# Milvus v2.0.0

## Contents

# 关于 Milvus

## Milvus 是什么

关于 Milvus

本文以问答的形式从几个维度来介绍 Milvus。通过阅读本文，你将了解 Milvus 是什么及其相关的基本原理、重要概念、核心优势、应用场景、周边工具等。此外，本文还简单介绍了 Milvus 的系统架构设计以及 Milvus 支持的索引和距离计算方式。

什么是 Milvus 向量数据库?

Milvus 于 2019 年开源，致力于存储、索引和管理由深度神经网络学习与其他机器学习模型生成的海量 Embedding 向量。

Milvus 向量数据库专为向量查询与检索设计，能够为万亿级向量数据建立索引。与现有的主要用作处理结构化数据的关系型数据库不同，Milvus 在底层设计上就是为了处理由各种非结构化数据转换而来的 Embedding 向量而生。

随着互联网不断发展，电子邮件、论文、物联网传感数据、社交媒体照片、蛋白质分子结构等非结构化数据已经变得越来越普遍。如果想要使用计算机来处理这些数据，需要使用 embedding 技术将这些数据转化为向量。随后，Milvus 会存储这些向量，并为其建立索引。Milvus 能够根据两个向量之间的距离来分析他们的相关性。如果两个向量十分相似，这说明向量所代表的源数据也十分相似。

重要概念

如果你刚刚接触向量数据库及向量相似度检索领域，可以通过阅读以下重要概念获得初步了解。

更多 Milvus 相关概念详见 Milvus 术语表。

非结构化数据

非结构化数据指的是数据结构不规则，没有统一的预定义数据模型，不方便用数据库二维逻辑表来表现的数据。非结构化数据包括图片、视频、音频、自然语言等，占所有数据总量的 80%。非结构化数据的处理可以通过各种人工智能（AI）或机器学习（ML）模型转化为向量数据进行。

特征向量

向量又称为 embedding vector，是指由 embedding 技术从离散变量（如图片、视频、音频、自然语言等等各种非结构化数据）转变而来的连续向量。在数学表示上，向量是一个由浮点数或者二值型数据组成的 n 维数组。通过现代的向量转化技术，比如各种人工智能（AI）或者机器学习（ML）模型，可以将非结构化数据抽象为 n 维特征向量空间的向量。这样就可以采用最近邻算法（ANN）计算非结构化数据之间的相似度。

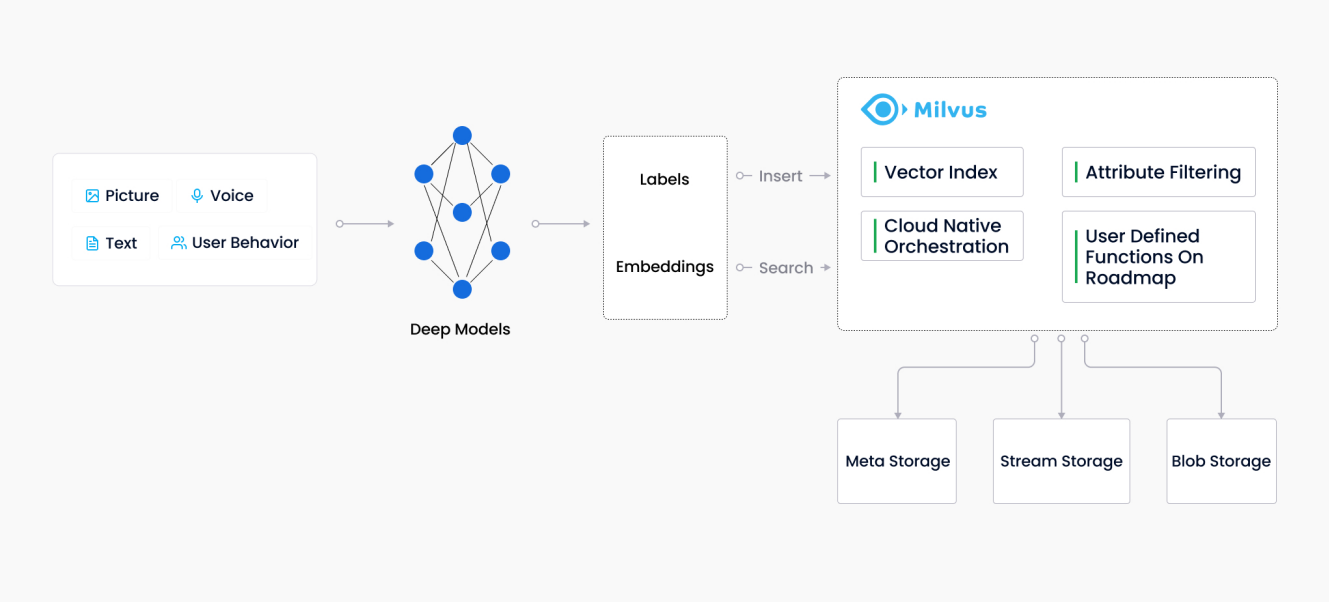



Figure 1: Workflow

向量相似度检索

相似度检索是指将目标对象与数据库中数据进行比对，并召回最相似的结果。同理，向量相似度检索返回的是最相似的向量数据。近似最近邻搜索（ANN）算法能够计算向量之间的距离，从而提升向量相似度检索的速度。如果两条向量十分相似，这就意味着他们所代表的源数据也十分相似。

为什么选择使用 Milvus?

- 高性能：性能高超，可对海量数据集进行向量相似度检索。
- 高可用、高可靠：Milvus 支持在云上扩展，其容灾能力能够保证服务高可用。
- 混合查询：Milvus 支持在向量相似度检索过程中进行标量字段过滤，实现混合查询。
- 开发者友好：支持多语言、多工具的 Milvus 生态系统。

Milvus 支持哪些索引类型及距离计算公式?

创建索引是一个组织数据的过程，是向量数据库实现快速查询百万、十亿、甚至万亿级数据集所依赖的一个巨大组成部分。在查询或检索数据前，必须先指定索引类型及距离计算公式。如未指定索引类型，Milvus 将默认执行暴搜。

索引类型

Milvus 目前支持的向量索引类型大都属于 ANNS（Approximate Nearest Neighbors Search，近似最近邻搜索）。

- FLAT：适用于需要 100% 召回率且数据规模相对较小（百万级）的向量相似性搜索应用。
- IVF_FLAT：基于量化的索引，适用于追求查询准确性和查询速度之间理想平衡的场景。
- IVF_SQ8：基于量化的索引，适用于磁盘或内存、显存资源有限的场景。
- IVF_PQ：基于量化的索引，适用于追求高查询速度、低准确性的场景。
- HNSW：基于图的索引，适用于追求高查询效率的场景。
- ANNOY：基于树的索引，适用于追求高召回率的场景。

更多内容详见向量索引。

距离计算公式

Milvus 基于不同的距离计算方式比较向量间的距离。根据插入数据的形式，选择合适的距离计算方式能极大地提高数据分类和聚类性能。

浮点型向量主要使用以下距离计算公式：

- 欧氏距离 (L2)：主要运用于计算机视觉领域。
- 内积 (IP): 主要运用于自然语言处理（NLP）领域。二值型向量主要使用以下距离计算公式：
- 汉明距离 (Hamming)：主要运用于自然语言处理（NLP）领域。
- 杰卡德距离 (Jaccard)：主要运用于化学分子式检索领域。
- 谷本距离 (Tanimoto)：主要运用于化学分子式检索领域。
- 超结构 (Superstructure)：主要运用于检索化学分子式的相似超结构。
- 子结构 (Substructure)：主要运用于检索化学分子式的相似子结构。

更多内容详见 距离计算方式。

### Milvus 应用场景

你可以使用 Milvus 搭建符合自己场景需求的向量相似度检索系统。Milvus 使用场景包括：

- 图片检索系统：以图搜图，从海量数据库中即时返回与上传图片最相似的图片。
- 视频检索系统：将视频关键帧转化为向量并插入 Milvus，便可检索相似视频，或进行实时视频推荐。
- 音频检索系统：快速检索海量演讲、音乐、音效等音频数据，并返回相似音频。
- 分子式检索系统：超高速检索相似化学分子结构、超结构、子结构。
- 推荐系统：根据用户行为及需求推荐相关信息或商品。
- 智能问答机器人：交互式智能问答机器人可自动为用户答疑解惑。
- DNA 序列分类系统：通过对比相似 DNA 序列，仅需几毫秒便可精确对基因进行分类。
- 文本搜索引擎：帮助用户从文本数据库中通过关键词搜索所需信息。

更多应用场景详见 Milvus 系统搭建教程及 Milvus 用户。

### Milvus 系统架构

Milvus 2.0 是一款云原生向量数据库，采用存储与计算分离的架构设计，所有组件均为无状态组件，极大地增强了系统弹性和灵活性。

整个系统分为四个层面：

- 接入层（Access Layer）：系统的门面，由一组无状态 proxy 组成。对外提供用户连接的 endpoint，负责验证客户端请求并合并返回结果。
- 协调服务（Coordinator Service）：系统的大脑，负责分配任务给执行节点。协调服务共有四种角色，分别为 root coord、data coord、query coord 和 index coord。
- 执行节点（Worker Node）：系统的四肢，负责完成协调服务下发的指令和 proxy 发起的数据操作语言（DML）命令。执行节点分为三种角色，分别为 data node、query node 和 index node。
- 存储服务（Storage）：系统的骨骼，负责 Milvus 数据的持久化，分为元数据存储（meta store）、消息存储（log broker）和对象存储（object storage）三个部分。

更多系统原理的相关内容详见 系统架构。

### 开发者工具

Milvus 为 DevOps 提供丰富的 API 及工具。

### API

Milvus 客户端库对 Milvus API 进行了封装。你可以使用 Milvus 客户端库通过应用代码进行数据插入、删除、查询等操作。

- PyMilvus
- Node.js SDK
- Go SDK

我们正在不断开发新的客户端库。如果你愿意为 Milvus 贡献代码，请前往相应的 Milvus 项目 仓库。

### Milvus 生态系统工具

Milvus 生态系统提供多种强大的工具，包括：

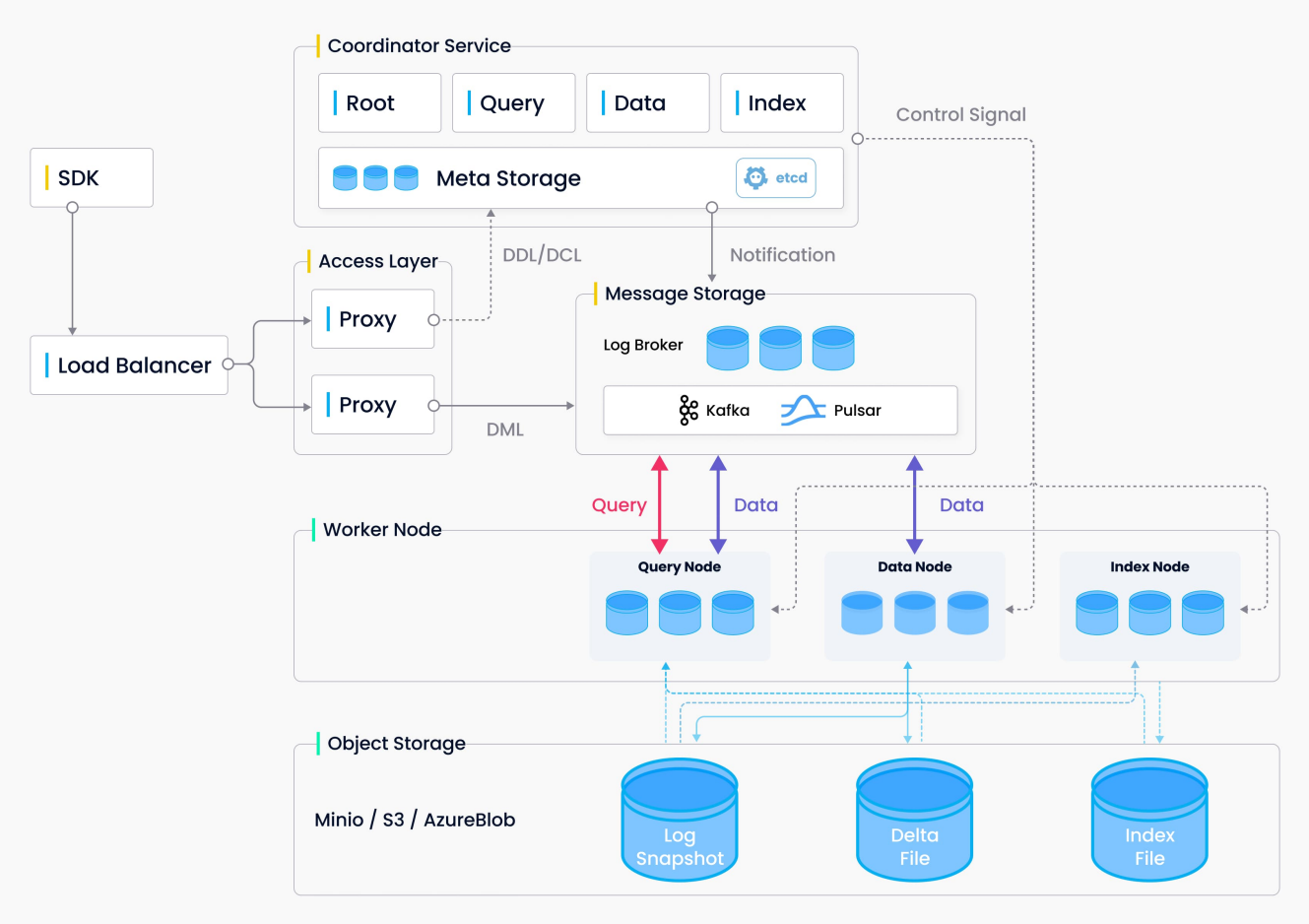



Figure 2: Architecture

- Milvus CLI
- Attu：图形化管理系统。
- MilvusDM：用于导入或导出 Milvus 数据
- Milvus sizing tool：用于根据向量数据量及索引类型估算所需的原始文件大小、内存大小及固态硬盘大小。

更多资源

- 3 分钟快速上手 Milvus：
  - Hello Milvus
- 在测试或生产环境中安装 Milvus：
  - 安装前提
  - 安装单机版 Milvus
  - 安装分布式版 Milvus
- 如果你想要深入了解 Milvus 系统架构设计：
  - 阅读 Milvus 系统架构

## Milvus 2.0 概述

为获得最佳使用体验，我们强烈建议启用最新版本的 Milvus。

设计理念

围绕以下三个理念，我们重新定义下一代云原生向量数据库：

云原生优先：我们认为，只有存储计算分离的架构才能发挥云的弹性，实现按需扩容的模式。另一个值得注意是 Milvus 2.0 采取了读写分离、实时离线分离、计算瓶颈/内存瓶颈/IO 瓶颈分离的微服务化设计模式，这有助于我们面对复杂的工作负载选择最佳的资源配比。

日志即数据（log as data）： Milvus 引入消息存储作为系统的骨架，数据的插入修改只通过消息存储交互，执行节点通过订阅消息流（publish/subscribe）来执行数据库的增删改查操作。这一设计的优势在于降低了系统的复杂度，将数据库关键的持久化和闪回等能力都下钻到存储层；另一方面，日志订阅机制提供了极大的灵活性，为系统未来的拓展奠定了基础。

批流一体：Milvus 2.0 实现了 unified Lambda 流式处理架构，增量数据和离线数据一体化处理。相比 Kappa 架构，Milvus 引入对日志流的批量计算将日志快照和构建索引存入对象存储，这大大提高了故障恢复速度和查询效率。为了将无界的流式数据拆分成有界的窗口，Milvus 采用 watermark 机制，通过写入时间（也可以是事件发生时间）将数据切分为多个小的处理单元（message pack），并维护了一条时间轴便于用户基于某个时间点进行查询。

功能亮点

Milvus 2.0 作为一款开源分布式向量数据库产品，始终将产品的易用性放在系统设计的第一优先级。一款数据库的使用成本不仅包含了运行态的资源消耗成本，也包含了运维成本和接入学习成本。Milvus 新版本支持了大量降低用户使用成本的功能。

持续可用

实现数据的可靠存储和可持续的服务是对数据库产品的基本要求。我们的理念是 Fail cheap, fail small, fail often。Fail cheap 指的是 Milvus 采取的存储计算分离架构，节点失败恢复的处理十分简单，且代价很低。Fail small 指的是 Milvus 采取分而治之的思想，每个协调服务仅处理读/写/增量/历史数据中的一个部分，设计被大大简化。Fail often 指的是混沌测试的引入，通过故障注入模拟硬件异常、依赖失效等场景，加速问题在测试环境被发现的概率。

向量/标量混合查询

为了解决结构化数据和非结构化数据的割裂问题，Milvus 2.0 支持标量存储和向量标量混合查询。混合查询帮助用户找出符合过滤表达式的近似邻，目前 Milvus 支持等于、大于、小于等关系运算以及 NOT、AND、OR 、IN 等逻辑运算。

多一致性

Milvus 2.0 是基于消息存储构建的分布式数据库，遵循 PACELC 定理所定义的，必须在一致性和可用性/延迟之间进行取舍。绝大多数 Milvus 场景在生产中不应过分关注数据一致性的问题，原因是接受少量数据不可见对整体召回率 (Recall) 的

影响极小，但对于性能的提升帮助很大。尽管如此，我们认为强一致性 (Strong)、有界一致性 (Bounded Staleness)、会话一致性 (Session) 等一致性保障语义依然有其独特的应用场景。比如，在功能测试场景下，用户可能期待使用强一致语义保证测试结果的正确性，因此 Milvus 支持请求级别的可调一致性级别。

时间旅行

数据工程师经常会因为脏数据、代码逻辑等问题需要回滚数据。传统的数据库通常通过快照方式来实现数据回滚，有时甚至需要重新训练，带来高昂的额外开销和维护成本。Milvus 对所有数据增删操作维护了一条时间轴，用户查询时可以指定时间戳以获取某个时间点之前的数据视图。基于 Time Travel，Milvus 还可以很轻量地实现备份和数据克隆功能。

ORM Python SDK：

对象关系映射（object relational mapping，ORM）技术使用户更加关注于业务模型而非底层的数据模型，便于开发者维护表、字段与程序之间的关联关系。为了弥补 AI 算法概念验证（Proof of concept）到实际生产部署之间的缺口，我们设计了对象关系映射 Python Milvus API，而其背后的实现可以是通过嵌入式的 library、单机部署、分布式集群，也可能是云服务。通过统一的 API 提供一致的使用体验，避免云端两侧重复开发、测试与上线效果不一致等问题。

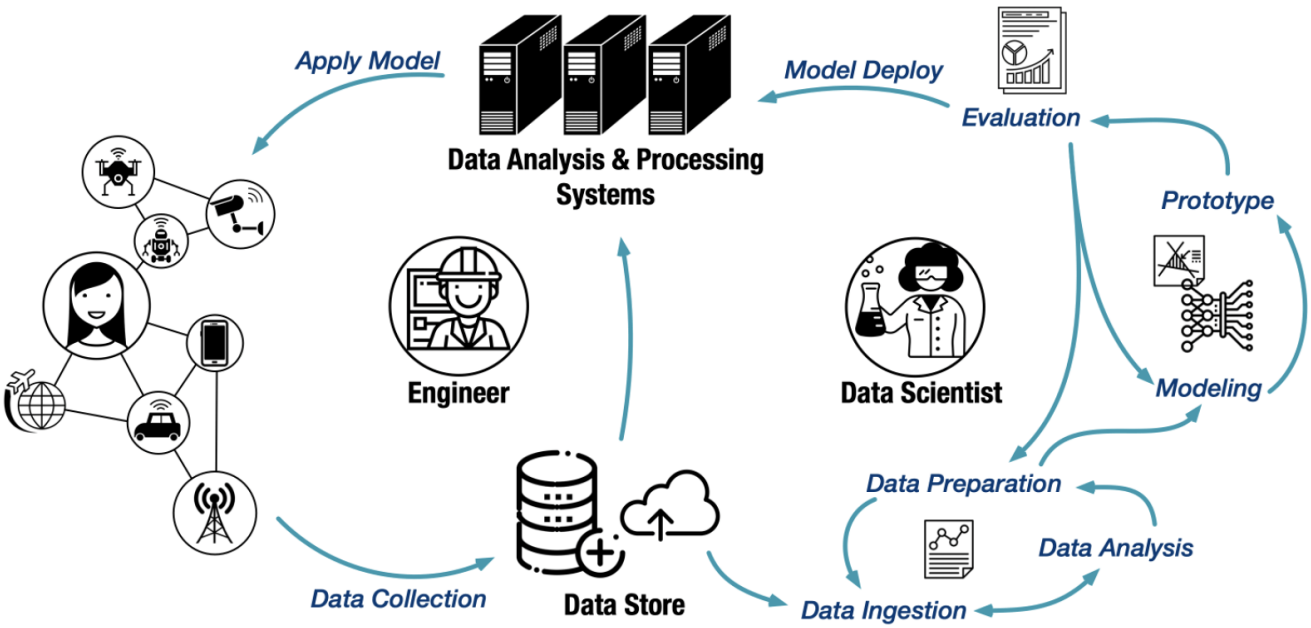



Figure 3: ORM_Python_SDK

丰富的周边支持：

- Attu 是 Milvus 图形化管理界面，包含了集群状态可视化、元数据管理、数据查询等实用功能。Attu 源码也会作为独立项目开源，期待有更多感兴趣的人加入共同建设。
- Milvus CLI 是基于 PyMilvus 的 Milvus 命令行界面，支持连接服务器、数据操作和数据导出/导入。
- 支持基于 helm 和 docker-compose 的一键部署。
- Milvus 2.0 使用开源时序数据库 Prometheus 存储性能和监控数据，同时依赖 Grafana 进行指标展示。

