
- PGP: Pretty Good Privacy
- PHP-FPM: PHP FastCGI Process Manager
- PII: Personally Identifiable Information
- POP3: Post Office Protocol version 3
- RBAC: Role-Based Access Control
- REST: Representational State Transfer
- RTO: Recovery Time Objective
- RPO: Recovery Point Objective

SAML: Security Assertion Markup Language
SFTP: SSH File Transfer Protocol
SIEM: Security Information and Event Management
SLA: Service Level Agreement
SMTP: Simple Mail Transfer Protocol
SOC: Service Organization Control
SPF: Sender Policy Framework
SQL: Structured Query Language
SSH: Secure Shell
SSL: Secure Sockets Layer
SSO: Single Sign-On
TLS: Transport Layer Security
TOTP: Time-based One-Time Password
TTL: Time To Live
UAT: User Acceptance Testing
VPS: Virtual Private Server
WAF: Web Application Firewall
WHOIS: Domain registration information
XSS: Cross-Site Scripting

Document Metadata


yaml

Document Information:
Version: 2.0 (Unified Edition)
Date Created: November 4, 2025
Last Updated: November 4, 2025
Author: Platform Development Team
Status: Production Ready Specification
Classification: Internal/Confidential

Review History:
- v1.0: Initial individual PRDs (Enhance, CloudPanel, WHMCS)
- v2.0: Unified PRD with RUST/Go/HTMX stack

Approval:
Technical Lead: [Signature Required]
Product Manager: [Signature Required]
CTO: [Signature Required]
CEO: [Signature Required]

Next Review: Quarterly (or as needed)

End of Product Requirements Document

This comprehensive PRD consolidates features from Enhance Panel, CloudPanel, and WHMCS billing system into a unified, modern hosting platform optimized for Hetzner Cloud infrastructure with a cutting-edge tech stack (RUST, Go, HTMX) and extensive automation capabilities (n8n, Ansible, Bash, Python).

The platform is designed for security, performance, scalability, and ease of use - providing hosting providers with a complete solution for managing customers, services, billing, and infrastructure.