Qdrant Self-Hosted Guide: Pinecone → 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
Qdrant Version	1.16.1

Table of Contents

1. Executive Summary
2. Why Self-Host Qdrant?
3. Infrastructure Setup
4. Performance Benchmarks
5. Migration Guide
6. Data Structure
7. API Reference
8. Maintenance Commands
9. Scaling Guide
10. Application Integration
11. Troubleshooting

---

1. Executive Summary

What We Did

Deployed a self-hosted Qdrant vector database on our Hetzner VPS via Coolify to replace Pinecone for semantic search and contact similarity matching.

Final Result

- **1.2ms search latency** (self-hosted) vs **50-100ms** (Pinecone) - **50-100x faster!**
- **103 vectors** successfully migrated across 2 collections
- **€0/month** vs potential Pinecone paid tier
- Full control over data and configuration (GDPR compliant)

Key Benefit

Vector search on the same server eliminates network round-trip, resulting in sub-2ms semantic search queries.

---

2. Why Self-Host Qdrant?

Pinecone vs Qdrant (Self-hosted) Comparison

Aspect	Pinecone	Qdrant (Self-hosted)
Cost	Free tier (100K vectors)	€0 (included in VPS)
Search Latency	50-100ms	**1.2ms**
API Latency	50-100ms	**0.5ms**
Vector Limit	100K (free)	Unlimited (disk-limited)
Data Location	AWS us-east-1	Your server (EU/GDPR)
Maintenance	None	Minimal
Features	Limited	Full (filtering, payloads)

When to Self-Host

- Need sub-10ms vector search latency
- Want data sovereignty (GDPR compliance)
- Approaching Pinecone free tier limits
- Running other services on the same VPS

When to Keep Pinecone

- No VPS available
- Need managed infrastructure
- Global distribution 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
```

Qdrant Resource Usage

```
Base Memory: ~100-200MB
Per 100K vectors (1024 dim): ~1GB RAM
Storage: ~1.5MB per 1000 vectors
```

Docker Compose Configuration (Coolify)

yaml

```yaml
services:
  qdrant:
    image: 'qdrant/qdrant:latest'
    restart: unless-stopped
    environment:
      - QDRANT__SERVICE__GRPC_PORT=6334
    volumes:
      - 'qdrant-storage:/qdrant/storage'
      - 'qdrant-snapshots:/qdrant/snapshots'
    healthcheck:
      test:
        - CMD
        - wget
        - '-q'
        - '--spider'
        - 'http://localhost:6333/healthz'
      interval: 30s
      timeout: 10s
      retries: 3
volumes:
  qdrant-storage: null
  qdrant-snapshots: null
```

Container Details

```
Container Name: qdrant-qkkkc8kskocgwo0o8c444cgo
Volume (storage): qkkkc8kskocgwo0o8c444cgo_qdrant-storage
Volume (snapshots): qkkkc8kskocgwo0o8c444cgo_qdrant-snapshots
HTTP Port: 6333
gRPC Port: 6334
Internal IP: 10.0.4.2
```

Connection URLs

```
HTTP API: http://qdrant-qkkkc8kskocgwo0o8c444cgo:6333
gRPC API: qdrant-qkkkc8kskocgwo0o8c444cgo:6334
Direct IP: http://10.0.4.2:6333
```

Ports

Port	Protocol	Purpose
6333	HTTP	REST API & Web Dashboard
6334	gRPC	High-performance API

> **Security Note**: Qdrant is only accessible within Docker's internal network. No public exposure needed.

---

4. Performance Benchmarks

Test Environment

- **Server**: Hetzner CX43 (8 vCPU, 16GB RAM)
- **Qdrant**: v1.16.1
- **Data**: 103 vectors, 1024 dimensions, Cosine distance

Latency Results

Operation	Time
Health Check	**0.000011s** (11μs)
List Collections	**0.83ms**
Vector Search (top 3)	**1.17ms**
Collection Info	**0.54ms**

Search Quality Test

Query: Find similar contacts to "David Chen, Data Engineer at Meta"

Rank	Score	Result	Relevance
1	1.00	David Chen @ Meta (exact match)	✅ Perfect
2	0.91	David Chen @ Tesla	✅ Same person, different company
3	0.91	Bob Johnson @ Meta (ML Research)	✅ Same company, similar role

Comparison with Pinecone

Metric	Pinecone	Qdrant	Improvement
Search Latency	50-100ms	**1.2ms**	**50-100x faster**
API Response	50-100ms	**0.5ms**	**100-200x faster**
Cold Start	~500ms	**~50ms**	**10x faster**

---

5. Migration Guide

Overview

Pinecone doesn't provide a direct export feature, so migration requires:

1. Fetch all vectors from Pinecone API
2. Export to JSON file
3. Import into Qdrant

Step 1: Check Pinecone Index Stats

bash

