# Руководство администратора
# РЕД Вектор Данных

## СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Руководство предназначено для использования администраторами РЕД Вектора Данных («глобальными администраторами»). Руководство администратора содержит важные инструкции и справочную информацию по темам, связанным с повседневными административными задачами.

Для работы с РЕД Вектор Данных необходимо установить последнюю версию Python. Если вы не знаете, как запустить этот код в виртуальной среде, сначала ознакомьтесь с документацией Python по созданию виртуальных сред. В этом руководстве предполагается, что вы находитесь в оболочке bash.

# 1 РАЗВЕРТЫВАНИЕ СИСТЕМЫ

## 1.1 Требования к установке

### 1.1.1 Хранилище

Для постоянного хранения данных РЕД Вектор Данных требует доступа на уровне блоков к устройствам хранения с файловой системой, совместимой с **POSIX**. Могут использоваться сетевые системы, такие как **iSCSI**, обеспечивающие доступ на уровне блоков. РЕД Вектор Данных не будет работать с сетевыми файловыми системами, такими как **NFS**, или объектными системами хранения, такими как **S3**. Если вы переносите векторные данные на локальный диск, мы рекомендуем использовать твердотельный накопитель (**SSD** или **NVMe**).

### 1.1.2 Требования к сети

Для работы каждого экземпляра РЕД Вектора Данных требуется три открытых порта:

6333- Для HTTP API, для конечных точек мониторинга состояния и метрик.

6334- Для gRPC API.

6335- Для распределенного развертывания.

Все экземпляры РЕД Вектор Данных в кластере должны взаимодействовать через эти порты, разрешите входящие соединения к портам 6333 и 6334 от клиентов, использующих РЕД Вектор Данных.

## 1.2 Установка и запуск РЕД Вектора Данных в составе Docker

### 1.2.1 Загрузка Docker-образа

Загрузить архив Docker-образа можно со съемного носителя либо по ссылке:

```
https://redv-dist.red-soft-center.ru/browser/server/docker%2Ftest_instance%2F
```

### 1.2.2 Импорт архива Docker-образа

После загрузки архива необходимо выполнить команду для импорта Docker образа:

```
docker load --input red_vector_db_image.tar.gz
```

### 1.2.3 Запуск сервера

Далее необходимо запустить контейнер с сервером:

```
docker run -d --name red-vector-db --restart=unless-stopped -p 6333:6333 -p 6334:6334 red-vector-db:latest
```

где:

- **-d** запуск контейнера в фоновом режиме;
- **--name red-v-server** задание имени контейнеру;
- **--restart=unless-stopped** автоматический перезапуск контейнера в случае остановки;
- **-p 6333:6333 -p 6334:6334** проброс портов 6333 и 6334 из контейнера на хост.

#### 1.2.4. Проверка успешного запуска сервера

Чтобы убедиться, что сервер успешно запущен, необходимо открыть веб-браузер и перейти по следующему адресу:

```
http://localhost:6333/
```

Если сервер запущен успешно, то откроется соответствующая веб-страница с информацией о сервере (Рисунок 1).

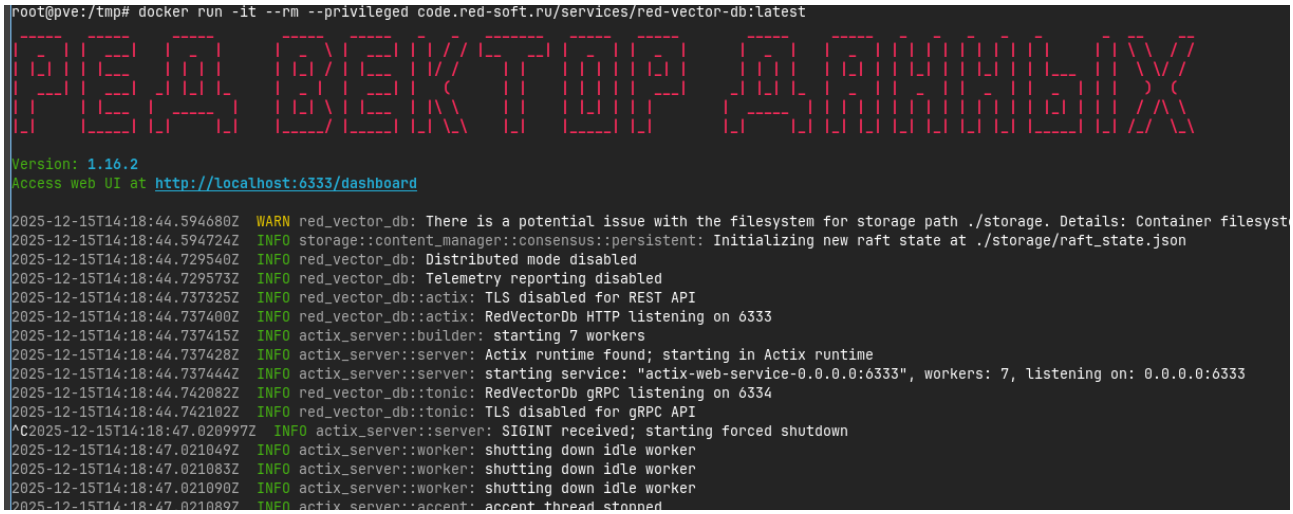


Рисунок 1 – Информация о сервере

## 1.3 Обновление РЕД Вектора Данных

### 1.3.1 Скачивание Docker-образа

Загрузите актуальный архив Docker-образа по следующей ссылке:

```
red_vector_db_image.tar.gz
```

### 1.3.2 Остановка и удаление контейнера и образа РЕД Вектора Данных

Проверьте, запущен ли контейнер со старой версией проекта:

```
docker ps
```

Если контейнер запущен, остановите его:

```
docker stop red-vector-db
```

Удалите контейнер старой версии:

```
docker rm red-vector-db
```

Удалите старый образ:

```
docker rmi red-vector-db
```

#### 1.3.3 Импорт нового Docker-образа

Выполните команду для импорта ранее скачанного Docker-образа:

```
docker load --input red_vector_db_mage.tar.gz
```

#### 1.3.4 Запуск сервера

Запустите новый контейнер:

```
docker run -d --name red-vector-db --restart=unless-stopped \
-p 6333:6333 -p 6334:6334 red-vector-db:latest
```

Разбор параметров:

**-d —** запускает контейнер в фоновом режиме;

**_-name red-vector-db —** задает имя контейнера;

**_-restart=unless-stopped —** автоматически перезапускает контейнер при сбое;

**-p 6333:6333 -p 6334:6334 —** пробрасывает порты 6333 и 6334 из контейнера на хост.

#### 1.3.5 Проверка успешного запуска сервера

Чтобы проверить, что сервер работает, откройте браузер и перейдите по адресу:

```
http://localhost:6333/
```

Если сервер успешно запущен, отобразится веб-страница с информацией о сервере.

# 2 ОБСЛУЖИВАНИЕ СИСТЕМЫ

## 2.1 Инструменты администрирования

Инструменты администрирования, позволяющие изменять поведение экземпляра РЕД Вектора Данных во время выполнения без необходимости вручную менять его конфигурацию.

### 2.1.1 Режим восстановления

Режим восстановления может помочь в ситуациях, когда РЕД Вектор Данных неоднократно не запускается. При запуске в режиме восстановления РЕД Вектор Данных загружает только метаданные коллекций, чтобы предотвратить нехватку памяти. Это позволяет, например, устранить проблемы с нехваткой памяти путем удаления коллекции. После устранения проблемы РЕД Вектор Данных можно перезапустить в обычном режиме для продолжения работы.

В режиме восстановления операции с коллекциями ограничиваются удалением коллекции. Это связано с тем, что во время восстановления загружаются только метаданные коллекции.

Для включения режима восстановления в образе Docker необходимо установить переменную среды **RED_VECTOR_DB_ALLOW_RECOVERY_MODE=true**. Контейнер сначала попытается запуститься в обычном режиме, а если инициализация завершится неудачей из-за ошибки нехватки памяти, перезапустится в режиме восстановления. По умолчанию это поведение отключено.

При использовании бинарного файла РЕД Вектор Данных режим восстановления можно включить, задав сообщение восстановления в переменной среды, например: **RED_VECTOR_DB__STORAGE__RECOVERY_MODE="My recovery message"**.

### 2.1.2 Строгий режим

Строгий режим позволяет ограничивать определенные типы операций с коллекцией в целях защиты кластера РЕД Вектор Данных. Цель строгого режима состоит в предотвращении неэффективных моделей использования, которые могут перегрузить систему.

Строгий режим обеспечивает более предсказуемую и быструю работу сервиса, когда вы не контролируете выполняемые запросы. При превышении лимита сервер вернет на стороне клиента сообщение об ошибке, содержащее информацию о превышенном лимите.

Это функция **strict_mode_config**, которую можно включить при создании новой коллекции:

| |
|---|
| **strict_mode_config** *object or any Optional* |
| **StrictModeConfig** *object Required* |

**Параметры функции:**

**enabled** boolean or null Optional

Включен ли строгий режим для коллекции или нет.

**max_query_limit** *integer or null Optional >=1*

Максимально допустимое значение параметра **limit** для всех API, для которых не установлен собственный максимальный лимит.

**max_timeout** *integer or null Optional >=1*

Максимально допустимый параметр **timeout.**

**unindexed_filtering_retrieve** *boolean or null Optional*

Разрешите использование неиндексированных полей в фильтрах, используемых для поиска (например, при поиске).

**unidexed_filtering_update** *boolean or null Optional*

Разрешите использование неиндексированных полей в фильтрованных обновлениях (например, при удалении по содержимому).

**search_max_hnsw_ef** *integer or null Optional >=0*

Максимально допустимое значение HNSW в параметрах поиска.

**search_allow_exact** *boolean or null Optional*

Разрешен ли точный поиск.

**search_max_oversampling** *double or null Optional*

Максимально допустимое значение передискретизации в поиске.

