Qdrant 向量数据库存储组件使用说明书

目录

Qdrant 向量数据库存储组件使用说明

1. 组件简介

该组件基于 Qdrant 向量数据库，专门为 `nFlow` 系统开发，旨在处理 `PdfParser`、`ImageParser` 等解析器生成的非结构化数据（如文本和图像），将其转换为向量并存储到 Qdrant 中。该组件支持通过不同的嵌入模型进行向量化处理，并提供高效的存储、检索解决方案，以满足大规模数据（如用于大模型训练）的存储和检索需求。

主要功能：

- **向量化非结构化数据**：通过 Hugging Face 预训练模型将文本或图像转换为向量。
- **数据批量写入**：支持批量写入到 Qdrant 数据库，优化数据存储效率。
- **多种距离度量支持**：支持多种向量相似度计算方法（欧几里得距离、余弦相似度、点积等）。
- **集成数据流**：与 `DATAConnect` 集成，读取数据并进行处理和存储。

2. 环境要求

依赖项

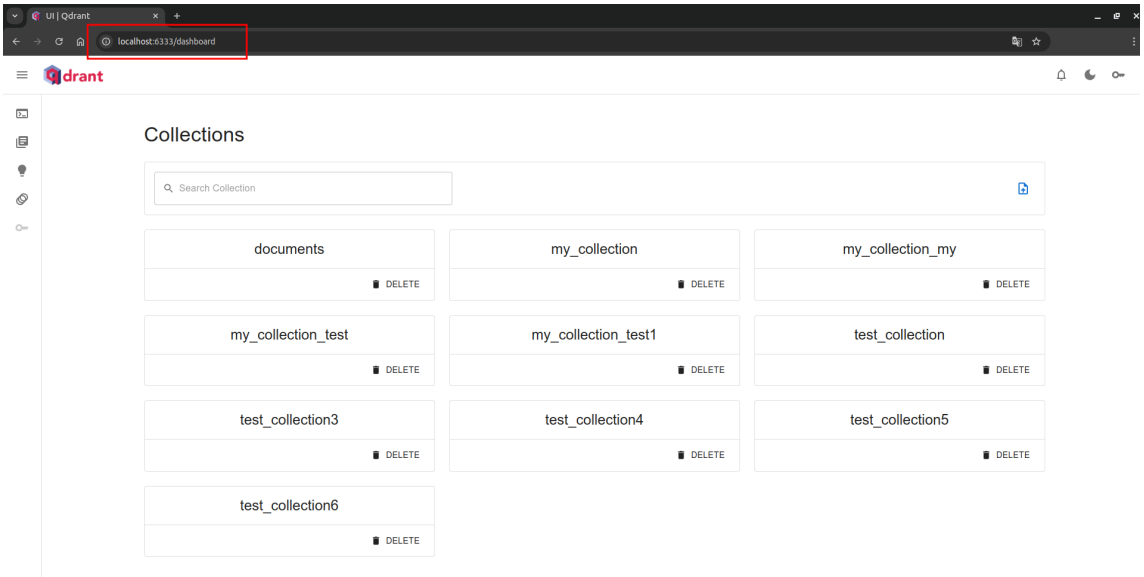
- **Python 版本**：需要 Python 3.10 或更高版本。
- **Qdrant 向量数据库环境**：
- **所需python库**：
  - `qdrant-client`：用于连接和操作 Qdrant 向量数据库
  - `transformers`：用于加载 Hugging Face 的预训练模型
  - `pandas`：用于数据处理
  - `langchain_huggingface`：用于嵌入向量生成
  - 其他依赖：`sys`、`warnings`

Qdrant 向量数据库：

- **本地实例**：可以使用本地数据库，或者通过 Docker 运行 Qdrant 实例。
- **远程实例**：支持通过指定 `host` 和 `port` 连接远程 Qdrant 实例。
- **Docker运行 Qdrant 示例：** 建议使用 Docker 来搭建 Qdrant 环境。可以使用以下命令来运行最新版本的 Qdrant Docker 镜像：

  ```
  docker run -it qdrant/qdrant:latest -p 6333:6333 -p 6334:6334
  ```

  该命令会启动 Qdrant 实例，并将本地的 6333 和 6334 端口映射到 Docker 容器中的相应端口。可以通过web访问http://localhost:6333/dashboard,查看qdrant数据库状态.



