CRISP Technical Installation Guide

Table of Contents

Overview

CRISP (Cybersecurity Resilience through Information Sharing Platform) is a Django-based threat intelligence sharing platform with a React frontend. It implements STIX/TAXII standards for cybersecurity threat data sharing and includes advanced trust management capabilities.

Architecture Overview

- **Backend**: Django 4.2.10 with Django REST Framework
- **Frontend**: React 19.1.0 with Vite 6.3.5
- **Database**: PostgreSQL 15+
- **Cache**: Redis 7+
- **Task Queue**: Celery with Redis broker
- **Authentication**: JWT-based with refresh tokens
- **API Standards**: STIX 2.1, TAXII 2.1

System Requirements

Minimum Hardware Requirements

- **CPU**: 2 cores (4 cores recommended)
- **RAM**: 4GB (8GB+ recommended)
- **Storage**: 20GB free space (50GB+ recommended)
- **Network**: Stable internet connection for threat feed updates

Software Dependencies

For Docker Installation (Recommended)

- Docker 24.0+
- Docker Compose 2.0+
- Git

For Manual Installation

- Python 3.10+ (3.11 recommended)
- Node.js 18+ with npm
- PostgreSQL 15+
- Redis 7+
- Git

Operating System Support

- Linux (Ubuntu 20.04+, CentOS 8+, RHEL 8+)
- macOS 12+
- Windows 10+ with WSL2

Installation Methods

Quick Start (Docker - Recommended)

```
git clone <repository-url>
cd Capstone-Unified
docker-compose up -d
```

Development with Test Data

```
docker-compose up -d
# Includes sample organizations, users, and threat data
```

Testing Environment with Large Dataset

```
docker-compose --profile test up -d
# Runs on separate ports with extensive test data
```

Docker Installation (Recommended)

1. Clone the Repository

```
git clone <repository-url>
cd Capstone-Unified
```

2. Environment Setup

Create a `.env` file in the project root:

```
# Copy example environment file
cp .env.example .env

# Edit environment variables (optional for development)
nano .env
```

3. Start Services

```
# Development environment (default)
docker-compose up -d

# View logs
docker-compose logs -f

# Check service status
docker-compose ps
```

4. Access the Application

- **Frontend**: http://localhost:5173
- **Backend API**: http://localhost:8000
- **Admin Interface**: http://localhost:8000/admin/

5. Default Credentials

- **Admin**: `admin` / `AdminPass123!`
- **Publisher**: `publisher` / `PublisherPass123!`
- **Viewer**: `viewer` / `ViewerPass123!`
- **Demo**: `demo` / `AdminPass123!`

- **Test**: `test` / `AdminPass123!`

Testing Environment

For testing with larger datasets:

```
# Start test environment (separate ports)
docker-compose --profile test up -d

# Access testing environment
# Frontend: http://localhost:5174
# Backend: http://localhost:8001
```

Service Management

```
# Stop services
docker-compose down

# Stop and remove volumes (data loss)
docker-compose down -v

# Restart specific service
docker-compose restart backend

# View service logs
docker-compose logs -f backend
docker-compose logs -f frontend
```

Database Management in Docker

```
# Access PostgreSQL database
docker-compose exec db psql -U crisp_user -d crisp

# Reset and repopulate database
docker-compose exec backend python manage.py flush --noinput
docker-compose exec backend python manage.py migrate
docker-compose exec backend python manage.py setup_base_users
docker-compose exec backend python manage.py populate_database --no-input

# Run custom management commands
docker-compose exec backend python manage.py <command_name>
```

Manual Installation

1. System Dependencies

Ubuntu/Debian

```
sudo apt update
sudo apt install -y python3.11 python3.11-venv python3-pip postgresql postgresql-contrib redis-server nodejs npm git build-essential
```

CentOS/RHEL

```
sudo dnf install -y python3.11 python3-pip postgresql postgresql-server redis nodejs npm git gcc postgresql-devel
sudo postgresql-setup --initdb
sudo systemctl enable postgresql redis
sudo systemctl start postgresql redis
```

macOS

```
# Install Homebrew if not installed
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

# Install dependencies
brew install python@3.11 postgresql@15 redis node git
brew services start postgresql@15
brew services start redis
```