```
curl -H "Api-Key: YOUR_PINECONE_API_KEY" \
  https://YOUR_INDEX.svc.YOUR_REGION.pinecone.io/describe_index_stats
```

Example response:

json

```
{
  "namespaces": {
    "user_ABC123": {"vectorCount": 102},
    "user_XYZ789": {"vectorCount": 1}
  },
  "totalVectorCount": 103,
  "dimension": 1024
}
```

Step 2: Export from Pinecone

Create `pinecone_export.py` on your local machine:

```
python
```

```
from pinecone import Pinecone
import json

# Initialize
pc = Pinecone(api_key="YOUR_PINECONE_API_KEY")
index = pc.Index("YOUR_INDEX_NAME", host="YOUR_INDEX_HOST")

# Get stats
stats = index.describe_index_stats()
print(f"Total vectors: {stats.total_vector_count}")

all_vectors = []

# Export each namespace
for namespace in stats.namespaces.keys():
    print(f"\nExporting namespace: '{namespace}'")

    ids_list = []
    for ids_batch in index.list(namespace=namespace):
        ids_list.extend(ids_batch)

    print(f"  Found {len(ids_list)} IDs")

    # Fetch in batches of 100
    for i in range(0, len(ids_list), 100):
        batch_ids = ids_list[i:i+100]
        fetched = index.fetch(ids=batch_ids, namespace=namespace)

        for id, vec in fetched.vectors.items():
            all_vectors.append({
                "id": id,
                "values": vec.values,
                "metadata": vec.metadata if vec.metadata else {},
                "namespace": namespace
            })

# Save to file
with open("pinecone_export.json", "w") as f:
    json.dump(all_vectors, f)

print(f"\n✅ Exported {len(all_vectors)} vectors to pinecone_export.json")
```

Run it:

```
bash

pip install pinecone
python3 pinecone_export.py
```

Step 3: Transfer to Server

```
bash

scp pinecone_export.json root@159.69.215.143:/root/
```

Step 4: Import into Qdrant

Create `/root/qdrant_import.py` on the server:

```
python
```

```
import json
import requests

QDRANT_URL = "http://10.0.4.2:6333"

# Load exported data
with open("/root/pinecone_export.json", "r") as f:
    vectors = json.load(f)

print(f"Loaded {len(vectors)} vectors")

# Get unique namespaces (will become collections in Qdrant)
namespaces = set(v["namespace"] for v in vectors)
print(f"Namespaces: {namespaces}")

# Create collections for each namespace
for namespace in namespaces:
    collection_name = namespace.replace("user_", "")  # Clean up name

    # Create collection
    resp = requests.put(
        f"{QDRANT_URL}/collections/{collection_name}",
        json={
            "vectors": {
                "size": 1024,
                "distance": "Cosine"
            }
        }
    )
    print(f"Created collection '{collection_name}': {resp.status_code}")

# Insert vectors
for namespace in namespaces:
    collection_name = namespace.replace("user_", "")
    ns_vectors = [v for v in vectors if v["namespace"] == namespace]

    # Prepare points
    points = []
    for i, v in enumerate(ns_vectors):
        points.append({
            "id": i + 1,  # Qdrant needs integer or UUID
            "vector": v["values"],
            "payload": {
                "original_id": v["id"],
                **v["metadata"]
            }
        })

    # Upsert in batches of 100
    for i in range(0, len(points), 100):
        batch = points[i:i+100]
        resp = requests.put(
            f"{QDRANT_URL}/collections/{collection_name}/points",
            json={"points": batch}
        )
        print(f"  Inserted {len(batch)} vectors into '{collection_name}': {resp.status_code}")

print("\n✅ Import complete!")