**upsert_max_batchsize** *integer or null Optional >=0*

Максимальный размер пакета при обновлении или вставке.

**max_collection_vector_size_bytes** *integer or null Optional >=0*

Максимальный размер хранилища векторных данных коллекции в байтах, без учета реплик.

**read_rate_limit** *integer or null Optional >=1*

Максимальное количество операций чтения в минуту на одну реплику.

**write_rate_limit** *integer or null Optional >=1*

Максимальное количество операций записи в минуту на одну реплику.

**max_collection_payload_size_bytes** *integer or null Optional >=0*

Максимальный размер хранилища полезной нагрузки коллекции в байтах.

**max_points_count** *integer or null Optional >=1*

Максимальное количество точек, оцененное в наборе.

**filter_max_conditions** *integer or null Optional >=0*

Максимальное количество условий, которое может иметь фильтр.

**multivector_config** *integer or null Optional >=0*

Многовекторная конфигурация строгого режима.

**sparse_config** *map from strings to objects or any Optional*

Конфигурация строгого режима разреженного вектора

**max_payload_index_count** *integer or null Optional >=0*

Максимальное количество индексов полезной нагрузки в коллекции

В рамках конфигурации это **enabled** поле выступает в качестве переключателя, позволяющего динамически включать или отключать строгий режим.

Можно увеличить значения по умолчанию и/или полностью отключить строгий режим. Однако для обеспечения стабильной работы кластера мы настоятельно рекомендуем оставить строгий режим включенным, используя его конфигурацию по умолчанию. Для отключения строгого режима в существующей коллекции используйте:

```
curl -X PUT http://localhost:6333/collections/{collection_name} \
  -H 'Content-Type: application/json' \
  --data-raw '{
    "strict_mode_config": {
      "enabled":" false
    }
  }'
```

### 2.1.3 Отключить получение данных с помощью неиндексированной полезной нагрузки

Установка **unindexed_filtering_retrieve** значения **false** предотвращает получение точек путем фильтрации по неиндексированному ключу полезной нагрузки, что может замедлить работу.

```
curl -X PUT http://localhost:6333/collections/{collection_name} \
  -H 'Content-Type: application/json' \
  --data-raw '{
```

```
  "strict_mode_config": {
    "enabled":" true,
    "unindexed_filtering_retrieve": false
  }
}'
```

Или же его можно отключить позже в существующей коллекции с помощью API обновления коллекции.

```
curl -X PUT http://localhost:6333/collections/{collection_name} \
 -H 'Content-Type: application/json' \
 --data-raw '{
  "strict_mode_config": {
    "enabled":" true,
    "unindexed_filtering_retrieve": true
  }
}'
```

#### 2.1.4 Отключить обновление через неиндексированную полезную нагрузку

Установка **unindexed_filtering_retrieve** значения **false** предотвращает получение точек путем фильтрации по неиндексированному ключу полезной нагрузки, что может замедлить работу.

```
curl -X PUT http://localhost:6333/collections/{collection_name} \
 -H 'Content-Type: application/json' \
 --data-raw '{
  "strict_mode_config": {
    "enabled":" true,
    "unindexed_filtering_update": false
  }
}'
```

#### 2.1.5 Максимальное количество индексов полезной нагрузки

Этот параметр **max_payload_index_count** ограничивает максимальное количество индексов полезной нагрузки, которые могут существовать в коллекции.

```
curl -X PUT http://localhost:6333/collections/{collection_name} \
```

```
 -H 'Content-Type: application/json' \
 --data-raw '{
   "strict_mode_config": {
     "enabled":" true,
     "max_payload_index_count": 10
   }
 }'
```

#### 2.1.6 Максимальное значение параметра запроса *limit*

Получение большого набора результатов — дорогостоящий процесс. Параметр **max_query_limitcaps** ограничивает максимальное количество точек, которые можно получить за один запрос.

```
curl -X PUT http://localhost:6333/collections/{collection_name} \
 -H 'Content-Type: application/json' \
 --data-raw '{
   "strict_mode_config": {
     "enabled":" true,
     "max_query_limit": 10
   }
 }'
```

#### 2.1.7 Максимальное значение параметра *timeout*

Параметр **max_timeout** ограничивает максимальное значение параметра в секундах для **timeout** всех операций API.

```
curl -X PUT http://localhost:6333/collections/{collection_name} \
 -H 'Content-Type: application/json' \
 --data-raw '{
   "strict_mode_config": {
     "enabled":" true,
     "max_timeout": 10
   }
 }'
```

#### 2.1.8 Максимальный размер условия фильтрации

Параметр **condition_max_size** ограничивает максимальное количество элементов, которое может содержать условие фильтрации.

```
curl -X PUT http://localhost:6333/collections/{collection_name} \
  -H 'Content-Type: application/json' \
  --data-raw '{
    "strict_mode_config": {
      "enabled":" true,
      "condition_max_size": 10
    }
  }'
```

#### 2.1.9 Максимальное количество условий в фильтре

Параметр **filter_max_conditions** ограничивает максимальное количество условий, которые могут быть у фильтров.

```
curl -X PUT http://localhost:6333/collections/{collection_name} \
  -H 'Content-Type: application/json' \
  --data-raw '{
    "strict_mode_config": {
      "enabled":" true,
      "filter_max_conditions": 10
    }
  }'
```

#### 2.1.10 Максимальный размер пакета при вставке векторов

Этот параметр **upsert_max_batchsize** ограничивает максимальный размер пакета в байтах во время операций вставки/обновления вектора.

```
curl -X PUT http://localhost:6333/collections/{collection_name} \
  -H 'Content-Type: application/json' \
  --data-raw '{
    "strict_mode_config": {
      "enabled":" true,
      "upsert_max_batchsize": 1000
    }
```

```
}'
```

### 2.1.11 Максимальный размер хранилища коллекции

Можно установить максимальный размер коллекции в виде векторов и/или размера хранилища полезной нагрузки. Устанавливает **max_collection_vector_size_bytes** и/или **max_collection_payload_size_bytes** ограничивает максимальный размер коллекции в байтах.

```
curl -X PUT http://localhost:6333/collections/{collection_name} \
  -H 'Content-Type: application/json' \
  --data-raw '{
    "strict_mode_config": {
      "enabled":" true,
      "max_collection_vector_size_bytes": 100000,
      "max_collection_payload_size_bytes": 100000
    }
  }'
```

### 2.1.12 Ограничение скорости

Установка **read_rate_limit** и/или **write_rate_limit** – ограничение максимального количества операций в минуту на одну реплику. При превышении максимального количества операций клиент получит код ошибки HTTP 429 с указанием рекомендуемой задержки перед повторной попыткой.

```
curl -X PUT http://localhost:6333/collections/{collection_name} \
  -H 'Content-Type: application/json' \
  --data-raw '{
    "strict_mode_config": {
      "enabled":" true,
      "read_rate_limit": 1000,
      "write_rate_limit": 100,
    }
  }'
```

## 2.2 Оптимизация производительности

В этом руководстве мы рассмотрим три основные стратегии оптимизации: быстрый поиск и низкое потребление памяти, высокая точность и низкое потребление памяти, высокоточный и высокоскоростной поиск.

### 2.2.1 Высокоскоростной поиск с низким потреблением памяти

Для достижения высокой скорости поиска при минимальном использовании памяти можно хранить векторы на диске, минимизируя при этом количество операций чтения с диска. Для хранения большого количества векторов в оперативной памяти, с использованием исходных векторов с диска необходимо создать коллекцию со следующими параметрами:

**on_disk** сохраняет исходные векторы на диске;

**quantization_config** сжимает квантованные векторы с **int8** помощью этого **scalar** метода;

**always_ram** хранит квантованные векторы в оперативной памяти.

```
PUT /collections/{collection_name}
{
  "vectors": {
    "size": 768,
    "distance": "Cosine",
    "on_disk": true
  },
  "quantization_config": {
    "scalar": {
      "type": "int8",
      "always_ram": true
    }
  }
}
```

### 2.2.2 Отключить переоценку результатов для ускорения поиска

Отключение переоценки результатов поиска **params** может дополнительно уменьшить количество операций чтения с диска. Обратите внимание, что это может немного снизить точность.

```
POST /collections/{collection_name}/points/query
{
    "query": [0.2, 0.1, 0.9, 0.7],
    "params": {
        "quantization": {
            "rescore": false
        }
    },
    "limit": 10
}
```

### 2.2.3 Высокая точность при низком потреблении памяти

Если требуется высокая точность, но ограничен объем оперативной памяти, можно хранить как векторы, так и индекс **HNSW** на диске. Такая конфигурация снижает потребление памяти, сохраняя при этом точность поиска.

Для хранения векторов **on_disk** необходимо настроить как сами векторы, так и индекс **HNSW**:

```
PUT /collections/{collection_name}
{
    "vectors": {
      "size": 768,
      "distance": "Cosine",
      "on_disk": true
    },
    "hnsw_config": {
        "on_disk": true
    }
}
```

### 2.2.4 Повышение точности

Увеличьте параметры **ef** и **m** индекса HNSW для повышения точности, даже при ограниченном объеме оперативной памяти:

```
...
```

```
"hnsw_config": {
   "m": 64,
   "ef_construct": 512,
   "on_disk": true
}
...
```

Примечание: Скорость этой конфигурации зависит от количества операций ввода-вывода в секунду (IOPS) диска. Для измерения IOPS диска можно использовать **fio**.

```
sudo apt update
sudo apt install -y fio lshw
sudo dnf install -y fio lshw
```

Для начала получите сводную информацию о вашей системе:

```
sudo lshw -short
```

Затем запустите тест производительности в папке, где смонтирован целевой диск:

```
cd /opt
sudo fio --profile=tiobench
```

### 2.2.5 Встроенное хранилище в индексе HNSW

При хранении векторов и индекса HNSW на диске можно повысить производительность поиска, включив соответствующую опцию **inline_storage** в файле **hnsw_config**. При использовании встроенного хранилища РЕД Вектор Данных хранит копии векторов непосредственно в файле индекса HNSW. Это ускоряет поиск за счет уменьшения количества операций ввода-вывода, но при этом увеличивает объем используемого хранилища в 3-4 раза. Для этого необходимо включить квантизацию.

```
PUT /collections/{collection_name}
{
   "vectors": {
     "size": 768,
     "distance": "Cosine",
     "on_disk": true
   },
   "quantization_config": {
      "binary": {
```

```
        "always_ram": false
      }
    },
    "hnsw_config": {
      "on_disk": true,
      "inline_storage": true
    }
}
```

### 2.2.6 Высокая точность и высокая скорость поиска

Совет**.** В сценариях, требующих как высокой скорости, так и высокой точности, храните как можно больше данных в оперативной памяти. Применяйте квантование с переоценкой для регулирования точности.

