NGINX in Docker Swarm

Production-Grade Configuration

JANUARY 12, 2026

THEDECIPHERIST.COM

TABLE OF CONTENTS

By a Human + Claude AI | 10 Years of Production Experience Documented

INTRODUCTION

Running NGINX in Docker Swarm provides high availability, automatic failover, rolling updates, and simplified scaling. This guide is based on a real production environment that has maintained 100% uptime serving high-traffic web applications for a billion-dollar company.

What this setup provides:

- Zero-downtime deployments
- Automatic container restart on failure
- Load balancing across multiple NGINX replicas
- Secure secrets management for SSL certificates
- Integration with monitoring systems (Datadog)
- ModSecurity WAF protection
- Stream proxying for databases (MongoDB, Elasticsearch)
- Separate production and development environments

This is Part 3 of my NGINX series:

- Part 1: NGINX Best Practices 2026
- Part 2: OWASP & ModSecurity Deep Dive
- Part 3: Docker Swarm Deployment (this guide)

TABLE OF CONTENTS

ARCHITECTURE OVERVIEW

```
┌──────────────────────────────────────────────────┐
│              Load Balancer (ELB)                 │
│                  Port 80/443                     │
└──────────────────┬───────────────────────────────┘
                   │
    ┌──────────────┼──────────────────┐
    │              │                  │
    ▼              ▼                  ▼
┌──────────┐   ┌──────────┐   ┌──────────┐
│ Manager 1│   │ Worker 1 │   │ Worker 2 │
│ ┌──────┐ │   │ ┌──────┐ │   │ ┌──────┐ │
│ │ NGINX│ │◄─►│ │ NGINX│ │◄─►│ │ NGINX│ │
│ │Replica│ │ Overlay │ │Replica│ │ Network │ │Replica│ │
│ └──────┘ │ (encrypted)└──────┘ │   │ └──────┘ │
└──────────┘   └──────────┘   └──────────┘
    │              │                  │
    └──────────────┼──────────────────┘
                   │
         ┌─────────┴─────────────┐
         │   Backend Services    │
         │ (Node.js, MongoDB, etc.) │
         └───────────────────────┘
```

Key Architecture Points:

- NGINX replicas are spread across nodes using `max_replicas_per_node: 1`
- Encrypted overlay network ensures secure inter-node communication
- Each replica operates independently and can handle requests
- If a node fails, remaining replicas continue serving traffic
- Docker Swarm's routing mesh distributes incoming requests

PREREQUISITES

Swarm Initialization

If you haven't initialized your swarm:

```
# On the manager node
docker swarm init --advertise-addr <MANAGER-IP>

# On worker nodes (use the token from init output)
docker swarm join --token <TOKEN> <MANAGER-IP>:2377
```

Verify Swarm Status

```
docker node ls
docker info | grep -A 5 "Swarm"
```

Required Ports

Ensure these ports are open between swarm nodes:

PORT	PROTOCOL	PURPOSE
2377	TCP	Cluster management communications
7946	TCP/UDP	Inter-node communication
4789	UDP	Overlay network traffic
50	ESP	Encrypted overlay network (IPsec)

NETWORK CONFIGURATION

Creating the Overlay Network

This is CRITICAL. Without proper network configuration, your swarm will have communication issues.

```
docker network create \
  --opt encrypted \
  --subnet 172.20.0.0/16 \
  --attachable \
  --driver overlay \
  appnet
```

Network Flags Explained

FLAG	PURPOSE	WHY IT MATTERS
`--opt encrypted`	Enables IPsec encryption between nodes	**Security**: Without this, traffic between nodes is plain text
`--subnet 172.20.0.0/16`	Custom IP range	**Avoid conflicts**: Prevents overlap with cloud VPC, default Docker ranges
`--attachable`	Allows standalone containers to connect	**Monitoring**: Required for monitoring agents and other standalone containers
`--driver overlay`	Multi-host networking	**Required**: Standard for swarm services

DNS Configuration

Docker's internal DNS server is at `127.0.0.11`. Configure NGINX to use it:

```
http {
    # Docker DNS resolver
    resolver 127.0.0.11 ipv6=off valid=10s;

    # ... rest of config
}
```

Important: The `valid=10s` parameter tells NGINX to re-resolve DNS every 10 seconds. This is crucial for swarm because container IPs can change during scaling or updates.