2. Database Setup

PostgreSQL Configuration

```
# Switch to postgres user
sudo -u postgres psql

# Create database and user
CREATE USER crisp_user WITH PASSWORD 'crisp_password';
CREATE DATABASE crisp_unified OWNER crisp_user;
GRANT ALL PRIVILEGES ON DATABASE crisp_unified TO crisp_user;
ALTER USER crisp_user CREATEDB;
\q

# Test connection
PGPASSWORD='crisp_password' psql -h localhost -U crisp_user -d crisp_unified -c "SELECT version();"
```

Alternative Database Reset Script

```
# Use provided reset script
chmod +x reset_db.sh
./reset_db.sh
```

3. Backend Setup

Clone and Setup Python Environment

```
git clone <repository-url>
cd Capstone-Unified

# Create virtual environment
python3.11 -m venv venv
source venv/bin/activate  # Linux/macOS
# or
venv\Scripts\activate  # Windows

# Install dependencies
pip install --upgrade pip
pip install -r requirements.txt
```

Environment Configuration

```
# Create .env file
cat > .env << EOF
DEBUG=True
SECRET_KEY=your-secret-key-change-in-production
DB_NAME=crisp_unified
DB_USER=crisp_user
DB_PASSWORD=crisp_password
DB_HOST=localhost
DB_PORT=5432
REDIS_URL=redis://localhost:6379/0
ALLOWED_HOSTS=localhost,127.0.0.1
CORS_ALLOWED_ORIGINS=http://localhost:3000,http://localhost:5173

# Email Configuration (optional)
EMAIL_HOST=smtp.gmail.com
EMAIL_PORT=587
EMAIL_USE_TLS=True
EMAIL_HOST_USER=your-email@example.com
EMAIL_HOST_PASSWORD=your-app-password

# OTX/TAXII Configuration (optional)
OTX_API_KEY=your-otx-api-key
OTX_ENABLED=True
EOF
```

Database Migration and Setup

```
# Run migrations
python manage.py migrate

# Create base users and organizations
python manage.py setup_base_users

# Optional: Populate with test data
python manage.py populate_database --no-input

# Create additional superuser (optional)
python manage.py createsuperuser

# Collect static files
python manage.py collectstatic --noinput
```

Start Backend Server

```
# Development server
python manage.py runserver 0.0.0.0:8000

# Or use Gunicorn for production-like environment
gunicorn --bind 0.0.0.0:8000 --workers 3 --timeout 120 crisp_unified.wsgi:application
```

4. Frontend Setup

Node.js and Dependencies

```
cd crisp-react

# Install dependencies
npm install

# Start development server
npm run dev

# Build for production
npm run build
```

Frontend Development

```
# Development with hot reload
npm run dev
# Access: http://localhost:5173

# Lint code
npm run lint

# Build production bundle
npm run build
```

Environment Configuration

Core Environment Variables

```
# Django Configuration
DEBUG=True                                    # Set to False in production
SECRET_KEY=your-unique-secret-key
DJANGO_SETTINGS_MODULE=crisp_unified.settings

# Database Configuration
DB_NAME=crisp_unified
DB_USER=crisp_user
DB_PASSWORD=crisp_password
DB_HOST=localhost
DB_PORT=5432
USE_SQLITE=false                              # Set to true for SQLite fallback

# Redis Configuration
REDIS_URL=redis://localhost:6379/0

# Security Configuration
ALLOWED_HOSTS=localhost,127.0.0.1,yourdomain.com
CORS_ALLOWED_ORIGINS=http://localhost:3000,http://localhost:5173
SESSION_COOKIE_AGE_SECONDS=3600

# JWT Configuration
JWT_ACCESS_TOKEN_LIFETIME_MINUTES=60
JWT_REFRESH_TOKEN_LIFETIME_DAYS=7
```

Optional Integration Settings

```
# Email Configuration
EMAIL_BACKEND=django.core.mail.backends.smtp.EmailBackend
EMAIL_HOST=smtp.gmail.com
EMAIL_PORT=587
EMAIL_USE_TLS=True
EMAIL_HOST_USER=your-email@example.com
EMAIL_HOST_PASSWORD=your-app-password
CRISP_SENDER_EMAIL=noreply@crisp-system.org

# OTX Threat Intelligence
OTX_API_KEY=your-otx-api-key-from-alienvault
OTX_ENABLED=True
OTX_FETCH_INTERVAL=3600
OTX_BATCH_SIZE=10
OTX_MAX_AGE_DAYS=1

# TAXII Server Configuration
TAXII_SERVER_TITLE=CRISP Threat Intelligence Platform
TAXII_SERVER_DESCRIPTION=Educational threat intelligence sharing platform
TAXII_CONTACT_EMAIL=admin@example.com
TAXII_FILTER_DAYS=1
```