Вот как можно настроить скалярное квантование для коллекции:

```
PUT /collections/{collection_name}
{
    "vectors": {
      "size": 768,
      "distance": "Cosine"
    },
    "quantization_config": {
      "scalar": {
        "type": "int8",
        "always_ram": true
      }
    }
}
```

### 2.2.7 Точная настройка параметров поиска

Можно настроить параметры поиска, такие как **hnsw_ef** и **exact** чтобы найти баланс между скоростью и точностью.

**hnsw_ef** Количество соседей, которых необходимо посетить во время поиска (чем выше значение, тем выше точность, но ниже скорость).

**exact** Установите значение **true** для точного поиска, который выполняется медленнее, но точнее. Вы можете использовать его для сравнения результатов поиска с различными **hnsw_ef** значениями с эталонными данными.

```
POST /collections/{collection_name}/points/query
{
   "query": [0.2, 0.1, 0.9, 0.7],
   "params": {
      "hnsw_ef": 128,
      "exact": false
   },
   "limit": 3
}
```

### 2.2.8 Минимизация задержки

Чтобы минимизировать задержку, можно настроить РЕД Вектор Данных на использование максимально возможного количества ядер для одного запроса. Для этого установите количество сегментов в коллекции равным количеству ядер в системе.

```
PUT /collections/{collection_name}
{
   "vectors": {
     "size": 768,
     "distance": "Cosine"
   },
   "optimizers_config": {
      "default_segment_number": 16
   }
}
```

### 2.2.9 Максимизация пропускной способности

Совет. Для максимальной пропускной способности настройте РЕД Вектор Данных на использование как можно большего количества ядер для параллельной обработки нескольких запросов. Для этого используйте меньшее количество сегментов (обычно 2) большего размера (по умолчанию 200 МБ на сегмент), чтобы обрабатывать больше запросов параллельно.

Крупные сегменты выигрывают от размера индекса и в целом меньшего количества сравнений векторов, необходимых для поиска ближайших соседей. Однако для построения индекса HNSW им потребуется больше времени.

```
PUT /collections/{collection_name}
{
   "vectors": {
     "size": 768,
     "distance": "Cosine"
   },
   "optimizers_config": {
      "default_segment_number": 2,
      "max_segment_size": 5000000
   }
}
```

## 2.3 Планирование оперативной памяти и дискового пространства

### 2.3.1 Расчет размера оперативной памяти

Для более быстрого доступа к данным следует хранить их в оперативной памяти. Если вы хотите хранить все векторы в памяти для оптимальной производительности, вы можете использовать следующую приблизительную формулу для оценки:

```
memory_size = number_of_vectors * vector_dimension * 4 bytes * 1.5
```

В итоге мы умножаем все на 1,5. Эти дополнительные 50% учитывают метаданные (такие как индексы и точечные версии) и временные сегменты, созданные в процессе оптимизации.

Допустим, вам нужно хранить 1 миллион векторов с 1024 измерениями:

```
memory_size = 1,000,000 * 1024 * 4 bytes * 1.5
```

Размер памяти составляет приблизительно 6 144 000 000 байт, или около 5,72 ГБ.

В зависимости от сценария использования, для больших наборов данных может быть полезно сократить объем памяти за счет квантования.

### 2.3.2 Расчет размера полезной нагрузки

Размер полезной нагрузки зависит от структуры и содержания ваших данных.

*Текстовые поля* занимают место в зависимости от длины и кодировки (например, большой фрагмент текста по сравнению с несколькими словами).

*Числа с плавающей запятой* имеют фиксированный размер 8 байт для **int64** или **float64**.

*Логические поля* обычно занимают 1 байт.

Совет. Самый простой способ рассчитать размер полезной нагрузки — использовать калькулятор размера JSON.

Расчет общего размера полезной нагрузки аналогичен расчету векторов. Для процессов индексирования на стороне сервера нам необходимо умножить его на 1,5.

```
total_payload_size = number_of_points * payload_size * 1.5
```

Допустим, вам нужно сохранить 1 миллион точек в формате JSON размером 5 КБ:

```
total_payload_size = 1,000,000 * 5KB * 1.5
```

Общий размер полезной нагрузки составляет приблизительно 5 000 000 байт, или около 4,77 ГБ.

### 2.3.3 Выбор между дисковых хранилищем и оперативной памятью

Для оптимальной производительности в оперативной памяти следует хранить только часто используемые данные. Остальные следует выгружать на диск. Например, дополнительные поля полезной нагрузки, которые не используются для фильтрации, можно хранить на диске. В оперативной памяти следует хранить только индексированные поля.

Если ваша приоритетная задача — обработка больших объемов векторов со средней задержкой поиска, рекомендуется настроить отображение в память (**mmap**). В этой конфигурации векторы хранятся на диске в файлах, отображаемых в память, а в оперативной памяти кэшируются только наиболее часто используемые векторы. Объем доступной оперативной памяти существенно влияет на производительность поиска. Как правило, если хранить в оперативной памяти вдвое меньше векторов, задержка поиска примерно удвоится.

Если в вашем сценарии использования требуется разделение векторов на несколько коллекций или подгрупп на основе значений полезной нагрузки (например, обработка поисковых запросов для нескольких пользователей, каждый со своим собственным подмножеством векторов), рекомендуется использовать хранилище, отображаемое в память. В этом сценарии в оперативной памяти будет кэшироваться только активное подмножество векторов, что позволит быстро находить самых последних и активных пользователей. Необходимый объем памяти можно оценить следующим образом:

```
memory_size = number_of_active_vectors * vector_dimension * 4 bytes * 1.5
```

## 2.4 Конфигурация, порядок и приоритет загрузки

### 2.4.1 Файлы конфигурации

Для настройки РЕД Вектор Данных для монтирования конфигурационного файла используются **.yaml** файлы.

#### 2.4.1.1 Основная конфигурация: red_vector_db/config/config.yaml

Подключите свой пользовательский **config.yaml** файл, чтобы переопределить настройки по умолчанию:

```
docker run -p 6333:6333 \
    -v $(pwd)/config.yaml:/red_vector_db/config/config.yaml \
    red_vector_db/red_vector_db
```

#### 2.4.1.2 Конфигурация, специфичная для конкретной среды: config/{RUN_MODE}.yaml

РЕД Вектор Данных ищет конфигурационный файл, специфичный для конкретной среды, на основе переменной **RUN_MODE**. По умолчанию образ Docker использует **RUN_MODE=production**, то есть он будет искать **config/production.yaml**.

Вы можете переопределить это, установив **RUN_MODE** другое значение (например, **dev**), и указав соответствующий файл:

```
docker run -p 6333:6333 \
    -v $(pwd)/dev.yaml:/red_vector_db/config/dev.yaml \
    -e RUN_MODE=dev \
    red_vector_db/red_vector_db:
```

#### 2.4.1.3 Локальная конфигурация: config/local.yaml

Этот **local.yaml** файл обычно используется для хранения настроек, специфичных для конкретного компьютера, которые не отслеживаются в системе контроля версий:

```
docker run -p 6333:6333 \
    -v $(pwd)/local.yaml:/red_vector_db/config/local.yaml \
    red_vector_db/red_vector_db
```

#### 2.4.1.4 Пользовательская настройка через--config-path

Вы можете указать путь к пользовательскому файлу конфигурации, используя **--config-path** аргумент. Это переопределит другие файлы конфигурации:

```
docker run -p 6333:6333 \
    -v $(pwd)/config.yaml:/path/to/config.yaml \
```

```
  red_vector_db/red_vector_db \
  ./red_vector_db --config-path /path/to/config.yaml
```

### 2.4.2 Переменные окружающей среды

Вы также можете настроить РЕД Вектор Данных с помощью переменных среды, которые всегда имеют наивысший приоритет и переопределяют любые настройки, основанные на файлах. Переменные окружения имеют следующий формат: они должны начинаться с символа **RED_VECTOR_DB**, а вложенные свойства должны разделяться двойными подчеркиваниями (__).

Пример.

```
docker run -p 6333:6333 \
  -e RED_VECTOR_DB__LOG_LEVEL=INFO \
  -e RED_VECTOR_DB__SERVICE__API_KEY=<MY_SECRET_KEY> \
  -e RED_VECTOR_DB__SERVICE__ENABLE_TLS=1 \
  -e RED_VECTOR_DB__TLS__CERT=./tls/cert.pem \
  red_vector_db/red_vector_db
```

В результате получается следующая конфигурация:

```
log_level: INFO
service:
  enable_tls: true
  api_key: <MY_SECRET_KEY>
tls:
  cert: ./tls/cert.pem
```

### 2.4.3 Порядок и приоритет загрузки

При запуске РЕД Вектор Данных объединяет несколько источников конфигурации в одну эффективную конфигурацию. Порядок загрузки следующий (от наименее значимого к наиболее значимому):

- встроенная конфигурация по умолчанию;
- config/config.yaml;
- config/{RUN_MODE}.yaml;
- config/local.yaml;
- пользовательский файл конфигурации;

– переменные окружающей среды.

### 2.4.4 Преобладающее поведение

Настройки из более поздних источников в списке имеют приоритет над настройками из более ранних источников.

Настройки в config/{RUN_MODE}.yaml переопределят настройки в config/config.yaml.

