Production Deployment Guide

Shark Knowledge Base System

Database: Memgraph 2.14

Generated: 2026-01-07 10:45:00

Status: Requires Phase 12 Optimization Before Deployment

Table of Contents

---

1. Prerequisites

Required Skills

- **Database Administration:** Experience with graph databases (Cypher query language)
- **DevOps:** Linux system administration, Docker/containers
- **Networking:** Firewall configuration, security groups
- **Application Deployment:** Python/Rust application deployment
- **Performance Tuning:** Redis caching, query optimization

Required Access

- Cloud provider account (AWS/Azure/GCP) or on-premises infrastructure
- SSH access to servers
- Database administration credentials
- Firewall/security group modification permissions

Required Software (Local Machine)

- SSH client
- Database client tools (mgconsole)
- Git
- Docker (for local testing)
- Redis CLI (for cache management)

2. Infrastructure Requirements

Database Server

Hardware Specifications:

- **CPU:** 8+ cores @ 2.4GHz (16 cores recommended for production)
- **RAM:** 16GB minimum (32GB recommended for headroom)
- **Storage:** 500GB SSD (NVMe preferred)
- **Network:** 1Gbps+ connectivity

Operating System:

- Ubuntu 22.04 LTS (recommended)
- RHEL 9 (alternative)
- Debian 12 (alternative)

Application Server

Hardware Specifications:

- **CPU:** 4+ cores
- **RAM:** 8GB minimum (16GB recommended)
- **Storage:** 100GB SSD
- **Network:** 1Gbps+ connectivity

Operating System:

- Ubuntu 22.04 LTS (recommended)

Redis Cache Server (Phase 12 - REQUIRED)

Hardware Specifications:

- **CPU:** 2+ cores
- **RAM:** 4GB minimum (8GB recommended)
- **Storage:** 50GB SSD

Purpose: Meet 10ms identifier lookup threshold

Network Configuration

Firewall Rules:

- Bolt Protocol: Port 7687 (from application server only)
- Application API: Port 8080 (from load balancer/users)
- Redis: Port 6379 (from application server only)
- SSH: Port 22 (from admin IPs only)
- Monitoring: Ports 9090, 3000 (Prometheus, Grafana)

---

3. Phase 12: Optimization (REQUIRED)

Critical: Cannot Deploy Without This Phase

Benchmark Results Show: - Identifier lookups: 138ms p99 vs 10ms target (14× too slow) - High concurrency: 0/4 queries pass at 50-100 users - Overall: 32.1% test pass rate

Status: All databases require optimization before production

Week 1-3: Optimization Implementation

Step 1: Redis Caching Layer (Week 1)   **Install Redis:**

```bash
# On Redis server
sudo apt update
sudo apt install -y redis-server

# Configure Redis for production
sudo nano /etc/redis/redis.conf
```

Redis Configuration:

```
# Bind to application server only (use private IP)
bind 10.0.1.100

# Memory management
maxmemory 4gb
maxmemory-policy allkeys-lru

# Persistence (RDB snapshots)
save 900 1
save 300 10
save 60 10000

# Performance tuning
tcp-backlog 511
timeout 0
tcp-keepalive 300
```

Start Redis:

```bash
sudo systemctl enable redis-server
sudo systemctl restart redis-server
sudo systemctl status redis-server
```

Verify:

```bash
redis-cli ping  # Should return "PONG"
redis-cli INFO memory
```

Step 2: Application Cache Integration (Week 1)   **Update API to use Redis:**

Add caching layer for identifier lookups (mode_s, mmsi):

```python
# Python example
import redis
import json

redis_client = redis.Redis(host='redis-server', port=6379, db=0)
CACHE_TTL = 300  # 5 minutes

def get_aircraft_cached(mode_s):
    # Check cache first
    cache_key = f"aircraft:mode_s:{mode_s}"
    cached = redis_client.get(cache_key)

    if cached:
        return json.loads(cached)
    # Cache miss - query database
    result = query_database(mode_s)

    # Store in cache
    redis_client.setex(cache_key, CACHE_TTL, json.dumps(result))

    return result
```

Expected Impact: Reduce identifier lookups from 138ms to <10ms (14× faster)

Step 3: Index Optimization (Week 2)