Milvus 2.0：性能更优、功能更强

Milvus 2.0

Milvus 1.x

架构

云原生

共享存储

可扩展性

500+ 个节点

1 - 32 个读节点，1 个写节点

持久性

对象存储 (OSS)

分布式文件系统 (DFS)

本地磁盘

网络文件系统 (NFS)

可用性

99.9%

99%

数据一致性

多种一致性

Strong

Bounded Staleness

Session

Consistent prefix

最终一致

数据类型支持

向量数据

标量数据

字符串与文本 (开发中)

向量数据

基本操作

插入数据

删除数据 (开发中)

数据查询

相似最邻近（ANN）搜索

基于半径的最近邻算法（RNN）(开发中)

插入数据

删除数据

相似最邻近（ANN）搜索

高级功能

标量字段过滤

Time Travel

多云/地域部署

数据管理工具

Mishards

Milvus DM 数据迁移工具

索引类型

Faiss

Annoy

Hnswlib

RNSG

ScaNN (开发中)

On-disk index (开发中)

Faiss

Annoy

Hnswlib

RNSG

SDK

Python

Node.js

Go (开发中)

Java (开发中)

C++ (开发中)

Python

Java

Go

RESTful

C++

当前状态

预览版本。预计 2021 年底发布稳定版本。

长期支持（LTS）版本

## Milvus 用户

Milvus 的全球企业用户超过 1000，应用领域十分广泛。以下是在开发中使用了 Milvus 的用户列表。

| 公司 | 行业 | 用户案例 |
| --- | --- | --- |
| 百度飞桨 (PaddlePaddle) | 深度学习 | 强强联手！Milvus 与 PaddlePaddle 深度整合，赋能工业级 AI 应用 |
| Vova | 跨境电商 | 基于 Milvus 的 VOVA 拍照购实践 |
| Opera | 浏览器 | 相似视频搜索—Opera 的 Milvus 实践 |
| 趋势科技 | 信息技术 | Milvus 在趋势科技的实践 |
| 知擎者 | 商标注册 | 图形商标近似检索-知擎者的 Milvus 实践 |
| 搜狐 | 互联网 | 基于语义向量的内容召回和短文本分类的错误查找-搜狐的 Milvus 实战 |
| 格像科技 | 互联网 | 基于 Milvus 构建的近似最近邻（ANN）搜索引擎 |

| 公司 | 行业 | 用户案例 |
| --- | --- | --- |
| 神州泰岳 | 信息技术 | 我的机器人新同事 |
| 汽车之家 | 汽车网站 | 相似问答检索——汽车之家的 Milvus 实践 |
| 不亦乐乎科技 | 音乐、科技 | 让音乐伴随你左右-Milvus 在丸音的应用 |
| 爱云动 | 运动、科技 | 我在哪？让 Milvus 给你答案—爱云动的 Milvus 实践 |
| 蓝灯鱼智能科技 | 人工智能、专利、商标 | 蓝灯鱼 AI 专利检索在 Milvus 的实践 |
| 焦点科技 | 互联网 | 垃圾询盘过滤，焦点科技的 Milvus 实践 |
| 小米 | 移动互联网 | 信息推流，小米浏览器的 Milvus 实践 |
| 贝壳 | 房地产 | AI 搜房，贝壳找房的 Milvus 实践 |
| 云从科技 | 硬件/软件、信息服务 | Milvus 在云从的深度实践 |
| Lucidworks | 信息技术 | Milvus x Lucidworks 快速构建语义检索 |
| 唯品会 | 电商 | Milvus 在唯品会搜索推荐的实践 |
| Tokopedia | 电商 | Milvus 在 Tokopedia 的应用 \| 让语义搜索更加智能 |
| Juicedata | 软件 | 基于 JuiceFS 搭建 Milvus 分布式集群 |
| 惠普 | 信息科技 | 解决非结构化数据搜索的难题 |
| KubeSphere | 开源分布式操作系统 | |
| 丁香园 | 医学、互联网 | |
| Mozat | 游戏 | |
| 飞书深诺集团 | 营销 | 千万量级图片视频快速检索，轻松配置设计师的灵感挖掘神器 |
| 妙医佳健康科技集团 | 健康科技 | 结合 Milvus 的医疗问答辅助标记平台 |
| 思必驰 | 科技 | |
| 美的 | 消费电器 | |
| 有赞 | 零售科技 | |
| 陌陌 | 社交 | |
| 涂鸦智能 | 软件 | |
| 北冥星眸 | 人工智能 | 结合 Milvus 的政务场景智能问答以及向量检索 |

## Milvus 使用限制

Milvus 致力于为用户提供性能最佳的向量数据库，方便用户将其运用到 AI 场景中或进行向量相似度搜索。Milvus 团队不断努力增添新功能以提升用户体验。本文列举了用户在使用 Milvus 时可能会遇到的一些使用限制。

标识符长度限制

| 标识符类型 | 最大长度（字符） |
| --- | --- |
| Collection | 255 |
| Field | 255 |
| Index | 255 |
| Partition | 255 |

标识符命名规则

标识符中仅能包含数字、字母、美元符号（$）、下划线（_）。标识符中第一个字符必须为字母或者下划线。

Collection 及 connection/proxy 数量限制

| 标识符类型 | 最大值 |
| --- | --- |
| Collections | 65536 |
| Connections / proxy | 65536 |

单个 collection 的限制

| 类型 | 最大数量 |
| --- | --- |
| Partitions | 4096 |
| Shards | 256 |
| Fields | 256 |
| Indexes | 1 |
| Entities | unlimited |

字符串限制

| 类型 | 最大长度（字符） |
| --- | --- |
| VARCHAR | 65535 |

未来版本 Milvus 将支持 VARCHAR 以及更多字符串类型。

向量数据限制

| 属性 | 最大值 |
| --- | --- |
| 维度 | 32768 |

RPC 传输数据量限制

| 操作 | 数据量（MB） |
| --- | --- |
| 插入 | 512 |
| 检索 | 512 |
| 查询 | 512 |

加载数据限制

在当前版本中，加载数据最大值不能超过所有 query node 内存总量的 90%，从而为执行引擎预留内存资源。

检索参数限制

| 参数 | 最大值 |
| --- | --- |
| topk（输出向量结果数） | 16384 |
| nq（目标输入向量数） | 16384 |

## 发版说明

v2.0.0

Release date: 2022-01-25

Compatibility

Milvus version

Python SDK version

Java SDK version

Go SDK version

Node.js SDK version

2.0.0

2.0.0

2.0.2

2.0.0

2.0.0

We are excited to announce the general release of Milvus 2.0 and it is now considered as production ready. Without changing the existing functionality released in the PreGA release, we fixed several critical bugs reported by users. We sincerely encourage all users to upgrade your Milvus to 2.0.0 release for better stability and performance.

Improvements

- Changes the default consistency level to Bounded Staleness: If consistency level Strong is adopted during a search, Milvus waits until data is synchronized before the search, thus spending longer even on a small dataset. Under the the default consistency level of Bounded Staleness, newly inserted data remain invisible for a could of seconds before they can be retrieved. For more information, see Guarantee Timestamp in Search Requests.
- #15223 Makes query nodes send search or query results by RPC.

Bug fixes

- Writing blocked by message storage quota exceed exception:
  - #15221 Unsubscribes channel when closing Pulsar consumer.
  - #15230 Unsubscribes channel after query node is down.
  - #15284 Adds retry logic when pulsar consumer unsubscribes channel.
  - #15353 Unsubscribes topic in data coord.
- Resource leakage:
  - #15303 Cleans flow graph if failed to `watchChannel`.
  - #15237 Calls for releasing memory in case that error occurs.
  - #15013 Closes payload writer when error occurs.
  - #14630 Checks leakage of index CGO object.
  - #14543 Fixes that Pulsar reader is not close.
  - #15068 Fixes that file is not close when `ReadAll` returns error in local chunk manager.
  - #15305 Fixes query node search exceptions will cause memory leak.
- High memory usage:
  - #15196 Releases memory to OS after index is built.
  - #15180 Refactors flush manager injection to reduce goroutine number.
  - #15100 Fixes storage memory leak caused by `runtime.SetFinalizer`.
- Cluster hang:
  - #15181 Stops handoff if the segment has been compacted.
  - #15189 Retains `nodeInfo` when query coord panic at `loadBalanceTask`.
  - #15250 Fixes `collectResultLoop` hang after search timeout.
  - #15102 Adds flow graph manager and event manager.
  - #15161 Panic when recover query node failed.
  - #15347 Makes index node panic when failed to save meta to `MetaKV`.
  - #15343 Fixes Pulsar client bug.
  - #15370 Releases collection first when drop collection.
- Incorrect returned data:
  - #15177 Removes global sealed segments in historical.
  - #14758 Fixes that deleted data returned when handoff is done for the segment.

Known issues

- #14077 Core dump happens under certain workload and it is still under reproducing. Solution: The system will be recovered automatically.
- #15283 Cluster fails to recover because Pulsar' s failure to create consumer Pulsar #13920. Solution: Restart pulsar cluster.

- The default dependency Pulsar use old log4j2 version and contains security vulnerability. Solution: Upgrade pulsar dependency to 2.8.2. We will soon release a minor version to upgrade Pulsar to newer releases.
- #15371 Data coord may fail to cleanup channel subscription if balance and node crash happens at same time. Solution: Remove the channel subscription with Pulsar admin.

v2.0.0-PreGA

发布时间：2021-12-31

版本兼容

Milvus 版本

Python SDK 版本

Java SDK 版本

Go SDK 版本

Node.js SDK 版本

2.0.0-PreGA

2.0.0rc9

2.0.0

即将上线

1.0.20

Milvus 2.0.0-PreGA 是 2.0 的预览版。它现在支持通过 primary key 删除 entity 和数据 Compaction 来清除已删除的数据。我们还在 Milvus 中引入了 Loadbalance 机制，以便均匀地分配每个 query node 的内存使用。在这个版本中我们修复了一些关键问题，包括被删除的 collection 数据清理，Jaccard 距离计算错误，以及一些导致系统卡死和内存泄漏的 bug。请注意，由于数据编码格式和 RocksMQ 数据格式的一些变化，Milvus 2.0.0-PreGA 与其他 Milvus 2.0 前期预览版本不兼容。

新增功能

- 删除 entity：Milvus 现在支持通过 primary key 删除 entity。由于 Milvus 依赖于仅追加存储，导致其只支持逻辑删除，也就是说，Milvus 为被删除的 entity 插入删除标记以覆盖实际数据，令搜索或查询不返回被标记的 entity。请注意，过度删除可能会导致搜索性能下降和存储使用量激增。参见删除数据获取更多信息。
- Compaction 机制：通过 Compaction 机制，清理 binlog 中删除或过期的 entity，节省存储空间。该机制为 data coord 触发，data node 执行的后台任务。
- 自动 Loadbalance #9481：Loadbalance 机制将 segment 均匀地分布在 query node 上，以平衡集群的内存使用。它可以自动触发，也可以由用户触发。
- Handoff #9481：Handoff 机制是指当一个 growing segment 转化为 sealed segment 时，query node 等待至该 segment 被 index node 构建索引后，将该 segment 加载到内存中进行搜索或查询。

主要改进

- #12199 在 segment 之间并行执行，以提高搜索性能。
- #11373 允许在 RocksMQ 内部循环中批量消费消息，以提高系统效率。
- #11665 延迟 Handoff 的执行，直到索引创建完成。

问题修复

- 删除 collection 时，etcd、Pulsar 和 MinIO 上的数据没有被清除：
  - #12191 清除 etcd 上被删除 segment 的元数据。
  - #11554 为 data coord 增加 garbage collector。
  - #11552 在 data node 中完成删除 collection 的过程。
  - #12227 删除 collection 时删除所有索引。
  - #11436 修改 `retentionSizeInMB` 默认值为 8192 (8GB)。
- #11901 不同度量类型的属性导致的距离计算错误。

- #12511 不同度量类型的属性导致的相似相关性错误。
- #12225 重复搜索时 RocksMQ 会卡死。
- #12255 RocksMQ 服务器在 Milvus 单机版退出时不会关闭。
- #12281 删除别名时的错误。
- #11769 错误更新 `serviceableTime`。
- #11325 合并搜索结果时崩溃。
- #11248 参数 `guarantee_timestamp` 不起作用。

其他增强

- #12351 更改代理默认的 RPC 传输限制。
- #12055 减少从 MinIO 加载时的内存成本。
- #12248 支持更多的部署指标。
- #11247 为集群增加 `getNodeInfoByID` 和 `getSegmentInfoByNode` 函数。
- #11181 重构 query coord 上的 segment 分配策略。

v2.0.0-RC8

发布时间：2021-11-5

版本兼容

Milvus 版本

Python SDK 版本

Java SDK 版本

Go SDK 版本

Node.js SDK 版本

2.0.0-RC8

2.0.0rc8

即将上线

即将上线

1.0.18

Milvus 2.0.0-RC8 是 2.0 的最后一个预览版本。在该版本中，Milvus 支持 Handoff 任务，Primary Key 去重，以及 Time Travel 搜索功能。随着 Timetick 机制的增强，系统的平均修复时间（mean time to recovery, MTTR）也大幅减少。在针对该版本的千万级数据集的压力测试中，单机版与分布式版 Milvus 都运行超过 84 小时。

目前，Primary Key（`pk`）去重功能无法保证插入新数据会覆盖与其 `pk` 相同的旧数据。因此，当前版本中基于相同 `pk` 的结构化匹配的返回结果为未知行为。该限制将在未来版本中修复。

主要改进

- 故障恢复速度:
  - #10737 实现 Proxy Session Checker。
  - #10723 修复寻求 `queryChannel` 错误。
  - #10907 修复 `LatestPosition` 选项与最早补丁冲突的问题。
  - #10616 删除 Common YAML 文件。
  - #10771 将 channel 查找起始位置更改为所有 segment 的最早先的检查点。
  - #10651 修复 query coord 设置查找位置错误。
  - #9543 初始化 global sealed segment 并在 `AddQueryChannel` 时寻找查询通道。
  - #9684 Data coord 重启时，避免重复消耗 timetick MsgStream。
- 重构 meta 快照:
  - #10288 减少存储在 `SnapshotMeta` 中的信息。

  - #10703 修复因兼容问题导致创建 meta table 失败。
  - #9778 简化 `meta_snapshot` 接口。
- #10563 修改默认平衡策略。
- #10730 获取查询 segment 信息时返回 segment 状态。
- #10534 支持从环境变量中读取 MinIO 配置。
- #10114 设定默认 `gracefulTime` 为 `0`。
- #9860 将 `liveChn` 隐藏至 `sessionutil` 并修复存活初始化顺序。
- #7115 使用 etcd 监听 data node。
- #7606 使 `knowhere` 独立编译。

新增功能

- Handoff:
  - #10330 添加 `handoffTask`。
  - #10084 向 `queryChannel` 发布 `sealedSegmentChangeInfo`。
  - #10619 修复当 query node 收到 `segmentChangeInfo` 时删除 segment。
  - #10045 监听 query node 中的 `changeInfo`。
  - #10011 当收到 `changeInfo` 时，更新被排除的 segment 信息。
  - #9606 为 `AddQueryChannelRequest` 添加初始化信息。
- Primary Key 去重:
  - #10834 在 query node 中删除 primary key 重复的查询结果。
  - #10355 #10967 在 proxy 中删除 primary key 重复的查询结果。
  - #10117 在 segcore reduce 中删除 primary key 重复的查询结果。
  - #10949 仅使用 primary key 检测重复的查询结果。
- Auto-flush:
  - #10659 为 `flushManager` 接口添加 `injectFlush` 方法。
  - #10580 为 `FlushManager` 添加 injection 逻辑。
  - #10550 为同一 ID 的 segment 合并自动和手动 flush。
  - #10539 允许已 flush 的 segment 触发 flush 流程。
  - #10197 添加定时 flush 触发机制。
  - #10142 在 data node 中应用 flush manager 逻辑。
  - #10075 使用单信号 channel 提示 flush。
  - #9986 添加 flush manager 结构。
- #10173 添加 binlog 迭代器。
- #10193 更改 bloom filter 使用 primary key。
- #9782 为 data node allocator 添加 `allocIDBatch`。

问题修复

- 当内存资源不足时 collection 加载行为错误:
  - #10796 修复获取 container 内存使用问题。
  - #10800 在 `GetContainerMemUsed` 中使用 `TotalInactiveFile`。
  - #10603 为 `EstimateMemorySize` 接口增加兼容。
  - #10363 添加 `cgroups` 以获取 container 内存并检查 segment loader 中的索引内存。
  - #10294 使用 proto 大小计算请求大小。
  - #9688 使用 descriptor event 估算内存大小。
  - #9681 修复 binlog 存储原始内存大小的方式问题。
  - #9628 在 extra information 中存储 binlog 原始内存大小。
- etcd 相关请求过大:

  - #10909 修复当存储 `segmentInfo` 时 `txn` 操作过多问题。
  - #10812 修复当加载 segment 时请求过大问题。
  - #10768 修复当加载 collection 时请求过大问题。
  - #10655 拆分监听操作。
  - #10587 精简 `multiSegmentChangeInfo` 为单条信息。
  - #10425 针对使用 `VChaninfo` 调整 `segmentinfo` binlog 。
  - #10340 修复 etcd `multiSave` `childTask` 失败。
  - #10310 修复分配加载 segment 请求错误。
  - #10125 拆分大型 `loadSegmentReq` 为多个小型请求。
- 系统崩溃:
  - #10832 添加查询 `mutex` 修复崩溃问题。
  - #10821 调整 index node 在 index coord 修改 meta 前完成任务。
  - #10182 修复当 flush segment 时系统崩溃。
  - #10681 修复当更新 `querychannelInfo` 时 query coord 崩溃。
- RocksMQ 相关问题:
  - #10367 优雅关闭 retention。
  - #9828 修复 retention 时的数据竞争。
  - #9933 修改 retention ticker time 为 10 分钟。
  - #9694 在删除 RocksMQ metadata 前删除信息。
  - #11029 修复 RocksMQ `SeekToLatest` 问题。
  - #11057 修复 `SeekToLatest` 内存泄漏并删除冗余逻辑。
  - #11081 修复 RocksMQ retention 未设定 ts。
  - #11083 为 RocksMQ `Seek` 添加 topic。
  - #11076 在 retention 过期清理中将 topic lock 移至最终删除之前。
- #10751 当 `indexFilePathInfo` 收到空列表时 `loadIndex` 不断重试。
- #10583 `ParseHybridTs` 返回数据类型问题。
- #10599 删除信息哈希错误。
- #10314 索引构建任务因 index coord 错误被取消。
- #9701 `CreateAlias/DropAlias/AlterAlias` 实现错误。
- #9573 Data coord 储存 binlog 超时。
- #9788 监听 channel 因网络问题被取消。
- #10994 Index node 无法平衡负载。
- #11152 当使用 Time Travel 搜索时不传过滤条件并调用 `num_entities` 搜索出错。
- #11249 #11277 Query node 死锁。
- #11222 空检索结果处理错误。

v2.0.0-RC7

发布时间：2021-10-11

版本兼容

Milvus 版本

Python SDK 版本

Java SDK 版本

Go SDK 版本

Node.js SDK 版本

2.0.0-RC7

2.0.0rc7

即将上线

即将上线

2.0.0

Milvus 2.0.0-RC7 是 2.0 的预览版本。该版本支持 collection 别名，PChannel 共享 `msgstream`，将默认 MinIO 与 Pulsar 依赖更改为分布式版本，并修复了一系列资源泄露、死锁等问题。