Why Service Names Matter

NEVER hardcode IP addresses in your services. Docker Swarm is designed to handle container lifecycle - containers can be destroyed and recreated at any time with new IPs. Always use service names for communication.

```
# WRONG - Never do this. Container IPs change constantly.
upstream backend {
    server 172.20.0.15:8080;
}

# CORRECT - Let Docker DNS handle resolution
upstream backend {
    server backend-service:8080;
}
```

Verifying Network Setup

```
# List networks
docker network ls

# Inspect network details
docker network inspect appnet

# Check connected services
docker network inspect appnet --format '{{range .Containers}}{{.Name}} {{end}}'

# Test DNS resolution from inside a container
docker exec <container_id> nslookup backend-service
```

BUILDING THE NGINX DOCKER IMAGE

Complete Production Dockerfile

This Dockerfile builds NGINX with ModSecurity WAF and optional Datadog tracing:

```
# syntax=docker/dockerfile:1
ARG NGINX_VERSION=1.27.0
ARG MODSECURITY_VERSION=v3

FROM nginx:$NGINX_VERSION as base

# Copy Datadog module (optional - if using Datadog)
COPY ngx_http_datadog_module-amd64-$NGINX_VERSION.so.tgz /usr/lib/nginx/modules

# Create required directories
RUN mkdir -p /var/cache/nginx_cache && \
    mkdir -p /var/logs/nginx && \
    mkdir -p /etc/nginx/sites-enabled && \
    mkdir -p /etc/nginx/sites-enabled-dev && \
    mkdir -p /tmp

# Install dependencies
RUN apt update && \
    apt install -y \
        git \
        dos2unix \
        apt-utils \
        autoconf \
        automake \
        build-essential \
        libcurl4-openssl-dev \
        libgeoip-dev \
        liblmdb-dev \
        libpcre3 \
        libpcre3-dev \
        libtool \
        libxml2-dev \
        libyajl-dev \
        pkgconf \
        wget \
        tar \
        jq \
        zlib1g-dev && \
    # Set timezone (adjust to your timezone)
    ln -snf /usr/share/zoneinfo/America/New_York /etc/localtime && \
    echo America/New_York > /etc/timezone && \
    # Extract Datadog module (optional)
    tar -xzf "/usr/lib/nginx/modules/ngx_http_datadog_module-amd64-$NGINX_VERSI
        -C "/usr/lib/nginx/modules" && \
    rm /usr/lib/nginx/modules/ngx_http_datadog_module-amd64-$NGINX_VERSION.so.t
```

```
# Build ModSecurity from source
RUN git clone --depth 1 -b v3/master --single-branch \
    https://github.com/SpiderLabs/ModSecurity

WORKDIR /ModSecurity

RUN git submodule init && \
    git submodule update && \
    ./build.sh && \
    ./configure && \
    make && \
    make install

# Build NGINX ModSecurity connector
RUN git clone --depth 1 https://github.com/SpiderLabs/ModSecurity-nginx.git
RUN wget http://nginx.org/download/nginx-$NGINX_VERSION.tar.gz
RUN tar zxvf nginx-$NGINX_VERSION.tar.gz

WORKDIR /ModSecurity/nginx-$NGINX_VERSION

RUN ./configure --with-compat --add-dynamic-module=../ModSecurity-nginx
RUN make modules
RUN cp objs/ngx_http_modsecurity_module.so /usr/lib/nginx/modules

# Final stage - clean image
FROM base AS final

# Clean up build artifacts to reduce image size
RUN rm -rf /ModSecurity

# Expose ports
EXPOSE 80    # HTTP
EXPOSE 81    # Status/monitoring
EXPOSE 82    # Health check
EXPOSE 83    # Additional services
EXPOSE 443   # HTTPS

# Copy NGINX configuration
COPY nginx/ /etc/nginx/

# Create symlinks for sites
RUN ln -s /etc/nginx/sites-available/* /etc/nginx/sites-enabled/
RUN ln -s /etc/nginx/sites-available-dev/* /etc/nginx/sites-enabled-dev/

WORKDIR /etc/nginx
```

Build Script

Create a build script for consistent builds:

```
#!/bin/bash
# build.sh

VERSION=${1:-latest}
REGISTRY="your-registry.example.com"

docker build \
    -t $REGISTRY/nginx:$VERSION \
    -t $REGISTRY/nginx:latest \
    .

# Push to registry
docker push $REGISTRY/nginx:$VERSION
docker push $REGISTRY/nginx:latest

echo "Built and pushed $REGISTRY/nginx:$VERSION"
```

Multi-Stage Build Benefits

The Dockerfile uses multi-stage builds:

1. **Build stage**: Compiles ModSecurity (large, ~2GB with build tools)
2. **Final stage**: Only includes runtime dependencies (~200MB)

This reduces image size significantly and improves deployment speed across your swarm nodes.