3. 参数说明

组件通过命令行参数接收配置参数。以下是各参数的详细说明：

参数名	类型	默认值	说明
`collection_name`	`str`	`default_collection`	Qdrant 中的集合名称，用于存储向量化数据。
`batch_size`	`int`	`100`	批量写入 Qdrant 的数据条数。
`host`	`str`	`127.0.0.1`	连接的 Qdrant 数据库主机地址。
`port`	`int`	`6333`	Qdrant 数据库的端口号。
`grpc_port`	`int`	`6334`	Qdrant gRPC 服务的端口号。
`embed_model`	`str`	`all_MiniLM_L6_v2`	使用的文本嵌入模型。
`distance_metric`	`str`	`cosine`	向量距离度量方式，支持 `cosine`、`euclid`、`dot`、`manhattan`。

支持的嵌入模型：

支持 Hugging Face 上的多种预训练模型，如 `all-MiniLM-L6-v2`、`all-roberta-large-v1`、`sentence-t5-xl` 等。

以下是已集成的向量化模型表格：

序号	名称	参数	介绍
1	`sentence-transformers/all-MiniLM-L6-v2`	`all_MiniLM_L6_v2`	通用文本嵌入(GTE)模型：将句子和段落映射到384维的密集向量空间，可以用于聚类或语义搜索等任务。
2	`sentence-transformers/all-roberta-large-v1`	`all-roberta-large-v1`	通用文本嵌入(GTE)模型：将句子和段落映射到1024维的密集向量空间，可以用于聚类或语义搜索等任务。
3	`sentence-transformers/average_word_embeddings_glove.840B.300d`	`average_word_embeddings_glove.840B.300d`	通用文本嵌入(GTE)模型：将句子和段落映射到300维的密集向量空间，可以用于聚类或语义搜索等任务。
4	`thenlper/gte-small`	`gte-small`	通用文本嵌入(GTE)模型：基于多阶段对比学习的通用文本嵌入，由阿里巴巴达摩学院训练。

序号	名称	参数	介绍
5	`sentence-transformers/sentence-t5-xl`	`sentence-t5-xl`	通用文本嵌入(GTE)模型：将句子和段落映射到768维的密集向量空间，可以用于聚类或语义搜索等任务。
6	`Snowflake/snowflake-arctic-embed-m`	`snowflake-arctic-embed-m`	通用文本嵌入(GTE)模型，专注于创建针对性能优化的高质量检索模型。
7	`embaas/sentence-transformers-e5-large-v2`	`sentence-transformers-e5-large-v2`	通用文本嵌入(GTE)模型：将句子和段落映射到1024维的密集向量空间，可用于聚类或语义搜索等任务。

这个表格简明地列出了支持的向量化模型及其介绍，帮助用户理解每个模型的特点和应用场景。

一些常用模型汇总： https://public.ukp.informatik.tu-darmstadt.de/reimers/sentence-transformers/v0.2/

支持的距离度量方式：

- `cosine`：余弦相似度
- `euclid`：欧几里得距离
- `dot`：点积
- `manhattan`：曼哈顿距离

---

4. 工作流程

1. **配置连接**：组件会尝试连接到本地或远程的 Qdrant 实例。如果没有提供 `host` 和 `port`，则会使用本地嵌入式 Qdrant 数据库。
2. **数据加载**：通过 `DATAConnect` 从数据流端口中读取输入数据流（如 PDF 文本或图像数据），并去重处理。
3. **数据向量化**：使用 Hugging Face 的 `transformers` 模型将文本或图像转换为向量。模型可配置，默认使用 `all_MiniLM_L6_v2` 模型。
4. **数据存储**：向量化的数据批量写入到 Qdrant 中，集合名称通过参数指定。写入完成后，显示上传进度及状态。
5. **检索支持**：写入后的数据可以通过 Qdrant 提供的接口进行检索，支持通过向量化搜索进行相似性查询。