由于对存储格式进行了一些更改，Milvus 2.0.0-RC7 与早先的 RC 版本不兼容。

主要改进

- #8215 为 query coord 中 `interTask` 添加最大重试次数。
- #9459 实现 collection 起始位置。
- #8721 为日志名称添加节点 ID。
- #8940 将流式 segment 内存添加到 `checkLoadMemory` 中的已用内存。
- #8542 使用 `proto.Marshal` 替换 `proto.MarshalTextString`。
- #8770 重构 flowgraph 以及相关调用。
- #8666 更改 CMake 版本。
- #8653 更新 `getCompareOpType`。
- #8697 #8682 #8657 开启 segment 时实现 collection 起始位置。
- #8608 更改 segment 副本结构。
- #8565 重构缓存大小计算。
- #8262 添加 `segcore` 日志记录器。
- #8138 在 `insertBufferNode` 中添加 `BufferData`。
- #7738 实现通过 `msgstream` 池为创建 collection 分配 `msgstream`。
- #8054 优化 `insertBufferNode` 代码。
- #7909 升级 `pulsar-client-go` 至 0.6.0 版本。
- #7913 转移 segcore rows_per_chunk 配置项至 query_node.yaml。
- #7792 从 `LongTermChecker` 中删除 `ctx`。
- #9269 优化表达式写法。
- #8159 修改 `FlushSegments` 为异步。
- #8278 重构 rocksmq 关闭逻辑并优化代码覆盖率。
- #7797 标注代码参数类型。

新增功能

- #9579 在 `getSystemInfoMetrics` 中添加副本缓存大小以及 `cacheSize` 。
- #9556 添加 `ProduceMark` 接口以返回 message ID。
- #9554 添加 `LoadPartial` 接口以支持 DataKV。
- #9471 支持通过 collection ID 调用 `DescribeCollection`。
- #9451 将 index 参数存储至 descriptor event。
- #8574 为搜索功能添加 `round_decimal` 参数以控制精度。
- #8947 Rocksmq 支持 `SubscriptionPositionLatest`。

- #8919 索引文件过大时，拆分为多行字符串。
- #8914 Binlog 解析器工具支持索引文件。
- #8514 重构索引文件格式。
- #8765 添加 `cacheSize` 以防止 query node 内存资源不足。
- #8673 #8420 #8212 #8272 #8166 支持多个 Milvus 集群共享 Pulsar 以及 MinIO。
- #8654 为 `Msgstream` 添加 `BroadcastMark` 以返回 Message ID。
- #8586 将 message ID 返回值添加至 producer 中。
- #8408 #8363 #8454 #8064 #8480 添加 session 存活检测。
- #8264 添加 description event 附加内容。
- #8341 在 root coord 中使用 `Marshal` 替代 `MarshalTextString`。
- #8228 支持 healthz 检测 API。
- #8276 初始化 index node 时初始化 SIMD 类型。
- #7967 添加 knowhere.yaml 以支持配置 knowhere。
- #7974 支持设定任务队列最大任务数。
- #7948 #7975 添加 `suffixSnapshot` 以实现 SnapshotKV。
- #7942 支持配置 SIMD 类型。
- #7814 在搜索以及结构化匹配中支持布尔值过滤器。
- #7635 支持通过配置文件设定 segcore rows_per_chunk。

问题修复

- #9572 调用 `DeleteRange` 后 Rocksdb 不删除 end key。
- #8735 Acked 信息占用内存资源。
- #9454 Query service 发生数据竞争。
- #8850 使用别名删除 collection 时，SDK 报错。
- #8930 由于从 `insertBuf` 中即时删除缓冲，导致当 `SaveBinlogPath` 调用失败时，flush 偶尔会卡住。
- #8868 跟踪日志捕获错误的文件名和行号。
- #8844 `SearchTask` 返回结果为空。
- #8835 应 pulsar-client-go 存在 bug 导致 Root coord 崩溃。
- #8780 #8268 #7255 集合别名相关问题。
- #8744 Rocksdb_kv 错误进程。
- #8752 mqconsumer 中发生数据竞争。
- #8686 Auto-flush 之后 flush 无法完成。
- #8564 #8405 #8743 #8798 #9509 #8884 Rocksdb 内存泄漏。
- #8671 调用删除之后对象没有被从 MinIO 中删除。
- #8050 #8545 #8567 #8582 #8562 tsafe 相关问题。
- #8137 因为 TSO 没有加载最新时间戳导致时间倒退。
- #8461 Data coord 中可能发生数据竞争。
- #8386 为 data node 分配 dm channel 的逻辑不完整。
- #8206 Proxy 搜索任务中结果合并算法错误。

- #8120 Root coord 中可能发生数据竞争。
- #8068 当查询结果为空且 `retrieve_ret_` 未被初始化时，query node 崩溃。
- #8060 查询任务崩溃。
- #8091 Proxy gRPC 客户端发生数据竞争。
- #8078 Root coord gRPC 客户端发生数据竞争。
- #7730 `CloseRocksMQ` 后 topic 和 ConsumerGroup 仍然存在。
- #8188 释放 collection 逻辑错误。

v2.0.0-RC6

发布时间：2021-09-10

版本兼容

Milvus 版本

Python SDK 版本

Java SDK 版本

Go SDK 版本

Node.js SDK 版本

2.0.0-RC6

2.0.0rc6

即将上线

即将上线

2.0.0

Milvus 2.0.0-RC6 是 2.0 的预览版本。该版本支持创建 collection 时设定 shard 数量，以及通过表达式进行结构性匹配。RC5 通过 API 进一步暴露分布式版指标。在该版本我们增加单元测试覆盖率至 80%，并修复了一系列资源泄露、系统错误等问题。

主要改进

- 增加单元测试覆盖率至 80%。

新增功能

- #7482 支持创建 collection 时设定 shard 数量。
- #7386 支持通过表达式进行结构性匹配。
- 通过 API 暴露系统指标
  - #7400 Proxy 指标与其他 coordinator 集成。
  - #7177 暴露 data node 以及 data coord 指标。
  - #7228 暴露 root coord 指标。
  - #7472 暴露更多详细指标信息。
  - #7436 支持缓存系统信息指标。

问题修复

- #7434 加载超过内存上限的 collection 导致 Query node 内存不足。
- #7678 从现有存储中恢复导致单机版 Milvus 内存不足。
- #7636 向已关闭的 channel 发送消息导致单机版 Milvus 发生错误。
- #7631 关闭 flowgraph 导致 Milvus 发生错误。
- #7605 运行每日 CI 测试时 Milvus 报错崩溃。
- #7596 Root coord 与 etcd 断联导致每日测试失败。
- #7557 表达式中内容顺序错误导致系统返回错误结果。
- #7536 `MqMsgStream` 查找逻辑错误。

- #7527 搜索时 `knowhere` 中数据集内存泄漏。
- #7444 Channels time ticker 死锁。
- #7428 `MqMsgStream` 广播失败时可能出现死锁。
- #7715 结构性匹配请求被同一 slice 上并发操作覆盖。

v2.0.0-RC5

发布时间：2021-08-30

版本兼容

Milvus 版本

Python SDK 版本

Java SDK 版本

Go SDK 版本

Nodejs SDK 版本

2.0.0-RC5

2.0.0rc5

即将上线

即将上线

2.0.0

Milvus 2.0.0-RC5 是 2.0 的预览版本。该版本支持 message queue 数据保留机制和 etcd 数据清理，通过 API 暴露分布式版指标，并为后续支持删除操作做准备。RC5 在系统稳定性方面也取得了很大的进步。该版本修复了一系列资源泄露、操作卡死、以及 Milvus 集群下单机 Pulsar 的配置错误等问题。

主要改进

- #7226 重构 data coord allocator。
- #6867 添加 connection manager。
- #7172 添加 seal 策略以限制 segment 的生命周期。
- #7163 增加创建索引时 gRPC 连接的超时时间。
- #6996 添加 segment flush 的最小间隔。
- #6590 在 `SegmentInfo` 中保存 binlog 路径。
- #6848 移除 `RetrieveRequest` 和 `RetrieveTask`。
- #7102 支持搜索输出向量 field。
- #7075 重构 `NewEtcdKV` API.
- #6965 为 data node 添加 channel 以监听 etcd。
- #7066 优化搜索聚合逻辑。
- #6993 针对解析 gRPC 收发参数增强日志系统。
- #7331 修改 context 至正确的 package。
- #7278 每 1000 次修订后启用 etcd 自动压缩。
- #7355 从 util/flowgraph 中清除 `fmt.Println`。

新增功能

- #7112 #7174 引入嵌入式 etcdKV（第一阶段完成）。
- #7231 添加 segment filter 接口。
- #7157 暴露 index coord 和 index nodes 的 metrics 信息。
- #7137 #7157 通过 proxy 暴露系统拓扑信息。
- #7113 #7157 暴露 query coord 和 query nodes 的指标信息。
- #7134 允许用户仅使用内存执行向量搜索。
- #6617 为 Rocksmq 添加 log 保留策略。
- #7303 添加 query node segment filter。
- #7304 在 proto 中添加 `delete` API。
- #7261 添加 delete node。

- #7268 插入数据时搭建 Bloom filter。

问题修复

- #7272 #7352 #7335 若已创建索引，则无法使用现有 volume 启动新的 Docker 容器：proxy 不健康。
- #7243 旧版本插入的数据在新版本 Milvus 中创建索引失败。
- #7253 释放不同的 partition 后，搜索结果为空。
- #7244 #7227 收到空搜索结果时 proxy 崩溃。
- #7203 gRPC 服务器关闭时连接卡住。
- #7188 单元测试逻辑不完整。
- #7175 未加载的情况下使用 collection ID 计算距离时返回的错误消息不明确。
- #7151 由于缺少 `DropCollection` 导致 data node flowgraph 不关闭。
- #7167 无法加载 IVF_FLAT 索引。
- #7123 "Timestamp go back" 问题。
- #7140 使用谷本距离计算相似度时，`calc_distance` 会返回错误的二元向量结果。
- #7143 KV 操作失败时，内存和 etcd 的状态不一致。
- #7141 #7136 当 index node pod 频繁重启时，索引构建会卡住。
- #7119 当使用相同的 topic 和 sub name 订阅时，Pulsar `msgStream` 可能会卡住。
- #6971 使用 HNSW 索引搜索时发生异常。
- #7104 如果 query node 只加载 sealed segment 而未监听 insert channel，搜索会卡住。
- #7085 Segment 无法自动 flush。
- #7074 Index node 等待至 index coord 启动后完成操作。
- #7061 如果 data coord 没有收到来自 data node 的 timetick 消息，则 segment allocation 不会过期。
- #7059 Query nodes 发生 producer 泄漏。
- #7005 当 `loadSegmentInternal` 失败时，query node 不会向 query coord 返回错误。
- #7054 当 `topk` 大于 `row_num.` 时，query node 返回错误的 ID。
- #7053 Allocation 逻辑不完整。
- #7044 在检索本地存储中的向量之前，未对内存中未建索引向量的检查。
- #6862 Data node 的 flush `cache` 内存泄露。
- #7346 重启分布式版 Milvus 后 query coord 容器在一分钟内退出。
- #7339 表达式边界问题。
- #7311 添加 `queryCollection` 时 collection 为空。
- #7266 Flowgraph 内存释放错误。
- #7310 释放和加载 partition 后搜索时 timeout 过长。
- #7320 嵌入式 etcd 和外部 etcd 之间的端口冲突。
- #7336 Data node 边界情况。

v2.0.0-RC4

发布时间：2021-08-13

版本兼容

| Milvus 版本 | Python SDK 版本 | Java SDK 版本 | Go SDK 版本 |
|---|---|---|---|
| 2.0.0-RC4 | 2.0.0rc4 | 即将上线 | 即将上线 |

Milvus 2.0.0-RC4 是 2.0 的预览版本。该版本主要修复了稳定性问题，并新增从对象存储中检索向量数据以及通过通配符匹配指定输出 field 的功能。

主要改进

- #6984 #6772 #6704 #6652 #6536 #6522 优化单元测试。
- #6859 提升 gRPC 客户端 `MaxCallRecvMsgSize` 和 `MaxCallSendMsgSize` 的上限。
- #6796 修复 MsgStream 指数重试策略。
- #6897 #6899 #6681 #6766 #6768 #6597 #6501 #6477 #6478 #6935 #6871 #6671 #6682 优化日志系统。
- #6440 重构 segment manager。

- #6421 创建索引时将原始向量拆分为几个较小的 binlog 文件。
- #6466 区分 query 和 search 的概念和使用。
- #6505 将 RetrieveRequest 中 `output_fields` 修改为 `out_fields_id` 。
- #6427 重构 index coord 的任务分配逻辑。
- #6529 #6599 重构时间戳统计的快照。
- #6692 #6343 创建 collection/partition 记录时间信息。
- #6629 为 etcdKV 添加 `WatchWithVersion` 接口。
- #6666 重构 expression executor 以使用单个 bitset。
- #6664 当分配的行数超过每个 segment 的最大行数时，自动创建新 segment。
- #6786 重构 `RangeExpr` 和 `CompareExpr`.
- #6497 放宽二元向量 field 搜索时的维度下限。

新增功能

- #6706 支持从磁盘读取向量。
- #6299 #6598 支持查询向量 field。
- #5210 扩展布尔表达式的语法。
- #6411 #6650 搜索/查询输出 field 支持通配符和通配符匹配。
- #6464 添加向量 chunk manager 以支持向量文件本地存储。
- #6701 为通过 Docker Compose 部署的 Milvus 添加数据持久化支持。
- #6767 为 Milvus 添加 Grafana 仪表盘 .json 文件。

问题修复

- #5443 从 collection 中获取向量时，`CalcDistance` 返回错误的结果。
- #7004 Pulsar 消费者导致 goroutine 泄漏。
- #6946 当 Flow Graph 在 `start()` 之后立即 `close()` 时，会发生数据竞争。
- #6903 在 query coord 中使用 `proto marshal` 以替代 `marshalTextString` 来避免由未知 field 名称崩溃触发的崩溃。
- #6374 #6849 加载 collection 失败。
- #6977 删除 partition/collection 后，搜索返回错误限制。
- #6515 #6567 #6552 #6483 Data node BackGroundGC 不运作并导致内存泄漏。
- #6943 MinIOKV `GetObject` 方法不会关闭客户端并导致每次调用产生 goroutine 泄漏。
- #6370 因加载 partition 提供的错误语义导致搜索卡住。
- #6831 Data node 在元服务中崩溃。
- #6469 当限制（`topK`）大于插入 entity 的数量时，使用汉明距离搜索二进制结果错误。
- #6693 因超时引起的 segment 竞争情况。
- #6097 短时间内频繁重启 query node 后导致加载卡住。
- #6464 处理 Data sorter 边界情况。
- #6419 Milvus 在插入空向量时崩溃。
- #6477 不同的组件在 MinIO 中重复创建桶。

- #6377 在部署了多个 query node 的情况下，由于从 etcd 获取的 globalSealedSegment 信息不正确导致 Milvus 集群查询结果返回不完整。
- #6499 TSO 分配错误的时间戳。
- #6501 Data node 崩溃后 channel 丢失。
- #6527 无法从 etcd 中删除 `watchQueryChannels` 的任务信息。
- #6576 #6526 检索 entity 时会添加重复的 primary field ID。
- #6627 #6569 当新记录的距离为 NaN 时，`std::sort` 无法正常过滤搜索结果。
- #6655 调用检索任务时 proxy 崩溃。
- #6762 Collection/partition 的创建时间戳不正确。
- #6644 Data node 自动重启失败。
- #6641 与 etcd 断开连接时无法停止 data coord。
- #6621 在插入的数据大小大于 segment 时，Milvus 抛出异常。
- #6436 #6573 #6507 时间同步处理不正确。
- #6732 创建 IVF-PQ 索引失败。

v2.0.0-RC2

发布时间: 2021-07-13

版本兼容

| Milvus 版本 | Python SDK 版本 | Java SDK 版本 | Go SDK 版本 |
|---|---|---|---|
| 2.0.0-RC2 | 2.0.0rc2 | 即将上线 | 即将上线 |

Milvus 2.0.0-RC2 是 2.0 的预览版本。该版本修复了 RC1 版本的稳定性和性能问题，并针对节点和存储管理进行了代码重构。

主要改进

- #6356 Data coordinator 集群代码重构。
- #6300 Data coordinator 元数据管理代码重构。
- #6289 `SegmentIndexInfo` 新增 `collectionID` 和 `partitionID` 信息。
- #6258 调用 `releaseCollection()` 方法时清除 proxy 中对应的 `searchMsgStream`。
- #6227 合并 query node 召回和查询的相关代码。
- #6196 Data coordinator 新增候选管理，用于维护管理 data node 集群。
- #6188 新增“使用 Docker Compose 安装”的相关技术文档。

新增功能

- #6386 支持调用 `fget_object()` 方法从 MinIO 加载文件到本地设备。
- #6253 支持在 data coordinator 调用 `GetFlushedSegments()` 方法。
- #6213 新增 `GetIndexStates()` 方法。

问题修复

- #6184 数据集规模增加导致查询准确性下降。
- #6308 NSG 索引的 KNNG 参数未达到满值会导致服务器崩溃。
- #6212 Query node 重启后查询操作宕机。
- #6265 服务器检测到节点在线后不检查节点状态。
- #6359 #6334 在 CentOS 系统上编译 Milvus 出现编译错误。

v2.0.0-RC1

发布时间：2021-06-28

版本兼容

| Milvus 版本 | Python SDK 版本 | Java SDK 版本 | Go SDK 版本 |
|---|---|---|---|
| 2.0.0-RC1 | 2.0.0rc1 | 即将上线 | 即将上线 |

Milvus 2.0.0-RC1 是 2.0 的预览版本。该版本引入 Go 语言搭建分布式系统，并采用了新的云原生分布式设计。后者大大提高了系统扩展性和系统弹性。

系统架构

Milvus 2.0 是一款云原生向量数据库，采用存储与计算分离的架构设计。该重构版本的所有组件均为无状态组件，极大地增强了系统弹性和灵活性。

整个系统分为四个层面：

- 接入层（Access Layer）
- 协调服务（Coordinator Service）
- 执行节点（Worker Node）
- 存储服务（Storage）

接入层 Access Layer：系统的门面，包含了一组对等的 proxy 节点。接入层是暴露给用户的统一 endpoint，负责转发请求并收集执行结果。

协调服务（Coordinator Service）：系统的大脑，负责分配任务给执行节点。总共有四类协调者角色，分别为 root 协调者、data 协调者、query 协调者和 index 协调者。

执行节点（Worker Node）：系统的四肢。执行节点只负责被动执行协调服务发起的读写请求。目前有三类执行节点，即 data 节点、query 节点和 index 节点。

存储服务（Storage）：系统的骨骼，是所有其他功能实现的基础。Milvus 依赖三类存储：元数据存储、消息存储（Log Broker）和对象存储。

> 更多系统原理的相关内容详见 Milvus 2.0 架构。

新增功能

SDK

- PyMilvus

  PyMilvus API 直接在 collection、partion 和 index 对象上进行操作。用户可专注于搭建业务数据模型，而不必担心具体实现。

核心功能

- 标量和向量数据混合查询

  Milvus 2.0 支持存储标量数据。支持使用大于、小于、等于、NOT、IN、AND、OR 等运算符在向量搜索之前进行标量过滤。当前支持的数据类型包括 bool、int8、int16、int32、int64、float 和 double。后期版本将逐步支持字符串和 VARBINARY 数据类型。

- 匹配查询（Match Query）

  与返回相似结果的搜索操作不同，匹配查询操作返回完全匹配表达式的对象，可用于按 primary key 或按搜索条件查询向量。

- 多一致性

- 分布式数据库需在一致性与可用性以及一致性与延迟之间进行权衡。Milvus 提供四种一致性级别，从强到弱分别为：强一致性 (Strong)、有界一致性 (Bounded Staleness)、会话一致性 (Session) 、前缀一致性（Consistent Prefix)。用户可以通过指定时间戳自定义读取一致性。一般情况下，一致性级别越弱，可用性越高，性能也越好。

- 时间旅行（Time Travel）

  通过时间旅行可以访问指定时间段内任意时刻的历史数据。用户可使用该功能查询、恢复和备份历史数据。

其他

- 支持基于 helm 和 docker-compose 一键部署 Milvus 2.0。
- 使用 Prometheus 和 Grafana 实现数据监测和报警功能。
- Milvus Insight

  Milvus Insight 是 Milvus 图形化管理工具，包含了集群状态可视化、元数据管理、数据查询等实用功能。Milvus Insight 源码未来也会作为独立项目开源。

不兼容改动

Milvus 2.0 使用的编程语言、数据格式以及分布式架构都与之前的版本完全不同，这意味着不能从之前的 Milvus 版本升级到 2.x 版本。不过，Milvus 1.x 是长期支持版本（LTS），相关的数据迁移工具将尽快上线。

具体改动如下:

- 暂不支持 JAVA、Go 和 C++ SDK。
- 暂不支持删除和更新操作。
- PyMilvus 不支持 force flush。
- 数据格式与之前版本不兼容。
- 废弃 Mishards ——Milvus 2.0 为分布式架构，无需分片中间件。
- 暂不支持本地文件存储和分布式系统存储。

# 快速开始

## 安装前提

Environment Checklist

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

Before you install Milvus, check your hardware and software to see if they meet the requirements.