DOCKER COMPOSE CONFIGURATION

Base Compose File Structure

For large projects, split your compose files for better organization:

```
project/
├── docker-compose.yaml          # Main orchestrator
├── docker-compose_nginx.yaml    # NGINX service definition
├── docker-compose_backend.yaml  # Backend services
├── docker-compose_monitoring.yaml  # Monitoring stack
└── .env                         # Environment variables
```

Main Compose File (docker-compose.yaml)

```
version: "3.8"

services:
    nginx:
        extends:
            file: docker-compose_nginx.yaml
            service: nginx

    backend:
        extends:
            file: docker-compose_backend.yaml
            service: backend

volumes:
    nginx-logs:
    backend-data:

networks:
    appnet:
        external: true
        name: appnet

secrets:
    nginx_dhparams_pem:
        file: ./.secrets/ssl/dhparams.pem
    nginx_server_pem:
        file: ./.secrets/ssl/server.pem
    nginx_server_key:
        file: ./.secrets/ssl/server.key
    nginx_ca_trust:
        file: ./.secrets/ssl/ca-trust.crt
```

NGINX Service Compose File (docker-compose_nginx.yaml)

```
version: "3.8"

services:
    nginx:
        # DNS Configuration - fallback if Docker DNS fails
        dns:
            - 8.8.8.8

        # Use init process for proper signal handling
        # This ensures NGINX receives SIGTERM correctly for graceful shutdown
        init: true

        # Monitoring autodiscovery labels (Datadog example)
        labels:
            com.datadoghq.ad.check_names: '["nginx"]'
            com.datadoghq.ad.logs: >-
                [
                    {"type": "docker", "source": "nginx", "service": "nginx"},
                    {"type": "file", "source": "modsecurity", "service": "nginx
                     "auto_multi_line_detection": true, "path": "/var/log/modse
                ]
            com.datadoghq.ad.init_configs: '[{}]'
            com.datadoghq.ad.instances: '[{"nginx_status_url":"http://localhost

        # Environment variables
        environment:
            - DD_AGENT_HOST=datadog-agent
            - DD_TRACE_AGENT_PORT=8126
            - DD_TRACE_ENABLED=true
            - DD_PROFILING_ENABLED=true
            - NGINX_RESOLVER=${NGINX_RESOLVER:-127.0.0.11}

        # Deployment configuration
        deploy:
            mode: replicated
            replicas: ${NGINX_REPLICAS:-2}

            placement:
                max_replicas_per_node: 1
                # Optional: restrict to specific nodes
                # constraints:
                #   - "node.role == worker"
                #   - "node.labels.nginx == true"

            # Rolling update configuration
            update_config:
```

```
            parallelism: 1
            delay: 20s
            failure_action: rollback
            monitor: 10s
            order: stop-first

        # Rollback configuration
        rollback_config:
            parallelism: 1
            delay: 20s
            monitor: 10s

        # Restart policy
        restart_policy:
            condition: on-failure
            delay: 10s
            max_attempts: 30
            window: 120s

        # Resource limits
        resources:
            limits:
                cpus: '1.0'
                memory: 1024M
            reservations:
                cpus: '0.50'
                memory: 512M

    # Build configuration (for local builds)
    build:
        context: ./nginx
        cache_from:
            - "your-registry.example.com/nginx:latest"

    # Image reference
    image: "your-registry.example.com/nginx:${BUILD_VERSION:-latest}"

    # Port mappings
    ports:
        - "80:80"       # HTTP
        - "81:81"       # NGINX status
        - "82:82"       # Health check
        - "443:443"     # HTTPS

    # Volume mounts
    volumes:
        - /docker/swarm/nginx:/var/log
        # Network
        networks:
            appnet:

        # Secrets
        secrets:
            - nginx_server_pem
            - nginx_server_key
            - nginx_dhparams_pem
            - nginx_ca_trust

        # Configs
        configs:
            - nginx_blocked_ips

# External configs
configs:
    nginx_blocked_ips:
        external: true

# Volumes
volumes:
    nginx-logs:

# Secrets (external for production)
secrets:
    nginx_server_pem:
        external: true
    nginx_server_key:
        external: true
    nginx_dhparams_pem:
        external: true
    nginx_ca_trust:
        external: true

# Networks
networks:
    appnet:
        external: true
        name: appnet
```

SECRETS MANAGEMENT

Why Use Docker Secrets?

Docker secrets provide secure storage for sensitive data:

- **Encrypted at rest and in transit**
- **Only accessible to services that need them**
- **Mounted as files in `/run/secrets/`**
- **Never exposed in `docker inspect` or logs**
- **Immutable** - can't be changed, only replaced

Creating Secrets

```
# From files
docker secret create nginx_server_pem ./ssl/server.pem
docker secret create nginx_server_key ./ssl/server.key
docker secret create nginx_dhparams_pem ./ssl/dhparams.pem
docker secret create nginx_ca_trust ./ssl/ca-trust.crt

# From stdin (for passwords, API keys)
echo "my-secret-value" | docker secret create my_api_key -

# List secrets
docker secret ls

# Inspect secret metadata (NOT the content - that's never exposed)
docker secret inspect nginx_server_pem
```

Using Secrets in NGINX Configuration

Secrets are mounted at `/run/secrets/`:

```
# ssl.conf
ssl_certificate /run/secrets/nginx_server_pem;
ssl_certificate_key /run/secrets/nginx_server_key;
ssl_dhparam /run/secrets/nginx_dhparams_pem;
ssl_trusted_certificate /run/secrets/nginx_ca_trust;

ssl_protocols TLSv1.2 TLSv1.3;
ssl_session_cache shared:SSL:60m;
ssl_session_timeout 60m;
ssl_prefer_server_ciphers on;
ssl_ciphers 'ECDHE-RSA-CHACHA20-POLY1305:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-
ssl_stapling on;
ssl_stapling_verify on;
```