Пользовательский файл конфигурации, предоставленный через --config-path, переопределит все остальные настройки, основанные на файлах.

Переменные среды имеют наивысший приоритет и будут переопределять любые настройки из файлов.

### 2.4.5 Проверка конфигурации

РЕД Вектор Данных проверяет конфигурацию во время запуска. Если обнаруживаются какие-либо проблемы, сервер немедленно завершает работу, предоставляя информацию об ошибке.

Пример. Error: invalid type: 64-bit integer `-1`, expected an unsigned 64-bit or smaller integer for key `storage.hnsw_index.max_indexing_threads` in config/production.yaml

Это гарантирует раннее выявление ошибок конфигурации, предотвращая запуск РЕД Вектор Данных с недопустимыми настройками.

### 2.4.6 Параметры конфигурации

В следующем примере в формате YAML описаны доступные параметры конфигурации.

```
log_level: INFO

# Logging configuration
# Red_vector_db logs to stdout. You may configure to also write logs to a file on disk.
# Be aware that this file may grow indefinitely.
# logger:
#   # Logging format, supports `text` and `json`
#   format: text
#   on_disk:
#     enabled: true
#     log_file: path/to/log/file.log
```

```
#    log_level: INFO
#    # Logging format, supports `text` and `json`
#    format: text
#    buffer_size_bytes: 1024

storage:
  # Where to store all the data
  storage_path: ./storage

  # Where to store snapshots
  snapshots_path: ./snapshots

  snapshots_config:
    # "local" or "s3" - where to store snapshots
    snapshots_storage: local
    # s3_config:
    #   bucket: ""
    #   region: ""
    #   access_key: ""
    #   secret_key: ""

  # Where to store temporary files
  # If null, temporary snapshots are stored in: storage/snapshots_temp/
  temp_path: null

  # If true - point payloads will not be stored in memory.
  # It will be read from the disk every time it is requested.
  # This setting saves RAM by (slightly) increasing the response time.
  # Note: those payload values that are involved in filtering and are indexed - remain in RAM.
  #
  # Default: true
```

```
on_disk_payload: true

# Maximum number of concurrent updates to shard replicas
# If `null` - maximum concurrency is used.
update_concurrency: null

# Write-ahead-log related configuration
wal:
  # Size of a single WAL segment
  wal_capacity_mb: 32

  # Number of WAL segments to create ahead of actual data requirement
  wal_segments_ahead: 0

# Normal node - receives all updates and answers all queries
node_type: "Normal"

# Listener node - receives all updates, but does not answer search/read queries
# Useful for setting up a dedicated backup node
# node_type: "Listener"

performance:
  # Number of parallel threads used for search operations. If 0 - auto selection.
  max_search_threads: 0

  # CPU budget, how many CPUs (threads) to allocate for an optimization job.
  # If 0 - auto selection, keep 1 or more CPUs unallocated depending on CPU size
  # If negative - subtract this number of CPUs from the available CPUs.
  # If positive - use this exact number of CPUs.
  optimizer_cpu_budget: 0
```

```
  # Prevent DDoS of too many concurrent updates in distributed mode.
  # One external update usually triggers multiple internal updates, which breaks internal
  # timings. For example, the health check timing and consensus timing.
  # If null - auto selection.
  update_rate_limit: null

  # Limit for number of incoming automatic shard transfers per collection on this node, does not affect user-requested transfers.
  # The same value should be used on all nodes in a cluster.
  # Default is to allow 1 transfer.
  # If null - allow unlimited transfers.
  #incoming_shard_transfers_limit: 1

  # Limit for number of outgoing automatic shard transfers per collection on this node, does not affect user-requested transfers.
  # The same value should be used on all nodes in a cluster.
  # Default is to allow 1 transfer.
  # If null - allow unlimited transfers.
  #outgoing_shard_transfers_limit: 1

  # Enable async scorer which uses io_uring when rescoring.
  # Only supported on Linux, must be enabled in your kernel.
  # See: <https://red_vector_db.tech/articles/io_uring/#and-what-about-red_vector_db>
  #async_scorer: false

optimizers:
  # The minimal fraction of deleted vectors in a segment, required to perform segment optimization
  deleted_threshold: 0.2

  # The minimal number of vectors in a segment, required to perform segment optimization
  vacuum_min_vector_number: 1000
```

```
  # Target amount of segments optimizer will try to keep.
  # Real amount of segments may vary depending on multiple parameters:
  #  - Amount of stored points
  #  - Current write RPS
  #
  # It is recommended to select default number of segments as a factor of the number of search threads,
  # so that each segment would be handled evenly by one of the threads.
  # If `default_segment_number = 0`, will be automatically selected by the number of available CPUs
  default_segment_number: 0

  # Do not create segments larger this size (in KiloBytes).
  # Large segments might require disproportionately long indexation times,
  # therefore it makes sense to limit the size of segments.
  #
  # If indexation speed have more priority for your - make this parameter lower.
  # If search speed is more important - make this parameter higher.
  # Note: 1Kb = 1 vector of size 256
  # If not set, will be automatically selected considering the number of available CPUs.
  max_segment_size_kb: null

  # Maximum size (in KiloBytes) of vectors allowed for plain index.
  # Default value based on experiments and observations.
  # Note: 1Kb = 1 vector of size 256
  # To explicitly disable vector indexing, set to `0`.
  # If not set, the default value will be used.
  indexing_threshold_kb: 10000

  # Interval between forced flushes.
  flush_interval_sec: 5
```

```
  # Max number of threads (jobs) for running optimizations per shard.
  # Note: each optimization job will also use `max_indexing_threads` threads by itself for index building.
  # If null - have no limit and choose dynamically to saturate CPU.
  # If 0 - no optimization threads, optimizations will be disabled.
  max_optimization_threads: null

  # This section has the same options as 'optimizers' above. All values specified here will overwrite the collections
  # optimizers configs regardless of the config above and the options specified at collection creation.
  # optimizers_overwrite:
  #  deleted_threshold: 0.2
  #  vacuum_min_vector_number: 1000
  #  default_segment_number: 0
  #  max_segment_size_kb: null
  #  indexing_threshold_kb: 10000
  #  flush_interval_sec: 5
  #  max_optimization_threads: null

  # Default parameters of HNSW Index. Could be overridden for each collection or named vector individually
  hnsw_index:
    # Number of edges per node in the index graph. Larger the value - more accurate the search, more space required.
    m: 16

    # Number of neighbours to consider during the index building. Larger the value - more accurate the search, more time required to build index.
    ef_construct: 100

    # Minimal size threshold (in KiloBytes) below which full-scan is preferred over HNSW search.
    # This measures the total size of vectors being queried against.
    # When the maximum estimated amount of points that a condition satisfies is smaller than
```

```
  # `full_scan_threshold_kb`, the query planner will use full-scan search instead of HNSW index
  # traversal for better performance.
  # Note: 1Kb = 1 vector of size 256
  full_scan_threshold_kb: 10000

  # Number of parallel threads used for background index building.
  # If 0 - automatically select.
  # Best to keep between 8 and 16 to prevent likelihood of building broken/inefficient HNSW graphs.
  # On small CPUs, less threads are used.
  max_indexing_threads: 0

  # Store HNSW index on disk. If set to false, index will be stored in RAM. Default: false
  on_disk: false

  # Custom M param for hnsw graph built for payload index. If not set, default M will be used.
  payload_m: null

# Default shard transfer method to use if none is defined.
# If null - don't have a shard transfer preference, choose automatically.
# If stream_records, snapshot or wal_delta - prefer this specific method.
# More info: https://red_vector_db.tech/documentation/guides/distributed_deployment/#shard-transfer-method
shard_transfer_method: null

# Default parameters for collections
collection:
  # Number of replicas of each shard that network tries to maintain
  replication_factor: 1

  # How many replicas should apply the operation for us to consider it successful
  write_consistency_factor: 1
```

```
# Default parameters for vectors.
vectors:
  # Whether vectors should be stored in memory or on disk.
  on_disk: null

# shard_number_per_node: 1

# Default quantization configuration.
# More info: https://red_vector_db.tech/documentation/guides/quantization
quantization: null

# Default strict mode parameters for newly created collections.
#strict_mode:
  # Whether strict mode is enabled for a collection or not.
  #enabled: false

  # Max allowed `limit` parameter for all APIs that don't have their own max limit.
  #max_query_limit: null

  # Max allowed `timeout` parameter.
  #max_timeout: null

  # Allow usage of unindexed fields in retrieval based (eg. search) filters.
  #unindexed_filtering_retrieve: null

  # Allow usage of unindexed fields in filtered updates (eg. delete by payload).
  #unindexed_filtering_update: null

  # Max HNSW value allowed in search parameters.
  #search_max_hnsw_ef: null
```

```
    # Whether exact search is allowed or not.
    #search_allow_exact: null

    # Max oversampling value allowed in search.
    #search_max_oversampling: null

  # Maximum number of collections allowed to be created
  # If null - no limit.
  max_collections: null

service:
  # Maximum size of POST data in a single request in megabytes
  max_request_size_mb: 32

  # Number of parallel workers used for serving the api. If 0 - equal to the number of available cores.
  # If missing - Same as storage.max_search_threads
  max_workers: 0

  # Host to bind the service on
  host: 0.0.0.0

  # HTTP(S) port to bind the service on
  http_port: 6333

  # gRPC port to bind the service on.
  # If `null` - gRPC is disabled. Default: null
  # Comment to disable gRPC:
  grpc_port: 6334

  # Enable CORS headers in REST API.
```

```
# If enabled, browsers would be allowed to query REST endpoints regardless of query origin.
# More info: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
# Default: true
enable_cors: true

# Enable HTTPS for the REST and gRPC API
enable_tls: false

# Check user HTTPS client certificate against CA file specified in tls config
verify_https_client_certificate: false

# Set an api-key.
# If set, all requests must include a header with the api-key.
# example header: `api-key: <API-KEY>`
#
# If you enable this you should also enable TLS.
# (Either above or via an external service like nginx.)
# Sending an api-key over an unencrypted channel is insecure.
#
# Uncomment to enable.
# api_key: your_secret_api_key_here

# Set an api-key for read-only operations.
# If set, all requests must include a header with the api-key.
# example header: `api-key: <API-KEY>`
#
# If you enable this you should also enable TLS.
# (Either above or via an external service like nginx.)
# Sending an api-key over an unencrypted channel is insecure.
#
# Uncomment to enable.
```

```
  # read_only_api_key: your_secret_read_only_api_key_here

  # Uncomment to enable JWT Role Based Access Control (RBAC).
  # If enabled, you can generate JWT tokens with fine-grained rules for access control.
  # Use generated token instead of API key.
  #
  # jwt_rbac: true

  # Hardware reporting adds information to the API responses with a
  # hint on how many resources were used to execute the request.
  #
  # Warning: experimental, this feature is still under development and is not supported yet.
  #
  # Uncomment to enable.
  # hardware_reporting: true
  #
  # Uncomment to enable.
  # Prefix for the names of metrics in the /metrics API.
  # metrics_prefix: red_vector_db_

cluster:
  # Use `enabled: true` to run Red_vector_db in distributed deployment mode
  enabled: false

  # Configuration of the inter-cluster communication
  p2p:
    # Port for internal communication between peers
    port: 6335

    # Use TLS for communication between peers
    enable_tls: false
```

```
  # Configuration related to distributed consensus algorithm
consensus:
  # How frequently peers should ping each other.
  # Setting this parameter to lower value will allow consensus
  # to detect disconnected nodes earlier, but too frequent
  # tick period may create significant network and CPU overhead.
  # We encourage you NOT to change this parameter unless you know what you are doing.
  tick_period_ms: 100

  # Compact consensus operations once we have this amount of applied
  # operations. Allows peers to join quickly with a consensus snapshot without
  # replaying a huge amount of operations.
  # If 0 - disable compaction
  compact_wal_entries: 128

# Set to true to prevent service from sending usage statistics to the developers.
# Read more: https://red_vector_db.tech/documentation/guides/telemetry
telemetry_disabled: false

# TLS configuration.
# Required if either service.enable_tls or cluster.p2p.enable_tls is true.
tls:
  # Server certificate chain file
  cert: ./tls/cert.pem

  # Server private key file
  key: ./tls/key.pem

  # Certificate authority certificate file.
  # This certificate will be used to validate the certificates
```

```
# presented by other nodes during inter-cluster communication.
#
# If verify_https_client_certificate is true, it will verify
# HTTPS client certificate
#
# Required if cluster.p2p.enable_tls is true.
ca_cert: ./tls/cacert.pem

# TTL in seconds to reload certificate from disk, useful for certificate rotations.
# Only works for HTTPS endpoints. Does not support gRPC (and intra-cluster communication).
# If `null` - TTL is disabled.
cert_ttl: 3600
```