使用 Docker Compose 安装使用 Kubernetes 安装

Hardware requirements

| Component | Requirement | Recommendation | Note |
| --- | --- | --- | --- |
| CPU | Intel CPU Sandy Bridge or later | | Current version of Milvus does not support AMD and Apple M1 CPUs. |
| CPU instruction set | | | Vector similarity search and index building within Milvus require CPU’s support of single instruction, multiple data (SIMD) extension sets. Ensure that the CPU supports at least one of the SIMD extensions listed. See CPUs with AVX for more information. |
| RAM | | | The size of RAM depends on the data volume. |

| Component | Requirement | Recommendation | Note |
|---|---|---|---|
| Hard drive | SATA 3.0 SSD or higher | SATA 3.0 SSD or higher | The size of hard drive depends on the data volume. |

Software requirements

| Operating system | Software | Note |
|---|---|---|
| macOS 10.14 or later | Docker Desktop | Set the Docker virtual machine (VM) to use a minimum of 2 virtual CPUs (vCPUs) and 8 GB of initial memory. Otherwise, installation might fail. See Install Docker Desktop on Mac for more information. |
| Linux platforms | | See Install Docker Engine and Install Docker Compose for more information. |
| Windows with WSL 2 enabled | Docker Desktop | We recommend that you store source code and other data bind-mounted into Linux containers in the Linux file system instead of the Windows file system.See Install Docker Desktop on Windows with WSL 2 backend for more information. |

What’s next

- If your hardware and software meet the requirements, you can:
  - Install Milvus standalone with Docker Compose
  - Install Milvus cluster with Docker Compose
- See System Configuration for parameters you can set while installing Milvus.

## 安装 Milvus

安装单机版 Milvus

安装 Milvus 单机版

你可以使用 Docker Compose 或 Kubernetes 安装 Milvus 单机版。安装前，请先阅读安装前提。

你也可以从源代码编译 Milvus。

Docker Compose 安装 Helm 安装 APT 或 YUM 安装

安装 Milvus 单机版

1. 下载 milvus-standalone-docker-compose.yml 配置文件并保存为 docker-compose.yml

```
wget https://github.com/milvus-io/milvus/releases/download/v2.0.0/milvus-standalone-docker-compose.yml -O
```

   你可以在 GitHub 直接 下载 docker-compose.yml。

如果你使用原始 docker-compose.yml 文件安装 Milvus, 数据将会被存储在./volume 路径下。如需修改映射路径，你可以直接修改 docker-compose.yml 文件，或运行 $ export DOCKER_VOLUME_DIRECTORY=。

2. 启动 Milvus 单机版：

```
$ sudo docker-compose up -d
```

```
Docker Compose is now in the Docker CLI, try `docker compose up`
Creating milvus-etcd  ... done
Creating milvus-minio ... done
Creating milvus-standalone ... done
```

如果 Milvus 单机版启动正常，可以看到有 3 个 Docker 容器在运行（2 个为基础服务，1 个为 Milvus 服务）：

```
$ sudo docker-compose ps
      Name                     Command                  State                            Ports
-------------------------------------------------------------------------------------------------------------
milvus-etcd         etcd -listen-peer-urls=htt ...   Up (healthy)   2379/tcp, 2380/tcp
milvus-minio        /usr/bin/docker-entrypoint ...   Up (healthy)   9000/tcp
milvus-standalone   /tini -- milvus run standalone   Up             0.0.0.0:19530->19530/tcp,:::19530->195
```

运行 $ sudo docker-compose down 停止 Milvus 单机版。

如果你想在停止 Milvus 后清理数据，运行 $ sudo rm -rf volume。

阅读 升级指南 2.0 了解如何升级 Milvus 2.0 版本。

安装分布式版 Milvus

安装 Milvus 分布式版

你可以使用 Docker Compose 或 Kubernetes 安装 Milvus 分布式版。安装前，请先阅读安装前提。

你也可以从源代码编译 Milvus。

Docker ComposeHelmMilvus Operator

安装 Milvus 分布式版

1. 下载 Docker Compose 配置文件 docker-compose.yml：

```
$ wget https://github.com/milvus-io/milvus/releases/download/v2.0.0/milvus-cluster-docker-compose.yml  -O
```

你可以在 GitHub 直接 下载 docker-compose.yml。

如果你使用原始 docker-compose.yml 文件安装 Milvus, 数据将会被存储在./volume 路径下。如需修改映射路径，你可以直接修改 docker-compose.yml 文件，或运行 $ export DOCKER_VOLUME_DIRECTORY=。

2. 启动 Milvus 分布式版：

```
$ sudo docker-compose up -d

Docker Compose is now in the Docker CLI, try `docker compose up`
Creating milvus-etcd   ... done
Creating milvus-minio  ... done
Creating milvus-pulsar ... done
Creating milvus-proxy      ... done
Creating milvus-rootcoord  ... done
Creating milvus-indexcoord ... done
Creating milvus-querycoord ... done
Creating milvus-datacoord  ... done
Creating milvus-querynode  ... done
Creating milvus-indexnode  ... done
Creating milvus-datanode   ... done
```

如果 Milvus 分布式版启动正常，可以看到有 11 个 docker 容器在运行（3 个为基础服务，8 个为 Milvus 服务）

```
$ sudo docker ps
      Name                     Command                  State                            Ports
-------------------------------------------------------------------------------------------------------------
milvus-datacoord    /tini -- milvus run datacoord    Up
milvus-datanode     /tini -- milvus run datanode     Up
milvus-etcd         etcd -listen-peer-urls=htt ...   Up (healthy)   2379/tcp, 2380/tcp
milvus-indexcoord   /tini -- milvus run indexcoord   Up
milvus-indexnode    /tini -- milvus run indexnode    Up
milvus-minio        /usr/bin/docker-entrypoint ...   Up (healthy)   9000/tcp
```

```
milvus-proxy        /tini -- milvus run proxy        Up             0.0.0.0:19530->19530/tcp,:::19530->195
milvus-pulsar       bin/pulsar standalone            Up
milvus-querycoord   /tini -- milvus run querycoord   Up
milvus-querynode    /tini -- milvus run querynode    Up
milvus-rootcoord    /tini -- milvus run rootcoord    Up
```

运行 $ sudo docker-compose down 停止 Milvus 分布式版。

如果你想在停止 Milvus 后清理数据，运行 $ sudo rm -rf volume。

阅读 升级指南 2.0 了解如何升级 Milvus 2.0 版本。

离线安装

离线安装 Milvus

本篇文档将展示如何在离线环境中部署 Milvus。相关文件可在 GitHub 下载。

使用 Docker Compose 安装使用 Kubernetes 安装

下载 Docker 镜像

镜像加载错误可能会导致安装 Milvus 失败。离线安装 Milvus 需要拉取保存所有镜像并转移至目标主机手动加载。

1. 下载 Milvus 配置文件 docker-compose.yml:

- 单机版 Milvus

```
$ wget https://github.com/milvus-io/milvus/releases/download/v2.0.0/milvus-standalone-docker-compose.yml -
$ wget https://raw.githubusercontent.com/milvus-io/milvus/master/deployments/offline/requirements.txt
$ wget https://raw.githubusercontent.com/milvus-io/milvus/master/deployments/offline/save_image.py
```

- 分布式版 Milvus

```
wget https://raw.githubusercontent.com/milvus-io/milvus/master/deployments/docker/cluster/docker-compose.y
$ wget https://raw.githubusercontent.com/milvus-io/milvus/master/deployments/offline/requirements.txt
$ wget https://raw.githubusercontent.com/milvus-io/milvus/master/deployments/offline/save_image.py
```

2. 拉取并保存 Docker 镜像:

```
pip3 install -r requirements.txt
python3 save_image.py --manifest docker-compose.yml
```

Docker 镜像文件将存储在 images 路径下。

3. 加载 Docker 镜像

```
cd images/for image in $(find . -type f -name "*.tar.gz") ; do gunzip -c $image | docker load; done
```

安装 Milvus

离线安装 Milvus:

```
docker-compose -f docker-compose.yml up -d
```

卸载 Milvus

卸载 Milvus:

```
docker-compose -f docker-compose.yml down
```

## 安装 SDK

Install Milvus SDK

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to install Milvus SDK for Milvus.

Current version of Milvus supports SDKs in Python, Node.js, GO, and Java.

Install PyMilvusInstall Node.js SDKInstall GO SDKInstall Java SDK

Requirement

Python 3 (3.71 or later) is required.

Install PyMilvus via pip

PyMilvus is available in Python Package Index.

It is recommended to install a PyMilvus version that matches the version of the Milvus server you installed. For more information, see Release Notes.

```
$ python3 -m pip install pymilvus==2.0.0
```

Verify installation

If PyMilvus is correctly installed, no exception will be raised when you run the following command.

```
$ python3 -c "from pymilvus import Collection"
```

What’s next

Having installed PyMilvus, you can:

- Learn the basic operations of Milvus:
  - Connect to Milvus server
  - Conduct a vector search
  - Conduct a hybrid search
- Explore PyMilvus API reference

## Hello Milvus

PythonNode.js

使用 Python 运行 Milvus

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

该篇文章介绍了如何使用 Python 运行 Milvus.

通过运行我们提供的示例代码，您将初步了解 Milvus 的功能。

准备工作

- Milvus 2.0.0
- Python 3 (3.71 或者更高版本)
- PyMilvus 2.0.0

下载示例代码

使用下面的命令 下载 hello_milvus.py

```
$ wget https://raw.githubusercontent.com/milvus-io/pymilvus/v2.0.0/examples/hello_milvus.py
```

代码详解

示例代码将执行以下步骤:

- 导入 PyMilvus 包:

```
from pymilvus import (
    connections,
    utility,
    FieldSchema,
    CollectionSchema,
    DataType,
    Collection,
)
```

- 连接服务:

```
connections.connect("default", host="localhost", port="19530")
```

- 创建一个 Collection:

```
fields = [
    FieldSchema(name="pk", dtype=DataType.INT64, is_primary=True, auto_id=False),
    FieldSchema(name="random", dtype=DataType.DOUBLE),
    FieldSchema(name="embeddings", dtype=DataType.FLOAT_VECTOR, dim=8)
]
schema = CollectionSchema(fields, "hello_milvus is the simplest demo to introduce the APIs")
hello_milvus = Collection("hello_milvus", schema)
```

- 在创建好的 Collection 中插入向量:

```
import random
entities = [
    [i for i in range(3000)],  ### field pk
    [float(random.randrange(-20, -10)) for _ in range(3000)],  ### field random
    [[random.random() for _ in range(8)] for _ in range(3000)],  ### field embeddings
]
insert_result = hello_milvus.insert(entities)
```

- 在数据上构建索引:

```
index = {
    "index_type": "IVF_FLAT",
    "metric_type": "L2",
    "params": {"nlist": 128},
}
hello_milvus.create_index("embeddings", index)
```

- 将 Collection 加载到内存并执行相似搜索:

```
hello_milvus.load()
vectors_to_search = entities[-1][-2:]
search_params = {
    "metric_type": "l2",
    "params": {"nprobe": 10},
}
result = hello_milvus.search(vectors_to_search, "embeddings", search_params, limit=3, output_fields=["rand
```

- 执行结构化查询：

```
result = hello_milvus.query(expr="random > -14", output_fields=["random", "embeddings"])
```

- 执行混合查询：

```
result = hello_milvus.search(vectors_to_search, "embeddings", search_params, limit=3, expr="random > -12",
```

- 根据 pk 删除数据：

```
expr = f"pk in [{ids[0]}, {ids[1]}]"
hello_milvus.delete(expr)
```

- 删除 Collection：

```
utility.drop_collection("hello_milvus")
```

运行示例代码

执行以下命令，运行示例代码：

```
$ python3 hello_milvus.py
```

运行结果如下所示：

```
=== start connecting to Milvus     ===

Does collection hello_milvus exist in Milvus: False

=== Create collection `hello_milvus` ===


=== Start inserting entities       ===

Number of entities in Milvus: 3000

=== Start Creating index IVF_FLAT  ===


=== Start loading                  ===


=== Start searching based on vector similarity ===

hit: (distance: 0.0, id: 2998), random field: -11.0
hit: (distance: 0.11455299705266953, id: 1581), random field: -18.0
hit: (distance: 0.1232629269361496, id: 2647), random field: -13.0
hit: (distance: 0.0, id: 2999), random field: -11.0
hit: (distance: 0.10560893267393112, id: 2430), random field: -18.0
hit: (distance: 0.13938161730766296, id: 377), random field: -14.0
search latency = 0.2796s

=== Start querying with `random > -14` ===

query result:
-{'pk': 9, 'random': -13.0, 'embeddings': [0.298433, 0.931987, 0.949756, 0.598713, 0.290125, 0.094323, 0.0
search latency = 0.2970s

=== Start hybrid searching with `random > -12` ===

hit: (distance: 0.0, id: 2998), random field: -11.0
hit: (distance: 0.15773043036460876, id: 472), random field: -11.0
```

```
hit: (distance: 0.3273330628871918, id: 2146), random field: -11.0
hit: (distance: 0.0, id: 2999), random field: -11.0
hit: (distance: 0.15844076871871948, id: 2218), random field: -11.0
hit: (distance: 0.1622171700000763, id: 1403), random field: -11.0
search latency = 0.3028s

=== Start deleting with expr `pk in [0, 1]` ===

query before delete by expr=`pk in [0, 1]` -> result:
-{'pk': 0, 'random': -18.0, 'embeddings': [0.142279, 0.414248, 0.378628, 0.971863, 0.535941, 0.107011, 0.2
-{'pk': 1, 'random': -15.0, 'embeddings': [0.57512, 0.358512, 0.439131, 0.862369, 0.083284, 0.294493, 0.00

query after delete by expr=`pk in [0, 1]` -> result: []


=== Drop collection `hello_milvus` ===
```

恭喜！您已经启动了 Milvus 单机版，并执行了第一次结构化查询。

# 操作指南

## 管理 Milvus 连接

管理 Milvus 连接

当前主题介绍怎么连接、断开 Milvus 服务器。

在进行其他操作前确保连接到 Milvus 服务器。

下面的例子使用 `localhost` 作为主机名，端口号 `19530` 展示连接或断开连接到 Milvus 服务器。

连接到 Milvus 服务器

构建一个 Milvus 连接。在进行其他操作前确保已连接 Milvus 服务。

Python Java GO Node.js CLI

```python
### Run `python3` in your terminal to operate in the Python interactive mode.
from pymilvus import connections
connections.connect(
    alias="default",
    host='localhost',
    port='19530'
    )
```

```javascript
import { MilvusClient } from "@zilliz/milvus2-sdk-node";
const address = "localhost:19530";
const milvusClient = new MilvusClient(address);
```

```go
    milvusClient, err := client.NewGrpcClient(
        context.Background(), // ctx
        "localhost:19530",    // addr
    )
    if err != nil {
        log.Fatal("failed to connect to Milvus:", err.Error())
    }
```

```java
final MilvusServiceClient milvusClient = new MilvusServiceClient(
        ConnectParam.newBuilder()
                .withHost("localhost")
```

```
                .withPort(19530)
                .build());
```

```
connect -h localhost -p 19530 -a default
```

参数

描述

alias

创建的 Milvus 连接的别名。

host

Milvus 服务 IP 地址。

port

Milvus 服务端口号。

参数

描述

address

Milvus 连接地址。

参数

描述

ctx

控制调用 API 的 context。

addr

Milvus 连接地址。

参数

描述

Host

Milvus IP 地址。

Port

Milvus 端口。

选项

全称

描述

-h

host

(可选) Milvus 服务 IP 地址。默认为 “127.0.0.1”。

-p

port

(可选) Milvus 服务端口。默认为 “19530”。

-a

alias

(可选) Milvus 连接别名。默认为 “default”。

-D

disconnect

(可选) 使用别名断开连接的标记。默认别名为 “default” .

help

n/a

显示命令行帮助

断开 MIlvus 连接

从 MIlvus 服务器断开。

Python Java GO Node.js CLI

```
connections.disconnect("default")
```

```
await milvusClient.closeConnection();
```

```
milvusClient.Close()
```

```
milvusClient.close()
```

```
connect -D
```

参数

描述

alias

Milvus 服务别名。

使用限制

最大连接数为 65536。

更多内容

链接 Milvus 后，还可以：

- 创建 collection
- 插入数据
- 创建索引
- 向量搜索
- 混合搜索

关于其他操作，参考

- PyMilvus API reference
- Node.js API reference

## 管理 Collection

创建 Collection

Create a Collection

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to create a collection in Milvus.

A collection consists of one or more partitions. While creating a new collection, Milvus creates a default partition `_default`. See Glossary - Collection for more information.

The following example builds a two-shard collection named `book`, with a primary key field named `book_id`, an `INT64` scalar field named `word_count`, and a two-dimensional floating point vector field named `book_intro`. Real applications will likely use much higher dimensional vectors than the example.

Prepare Schema

```
<ul>
    <li><a href="manage_connection.md">Connecting to Milvus server</a> before any operation.</li>
    <li>The collection to create must contain a primary key field and a vector field. INT64 is the only su
</ul>
```

First, prepare necessary parameters, including field schema, collection schema, and collection name.

Python Java GO Node.js CLI

```python
from pymilvus import CollectionSchema, FieldSchema, DataType
book_id = FieldSchema(
    name="book_id",
    dtype=DataType.INT64,
    is_primary=True,
    )
word_count = FieldSchema(
    name="word_count",
    dtype=DataType.INT64,
    )
book_intro = FieldSchema(
    name="book_intro",
    dtype=DataType.FLOAT_VECTOR,
    dim=2
    )
schema = CollectionSchema(
    fields=[book_id, word_count, book_intro],
    description="Test book search"
    )
collection_name = "book"
```

```javascript
const params = {
  collection_name: "book",
  description: "Test book search"
  fields: [
    {
      name: "book_intro",
      description: "",
      data_type: 101,  // DataType.FloatVector
      type_params: {
        dim: "2",
      },
    },
    {
      name: "book_id",
      data_type: 5,   //DataType.Int64
      is_primary_key: true,
      description: "",
    },
    {
      name: "word_count",
```

```
      data_type: 5,    //DataType.Int64
      description: "",
    },
  ],
};
```

```
var (
        collectionName = "book"
    )
schema := &entity.Schema{
    CollectionName: collectionName,
    Description:    "Test book search",
    Fields: []*entity.Field{
        {
            Name:       "book_id",
            DataType:   entity.FieldTypeInt64,
            PrimaryKey: true,
            AutoID:     false,
        },
        {
            Name:       "word_count",
            DataType:   entity.FieldTypeInt64,
            PrimaryKey: false,
            AutoID:     false,
        },
        {
            Name:     "book_intro",
            DataType: entity.FieldTypeFloatVector,
            TypeParams: map[string]string{
                "dim": "2",
            },
        },
    },
}
```

```
FieldType fieldType1 = FieldType.newBuilder()
        .withName("book_id")
        .withDataType(DataType.Int64)
        .withPrimaryKey(true)
        .withAutoID(false)
        .build();
FieldType fieldType2 = FieldType.newBuilder()
        .withName("word_count")
        .withDataType(DataType.Int64)
        .build();
FieldType fieldType3 = FieldType.newBuilder()
        .withName("book_intro")
        .withDataType(DataType.FloatVector)
        .withDimension(2)
        .build();
CreateCollectionParam createCollectionReq = CreateCollectionParam.newBuilder()
        .withCollectionName("book")
        .withDescription("Test book search")
        .withShardsNum(2)
        .addFieldType(fieldType1)
        .addFieldType(fieldType2)
        .addFieldType(fieldType3)
```

```
        .build();
```

```
create collection -c book -f book_id:INT64 -f word_count:INT64 -f book_intro:FLOAT_VECTOR:2 -p book_id
```

Parameter

Description

Option

FieldSchema

Schema of the fields within the collection to create. Refer to Field Schema for more information.

N/A

name

Name of the field to create.

N/A

dtype

Data type of the field to create.