Rotating Secrets

Secrets are immutable - you can't update them, only replace them:

```
# 1. Create new secret with different name
docker secret create nginx_server_pem_v2 ./new-ssl/server.pem

# 2. Update service to use new secret
docker service update \
    --secret-rm nginx_server_pem \
    --secret-add source=nginx_server_pem_v2,target=nginx_server_pem \
    nginx

# 3. Verify the service is working with new cert

# 4. Remove old secret
docker secret rm nginx_server_pem
```

CONFIGS MANAGEMENT

Docker Configs vs Secrets

FEATURE	CONFIGS	SECRETS
Encryption at rest	No	Yes
Use case	Non-sensitive configuration	Sensitive data (certs, passwords)
Mount location	Customizable	`/run/secrets/`
Size limit	500KB	500KB

Creating and Using Configs

```
# Create config from file
docker config create nginx_blocked_ips ./nginx/blockips.conf

# List configs
docker config ls

# Inspect config (you CAN see the content, unlike secrets)
docker config inspect nginx_blocked_ips --pretty
```

In compose:

```
services:
    nginx:
        configs:
            - source: nginx_blocked_ips
              target: /etc/nginx/blockips.conf
              mode: 0444  # Read-only
```

Updating Configs

Like secrets, configs are immutable:

```
# Create new version
docker config create nginx_blocked_ips_v2 ./nginx/blockips_updated.conf

# Update service
docker service update \
    --config-rm nginx_blocked_ips \
    --config-add source=nginx_blocked_ips_v2,target=/etc/nginx/blockips.conf \
    nginx

# Remove old config
docker config rm nginx_blocked_ips
```

SERVICE DEPLOYMENT STRATEGIES

Deployment Configuration Explained

```
deploy:
    mode: replicated
    replicas: 2

    placement:
        max_replicas_per_node: 1
        # constraints:
        #   - "node.role == worker"
        #   - "node.labels.nginx == true"

    update_config:
        parallelism: 1      # Update one container at a time
        delay: 20s          # Wait 20s between each container update
        failure_action: rollback  # Automatically rollback on failure
        monitor: 10s        # Monitor new container for 10s before continuing
        order: stop-first   # Stop old container before starting new one

    rollback_config:
        parallelism: 1
        delay: 20s
        monitor: 10s

    restart_policy:
        condition: on-failure  # Only restart on failure, not on manual stop
        delay: 10s            # Wait 10s before restart attempt
        max_attempts: 30      # Maximum restart attempts
        window: 120s          # Time window for counting restart attempts
```

Placement Strategies

Spread across nodes (recommended for HA):

```
placement:
    max_replicas_per_node: 1
```

This ensures if one node dies, you still have replicas on other nodes handling traffic.

Run only on workers:

```
placement:
    constraints:
        - "node.role == worker"
```

Run on specific labeled nodes:

```
placement:
    constraints:
        - "node.labels.nginx == true"
```

Label your nodes:

```
docker node update --label-add nginx=true worker-1
docker node update --label-add nginx=true worker-2
```

Update Strategies

Rolling update (zero downtime) - Recommended:

```
update_config:
    parallelism: 1
    delay: 20s
    failure_action: rollback
    order: stop-first
```

Blue-green style (start new first):

```
update_config:
    parallelism: 1
    delay: 10s
    order: start-first  # Start new before stopping old
```

RESOURCE MANAGEMENT

Resource Limits and Reservations

```
resources:
    limits:
        cpus: '1.0'        # Maximum 1 CPU core
        memory: 1024M      # Maximum 1GB RAM
    reservations:
        cpus: '0.50'       # Guaranteed 0.5 CPU cores
        memory: 512M       # Guaranteed 512MB RAM
```

Why Both Limits and Reservations?

- **Reservations**: Guaranteed resources. Swarm won't schedule the container on a node that can't provide these resources.
- **Limits**: Maximum resources. Container is killed (OOM) or throttled if it tries to exceed these.

Recommended Resources for NGINX

WORKLOAD	CPU LIMIT	MEMORY LIMIT	CPU RESERVE	MEMORY RESERVE
Light (simple proxy)	0.5	256M	0.25	128M
Medium (caching, SSL)	1.0	512M	0.5	256M
Heavy (high traffic)	2.0	1024M	1.0	512M
With ModSecurity WAF	2.0	1024M	1.0	512M

Monitoring Resource Usage

```
# Real-time stats for all NGINX containers
docker stats $(docker ps -q --filter name=nginx)

# Service resource usage
docker service ps nginx --format "table {{.Node}}\t{{.CurrentState}}\t{{.Error}
```

HEALTH CHECKS

Why Separate Ports?

