Cyber Trust Sensor Dashboard - Complete Deployment Guide

Table of Contents

---

1. Prerequisites

System Requirements

- **OS**: Ubuntu 22.04 LTS or 24.04 LTS (64-bit)
- **RAM**: Minimum 4GB (8GB recommended)
- **CPU**: 2 cores minimum (4 cores recommended)
- **Storage**: 20GB minimum free space
- **Network**: Static IP address with ports 80, 443 accessible
- **Access**: Root or sudo privileges

Required Ports

The following ports need to be accessible:

- **22**: SSH (for remote management)
- **80**: HTTP (web traffic)
- **443**: HTTPS (secure web traffic)
- **3000**: Development server (optional, for development only)
- **9090**: Prometheus metrics (optional, for monitoring)

- **3001**: API service (internal use)

---

2. Server Preparation

2.1 Initial Server Setup

```bash
# Update system packages
sudo apt update && sudo apt upgrade -y

# Install essential tools
sudo apt install -y \
  curl \
  wget \
  git \
  vim \
  nano \
  htop \
  net-tools \
  build-essential \
  software-properties-common \
  apt-transport-https \
  ca-certificates \
  gnupg \
  lsb-release

# Set timezone (replace with your timezone)
sudo timedatectl set-timezone UTC

# Configure hostname (replace with your domain/hostname)
sudo hostnamectl set-hostname cyber-trust-dashboard
```

2.2 Create Application User (Optional but Recommended)

```bash
# Create a dedicated user for the application
sudo adduser --system --group --home /opt/cyber-trust cyber-trust

# Add your user to the cyber-trust group for management
sudo usermod -aG cyber-trust $USER
```

2.3 Configure Firewall

```bash
```

```bash
# Install UFW if not already installed
sudo apt install -y ufw

# Configure firewall rules
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow 22/tcp comment 'SSH'
sudo ufw allow 80/tcp comment 'HTTP'
sudo ufw allow 443/tcp comment 'HTTPS'

# Enable firewall
sudo ufw --force enable

# Check status
sudo ufw status verbose
```

2.4 Configure Swap (Important for Low-Memory Servers)

bash

```bash
# Check if swap exists
sudo swapon --show

# Create swap file if it doesn't exist
sudo fallocate -l 4G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile

# Make swap permanent
echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab

# Configure swappiness for better performance
echo 'vm.swappiness=10' | sudo tee -a /etc/sysctl.conf
sudo sysctl -p
```

---

3. Docker Installation

3.1 Remove Old Docker Versions

bash

```bash
# Remove any existing Docker installations
sudo apt remove -y docker docker-engine docker.io containerd runc 2>/dev/null || true
sudo apt autoremove -y
```

3.2 Install Docker

bash

```bash
# Add Docker's official GPG key
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

# Set up the repository
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Update package index
sudo apt update

# Install Docker Engine and Docker Compose
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

# Verify installation
docker --version
docker compose version
```

3.3 Configure Docker

bash

```
# Add current user to docker group (allows running docker without sudo)
sudo usermod -aG docker $USER

# Configure Docker daemon
sudo tee /etc/docker/daemon.json <<EOF
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3"
  },
  "storage-driver": "overlay2",
  "metrics-addr": "0.0.0.0:9323",
  "experimental": false
}
EOF

# Restart Docker
sudo systemctl restart docker
sudo systemctl enable docker

# Verify Docker is running
sudo systemctl status docker

# Test Docker installation
docker run hello-world
```

Important: Log out and log back in for group changes to take effect, or run:

```bash
newgrp docker
```

---

4. Application Setup

4.1 Clone the Repository

```bash
```

```
# Navigate to home directory
cd ~

# Clone the repository
git clone https://github.com/your-username/my-working-prototype-dashbaord.git
cd my-working-prototype-dashbaord

# Or if you have the code locally, upload it:
# scp -r local-path/* user@server:~/my-working-prototype-dashbaord/
```

4.2 Project Structure Setup

```bash
# Create necessary directories
mkdir -p nginx/{conf.d,ssl,logs}
mkdir -p build
mkdir -p data/{mongodb,redis}
mkdir -p logs/{app,nginx}
mkdir -p backups

# Set permissions
chmod -R 755 nginx
chmod -R 755 build
chmod -R 755 data
chmod -R 755 logs
```

4.3 Environment Configuration

Create a `.env` file for environment variables:

```bash
```

```
# Create .env file
cat > .env <<'EOF'
# Application Settings
NODE_ENV=production
APP_NAME=cyber-trust-dashboard
APP_PORT=3000
APP_URL=http://your-domain.com

# API Settings
API_PORT=3001
API_URL=http://localhost:3001

# Database Settings (if using MongoDB)
MONGODB_URI=mongodb://mongodb:27017/cyber-trust
MONGODB_DB=cyber-trust

# Redis Settings (if using Redis)
REDIS_HOST=redis
REDIS_PORT=6379

# JWT Settings
JWT_SECRET=your-super-secret-jwt-key-change-this
JWT_EXPIRY=7d

# Logging
LOG_LEVEL=info
LOG_DIR=/var/log/app

# Security
CORS_ORIGIN=http://your-domain.com
SESSION_SECRET=your-session-secret-change-this

# Feature Flags
ENABLE_MONITORING=true
ENABLE_ANALYTICS=true
EOF

# Secure the .env file
chmod 600 .env
```

---

5. Building the Application

5.1 Install Node.js

```bash
```

```bash
# Install Node.js 20.x (LTS)
curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -
sudo apt install -y nodejs

# Verify installation
node --version
npm --version
```

5.2 Install Dependencies and Build

```bash
# Clean any previous builds
rm -rf node_modules package-lock.json build

# Install dependencies
npm install

# If you encounter dependency conflicts, try:
# npm install --legacy-peer-deps

# Build the production version
CI=false npm run build

# Verify build was successful
ls -la build/
```

5.3 Optimize Build (Optional)

```bash
# Install build optimization tools
npm install -g npm-check-updates

# Check for updates (don't apply automatically)
ncu

# Analyze bundle size
npm run build -- --stats
```

---

6. Docker Configuration

6.1 Create Docker Compose Configuration

```bash
```

```
# Create docker-compose.yml
cat > docker-compose.yml <<'EOF'
version: '3.8'

services:
  nginx:
    image: nginx:alpine
    container_name: cyber-trust-nginx
    restart: unless-stopped
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx/conf.d:/etc/nginx/conf.d:ro
      - ./nginx/ssl:/etc/nginx/ssl:ro
      - ./nginx/logs:/var/log/nginx
      - ./build:/usr/share/nginx/html:ro
    networks:
      - cyber-trust-network
    depends_on:
      - api
    healthcheck:
      test: ["CMD", "wget", "-qO-", "http://localhost/health"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s

  api:
    build:
      context: ./api
      dockerfile: Dockerfile
    container_name: cyber-trust-api
    restart: unless-stopped
    environment:
      - NODE_ENV=production
      - PORT=3001
    env_file:
      - .env
    volumes:
      - ./api:/app
      - /app/node_modules
      - ./logs/app:/var/log/app
    networks:
      - cyber-trust-network
    depends_on:
      - mongodb
      - redis
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:3001/health"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s

  mongodb:
    image: mongo:7
    container_name: cyber-trust-mongodb
    restart: unless-stopped
    environment:
      - MONGO_INITDB_ROOT_USERNAME=admin
      - MONGO_INITDB_ROOT_PASSWORD=secure-password-change-this
      - MONGO_INITDB_DATABASE=cyber-trust
    volumes:
      - ./data/mongodb:/data/db
      - ./backups/mongodb:/backup
    networks:
      - cyber-trust-network
    command: mongod --quiet --logpath /dev/null

  redis:
    image: redis:alpine
    container_name: cyber-trust-redis
    restart: unless-stopped
    command: redis-server --appendonly yes
    volumes:
      - ./data/redis:/data
    networks:
      - cyber-trust-network

networks:
  cyber-trust-network:
    driver: bridge
    ipam:
      config:
        - subnet: 172.20.0.0/16

volumes:
  nginx-cache:
    driver: local
EOF
```

6.2 Create Nginx Configuration

bash

bash

```
# Create nginx configuration
cat > nginx/conf.d/default.conf <<'EOF'
# Upstream configuration for API
upstream api_backend {
    server api:3001;
}

# Rate limiting zones
limit_req_zone $binary_remote_addr zone=api_limit:10m rate=10r/s;
limit_req_zone $binary_remote_addr zone=general_limit:10m rate=50r/s;

# Main server block
server {
    listen 80;
    server_name _;

    # Security headers
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header X-XSS-Protection "1; mode=block" always;
    add_header Referrer-Policy "strict-origin-when-cross-origin" always;
    add_header Content-Security-Policy "default-src 'self' http: https: data: blob: 'unsafe-inline'" always;

    # Gzip compression
    gzip on;
    gzip_vary on;
    gzip_min_length 1024;
    gzip_types text/plain text/css text/xml text/javascript application/javascript application/xml+rss application/json;

    # Main application
    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
        try_files $uri $uri/ /index.html;

        # Apply rate limiting
        limit_req zone=general_limit burst=20 nodelay;
    }

    # API proxy
    location /api/ {
        # Remove /api prefix when proxying
        rewrite ^/api/(.*) /$1 break;

        proxy_pass http://api_backend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        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_cache_bypass $http_upgrade;

        # Timeouts
        proxy_connect_timeout 60s;
        proxy_send_timeout 60s;
        proxy_read_timeout 60s;

        # Apply rate limiting
        limit_req zone=api_limit burst=10 nodelay;
    }

    # Health check endpoint
    location /health {
        access_log off;
        add_header Content-Type application/json;
        return 200 '{"status":"UP","timestamp":"$date_gmt"}';
    }

    # Static assets caching
    location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg|woff|woff2|ttf|eot)$ {
        root /usr/share/nginx/html;
        expires 1y;
        add_header Cache-Control "public, immutable";
        access_log off;
    }

    # Deny access to hidden files
    location ~ /\. {
        deny all;
        access_log off;
        log_not_found off;
    }
}

# HTTPS server block (uncomment when SSL is configured)
# server {
#     listen 443 ssl http2;
#     server_name your-domain.com;
#
#     ssl_certificate /etc/nginx/ssl/cert.pem;
#     ssl_certificate_key /etc/nginx/ssl/key.pem;
#     ssl_protocols TLSv1.2 TLSv1.3;
#     ssl_ciphers HIGH:!aNULL:!MD5;
#
#     # Include all location blocks from above
# }
EOF
```

6.3 Create API Dockerfile (if you have a backend API)

```bash
# Create API directory and Dockerfile
mkdir -p api

cat > api/Dockerfile <<'EOF'
FROM node:20-alpine

WORKDIR /app

# Copy package files
COPY package*.json ./

# Install dependencies
RUN npm ci --only=production

# Copy application code
COPY . .

# Create non-root user
RUN addgroup -g 1001 -S nodejs && \
    adduser -S nodejs -u 1001

# Change ownership
RUN chown -R nodejs:nodejs /app

USER nodejs

EXPOSE 3001

CMD ["node", "server.js"]
EOF
```

6.4 Create a Simple API Server (placeholder if no backend yet)

```bash
```

```
cat > api/server.js <<'EOF'
const express = require('express');
const app = express();
const port = process.env.PORT || 3001;

// Middleware
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// CORS (if needed)
app.use((req, res, next) => {
  res.header('Access-Control-Allow-Origin', '*');
  res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-Type, Accept');
  next();
});

// Routes
app.get('/', (req, res) => {
  res.json({ message: 'Cyber Trust API', version: '1.0.0' });
});

app.get('/health', (req, res) => {
  res.json({
    status: 'UP',
    timestamp: new Date().toISOString(),
    uptime: process.uptime()
  });
});

// Sample API endpoints
app.get('/api/dashboard', (req, res) => {
  res.json({
    trustScore: 85,
    threats: 12,
    risks: 5,
    compliance: 92
  });
});

// Error handling
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ error: 'Something went wrong!' });
});

// Start server
```

```
app.listen(port, '0.0.0.0', () => {
    console.log(`API server running on port ${port}`);
});
EOF

# Create package.json for API
cat > api/package.json <<'EOF'
{
  "name": "cyber-trust-api",
  "version": "1.0.0",
  "description": "Cyber Trust Dashboard API",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js"
  },
  "dependencies": {
    "express": "^4.18.2",
    "cors": "^2.8.5",
    "dotenv": "^16.3.1",
    "mongoose": "^8.0.0",
    "redis": "^4.6.0"
  },
  "devDependencies": {
    "nodemon": "^3.0.1"
  }
}
EOF
```

