Redis Self-Hosted Guide: Redis Cloud → Self-Hosted

Project Information

Field	Value
Project	Weavink - NFC Business Card Platform
Author	Leo (CTO, Weavink)
Date	November 30, 2025
Server	Hetzner CX43 (8 vCPU, 16GB RAM, 160GB SSD)
Deployment Platform	Coolify
Redis Version	7 Alpine

Table of Contents

---

1. Executive Summary

What We Did

Deployed a self-hosted Redis instance on our Hetzner VPS via Coolify to replace Redis Cloud for caching and session management.

Final Result

- **0.3ms average latency** (self-hosted) vs **10-50ms** (Redis Cloud) **- 30-150x faster!**
- **€0/month** vs €5-10/month on Redis Cloud

- **81,967 SET operations/second** throughput
- Full control over configuration and data

Key Benefit

Redis on the same server as your app eliminates network round-trip, resulting in sub-millisecond latency.

---

2. Why Self-Host Redis?

Redis Cloud vs Self-Hosted Comparison

Aspect	Redis Cloud	Self-Hosted
Cost	€5-25/month	€0 (included in VPS)
Latency	10-50ms	**0.3ms**
Throughput	Limited by plan	**80K+ ops/sec**
Connection Limits	Plan-dependent	Unlimited
Data Location	Cloud provider	Your server (GDPR)
Maintenance	None	Minimal

When to Self-Host

- Running other services on the same VPS
- Need sub-millisecond latency
- Want to reduce costs
- Require data sovereignty (GDPR)

When to Keep Redis Cloud

- No VPS available
- Need managed backups and failover
- Multi-region requirements

---

3. Infrastructure Setup

Server Specifications

```
Provider: Hetzner
Model: CX43
vCPU: 8
RAM: 16GB
```

Storage: 160GB SSD

Location: Falkenstein, Germany (EU)

Cost: €8.99/month

Redis Resource Allocation

Max Memory: 2GB

Eviction Policy: allkeys-lru

Persistence: RDB snapshots (default)

Docker Compose Configuration (Coolify)

yaml

```yaml
services:
  redis:
    image: 'redis:7-alpine'
    restart: unless-stopped
    command: redis-server --maxmemory 2gb --maxmemory-policy allkeys-lru
    volumes:
      - 'redis-data:/data'
    healthcheck:
      test:
        - CMD
        - redis-cli
        - ping
      interval: 10s
      timeout: 5s
      retries: 5
volumes:
  redis-data: null
```

Container Details

Container Name: redis-hgw008ssw0ssc4kcoks40osk

Volume: hgw008ssw0ssc4kcoks40osk_redis-data

Internal Port: 6379

Network: Coolify internal network

Connection URL

```
redis://redis-hgw008ssw0ssc4kcoks40osk:6379
```

> **Security Note**: Redis is only accessible within Docker's internal network. No public exposure needed - your app connects via the internal hostname.

---

4. Performance Benchmarks

Test Environment

- **Server**: Hetzner CX43 (8 vCPU, 16GB RAM)
- **Redis**: 7 Alpine with 2GB max memory
- **Test**: redis-benchmark with 10,000 operations

Latency Results

Metric	Value
Minimum	**0ms**
Maximum	**2ms**
Average	**0.30ms**

Throughput Results

Operation	Requests/sec	p50 Latency
PING_INLINE	59,171	0.415ms
PING_MBULK	58,823	0.375ms
SET	**81,967**	0.343ms
GET	69,444	0.415ms

Comparison with Redis Cloud

Metric	Redis Cloud	Self-Hosted	Improvement
Latency	10-50ms	**0.3ms**	**30-150x faster**
SET ops/sec	~1,000-5,000	**81,967**	**16-80x faster**
GET ops/sec	~1,000-5,000	**69,444**	**14-70x faster**

Why Self-Hosted is Faster

1. **Zero network latency**: Same server as app
2. **No TLS overhead**: Internal network doesn't need encryption
3. **No multi-tenant contention**: Dedicated resources
4. **Direct memory access**: No proxy layers

5. Migration Guide

Step 1: Deploy Redis in Coolify

1. Go to **Coolify** → Your Project → **+ Add Resource**
2. Select **Docker Compose**
3. Paste the docker-compose configuration from Section 3
4. Name it `weavink-redis`
5. Click **Save** → **Deploy**

Step 2: Verify Deployment

```bash
# SSH to server
ssh root@159.69.215.143

# Check container is running
docker ps | grep redis

# Test connectivity
docker exec redis-hgw008ssw0ssc4kcoks40osk redis-cli ping
# Should return: PONG
```

Step 3: Update Weavink Environment Variables

In Coolify, update your Weavink app's environment variables:

Before (Redis Cloud):

```env
REDIS_URL=redis://default:PASSWORD@redis-xxxxx.cloud.redislabs.com:12345
```

After (Self-hosted):

```env
REDIS_URL=redis://redis-hgw008ssw0ssc4kcoks40osk:6379
```

Step 4: Redeploy Weavink

After updating environment variables, redeploy the Weavink application in Coolify.