# Verify
for namespace in namespaces:
    collection_name = namespace.replace("user_", "")
    resp = requests.get(f"{QDRANT_URL}/collections/{collection_name}")
    info = resp.json()
    print(f"Collection '{collection_name}': {info['result']['points_count']} points")
```

Run it:

```
bash

python3 /root/qdrant_import.py
```

Step 5: Verify Migration

```
bash
```

```
# List collections
curl -s http://10.0.4.2:6333/collections | jq

# Check collection details
curl -s http://10.0.4.2:6333/collections/YOUR_COLLECTION | jq

# Test search
curl -s -X POST http://10.0.4.2:6333/collections/YOUR_COLLECTION/points/search \
  -H "Content-Type: application/json" \
  -d '{
    "vector": [0.1, 0.2, ...],
    "limit": 5,
    "with_payload": true
  }' | jq
```

---

6. Data Structure

Pinecone → Qdrant Mapping

Pinecone Concept	Qdrant Equivalent
Index	Instance (server)
Namespace	Collection
Vector ID	Point ID
Metadata	Payload
Dimension	Vector Size
Metric (cosine)	Distance (Cosine)

Current Collections

Collection	Points	Dimension	Distance
`IFxPCgSA8NapEq5W8jh6yHrtJGJ2`	102	1024	Cosine
`ScmVq6p8ubQ9JFbniF2Vg5ocmbv2`	1	1024	Cosine

Payload Schema

Each vector point contains:

```
json
```

```
{
  "id": 1,
  "vector": [0.026, -0.055, ...],  // 1024 dimensions
  "payload": {
    "original_id": "contact_1764088271366_mj1ve5wuv",
    "name": "David Chen",
    "email": "david.chen@meta.com",
    "company": "Meta",
    "jobTitle": "Data Engineer",
    "message": "Contact from Meta - Data Engineer",
    "status": "active",
    "tags": "engineering,frontend,ai,nlp",
    "userId": "IFxPCgSA8NapEq5W8jh6yHrtJGJ2"
  }
}
```

---

7. API Reference

Base URL

```
http://qdrant-qkkkc8kskocgwo0o8c444cgo:6333
```

Health Check

bash

```bash
curl http://10.0.4.2:6333/healthz
# Response: healthz check passed
```

List Collections

bash

```bash
curl http://10.0.4.2:6333/collections
```

Get Collection Info

bash

```bash
curl http://10.0.4.2:6333/collections/{collection_name}
```

Create Collection

bash

```
curl -X PUT http://10.0.4.2:6333/collections/{collection_name} \
  -H "Content-Type: application/json" \
  -d '{
    "vectors": {
      "size": 1024,
      "distance": "Cosine"
    }
  }'
```

Insert/Update Vectors

bash

```
curl -X PUT http://10.0.4.2:6333/collections/{collection_name}/points \
  -H "Content-Type: application/json" \
  -d '{
    "points": [
      {
        "id": 1,
        "vector": [0.1, 0.2, ...],
        "payload": {"name": "John", "email": "john@example.com"}
      }
    ]
  }'
```

Search Vectors

bash

```
curl -X POST http://10.0.4.2:6333/collections/{collection_name}/points/search \
  -H "Content-Type: application/json" \
  -d '{
    "vector": [0.1, 0.2, ...],
    "limit": 10,
    "with_payload": true,
    "score_threshold": 0.7
  }'
```

Search with Filtering

bash

```
curl -X POST http://10.0.4.2:6333/collections/{collection_name}/points/search \
  -H "Content-Type: application/json" \
  -d '{
    "vector": [0.1, 0.2, ...],
    "limit": 10,
    "with_payload": true,
    "filter": {
      "must": [
        {"key": "status", "match": {"value": "active"}}
      ]
    }
  }'
```

Delete Points

bash

```
curl -X POST http://10.0.4.2:6333/collections/{collection_name}/points/delete \
  -H "Content-Type: application/json" \
  -d '{
    "points": [1, 2, 3]
  }'
```

Delete Collection

bash

```
curl -X DELETE http://10.0.4.2:6333/collections/{collection_name}
```

---

8. Maintenance Commands

Daily Operations

bash

```
# Check container status
docker ps | grep qdrant

# Health check
curl -s http://10.0.4.2:6333/healthz

# List all collections
curl -s http://10.0.4.2:6333/collections | jq

# Get telemetry/stats
curl -s http://10.0.4.2:6333/telemetry | jq
```

Collection Management

bash

```
# Get collection info
curl -s http://10.0.4.2:6333/collections/COLLECTION_NAME | jq