## 2.5 Параметры безопасности

Перед использованием в производственной среде необходимо включить меры безопасности. В противном случае они будут полностью доступны для всех.

### 2.5.1 Аутентификация

РЕД Вектор Данных поддерживает простую форму аутентификации клиента с использованием статического ключа API. Для включения аутентификации на основе API-ключа необходимо указать ключ в конфигурации:

```
service:
  # Set an api-key.
  # If set, all requests must include a header with the api-key.
  # example header: `api-key: <API-KEY>`
  #
  # If you enable this you should also enable TLS.
  # (Either above or via an external service like nginx.)
  # Sending an api-key over an unencrypted channel is insecure.
  api_key: your_secret_api_key_here
```

Или же вы можете использовать переменную окружения:

```
docker run -p 6333:6333 \
```

```
  -e RED_VECTOR_DB__SERVICE__API_KEY=your_secret_api_key_here \
  red_vector_db/red_vector_db
```

**Важно! Для предотвращения утечки ключа API через незашифрованное соединение необходимо использовать протокол TLS.**

Затем ключ API должен присутствовать во всех **REST** или **gRPC** запросах к вашему экземпляру.

```
curl \
  -X GET https://localhost:6333 \
  --header 'api-key: your_secret_api_key_here'
```

**Важно! Внутренние каналы связи никогда не защищаются ключом API или токенами носителя. Внутренний gRPC по умолчанию использует порт 6335 при работе в распределенном режиме. Необходимо убедиться, что этот порт недоступен извне и может использоваться только для связи между узлами.**

### 2.5.2 Ключ API только для чтения

В дополнение к обычному ключу API, РЕД Вектор Данных также поддерживает ключ API только для чтения. Этот ключ можно использовать для доступа к операциям только для чтения в экземпляре.

```
service:
  read_only_api_key: your_secret_read_only_api_key_here
```

Или с помощью переменной окружения:

```
export RED_VECTOR_DB__SERVICE__READ_ONLY_API_KEY=your_secret_read_only_api_key_here
```

Оба API-ключа можно использовать одновременно.

### 2.5.3 Дозированный контроль доступа с помощью JWT

В более сложных случаях РЕД Вектор Данных поддерживает детальный контроль доступа с помощью JSON Web Tokens (JWT). Это позволяет создавать токены, ограничивающие доступ к данным, хранящимся в вашем кластере, и на их основе строить систему управления доступом на основе ролей (RBAC). Таким образом, вы можете определять разрешения для пользователей и ограничивать доступ к конфиденциальным конечным точкам.

Для включения аутентификации на основе JWT в вашем экземпляре РЕД Вектор Данных необходимо указать **api-key** и активировать эту **jwt_rbac** функцию в конфигурации:

```
service:
```

```
api_key: you_secret_api_key_here
jwt_rbac: true
```

Или с использованием переменных окружения:

```
export RED_VECTOR_DB__SERVICE__API_KEY=your_secret_api_key_here
export RED_VECTOR_DB__SERVICE__JWT_RBAC=true
```

Настройки, **api_key** заданные в конфигурации, будут использоваться для кодирования и декодирования JWT, поэтому, разумеется, необходимо обеспечить их безопасность. В случае внесения **api_key** изменений все существующие токены станут недействительными.

Для использования аутентификации на основе JWT необходимо указать его в качестве токена носителя в **Authorization** заголовке запроса или в качестве ключа в **Api-Key** заголовке запроса.

```
Authorization: Bearer <JWT>
// or
Api-Key: <JWT>
```

### 2.5.4 Генерация JSON-токенов для веб-приложений

В силу особенностей JWT, любой, кто знаком с этой технологией, **api_key** может генерировать токены, используя любые существующие библиотеки и инструменты; для этого ему не обязательно иметь доступ к экземпляру РЕД Вектор Данных.

Заголовок **JWT** — РЕД Вектор Данных использует **HS256** алгоритм для декодирования токенов.

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Полезная нагрузка **JWT** — вы можете включить в полезную нагрузку любую комбинацию доступных параметров.

```
{
  "exp": 1640995200, // Expiration time
  "value_exists": ..., // Validate this token by looking for a point with a payload value
  "access": "r", // Define the access level.
```

```
}
```

Подписание токена — чтобы подтвердить действительность сгенерированного токена, его необходимо подписать с помощью токена, **api_key** указанного в конфигурации. Это означает, что тот, кто знает **api_key** токен, дает разрешение на его использование в экземпляре РЕД Вектор Данных. РЕД Вектор Данных может проверить подпись, поскольку знает **api_key** токен и может его расшифровать.

Процесс генерации токенов может выполняться на стороне клиента в автономном режиме и не требует взаимодействия с экземпляром РЕД Вектор Данных.

Вот пример использования **jwt-cli**:

```
jwt encode --payload '{
  "access": "r",
  "exp": 1766055305
}' --secret 'your-api-key'
```

### 2.5.5 Протокол TLS

Для защиты соединений в вашем экземпляре РЕД Вектор Данных можно включить протокол **TLS** для зашифрованных соединений.

Для начала убедитесь, что у вас есть сертификат и закрытый ключ для **TLS**, обычно в **.pem** формате. На локальном компьютере вы можете использовать **mkcert** для генерации самоподписанного сертификата.

Для включения **TLS** установите следующие параметры в конфигурации РЕД Вектор Данных, указав правильные пути, и перезапустите систему:

```
service:
  # Enable HTTPS for the REST and gRPC API
  enable_tls: true

# TLS configuration.
# Required if either service.enable_tls or cluster.p2p.enable_tls is true.
tls:
  # Server certificate chain file
  cert: ./tls/cert.pem

  # Server private key file
```

```
key: ./tls/key.pem
```

Для внутренней связи в кластерном режиме TLS можно включить следующим образом:

```
cluster:
  # Configuration of the inter-cluster communication
  p2p:
    # Use TLS for communication between peers
    enable_tls: true
```

При включенном TLS необходимо начать использовать HTTPS-соединения.

Пример.

```
curl -X GET https://localhost:6333
```

Функция ротации сертификатов включена с интервалом обновления по умолчанию в один час. Это означает, что файлы сертификатов перезагружаются каждый час во время работы РЕД Вектор Данных. Таким образом, измененные сертификаты подхватываются при их внешнем обновлении. Время обновления можно настроить, изменив соответствующий параметр **tls.cert_ttl**. Можно оставить эту функцию включенной, даже если не планируется обновлять свои сертификаты. В настоящее время это поддерживается только для **REST API**.

При желании можно включить проверку клиентского сертификата на сервере с использованием локального центра сертификации. Установите следующие параметры и перезапустите сервер:

```
service:
  # Check user HTTPS client certificate against CA file specified in tls config
  verify_https_client_certificate: false

# TLS configuration.
# Required if either service.enable_tls or cluster.p2p.enable_tls is true.
tls:
  # Certificate authority certificate file.
  # This certificate will be used to validate the certificates
  # presented by other nodes during inter-cluster communication.
  #
```

```
# If verify_https_client_certificate is true, it will verify
# HTTPS client certificate
#
# Required if cluster.p2p.enable_tls is true.
ca_cert: ./tls/cacert.pem
```

# 3 ИСПОЛЬЗОВАНИЕ БАЗЫ ДАННЫХ

## 3.1 МАССОВАЯ ЗАГРУЗКА ВЕКТОРОВ В КОЛЛЕКЦИЮ

При загрузке большого набора данных следует отдавать предпочтение высокопроизводительной клиентской библиотеке. Для этой цели мы рекомендуем использовать клиентскую библиотеку на Rust, поскольку это самая быстрая клиентская библиотека, доступная для РЕД Вектор Данных. Если вы не используете Rust, возможно, вам стоит рассмотреть возможность распараллеливания процесса загрузки.