Trust Management Settings

```
# Trust System Configuration
DEFAULT_TRUST_LEVEL=restricted
AUTO_APPROVE_TRUST_RELATIONSHIPS=False
TRUST_RELATIONSHIP_EXPIRY_DAYS=365
MAX_TRUST_GROUPS_PER_ORG=10
ENABLE_TRUST_INHERITANCE=True

# User Management
REQUIRE_EMAIL_VERIFICATION=True
PASSWORD_RESET_TIMEOUT_DAYS=1
ACCOUNT_LOCKOUT_THRESHOLD=5
ACCOUNT_LOCKOUT_DURATION_MINUTES=30
ENABLE_TWO_FACTOR_AUTH=True
REQUIRE_STRONG_PASSWORDS=True

# Security Features
ENABLE_AUDIT_LOGGING=True
AUDIT_LOG_RETENTION_DAYS=90
ENABLE_RATE_LIMITING=True
ENABLE_IP_WHITELISTING=False
MAX_UPLOAD_SIZE_MB=50
```

Database Setup

PostgreSQL Installation and Configuration

Ubuntu/Debian

```
# Install PostgreSQL
sudo apt update
sudo apt install postgresql postgresql-contrib

# Start and enable service
sudo systemctl start postgresql
sudo systemctl enable postgresql

# Create database and user
sudo -u postgres createuser --interactive crisp_user
sudo -u postgres createdb crisp_unified --owner crisp_user

# Set password
sudo -u postgres psql -c "ALTER USER crisp_user PASSWORD 'crisp_password';"
```

Advanced Database Configuration

```
# Edit PostgreSQL configuration for performance
sudo nano /etc/postgresql/15/main/postgresql.conf

# Recommended settings for CRISP:
shared_buffers = 256MB
effective_cache_size = 1GB
maintenance_work_mem = 64MB
checkpoint_completion_target = 0.9
wal_buffers = 16MB
default_statistics_target = 100
random_page_cost = 1.1

# Restart PostgreSQL
sudo systemctl restart postgresql
```

Database Permissions Script

```
# Use the provided permissions script
sudo -u postgres psql -d crisp_unified -f fix_permissions.sql

# Or run manually:
sudo -u postgres psql -d crisp_unified << EOF
GRANT ALL PRIVILEGES ON SCHEMA public TO crisp_user;
ALTER SCHEMA public OWNER TO crisp_user;
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO crisp_user;
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO crisp_user;
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT ALL ON TABLES TO crisp_user;
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT ALL ON SEQUENCES TO crisp_user;
EOF
```

Database Backup and Restore

```
# Create backup
pg_dump -h localhost -U crisp_user -d crisp_unified > crisp_backup_$(date +%Y%m%d_%H%M%S).sql

# Restore from backup
psql -h localhost -U crisp_user -d crisp_unified < crisp_backup.sql

# Automated backup script
cat > backup_crisp.sh << 'EOF'
#!/bin/bash
BACKUP_DIR="/var/backups/crisp"
DATE=$(date +%Y%m%d_%H%M%S)
mkdir -p $BACKUP_DIR
pg_dump -h localhost -U crisp_user -d crisp_unified > $BACKUP_DIR/crisp_backup_$DATE.sql
find $BACKUP_DIR -name "crisp_backup_*.sql" -mtime +7 -delete
EOF
chmod +x backup_crisp.sh
```

Frontend Setup

React Application Configuration

Development Setup

```
cd crisp-react

# Install dependencies
npm install

# Start development server with hot reload
npm run dev

# Build for development
npm run build

# Lint code
npm run lint
```

Vite Configuration

The frontend uses Vite for fast development and building. Key configuration in `vite.config.js`:

```
export default defineConfig({
  plugins: [react()],
  build: {
    outDir: 'dist',
    assetsDir: 'assets',
  },
  base: '/static/react/'  // For Django static file serving
})
```

Environment Variables for Frontend

Create `crisp-react/.env`:

```
VITE_API_URL=http://localhost:8000
VITE_ENABLE_DEBUG=true
VITE_APP_NAME=CRISP Platform
VITE_APP_VERSION=1.0.0
```

Static File Serving

Development

In development, Vite serves files directly on port 5173.

