CasperVPN DevOps Overview

📋 Table of Contents

Architecture

CasperVPN uses a modern microservices architecture deployed via Docker containers, orchestrated by Docker Compose. The system is designed for high availability, scalability, and ease of deployment.

High-Level Architecture

```
┌─────────────┐
│   Internet  │
└──────┬──────┘
       │
       │
┌──────▼──────────────────────┐
│    Nginx Reverse Proxy      │
│  (Load Balancer & SSL)      │
└──────┬──────────────────────┘
       │
       │
       ├─────────────┬──────────────┬───────────────┐
       │             │              │               │
       │             │              │               │
┌──────▼──────┐ ┌───▼───────┐ ┌────▼────────┐ ┌────▼─────────┐
│  .NET API   │ │  React    │ │ PHP Laravel│ │ Rust Agent   │
│  Backend    │ │  Admin    │ │   Admin     │ │  Monitor     │
└──────┬──────┘ └───┬───────┘ └────┬────────┘ └──────────────┘
       │            │              │
       └────────────┼──────────────┘
                    │
                    │
        ┌───────────┴───────────┐
        │                       │
  ┌─────▼──────┐          ┌─────▼──────┐
  │ PostgreSQL │          │   Redis    │
  │  Database  │          │   Cache    │
  └────────────┘          └────────────┘
```

Infrastructure Components

1. Application Services

.NET Core API Backend

- **Purpose:** Main backend API serving VPN operations
- **Technology:** .NET 8.0, ASP.NET Core

- **Port:** 8080
- **Health Check:** `/health`
- **Metrics:** `/metrics`

React Admin Panel

- **Purpose:** Modern web-based admin interface
- **Technology:** React 18, Material-UI
- **Port:** 3000
- **Serves:** Static files via Nginx

PHP Laravel Admin Panel

- **Purpose:** Legacy admin panel (full-featured)
- **Technology:** Laravel 10, PHP 8.2
- **Port:** 9000
- **Runtime:** PHP-FPM + Nginx

Rust Server Agent

- **Purpose:** Server monitoring and management
- **Technology:** Rust, Actix-web
- **Port:** 8081
- **Features:** System metrics, server health monitoring

2. Data Layer

PostgreSQL Database

- **Version:** 16 Alpine
- **Port:** 5432
- **Purpose:** Primary data storage
- **Persistence:** Volume-backed (`postgres_data`)
- **Backup:** Automated via `backup.sh` script

Redis Cache

- **Version:** 7 Alpine
- **Port:** 6379
- **Purpose:** Session storage, caching, queue management
- **Persistence:** RDB snapshots

3. Proxy & Load Balancing

Nginx Reverse Proxy

- **Version:** 1.25 Alpine
- **Ports:** 80 (HTTP), 443 (HTTPS)
- **Features:**
- SSL/TLS termination
- Load balancing
- Rate limiting
- Request routing
- Static file serving
- Security headers

4. Monitoring Stack

Prometheus

- **Purpose:** Metrics collection and storage
- **Port:** 9090
- **Retention:** 30 days
- **Scrape Interval:** 15s

Grafana

- **Purpose:** Metrics visualization and dashboards
- **Port:** 3001
- **Features:** Pre-configured dashboards, alerting

Alertmanager

- **Purpose:** Alert management and routing
- **Port:** 9093
- **Integrations:** Email, Slack (configurable)

Exporters

- **Node Exporter:** System metrics (CPU, Memory, Disk)
- **PostgreSQL Exporter:** Database metrics
- **Redis Exporter:** Cache metrics
- **Nginx Exporter:** Proxy metrics
- **cAdvisor:** Container metrics

Service Overview

Service Dependencies

```
nginx
  ├── api
  │   ├── postgres
  │   └── redis
  ├── admin-react
  │   └── api
  ├── admin-php
  │   ├── postgres
  │   └── redis
  └── server-agent
      └── api
```

Service Startup Order

1. **Infrastructure:** postgres, redis
2. **Backend:** api, server-agent
3. **Frontend:** admin-react, admin-php
4. **Proxy:** nginx

Networking

Networks

caspervpn-network

- **Type:** Bridge network
- **Purpose:** Main application network
- **Services:** All application services communicate here

monitoring-network

- **Type:** Bridge network
- **Purpose:** Monitoring services isolation
- **Services:** Prometheus, Grafana, exporters