### 3.1.1 ВЫБЕРИТЕ СТРАТЕГИЮ ИНДЕКСИРОВАНИЯ

РЕД Вектор Данных постепенно формирует индекс HNSW для плотных векторов по мере поступления новых данных. Это обеспечивает быстрый поиск, но индексирование требует больших объемов памяти и ресурсов процессора. При пакетной загрузке частые обновления индекса могут снизить пропускную способность и увеличить потребление ресурсов.

Для управления этим поведением и оптимизации с учетом ограничений вашей системы, отрегулируйте следующие параметры:

| **Ваша цель** | **Что делать** | **Конфигурация** |
|---|---|---|
| Максимальная скорость загрузки, допускает высокое потребление оперативной памяти. | Полностью отключить индексирование | indexing_threshold: ⊙ |
| Низкий уровень потребления памяти во время загрузки. | Отложить построение графика HNSW (рекомендуется) | m: ⊙ |
| Более быстрая доступность индекса после загрузки. | Оставьте индексирование включенным (поведение по умолчанию) | m: 16, indexing_threshold: 20000 (по умолчанию) |

Для активации быстрого поиска HNSW, если он был отключен во время загрузки, индексирование необходимо повторно включить после загрузки.

### 3.1.2 ОТЛОЖИТЬ ПОСТРОЕНИЕ ГРАФИКА HNSW

Для плотных векторов **m** установка параметра HNSW ⊙ полностью отключает построение индекса. Векторы по-прежнему будут храниться, но не будут индексироваться до тех пор, пока вы не включите индексирование позже.

```
PUT /collections/{collection_name}
{
```

```
  "vectors": {
    "size": 768,
    "distance": "Cosine"
  },
  "hnsw_config": {
     "m": ⊙
  }
}
```

По завершению повторно включите HNSW, установив значение **m**, соответствующее вашей версии программного продукта (обычно 16 или 32).

```
PATCH /collections/{collection_name}
{
  "vectors": {
    "size": 768,
    "distance": "Cosine"
  },
  "hnsw_config": {
     "m": 16
  }
}
```

### 3.1.3 Полностью отключить индексирование ( indexing_threshold: ⊙)

Если вы выполняете первоначальную загрузку большого набора данных, возможно, вам стоит отключить индексирование во время загрузки. Это позволит избежать ненужного индексирования векторов, которые будут перезаписаны следующей партией данных.

Установка **indexing_threshold** этого параметра ⊙ полностью отключает индексирование:

```
PUT /collections/{collection_name}
{
  "vectors": {
    "size": 768,
```

```
    "distance": "Cosine"
  },
  "optimizers_config": {
    "indexing_threshold": ⊙
  }
}
```

После завершения загрузки вы можете включить индексирование, установив **indexing_threshold** желаемое значение (по умолчанию — 20000):

```
PATCH /collections/{collection_name}
{
  "optimizers_config": {
    "indexing_threshold": 20000
  }
}
```

## 3.2 Резервное копирование и восстановление коллекций с использованием снимков

Коллекция — это базовая единица хранения данных в РЕД Вектор Данных. Она содержит векторы, их идентификаторы и полезную нагрузку. Однако для обеспечения эффективности поиска необходимо создавать дополнительные структуры данных поверх этих данных. Создание таких структур данных может занять некоторое время, особенно для больших коллекций. Именно поэтому использование снимков — лучший способ экспорта и импорта коллекций РЕД Вектор Данных, поскольку они содержат все необходимые компоненты для эффективного восстановления всей коллекции. В работе будем использовать кластер РЕД Вектор Данных из 3 узлов. Однако тот же подход применим и к одноузловой конфигурации.

### 3.2.1 Установка соединения с РЕД Вектор Данных

Предварительные требования: установка необходимых зависимостей.

```
pip install red_vector_db-client datasets
```

Для взаимодействия с РЕД Вектор Данных можно использовать Python SDK и прямые HTTP-запросы. Поскольку мы будем использовать кластер из 3 узлов, нам необходимо знать URL-адреса всех узлов. Для простоты давайте сохраним их все в виде констант, вместе с ключом API, чтобы мы могли обращаться к ним позже:

```
RED_VECTOR_DB_MAIN_URL = "https://my-cluster.com:6333"
```

```
RED_VECTOR_DB_NODES = (
    "https://node-0.my-cluster.com:6333",
    "https://node-1.my-cluster.com:6333",
    "https://node-2.my-cluster.com:6333",
)
RED_VECTOR_DB_API_KEY = "my-api-key"
```

Теперь мы можем создать экземпляр клиента:

```
from red_vector_db_client import Red_vector_dbClient
client = Red_vector_dbClient(RED_VECTOR_DB_MAIN_URL, api_key=RED_VECTOR_DB_API_KEY)
```

Прежде всего, мы создадим коллекцию на основе предварительно вычисленного набора данных. Если у вас уже есть коллекция, вы можете пропустить этот шаг и начать с создания снимка.

Мы будем использовать набор данных с предварительно вычисленными эмбеддингами, доступный на **Hugging Face Hub**. Этот набор данных называется **Red_vector_db/arxiv-titles-instructorxl-embeddings** и был создан с использованием модели **InstructorXL**. Он содержит 2,25 миллиона эмбеддингов для заголовков статей из набора данных **arXiv**.

Загрузка набора данных осуществляется очень просто:

```
from datasets import load_dataset
dataset = load_dataset(
    "Red_vector_db/arxiv-titles-instructorxl-embeddings", split="train", streaming=True
)
```

Мы использовали потоковый режим, поэтому набор данных не загружается в память. Вместо этого мы можем пройтись по нему и извлечь идентификатор и векторное представление:

```
for payload in dataset:
    id_ = payload.pop("id")
    vector = payload.pop("vector")
    print(id_, vector, payload)
```

Один полезный груз выглядит следующим образом:

```
{
 'title': 'Dynamics of partially localized brane systems',
 'DOI': '1109.1415'
}
```

### 3.2.2 Создание коллекции

Создадим коллекцию, не меняя ее конфигурации. Конфигурация также является частью снимка коллекции.

```
from red_vector_db_client import models
if not client.collection_exists("test_collection"):
    client.create_collection(
        collection_name="test_collection",
        vectors_config=models.VectorParams(
            size=768,  # Size of the embedding vector generated by the InstructorXL model
            distance=models.Distance.COSINE
        ),
    )
```

Загрузим небольшую часть набора данных.

```
ids, vectors, payloads = [ ], [ ], [ ]
for payload in dataset:
    id_ = payload.pop("id")
    vector = payload.pop("vector")

    ids.append(id_)
    vectors.append(vector)
    payloads.append(payload)

    # We are going to upload only 1000 vectors
```

```
    if len(ids) == 1000:
        break

client.upsert(
    collection_name="test_collection",
    points=models.Batch(
        ids=ids,
        vectors=vectors,
        payloads=payloads,
    ),
)
```

Если у вас уже есть коллекция, можно пропустить предыдущий шаг и начать с создания снимка.

### 3.2.3 Создание и загрузка снимков

РЕД Вектор Данных предоставляет HTTP конечную точку для запроса создания снимка, но мы также можем вызвать её с помощью Python SDK. Наша конфигурация состоит из 3 узлов, поэтому нам нужно вызвать конечную точку на каждом из них и создать снимок на каждом узле. При использовании Python SDK это означает создание отдельного экземпляра клиента для каждого узла.

При большом размере коллекции может возникнуть ошибка тайм-аута. Вы можете запустить процесс создания снимков в фоновом режиме, не дожидаясь результата, используя соответствующий параметр. Список всех снимков **wait=false** всегда можно получить позже через API.

```
snapshot_urls = []
for node_url in RED_VECTOR_DB_NODES:
    node_client = Red_vector_dbClient(node_url, api_key=RED_VECTOR_DB_API_KEY)
    snapshot_info = node_client.create_snapshot(collection_name="test_collection")

    snapshot_url = f"{node_url}/collections/test_collection/snapshots/{snapshot_info.name}"
    snapshot_urls.append(snapshot_url)
```

Ответ.

```
{
  "result": {
    "name": "test_collection-559032209313046-2024-01-03-13-20-11.snapshot",
    "creation_time": "2024-01-03T13:20:11",
    "size": 18956800
  },
  "status": "ok",
  "time": 0.307644965
}
```

Получив URL-адреса снимков, мы можем их загрузить. Пожалуйста, убедитесь, что вы включили ключ API в заголовки запроса. Загрузка снимка возможна только через HTTP API, поэтому мы будем использовать **requests** библиотеку.

```
import requests
import os

# Create a directory to store snapshots
os.makedirs("snapshots", exist_ok=True)

local_snapshot_paths = []
for snapshot_url in snapshot_urls:
    snapshot_name = os.path.basename(snapshot_url)
    local_snapshot_path = os.path.join("snapshots", snapshot_name)

    response = requests.get(
        snapshot_url, headers={"api-key": RED_VECTOR_DB_API_KEY}
    )
```

```
    with open(local_snapshot_path, "wb") as f:
        response.raise_for_status()
        f.write(response.content)

    local_snapshot_paths.append(local_snapshot_path)
```

В качестве альтернативы можно использовать **wget** команду:

```
wget https://node-0.my-cluster.com:6333/collections/test_collection/snapshots/test_collection-559032209313046-2024-01-03-13-20-11.snapshot \
    --header="api-key: ${RED_VECTOR_DB_API_KEY}" \
    -O node-0-shapshot.snapshot

wget https://node-1.my-cluster.com:6333/collections/test_collection/snapshots/test_collection-559032209313047-2024-01-03-13-20-12.snapshot \
    --header="api-key: ${RED_VECTOR_DB_API_KEY}" \
    -O node-1-shapshot.snapshot

wget https://node-2.my-cluster.com:6333/collections/test_collection/snapshots/test_collection-559032209313048-2024-01-03-13-20-13.snapshot \
    --header="api-key: ${RED_VECTOR_DB_API_KEY}" \
    -O node-2-shapshot.snapshot
```

Снимки теперь хранятся локально. Можно использовать их для восстановления коллекции на другом экземпляре РЕД Вектор Данных или рассматривать их как резервную копию.