For primary key field:

DataType.INT64 (numpy.int64)

```
            For scalar field:
            <ul>
                <li><code>DataType.BOOL</code> (Boolean)</li>
                <li><code>DataType.INT64</code> (numpy.int64)</li>
                <li><code>DataType.FLOAT</code> (numpy.float32)</li>
                <li><code>DataType.DOUBLE</code> (numpy.double)</li>
            </ul>
            For vector field:
            <ul>
                <li><code>BINARY_VECTOR</code> (Binary vector)</li>
                <li><code>FLOAT_VECTOR</code> (Float vector)</li>
            </ul>
        </td>
    </tr>
    <tr>
        <td><code>is_primary</code> (Mandatory for primary key field)</td>
        <td>Switch to control if the field is primary key field.</td>
        <td><code>True</code> or <code>False</code></td>
    </tr>
    <tr>
        <td><code>auto_id</code> (Mandatory for primary key field)</td>
        <td>Switch to enable or disable Automatic ID (primary key) allocation.</td>
        <td><code>True</code> or <code>False</code></td>
    </tr>
    <tr>
        <td><code>dim</code> (Mandatory for vector field)</td>
        <td>Dimension of the vector.</td>
        <td>[1, 32,768]</td>
    </tr>
    <tr>
        <td><code>description</code> (Optional)</td>
        <td>Description of the field.</td>
        <td>N/A</td>
    </tr>
```

```
    <tr>
        <td><code>CollectionSchema</code></td>
    <td>Schema of the collection to create. Refer to <a href="collection_schema.md">Collection Schema</a> f
    <td>N/A</td>
    </tr>
    <tr>
        <td><code>fields</code></td>
        <td>Fields of the collection to create.</td>
        <td>N/A</td>
    </tr>
    <tr>
        <td><code>description</code> (Optional)</td>
        <td>Description of the collection to create.</td>
        <td>N/A</td>
    </tr>
    <tr>
        <td><code>collection_name</code></td>
        <td>Name of the collection to create.</td>
        <td>N/A</td>
    </tr>
</tbody>
```

Parameter

Description

Option

collectionName

Name of the collection to create.

N/A

description

Description of the collection to create.

N/A

Fields

Schema of the fields within the collection to create. Refer to Field Schema for more information.

N/A

Name

Name of the field to create.

N/A

DataType

Data type of the field to create.

For primary key field:

entity.FieldTypeInt64 (numpy.int64)

```
            For scalar field:
            <ul>
                <li><code>entity.FieldTypeBool</code> (Boolean)</li>
                <li><code>entity.FieldTypeInt64</code> (numpy.int64)</li>
                <li><code>entity.FieldTypeFloat</code> (numpy.float32)</li>
                <li><code>entity.FieldTypeDouble</code> (numpy.double)</li>
```

```
            </ul>
            For vector field:
            <ul>
                <li><code>entity.FieldTypeBinaryVector</code> (Binary vector)</li>
                <li><code>entity.FieldTypeFloatVector</code> (Float vector)</li>
            </ul>
        </td>
    </tr>
    <tr>
        <td><code>PrimaryKey</code> (Mandatory for primary key field)</td>
        <td>Switch to control if the field is primary key field.</td>
        <td><code>True</code> or <code>False</code></td>
    </tr>
    <tr>
        <td><code>AutoID</code> (Mandatory for primary key field)</td>
        <td>Switch to enable or disable Automatic ID (primary key) allocation.</td>
        <td><code>True</code> or <code>False</code></td>
    </tr>
    <tr>
        <td><code>dim</code> (Mandatory for vector field)</td>
        <td>Dimension of the vector.</td>
        <td>[1, 32768]</td>
    </tr>
</tbody>
```

Parameter

Description

Option

collection_name

Name of the collection to create.

N/A

description

Description of the collection to create.

N/A

fields

Schema of the filed and the collection to create.

Refer to Field Schema and Collection Schema for more information.

data_type

Data type of the filed to create.

Refer to data type reference number for more information.

is_primary (Mandatory for primary key field)

Switch to control if the field is primary key field.

True or False

auto_id

Switch to enable or disable Automatic ID (primary key) allocation.

True or False

dim (Mandatory for vector field)

Dimension of the vector.

[1, 32768]

description (Optional)

Description of the field.

N/A

Parameter

Description

Option

Name

Name of the field to create.

N/A

Description

Description of the field to create.

N/A

DataType

Data type of the field to create.

For primary key field:

entity.FieldTypeInt64 (numpy.int64)

```
            For scalar field:
            <ul>
                <li><code>entity.FieldTypeBool</code> (Boolean)</li>
                <li><code>entity.FieldTypeInt64</code> (numpy.int64)</li>
                <li><code>entity.FieldTypeFloat</code> (numpy.float32)</li>
                <li><code>entity.FieldTypeDouble</code> (numpy.double)</li>
            </ul>
            For vector field:
            <ul>
                <li><code>entity.FieldTypeBinaryVector</code> (Binary vector)</li>
                <li><code>entity.FieldTypeFloatVector</code> (Float vector)</li>
            </ul>
        </td>
    </tr>
    <tr>
        <td><code>PrimaryKey</code> (Mandatory for primary key field)</td>
        <td>Switch to control if the field is primary key field.</td>
        <td><code>True</code> or <code>False</code></td>
    </tr>
    <tr>
        <td><code>AutoID</code></td>
        <td>Switch to enable or disable Automatic ID (primary key) allocation.</td>
        <td><code>True</code> or <code>False</code></td>
    </tr>
    <tr>
        <td><code>Dimension</code> (Mandatory for vector field)</td>
        <td>Dimension of the vector.</td>
        <td>[1, 32768]</td>
```

```
    </tr>
    <tr>
        <td><code>CollectionName</code></td>
        <td>Name of the collection to create.</td>
        <td>N/A</td>
    </tr>
    <tr>
        <td><code>Description</code> (Optional)</td>
        <td>Description of the collection to create.</td>
        <td>N/A</td>
    </tr>
    <tr>
        <td><code>ShardsNum</code></td>
        <td>Number of the shards for the collection to create.</td>
        <td>[1,256]</td>
    </tr>
</tbody>
```

Option

Description

-c

The name of the collection.

-f (Multiple)

The field schema in the `<fieldName>:<dataType>:<dimOfVector/desc>` format.

-p

The name of the primary key field.

-a (Optional)

Flag to generate IDs automatically.

-d (Optional)

The description of the collection.

Create a collection with the schema

Then, create a collection with the schema you specified above.

Python Java GO Node.js CLI

```
from pymilvus import Collection
collection = Collection(
    name=collection_name,
    schema=schema,
    using='default',
    shards_num=2
    )
```

```
await milvusClient.collectionManager.createCollection(params);
```

```
err = milvusClient.CreateCollection(
    context.Background(), // ctx
    schema,
    2, // shardNum
)
if err != nil {
```

```
    log.Fatal("failed to create collection:", err.Error())
}
```

```
milvusClient.createCollection(createCollectionReq);
```

```
### Follow the previous step.
```

Parameter

Description

Option

using (optional)

By specifying the server alias here, you can choose in which Milvus server you create a collection.

N/A

shards_num (optional)

Number of the shards for the collection to create.

[1,256]

Parameter

Description

Option

ctx

Context to control API invocation process.

N/A

shardNum

Number of the shards for the collection to create.

[1,256]

Limits

| Feature | Maximum limit |
|---|---|
| Length of a collection name | 255 characters |
| Number of partitions in a collection | 4,096 |
| Number of fields in a collection | 256 |
| Number of shards in a collection | 256 |

What’s next

- Learn more basic operations of Milvus:
  - Insert data into Milvus
  - Create a partition
  - Build an index for vectors
  - Conduct a vector search
  - Conduct a hybrid search
- Explore API references for Milvus SDKs:
  - PyMilvus API reference
  - Node.js API reference

查看 Collection 信息

Check Collection Information

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to check the information of the collection in Milvus.

Check if a collection exists

Verify if a collection exists in Milvus.

Python Java GO Node.js CLI

```
from pymilvus import utility
utility.has_collection("book")
```

```
await milvusClient.collectionManager.hasCollection({
  collection_name: "book",
});
```

```
hasColl, err := milvusClient.HasCollection(
    context.Background(), // ctx
    collectionName,       // CollectionName
)
if err != nil {
    log.Fatal("failed to check whether collection exists:", err.Error())
}
log.Println(hasColl)
```

```
R<Boolean> respHasCollection = milvusClient.hasCollection(
        HasCollectionParam.newBuilder()
                .withCollectionName("book")
                .build());
if (respHasCollection.getData() == Boolean.TRUE) {
    System.out.println("Collection exists.");
}
```

```
describe collection -c book
```

Parameter

Description

collection_name

Name of the collection to check.

Parameter

Description

collection_name

Name of the collection to check.

Parameter

Description

ctx

Context to control API invocation process.

CollectionName

Name of the collection to check.

Parameter

Description

CollectionName

Name of the collection to check.

Option

Description

-c

Name of the collection to check.

Check collection details

Check the details of a collection.

Python Java GO Node.js CLI

```
from pymilvus import Collection
collection = Collection("book")  ### Get an existing collection.

collection.schema                ### Return the schema.CollectionSchema of the collection.
collection.description           ### Return the description of the collection.
collection.name                  ### Return the name of the collection.
collection.is_empty              ### Return the boolean value that indicates if the collection is empty.
collection.num_entities          ### Return the number of entities in the collection.
collection.primary_field         ### Return the schema.FieldSchema of the primary key field.
collection.partitions            ### Return the list[Partition] object.
collection.indexes               ### Return the list[Index] object.
```

```
await milvusClient.collectionManager.describeCollection({          // Return the name and schema of the co
  collection_name: "book",
});

await milvusClient.collectionManager.getCollectionStatistics({     // Return the statistics information of
  collection_name: "book",
});
```

```
collDesc, err := milvusClient.DescribeCollection(               // Return the name and schema of the colle
    context.Background(),   // ctx
    "book",                 // CollectionName
)
if err != nil {
    log.Fatal("failed to check collection schema:", err.Error())
}
log.Printf("%v\n", collDesc)

collStat, err := milvusClient.GetCollectionStatistics(          // Return the statistics information of th
    context.Background(),   // ctx
    "book",                 // CollectionName
)
if err != nil {
log.Fatal("failed to check collection statistics:", err.Error())
}
```

```
R<DescribeCollectionResponse> respDescribeCollection = milvusClient.describeCollection(          // Return
        DescribeCollectionParam.newBuilder()
```

```
                .withCollectionName("book")
                .build());
DescCollResponseWrapper wrapperDescribeCollection = new DescCollResponseWrapper(respDescribeCollection.getS
System.out.println(wrapperDescribeCollection);

R<GetCollectionStatisticsResponse> respCollectionStatistics = milvusClient.getCollectionStatistics(   // R
        GetCollectionStatisticsParam.newBuilder()
                .withCollectionName("book")
                .build());
GetCollStatResponseWrapper wrapperCollectionStatistics = new GetCollStatResponseWrapper(respCollectionStati
System.out.println("Collection row count: " + wrapperCollectionStatistics.getRowCount());
```

```
describe collection -c book
```

Property

Return

Exception

schema

The schema of the collection.

description

The description of the collection.

name

The name of the collection.

is_empty

A boolean value that indicates whether the collection is empty.

num_entities

The number of entities in the collection.

CollectionNotExistException is raised if the collection does not exist.

primary_field

The primary field of the collection.

partitions

A list of all partitions.

CollectionNotExistException is raised if the collection does not exist.

indexes

A list of all indexes.

CollectionNotExistException is raised if the collection does not exist.

Parameter

Description

collection_name

Name of the collection to check.

Property

Description

status

{ error_code: number, reason: string }

schema

Information of all fields in this collection

collectionID

collectionID

Parameter

ctx

Context to control API invocation process.

CollectionName

Name of the collection to check.

Parameter

Description

CollectionName

Name of the collection to check.

Option

Description

-c

Name of the collection to check.

List all collections

List all collections in this Milvus Instance.

Python Java GO Node.js CLI

```
from pymilvus import utility
utility.list_collections()
```

```
await milvusClient.collectionManager.showCollections();
```

```
listColl, err := milvusClient.ListCollection(
    context.Background(),   // ctx
    )
if err != nil {
        log.Fatal("failed to list all collections:", err.Error())
    }
log.Println(listColl)
```

```
R<ShowCollectionsResponse> respShowCollections = milvusClient.showCollections(
        ShowCollectionsParam.newBuilder()
                .build());
System.out.println(respShowCollections);
```

```
list collections
```

Parameter

Description

ctx

Context to control API invocation process.

What’s next

- Learn more basic operations of Milvus:
  - Insert data into Milvus
  - Create a partition
  - Build an index for vectors
  - Conduct a vector search
  - Conduct a hybrid search
- Explore API references for Milvus SDKs:
  - PyMilvus API reference
  - Node.js API reference

删除 Collection

Drop a collection

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to drop a collection and the data within.

Dropping a collection irreversibly deletes all data within it.

Python Java GO Node.js CLI

```
from pymilvus import utility
utility.drop_collection("book")
```

```
await milvusClient.collectionManager.dropCollection({  collection_name: "book",});
```

```
err = milvusClient.DropCollection(
        context.Background(), // ctx
        "book",               // CollectionName
)
if err != nil {
    log.Fatal("fail to drop collection:", err.Error())
}
```

```
milvusClient.dropCollection(
        DropCollectionParam.newBuilder()
                .withCollectionName("book")
                .build());
```

```
delete collection -c book
```

Parameter

Description

collection_name

Name of the collection to drop.

Parameter

Description

collection_name

Name of the collection to drop.

Parameter

Description

ctx

Context to control API invocation process.

CollectionName

Name of the collection to drop.

Parameter

Description

CollectionName

Name of the collection to drop.

Option

Description

-c

Name of the collection to drop.

What’s next

- Learn more basic operations of Milvus:
  - Insert data into Milvus
  - Create a partition
  - Build an index for vectors
  - Conduct a vector search
  - Conduct a hybrid search
- Explore API references for Milvus SDKs:
  - PyMilvus API reference
  - Node.js API reference

管理 Collection 别名

Collection Alias

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to manage collection alias. Milvus supports specifying a unique alias for a collection.

A collection alias is globally unique, hence you cannot assign the same alias to different collections. However, you can assign multiple aliases to one collection.

The following example is based on the alias `publication`.

Create a collection alias

Specify an an alias for a collection.

Python Java GO Node.js CLI

```
from pymilvus import utility
utility.create_alias(
collection_name = "book",
alias = "publication"
)
```

```
await milvusClient.collectionManager.createAlias({
  collection_name: "book",
  alias: "publication",
});
```

```
// This function is under active development on the GO client.
milvusClient.createAlias(
    CreateAliasParam.newBuilder()
    .withCollectionName("book")
    .withAlias("publication")
    .build());
```

```
create alias -c book -a publication
```

Parameter

Description

collection_name

Name of the collection to create alias on.

alias

Collection alias to create.

Parameter

Description

collection_name

Name of the collection to create alias on.

alias

Collection alias to create.

Parameter

Description

CollectionName

Name of the collection to create alias on.

Alias

Collection alias to create.

Option

Description

-c

Name of the collection to create alias on.

-a

Collection alias to create.

-A (Optional)

Flag to transfer the alias to a specified collection.

Drop a collection alias

Drop a specified alias.

Python Java GO Node.js CLI

```
from pymilvus import utility
utility.drop_alias(
alias = "publication"
)
```

```
await milvusClient.collectionManager.dropAlias({
  alias: "publication",
});
```

```
// This function is under active development on the GO client.
```

```
milvusClient.dropAlias(
    DropAliasParam.newBuilder()
    .withAlias("publication")
    .build());
```

```
delete alias -c book -a publication
```

Parameter

Description

alias

Collection alias to drop.

Parameter

Description

alias

Collection alias to drop.

Parameter

Description

Alias

Collection alias to drop.

Option

Description

-c

Name of the collection to drop alias on.

-a

Collection alias to drop.

Alter a collection alias

Alter an existing alias to another collection. The following example is based on the situation that the alias `publication` was originally created for another collection.

Python Java GO Node.js CLI

```
from pymilvus import utility
utility.alter_alias(
collection_name = "book",
alias = "publication"
)
```

```
await milvusClient.collectionManager.alterAlias({
  collection_name: "book",
  alias: "publication",
});
```

```
// This function is under active development on the GO client.
```

```
milvusClient.alterAlias(
    AlterAliasParam.newBuilder()
    .withCollectionName("book")
    .withAlias("publication")
    .build());
```

```
create alias -c book -A -a publication
```

Parameter

Description

collection_name

Name of the collection to alter alias to.

alias

Collection alias to alter.

Parameter

Description

collection_name

Name of the collection to alter alias to.

alias

Collection alias to alter.

Parameter

Description

CollectionName

Name of the collection to alter alias to.

Alias

Collection alias to alter.

Option

Description

-c

Name of the collection to alter alias to.

-a

Collection alias to alter.

-A

Flag to transfer the alias to a specified collection.

Limits

| Feature | Maximum limit |
|---|---|
| Length of an alias | 255 characters |

What’s next

- Learn more basic operations of Milvus:
  - Insert data into Milvus
  - Create a partition
  - Build an index for vectors
  - Conduct a vector search
  - Conduct a hybrid search
- Explore API references for Milvus SDKs:
  - PyMilvus API reference
  - Node.js API reference

加载 Collection

Load a collection

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to load the collection to memory before a search or a query. All search and query operations within Milvus are executed in memory.

In current release, volume of the data to load must be under 90% of the total memory resources of all query nodes to reserve memory resources for execution engine.

Python Java GO Node.js CLI

```
from pymilvus import Collection
collection = Collection("book")      ### Get an existing collection.
collection.load()
```

```
await milvusClient.collectionManager.loadCollection({
  collection_name: "book",
});
```

```
err := milvusClient.LoadCollection(
    context.Background(),   // ctx
    "book",                 // CollectionName
    false                   // async
    )
if err != nil {
    log.Fatal("failed to load collection:", err.Error())
}
```

```
milvusClient.loadCollection(
        LoadCollectionParam.newBuilder()
                .withCollectionName("book")
                .build());
```

```
load -c book
```

Parameter

Description

partition_name (optional)

Name of the partition to load.

Parameter

Description

collection_name

Name of the collection to load.

Parameter

Description

ctx

Context to control API invocation process.

CollectionName

Name of the collection to load.

async

Switch to control sync/async behavior. The deadline of context is not applied in sync load.

Parameter

Description

CollectionName

Name of the collection to load.

Option

Description

-c

Name of the collection to load.

-p (Optional/Multiple)

The name of the partition to load.

What’s next

- Learn more basic operations of Milvus:
  - Insert data into Milvus
  - Create a partition
  - Build an index for vectors
  - Conduct a vector search
  - Conduct a hybrid search
- Explore API references for Milvus SDKs:
  - PyMilvus API reference
  - Node.js API reference

释放 Collection

Release a collection

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to release a collection from memory after a search or a query to reduce memory usage.

Python Java GO Node.js CLI

```
from pymilvus import Collection
collection = Collection("book")      ### Get an existing collection.
collection.release()
```

```
await milvusClient.collectionManager.releaseCollection({
  collection_name: "book",
});
```

```
err := milvusClient.ReleaseCollection(
    context.Background(),                           // ctx
    "book",                                         // CollectionName
    )
if err != nil {
    log.Fatal("failed to release collection:", err.Error())
}
```

```
milvusClient.releaseCollection(
        ReleaseCollectionParam.newBuilder()
                .withCollectionName("book")
                .build());
```

```
release -c book
```

Parameter

Description

partition_name (optional)

Name of the partition to release.

Parameter

Description

collection_name

Name of the collection to release.

Parameter

Description

ctx

Context to control API invocation process.

CollectionName

Name of the collection to release.

Parameter

Description

CollectionName

Name of the collection to release.

Option

Description

-c

Name of the collection to release.

-p (Optional/Multiple)

The name of the partition to release.

What’s next

- Learn more basic operations of Milvus:
  - Insert data into Milvus
  - Create a partition
  - Build an index for vectors
  - Conduct a vector search
  - Conduct a hybrid search
- Explore API references for Milvus SDKs:
  - PyMilvus API reference
  - Node.js API reference

## 管理 Partition

创建 Partition

Create a Partitions

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to create a partition in Milvus.

Milvus allows you to divide the bulk of vector data into a small number of partitions. Search and other operations can then be limited to one partition to improve the performance.

A collection consists of one or more partitions. While creating a new collection, Milvus creates a default partition `_default`. See Glossary - Partition for more information.

The following example builds a partition `novel` in the collection `book`.