# Count points in collection
curl -s http://10.0.4.2:6333/collections/COLLECTION_NAME | jq '.result.points_count'

# Get collection size
curl -s http://10.0.4.2:6333/collections/COLLECTION_NAME | jq '.result.segments_count'
```

Backup & Snapshots

bash

```
# Create snapshot of a collection
curl -X POST http://10.0.4.2:6333/collections/COLLECTION_NAME/snapshots

# List snapshots
curl http://10.0.4.2:6333/collections/COLLECTION_NAME/snapshots

# Create full storage snapshot
curl -X POST http://10.0.4.2:6333/snapshots
```

View Logs

bash

```
# View Qdrant logs
docker logs qdrant-qkkkc8kskocgwo0o8c444cgo --tail 50

# Follow logs in real-time
docker logs -f qdrant-qkkkc8kskocgwo0o8c444cgo
```

Restart Qdrant

```bash
# Via Docker
docker restart qdrant-qkkkc8kskocgwo0o8c444cgo

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

Check Resource Usage

```bash
# Memory and CPU usage
docker stats qdrant-qkkkc8kskocgwo0o8c444cgo --no-stream

# Disk usage
docker exec qdrant-qkkkc8kskocgwo0o8c444cgo du -sh /qdrant/storage
```

---

9. Scaling Guide

Memory Usage Estimates

Qdrant uses memory for:

1. **HNSW Index**: Graph structure for fast search
2. **Vectors**: Can be in RAM or memory-mapped
3. **Payloads**: Stored on disk by default

Vectors	Dimensions	Approx RAM
1K	1024	~50MB
10K	1024	~200MB
100K	1024	~1-2GB
500K	1024	~4-5GB
1M	1024	~8-10GB

Current Usage

```
Vectors: 103
Dimensions: 1024
Estimated RAM: ~50MB
Disk Storage: ~1.5MB
```

Capacity Planning

Users	Est. Contacts	Est. Vectors	RAM Needed
1-10	~500	~500	~100MB
10-50	~2,500	~2,500	~200MB
50-200	~10,000	~10,000	~500MB
200-500	~50,000	~50,000	~1-2GB
500-1000	~100,000	~100,000	~2-4GB

Memory Optimization Options

If RAM becomes constrained, enable memory-mapped storage:

bash

```bash
curl -X PATCH http://10.0.4.2:6333/collections/COLLECTION_NAME \
  -H "Content-Type: application/json" \
  -d '{
    "optimizers_config": {
      "memmap_threshold": 10000
    }
  }'
```

Current Server Memory Budget

```
Total RAM: 16GB
├── Neo4j Page Cache: 4GB
├── Neo4j Heap: 2GB
├── Redis: 2GB
├── Qdrant: ~200MB (current), up to 2GB (growth)
├── Weavink App: ~1GB
├── OS + Docker: ~1GB
└── Available: ~4-6GB buffer
```

---

10. Application Integration

Environment Variables for Weavink

```env
# Old (Pinecone)
PINECONE_API_KEY=pcsk_xxxxx
PINECONE_INDEX=weavink
PINECONE_HOST=weavink-xxx.svc.pinecone.io

# New (Qdrant)
QDRANT_URL=http://qdrant-qkkkc8kskocgwo0o8c444cgo:6333
QDRANT_API_KEY=  # Optional, not set for internal network
```

Code Migration: Pinecone → Qdrant

JavaScript/TypeScript Example

Before (Pinecone):

```typescript
```

```
import { Pinecone } from '@pinecone-database/pinecone';

const pinecone = new Pinecone({ apiKey: process.env.PINECONE_API_KEY });
const index = pinecone.index('weavink');

// Upsert
await index.namespace(userId).upsert([{
  id: contactId,
  values: embedding,
  metadata: { name, email, company }
}]);

// Search
const results = await index.namespace(userId).query({
  vector: queryEmbedding,
  topK: 10,
  includeMetadata: true
});
```