### 3.2.4 Восстановление из снимка

Снимок готов к восстановлению. Обычно он используется для перемещения коллекции в другой экземпляр РЕД Вектор Данных, но мы собираемся использовать его для создания новой коллекции в том же кластере. У нее просто будет другое имя. **test_collection_import.** Не нужно сначала создавать коллекцию, так как она будет создана автоматически.

Восстановление коллекции также выполняется отдельно на каждом узле, но **Python SDK** пока этого не поддерживает. Вместо этого мы будем использовать HTTP API и отправлять запрос на каждый узел с помощью **requests** библиотеки.

```
for node_url, snapshot_path in zip(RED_VECTOR_DB_NODES, local_snapshot_paths):
    snapshot_name = os.path.basename(snapshot_path)
    requests.post(
        f"{node_url}/collections/test_collection_import/snapshots/upload?priority=snapshot",
        headers={
            "api-key": RED_VECTOR_DB_API_KEY,
        },
        files={"snapshot": (snapshot_name, open(snapshot_path, "rb"))},
    )
```

В качестве альтернативы можно использовать **curl** команду:

```
curl -X POST 'https://node-0.my-cluster.com:6333/collections/test_collection_import/snapshots/upload?priority=snapshot' \
    -H 'api-key: ${RED_VECTOR_DB_API_KEY}' \
    -H 'Content-Type:multipart/form-data' \
    -F 'snapshot=@node-0-shapshot.snapshot'

curl -X POST 'https://node-1.my-cluster.com:6333/collections/test_collection_import/snapshots/upload?priority=snapshot' \
    -H 'api-key: ${RED_VECTOR_DB_API_KEY}' \
    -H 'Content-Type:multipart/form-data' \
    -F 'snapshot=@node-1-shapshot.snapshot'

curl -X POST 'https://node-2.my-cluster.com:6333/collections/test_collection_import/snapshots/upload?priority=snapshot' \
```

```
-H 'api-key: ${RED_VECTOR_DB_API_KEY}' \
-H 'Content-Type:multipart/form-data' \
-F 'snapshot=@node-2-shapshot.snapshot'
```

**Важно!** Выбран параметр **priority=snapshot**, чтобы обеспечить приоритет снимка над данными, хранящимися на узле.

## 3.3 Крупномасштабный поиск

### 3.3.1 Набор данных и аппаратное обеспечение

В качестве набора данных мы будем использовать LAION-400M, коллекцию из приблизительно 400 миллионов векторов, полученных из изображений, извлеченных из набора данных Common Crawl. Каждый вектор имеет 512-мерность и генерируется с использованием модели CLIP.

Векторы связаны с рядом полей метаданных, таких как **url**, **caption**, **LICENSE**, и т.д. Общий размер полезной нагрузки составляет приблизительно 200 ГБ, а размер векторов — 400 ГБ.

**Внимание! В наборе данных сами изображения не хранятся, а содержатся только URL-адреса источников изображений. На момент написания статьи некоторые из этих URL-адресов уже недоступны.**

Набор данных доступен в виде 409 фрагментов, каждый из которых содержит приблизительно 1 миллион векторов. Для загрузки фрагментов набора данных по одному мы будем использовать следующий скрипт на Python.

Минимальная конфигурация оборудования для решения этой задачи:

- 8 ядер ЦП,
- 64 ГБ оперативной памяти,
- 650 ГБ дискового пространства.

Этой конфигурации достаточно для индексирования и исследования набора данных в однопользовательском режиме; задержка достаточно приемлема для построения интерактивных графиков и навигации по панели управления. Для конфигураций производственного уровня может потребоваться больше ядер процессора и оперативной памяти.

### 3.3.2 Загрузка и индексирование

Для загрузки фрагментов набора данных по одному будем использовать следующий скрипт на Python:

```
export RED_VECTOR_DB_URL="https://xxxx-xxxx.xxxx.cloud.red_vector_db.io"
```

```
export RED_VECTOR_DB_API_KEY="xxxx-xxxx-xxxx-xxxx"

python upload.py
```

Этот скрипт будет загружать фрагменты набора данных LAION по одному и выгружать их в Red_vector_db. Промежуточные данные не сохраняются на диске, поэтому скрипт не требует много дискового пространства на стороне клиента.

Давайте рассмотрим использованную нами конфигурацию коллекции:

```
client.create_collection(
    RED_VECTOR_DB_COLLECTION_NAME,
    vectors_config=models.VectorParams(
        size=512, # CLIP model output size
        distance=models.Distance.COSINE, # CLIP model uses cosine distance
        datatype=models.Datatype.FLOAT16, # We only need 16 bits for float, otherwise disk usage would be 800Gb instead of 400Gb
        on_disk=True # We don't need original vectors in RAM
    ),
    # Even though CLIP vectors don't work well with binary quantization, out of the box,
    # we can rely on query-time oversampling to get more accurate results
    quantization_config=models.BinaryQuantization(
        binary=models.BinaryQuantizationConfig(
            always_ram=True,
        )
    ),
    optimizers_config=models.OptimizersConfigDiff(
        # Bigger size of segments are desired for faster search
        # However it might be slower for indexing
        max_segment_size=5_000_000,
    ),
    # Having larger M value is desirable for higher accuracy,
    # but in our case we care more about memory usage
    # We could still achieve reasonable accuracy even with M=6 + oversampling
    hnsw_config=models.HnswConfigDiff(
```

```
        m=6, # decrease M for lower memory usage
        on_disk=False
    ),
)
```

Следует отметить несколько важных моментов:

1. Используем **FLOAT16** тип данных для векторов, что позволяет хранить векторы вдвое меньшего размера по сравнению с обычным **FLOAT32** типом данных. Для этого набора данных не наблюдается существенной потери точности.

2. Используем **BinaryQuantization** с **always_ram=True** для включения передискретизации во время выполнения запроса. Это позволяет нам получать точный и ресурсоэффективный поиск, даже несмотря на то, что 512-мерные **CLIP**-векторы по умолчанию плохо работают с бинарным квантованием.

3. Используем **HnswConfig** оператор с **m=6** для уменьшения потребления памяти. Более подробно об использовании памяти мы поговорим в следующем разделе.

Цель этой конфигурации — гарантировать, что компоненту предварительной выборки в процессе поиска никогда не потребуется загружать данные с диска, и что как минимум минимальная версия векторов и индекса векторов всегда будет находиться в оперативной памяти. На втором этапе поиска можно явно определить, сколько раз мы можем позволить себе загружать данные с диска.

В нашем эксперименте процесс загрузки данных происходил со скоростью 5000 точек в секунду. Процесс индексации выполнялся параллельно с загрузкой и происходил со скоростью приблизительно 4000 точек в секунду.

### 3.3.3 Поисковый запрос

Для точного контроля степени избыточной выборки будем использовать следующий поисковый запрос:

```
limit = 50
rescore_limit = 1000 # oversampling factor is 20

query = vectors[query_id] # One of existing vectors

response = client.query_points(
    collection_name=RED_VECTOR_DB_COLLECTION_NAME,
    query=query,
    limit=limit,
```

```
    # Go to disk
    search_params=models.SearchParams(
      quantization=models.QuantizationSearchParams(
        rescore=True,
      ),
    ),
    # Prefetch is performed using only in-RAM data,
    # so querying even large amount of data is fast
    prefetch=models.Prefetch(
      query=query,
      limit=rescore_limit,
      params=models.SearchParams(
        quantization=models.QuantizationSearchParams(
          # Avoid rescoring in prefetch
          # We should do it explicitly on the second stage
          rescore=False,
        ),
      )
    )
  )
```

Этот запрос состоит из двух этапов:

Первый этап — это предварительная выборка, которая выполняется с использованием только данных из оперативной памяти. Она очень быстрая и позволяет получить большое количество кандидатов.

Второй этап — это переоценка, которая выполняется с использованием векторов полного размера, хранящихся на дисках.

Использование двухэтапного поиска позволяет точно контролировать объем данных, загружаемых с диска, и обеспечивать баланс между скоростью и точностью поиска.

## 3.4 Миграция данных между векторными базами данными

### 3.4.1 Использование инструмента миграции РЕД Вектор Данных

Инструмент миграции РЕД Вектор данных можно запустить через Docker. Для установки указать:

```
docker pull registry.cloud.red_vector_db.io/library/red_vector_db-migration
```

Пример выполнения миграции из одного массива РЕД Вектор Данных в другой:

```
docker run --rm -it \
  registry.cloud.red_vector_db.io/library/red_vector_db-migration red_vector_db \
  --source.url 'https://source-instance.cloud.red_vector_db.io:6334' \
  --source.api-key 'red_vector_db-source-key' \
  --source.collection 'benchmark' \
  --target.url 'https://target-instance.cloud.red_vector_db.io:6334' \
  --target.api-key 'red_vector_db-target-key' \
  --target.collection 'benchmark'
```

**Примечание**: Интерфейс командной строки миграции использует API Red_vector_db GRPC, это означает, что для URL-адресов Red_vector_db с помощью интерфейса командной строки миграции всегда необходимо настраивать порт GRPC (по умолчанию: 6334).

### 3.4.2 Пример миграции с Pinecone на РЕД Вектор Данных

Вам потребуется следующая информация от Pinecone:

- ваш API-ключ Pinecone;
- название индекса;
- URL-адрес хоста индекса.