Python Java GO Node.js CLI

```
from pymilvus import Collection
collection = Collection("book")      ### Get an existing collection.
collection.create_partition("novel")
```

```
await milvusClient.partitionManager.createPartition({
  collection_name: "book",
  partition_name: "novel",
});
```

```
err := milvusClient.CreatePartition(
    context.Background(),   // ctx
    "book",                 // CollectionName
    "novel"                 // partitionName
)
if err != nil {
    log.Fatal("failed to create partition:", err.Error())
}
```

```
milvusClient.createPartition(
        CreatePartitionParam.newBuilder()
                .withCollectionName("book")
                .withPartitionName("novel")
                .build());
```

```
create partition -c book -p novel
```

Parameter

Description

partition_name

Name of the partition to create.

description (optional)

Description of the partition to create.

Parameter

Description

collection_name

Name of the collection to create a partition in.

partition_name

Name of the partition to create.

Parameter

Description

ctx

Context to control API invocation process.

CollectionName

Name of the collection to create a partition in.

partitionName

Name of the partition to create.

Parameter

Description

CollectionName

Name of the collection to create a partition in.

PartitionName

Name of the partition to create.

Option

Description

-c

The name of the collection.

-p

The partition name.

-d (Optional)

The description of the partition.

Limits

| Feature | Maximum limit |
|---|---|
| Number of partitions in a collection | 4,096 |

What’s next

- Learn more basic operations of Milvus:
  - Insert data into Milvus
  - Build an index for vectors
  - Conduct a vector search
  - Conduct a hybrid search
- Explore API references for Milvus SDKs:
  - PyMilvus API reference
  - Node.js API reference

查看 Partition 信息

Check Partition Information

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to check the information of the partition in Milvus.

Verify if a partition exist

Verify if a partition exists in the specified collection.

Python Java GO Node.js CLI

```
from pymilvus import Collection
collection = Collection("book")      ### Get an existing collection.
collection.has_partition("novel")
```

```
await milvusClient.partitionManager.hasPartition({
  collection_name: "book",
  partition_name: "novel",
});
```

```
hasPar, err := milvusClient.HasPartition(
    context.Background(),   // ctx
    "book",                 // CollectionName
    "novel",                // partitionName
    )
if err != nil {
    log.Fatal("failed to check the partition:", err.Error())
}
log.Println(hasPar)
```

```
R<Boolean> respHasPartition = milvusClient.hasPartition(
        HasPartitionParam.newBuilder()
                .withCollectionName("book")
                .withPartitionName("novel")
                .build());
if (respHasCollection.getData() == Boolean.TRUE) {
    System.out.println("Partition exists.");
}
```

```
describe partition -c book -p novel
```

Parameter

Description

partition_name

Name of the partition to check.

Parameter

Description

collection_name

Name of the collection to check.

partition_name

Name of the partition to check.

Parameter

Description

ctx

Context to control API invocation process.

CollectionName

Name of the collection to check.

partitionName

Name of the partition to check.

Option

Description

-c

Name of the collection to check.

-p

Name of the partition to check.

List all partitions

Python Java GO Node.js CLI

```
from pymilvus import Collection
collection = Collection("book")      ### Get an existing collection.
collection.partitions
```

```
await milvusClient.partitionManager.showPartitions({
  collection_name: "book",
});
```

```
partitions, err := milvusClient.ShowPartitions(
    context.Background(),   // ctx
    "book",                 // CollectionName
    )
if err != nil {
    log.Fatal("failed to list partitions:", err.Error())
}
log.Println(listPar)
```

```
R<ShowPartitionsResponse> respShowPartitions = milvusClient.showPartitions(
        ShowPartitionsParam.newBuilder()
                .withCollectionName("book")
                .build());
System.out.println(respShowPartitions);
```

```
list partitions -c book
```

Parameter

Description

collection_name

Name of the collection to check.

Parameter

Description

ctx

Context to control API invocation process.

CollectionName

Name of the collection to check.

Parameter

Description

CollectionName

Name of the collection to check.

Option

Description

-c

Name of the collection to check.

What’s next

- Learn more basic operations of Milvus:
  - Insert data into Milvus
  - Build an index for vectors
  - Conduct a vector search
  - Conduct a hybrid search
- Explore API references for Milvus SDKs:
  - PyMilvus API reference
  - Node.js API reference

删除 Partition

Drop Partitions

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to drop a partition in a specified collection.

Dropping a partition irreversibly deletes all data within it.

Python Java GO Node.js CLI

```
from pymilvus import Collection
collection.drop_partition("novel")
```

```
await milvusClient.partitionManager.dropPartition({
  collection_name: "book",
  partition_name: "novel",
});
```

```
err := milvusClient.DropPartition(
    context.Background(),   // ctx
    "book",                 // CollectionName
    "novel",                // partitionName
    )
if err != nil {
    log.Fatal("fail to drop partition:", err.Error())
}
```

```
milvusClient.dropPartition(
        DropPartitionParam.newBuilder()
                .withCollectionName("book")
                .withPartitionName("novel")
                .build());
```

```
delete partition -c book -p novel
```

Parameter

Description

partition_name

Name of the partition to drop.

Parameter

Description

collection_name

Name of the collection to drop partition from.

partition_name

Name of the partition to drop.

Parameter

Description

ctx

Context to control API invocation process.

CollectionName

Name of the collection to drop a partition in.

partitionName

Name of the partition to drop.

Parameter

Description

CollectionName

Name of the collection to drop a partition in.

PartitionName

Name of the partition to drop.

Option

Description

-c

Name of the collection to drop partition from.

-p

Name of the partition to drop.

What’s next

- Learn more basic operations of Milvus:
  - Insert data into Milvus
  - Build an index for vectors
  - Conduct a vector search
  - Conduct a hybrid search
- Explore API references for Milvus SDKs:
  - PyMilvus API reference
  - Node.js API reference

## 管理数据

插入数据

Insert Data

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to insert data in Milvus via client.

You can also migrate data to Milvus with MilvusDM, an open-source tool designed specifically for importing and exporting data with Milvus.

The following example inserts 2,000 rows of randomly generated data as the example data (Milvus CLI example uses a pre-built, remote CSV file containing similar data). Real applications will likely use much higher dimensional vectors than the example. You can prepare your own data to replace the example.

Prepare data

First, prepare the data to insert. Data type of the data to insert must match the schema of the collection, otherwise Milvus will raise exception.

Python Java GO Node.js CLI

```
import random
data = [
            [i for i in range(2000)],
        [i for i in range(10000, 12000)],
            [[random.random() for _ in range(2)] for _ in range(2000)],
        ]
```

```
const data = Array.from({ length: 2000 }, (v,k) => ({
  "book_id": k,
  "word_count": k+10000,
  "book_intro": Array.from({ length: 2 }, () => Math.random()),
}));
```

```
bookIDs := make([]int64, 0, 2000)
wordCounts := make([]int64, 0, 2000)
bookIntros := make([][]float32, 0, 2000)
for i := 0; i < 2000; i++ {
    bookIDs = append(bookIDs, int64(i))
    wordCounts = append(wordCounts, int64(i+10000))
    v := make([]float32, 0, 2)
    for j := 0; j < 2; j++ {
        v = append(v, rand.Float32())
    }
    bookIntros = append(bookIntros, v)
}
idColumn := entity.NewColumnInt64("book_id", bookIDs)
wordColumn := entity.NewColumnInt64("word_count", wordCounts)
introColumn := entity.NewColumnFloatVector("book_intro", 2, bookIntros)
```

```
Random ran = new Random();
List<Long> book_id_array = new ArrayList<>();
List<Long> word_count_array = new ArrayList<>();
List<List<Float>> book_intro_array = new ArrayList<>();
for (long i = 0L; i < 2000; ++i) {
    book_id_array.add(i);
    word_count_array.add(i + 10000);
    List<Float> vector = new ArrayList<>();
    for (int k = 0; k < 2; ++k) {
        vector.add(ran.nextFloat());
    }
    book_intro_array.add(vector);
}
```

```
### Prepare your data in a CSV file. Milvus CLI only supports importing data from local or remote files.
```

Insert data to Milvus

Insert the data to the collection.

By specifying `partition_name`, you can optionally decide to which partition to insert the data.

Python Java GO Node.js CLI

```
from pymilvus import Collection
collection = Collection("book")      ### Get an existing collection.
mr = collection.insert(data)
```

```
const mr = await milvusClient.dataManager.insert({{
  collection_name: "book",
  fields_data: data,
});
```

```
_, err = milvusClient.Insert(
    context.Background(), // ctx
    "book",               // CollectionName
    "",                   // partitionName
    idColumn,             // columnarData
    wordColumn,           // columnarData
    introColumn,          // columnarData
)
if err != nil {
    log.Fatal("failed to insert data:", err.Error())
}
```

```
List<InsertParam.Field> fields = new ArrayList<>();
fields.add(new InsertParam.Field("book_id", DataType.Int64, book_id_array));
fields.add(new InsertParam.Field("word_count", DataType.Int64, word_count_array));
fields.add(new InsertParam.Field("book_intro", DataType.FloatVector, book_intro_array));

InsertParam insertParam = InsertParam.newBuilder()
        .withCollectionName("book")
        .withPartitionName("novel")
        .withFields(fields)
        .build();
milvusClient.insert(insertParam);
```

```
import -c book 'https://raw.githubusercontent.com/milvus-io/milvus_cli/main/examples/user_guide/search.csv
```

Parameter

Description

data

Data to insert into Milvus.

partition_name (optional)

Name of the partition to insert data into.

Parameter

Description

collection_name

Name of the collection to insert data into.

partition_name (optional)

Name of the partition to insert data into.

fields_data

Data to insert into Milvus.

Parameter

Description

ctx

Context to control API invocation process.

CollectionName

Name of the collection to insert data in.

partitionName

Name of the partition to insert data in. Data will be inserted in the default partition if left blank.

columnarData

Data to insert into each field.

Parameter

Description

fieldName

Name of the field to insert data in.

DataType

Data type of the field to insert data in.

data

Data to insert into each field.

```
<tr>
    <td><code>CollectionName</code></td>
    <td>Name of the collection to insert data into.</td>
</tr>
<tr>
    <td><code>PartitionName</code> (optional)</td>
    <td>Name of the partition to insert data into.</td>
</tr>
</tbody>
```

Option

Description

-c

Name of the collection to insert data into.

-p (Optional)

Name of the partition to insert data into.

Limits

| Feature | Maximum limit |
|---|---|
| Dimensions of a vector | 32,768 |

What’s next

- Learn more basic operations of Milvus:
  - Build an index for vectors
  - Conduct a vector search
  - Conduct a hybrid search
- Explore API references for Milvus SDKs:
  - PyMilvus API reference
  - Node.js API reference

删除数据

Delete Entities

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to delete entities in Milvus.

Milvus supports deleting entities by primary key filtered with boolean expression.

```
<ul>
      <li>Deleted entities can still be retrieved immediately after the deletion if the consistency level
      <li>Entities deleted beyond the pre-specified span of time for Time Travel cannot be retrieved again
    <li>Frequent deletion operations will impact the system performance.</li>
</ul>
```

Prepare boolean expression

Prepare the boolean expression that filters the entities to delete. See Boolean Expression Rules for more information.

The following example filters data with primary key values of `0` and `1`.

Python Java GO Node.js CLI

```
expr = "book_id in [0,1]"
```

```
const expr = "book_id in [0,1]";
```

```
private static final String DELETE_EXPR = "book_id in [0,1]";
```

```
delete entities -c book
The expression to specify entities to be deleted:  book_id in [0,1]
```

Option

Description

-c

The name of the collection.

-p (Optional)

The name of the partition that the entities belong to.

Delete entities

Delete the entities with the boolean expression you created. Milvus returns the ID list of the deleted entities.

Python Java GO Node.js CLI

```
from pymilvus import Collection
collection = Collection("book")      ### Get an existing collection.
collection.delete(expr)
```

```
await milvusClient.dataManager.deleteEntities({
  collection_name: "book",
  expr: expr,
});
```

```
// This function is under active development on the GO client.
```

```
milvusClient.delete(
        DeleteParam.newBuilder()
                .withCollectionName("book")
                .withExpr(DELETE_EXPR)
                .build());
```

```
You are trying to delete the entities of collection. This action cannot be undone!
Do you want to continue? [y/N]: y
```

Parameter

Description

expr

Boolean expression that specifies the entities to delete.

partition_name (optional)

Name of the partition to delete entities from.

Parameter

Description

collection_name

Name of the collection to delete entities from.

expr

Boolean expression that specifies the entities to delete.

partition_name (optional)

Name of the partition to delete entities from.

Parameter

Description

CollectionName

Name of the collection to delete entities from.

expr

Boolean expression that specifies the entities to delete.

PartitionName (optional)

Name of the partition to delete entities from.

What’s next

- Learn more basic operations of Milvus:
  - Build an index for vectors
  - Conduct a vector search
  - Conduct a hybrid search
- Explore API references for Milvus SDKs:
  - PyMilvus API reference
  - Node.js API reference

清理数据

Compact Data

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to compact data in Milvus.

Milvus supports automatic data compaction by default. You can configure your Milvus to enable or disable compaction and automatic compaction.

If automatic compaction is disabled, you can still compact data manually.

To ensure accuracy of searches with Time Travel, Milvus retains the data operation log within the span specified in common.retentionDuration. Therefore, data operated within this period will not be compacted.

Compact data manually

Compaction requests are processed asynchronously because they are usually time-consuming.

Python Java GO Node.js CLI

```
from pymilvus import Collection
collection = Collection("book")      ### Get an existing collection.
collection.compact()
```

```
const res = await milvusClient.collectionManager.compact({
  collection_name: "book",
});
const compactionID = res.compactionID;
```

```
// This function is under active development on the GO client.
```

```
R<ManualCompactionResponse> response = milvusClient.manualCompaction(
    ManualCompactionParam.newBuilder()
        .withCollectionName("book")
        .build());
long compactionID = response.getData().getCompactionID();
```

```
compact -c book
```

Parameter

Description

collection_name

Name of the collection to compact data.

Parameter

Description

CollectionName

Name of the collection to compact data.

Option

Description

-c

Name of the collection to compact data.

Check compaction status

You can check the compaction status with the compaction ID returned when the manual compaction is triggered.

Python Java GO Node.js CLI

```
collection.get_compaction_state()
```

```
const state = await milvusClient.collectionManager.getCompactionState({
    compactionID
});
```

```
// This function is under active development on the GO client.
```

```
milvusClient.getCompactionState(GetCompactionStateParam.newBuilder()
        .withCompactionID(compactionID)
        .build());
```

```
show compaction_state -c book
```

What's next

- Learn more basic operations of Milvus:
  - Insert data into Milvus
  - Create a partition
  - Build an index for vectors
  - Conduct a vector search
  - Conduct a hybrid search

- Explore API references for Milvus SDKs:
  - PyMilvus API reference
  - Node.js API reference

## 管理索引

创建索引

Build an Index

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to build an index for vectors in Milvus.

Vector indexes are an organizational unit of metadata used to accelerate vector similarity search. Without index built on vectors, Milvus will perform a brute-force search by default.

See Vector Index for more information about mechanism and varieties of vector indexes.

Current release of Milvus only supports index on vector field. Future releases will support index on scalar field.

By default, Milvus does not index a segment with less than 1,024 rows. To change this parameter, configure rootCoord.minSegmentSizeToEnableIndex in milvus.yaml.

The following example builds a 1024-cluster IVF_FLAT index with Euclidean distance (L2) as the similarity metrics. You can choose the index and metrics that suit your scenario. See Similarity Metrics for more information.

Prepare index parameter

Prepare the index parameters.

Python Java GO Node.js CLI

```
index_params = {
        "metric_type":"L2",
        "index_type":"IVF_FLAT",
        "params":{"nlist":1024}
    }
```

```
const index_params = {
  metric_type: "L2",
  index_type: "IVF_FLAT",
  params: JSON.stringify({ nlist: 1024 }),
};
```

```
idx, err := entity.NewIndexIvfFlat(   // NewIndex func
    entity.L2,                        // metricType
    1024,                             // ConstructParams
)
if err != nil {
    log.Fatal("fail to create ivf flat index parameter:", err.Error())
}
```

```
final IndexType INDEX_TYPE = IndexType.IVF_FLAT;   // IndexType
final String INDEX_PARAM = "{\"nlist\":1024}";     // ExtraParam
```

```
create index

Collection name (book): book

The name of the field to create an index for (book_intro): book_intro
```

```
Index type (FLAT, IVF_FLAT, IVF_SQ8, IVF_PQ, RNSG, HNSW, ANNOY): IVF_FLAT

Index metric type (L2, IP, HAMMING, TANIMOTO): L2

Index params nlist: 1024

Timeout []:
```

Parameter

Description

Options

metric_type

Type of metrics used to measure similarity of vectors.

For floating point vectors:

L2 (Euclidean distance)

IP (Inner product)

```
        For binary vectors:
        <ul>
            <li><code>JACCARD</code> (Jaccard distance)</li>
            <li><code>TANIMOTO</code> (Tanimoto distance)</li>
            <li><code>HAMMING</code> (Hamming distance)</li>
            <li><code>SUPERSTRUCTURE</code> (Superstructure)</li>
            <li><code>SUBSTRUCTURE</code> (Substructure)</li>
        </ul>
    </td>
</tr>
<tr>
    <td><code>index_type</code></td>
    <td>Type of index used to accelerate the vector search.</td>
    <td>For floating point vectors:
        <ul>
            <li><code>FLAT</code> (FLAT)</li>
            <li><code>IVF_FLAT</code> (IVF_FLAT)</li>
            <li><code>IVF_SQ8</code> (IVF_SQ8)</li>
            <li><code>IVF_PQ</code> (IVF_PQ)</li>
            <li><code>HNSW</code> (HNSW)</li>
            <li><code>ANNOY</code> (ANNOY)</li>
            <li><code>RHNSW_FLAT</code> (RHNSW_FLAT)</li>
            <li><code>RHNSW_PQ</code> (RHNSW_PQ)</li>
            <li><code>RHNSW_SQ</code> (RHNSW_SQ)</li>
        </ul>
        For binary vectors:
        <ul>
            <li><code>BIN_FLAT</code> (BIN_FLAT)</li>
            <li><code>BIN_IVF_FLAT</code> (BIN_IVF_FLAT)</li>
        </ul>
    </td>
</tr>
<tr>
    <td><code>params</code></td>
    <td>Building parameter(s) specific to the index.</td>
    <td>See <a href="index.md">Vector Index</a> for more information.</td>
```

```
</tr>
</tbody>
```

Parameter

Description

Option

metric_type

Type of metrics used to measure similarity of vectors.

For floating point vectors:

L2 (Euclidean distance)

IP (Inner product)

```
        For binary vectors:
        <ul>
            <li><code>JACCARD</code> (Jaccard distance)</li>
            <li><code>TANIMOTO</code> (Tanimoto distance)</li>
            <li><code>HAMMING</code> (Hamming distance)</li>
            <li><code>SUPERSTRUCTURE</code> (Superstructure)</li>
            <li><code>SUBSTRUCTURE</code> (Substructure)</li>
        </ul>
    </td>
</tr>
<tr>
    <td><code>index_type</code></td>
    <td>Type of index used to accelerate the vector search.</td>
    <td>For floating point vectors:
        <ul>
            <li><code>FLAT</code> (FLAT)</li>
            <li><code>IVF_FLAT</code> (IVF_FLAT)</li>
            <li><code>IVF_SQ8</code> (IVF_SQ8)</li>
            <li><code>IVF_PQ</code> (IVF_PQ)</li>
            <li><code>HNSW</code> (HNSW)</li>
            <li><code>ANNOY</code> (ANNOY)</li>
            <li><code>RHNSW_FLAT</code> (RHNSW_FLAT)</li>
            <li><code>RHNSW_PQ</code> (RHNSW_PQ)</li>
            <li><code>RHNSW_SQ</code> (RHNSW_SQ)</li>
        </ul>
        For binary vectors:
        <ul>
            <li><code>BIN_FLAT</code> (BIN_FLAT)</li>
            <li><code>BIN_IVF_FLAT</code> (BIN_IVF_FLAT)</li>
        </ul>
    </td>
</tr>
<tr>
    <td><code>params</code></td>
    <td>Building parameter(s) specific to the index.</td>
    <td>See <a href="index.md">Vector Index</a> for more information.</td>
</tr>
</tbody>
```

Parameter

Description

Options

NewIndex func

Function to create entity.Index according to different index types.

