Deployment Guide - SMSC Firewall

This guide covers deploying the SMSC Firewall in production environments.

Deployment Options

1. **Docker Compose** (Recommended for single-server deployments)
2. **Kubernetes** (Recommended for high-availability)
3. **Systemd Services** (Traditional deployment)
4. **Cloud Platforms** (AWS, GCP, Azure)

Option 1: Docker Compose Deployment

Prerequisites

- Docker 20.10+
- Docker Compose 2.0+
- 8GB RAM minimum
- 4 CPU cores minimum
- 50GB disk space

Quick Start

```bash
# Clone repository
cd smsc-firewall

# Create environment file
cat > .env << EOF
DB_PASSWORD=your_secure_db_password
REDIS_PASSWORD=your_secure_redis_password
SMSC_GATEWAY_URL=http://your-smsc-gateway:8081
SMSC_API_KEY=your_smsc_api_key
GRAFANA_PASSWORD=your_grafana_password
EOF

# Start services
cd deployment
docker-compose up -d

# Check status
docker-compose ps

# View logs
docker-compose logs -f firewall-backend
```

Verify Deployment

```bash
# Health check
curl http://localhost:8080/health

# Access dashboard
open http://localhost:3000

# Access Grafana
open http://localhost:3001  # admin / your_grafana_password
```

Option 2: Kubernetes Deployment

Prerequisites

- Kubernetes 1.24+
- kubectl configured
- Helm 3.0+ (optional)

Deploy with kubectl

```bash
# Create namespace
kubectl create namespace smsc-firewall

# Create secrets
kubectl create secret generic smsc-firewall-secrets \
  --from-literal=db-password=your_db_password \
  --from-literal=redis-password=your_redis_password \
  --from-literal=smsc-api-key=your_api_key \
  -n smsc-firewall

# Deploy resources
kubectl apply -f deployment/k8s/ -n smsc-firewall

# Check status
kubectl get pods -n smsc-firewall

# Get service URL
kubectl get svc -n smsc-firewall
```

Scaling

```bash
# Scale backend
kubectl scale deployment firewall-backend --replicas=3 -n smsc-firewall

# Scale frontend
kubectl scale deployment firewall-frontend --replicas=2 -n smsc-firewall
```

Option 3: Systemd Service Deployment

Backend Service

```bash
# Build backend
cd backend
go build -o /usr/local/bin/smsc-firewall

# Create systemd service
sudo cat > /etc/systemd/system/smsc-firewall.service << EOF
[Unit]
Description=SMSC Firewall Service
After=network.target postgresql.service redis.service

[Service]
Type=simple
User=smsc
WorkingDirectory=/opt/smsc-firewall
ExecStart=/usr/local/bin/smsc-firewall
Restart=always
RestartSec=10
Environment="CONFIG_PATH=/etc/smsc-firewall"

[Install]
WantedBy=multi-user.target
EOF

# Enable and start
sudo systemctl enable smsc-firewall
sudo systemctl start smsc-firewall
sudo systemctl status smsc-firewall
```

Frontend Service (Nginx)

```bash
# Build frontend
cd frontend
npm run build

# Copy to nginx
sudo cp -r build/* /var/www/smsc-firewall/

# Nginx configuration
sudo cat > /etc/nginx/sites-available/smsc-firewall << EOF
server {
    listen 80;
    server_name your-domain.com;
    root /var/www/smsc-firewall;

    location / {
        try_files \$uri /index.html;
    }

    location /api {
        proxy_pass http://localhost:8080;
        proxy_http_version 1.1;
        proxy_set_header Upgrade \$http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host \$host;
    }
}
EOF

# Enable site
sudo ln -s /etc/nginx/sites-available/smsc-firewall /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl restart nginx
```

Production Configuration

1. Database Tuning (PostgreSQL)

sql

```
-- postgresql.conf
max_connections = 200
shared_buffers = 2GB
effective_cache_size = 6GB
maintenance_work_mem = 512MB
checkpoint_completion_target = 0.9
wal_buffers = 16MB
default_statistics_target = 100
random_page_cost = 1.1
effective_io_concurrency = 200
work_mem = 10MB
min_wal_size = 1GB
max_wal_size = 4GB
```

2. Redis Configuration

conf

```
# redis.conf
maxmemory 4gb
maxmemory-policy allkeys-lru
save 900 1
save 300 10
save 60 10000
appendonly yes
appendfsync everysec
```

3. Backend Configuration

```yaml
# config/config.yaml
server:
  port: "8080"
  mode: "release"
  max_connections: 10000
  read_timeout: 30
  write_timeout: 30

database:
  host: "postgres-host"
  port: 5432
  user: "smsc_firewall"
  password: "${DB_PASSWORD}"
  dbname: "smsc_firewall"
  max_conns: 200
  min_conns: 20

redis:
  host: "redis-host"
  port: 6379
  password: "${REDIS_PASSWORD}"
  pool_size: 100

firewall:
  max_tps: 100000
  default_rate_limit: 100
  enable_content_filter: true
  enable_fraud_detect: true
  log_level: "info"
```

High Availability Setup

Load Balancer Configuration (HAProxy)

conf

```
# haproxy.cfg
global
    maxconn 50000

defaults
    mode http
    timeout connect 5000ms
    timeout client 50000ms
    timeout server 50000ms

frontend smsc_firewall_frontend
    bind *:80
    default_backend smsc_firewall_backend

backend smsc_firewall_backend
    balance roundrobin
    option httpchk GET /health
    server firewall1 10.0.1.10:8080 check
    server firewall2 10.0.1.11:8080 check
    server firewall3 10.0.1.12:8080 check
```