Имея эту информацию, вы можете перенести свою базу данных векторов из Pinecone в РЕД Вектор Данных с помощью следующей команды:

```
docker run --net=host --rm -it registry.cloud.red_vector_db.io/library/red_vector_db-migration pinecone \
  --pinecone.index-host 'https://sample-movies-efgjrye.svc.aped-4627-b74a.pinecone.io' \
  --pinecone.index-name 'sample-movies' \
  --pinecone.api-key 'pcsk_7Dh5MW_...' \
  --red_vector_db.url 'https://5f1a5c6c-7d47-45c3-8d47-d7389b1fad66.eu-west-1-0.aws.cloud.red_vector_db.io:6334' \
```

```
  --red_vector_db.api-key 'eyJhbGciOiJIUzI1NiIsInR5c…' \
  --red_vector_db.collection 'sample-movies' \
  --migration.batch-size 64
```

## 3.5 Статические вложения в РЕД Вектор Данных

С точки зрения векторной базы данных, статические вложения ничем не отличаются от любых других моделей. В конце концов, это плотные векторы, и вы можете просто хранить их в коллекции РЕД Вектор Данных. Вот как это делается с моделью **sentence-transformers/static-retrieval-mrl-en-v1**:

```
import uuid

from sentence_transformers import SentenceTransformer
from red_vector_db_client import Red_vector_dbClient, models

# The model produces vectors of size 1024
model = SentenceTransformer(
    "sentence-transformers/static-retrieval-mrl-en-v1"
)

# Let's assume we have a collection "my_collection"
# with a single vector called "static"
client = Red_vector_dbClient("http://localhost:6333")

# Calling the sentence transformer model to encode
# the text is not different compared to any other model
client.upsert(
    "my_collection",
    points=[
        models.PointStruct(
            id=uuid.uuid4().hex,
            vector=model.encode("Hello, world!"),
            payload={"static": "Hello, world!"},
```

```
    )
  ]
)
```

Процесс поиска не станет быстрее только потому, что вы используете статические вложения. Для сравнения можно закодировать и полностью проиндексировать 171 000 документов в РЕД Вектор данных примерно за 7,5 минут. Все это было сделано на обычном ноутбуке, без ускорения с помощью графического процессора.

## 3.6 Вопросы и распространенные ошибки

### 3.6.1 Векторы

1. *Какова максимальная размерность вектора, поддерживаемая РЕД Вектор Данных?*

В случае плотных векторов РЕД Вектор Данных поддерживает до 65 535 измерений.

2. *Каков максимальный размер векторных метаданных, которые можно хранить?*

Ограничений по размеру метаданных нет, но их следует оптимизировать для повышения производительности и эффективности использования ресурсов. Пользователи могут установить верхние пределы в конфигурации.

3. *Может ли один и тот же поисковый запрос на сходство давать разные результаты на разных компьютерах?*

Да, из-за различий в конфигурации оборудования и параллельной обработке результаты могут незначительно отличаться.

4. *Как выбрать подходящие векторные представления для моего конкретного случая?*

Это зависит от характера ваших данных и конкретного приложения. Учитывайте такие факторы, как размерность, модели, специфичные для предметной области, и характеристики производительности различных вложений.

5. *Как РЕД Вектор Данных обрабатывает различные векторные представления от разных поставщиков в одной и той же коллекции?*

РЕД Вектор Данных изначально поддерживает несколько векторов для каждой точки данных, что позволяет различным вложениям от разных поставщиков сосуществовать в одной коллекции.

6. *Могу ли я перенести свои векторные представления из другого хранилища в РЕД Вектор Данных?*

Да, РЕД Вектор Данных поддерживает миграцию векторных представлений из других хранилищ векторной графики, что упрощает переход и освоение функций РЕД Вектора Данных.

7. *Почему количество индексированных векторов не соответствует количеству векторов в коллекции?*

Не всегда нужно индексировать все векторы в коллекции. Он хранит данные в виде сегментов, и, если сегмент достаточно мал, эффективнее выполнить поиск по нему с полным сканированием.

Убедитесь, что статус коллекции **green** и количество неиндексированных векторов меньше порогового значения для индексации.

8. *Почему в информации о коллекции отображается неточное количество точек?*

API для работы с коллекцией в РЕД Вектор Данных возвращает приблизительное количество точек в коллекции. Если вам нужно точное число, вы можете использовать API для подсчета.

9. *Векторные изображения в коллекции не соответствуют тем, что я загрузил.*

Этому могут быть две возможные причины:

– В настройках коллекции вы использовали метрику расстояния **Cosine**. В этом случае РЕД Вектор Данных предварительно нормализует ваши векторы для более быстрого вычисления расстояния. Если вам строго необходимо сохранить исходные векторы, рассмотрите возможность использования **Dot** метрики расстояния вместо этого.

– Вы использовали тип данных для хранения векторов **uint8**. Он требует специального формата для входных значений, который может быть несовместим с типичным выходным форматом моделей встраивания.

### 3.6.2 Поиск

1. *Как РЕД Вектор Данных обрабатывает обновления данных и поиск в режиме реального времени?*

РЕД Вектор Данных поддерживает обновление векторных данных в режиме реального времени, при этом новые, обновленные и удаленные векторы доступны для немедленного поиска. Система использует полносканирующий поиск по неиндексированным сегментам во время фонового обновления индекса.

2. *В результатах поиска содержатся векторы с нулевыми значениями. Почему?*

По умолчанию РЕД Вектор Данных пытается минимизировать сетевой трафик и не возвращает векторы в результатах поиска. Но вы можете заставить РЕД Вектор Данных делать это, установив **with_vector** параметр **Search/Scroll** в значение **true**. Если в результатах по-прежнему отображается ошибка **"vector": null**, возможно,

передаваемый вектор имеет неправильный формат или возникла проблема с вызовом метода **upsert**.

3. *Как выполнить поиск без вектора?*

Вероятно, вам нужен метод прокрутки. Он позволяет получать записи на основе фильтров или даже перебирать все записи в коллекции.

4. *Поддерживает ли РЕД Вектор Данных полнотекстовый или гибридный поиск?*

РЕД Вектор Данных — это, прежде всего, поисковая система для векторной графики, и мы внедряем поддержку полнотекстового поиска только в том случае, если это не ставит под угрозу возможности векторного поиска. Это касается как интерфейса, так и производительности.

Что умеет РЕД Вектор Данных:

– поиск с использованием полнотекстовых фильтров;

– применение полнотекстового фильтра к векторному поиску (т.е. выполните векторный поиск среди записей, содержащих определенные слова или фразы),

– использование поиска по префиксу и семантический поиск по мере ввода,

– разреженные векторы, как в **SPLADE** или аналогичных моделях,

– мультивекторы, например, **ColBERT** и другие модели позднего взаимодействия,

– комбинация нескольких поисков.

### 3.6.3 Коллекции

1. *Сколько коллекций я могу создать?*

Создавайте столько коллекций, сколько хотите, но имейте в виду, что каждая из них требует дополнительных ресурсов. Настоятельно рекомендуется не создавать много небольших коллекций, так как это приведет к значительному увеличению потребления ресурсов.

2. *Как загрузить большое количество векторов в коллекцию РЕД Вектор Данных?*

Ознакомьтесь с нашими рекомендациями в руководстве по массовой загрузке.

3. *Могу ли я хранить только квантованные векторы и отбрасывать векторы с полной точностью?*

Нет, РЕД Вектор Данных требует векторов с полной точностью для таких операций, как переиндексация, переоценка и т. д.

### 3.6.4 Совместимость

1. *Совместим ли РЕД Вектор Данных с центральными или графическими процессорами для векторных вычислений?*

РЕД Вектор Данных в основном полагается на ускорение ЦП для обеспечения масштабируемости и эффективности. Однако мы также поддерживаем индексирование с ускорением на графических процессорах у всех основных производителей.

2. *Вы гарантируете совместимость между версиями?*

В случае, если ваша версия устарела, мы гарантируем совместимость только между двумя последовательными минорными версиями. Это также относится к версиям клиента. Убедитесь, что версия вашего клиента никогда не отличается от версии кластера более чем на одну минорную версию. Хотя мы окажем помощь в устранении неполадок и ошибок, специфичных для наших продуктов, РЕД Вектор Данных не несет ответственности за проверку, написание (или переписывание) или отладку пользовательского кода.

3. *Вы поддерживаете понижение рейтинга?*

Мы не поддерживаем понижение версии кластера ни в одном из наших продуктов. Если вы развернете более новую версию РЕД Вектор Данных, ваши данные будут автоматически перенесены в новый формат хранения. Эта миграция необратима.

4. *Следует ли создавать индексы полезной нагрузки до или после загрузки?*

Создавайте индексы полезной нагрузки до загрузки, чтобы избежать перестроения индексов. Однако существуют сценарии, когда определение индексов после загрузки допустимо. Например, можно настроить новую логику фильтрации после запуска.

Если вы заранее знаете свои фильтры, всегда следует сначала индексировать данные. Если позже потребуется проиндексировать другой набор данных, это тоже можно сделать, но следует учитывать возможное снижение производительности.

5. *Следует ли создавать отдельную коллекцию РЕД Вектор Данных для каждого пользователя?*

Нет. Создание отдельной коллекции для каждого пользователя требует больше ресурсов. Вместо создания отдельных коллекций для каждого пользователя мы рекомендуем создать единую коллекцию и обеспечить отдельный доступ с использованием полезных нагрузок. Каждая точка РЕД Вектор Данных может иметь полезную нагрузку в качестве метаданных. Для многопользовательского режима вы можете включить **user_id** или **tenant_id** для каждой точки. Для дальнейшей оптимизации хранения вы можете включить индексирование полей полезной нагрузки для каждого арендатора.

# ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

| *№ версии* | *Дата изменения* | *Автор* | *Состав изменений* |
|---|---|---|---|
| 1.0 | 18.12.2025 | Галкина М.Н. | Первоначальная версия документа. |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |