THREADSYNC — MAGIC RUNTIME

Magic Runtime — Operations Runbook

February 2026 · v2.0 · magic.threadsync.io

Operations Runbook

Production operations procedures for Magic Runtime. Go-live, on-call, rollback, monitoring, and disaster recovery.

Markdown version

Deploy Center

Go-Live Checklist

Pre-deployment

- ☐ Create production `.env` from `.env.example`
- ☐ Generate strong secrets (32+ chars) for `JWT_SECRET` and `ADMIN_API_KEY`
- ☐ Set exact `ALLOWED_ORIGINS` (no wildcards)
- ☐ Configure tight `CONTROLLER_ALLOWLIST`
- ☐ Set `ENVIRONMENT=prod`
- ☐ Configure TLS termination (load balancer, Cloudflare, or Nginx+certbot)

Generate secrets

```
 openssl rand -hex 32  # For JWT_SECRET
openssl rand -hex 32  # For ADMIN_API_KEY
```

Deployment Steps

1. Download and verify

```
 wget https://magic.threadsync.io/deploy/magic-runtime-2.0.0-
production.tar.gz
wget https://magic.threadsync.io/deploy/SHA256SUMS
wget https://magic.threadsync.io/deploy/SHA256SUMS.sig
sha256sum -c SHA256SUMS --ignore-missing
gpg --verify SHA256SUMS.sig SHA256SUMS
```

2. Extract and configure

```
 tar -xzf magic-runtime-2.0.0-production.tar.gz
cd magic-runtime
cp .env.example .env
chmod 600 .env
# Edit .env with production values
```

3. Deploy stack

```
 docker compose up -d --build
./scripts/init_db.sh
```

4. Verify deployment

```
 curl -s http://localhost/api/health | jq .
curl -s http://localhost/api/readyz | jq .

# Test controller execution
curl -s -X POST http://localhost/api/execute/DemoController \
  -H 'Content-Type: application/json' \
  -H "X-API-Key: $ADMIN_API_KEY" \
  -d '{"message": "Hello, Magic!"}' | jq .
```

On-Call Checklist

Triage (first 2 minutes)

1. Check site: `curl -sI https://your-domain.com/api/health`
2. Check containers: `docker ps -a --format "table {{.Names}}\t{{.Status}}"`
3. Check recent logs: `docker logs magic-api --since 5m --tail 50`
4. Classify severity (see table below)

SEVERITY	CRITERIA	RESPONSE	ESCALATE AFTER
P1	Site down, all requests failing	Immediate	15 min
P2	Error rate >5% or latency >5s	15 min	1 hour
P3	Resource threshold breached	Next business day	-

All-in-one status check

```
 echo "=== Containers ===" && docker ps --format "{{.Names}}: {{.Status}}"
&& \
echo "=== API Health ===" && curl -s http://localhost/api/health | jq . &&
\
echo "=== Readiness ===" && curl -s http://localhost/api/readyz | jq . && \
echo "=== Disk ===" && df -h / && \
echo "=== Memory ===" && free -h && \
echo "=== CPU ===" && uptime
```

Rollback Procedure

Quick rollback (< 2 minutes)

Application rollback

```
cd /opt/magic-runtime

# Stop current containers
docker compose down

# Restore previous docker-compose.yml (if modified)
cp docker-compose.yml.bak docker-compose.yml

# Start previous version
docker compose up -d

# Verify
curl -s http://localhost/api/health | jq .
```

Controller rollback

Roll back a specific controller

```
# Via Magic CLI
docker compose exec runtime magic rollback <ControllerName> --to <previous-version>

# Verify the rollback
docker compose exec runtime magic status <ControllerName>
```

Full rollback (with database)

Database rollback is destructive

Only use full rollback if the deployment included database migrations that caused issues. This restores from backup and may lose data written after the backup.

Full rollback with database restore

```
 # Stop services
docker compose down

# Restore database from backup
./scripts/restore.sh backups/magic_backup_YYYYMMDD_HHMMSS.sql

# Restore previous application version
cd /opt && tar -xzf magic-runtime-<previous-version>.tar.gz
cd magic-runtime

# Start previous version
docker compose up -d

# Verify
curl -s http://localhost/api/health | jq .
curl -s http://localhost/api/readyz | jq .
```

Rollback checklist

- ☐ Identify what changed (controller? config? migration?)
- ☐ Choose rollback scope (controller-only, app-only, or full+DB)
- ☐ Execute rollback steps above
- ☐ Verify health + readiness endpoints
- ☐ Verify existing controllers still execute

- ☐ Notify team with incident summary
- ☐ Create post-incident ticket

Monitoring & Alerts

Key Metrics

METRIC	WARNING	CRITICAL
API Down	-	2 min
Error Rate	> 5%	> 10%
Response Time (p95)	> 2s	> 5s
CPU Usage	> 70%	> 90%
Memory Usage	> 70%	> 90%
Disk Usage	> 70%	> 85%

Incident Response

API Down

1. Check container status: `docker ps -a`
2. Check logs: `docker logs magic-api --tail 100`
3. Check database: `docker exec magic-api python -c "from app.db.session import engine; engine.connect()"`
4. Restart: `docker compose down && docker compose up -d`

High Error Rate

1. Check recent errors: `docker logs magic-api --since 10m | grep ERROR`
2. Identify controller: `docker logs magic-api --since 10m | grep '"level":"error"' | jq '.controller'`
3. Disable if needed: Remove from `CONTROLLER_ALLOWLIST`, then `docker compose restart`

Security Procedures

Audit Log Queries

Security audit commands

```
 # Check capability violations
docker logs magic-api | grep '"event":"capability_denied"'

# Check auth failures
docker logs magic-api | grep '"event":"auth_failed"'

# Check admin actions
docker logs magic-api | grep '"event":"admin_action"'
```

Rotate Secrets

1. Generate new secrets: `openssl rand -hex 32`
2. Update `.env`
3. Restart: `docker compose restart`
4. Update any external API key references

Disaster Recovery

TARGET	VALUE
RTO (Recovery Time)	2 hours
RPO (Recovery Point)	24 hours (daily backups)

Restore to New Server

1. Provision new server (4GB+ RAM, Docker installed)
2. Copy backup + application tarball
3. Extract, configure `.env`, deploy
4. Restore database: `./scripts/restore.sh backups/<file>.sql`
5. Update DNS / load balancer target
6. Verify all endpoints

Capacity Planning

METRIC	SINGLE SERVER LIMIT
Concurrent requests	~100-200
Controllers/sec	~20-50 (process sandbox)
Database connections	100 (default pool)

Scaling Options

1. **Vertical:** Upgrade server (4GB → 8GB → 16GB)
2. **Horizontal:** Add runtime replicas behind load balancer (runtime is stateless)
3. **Database:** Read replicas, connection pooling (PgBouncer)
4. **Caching:** Redis for controller result caching