Production

For production, the frontend is built and served as Django static files:

```
cd crisp-react
npm run build

# Files are built to dist/ and served from Django's static files
cd ..
python manage.py collectstatic --noinput
```

Production Deployment

Production Environment Setup

Environment Variables for Production

```
DEBUG=False
SECRET_KEY=your-very-secure-secret-key-256-bits-long
ALLOWED_HOSTS=yourdomain.com,www.yourdomain.com
CORS_ALLOWED_ORIGINS=https://yourdomain.com,https://www.yourdomain.com

# Database with connection pooling
DB_HOST=your-db-server
DB_PORT=5432
DB_NAME=crisp_production
DB_USER=crisp_prod_user
DB_PASSWORD=your-secure-db-password

# Redis for production
REDIS_URL=redis://your-redis-server:6379/0

# Security settings
SECURE_SSL_REDIRECT=True
SESSION_COOKIE_SECURE=True
CSRF_COOKIE_SECURE=True
SECURE_BROWSER_XSS_FILTER=True
SECURE_CONTENT_TYPE_NOSNIFF=True

# Email for production
EMAIL_HOST=your-smtp-server
EMAIL_PORT=587
EMAIL_USE_TLS=True
EMAIL_HOST_USER=noreply@yourdomain.com
EMAIL_HOST_PASSWORD=your-email-password
```

Docker Production Deployment

```
# Use production target in Dockerfile
docker build --target production -t crisp:latest .

# Run with production environment
docker run -d \
  --name crisp-production \
  -p 80:8000 \
  -e DEBUG=False \
  -e SECRET_KEY=your-secure-key \
  -e DB_HOST=your-db-host \
  --restart unless-stopped \
  crisp:latest

# Or use docker-compose with production overrides
docker-compose -f docker-compose.yml -f docker-compose.prod.yml up -d
```

Nginx Reverse Proxy

```
server {
    listen 80;
    server_name yourdomain.com;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl;
    server_name yourdomain.com;

    ssl_certificate /etc/ssl/certs/yourdomain.pem;
    ssl_certificate_key /etc/ssl/private/yourdomain.key;

    client_max_body_size 100M;

    location / {
        proxy_pass http://localhost:8000;
        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;
    }

    location /static/ {
        alias /path/to/crisp/staticfiles/;
        expires 1y;
        add_header Cache-Control "public, immutable";
    }

    location /media/ {
        alias /path/to/crisp/media/;
        expires 1d;
    }
}
```

Systemd Service (Alternative to Docker)

```
# Create systemd service file
sudo tee /etc/systemd/system/crisp.service << EOF
[Unit]
Description=CRISP Threat Intelligence Platform
After=network.target postgresql.service redis.service

[Service]
Type=notify
User=crisp
Group=crisp
WorkingDirectory=/opt/crisp/Capstone-Unified
Environment=PATH=/opt/crisp/venv/bin
ExecStart=/opt/crisp/venv/bin/gunicorn --bind 127.0.0.1:8000 --workers 3 --timeout 120 crisp_unified.wsgi:application
ExecReload=/bin/kill -s HUP $MAINPID
Restart=always
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF

# Enable and start service
sudo systemctl enable crisp
sudo systemctl start crisp
```

Testing and Verification

System Health Checks

```
# Run built-in health check
python manage.py system_health_check

# Check database connectivity
python manage.py dbshell

# Verify Redis connection
redis-cli ping

# Test API endpoints
curl -X GET http://localhost:8000/api/v1/health/
curl -X POST http://localhost:8000/api/v1/auth/login/ \
  -H "Content-Type: application/json" \
  -d '{"username": "admin", "password": "AdminPass123!"}'
```

Running Tests

```
# Run all tests
python manage.py test

# Run specific test modules
python manage.py test core.tests.test_integration
python manage.py test core.tests.test_auth_service

# Run tests with coverage
pip install coverage
coverage run --source='.' manage.py test
coverage report
coverage html
```

Frontend Testing

```
cd crisp-react

# Lint JavaScript/React code
npm run lint

# Build production bundle
npm run build

# Check for vulnerabilities
npm audit
npm audit fix
```

Load Testing

```
# Install locust for load testing
pip install locust

# Create basic load test
cat > locustfile.py << 'EOF'
from locust import HttpUser, task, between

class CrispUser(HttpUser):
    wait_time = between(1, 3)

    def on_start(self):
        # Login
        response = self.client.post("/api/v1/auth/login/", {
            "username": "admin",
            "password": "AdminPass123!"
        })
        if response.status_code == 200:
            self.token = response.json()['access']
            self.client.headers.update({
                'Authorization': f'Bearer {self.token}'
            })

    @task
    def dashboard(self):
        self.client.get("/api/v1/dashboard/")

    @task
    def threat_feeds(self):
        self.client.get("/api/v1/threat-feeds/")
EOF

# Run load test
locust -f locustfile.py --host=http://localhost:8000
```

Troubleshooting

Common Issues and Solutions

Database Connection Issues

```
# Check PostgreSQL status
sudo systemctl status postgresql

# Check if database exists
sudo -u postgres psql -l | grep crisp

# Test connection with explicit host
PGPASSWORD='crisp_password' psql -h 127.0.0.1 -U crisp_user -d crisp_unified -c "SELECT 1;"

# Reset database permissions
sudo -u postgres psql -d crisp_unified -f fix_permissions.sql
```

Docker Issues

```
# Check Docker service status
sudo systemctl status docker

# Check container logs
docker-compose logs -f backend
docker-compose logs -f db

# Restart services in order
docker-compose restart db
docker-compose restart redis
docker-compose restart backend

# Clean up Docker resources
docker system prune -a
docker volume prune
```

Permission Issues

```
# Fix file permissions
sudo chown -R $USER:$USER /path/to/crisp
chmod +x manage.py
chmod +x reset_db.sh

# Fix Docker permissions (if needed)
sudo usermod -aG docker $USER
newgrp docker
```

Port Conflicts

```
# Check what's using port 8000
sudo netstat -tlnp | grep 8000
sudo lsof -i :8000

# Kill process using port
sudo kill -9 $(sudo lsof -t -i:8000)

# Use different ports in docker-compose.yml
# Change "8000:8000" to "8001:8000"
```

Frontend Build Issues

```
cd crisp-react

# Clear npm cache
npm cache clean --force

# Remove node_modules and reinstall
rm -rf node_modules package-lock.json
npm install

# Check for version conflicts
npm ls

# Build with verbose output
npm run build --verbose
```

Memory Issues

```
# Check memory usage
free -h
docker stats

# Increase swap space (Linux)
sudo fallocate -l 2G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile

# Add to /etc/fstab for persistence
echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab
```

SSL/TLS Issues

```
# Test SSL certificate
openssl s_client -connect yourdomain.com:443 -servername yourdomain.com

# Check certificate expiration
openssl x509 -in certificate.pem -text -noout | grep -A 2 "Validity"

# Generate self-signed certificate for development
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -nodes
```

Performance Optimization

Database Optimization

```
-- Add indexes for common queries
CREATE INDEX idx_threat_feed_created_at ON core_threatfeed(created_at);
CREATE INDEX idx_user_organization ON core_customuser(organization_id);
CREATE INDEX idx_trust_relationship_org ON core_trustrelationship(trusting_org_id, trusted_org_id);

-- Analyze database performance
ANALYZE;

-- Check slow queries
SELECT query, mean_time, calls FROM pg_stat_statements ORDER BY mean_time DESC LIMIT 10;
```

Django Performance

```
# Add to settings.py for production
DATABASES['default']['CONN_MAX_AGE'] = 60
DATABASES['default']['OPTIONS'] = {
    'MAX_CONNS': 20,
    'MIN_CONNS': 5,
}

# Enable query optimization
CRISP_SETTINGS['PERFORMANCE']['ENABLE_QUERY_OPTIMIZATION'] = True
CRISP_SETTINGS['PERFORMANCE']['ENABLE_CACHING'] = True
```