---

5. 代码说明

主程序结构：

```
DATAConnectif __name__ == "__main__":
    # 获取组件配置参数，使用命令行参数或默认值进行赋值
    collection_name = sys.argv[1] if len(sys.argv) > 1 else
DEFAULT_COLLECTION_NAME
    batch_size = int(sys.argv[2]) if len(sys.argv) > 2 else DEFAULT_BATCH_SIZE
    host = sys.argv[3] if len(sys.argv) > 3 else DEFAULT_HOST
    port = int(sys.argv[4]) if len(sys.argv) > 4 else DEFAULT_PORT
    grpc_port = int(sys.argv[5]) if len(sys.argv) > 5 else DEFAULT_GRPC_PORT
    embed_model = sys.argv[6] if len(sys.argv) > 6 else DEFAULT_EMBED_MODEL
    distance_metric = sys.argv[7] if len(sys.argv) > 7 else
DEFAULT_DISTANCE_METRIC

    # 初始化 Qdrant 客户端
    if host and port:
        client = QdrantClient(host=host, port=port, grpc_port=grpc_port,
prefer_grpc=True, headers=DEFAULT_HEADERS)
    else:
        client = QdrantClient(path="./qdrant_local.db")

    # 数据输入和去重
    dataConnect = DATAConncet()
    df = dataConnect.dataInputStream(port="input_read")
    df.drop_duplicates(subset=["element_id"], inplace=True)

    # 向量化和写入
    write_dict(collection_name=collection_name, elements_dict=datas,
client=client, batch_size=batch_size)

    # 关闭客户端
    client.close()
```

关键函数：

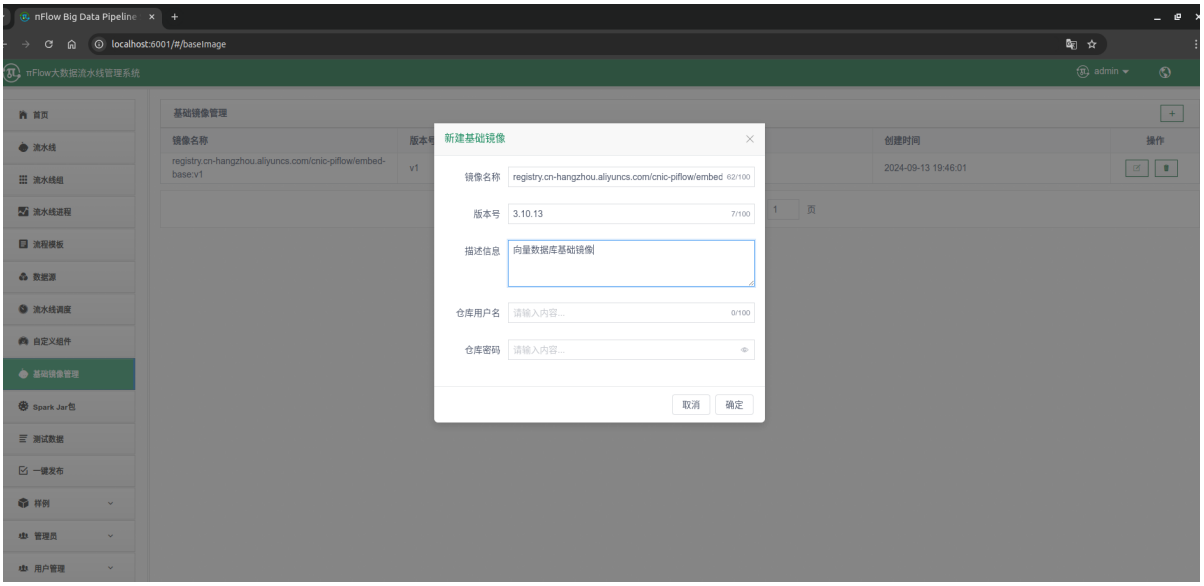
- **`write_dict`**：负责将数据转换为向量并批量写入 Qdrant。

```
def write_dict(collection_name: str, elements_dict: t.List[t.Dict[str, str]],
client: QdrantClient, batch_size: int):
    embedFunc = hfe(model_name=embed_model)

    def _embedText(s: str) -> t.List[float]:
        return embedFunc.embed_documents(texts=[s])[0]

    points = []
    for i in range(len(elements_dict)):
        content = elements_dict[i]
        vector = _embedText(str(content['text']))
        points.append(PointStruct(id=i, vector=vector, payload=content))

        if (i + 1) % batch_size == 0 or i == len(elements_dict) - 1:
            try:
```