Using separate ports for health checks:

1. **Doesn't impact production traffic** on port 80
2. **Can have different access controls**
3. **Allows disabling access logs** for health checks
4. **Cleaner separation of concerns**

Health Check Endpoints in NGINX

```
# Port 82 - Load Balancer / ELB Health Check
server {
    listen 82;

    # Restrict to internal networks
    allow 127.0.0.1;
    allow 10.0.0.0/8;
    allow 172.16.0.0/12;
    allow 192.168.0.0/16;
    deny all;

    location /health {
        access_log off;  # Don't log health checks
        add_header Content-Type text/plain;
        return 200 "OK";
    }
}

# Port 81 - NGINX Status for Metrics (Datadog, Prometheus)
server {
    listen 81;

    allow 127.0.0.1;
    allow 10.0.0.0/8;
    deny all;

    location /nginx_status {
        stub_status on;
        server_tokens on;  # Required for version info in metrics
    }
}

# Port 86 - Deep Health Check (verifies upstream connectivity)
server {
    listen 86;

    allow 172.20.0.0/16;  # Swarm network only
    deny all;

    location /deep_health {
        proxy_pass http://backend_upstream/api/health;
        proxy_connect_timeout 5s;
        proxy_read_timeout 10s;
    }
}
```

Container Health Check

Add to Dockerfile:

```
HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 \
    CMD curl -sf http://localhost:82/health || exit 1
```

Or in compose:

```
healthcheck:
    test: ["CMD", "curl", "-sf", "http://localhost:82/health"]
    interval: 30s
    timeout: 10s
    retries: 3
    start_period: 5s
```

Health Check Best Practices

1. **Use separate ports** - Don't mix with production traffic
2. **Restrict access** - Only internal IPs
3. **Disable logging** - Reduces log noise significantly
4. **Keep them fast** - Health checks should return instantly
5. **Consider deep checks** - Verify upstream connectivity for critical services

---

LOGGING AND MONITORING

JSON Log Format for Log Aggregation

Structured JSON logs enable powerful filtering and correlation in log aggregation tools:

```
# Maps for extracting useful data
map $msec $msec_no_decimal {
    ~(.*)\.(.*) $1$2;
}

map $uri $file_extension {
    default "";
    ~\.([0-9a-z]+)$ $1;
}

map $http_referer $http_referer_hostname {
    ~^.*://([^/?]+) $1;
}

# JSON log format
log_format json_log escape=json '{'
    '"timestamp":$msec_no_decimal,'
    '"http":{'
        '"method":"$request_method",'
        '"status_code":$status,'
        '"uri":"$request_uri",'
        '"useragent":"$http_user_agent",'
        '"referer":"$http_referer"'
    '},'
    '"nginx":{'
        '"request_time":$request_time,'
        '"upstream_response_time":"$upstream_response_time",'
        '"upstream_addr":"$upstream_addr",'
        '"cache_status":"$upstream_cache_status"'
    '},'
    '"network":{'
        '"bytes_sent":$bytes_sent,'
        '"client_ip":"$remote_addr",'
        '"x_forwarded_for":"$http_x_forwarded_for"'
    '}'
'}';

access_log /dev/stdout json_log;
error_log /dev/stderr warn;
```

Per-Location Log Types

Tag different endpoints for better filtering:

```
location /api {
    set $log_type api;
    access_log /dev/stdout json_log;
    # ...
}

location ~* \.(js|css|png|jpg|gif|ico)$ {
    set $log_type static;
    access_log /dev/stdout json_log;
    # ...
}

location / {
    set $log_type frontend;
    access_log /dev/stdout json_log;
    # ...
}
```

Volume Mounts for Log Persistence

```
volumes:
    # Mount to host for log aggregation tools
    - /docker/swarm/nginx:/var/log
```

This allows:

- Log persistence across container restarts
- External log collectors to access logs
- ModSecurity audit logs to be collected

MULTI-ENVIRONMENT SETUP

Directory Structure

Run production and development sites on the same NGINX instance:

```
nginx/
├── sites-available/        # Production site configs
│   ├── site1.conf
│   ├── site2.conf
│   └── ...
├── sites-available-dev/    # Development site configs
│   ├── site1_dev.conf
│   ├── site2_dev.conf
│   └── ...
├── sites-enabled/          # Symlinks to production (created at build)
├── sites-enabled-dev/      # Symlinks to development (created at build)
├── locations.conf          # Production location blocks
├── locations_dev.conf      # Development location blocks
├── proxy_headers.conf      # Production headers (strict)
└── proxy_headers_dev.conf  # Development headers (relaxed)
```

Separate Upstreams

```
# Production upstream
upstream backend_upstream {
    server backend-service:8080;
    keepalive 32;
}

# Development upstream
upstream backend_dev_upstream {
    server backend-service-dev:8080;
    keepalive 32;
}
```