Database Replication

bash

```
# Master-Slave PostgreSQL setup
# On master:
wal_level = replica
max_wal_senders = 3
wal_keep_segments = 64

# On slave:
standby_mode = 'on'
primary_conninfo = 'host=master_ip port=5432 user=replicator password=xxx'
```

Redis Sentinel

```bash
# sentinel.conf
sentinel monitor smsc-firewall-redis redis-master 6379 2
sentinel down-after-milliseconds smsc-firewall-redis 5000
sentinel parallel-syncs smsc-firewall-redis 1
sentinel failover-timeout smsc-firewall-redis 10000
```

Monitoring Setup

Prometheus Configuration

```yaml
# prometheus.yml
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'smsc-firewall'
    static_configs:
      - targets: ['firewall-backend:8080']
```

Grafana Dashboards

1. Import dashboard: `deployment/grafana/dashboards/smsc-firewall.json`
2. Configure datasource: Prometheus (http://prometheus:9090)

Alerts

```yaml
# alerts.yml
groups:
  - name: smsc_firewall
    rules:
      - alert: HighBlockRate
        expr: rate(firewall_messages_blocked_total[5m]) > 100
        annotations:
          summary: "High message block rate detected"

      - alert: FirewallDown
        expr: up{job="smsc-firewall"} == 0
        for: 1m
        annotations:
          summary: "Firewall instance is down"
```

Security Hardening

1. Enable TLS

```bash
# Generate certificates
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
  -keyout /etc/ssl/private/smsc-firewall.key \
  -out /etc/ssl/certs/smsc-firewall.crt

# Update nginx config
server {
    listen 443 ssl;
    ssl_certificate /etc/ssl/certs/smsc-firewall.crt;
    ssl_certificate_key /etc/ssl/private/smsc-firewall.key;
    ...
}
```

2. Firewall Rules

```bash
# UFW
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
sudo ufw allow from 10.0.0.0/8 to any port 8080
sudo ufw enable

# iptables
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT
iptables -A INPUT -p tcp --dport 8080 -s 10.0.0.0/8 -j ACCEPT
```

3. API Authentication

Add JWT authentication to the backend:

```go
// middleware/auth.go
func AuthMiddleware() gin.HandlerFunc {
    return func(c *gin.Context) {
        token := c.GetHeader("Authorization")
        if token == "" {
            c.AbortWithStatus(401)
            return
        }
        // Validate token
        c.Next()
    }
}
```

Backup Strategy

Database Backup

```bash
# Daily backup script
#!/bin/bash
DATE=$(date +%Y%m%d)
pg_dump smsc_firewall > /backups/smsc_firewall_$DATE.sql
gzip /backups/smsc_firewall_$DATE.sql

# Keep only last 7 days
find /backups -name "*.sql.gz" -mtime +7 -delete

# Cron job
0 2 * * * /opt/scripts/backup-db.sh
```

Redis Backup

```bash
# Enable AOF persistence
redis-cli CONFIG SET appendonly yes

# Backup RDB file
cp /var/lib/redis/dump.rdb /backups/redis_$(date +%Y%m%d).rdb
```

Disaster Recovery

Recovery Procedure

1. **Database Recovery**

```bash
# Stop application
systemctl stop smsc-firewall

# Restore database
gunzip -c backup.sql.gz | psql smsc_firewall

# Start application
systemctl start smsc-firewall
```

2. **Redis Recovery**

```bash
# Stop Redis
systemctl stop redis

# Restore RDB
cp backup.rdb /var/lib/redis/dump.rdb

# Start Redis
systemctl start redis
```

Performance Optimization

1. Connection Pooling

```yaml
database:
  max_conns: 200
  min_conns: 20
  conn_max_lifetime: 3600

redis:
  pool_size: 100
  min_idle_conns: 10
```

2. Cache Configuration

```go
// Increase cache sizes
const (
    ruleCacheSize      = 50000
    blacklistCacheSize = 100000
    rateLimitWindow    = 1 * time.Minute
)
```

3. Database Indexing

```sql
-- Add indexes for common queries
CREATE INDEX idx_traffic_log_timestamp ON traffic_log(timestamp);
CREATE INDEX idx_traffic_log_source ON traffic_log(source_addr);
CREATE INDEX idx_traffic_log_action ON traffic_log(action);
CREATE INDEX idx_fraud_alert_status ON fraud_alert(status);
```

Troubleshooting

Common Issues

1. **High Memory Usage**
   - Check Redis memory: `redis-cli INFO memory`
   - Adjust cache sizes
   - Enable memory limits
2. **Slow Queries**
   - Enable slow query log: `postgresql.conf`
   - Add indexes
   - Optimize queries
3. **Connection Timeouts**
   - Increase timeout values
   - Check network latency
   - Verify firewall rules

Logs

```bash
# Backend logs
tail -f /var/log/smsc-firewall/app.log

# Database logs
tail -f /var/log/postgresql/postgresql-15-main.log

# Redis logs
tail -f /var/log/redis/redis-server.log

# Nginx logs
tail -f /var/log/nginx/access.log
```

Maintenance

Regular Tasks

- **Daily**: Check logs, monitor alerts
- **Weekly**: Review fraud alerts, update blacklists
- **Monthly**: Database optimization, backup verification
- **Quarterly**: Security updates, performance review

Updates

```bash
# Backup before update
./scripts/backup.sh

# Pull latest changes
git pull origin main

# Rebuild and restart
docker-compose down
docker-compose build
docker-compose up -d

# Verify
curl http://localhost:8080/health
```

Support

For deployment issues:

- Check logs in `/var/log/smsc-firewall/`
- Review health endpoint: `/health`
- Contact support with system details