For floating point vectors:

NewIndexFlat (FLAT)

NewIndexIvfFlat (IVF_FLAT)

NewIndexIvfSQ8 (IVF_SQ8)

NewIndexIvfPQ (RNSG)

NewIndexRNSG (HNSW)

NewIndexHNSW (HNSW)

NewIndexANNOY (ANNOY)

NewIndexRHNSWFlat (RHNSW_FLAT)

NewIndexRHNSW_PQ (RHNSW_PQ)

NewIndexRHNSW_SQ (RHNSW_SQ)

```
        For binary vectors:
        <ul>
            <li><code>NewIndexBinFlat</code> (BIN_FLAT)</li>
            <li><code>NewIndexBinIvfFlat</code> (BIN_IVF_FLAT)</li>
        </ul>
    </td>
</tr>
<tr>
    <td><code>metricType</code></td>
    <td>Type of metrics used to measure similarity of vectors.</td>
    <td>For floating point vectors:
        <ul>
            <li><code>L2</code> (Euclidean distance)</li>
            <li><code>IP</code> (Inner product)</li>
        </ul>
        For binary vectors:
        <ul>
            <li><code>JACCARD</code> (Jaccard distance)</li>
            <li><code>TANIMOTO</code> (Tanimoto distance)</li>
            <li><code>HAMMING</code> (Hamming distance)</li>
            <li><code>SUPERSTRUCTURE</code> (Superstructure)</li>
            <li><code>SUBSTRUCTURE</code> (Substructure)</li>
        </ul>
    </td>
</tr>
<tr>
    <td><code>ConstructParams</code></td>
    <td>Building parameter(s) specific to the index.</td>
    <td>See <a href="index.md">Vector Index</a> for more information.</td>
</tr>
</tbody>
```

Parameter

Description

Options

IndexType

Type of index used to accelerate the vector search.

For floating point vectors:

FLAT (FLAT)

IVF_FLAT (IVF_FLAT)

IVF_SQ8 (IVF_SQ8)

IVF_PQ (IVF_PQ)

HNSW (HNSW)

ANNOY (ANNOY)

RHNSW_FLAT (RHNSW_FLAT)

RHNSW_PQ (RHNSW_PQ)

RHNSW_SQ (RHNSW_SQ)

```
        For binary vectors:
        <ul>
            <li><code>BIN_FLAT</code> (BIN_FLAT)</li>
            <li><code>BIN_IVF_FLAT</code> (BIN_IVF_FLAT)</li>
        </ul>
    </td>
</tr>
<tr>
    <td><code>ExtraParam</code></td>
    <td>Building parameter(s) specific to the index.</td>
    <td>See <a href="index.md">Vector Index</a> for more information.</td>
</tr>
</tbody>
```

Option

Description

help

Displays help for using the command.

Build index

Build the index by specifying the vector field name and index parameters.

Python Java GO Node.js CLI

```
from pymilvus import Collection
collection = Collection("book")      ### Get an existing collection.
collection.create_index(
    field_name="book_intro",
    index_params=index_params
    )
```

```
Status(code=0, message='')
```

```
await milvusClient.indexManager.createIndex({
  collection_name: "book",
  field_name: "book_intro",
  extra_params: index_params,
});
```

```
err = milvusClient.CreateIndex(
    context.Background(),        // ctx
    "book",                      // CollectionName
    "book_intro",                // fieldName
    idx,                         // entity.Index
    false,                       // async
)
if err != nil {
    log.Fatal("fail to create index:", err.Error())
}
```

```
milvusClient.createIndex(
        CreateIndexParam.newBuilder()
                .withCollectionName("book")
                .withFieldName("book_intro")
                .withIndexType(INDEX_TYPE)
                .withMetricType(MetricType.L2)
                .withExtraParam(INDEX_PARAM)
                .withSyncMode(Boolean.FALSE)
                .build());
```

```
### Follow the previous step.
```

Parameter

Description

field_name

Name of the vector field to build index on.

index_params

Parameters of the index to build.

Parameter

Description

collection_name

Name of the collection to build index in.

field_name

Name of the vector field to build index on.

extra_params

Parameters of the index to build.

Parameter

Description

ctx

Context to control API invocation process.

CollectionName

Name of the collection to build index on.

fieldName

Name of the vector field to build index on.

entity.Index

Parameters of the index to build.

async

Switch to control sync/async behavior. The deadline of context is not applied in sync building process.

What’ s next

- Learn more basic operations of Milvus:
  - Conduct a vector search
  - Conduct a hybrid search
  - Search with Time Travel
- Explore API references for Milvus SDKs:
  - PyMilvus API reference
  - Node.js API reference

删除索引

Drop an Index

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to drop an index in Milvus.

Dropping an index irreversibly removes all corresponding index files.

Python Java GO Node.js CLI

```
from pymilvus import Collection
collection = Collection("book")      ### Get an existing collection.
collection.drop_index()
```

```
await milvusClient.indexManager.dropIndex({
  collection_name: "book",
});
```

```
err = milvusClient.DropIndex(
    context.Background(),     // ctx
    "book",                   // CollectionName
    "book_intro",             // fieldName
)
if err != nil {
    log.Fatal("fail to drop index:", err.Error())
}
```

```
milvusClient.dropIndex(
        DropIndexParam.newBuilder()
                .withCollectionName("book")
                .withFieldName("book_intro")
                .build());
```

```
delete index -c book
```

Parameter

Description

collection_name

Name of the collection to drop index from.

Parameter

Description

ctx

Context to control API invocation process.

CollectionName

Name of the collection to drop index on.

fieldName

Name of the vector field to drop index on.

Parameter

Description

CollectionName

Name of the collection to drop index on.

FieldName

Name of the vector field to drop index on.

Option

Description

-c

Name of the collection to drop index from.

What’s next

- Learn more basic operations of Milvus:
  - Conduct a vector search
  - Conduct a hybrid search
  - Search with Time Travel
- Explore API references for Milvus SDKs:
  - PyMilvus API reference
  - Node.js API reference

## 搜索与结构化匹配

向量搜索

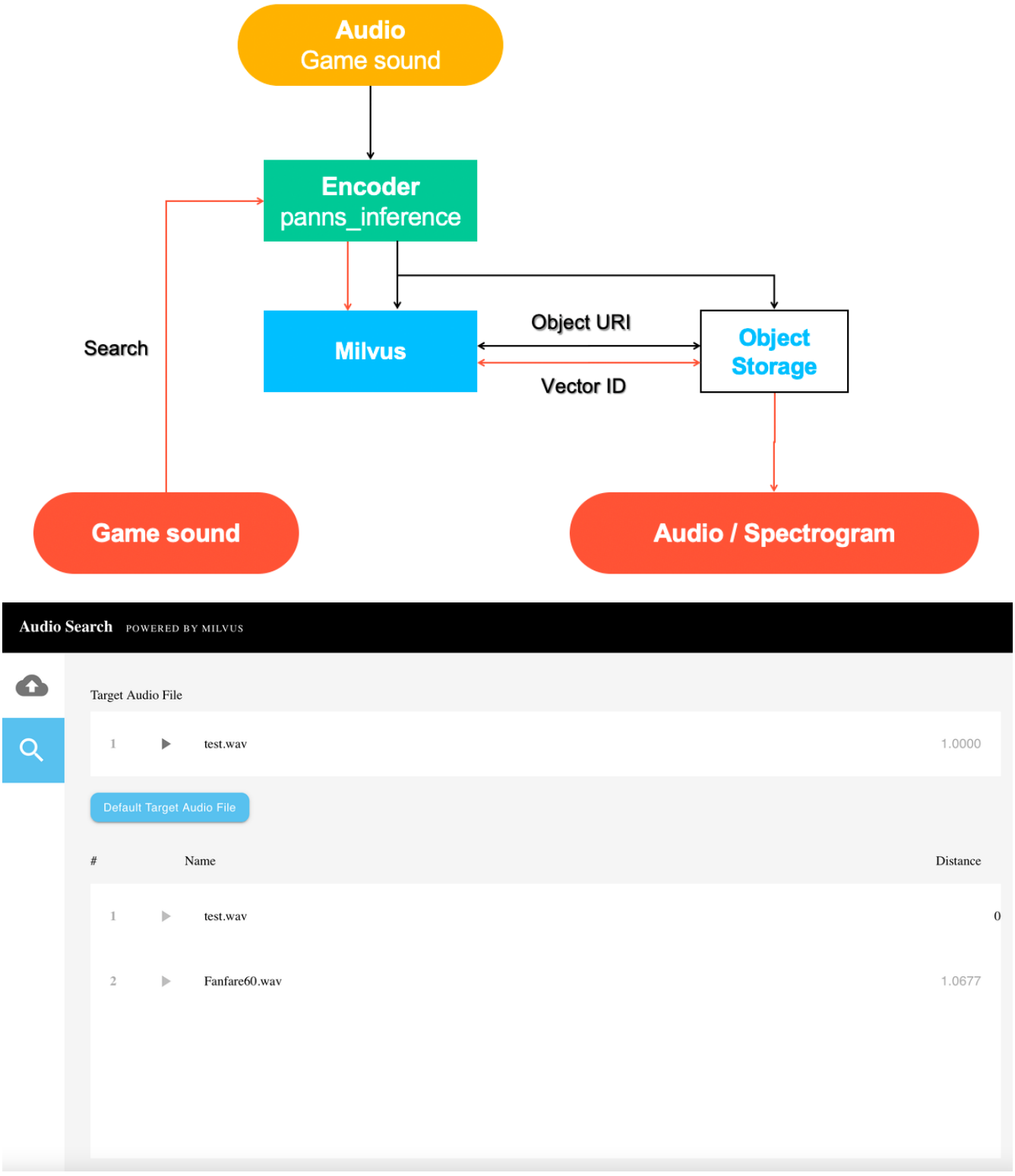
音频相似度检索

本教程将介绍如何使用开源向量数据库 Milvus 搭建音频相似度检索系统。

- 打开 Jupyter notebook
- 快速部署 本教程中使用到的 ML 模型及第三方软件包括：
- PANNs (大规模预训练音频神经网络)
- MySQL

音频检索（如演讲、音乐、音效等检索）实现了在海量音频数据中查询并找出相似声音片段。音频相似性检索系统可用于识别相似的音效、最大限度减少知识产权侵权等。音频检索还可以用于实时网络媒体的搜索和监控，来打击侵犯知识产权的行为。在音频数据的分类和统计分析中，音频检索也发挥着重要作用。

在本教程中，你将学会如何构建一个音频检索系统，用来检索相似的声音片段。使用 PANNs 将上传的音频片段转换为向量数据，并存储在 Milvus 中。Milvus 自动为每个向量生成唯一的 ID。然后用户就可以在 Milvus 中进行向量相似度搜索，Milvus 返回的检索结果为向量 ID，每个 ID 对应音频片段数据的路径。




结构化匹配

Conduct a Vector Query

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献

你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to conduct a vector query.

Unlike a vector similarity search, a vector query retrieves vectors via scalar filtering based on boolean expression. Milvus supports many data types in the scalar fields and a variety of boolean expressions. The boolean expression filters on scalar fields or the primary key field, and it retrieves all results that match the filters.

The following example shows how to perform a vector query on a 2000-row dataset of book ID (primary key), word count (scalar field), and book introduction (vector field), simulating the situation where you query for certain books based on their IDs.

### Load collection

All search and query operations within Milvus are executed in memory. Load the collection to memory before conducting a vector query.

Python Java GO Node.js CLI

```
from pymilvus import Collection
collection = Collection("book")      ### Get an existing collection.
collection.load()
```

```
await milvusClient.collectionManager.loadCollection({
  collection_name: "book",
});
```

```
err := milvusClient.LoadCollection(
    context.Background(),   // ctx
    "book",                 // CollectionName
    false                   // async
    )
if err != nil {
    log.Fatal("failed to load collection:", err.Error())
}
```

```
milvusClient.loadCollection(
        LoadCollectionParam.newBuilder()
                .withCollectionName("book")
                .build());
```

```
load -c book
```

### Conduct a vector query

The following example filters the vectors with certain `book_id` values, and returns the `book_id` field and `book_intro` of the results.

Python Java GO Node.js CLI

```
res = collection.query(expr = "book_id in [2,4,6,8]", output_fields = ["book_id", "book_intro"])
```

```
const results = await milvusClient.dataManager.query({
  collection_name: "book",
  expr: "book_id in [2,4,6,8]",
  output_fields: ["book_id", "book_intro"],
});
```

```
queryResult, err := milvusClient.Query(
    context.Background(),                                   // ctx
    "book",                                                 // CollectionName
    "",                                                     // PartitionName
    entity.NewColumnInt64("book_id", []int64{2,4,6,8}),     // expr
```

```
    []string{"book_id", "book_intro"}                        // OutputFields
)
if err != nil {
    log.Fatal("fail to query collection:", err.Error())
}
```

```
List<String> query_output_fields = Arrays.asList("book_id", "word_count");
QueryParam queryParam = QueryParam.newBuilder()
        .withCollectionName("book")
        .withExpr("book_id in [2,4,6,8]")
        .withOutFields(query_output_fields)
        .build();
R<QueryResults> respQuery = milvusClient.query(queryParam);
```

```
query

collection_name: book

The query expression: book_id in [2,4,6,8]

Name of partitions that contain entities(split by "," if multiple) []:

A list of fields to return(split by "," if multiple) []: book_id, book_intro

timeout []:
```

Parameter

Description

expr

Boolean expression used to filter attribute. Find more expression details in Boolean Expression Rules.

output_fields (optional)

List of names of the field to return.

partition_names (optional)

List of names of the partitions to query on.

Parameter

Description

collection_name

Name of the collection to query.

expr

Boolean expression used to filter attribute. Find more expression details in Boolean Expression Rules.

output_fields (optional)

List of names of the field to return.

partition_names (optional)

List of names of the partitions to query on.

Parameter

Description

Options

ctx

Context to control API invocation process.

N/A

CollectionName

Name of the collection to query.

N/A

partitionName

List of names of the partitions to load. All partitions will be queried if it is left empty.

N/A

expr

Boolean expression used to filter attribute.

See Boolean Expression Rules for more information.

OutputFields

Name of the field to return.

Vector field is not supported in current release.

Parameter

Description

Options

CollectionName

Name of the collection to load.

N/A

OutFields

Name of the field to return.

Vector field is not supported in current release.

Expr

Boolean expression used to filter attribute.

See Boolean Expression Rules for more information.

Option

Full name

Description

help

n/a

Displays help for using the command.

Check the returned results.

Python Java GO Node.js CLI

```
sorted_res = sorted(res, key=lambda k: k['book_id'])
sorted_res
```

```
console.log(results.data)
```

```
fmt.Printf("%#v\n", queryResult)
for _, qr := range queryResult {
    fmt.Println(qr.IDs)
}
```

```
QueryResultsWrapper wrapperQuery = new QueryResultsWrapper(respQuery.getData());
System.out.println(wrapperQuery.getFieldWrapper("book_id").getFieldData());
System.out.println(wrapperQuery.getFieldWrapper("word_count").getFieldData());
```

```
### Milvus CLI automatically returns the entities with the pre-defined output fields.
```

What’s next

- Learn more basic operations of Milvus:
  - Conduct a vector search
  - Conduct a hybrid search
  - Search with Time Travel
- Explore API references for Milvus SDKs:
  - PyMilvus API reference
  - Node.js API reference

混合搜索

Conduct a Hybrid Search

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to conduct a hybrid search.

A hybrid search is essentially a vector search with attribute filtering. By specifying boolean expressions that filter the scalar fields or the primary key field, you can limit your search with certain conditions.

The following example shows how to perform a hybrid search on the basis of a regular vector search. Suppose you want to search for certain books based on their vectorized introductions, but you only want those within a specific range of word count. You can then specify the boolean expression to filter the `word_count` field in the search parameters. Milvus will search for similar vectors only among those entities that match the expression.

Load collection

All search and query operations within Milvus are executed in memory. Load the collection to memory before conducting a vector search.

Python Java GO Node.js CLI

```
from pymilvus import Collection
collection = Collection("book")      ### Get an existing collection.
collection.load()
```

```
await milvusClient.collectionManager.loadCollection({
  collection_name: "book",
});
```

```
err := milvusClient.LoadCollection(
    context.Background(),   // ctx
    "book",                 // CollectionName
    false                   // async
    )
if err != nil {
    log.Fatal("failed to load collection:", err.Error())
}
```

```
milvusClient.loadCollection(
        LoadCollectionParam.newBuilder()
                .withCollectionName("book")
                .build());
```

```
load -c book
```

Conduct a hybrid vector search

By specifying the boolean expression, you can filter the scalar field of the entities during the vector search. The following example limits the scale of search to the vectors within a specified `word_count` value range.

Python Java GO Node.js CLI

```
search_param = {
    "data": [[0.1, 0.2]],
    "anns_field": "book_intro",
    "param": {"metric_type": "L2", "params": {"nprobe": 10}},
    "limit": 2,
    "expr": "word_count <= 11000",
}
res = collection.search(**search_param)
```

```
const results = await milvusClient.dataManager.search({
  collection_name: "book",
  expr: "word_count <= 11000",
  vectors: [[0.1, 0.2]],
  search_params: {
    anns_field: "book_intro",
    topk: "2",
    metric_type: "L2",
    params: JSON.stringify({ nprobe: 10 }),
  },
  vector_type: 101,    // DataType.FloatVector,
});
```

```
sp, _ := entity.NewIndexFlatSearchParam(   // NewIndex*SearchParam func
  10,                                      // searchParam
)
searchResult, err := milvusClient.Search(
  context.Background(),                    // ctx
  "book",                                  // CollectionName
  []string{},                              // partitionNames
  "word_count <= 11000",                   // expr
  []string{"book_id"},                     // outputFields
  []entity.Vector{entity.FloatVector([]float32{0.1, 0.2})}, // vectors
  "book_intro",                            // vectorField
  entity.L2,                               // metricType
  2,                                       // topK
  sp,                                      // sp
)
if err != nil {
  log.Fatal("fail to search collection:", err.Error())
}
```

```
final Integer SEARCH_K = 2;
final String SEARCH_PARAM = "{\"nprobe\":10}";
List<String> search_output_fields = Arrays.asList("book_id");
List<List<Float>> search_vectors = Arrays.asList(Arrays.asList(0.1f, 0.2f));
```

```
SearchParam searchParam = SearchParam.newBuilder()
        .withCollectionName("book")
        .withMetricType(MetricType.L2)
        .withOutFields(search_output_fields)
        .withTopK(SEARCH_K)
        .withVectors(search_vectors)
        .withVectorFieldName("book_intro")
        .withExpr("word_count <= 11000")
        .withParams(SEARCH_PARAM)
        .build();
R<SearchResults> respSearch = milvusClient.search(searchParam);
```

```
search

Collection name (book): book

The vectors of search data(the length of data is number of query (nq), the dim of every vector in data mus

The vector field used to search of collection (book_intro): book_intro

Metric type: L2

Search parameter nprobe's value: 10

The max number of returned record, also known as topk: 2

The boolean expression used to filter attribute []: word_count <= 11000

The names of partitions to search (split by "," if multiple) ['_default'] []:

timeout []:

Guarantee Timestamp(It instructs Milvus to see all operations performed before a provided timestamp. If no

Travel Timestamp(Specify a timestamp in a search to get results based on a data view) [0]:
```

Parameter

Description

data

Vectors to search with.

anns_field

Name of the field to search on.

params

Search parameter(s) specific to the index. See Vector Index for more information.

limit

Number of the most similar results to return.

expr

Boolean expression used to filter attribute. See Boolean Expression Rules for more information.

partition_names (optional)

List of names of the partition to search in.

output_fields (optional)

Name of the field to return. Vector field is not supported in current release.

timeout (optional)

A duration of time in seconds to allow for RPC. Clients wait until server responds or error occurs when it is set to None.

round_decimal (optional)

Number of decimal places of returned distance.

Parameter

Description

collection_name

Name of the collection to search in.

search_params

Parameters (as an object) used for search.

vectors

Vectors to search with.

vector_type

Pre-check of binary or float vectors. 100 for binary vectors and 101 for float vectors.

partition_names (optional)

List of names of the partition to search in.

expr (optional)

Boolean expression used to filter attribute. See Boolean Expression Rules for more information.

output_fields (optional)

Name of the field to return. Vector field not support in current release.

Parameter

Description

Options

NewIndex*SearchParam func

Function to create entity.SearchParam according to different index types.

For floating point vectors:

NewIndexFlatSearchParam (FLAT)

NewIndexIvfFlatSearchParam (IVF_FLAT)

NewIndexIvfSQ8SearchParam (IVF_SQ8)

NewIndexIvfPQSearchParam (RNSG)

NewIndexRNSGSearchParam (HNSW)

NewIndexHNSWSearchParam (HNSW)

NewIndexANNOYSearchParam (ANNOY)

NewIndexRHNSWFlatSearchParam (RHNSW_FLAT)

NewIndexRHNSW_PQSearchParam (RHNSW_PQ)

NewIndexRHNSW_SQSearchParam (RHNSW_SQ)

```
        For binary vectors:
        <ul>
            <li><code>NewIndexBinFlatSearchParam</code> (BIN_FLAT)</li>
            <li><code>NewIndexBinIvfFlatSearchParam</code> (BIN_IVF_FLAT)</li>
        </ul>
    </td>
</tr>
<tr>
    <td><code>searchParam</code></td>
    <td>Search parameter(s) specific to the index.</td>
<td>See <a href="index.md">Vector Index</a> for more information.</td>
</tr>
```

ctx

Context to control API invocation process.