---

7. Deployment

7.1 Pre-deployment Checks

```
bash
```

```
# Verify all files are in place
ls -la
ls -la nginx/conf.d/
ls -la build/

# Check Docker is running
docker ps

# Validate docker-compose configuration
docker compose config

# Check for port conflicts
sudo netstat -tlnp | grep -E ':80|:443|:3001'
```

7.2 Deploy the Application

```bash
# Pull latest images
docker compose pull

# Build custom images (if any)
docker compose build

# Start all services
docker compose up -d

# Check if containers are running
docker compose ps

# View logs
docker compose logs -f

# To see logs for specific service
docker compose logs -f nginx
docker compose logs -f api
```

7.3 Verify Deployment

```bash
```

```bash
# Check if services are healthy
docker compose ps

# Test nginx health endpoint
curl http://localhost/health

# Test API (if running)
curl http://localhost/api/health

# Check from outside (replace with your server IP)
curl http://YOUR_SERVER_IP
```

---

8. Post-Deployment Configuration

8.1 SSL/TLS Configuration with Let's Encrypt

bash

```bash
# Install Certbot
sudo apt install -y certbot python3-certbot-nginx

# Stop nginx temporarily
docker compose stop nginx

# Get SSL certificate (replace with your domain)
sudo certbot certonly --standalone -d your-domain.com -d www.your-domain.com

# Update nginx configuration to use SSL
# Edit nginx/conf.d/default.conf and uncomment the HTTPS server block

# Restart nginx
docker compose start nginx

# Set up auto-renewal
sudo crontab -e
# Add this line:
# 0 0 * * * certbot renew --pre-hook "docker compose stop nginx" --post-hook "docker compose start nginx"
```

8.2 Configure Monitoring

bash

```
# Create docker-compose.monitoring.yml
cat > docker-compose.monitoring.yml <<'EOF'
version: '3.8'

services:
  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    volumes:
      - ./monitoring/prometheus:/etc/prometheus
      - prometheus_data:/prometheus
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
    ports:
      - "9090:9090"
    networks:
      - cyber-trust-network

  grafana:
    image: grafana/grafana:latest
    container_name: grafana
    volumes:
      - grafana_data:/var/lib/grafana
      - ./monitoring/grafana:/etc/grafana/provisioning
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin
      - GF_INSTALL_PLUGINS=
    ports:
      - "3000:3000"
    networks:
      - cyber-trust-network

volumes:
  prometheus_data:
  grafana_data:

networks:
  cyber-trust-network:
    external: true
EOF

# Start monitoring stack
docker compose -f docker-compose.monitoring.yml up -d
```

8.3 Set Up Logging

```bash
# Create log rotation configuration
sudo tee /etc/logrotate.d/cyber-trust <<EOF
/home/ubuntu/my-working-prototype-dashbaord/logs/*/*.log {
    daily
    rotate 14
    compress
    delaycompress
    missingok
    notifempty
    create 0644 root root
    sharedscripts
    postrotate
        docker compose exec nginx nginx -s reload > /dev/null 2>&1 || true
    endscript
}
EOF
```

---

9. Troubleshooting

9.1 Common Issues and Solutions

Port Already in Use

```bash
# Find what's using port 80
sudo lsof -i :80

# Stop conflicting service
sudo systemctl stop apache2  # or nginx if installed on host

# Or change port in docker-compose.yml
# Change "80:80" to "8080:80"
```

Container Won't Start

```bash
```

```bash
# Check logs
docker compose logs nginx
docker compose logs api

# Check container status
docker ps -a

# Rebuild containers
docker compose down
docker compose build --no-cache
docker compose up -d
```

Permission Issues

bash

```bash
# Fix ownership
sudo chown -R $USER:$USER .

# Fix permissions
chmod -R 755 nginx
chmod -R 755 build
chmod 600 .env
```

Out of Memory

bash

```bash
# Check memory usage
docker stats

# Limit container memory in docker-compose.yml
# Add under service:
#   deploy:
#     resources:
#       limits:
#         memory: 512M
```

9.2 Debug Commands

bash

```
# Enter container shell
docker compose exec nginx sh
docker compose exec api sh

# Check network connectivity
docker compose exec nginx ping api
docker compose exec api ping mongodb

# Inspect container
docker inspect cyber-trust-nginx

# Check Docker logs
sudo journalctl -u docker -f

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

---

10. Maintenance

10.1 Regular Updates