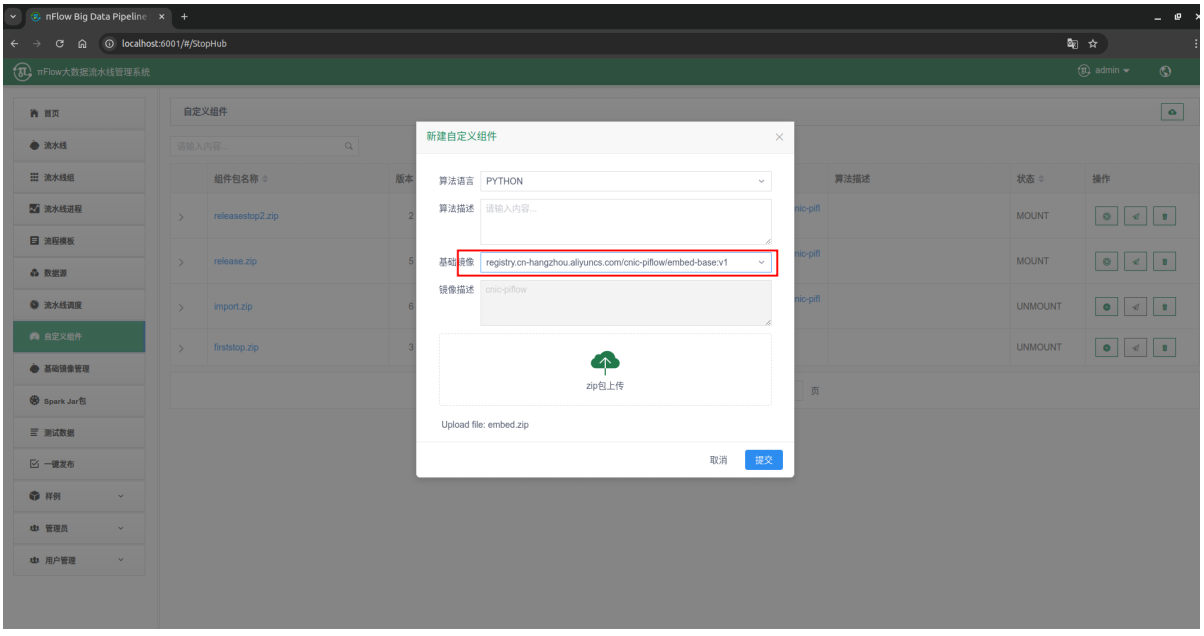
```
                client.upsert(collection_name=collection_name, points=points)
                points.clear()
            except Exception as e:
                print(f"Error during upsert: {e}")
```