Include Both in nginx.conf

```
http {
    # ... base config ...

    # Production Sites
    include /etc/nginx/sites-enabled/*.conf;

    # Development Sites
    include /etc/nginx/sites-enabled-dev/*.conf;
}
```

Environment-Specific Headers

Production (proxy_headers.conf) - Strict security:

```
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 Strict-Transport-Security "max-age=31536000; includeSubDomains" alwa
add_header Content-Security-Policy "default-src 'self'; ..." always;
```

Development (proxy_headers_dev.conf) - Relaxed for testing:

```
add_header X-Frame-Options "" always;
# More permissive CSP or none for local development
```

STREAM PROXYING

TCP/UDP Proxying with Stream Module

NGINX can proxy non-HTTP protocols using the stream module. This is useful for databases, message queues, and other TCP services.

MongoDB Proxy Configuration

```
# mongo.conf - MUST be outside the http block
stream {
    upstream mongo_backend {
        server mongodb-primary.internal:27017;
        server mongodb-secondary1.internal:27017;
        server mongodb-secondary2.internal:27017;
    }

    server {
        listen 27017;

        # Connection settings
        proxy_connect_timeout 1s;
        proxy_timeout 3s;

        proxy_pass mongo_backend;
    }
}
```

Elasticsearch Proxy with Caching

```
# elasticsearch.conf - inside http block
upstream elasticsearch {
    server es-node1.internal:9200;
    server es-node2.internal:9200;
    server es-node3.internal:9200;
}

server {
    listen 9200;

    location / {
        access_log on;
        proxy_redirect off;
        proxy_http_version 1.1;
        proxy_connect_timeout 5s;
        proxy_read_timeout 10s;

        # Caching for read-heavy workloads
        proxy_cache es_cache;
        proxy_cache_methods GET HEAD;
        proxy_cache_valid 200 1m;
        proxy_cache_key $host$uri$args;
        proxy_cache_bypass $http_cache_buster $arg_nocache;
        proxy_cache_use_stale updating error timeout http_500 http_502 http_503

        # Hide backend info
        proxy_hide_header X-Powered-By;
        add_header X-Proxy-Cache $upstream_cache_status;

        proxy_pass http://elasticsearch;
    }
}
```

Include Order in nginx.conf

Important: Stream blocks must be outside the http block!

```
# Load modules
load_module modules/ngx_http_modsecurity_module.so;

# Stream block - OUTSIDE http block
include /etc/nginx/mongo.conf;

http {
    # ... http config ...

    # Elasticsearch - INSIDE http block (it's HTTP)
    include /etc/nginx/elasticsearch.conf;
}
```

ROLLING UPDATES AND ROLLBACKS

Zero-Downtime Update Process

With the configuration above, here's what happens during an update:

1. Swarm starts a new container with the new image
2. New container passes health checks
3. Swarm updates the routing mesh to include new container
4. Swarm stops the old container
5. Process repeats for each replica (with `parallelism: 1`)

Deploying Updates

```
# Update to new image version
docker service update --image your-registry/nginx:v2.0.0 nginx

# Watch the rollout in real-time
docker service ps nginx --watch

# Check update status
docker service inspect nginx --format '{{.UpdateStatus.State}}'
```

Manual Rollback

```
# Rollback to previous version
docker service rollback nginx

# Or update to a specific version
docker service update --image your-registry/nginx:v1.9.0 nginx
```

Automatic Rollback

With `failure_action: rollback` and `monitor: 10s` configured, if a new container fails health checks within 10 seconds of starting, Swarm automatically rolls back to the previous working version.

SCALING STRATEGIES

Manual Scaling

```
# Scale up
docker service scale nginx=4

# Scale down
docker service scale nginx=2

# Check current scale
docker service ls
```

Environment Variable Scaling

In compose:

```
deploy:
    replicas: ${NGINX_REPLICAS:-2}
```

Set in `.env`:

```
NGINX_REPLICAS=3
```

Scaling Considerations

REPLICAS	USE CASE	NOTES
1	Development/testing	No high availability
2	Production minimum	One can fail while other handles traffic
3+	High traffic production	Better load distribution

Scaling Limits with Placement Constraints

With `max_replicas_per_node: 1`, you can only scale to the number of nodes:

- 3 nodes = max 3 replicas
- Remove the constraint if you need more replicas than nodes

TROUBLESHOOTING

Service Won't Start

```
# Check service status with full error messages
docker service ps nginx --no-trunc

# Check recent logs
docker service logs nginx --tail 100

# Check a specific container
docker logs <container_id>
```

Network Issues

```
# Verify network exists
docker network ls | grep appnet

# Check if services are connected
docker network inspect appnet

# Test DNS resolution from inside container
docker exec <container_id> nslookup backend-service

# Test connectivity to upstream
docker exec <container_id> curl -v http://backend-service:8080/health
```