Step 5: Verify Application Connectivity

```bash
# Check Weavink logs for Redis connection
docker logs $(docker ps -q -f name=weavink) | grep -i redis
```

Step 6: Cancel Redis Cloud (After Verification)

Once confirmed working:

1. Monitor for 24-48 hours
2. Cancel Redis Cloud subscription
3. Delete Redis Cloud instance

---

6. Configuration Reference

Memory Configuration

Setting	Value	Description
`--maxmemory`	2gb	Maximum memory Redis will use
`--maxmemory-policy`	allkeys-lru	Eviction policy when memory is full

Eviction Policies

Policy	Description	Use Case
`noeviction`	Return error on write when full	When data loss is unacceptable
`allkeys-lru`	Evict least recently used keys	**General caching (recommended)**
`volatile-lru`	Evict LRU keys with TTL set	Mixed persistent + cache data
`allkeys-random`	Evict random keys	When all keys equally important
`volatile-ttl`	Evict keys with shortest TTL	Time-sensitive cache

Persistence Options

Redis saves data to disk by default (RDB snapshots). Current config uses defaults:

Setting	Default	Description
`save 900 1`	Enabled	Save if 1 key changed in 900 seconds
`save 300 10`	Enabled	Save if 10 keys changed in 300 seconds
`save 60 10000`	Enabled	Save if 10000 keys changed in 60 seconds

Disable Persistence (Pure Cache Mode)

If you want Redis as pure cache with no disk persistence:

```yaml
command: redis-server --maxmemory 2gb --maxmemory-policy allkeys-lru --save "" --appendonly no
```

Enable AOF Persistence (Maximum Durability)

For maximum data durability:

```yaml
command: redis-server --maxmemory 2gb --maxmemory-policy allkeys-lru --appendonly yes --appendfsync everysec
```

---

7. Maintenance Commands

Daily Operations

```bash
# Check container status
docker ps | grep redis

# Check Redis is responding
docker exec redis-hgw008ssw0ssc4kcoks40osk redis-cli ping

# View Redis info
docker exec redis-hgw008ssw0ssc4kcoks40osk redis-cli info
```

Memory Monitoring

```bash
# Check memory usage
docker exec redis-hgw008ssw0ssc4kcoks40osk redis-cli info memory

# Key metrics to watch:
# - used_memory_human: Current memory usage
# - used_memory_peak_human: Peak memory usage
# - maxmemory_human: Max allowed memory
```

Key Statistics

```bash
```

```
# Count total keys
docker exec redis-hgw008ssw0ssc4kcoks40osk redis-cli dbsize

# Get all keys (use with caution in production)
docker exec redis-hgw008ssw0ssc4kcoks40osk redis-cli keys '*'

# Get key info
docker exec redis-hgw008ssw0ssc4kcoks40osk redis-cli type <key>
docker exec redis-hgw008ssw0ssc4kcoks40osk redis-cli ttl <key>
```

Performance Testing

```bash
# Quick latency test
docker exec redis-hgw008ssw0ssc4kcoks40osk redis-cli --latency

# Full benchmark
docker exec redis-hgw008ssw0ssc4kcoks40osk redis-benchmark -t ping,set,get -n 10000 -q
```

Clear Cache

```bash
# Clear all data (use with caution!)
docker exec redis-hgw008ssw0ssc4kcoks40osk redis-cli flushall

# Clear current database only
docker exec redis-hgw008ssw0ssc4kcoks40osk redis-cli flushdb
```

Restart Redis

```bash
# Via Docker
docker restart redis-hgw008ssw0ssc4kcoks40osk

# Via Coolify UI
# Go to Coolify → Project → Redis → Restart
```

View Logs

```bash
```

```
# View Redis logs
docker logs redis-hgw008ssw0ssc4kcoks40osk --tail 50

# Follow logs in real-time
docker logs -f redis-hgw008ssw0ssc4kcoks40osk
```