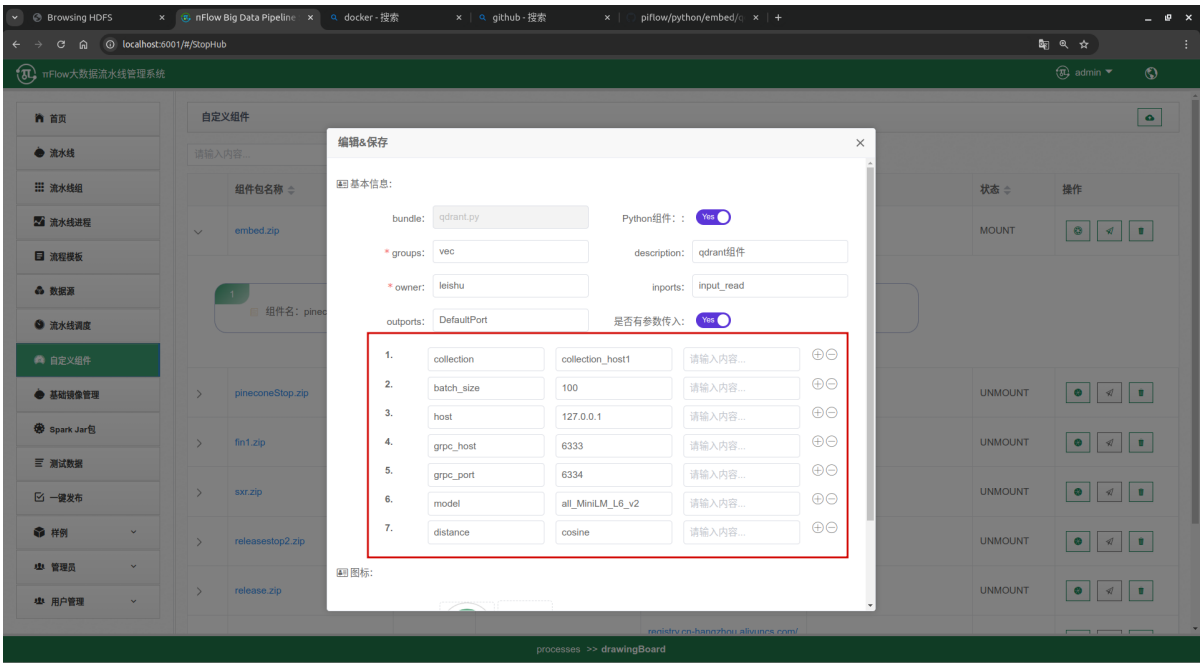
6. 使用示例

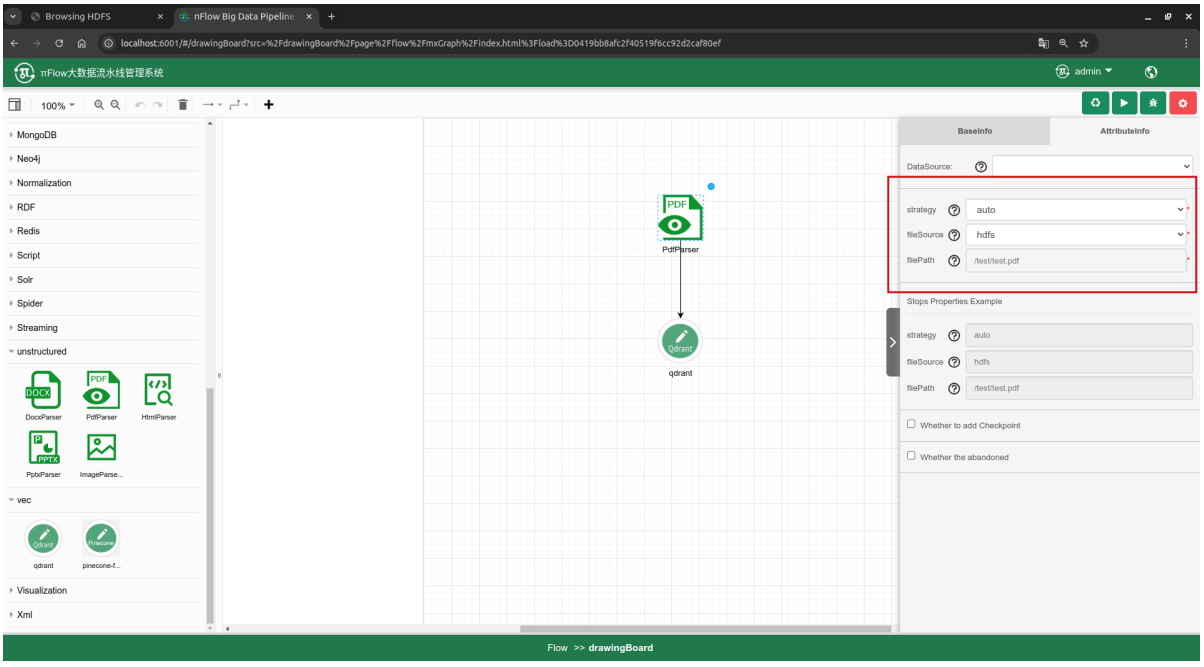
1）**配置基础镜像**：在基础镜像管理菜单中，可以选择已有镜像或从官方镜像拉取制定版本的python镜像（python本版3.10以上，此处我们设置基础镜像为 `registry.cn-hangzhou.aliyuncs.com/cnic-piflow/embed-base:v1`）。配置的具体步骤请参考下图：