Memgraph Index Creation:

```
// Create indexes for identifier lookups
CREATE INDEX ON :Aircraft(mode_s);
CREATE INDEX ON :Ship(mmsi);
CREATE INDEX ON :GroundUnit(unit_id);

// Create indexes for common queries
CREATE INDEX ON :Aircraft(nationality);
CREATE INDEX ON :Aircraft(affiliation);

// Verify indexes
SHOW INDEX INFO;
```

Expected Impact: 10-15% latency reduction on cache misses

Step 4: Connection Pooling (Week 2)

Configure connection pooling for high concurrency:

```
from neo4j import GraphDatabase

driver = GraphDatabase.driver(
    "bolt://memgraph:7687",
    auth=None,
    max_connection_pool_size=100,  # Increase from default 10
    connection_acquisition_timeout=60.0
)
```

Expected Impact: Support 100+ concurrent users

Step 5: Query Optimization (Week 2-3)

Review slow queries:

```
// Before: Slow traversal query
MATCH (a:Aircraft {nationality: $country})
OPTIONAL MATCH (a)-[r]-()
RETURN a, count(r) as rel_count

// After: Optimized with LIMIT and index hint
MATCH (a:Aircraft)
WHERE a.nationality = $country
WITH a LIMIT 100
OPTIONAL MATCH (a)-[r]-()
RETURN a, count(r) as rel_count
```

Expected Impact: 20-30% faster traversal queries

Step 6: Validation Testing (Week 3) **Re-run benchmark suite:**

```
cd benchmark/harness

# Test identifier lookups with cache
python3 runner.py http://app-server:8080 \
  --pattern lookup-95 \
  --requests 10000 \
  --concurrency 20 \
  --output phase12_validation

# Verify p99 < 10ms for cached queries
# Target: >90% cache hit rate
```

Success Criteria: - Identifier lookups p99 < 10ms (cached) - Identifier lookups p99 < 100ms (cache miss) - Support 100 concurrent users - Overall test pass rate > 80%

---

4. Database Installation

Memgraph 2.14 Installation

Step 1: Download Memgraph

```
wget https://download.memgraph.com/memgraph/v2.14.0/ubuntu-22.04/memgraph_2.14.0-1_amd64.deb
```

Step 2: Install Memgraph

```
sudo dpkg -i memgraph_2.14.0-1_amd64.deb
```

Step 3: Start Memgraph

```
sudo systemctl enable memgraph
sudo systemctl start memgraph
sudo systemctl status memgraph
```

Step 4: Verify installation

```
# Connect using mgconsole
sudo apt install -y mgconsole
mgconsole --host 127.0.0.1 --port 7687
```

---

5. Database Configuration

Apply Optimized Memgraph Configuration

Configuration file location: `/etc/memgraph/memgraph.conf`

Optimized settings (from Phase A + Phase 12):

```
--memory-limit=12GB
--memory-warning-threshold=10GB
--query-plan-cache-size=10000
--bolt-num-workers=16
--bolt-port=7687
--log-level=WARNING

# Phase 12 optimizations
--query-execution-timeout-sec=30
```

```
--storage-snapshot-interval-sec=300
--storage-wal-enabled=true
```

Restart database to apply configuration:

```
sudo systemctl restart memgraph
```

---

6. Dataset Loading

Generate Dataset

On your local machine (or jump box):

```
cd data/generators

# For testing: 5,560 entities
python3 generate_air_data.py --count 4000 --output air_instances.csv
python3 generate_surface_data.py --count 1000 --output surface_instances.csv
python3 generate_ground_data.py --count 560 --output ground_instances.csv

# For production: 200,000 entities
python3 generate_air_data.py --count 140000 --output air_instances.csv
python3 generate_surface_data.py --count 50000 --output surface_instances.csv
python3 generate_ground_data.py --count 10000 --output ground_instances.csv
```

Load Dataset

Transfer data files to database server:

```
scp *.csv user@database-server:/tmp/
```

Load data into Memgraph:

```
cd data/loaders

python3 load_memgraph.py \
  --uri bolt://database-server:7687 \
  --air-file /tmp/air_instances.csv \
  --surface-file /tmp/surface_instances.csv \
  --ground-file /tmp/ground_instances.csv
```

Expected load time: - 5,560 entities: 30-60 seconds - 200,000 entities: 5-10 minutes

Create Indexes (Phase 12)

```
mgconsole --host database-server --port 7687

# Run index creation commands
CREATE INDEX ON :Aircraft(mode_s);
CREATE INDEX ON :Ship(mmsi);
CREATE INDEX ON :GroundUnit(unit_id);
CREATE INDEX ON :Aircraft(nationality);
CREATE INDEX ON :Aircraft(affiliation);
```