---

8. Scaling Guide

Current Resource Usage

Resource	Allocated	Typical Usage
Memory	2GB	~50-200MB for <100 users
CPU	Shared	Minimal
Disk	Volume	~10-50MB

Capacity Planning

Users	Estimated Cache Size	Recommended Memory
1-50	~50MB	256MB
50-200	~100-200MB	512MB
200-500	~200-500MB	1GB
500-1000	~500MB-1GB	2GB
1000+	1GB+	4GB+

How to Change Memory Allocation

Update docker-compose in Coolify:

```yaml
command: redis-server --maxmemory 4gb --maxmemory-policy allkeys-lru
```

Then **Save** and **Redeploy**.

Memory Allocation Recommendations

Total Server RAM	Redis Allocation	Notes
8GB	1-2GB	Leave room for app + Neo4j
16GB	2-4GB	Current setup
32GB	4-8GB	Heavy caching

Current Server Memory Budget

```
Total RAM: 16GB
├── Neo4j Page Cache: 4GB
├── Neo4j Heap: 2GB
├── Redis: 2GB
├── Weavink App: ~1GB
├── OS + Docker: ~1GB
└── Available: ~6GB buffer
```

---

9. Troubleshooting

Problem: Container won't start

Symptoms: Container exits immediately after starting

Solution:

```bash
# Check logs
docker logs redis-hgw008ssw0ssc4kcoks40osk

# Common issues:
# - Memory allocation too high
# - Volume permissions
```

Problem: Connection refused

Symptoms: App can't connect to Redis

Causes & Solutions:

1. **Container not running**:

```bash
   docker ps | grep redis
   # If not running, restart via Coolify
```

2. **Wrong hostname**:

```bash
```

```
  # Verify container name
  docker ps --format "{{.Names}}" | grep redis
  # Use exact name in REDIS_URL
```

3. **Network isolation**:

```bash
  # Ensure app and Redis are in same Docker network
  docker network inspect coolify
```

Problem: High memory usage

Symptoms: Redis using more memory than expected

Solution:

```bash
# Check memory stats
docker exec redis-hgw008ssw0ssc4kcoks40osk redis-cli info memory

# If near maxmemory, either:
# 1. Increase maxmemory in config
# 2. Reduce TTL on cached items
# 3. Clear unnecessary keys
```

Problem: Slow performance

Symptoms: Higher than expected latency

Diagnosis:

```bash
# Run latency test
docker exec redis-hgw008ssw0ssc4kcoks40osk redis-cli --latency

# Check for slow commands
docker exec redis-hgw008ssw0ssc4kcoks40osk redis-cli slowlog get 10
```

Common causes:

- Large keys (> 1MB)

- Blocking commands (KEYS, SMEMBERS on large sets)
- Persistence causing I/O spikes

Problem: Data loss after restart

Symptoms: Keys disappear after container restart

Cause: Persistence not configured or volume not mounted

Solution:

```bash
# Verify volume is mounted
docker inspect redis-hgw008ssw0ssc4kcoks40osk | grep -A 10 Mounts

# Check RDB file exists
docker exec redis-hgw008ssw0ssc4kcoks40osk ls -la /data/
```

Quick Reference Card

Container Name

```
redis-hgw008ssw0ssc4kcoks40osk
```

Connection URL

```
redis://redis-hgw008ssw0ssc4kcoks40osk:6379
```

Server Details

```
IP: 159.69.215.143
SSH: ssh root@159.69.215.143
```

Common Commands

```bash
```

```
# Ping test
docker exec redis-hgw008ssw0ssc4kcoks40osk redis-cli ping

# Memory info
docker exec redis-hgw008ssw0ssc4kcoks40osk redis-cli info memory

# Key count
docker exec redis-hgw008ssw0ssc4kcoks40osk redis-cli dbsize

# Latency test
docker exec redis-hgw008ssw0ssc4kcoks40osk redis-cli --latency

# Full benchmark
docker exec redis-hgw008ssw0ssc4kcoks40osk redis-benchmark -t ping,set,get -n 10000 -q

# Clear all data
docker exec redis-hgw008ssw0ssc4kcoks40osk redis-cli flushall

# View logs
docker logs redis-hgw008ssw0ssc4kcoks40osk --tail 50
```

Document History

Date	Version	Changes
2025-11-30	1.0	Initial deployment and documentation

Document created after successful deployment of self-hosted Redis on Hetzner VPS via Coolify.