```bash
# Update system packages
sudo apt update && sudo apt upgrade -y

# Update Docker images
docker compose pull
docker compose up -d

# Update application
git pull
npm install
npm run build
docker compose restart nginx
```

10.2 Backup Procedures

```bash
```

```
# Create backup script
cat > backup.sh <<'EOF'
#!/bin/bash
BACKUP_DIR="/home/ubuntu/backups/$(date +%Y%m%d_%H%M%S)"
mkdir -p "$BACKUP_DIR"

# Backup application files
tar -czf "$BACKUP_DIR/app.tar.gz" /home/ubuntu/my-working-prototype-dashbaord

# Backup Docker volumes
docker run --rm -v cyber-trust-mongodb:/data -v "$BACKUP_DIR":/backup alpine tar -czf /backup/mongodb.tar.gz /da

# Backup environment files
cp .env "$BACKUP_DIR/"

echo "Backup completed: $BACKUP_DIR"
EOF

chmod +x backup.sh

# Run backup
./backup.sh

# Schedule daily backups
crontab -e
# Add: 0 2 * * * /home/ubuntu/backup.sh
```

10.3 Monitoring Health

```bash
```

```
# Create health check script
cat > health-check.sh <<'EOF'
#!/bin/bash

# Check if containers are running
if ! docker compose ps | grep -q "Up"; then
    echo "WARNING: Some containers are down"
    docker compose up -d
fi

# Check web service
if ! curl -f http://localhost/health > /dev/null 2>&1; then
    echo "WARNING: Web service not responding"
    docker compose restart nginx
fi

# Check disk space
DISK_USAGE=$(df -h / | awk 'NR==2 {print int($5)}')
if [ $DISK_USAGE -gt 80 ]; then
    echo "WARNING: Disk usage is ${DISK_USAGE}%"
    docker system prune -af
fi
EOF

chmod +x health-check.sh

# Schedule health checks
crontab -e
# Add: */5 * * * * /home/ubuntu/health-check.sh
```

11. Security Considerations

11.1 Security Hardening

bash

```bash
# Disable root login
sudo sed -i 's/PermitRootLogin yes/PermitRootLogin no/' /etc/ssh/sshd_config
sudo systemctl restart sshd

# Install fail2ban
sudo apt install -y fail2ban
sudo systemctl enable fail2ban
sudo systemctl start fail2ban

# Configure fail2ban for Docker
sudo tee /etc/fail2ban/jail.local <<EOF
[DEFAULT]
bantime = 3600
findtime = 600
maxretry = 5

[sshd]
enabled = true

[nginx-limit-req]
enabled = true
filter = nginx-limit-req
logpath = /home/ubuntu/my-working-prototype-dashbaord/nginx/logs/error.log
EOF

sudo systemctl restart fail2ban
```

11.2 Secret Management

```bash
# Use Docker secrets for sensitive data
docker secret create db_password - <<< "your-secure-password"

# Reference in docker-compose.yml:
# secrets:
#   db_password:
#     external: true
```

11.3 Network Security

```bash
```

```
# Restrict Docker API access
sudo tee /etc/docker/daemon.json <<EOF
{
  "hosts": ["unix:///var/run/docker.sock"],
  "iptables": true,
  "live-restore": true,
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3"
  }
}
EOF

sudo systemctl restart docker
```

---

12. Backup and Recovery

12.1 Automated Backup Setup

```bash
```

```bash
# Create comprehensive backup script
cat > /home/ubuntu/backup-all.sh <<'EOF'
#!/bin/bash

# Configuration
BACKUP_ROOT="/home/ubuntu/backups"
BACKUP_DIR="$BACKUP_ROOT/$(date +%Y%m%d_%H%M%S)"
RETENTION_DAYS=30
APP_DIR="/home/ubuntu/my-working-prototype-dashbaord"

# Create backup directory
mkdir -p "$BACKUP_DIR"

echo "Starting backup at $(date)"

# 1. Stop containers for consistency
cd "$APP_DIR"
docker compose stop

# 2. Backup application files
tar -czf "$BACKUP_DIR/application.tar.gz" "$APP_DIR" \
  --exclude="$APP_DIR/node_modules" \
  --exclude="$APP_DIR/data" \
  --exclude="$APP_DIR/logs"

# 3. Backup Docker volumes
for volume in $(docker volume ls -q | grep cyber-trust); do
  docker run --rm -v "$volume":/data -v "$BACKUP_DIR":/backup \
    alpine tar -czf "/backup/${volume}.tar.gz" /data
done

# 4. Backup database (if using MongoDB)
docker compose up -d mongodb
sleep 5
docker compose exec -T mongodb mongodump --out /backup
docker run --rm -v cyber-trust-mongodb:/data -v "$BACKUP_DIR":/backup \
  alpine tar -czf /backup/mongodb-dump.tar.gz /data/backup

# 5. Start containers again
docker compose up -d

# 6. Clean old backups
find "$BACKUP_ROOT" -type d -mtime +$RETENTION_DAYS -exec rm -rf {} + 2>/dev/null

echo "Backup completed at $(date)"
echo "Backup location: $BACKUP_DIR"
```

```
# 7. Optional: Upload to S3 or remote storage
# aws s3 sync "$BACKUP_DIR" s3://your-bucket/backups/$(date +%Y%m%d_%H%M%S)/
EOF

chmod +x /home/ubuntu/backup-all.sh
```

12.2 Recovery Procedure

```bash
```

```
# Create recovery script
cat > /home/ubuntu/recover.sh <<'EOF'
#!/bin/bash

if [ $# -ne 1 ]; then
    echo "Usage: $0 <backup-directory>"
    exit 1
fi

BACKUP_DIR="$1"
APP_DIR="/home/ubuntu/my-working-prototype-dashbaord"

echo "Starting recovery from $BACKUP_DIR"

# Stop all containers
cd "$APP_DIR"
docker compose down

# Restore application files
tar -xzf "$BACKUP_DIR/application.tar.gz" -C /

# Restore Docker volumes
for backup in "$BACKUP_DIR"/*.tar.gz; do
    if [[ $backup == *"cyber-trust"* ]]; then
        volume_name=$(basename "$backup" .tar.gz)
        docker volume create "$volume_name"
        docker run --rm -v "$volume_name":/data -v "$BACKUP_DIR":/backup \
            alpine tar -xzf "/backup/$(basename $backup)" -C /
    fi
done

# Start containers
docker compose up -d

echo "Recovery completed"
EOF

chmod +x /home/ubuntu/recover.sh
```

Appendix A: Quick Reference Commands

```
bash
```

```bash
# Start application
docker compose up -d

# Stop application
docker compose down

# Restart application
docker compose restart

# View logs
docker compose logs -f

# Update application
git pull && npm install && npm run build && docker compose restart nginx

# Check status
docker compose ps

# Enter container
docker compose exec nginx sh

# Clean up Docker
docker system prune -af

# Backup
./backup-all.sh

# Restore
./recover.sh /path/to/backup
```

Appendix B: Environment Variables Reference

Variable	Description	Default	Required
NODE_ENV	Environment mode	development	Yes
APP_PORT	Application port	3000	Yes
API_PORT	API server port	3001	Yes
MONGODB_URI	MongoDB connection string	mongodb://localhost:27017	No
REDIS_HOST	Redis server host	localhost	No
JWT_SECRET	JWT signing secret	-	Yes
SESSION_SECRET	Session secret	-	Yes
LOG_LEVEL	Logging level	info	No

Appendix C: Troubleshooting Checklist

- [ ] Docker installed and running?
- [ ] All required ports available?
- [ ] Sufficient disk space (> 5GB)?
- [ ] Sufficient memory (> 2GB)?
- [ ] .env file configured?
- [ ] Build directory exists and populated?
- [ ] Nginx configuration valid?
- [ ] Network connectivity between containers?
- [ ] Firewall rules configured?
- [ ] DNS resolving correctly?
- [ ] SSL certificates valid (if HTTPS)?
- [ ] Log files accessible and not full?
- [ ] Database connections working?
- [ ] API health check passing?

Support and Additional Resources

- **Docker Documentation**: https://docs.docker.com/
- **Docker Compose Reference**: https://docs.docker.com/compose/
- **Nginx Documentation**: https://nginx.org/en/docs/
- **Node.js Best Practices**: https://github.com/goldbergyoni/nodebestpractices
- **Security Headers**: https://securityheaders.com/
- **SSL Test**: https://www.ssllabs.com/ssltest/

Version History

Version	Date	Changes
1.0.0	2025-08-07	Initial comprehensive deployment guide

Note: This guide assumes Ubuntu 22.04/24.04 LTS. Adjust commands for other distributions as needed. Always test in a staging environment before deploying to production.