Container Keeps Restarting

```
# Check restart count and errors
docker service ps nginx

# Disable restarts temporarily for debugging
docker service update --restart-max-attempts 0 nginx

# Check logs for the actual error
docker service logs --tail 100 nginx

# Re-enable restarts after debugging
docker service update --restart-max-attempts 30 nginx
```

Secrets Not Available

```
# Verify secret exists
docker secret ls

# Check secret is attached to service
docker service inspect nginx --format '{{.Spec.TaskTemplate.ContainerSpec.Secre

# Verify inside container
docker exec <container_id> ls -la /run/secrets/

# Check secret content (from inside container only)
docker exec <container_id> cat /run/secrets/nginx_server_pem
```

Config Issues

```
# Test NGINX configuration syntax
docker exec <container_id> nginx -t

# Reload NGINX without restart
docker exec <container_id> nginx -s reload

# Check loaded modules
docker exec <container_id> nginx -V 2>&1 | grep modules
```

Useful Debug Commands

```
# Service overview
docker service ls

# Detailed service info
docker service inspect nginx --pretty

# Task/container info with errors
docker service ps nginx --no-trunc

# Real-time logs
docker service logs -f nginx

# Container resource usage
docker stats $(docker ps -q --filter name=nginx)

# Execute shell in running container
docker exec -it $(docker ps -q --filter name=nginx | head -1) /bin/bash
```

DATADOG INTEGRATION

This bonus section covers complete Datadog APM integration for distributed tracing, log correlation, and metrics.

Why Integrate Datadog?

- **Distributed tracing**: Follow requests from NGINX through your entire backend stack
- **Log correlation**: Click from any log entry directly to the full trace

- **Metrics**: Connection counts, request rates, response times
- **Alerting**: Get notified on error spikes, latency issues

The Datadog NGINX Module

The module adds APM tracing directly into NGINX, generating trace IDs that propagate to backend services.

```
ARG NGINX_VERSION=1.27.0

# Copy the pre-downloaded module
COPY ngx_http_datadog_module-amd64-$NGINX_VERSION.so.tgz /usr/lib/nginx/modules

RUN tar -xzf "/usr/lib/nginx/modules/ngx_http_datadog_module-amd64-$NGINX_VERSI
    -C "/usr/lib/nginx/modules" && \
    rm /usr/lib/nginx/modules/*.tgz
```

Load in nginx.conf:

```
load_module modules/ngx_http_datadog_module.so;

http {
    datadog_service_name "nginx";
    # ...
}
```

JSON Logs with Trace Correlation

The magic is in `$datadog_trace_id` and `$datadog_span_id` - these let you click from any log line directly to the full trace.

```
log_format json_log_datadog escape=json '{'
    '"timestamp":$msec,'
    '"trace_id": $datadog_trace_id,'
    '"span_id": $datadog_span_id,'
    '"http":{'
        '"method":"$request_method",'
        '"status_code":$status,'
        '"uri":"$request_uri",'
        '"useragent":"$http_user_agent"'
    '},'
    '"dd":{'
        '"trace_id": $datadog_trace_id,'
        '"span_id": $datadog_span_id,'
        '"service": "nginx",'
        '"source": "nginx"'
    '},'
    '"nginx":{'
        '"request_time":$request_time,'
        '"upstream_time":"$upstream_response_time",'
        '"upstream_addr":"$upstream_addr",'
        '"cache_status":"$upstream_cache_status"'
    '}'
'}';

access_log /dev/stdout json_log_datadog;
```

Autodiscovery Labels

Datadog's autodiscovery automatically configures integrations based on container labels:

```
labels:
    com.datadoghq.ad.check_names: '["nginx"]'
    com.datadoghq.ad.logs: >-
        [
            {"type":"docker","source":"nginx","service":"nginx"},
            {"type":"file","source":"modsecurity","service":"nginx-waf",
             "auto_multi_line_detection":true,"path":"/var/log/modsec_audit.jsc
        ]
    com.datadoghq.ad.init_configs: '[{}]'
    com.datadoghq.ad.instances: '[{"nginx_status_url":"http://localhost:81/ngir
```

Environment Variables

```
environment:
    - DD_AGENT_HOST=datadog-agent
    - DD_TRACE_AGENT_PORT=8126
    - DD_TRACE_ENABLED=true
    - DD_PROFILING_ENABLED=true
    - DD_APPSEC_ENABLED=true
    - DD_LOGS_CONFIG_AUTO_MULTI_LINE_DETECTION=true
```

Datadog Agent in Swarm

```
services:
    datadog-agent:
        image: datadog/agent:latest
        environment:
            - DD_API_KEY=${DD_API_KEY}
            - DD_SITE=datadoghq.com
            - DD_APM_ENABLED=true
            - DD_APM_NON_LOCAL_TRAFFIC=true
            - DD_LOGS_ENABLED=true
            - DD_LOGS_CONFIG_CONTAINER_COLLECT_ALL=true
        volumes:
            - /var/run/docker.sock:/var/run/docker.sock:ro
            - /proc/:/host/proc/:ro
            - /sys/fs/cgroup/:/host/sys/fs/cgroup:ro
            - /docker/swarm/nginx:/docker/swarm/nginx:ro
        deploy:
            mode: global  # One agent per node
        networks:
            appnet:
```