After (Qdrant):

typescript

```typescript
import { QdrantClient } from '@qdrant/js-client-rest';

const qdrant = new QdrantClient({ url: process.env.QDRANT_URL });

// Upsert
await qdrant.upsert(userId, {
  points: [{
    id: pointId,  // Must be integer or UUID
    vector: embedding,
    payload: { name, email, company, original_id: contactId }
  }]
});

// Search
const results = await qdrant.search(userId, {
  vector: queryEmbedding,
  limit: 10,
  with_payload: true
});
```

Key Differences

Feature	Pinecone	Qdrant
Client Package	`@pinecone-database/pinecone`	`@qdrant/js-client-rest`
Namespace	`index.namespace(ns)`	Collection name = namespace
Vector ID	String	Integer or UUID
Metadata	`metadata`	`payload`
Search	`query()`	`search()`
Results	`matches`	Array of points

Install Qdrant Client

```bash
npm install @qdrant/js-client-rest
```

---

11. Troubleshooting

Problem: Container won't start

Check logs:

```bash
docker logs qdrant-qkkkc8kskocgwo0o8c444cgo
```

Common causes:

- Insufficient disk space
- Port already in use
- Corrupted storage volume

Solution:

```bash
# Check disk space
df -h

# Restart with fresh storage (WARNING: deletes data)
docker volume rm qkkkc8kskocgwo0o8c444cgo_qdrant-storage
# Then redeploy from Coolify
```

Problem: Connection refused

Check container is running:

```bash
docker ps | grep qdrant
```

Verify IP address:

```bash
docker inspect qdrant-qkkkc8kskocgwo0o8c444cgo | grep IPAddress
```

Test connectivity:

```bash
curl http://10.0.4.2:6333/healthz
```

Problem: Search returns no results

Verify collection exists:

```bash
curl http://10.0.4.2:6333/collections
```

Check collection has points:

```bash
curl http://10.0.4.2:6333/collections/COLLECTION_NAME | jq '.result.points_count'
```

Verify vector dimensions match:

```bash
curl http://10.0.4.2:6333/collections/COLLECTION_NAME | jq '.result.config.params.vectors.size'
# Should be 1024
```

Problem: Slow search performance

Check if index is built:

```bash
curl http://10.0.4.2:6333/collections/COLLECTION_NAME | jq '.result.indexed_vectors_count'
```

Note: HNSW index is built automatically when collection exceeds `indexing_threshold` (default: 10,000 vectors). For smaller collections, brute-force search is used (still fast).

Problem: High memory usage

Check current usage:

```bash
docker stats qdrant-qkkkc8kskocgwo0o8c444cgo --no-stream
```

Enable memory mapping:

```bash
curl -X PATCH http://10.0.4.2:6333/collections/COLLECTION_NAME \
  -H "Content-Type: application/json" \
  -d '{"optimizers_config": {"memmap_threshold": 10000}}'
```

Quick Reference Card

Container Name

```
qdrant-qkkkc8kskocgwo0o8c444cgo
```

Connection URLs

```
HTTP: http://qdrant-qkkkc8kskocgwo0o8c444cgo:6333
gRPC: qdrant-qkkkc8kskocgwo0o8c444cgo:6334
Direct: http://10.0.4.2:6333
```

Server Details

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

Common Commands

```bash
```

```
# Health check
curl http://10.0.4.2:6333/healthz

# List collections
curl -s http://10.0.4.2:6333/collections | jq

# Collection info
curl -s http://10.0.4.2:6333/collections/COLLECTION | jq

# Search
curl -X POST http://10.0.4.2:6333/collections/COLLECTION/points/search \
  -H "Content-Type: application/json" \
  -d '{"vector": [...], "limit": 10, "with_payload": true}'

# Create snapshot
curl -X POST http://10.0.4.2:6333/collections/COLLECTION/snapshots

# View logs
docker logs qdrant-qkkkc8kskocgwo0o8c444cgo --tail 50

# Resource usage
docker stats qdrant-qkkkc8kskocgwo0o8c444cgo --no-stream
```

Collections

Collection	Points	User ID
`IFxPCgSA8NapEq5W8jh6yHrtJGJ2`	102	Primary test user
`ScmVq6p8ubQ9JFbniF2Vg5ocmbv2`	1	Secondary user

Document History

Date	Version	Changes
2025-11-30	1.0	Initial deployment, migration from Pinecone, documentation

---

Document created after successful migration from Pinecone to self-hosted Qdrant on Hetzner VPS via Coolify.