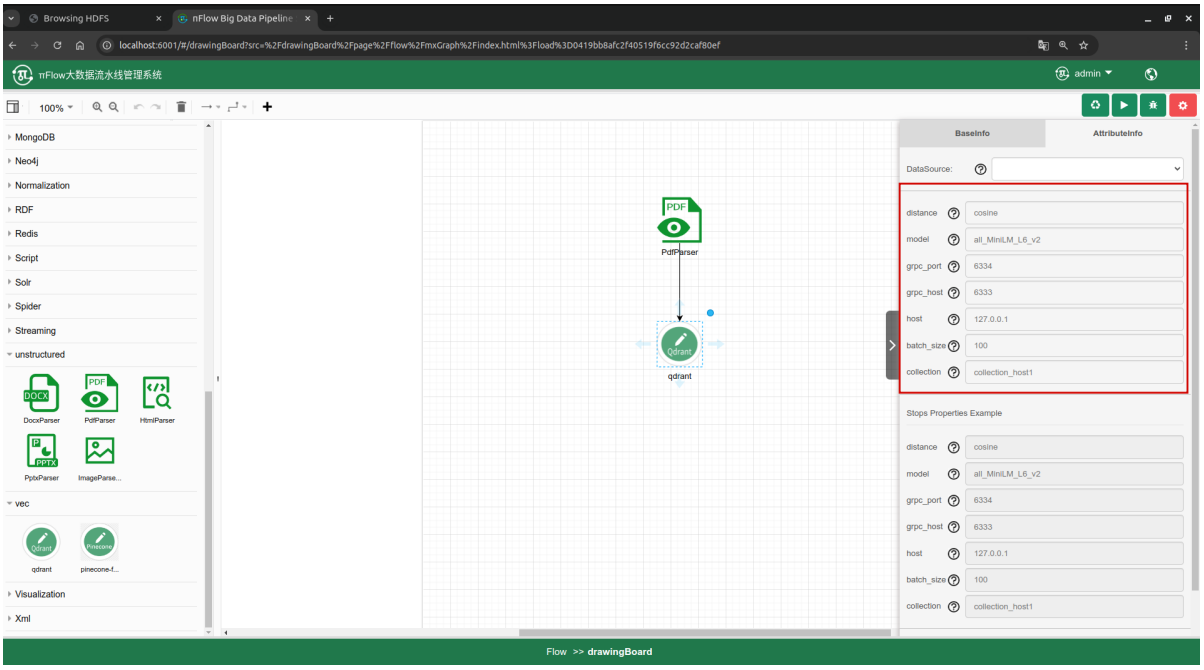


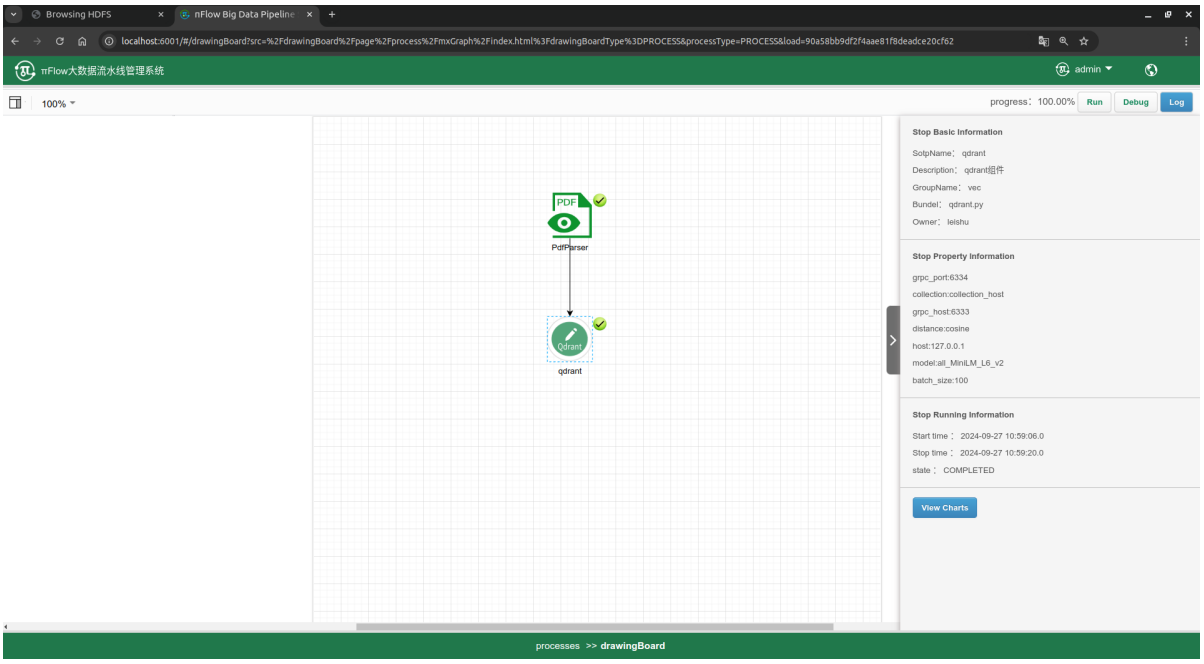
2）**安装向量数据库存储组件**：首先，从 GitHub 下载包含向量数据库存储组件的 ZIP 文件。然后，将 ZIP 文件上传到系统并进行挂载（mount）。挂载成功后，选择组件并编辑其基本信息和图标。配置的详细步骤请参考下图：




编辑&保存
基本信息:
bundle: qdrant.py
Python组件：: Yes
* groups: vec
description: qdrant组件
* owner: leishu
inports: input_read
outports: DefaultPort
是否有参数传入: Yes
1. collection collection_host1
2. batch_size 100
3. host 127.0.0.1
4. grpc_host 6333
5. grpc_port 6334
6. model all_MiniLM_L6_v2
7. distance cosine
图标:


PdfParser
qdrant
BaseInfo
AttributeInfo
DataSource:
strategy auto
fileSource hdfs
filePath /test/test.pdf
Stops Properties Example
Whether to add Checkpoint
Whether the abandoned
Flow >> drawingBoard


PdfParser
qdrant
distance cosine
model all_MiniLM_L6_v2
grpc_port 6334
grpc_host 6333
host 127.0.0.1
batch_size 100
collection collection_host1
