**Qdrant Basic Setup, Use Case, and Retrieval Example**

### **What is Qdrant?**

Qdrant is an open-source vector search engine and database that specializes in high-performance similarity searches. It's designed for scalable real-time searches on embeddings, often used in applications such as semantic search, recommendation systems, and conversational AI.

### **Basic Setup**

#### **1. Installation**

Qdrant can be set up locally or using Docker.

**Using Docker:**bash  
Copy code  
docker run -p 6333:6333 qdrant/qdrant

**Using pip (Python):**bash  
Copy code  
pip install qdrant-client

#### **2. Running the Server**

If using Docker, ensure the container is running. The server will be available on http://localhost:6333.

### **Use Case: Document Retrieval**

Suppose we are building a document retrieval system where we index textual data as embeddings and retrieve the most relevant documents for a query.

#### **Steps:**

1. **Prepare Embeddings** Use a pre-trained model (like OpenAI, Hugging Face Transformers, etc.) to convert text into dense vector embeddings.
2. **Store Embeddings in Qdrant** Push these embeddings into Qdrant for indexing.
3. **Query the Index** Search for similar embeddings in Qdrant to retrieve relevant documents.

### **Example Code**

python

Copy code

from qdrant\_client import QdrantClient

from qdrant\_client.http.models import Distance, VectorParams, PointStruct

from sentence\_transformers import SentenceTransformer

# Step 1: Initialize Qdrant Client

client = QdrantClient(host="localhost", port=6333)

# Step 2: Create a Collection

collection\_name = "document\_collection"

client.recreate\_collection(

collection\_name=collection\_name,

vectors\_config=VectorParams(size=768, distance=Distance.COSINE) # 768 is embedding size

)

# Step 3: Generate and Insert Embeddings

model = SentenceTransformer('all-MiniLM-L6-v2')

documents = [

"Qdrant is a vector search engine.",

"Vector search is essential for modern AI applications.",

"Document retrieval can leverage embeddings.",

"Similarity search helps in recommendation systems."

]

# Convert text to embeddings

embeddings = model.encode(documents)

# Insert data into Qdrant

points = [

PointStruct(id=i, vector=embedding.tolist(), payload={"content": doc})

for i, (embedding, doc) in enumerate(zip(embeddings, documents))

]

client.upsert(collection\_name=collection\_name, points=points)

# Step 4: Query the Collection

query\_text = "What is Qdrant used for?"

query\_vector = model.encode([query\_text])[0] # Convert query to embedding

# Search for the top-2 most similar documents

results = client.search(

collection\_name=collection\_name,

query\_vector=query\_vector.tolist(),

limit=2 # Number of results to return

)

# Display Results

for result in results:

print(f"Document: {result.payload['content']}, Score: {result.score}")

### **Output Example**

For the query "What is Qdrant used for?", the results might look like:

vbnet

Copy code

Document: Qdrant is a vector search engine., Score: 0.89

Document: Vector search is essential for modern AI applications., Score: 0.83

### **Key Features**

* **Payloads:** Store metadata with vectors for contextual filtering.
* **Filters:** Apply Boolean logic to narrow down results during searches.
* **Distance Metrics:** Supports COSINE, DOT, and EUCLIDEAN.

### **Advantages of Qdrant**

1. **Performance:** Optimized for real-time and high-throughput queries.
2. **Scalability:** Handles large-scale datasets with ease.
3. **Integration:** Compatible with popular ML tools (e.g., Hugging Face, TensorFlow).

This setup and example should give you a clear idea of how to use Qdrant for a basic document retrieval use case. Let me know if you need further details!

4o

##### **You said:**

show one example of using qdrant cluster cloud

3/3

###### **ChatGPT said:**

ChatGPT

Here's an example of using **Qdrant Cloud** in a cluster environment for a simple use case of storing and retrieving embeddings.

### **Prerequisites**

1. **Create a Qdrant Cloud Cluster**:
   * Sign up for Qdrant Cloud [here](https://qdrant.tech/).
   * Create a cluster and note down the endpoint and API key.

**Install Required Libraries**:  
bash  
Copy code  
pip install qdrant-client sentence-transformers

### **Code Example: Using Qdrant Cloud**

This example demonstrates creating a collection, uploading embeddings, and querying the Qdrant Cloud cluster.

python

Copy code

from qdrant\_client import QdrantClient

from qdrant\_client.http.models import Distance, VectorParams, PointStruct

from sentence\_transformers import SentenceTransformer

# Step 1: Connect to Qdrant Cloud Cluster

client = QdrantClient(

url="https://<your-cluster-endpoint>", # Replace with your Qdrant Cloud cluster URL

api\_key="<your-api-key>" # Replace with your Qdrant API Key

)

# Step 2: Create a Collection

collection\_name = "cloud\_document\_collection"

client.recreate\_collection(

collection\_name=collection\_name,

vectors\_config=VectorParams(size=768, distance=Distance.COSINE) # Adjust size for your embedding model

)

# Step 3: Prepare and Upload Embeddings

# Using Sentence Transformers to generate embeddings

model = SentenceTransformer('all-MiniLM-L6-v2')

documents = [

"Qdrant cloud provides a managed service.",

"Vector search is powerful for semantic search.",

"Embeddings enable similarity-based retrieval.",

"Cloud-based solutions are scalable and reliable."

]

# Generate embeddings

embeddings = model.encode(documents)

# Insert data points

points = [

PointStruct(id=i, vector=embedding.tolist(), payload={"content": doc})

for i, (embedding, doc) in enumerate(zip(embeddings, documents))

]

# Upload points to Qdrant Cloud

client.upsert(collection\_name=collection\_name, points=points)

# Step 4: Query the Cloud Collection

query\_text = "What are the benefits of Qdrant cloud?"

query\_vector = model.encode([query\_text])[0] # Convert query to embedding

# Search for the top-3 most similar documents

results = client.search(

collection\_name=collection\_name,

query\_vector=query\_vector.tolist(),

limit=3 # Number of results to return

)

# Step 5: Display Results

print("Query Results:")

for result in results:

print(f"Document: {result.payload['content']}, Score: {result.score}")

### **Explanation**

1. **Connect to Qdrant Cloud**:
   * Use the cloud endpoint and API key provided by Qdrant Cloud.
2. **Create a Collection**:
   * Collections organize your data, with a specified embedding size and distance metric.
3. **Upload Data**:
   * Generate embeddings using a model and upload them with metadata (payload).
4. **Query the Collection**:
   * Use a query vector to search for similar vectors in the collection.

### **Expected Output**

For a query like "What are the benefits of Qdrant cloud?", you might see:

yaml

Copy code

Query Results:

Document: Qdrant cloud provides a managed service., Score: 0.92

Document: Cloud-based solutions are scalable and reliable., Score: 0.88

Document: Vector search is powerful for semantic search., Score: 0.84

### **Benefits of Qdrant Cloud**

1. **Managed Service**: No need to worry about setup, scaling, or maintenance.
2. **Scalability**: Ideal for large-scale data with high availability.
3. **Security**: Secure data access with API keys.