Verify Data Load

Check entity counts:

```
MATCH (a:Aircraft) RETURN count(a);  // Should return 4000 or 140000
MATCH (s:Ship) RETURN count(s);      // Should return 1000 or 50000
MATCH (g:GroundUnit) RETURN count(g); // Should return 560 or 10000
```

---

7. Application Deployment

Install Python (if using Python API)

On application server:

```
sudo apt update
sudo apt install -y python3 python3-pip python3-venv
```

Clone Repository

```
git clone https://github.com/your-org/shark-bakeoff.git
cd shark-bakeoff/implementations/python
```

Configure Application

Create .env file:

```
cat > .env <<EOF
# Memgraph connection
MEMGRAPH_URI=bolt://database-server:7687

# Redis cache (Phase 12)
REDIS_URL=redis://redis-server:6379
CACHE_TTL_SECONDS=300

# Kafka (optional)
KAFKA_BROKERS=kafka-server:9092

# Application settings
LOG_LEVEL=info
HOST=0.0.0.0
PORT=8080
EOF
```

Install Dependencies

```
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

Create Systemd Service

Create service file:

```
sudo tee /etc/systemd/system/shark-api.service > /dev/null <<EOF
[Unit]
Description=Shark Knowledge Base API
After=network.target

[Service]
Type=simple
```

```
User=shark
WorkingDirectory=/home/shark/shark-bakeoff/implementations/python
EnvironmentFile=/home/shark/shark-bakeoff/implementations/python/.env
ExecStart=/home/shark/shark-bakeoff/implementations/python/venv/bin/python api_memgraph.py
Restart=on-failure
RestartSec=5s

[Install]
WantedBy=multi-user.target
EOF
```

Start Application

```
sudo systemctl daemon-reload
sudo systemctl enable shark-api
sudo systemctl start shark-api
sudo systemctl status shark-api
```

Verify Application

```
# Test health endpoint
curl http://localhost:8080/health

# Test cached query (should be fast after first query)
time curl http://localhost:8080/api/aircraft/mode_s/A12345
time curl http://localhost:8080/api/aircraft/mode_s/A12345  # Should be <10ms
```

---

8. Caching Setup (Phase 12)

Redis Cache Warmup

Pre-populate cache with common queries:

```
# warmup_cache.py
import redis
import requests
import json

redis_client = redis.Redis(host='redis-server', port=6379)

# Load common identifiers from database
common_aircraft = get_most_queried_aircraft()  # Top 1000

for aircraft in common_aircraft:
    # Fetch from API (which populates cache)
    requests.get(f"http://localhost:8080/api/aircraft/mode_s/{aircraft['mode_s']}")

print(f"Cache warmed up with {len(common_aircraft)} entries")
```

Run warmup:

```
python3 warmup_cache.py
```

Monitor Cache Performance

```
# Check cache hit rate
redis-cli INFO stats | grep keyspace_hits
redis-cli INFO stats | grep keyspace_misses

# Monitor cache size
redis-cli INFO memory | grep used_memory_human
```

Target Metrics: - Cache hit rate: >70% (>90% ideal) - Used memory: <3GB - Evictions: <100/minute

---

9. Monitoring & Alerting

Install Prometheus

```
# Download Prometheus
wget https://github.com/prometheus/prometheus/releases/download/v2.45.0/prometheus-2.45.0.linux-amd64.ta
tar xvfz prometheus-*.tar.gz
sudo mv prometheus-2.45.0.linux-amd64 /opt/prometheus
```

Configure Prometheus

Create `/opt/prometheus/prometheus.yml`:

```
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'shark-api'
    static_configs:
      - targets: ['localhost:8080']

  - job_name: 'redis'
    static_configs:
      - targets: ['redis-server:9121']  # redis_exporter
```

Install Grafana

```
sudo apt install -y grafana
sudo systemctl enable grafana-server
sudo systemctl start grafana-server
```

Configure Alerts

Critical Alerts:

- **Identifier lookups p99 >100ms** (cache misses)
- **Cache hit rate <70%**
- **Error rate >1%**
- **Database connection failures**
- **API unavailable**

Warning Alerts:

- **Identifier lookups p99 >50ms** (cached)
- **Traversals p99 >200ms**
- **Cache hit rate <90%**

- **Memory usage >80%**
- **Disk usage >80%**

---

10. Backup & Disaster Recovery

Backup Strategy

Frequency: - Full backup: Daily at 2 AM - Incremental backup: Every 6 hours

Retention: - Daily backups: 30 days - Weekly backups: 90 days - Monthly backups: 1 year

Memgraph Backup Script

Create `/usr/local/bin/backup-memgraph.sh`:

```
#!/bin/bash
BACKUP_DIR=/backups/memgraph
DATE=$(date +%Y%m%d_%H%M%S)

# Create snapshot via mgconsole
echo 'CREATE SNAPSHOT;' | mgconsole --host 127.0.0.1

# Backup snapshot directory
tar -czf $BACKUP_DIR/memgraph_$DATE.tar.gz /var/lib/memgraph/snapshots

# Upload to S3
aws s3 cp $BACKUP_DIR/memgraph_$DATE.tar.gz s3://backups/shark/

# Cleanup
find $BACKUP_DIR -name '*.tar.gz' -mtime +30 -delete
```

Schedule Backups

Add to crontab:

```
0 2 * * * /usr/local/bin/backup-memgraph.sh
```

Disaster Recovery

Recovery Time Objective (RTO): 4 hours

Recovery Point Objective (RPO): 1 hour

Recovery Steps: 1. Provision new database server (if needed) 2. Install Memgraph 3. Download latest backup from S3 4. Restore backup 5. Recreate indexes 6. Update application configuration 7. Verify data integrity 8. Resume operations

---

11. Performance Validation

Phase 12 Validation (After Optimization)

Post-optimization benchmark:

```
cd benchmark/harness

# Run comprehensive benchmark
```

```
python3 runner.py http://app-server:8080 \
  --pattern lookup-95 \
  --requests 10000 \
  --concurrency 20 \
  --output post_phase12
```

Expected Results (with Phase 12 optimizations):

Query Type	Target	Pre-Phase 12	Post-Phase 12
Identifier p99 (cached)	<10ms	138ms	<10ms
Identifier p99 (miss)	<100ms	138ms	<80ms
Traversal p99	<300ms	144ms	<120ms
100 concurrent users	Pass 4/4	Fail 0/4	Pass 4/4

Load Test (Production Scale)

```
python3 runner.py http://app-server:8080 \
  --pattern balanced-50 \
  --requests 100000 \
  --concurrency 50 \
  --output production_load_test
```

Stress Test (High Concurrency)

```
# Test at 100 concurrent users
python3 runner.py http://app-server:8080 \
  --pattern balanced-50 \
  --requests 10000 \
  --concurrency 100
```

---

12. Curation Tools

Memgraph Lab

Install Memgraph Lab:

```
# Download Lab
wget https://download.memgraph.com/memgraph-lab/v2.14.0/memgraph-lab-2.14.0-linux-x86_64.AppImage

# Make executable
chmod +x memgraph-lab-2.14.0-linux-x86_64.AppImage

# Run Lab
./memgraph-lab-2.14.0-linux-x86_64.AppImage
```

Connect to Memgraph: - Host: `database-server` - Port: `7687` - No authentication required (Community Edition)

Curator Training

Week 8-9: Train curators on:

1. Graph visualization and exploration
2. Adding properties to entities
3. Creating relationships

4. Running queries
5. Exporting data
6. Schema evolution (self-service!)

---

13. Rollback Plan

When to Rollback

Rollback if any of the following occur within 48 hours of go-live:

- p99 latency exceeds thresholds by >10%
- Cache hit rate <50%
- Error rate >1%
- Data corruption detected
- Critical application bugs

Rollback Procedure

Step 1: Stop new traffic

```
# Update load balancer to redirect to old system
# Or stop Shark API
sudo systemctl stop shark-api
```

Step 2: Restore previous system

- Restore old database from backup
- Restore old application version
- Verify data integrity

Step 3: Validate rollback