Flow >> drawingBoard



7. 注意事项

- 运行前请确保 Qdrant 数据库已经启动并正确配置（本地或远程）。
- `batch_size` 应根据实际数据量设置，建议在大批量数据时适当调整该值以优化性能。
- 默认情况下，组件会尝试连接本地 Qdrant 实例；如果要连接远程实例，请确保提供正确的 `host` 和 `port`。
- **禁止将不同预训练模型得到的嵌入存储到同一个数据库中：** qdrant数据库在创建时要求设置向量维度,不同预训练模型获得的嵌入向量维度有所差异.选择了不同的预训练模型时,`collection_name`参数不能一致.

8. 扩展功能

- **模型切换**：通过修改 `embed_model` 参数可以选择不同的预训练模型，支持对不同类型的文本和图像进行向量化处理。
- **距离度量方式**：可以通过修改 `distance_metric` 参数切换不同的距离度量方法，如余弦相似度、欧几里得距离、曼哈顿距离等。
- **模型扩展：**关于文本嵌入模型还有很多可选择,可根据自身任务需要选择合适的预训练模型.(一些常用模型汇总:https://public.ukp.informatik.tu-darmstadt.de/reimers/sentence-transformers/v0.2/)

9. 结语

通过以上步骤，已经成功配置了所需的基础镜像，并将向量数据库存储组件集成到系统中。这些设置为处理和存储来自不同解析组件的非结构化数据提供了强大的支持，确保了数据在 Qdrant 向量数据库中的高效存储与检索。

在进行后续操作时，可以利用此组件来优化大模型训练的数据需求，提升数据检索与分析的效率。Qdrant 的向量化处理和存储能力将更好地管理和利用非结构化数据，为数据驱动的决策提供坚实的基础。

10. 代码地址

https://github.com/cas-bigdatalab/piflow/tree/master/python/embed/qdrant