Documentation Local Quickstart

How to Get Started with Qdrant Locally

In this short example, you will use the Python Client to create a Collection, load data into it and run a basic search query.

ⓘ Before you start, please make sure Docker is installed and running on your system.

Download and run

First, download the latest Qdrant image from Dockerhub:

```
docker pull qdrant/qdrant
```

Then, run the service:

```
docker run -p 6333:6333 -p 6334:6334 \
    -v "$(pwd)/qdrant_storage:/qdrant/storage:z" \
    qdrant/qdrant
```

ⓘ On Windows, you may need to create a named Docker volume instead of mounting a local folder.

Under the default configuration all data will be stored in the `./qdrant_storage` directory. This will also be the only directory that both the Container and the host machine can both see.

Qdrant is now accessible:

- REST API: localhost:6333
- Web UI: localhost:6333/dashboard
- GRPC API: localhost:6334

Initialize the client

```
from qdrant_client import QdrantClient

client = QdrantClient(url="http://localhost:6333")
```

ⓘ By default, Qdrant starts with no encryption or authentication . This means anyone with network access to your machine can access your Qdrant container instance. Please read Security carefully for details on how to secure your instance.

Create a collection

You will be storing all of your vector data in a Qdrant collection. Let's call it `test_collection` . This collection will be using a dot product distance metric to compare vectors.

```
from qdrant_client.models import Distance, VectorParams

client.create_collection(
    collection_name="test_collection",
    vectors_config=VectorParams(size=4, distance=Distance.DOT),
)
```

Add vectors

Let's now add a few vectors with a payload. Payloads are other data you want to associate with the vector:

```
from qdrant_client.models import PointStruct

operation_info = client.upsert(
```

```
    collection_name="test_collection",
    wait=True,
    points=[
        PointStruct(id=1, vector=[0.05, 0.61, 0.76, 0.74], payload={"c
        PointStruct(id=2, vector=[0.19, 0.81, 0.75, 0.11], payload={"c
        PointStruct(id=3, vector=[0.36, 0.55, 0.47, 0.94], payload={"c
        PointStruct(id=4, vector=[0.18, 0.01, 0.85, 0.80], payload={"c
        PointStruct(id=5, vector=[0.24, 0.18, 0.22, 0.44], payload={"c
        PointStruct(id=6, vector=[0.35, 0.08, 0.11, 0.44], payload={"c
    ],
)

print(operation_info)
```

Response:

```
operation_id=0 status=<UpdateStatus.COMPLETED: 'completed'>
```

Run a query

Let's ask a basic question - Which of our stored vectors are most similar to the query vector `[0.2, 0.1, 0.9, 0.7]`?

```
search_result = client.query_points(
    collection_name="test_collection",
    query=[0.2, 0.1, 0.9, 0.7],
    with_payload=False,
    limit=3
).points

print(search_result)
```

Response:

```
[
```

```
    {
      "id": 4,
      "version": 0,
      "score": 1.362,
      "payload": null,
      "vector": null
    },
    {
      "id": 1,
      "version": 0,
      "score": 1.273,
      "payload": null,
      "vector": null
    },
    {
      "id": 3,
      "version": 0,
      "score": 1.208,
      "payload": null,
      "vector": null
    }
]
```


The results are returned in decreasing similarity order. Note that payload and vector data is missing in these results by default. See payload and vector in the result on how to enable it.

Add a filter

We can narrow down the results further by filtering by payload. Let's find the closest results that include "London".

```
from qdrant_client.models import Filter, FieldCondition, MatchValu

search_result = client.query_points(
    collection_name="test_collection",
    query=[0.2, 0.1, 0.9, 0.7],
    query_filter=Filter(
        must=[FieldCondition(key="city", match=MatchValue(value="Londo
```

```
    ),
    with_payload=True,
    limit=3,
).points

print(search_result)
```

Response:

```
[
    {
        "id": 2,
        "version": 0,
        "score": 0.871,
        "payload": {
            "city": "London"
        },
        "vector": null
    }
]
```

ⓘ To make filtered search fast on real datasets, we highly recommend to create payload indexes!

You have just conducted vector search. You loaded vectors into a database and queried the database with a vector of your own. Qdrant found the closest results and presented you with a similarity score.

Next steps

Now you know how Qdrant works. Getting started with Qdrant Cloud is just as easy. Create an account and use our SaaS completely free. We will take care of infrastructure maintenance and software updates.

To move onto some more complex examples of vector search, read our Tutorials and create your own app with the help of our Examples.

Note: There is another way of running Qdrant locally. If you are a Python developer, we recommend that you try Local Mode in Qdrant Client, as it only takes a few moments to get setup.

Ready to get started with Qdrant?

Start Free