Port Mapping

Service	Internal Port	External Port	Protocol
API	8080	8080	HTTP
React Admin	3000	3000	HTTP
PHP Admin	9000	9000	HTTP
Server Agent	8081	8081	HTTP
PostgreSQL	5432	5432	TCP
Redis	6379	6379	TCP
Nginx	80/443	80/443	HTTP/HTTPS
Prometheus	9090	9090	HTTP
Grafana	3000	3001	HTTP

Data Persistence

Volumes

Volume	Purpose	Service	Backup
`postgres_data`	Database files	PostgreSQL	Yes
`redis_data`	Cache persistence	Redis	Yes
`nginx_logs`	Access/error logs	Nginx	No
`prometheus_data`	Metrics storage	Prometheus	No
`grafana_data`	Dashboard configs	Grafana	Optional

Backup Strategy

- **Frequency:** Daily (configurable via cron)
- **Retention:** 7 days local, 30 days S3
- **Components:** Database, Redis, configuration
- **Method:** `scripts/backup.sh`
- **Storage:** Local + S3 (if configured)

Security

Container Security

1. **Non-root Users:** All services run as non-root users
2. **Minimal Base Images:** Alpine Linux for smaller attack surface
3. **Health Checks:** Automated health monitoring
4. **Resource Limits:** CPU and memory constraints
5. **Read-only Filesystems:** Where applicable

Network Security

1. **Private Networks:** Services isolated in bridge networks
2. **Port Exposure:** Minimal external port exposure
3. **Rate Limiting:** Nginx-based rate limiting
4. **CORS:** Configured for API endpoints

Application Security

1. **Environment Variables:** Secrets via environment variables
2. **SSL/TLS:** HTTPS support with Let’s Encrypt
3. **Security Headers:** X-Frame-Options, X-XSS-Protection, etc.
4. **Authentication:** JWT-based API authentication
5. **Password Hashing:** Bcrypt for user passwords

Secrets Management

- **Development:** `.env` file (not committed)

- **Production:** Environment variables from secure storage
- **CI/CD:** GitHub Secrets
- **Recommended:** Integrate with AWS Secrets Manager or HashiCorp Vault

Scalability

Horizontal Scaling

Each service can be scaled independently:

```
# Scale API to 3 instances
docker-compose up -d --scale api=3

# Scale Server Agent to 5 instances
docker-compose up -d --scale server-agent=5
```

Vertical Scaling

Adjust resource limits in `docker-compose.yml`:

```
services:
  api:
    deploy:
      resources:
        limits:
          cpus: '2'
          memory: 2G
        reservations:
          cpus: '1'
          memory: 1G
```

Load Balancing

Nginx automatically load balances across multiple instances using least-connection algorithm.

Database Scaling

- **Read Replicas:** Add PostgreSQL read replicas for read-heavy workloads
- **Connection Pooling:** PgBouncer for connection management
- **Caching:** Redis for frequently accessed data

Monitoring Scalability

- **Prometheus:** Federated setup for multiple clusters
- **Grafana:** Multiple Grafana instances with shared datasource
- **Logs:** Centralized logging with ELK stack (future enhancement)

CI/CD Pipeline

GitHub Actions Workflow

1. **Code Quality:** Linting, formatting checks
2. **Build:** Compile all services
3. **Test:** Run unit and integration tests
4. **Security Scan:** Trivy vulnerability scanning

5. **Docker Build:** Build and push images to GHCR
6. **Deploy:** Automated deployment to staging/production
7. **Smoke Tests:** Post-deployment verification

Deployment Environments

- **Development:** Local Docker Compose
- **Staging:** Automated deployment on `develop` branch
- **Production:** Manual approval required for `main` branch

Best Practices

1. **Infrastructure as Code:** All configuration version-controlled
2. **Immutable Infrastructure:** Containers are never updated in place
3. **Blue-Green Deployments:** Zero-downtime deployments
4. **Health Checks:** All services have health endpoints
5. **Monitoring:** Comprehensive metrics and alerting
6. **Logging:** Structured logging with correlation IDs
7. **Backup:** Automated daily backups with testing
8. **Documentation:** Keep docs updated with changes

Next Steps

1. **Review:** Deployment Guide (./DEPLOYMENT.md)
2. **Setup:** Local Development Guide (./LOCAL_DEVELOPMENT.md)
3. **Monitor:** Monitoring Guide (./MONITORING.md)
4. **Troubleshoot:** Troubleshooting Guide (./TROUBLESHOOTING.md)

Last Updated: December 5, 2025
Maintainer: DevOps Team
Version: 1.0.0