```
# Run smoke test on old system
curl http://old-system:8080/health
```

Step 4: Resume traffic

- Update load balancer to old system
- Monitor for 1 hour

Step 5: Post-mortem

- Document rollback reason
- Identify root cause
- Create mitigation plan
- Schedule retry

---

14. Go-Live Checklist

Pre-Launch (T-24 hours)

☐ Phase 12 optimization complete and validated
☐ Memgraph optimized and running
☐ 5,560 entities loaded and verified (or 200,000 for production)
☐ Indexes created and verified
☐ Application deployed and tested
☐ Redis cache configured and warmed up

□ Cache hit rate >70% verified
□ Monitoring and alerting active
□ Backups configured and tested
□ Post-Phase 12 benchmarks passed (>80% pass rate)
□ Rollback plan documented
□ Stakeholders notified

Launch Day (Week 10)

Phase 1: 10% Traffic (Hour 0-4)

□ Route 10% of traffic to new system
□ Monitor p99 latency every 15 minutes
□ Monitor cache hit rate (target >70%)
□ Monitor error rate
□ Verify all metrics within thresholds

Phase 2: 50% Traffic (Hour 4-8)

□ Increase to 50% traffic
□ Continue monitoring
□ Verify no degradation
□ Check cache performance

Phase 3: 100% Traffic (Hour 8+)

□ Route 100% traffic to new system
□ Intensive monitoring for 4 hours
□ Verify all thresholds met
□ Collect curator feedback

Post-Launch (Day 1-2)

□ 24-hour stability monitoring
□ Daily performance reports
□ Curator feedback sessions
□ Issue tracking and resolution
□ Fine-tune cache TTLs
□ Verify cache hit rate >90%

---

15. Post-Deployment

Month 1: Intensive Monitoring

Daily Activities: - Review p99 latency metrics - Check cache hit rate (target >90%) - Check error logs - Monitor resource usage (CPU, memory, disk) - Collect curator feedback

Weekly Activities: - Performance report to stakeholders - Issue review and prioritization - Cache effectiveness analysis - Optimization opportunities

Month 2-3: Optimization

Fine-Tuning: - Adjust cache TTLs based on real traffic patterns - Optimize slow queries identified in production - Update database configuration if needed - Scale Redis if cache hit rate <90%

Training: - Advanced curator training sessions - Best practices documentation

Month 6: Review

Validation: - Verify database choice still correct - Review dataset growth trends - Assess if scaling needed - Document production learnings

Planning: - Forecast next 12 months growth - Plan infrastructure scaling if needed - Monitor RAM usage (Memgraph limitation) - Evaluate read replicas if needed

Year 1: Long-Term Planning

Growth Planning: - Dataset size projection - Infrastructure scaling plan - Migration plan if approaching RAM limit - Budget for next year

---

16. Troubleshooting

Common Issues

Issue: High Latency (Post-Phase 12) **Symptoms:** p99 >10ms for cached identifier lookups

Possible Causes: - Low cache hit rate - Redis overloaded - Network latency

Solutions: 1. Check cache hit rate: `redis-cli INFO stats` 2. Increase cache TTL if hit rate <70% 3. Verify Redis memory: `redis-cli INFO memory` 4. Check network latency with `ping` 5. Scale Redis if needed

Issue: Low Cache Hit Rate **Symptoms:** Cache hit rate <70%

Solutions: 1. Increase cache TTL (current: 300s) 2. Warm up cache with common queries 3. Check query diversity (high diversity = low hit rate) 4. Consider caching more query types 5. Pre-populate cache at startup

Issue: Database Connection Failures **Symptoms:** Application cannot connect to Memgraph

Solutions: 1. Verify Memgraph is running: `systemctl status memgraph` 2. Check firewall rules 3. Verify connection string in `.env` 4. Check Memgraph logs: `/var/log/memgraph/memgraph.log` 5. Increase connection pool size if needed

Issue: Out of Memory (Memgraph) **Symptoms:** Memgraph crashes, OOM errors

Solutions: 1. Check dataset size vs RAM 2. Increase server RAM 3. Archive old data 4. Consider migration to Neo4j (disk-based)

Issue: High Concurrency Failures **Symptoms:** Queries fail at 50-100 concurrent users

Solutions: 1. Increase connection pool size 2. Add read replicas 3. Implement request queuing 4. Scale horizontally (load balancer)

Support Resources

Database-Specific: - Memgraph Documentation: https://memgraph.com/docs/ - Memgraph Discord Community

Project-Specific: - Implementation Team - Database Administrator - DevOps Engineer

Benchmark Results: - `/tmp/bakeoff-results/detailed_analysis.json` - `/tmp/bakeoff-results/comprehensive_res`

End of Production Deployment Guide

CRITICAL REMINDER: Phase 12 optimization is **REQUIRED** before production deployment. Do not skip this phase.