Management Commands

Available Management Commands

System Initialization

```
# Initialize system with default data
python manage.py initialize_system

# Create base users (admin, publisher, viewer)
python manage.py setup_base_users

# Populate database with test data
python manage.py populate_database --no-input

# System health check
python manage.py system_health_check
```

User Management

```
# Create superuser
python manage.py createsuperuser

# Change user password
python manage.py changepassword username

# Create custom user
python manage.py shell
>>> from django.contrib.auth import get_user_model
>>> User = get_user_model()
>>> user = User.objects.create_user('testuser', 'test@example.com', 'password')
```

Trust Management

```
# Initialize trust levels
python manage.py init_trust_levels

# Run trust relationship cleanup
python manage.py cleanup_expired_relationships

# Generate trust metrics
python manage.py generate_trust_metrics
```

Threat Intelligence

```
# Fetch OTX threat data
python manage.py fetch_otx_data

# Process TAXII feeds
python manage.py taxii_operations --fetch-all

# Test MITRE mapping
python manage.py test_mitre_mapping

# Aggregate TTPs
python manage.py ttp_aggregation
```

Testing and Development

```
# Run orchestrated tests
python manage.py run_orchestrated_tests

# Clean up test data
python manage.py cleanup_test_data

# Test email functionality
python manage.py test_gmail_email

# Test TAXII connectivity
python manage.py test_taxii
```

Data Management

```
# Backup database
python manage.py dumpdata > backup.json

# Load data from fixture
python manage.py loaddata backup.json

# Clear all data
python manage.py flush

# Migrate database
python manage.py migrate

# Create migrations
python manage.py makemigrations
```

Custom Management Commands

Creating Custom Commands

```
# core/management/commands/custom_command.py
from django.core.management.base import BaseCommand

class Command(BaseCommand):
    help = 'Custom management command'

    def add_arguments(self, parser):
        parser.add_argument('--option', type=str, help='Custom option')

    def handle(self, *args, **options):
        self.stdout.write(
            self.style.SUCCESS('Custom command executed')
        )
```

Running Custom Commands

```
# Run custom command
python manage.py custom_command --option value

# Get help for command
python manage.py help custom_command

# List all available commands
python manage.py help
```

Automated Scripts

Backup Script

```
#!/bin/bash
# backup_crisp.sh
DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_DIR="/var/backups/crisp"

mkdir -p $BACKUP_DIR

# Database backup
pg_dump -h localhost -U crisp_user -d crisp_unified > $BACKUP_DIR/db_backup_$DATE.sql

# Media files backup
tar -czf $BACKUP_DIR/media_backup_$DATE.tar.gz /path/to/crisp/media/

# Configuration backup
cp /path/to/crisp/.env $BACKUP_DIR/env_backup_$DATE

# Clean old backups (keep 30 days)
find $BACKUP_DIR -name "*backup*" -mtime +30 -delete

echo "Backup completed: $BACKUP_DIR"
```

Health Check Script

```
#!/bin/bash
# health_check.sh
echo "CRISP Health Check $(date)"

# Check services
services=("postgresql" "redis" "nginx")
for service in "${services[@]}"; do
    if systemctl is-active --quiet $service; then
        echo "□ $service is running"
    else
        echo "□ $service is not running"
    fi
done

# Check ports
ports=("5432" "6379" "8000")
for port in "${ports[@]}"; do
    if netstat -tln | grep -q ":$port "; then
        echo "□ Port $port is listening"
    else
        echo "□ Port $port is not listening"
    fi
done

# Check disk space
df -h | grep -E "(/$|/var|/opt)"

# Check memory usage
free -h

# Test API endpoint
if curl -s http://localhost:8000/api/v1/health/ | grep -q "healthy"; then
    echo "□ API is responding"
else
    echo "□ API is not responding"
fi
```

Network Security

```
# Configure firewall (UFW example)
sudo ufw allow 22/tcp    # SSH
sudo ufw allow 80/tcp    # HTTP
sudo ufw allow 443/tcp   # HTTPS
sudo ufw enable

# Block direct database access from external
sudo ufw deny 5432/tcp

# Configure fail2ban for SSH protection
sudo apt install fail2ban
sudo systemctl enable fail2ban
sudo systemctl start fail2ban
```

This comprehensive installation guide provides all the necessary information to deploy CRISP in various environments, from development to production. Choose the installation method that best fits your requirements and follow the detailed steps for a successful deployment.