Tracing Flow

```
User Request (no trace headers)
              │
              ▼
┌─────────────────────────────────────┐
│            NGINX                     │
│  • Generates trace_id: 1234567890   │
│  • Generates span_id: 9876543210    │
│  • Adds x-datadog-trace-id header   │
│  • Logs include trace_id            │
└─────────────┬───────────────────────┘
              │
              ▼
┌─────────────────────────────────────┐
│         Backend Service             │
│  • Reads x-datadog-trace-id header  │
│  • Continues same trace             │
│  • Creates child span               │
└─────────────┬───────────────────────┘
              │
              ▼
┌─────────────────────────────────────┐
│         Datadog APM UI              │
│  • Single trace shows full path     │
│  • NGINX span → Backend span        │
│  • Click log → see trace            │
└─────────────────────────────────────┘
```

Troubleshooting Datadog

```
# Check module is loaded
docker exec <container> nginx -V 2>&1 | grep datadog

# Test agent connectivity
docker exec <container> curl -v http://datadog-agent:8126/info

# Verify trace IDs in logs
docker logs <container> 2>&1 | head -1 | jq '.trace_id'

# Check autodiscovery
docker exec <datadog-agent> agent configcheck
```

COMPLETE CONFIGURATION REFERENCE

Environment File (.env)

```
# Build
BUILD_VERSION=1.0.0

# Swarm
NGINX_REPLICAS=2
NGINX_RESOLVER=127.0.0.11

# Datadog (optional)
DD_API_KEY=your-api-key
DD_SITE=datadoghq.com
```

Deployment Commands

```
# Build and push
./build.sh 1.0.0

# Deploy stack
docker stack deploy -c docker-compose.yaml mystack

# View status
docker stack ps mystack
docker stack services mystack

# Remove stack
docker stack rm mystack
```

Quick Reference

```
# Stack operations
docker stack deploy -c docker-compose.yaml mystack
docker stack ps mystack
docker stack services mystack
docker stack rm mystack

# Service operations
docker service ls
docker service ps nginx
docker service logs -f nginx
docker service scale nginx=3
docker service update --image registry/nginx:v2 nginx
docker service rollback nginx

# Secret operations
docker secret create <name> <file>
docker secret ls
docker secret rm <name>

# Config operations
docker config create <name> <file>
docker config ls
docker config rm <name>

# Network operations
docker network create --opt encrypted --driver overlay --attachable appnet
docker network ls
docker network inspect appnet

# Debug
docker exec -it <container> /bin/bash
docker exec <container> nginx -t
docker stats
```

CHECKLIST

Pre-Deployment

- [ ] Overlay network created with `--opt encrypted`
- [ ] SSL certificates ready
- [ ] Secrets created (`docker secret create`)

- [ ] Image built and pushed to registry
- [ ] Environment variables set in `.env`

NGINX Configuration

- [ ] `resolver 127.0.0.11` set in nginx.conf
- [ ] Upstreams use service names (NOT hardcoded IPs)
- [ ] Health check endpoints configured (ports 81, 82)
- [ ] JSON logging format configured
- [ ] ModSecurity enabled and configured

Compose Configuration

- [ ] `init: true` for proper signal handling
- [ ] `max_replicas_per_node: 1` for HA distribution
- [ ] Resource limits AND reservations set
- [ ] `failure_action: rollback` configured
- [ ] `parallelism: 1` for safe rolling updates
- [ ] Secrets mounted correctly

Datadog (if using)

- [ ] Module loaded in nginx.conf
- [ ] Autodiscovery labels configured
- [ ] JSON log format includes `$datadog_trace_id`
- [ ] `DD_AGENT_HOST` points to agent service
- [ ] stub_status endpoint accessible on port 81

Post-Deployment

- [ ] Service running with correct replica count
- [ ] Health checks passing
- [ ] Logs appearing in aggregation tool
- [ ] SSL/TLS working correctly

- [ ] Upstream services reachable
- [ ] **Test a rollback!**

CONCLUSION

This setup has maintained 100% uptime in production for years. The key takeaways:

1. **Always encrypt your overlay network** - `--opt encrypted` is non-negotiable
2. **Never hardcode IPs** - Use service names and Docker DNS
3. **Use secrets for sensitive data** - SSL certs, API keys, passwords
4. **Set proper health checks** - Swarm needs to know if containers are healthy
5. **Configure rolling updates** - Zero-downtime deployments are the goal
6. **Set resource limits** - Protect your nodes from runaway containers
7. **Test your rollbacks** - Know they work before you need them

The extra effort in configuration pays off when you're sleeping soundly instead of getting 3am alerts.

This guide was created by a human with 10 years of production experience, with help from Claude AI to organize and document everything properly.