N/A

CollectionName

Name of the collection to load.

N/A

partitionNames

List of names of the partitions to load. All partitions will be searched if it is left empty.

N/A

expr

Boolean expression used to filter attribute.

See Boolean Expression Rules for more information.

output_fields

Name of the field to return.

Vector field is not supported in current release.

vectors

Vectors to search with.

N/A

vectorField

Name of the field to search on.

N/A

metricType

Metric type used for search.

This parameter must be set identical to the metric type used for index building.

topK

Number of the most similar results to return.

N/A

sp

entity.SearchParam specific to the index.

N/A

</tbody>

Parameter

Description

Options

CollectionName

Name of the collection to load.

N/A

MetricType

Metric type used for search.

This parameter must be set identical to the metric type used for index building.

OutFields

Name of the field to return.

Vector field is not supported in current release.

TopK

Number of the most similar results to return.

N/A

Vectors

Vectors to search with.

N/A

VectorFieldName

Name of the field to search on.

N/A

Expr

Boolean expression used to filter attribute.

See Boolean Expression Rules for more information.

Params

Search parameter(s) specific to the index.

See Vector Index for more information.

Option

Full name

Description

help

n/a

Displays help for using the command.

Check the returned results.

Python Java GO Node.js CLI

```
assert len(res) == 1
hits = res[0]
assert len(hits) == 2
print(f"- Total hits: {len(hits)}, hits ids: {hits.ids} ")
print(f"- Top1 hit id: {hits[0].id}, distance: {hits[0].distance}, score: {hits[0].score} ")
```

```
console.log(results.results)
```

```
fmt.Printf("%#v\n", searchResult)
for _, sr := range searchResult {
  fmt.Println(sr.IDs)
  fmt.Println(sr.Scores)
}
```

```
SearchResultsWrapper wrapperSearch = new SearchResultsWrapper(respSearch.getData().getResults());
System.out.println(wrapperSearch.getIDScore(0));
System.out.println(wrapperSearch.getFieldData("book_id", 0));
```

```
### Milvus CLI automatically returns the primary key values of the most similar vectors and their distances
```

What’s next

- Learn more basic operations of Milvus:
  - Search with Time Travel
- Explore API references for Milvus SDKs:
  - PyMilvus API reference
  - Node.js API reference

使用 Time Travel 搜索

Search with Time Travel

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to use the Time Travel feature during vector search.

Milvus maintains a timeline for all data insert and delete operations. It allows users to specify a timestamp in a search to retrieve a data view at a specified point in time, without spending tremendously on maintenance for data rollback.

By default, Milvus allows Time Travel span of 432,000 seconds (120h0m0s). You can configure this parameter in common.retentionDuration.

Preparations

The following example code demonstrates the steps prior to inserting data.

If you work with your own dataset in an existing Milvus instance, you can move forward to the next step.

Python Java GO Node.js CLI

```
from pymilvus import connections, Collection, FieldSchema, CollectionSchema, DataType
connections.connect("default", host='localhost', port='19530')
collection_name = "test_time_travel"
schema = CollectionSchema([
    FieldSchema("pk", DataType.INT64, is_primary=True),
    FieldSchema("example_field", dtype=DataType.FLOAT_VECTOR, dim=2)
])
collection = Collection(collection_name, schema)
```

```
const { MilvusClient } =require("@zilliz/milvus2-sdk-node");
const milvusClient = new MilvusClient("localhost:19530");
const params = {
  collection_name: "test_time_travel",
  fields: [{
      name: "example_field",
      description: "",
      data_type: 101, // DataType.FloatVector
      type_params: {
        dim: "2",
      },
    },
    {
      name: "pk",
      data_type: 5, //DataType.Int64
      is_primary_key: true,
      description: "",
    },
  ],
};
await milvusClient.collectionManager.createCollection(params);
```

```
// This function is under active development on the GO client.
```

```
// Java User Guide will be ready soon.
```

```
connect -h localhost -p 19530 -a default
create collection -c test_time_travel -f pk:INT64:primary_field -f example_field:FLOAT_VECTOR:2 -p pk
```

Insert the first batch of data

Insert random data to simulate the original data (Milvus CLI example uses a pre-built, remote CSV file containing similar data).

Python Java GO Node.js CLI

```
import random
data = [
    [i for i in range(10)],
    [[random.random() for _ in range(2)] for _ in range(10)],
]
batch1 = collection.insert(data)
```

```
const entities1 = Array.from({ length: 10 }, (v, k) => ({
  "example_field": Array.from({   length: 2  }, () => Math.random()),
  "pk": k,
}));
const batch1 = milvusClient.dataManager.insert({
  collection_name: "test_time_travel",
  fields_data: entities1,
});
```

```
// This function is under active development on the GO client.
```

```
// Java User Guide will be ready soon.
```

```
import -c test_time_travel https://raw.githubusercontent.com/zilliztech/milvus_cli/main/examples/user_guid
Reading file from remote URL.
Reading csv rows...  [####################################]  100%
Column names are ['pk', 'example_field']
Processed 11 lines.
```

```
Inserted successfully.

-------------------------  ------------------
Total insert entities:                     10
Total collection entities:                 10
Milvus timestamp:          430390410783752199
-------------------------  ------------------
```

Check the timestamp of the first data batch

Check the timestamp of the first data batch for search with Time Travel. Data inserted within the same batch share an identical timestamp.

```
batch1.timestamp
428828271234252802
```

```
batch1.timestamp
428828271234252802
```

```
// This function is under active development on the GO client.
```

```
// Java User Guide will be ready soon.
```

```
### Milvus CLI automatically returns the timestamp as shown in the previous step.
```

Milvus adopts a combination of physical clock and logic counter as a hybrid timestamp. The 64-bit timestamp consists of a 46-bit physical part (high-order bits) and an 18-bit logic part (low-order bits). The physical part is the number of milliseconds that have elapsed since January 1, 1970 (midnight UTC/GMT).

Insert the second batch of data

Insert the second batch of data to simulate the dirty data, among which a piece of data with primary key value `19` and vector value `[1.0,1.0]` is appended as the target data to search with in the following step (Milvus CLI example uses a pre-built, remote CSV file containing similar data).

Python Java GO Node.js CLI

```python
data = [
    [i for i in range(10, 20)],
    [[random.random() for _ in range(2)] for _ in range(9)],
]
data[1].append([1.0,1.0])
batch2 = collection.insert(data)
```

```javascript
const entities2 = Array.from({
  length: 9
}, (v, k) => ({
  "example_field": Array.from({
    length: 2
  }, () => Math.random()),
  "pk": k + 10,
}));
entities2.push({
  "pk": 19,
  "example_field": [1.0, 1.0],
});
const batch2 = await milvusClient.dataManager.insert({
  collection_name: "test_time_travel",
  fields_data: entities2,
});
```

```
// This function is under active development on the GO client.

// Java User Guide will be ready soon.
```

```
import -c test_time_travel https://raw.githubusercontent.com/zilliztech/milvus_cli/main/examples/user_guid
Reading file from remote URL.
Reading csv rows...  [####################################]  100%
Column names are ['pk', 'example_field']
Processed 11 lines.

Inserted successfully.

-------------------------  ------------------
Total insert entities:                     10
Total collection entities:                 20
Milvus timestamp:          430390435713122310
-------------------------  ------------------
```

Search with a specified timestamp

Load the collection and search the target data with the timestamp of the first data batch. With the timestamp specified, Milvus only retrieves the data view at the point of time the timestamp indicates.

Python Java GO Node.js CLI

```
collection.load()
search_param = {
    "data": [[1.0, 1.0]],
    "anns_field": "example_field",
    "param": {"metric_type": "L2"},
    "limit": 10,
    "travel_timestamp": batch1.timestamp,
}
res = collection.search(**search_param)
res[0].ids
```

```
await milvusClient.collectionManager.loadCollection({
  collection_name: "test_time_travel",
});
const res = await milvusClient.dataManager.search({
  collection_name: "test_time_travel",
  vectors: [
    [1.0, 1.0]
  ],
  travel_timestamp: batch1.timestamp,
  search_params: {
    anns_field: "example_field",
    topk: "10",
    metric_type: "L2",
    params: JSON.stringify({
      nprobe: 10
    }),
  },
  vector_type: 101, // DataType.FloatVector,
});
console.log(res1.results)
```

```
// This function is under active development on the GO client.

// Java User Guide will be ready soon.
```

```
search
Collection name (test_collection_query, test_time_travel): test_time_travel
The vectors of search data (the length of data is number of query (nq), the dim of every vector in data mu
The vector field used to search of collection (example_field): example_field
The specified number of decimal places of returned distance [-1]:
The max number of returned record, also known as topk: 10
The boolean expression used to filter attribute []:
The names of partitions to search (split by "," if multiple) ['_default'] []:
Timeout []:
Guarantee Timestamp(It instructs Milvus to see all operations performed before a provided timestamp. If no
Travel Timestamp(Specify a timestamp in a search to get results based on a data view) [0]: 430390410783752
```

As shown below, the target data itself and other data inserted later are not returned as results.

```
[8, 7, 4, 2, 5, 6, 9, 3, 0, 1]
```

```
[8, 7, 4, 2, 5, 6, 9, 3, 0, 1]
```

```
// This function is under active development on the GO client.
```

```
// Java User Guide will be ready soon.
```

```
Search results:

No.1:
+---------+------+------------+-----------+
|   Index |   ID |   Distance |     Score |
+=========+======+============+===========+
|       0 |    2 |  0.0563737 | 0.0563737 |
+---------+------+------------+-----------+
|       1 |    5 |  0.122474  | 0.122474  |
+---------+------+------------+-----------+
|       2 |    3 |  0.141737  | 0.141737  |
+---------+------+------------+-----------+
|       3 |    8 |  0.331008  | 0.331008  |
+---------+------+------------+-----------+
|       4 |    0 |  0.618705  | 0.618705  |
+---------+------+------------+-----------+
|       5 |    1 |  0.676788  | 0.676788  |
+---------+------+------------+-----------+
|       6 |    9 |  0.69871   | 0.69871   |
+---------+------+------------+-----------+
|       7 |    6 |  0.706456  | 0.706456  |
+---------+------+------------+-----------+
|       8 |    4 |  0.956929  | 0.956929  |
+---------+------+------------+-----------+
|       9 |    7 |  1.19445   | 1.19445   |
+---------+------+------------+-----------+
```

If you do not specify the timestamp or specify it with the timestamp of the second data batch, Milvus will return the results from both batches.

Python Java GO Node.js CLI

```
batch2.timestamp
428828283406123011
search_param = {
    "data": [[1.0, 1.0]],
    "anns_field": "example_field",
    "param": {"metric_type": "L2"},
    "limit": 10,
```

```
    "travel_timestamp": batch2.timestamp,
}
res = collection.search(**search_param)
res[0].ids
[19, 10, 8, 7, 4, 17, 2, 5, 13, 15]
```

```
batch2.timestamp
428828283406123011
```

```
const res2 = await milvusClient.dataManager.search({
  collection_name: "test_time_travel",
  vectors: [
    [1.0, 1.0]
  ],
  travel_timestamp: batch2.timestamp,
  search_params: {
    anns_field: "example_field",
    topk: "10",
    metric_type: "L2",
    params: JSON.stringify({
      nprobe: 10
    }),
  },
  vector_type: 101, // DataType.FloatVector,
});
console.log(res2.results)
```

```
// This function is under active development on the GO client.
```

```
// Java User Guide will be ready soon.
```

```
search
Collection name (test_collection_query, test_time_travel): test_time_travel
The vectors of search data (the length of data is number of query (nq), the dim of every vector in data mu
The vector field used to search of collection (example_field): example_field
The specified number of decimal places of returned distance [-1]:
The max number of returned record, also known as topk: 10
The boolean expression used to filter attribute []:
The names of partitions to search (split by "," if multiple) ['_default'] []:
Timeout []:
Guarantee Timestamp(It instructs Milvus to see all operations performed before a provided timestamp. If no
Travel Timestamp(Specify a timestamp in a search to get results based on a data view) [0]:
Search results:

No.1:
+---------+------+------------+------------+
|   Index |   ID |   Distance |      Score |
+=========+======+============+============+
|       0 |   19 | 0          | 0          |
+---------+------+------------+------------+
|       1 |   12 | 0.00321393 | 0.00321393 |
+---------+------+------------+------------+
|       2 |    2 | 0.0563737  | 0.0563737  |
+---------+------+------------+------------+
|       3 |    5 | 0.122474   | 0.122474   |
+---------+------+------------+------------+
|       4 |    3 | 0.141737   | 0.141737   |
+---------+------+------------+------------+
|       5 |   10 | 0.238646   | 0.238646   |
```

```
+---------+------+------------+------------+
|       6 |    8 | 0.331008   | 0.331008   |
+---------+------+------------+------------+
|       7 |   18 | 0.403166   | 0.403166   |
+---------+------+------------+------------+
|       8 |   13 | 0.508617   | 0.508617   |
+---------+------+------------+------------+
|       9 |   11 | 0.531529   | 0.531529   |
+---------+------+------------+------------+
```

Generate a timestamp for search

In the case that the previous timestamp is not recorded, Milvus allows you to generate a timestamp using an existing timestamp, Unix Epoch time, or date time.

The following example simulates an unwanted deletion operation and shows how to generate a timestamp prior to the deletion and search with it.

Generate a timestamp based on the date time or Unix Epoch time prior to the deletion.

```
import datetime
datetime = datetime.datetime.now()
from pymilvus import utility
pre_del_timestamp = utility.mkts_from_datetime(datetime)
```

```
const {  datetimeToHybrids } = require("@zilliz/milvus2-sdk-node/milvus/utils/Format");
const datetime = new Date().getTime()
const pre_del_timestamp = datetimeToHybrids(datetime)
```

```
// This function is under active development on the GO client.
```

```
// Java User Guide will be ready soon.
```

```
calc mkts_from_unixtime -e 1641809375
430390476800000000
```

Delete part of the data to simulate an accidental deletion operation.

```
expr = "pk in [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]"
collection.delete(expr)
```

```
const expr = "pk in [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]"
await milvusClient.dataManager.deleteEntities({
  collection_name: "test_time_travel",
  expr: expr,
});
```

```
// This function is under active development on the GO client.
```

```
// Java User Guide will be ready soon.
```

```
delete entities -c test_time_travel
The expression to specify entities to be deleted, such as "film_id in [ 0, 1 ]": pk in [0, 2, 4, 6, 8, 10,
You are trying to delete the entities of collection. This action cannot be undone!

Do you want to continue? [y/N]: y
(insert count: 0, delete count: 10, upsert count: 0, timestamp: 430390494161534983)
```

As shown below, the deleted entities are not returned in the results if you search without specifying the timestamp.

```
search_param = {
    "data": [[1.0, 1.0]],
    "anns_field": "example_field",
    "param": {"metric_type": "L2"},
```

```
    "limit": 10,
}
res = collection.search(**search_param)
res[0].ids
```

```
const res3 = await milvusClient.dataManager.search({
  collection_name: "test_time_travel",
  vectors: [
    [1.0, 1.0]
  ],
  search_params: {
    anns_field: "example_field",
    topk: "10",
    metric_type: "L2",
    params: JSON.stringify({
      nprobe: 10
    }),
  },
  vector_type: 101, // DataType.FloatVector,
});
console.log(res3.results)
```

```
// This function is under active development on the GO client.
```

```
// Java User Guide will be ready soon.
```

```
search
Collection name (test_collection_query, test_time_travel): test_time_travel
The vectors of search data (the length of data is number of query (nq), the dim of every vector in data mu
The vector field used to search of collection (example_field): example_field
The specified number of decimal places of returned distance [-1]:
The max number of returned record, also known as topk: 10
The boolean expression used to filter attribute []:
The names of partitions to search (split by "," if multiple) ['_default'] []:
Timeout []:
Guarantee Timestamp(It instructs Milvus to see all operations performed before a provided timestamp. If no
Travel Timestamp(Specify a timestamp in a search to get results based on a data view) [0]:
Search results:

No.1:
+---------+------+------------+----------+
|   Index |   ID |   Distance |    Score |
+=========+======+============+==========+
|       0 |   19 |   0        | 0        |
+---------+------+------------+----------+
|       1 |    5 |   0.122474 | 0.122474 |
+---------+------+------------+----------+
|       2 |    3 |   0.141737 | 0.141737 |
+---------+------+------------+----------+
|       3 |   13 |   0.508617 | 0.508617 |
+---------+------+------------+----------+
|       4 |   11 |   0.531529 | 0.531529 |
+---------+------+------------+----------+
|       5 |   17 |   0.593702 | 0.593702 |
+---------+------+------------+----------+
|       6 |    1 |   0.676788 | 0.676788 |
+---------+------+------------+----------+
|       7 |    9 |   0.69871  | 0.69871  |
```

```
+---------+------+------------+----------+
|       8 |    7 |   1.19445  | 1.19445  |
+---------+------+------------+----------+
|       9 |   15 |   1.53964  | 1.53964  |
+---------+------+------------+----------+
```

Search with the prior-to-deletion timestamp. Milvus retrieves entities from the data before the deletion.

```
search_param = {
    "data": [[1.0, 1.0]],
    "anns_field": "example_field",
    "param": {"metric_type": "L2"},
    "limit": 10,
    "travel_timestamp": pre_del_timestamp,
}
res = collection.search(**search_param)
res[0].ids
```

```
const res4 = await milvusClient.dataManager.search({
  collection_name: "test_time_travel",
  vectors: [
    [1.0, 1.0]
  ],
  travel_timestamp: pre_del_timestamp,
  search_params: {
    anns_field: "example_field",
    topk: "10",
    metric_type: "L2",
    params: JSON.stringify({
      nprobe: 10
    }),
  },
  vector_type: 101, // DataType.FloatVector,
});
console.log(res4.results)
```

```
// This function is under active development on the GO client.
```

```
// Java User Guide will be ready soon.
```

```
search
Collection name (test_collection_query, test_time_travel): test_time_travel
The vectors of search data (the length of data is number of query (nq), the dim of every vector in data mu
The vector field used to search of collection (example_field): example_field
The specified number of decimal places of returned distance [-1]:
The max number of returned record, also known as topk: 10
The boolean expression used to filter attribute []:
The names of partitions to search (split by "," if multiple) ['_default'] []:
Timeout []:
Guarantee Timestamp(It instructs Milvus to see all operations performed before a provided timestamp. If no
Travel Timestamp(Specify a timestamp in a search to get results based on a data view) [0]: 430390476800000
Search results:

No.1:
+---------+------+------------+------------+
|   Index |   ID |   Distance |      Score |
+=========+======+============+============+
|       0 |   19 | 0          | 0          |
+---------+------+------------+------------+
```

```
|       1 |   12 | 0.00321393 | 0.00321393 |
+---------+------+------------+------------+
|       2 |    2 | 0.0563737  | 0.0563737  |
+---------+------+------------+------------+
|       3 |    5 | 0.122474   | 0.122474   |
+---------+------+------------+------------+
|       4 |    3 | 0.141737   | 0.141737   |
+---------+------+------------+------------+
|       5 |   10 | 0.238646   | 0.238646   |
+---------+------+------------+------------+
|       6 |    8 | 0.331008   | 0.331008   |
+---------+------+------------+------------+
|       7 |   18 | 0.403166   | 0.403166   |
+---------+------+------------+------------+
|       8 |   13 | 0.508617   | 0.508617   |
+---------+------+------------+------------+
|       9 |   11 | 0.531529   | 0.531529   |
+---------+------+------------+------------+
```

What’s next

- Learn more basic operations of Milvus:
  - Query vectors
  - Conduct a hybrid search
- Explore API references for Milvus SDKs:
  - PyMilvus API reference
  - Node.js API reference

## 负载平衡

Balance Query Load

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to balance query load in Milvus.

Milvus supports automatic load balance by default. You can configure your Milvus to enable or disable automatic load balance. By specifying `queryCoord.balanceIntervalSeconds`, `queryCoord.overloadedMemoryThresholdPercentage`, and `queryCoord.memoryUsageMaxDifferencePercentage`, you can change the thresholds that trigger the automatic load balance.

If automatic load balance is disabled, you can still balance the load manually.

Check segment information

Get the `segmentID` of the sealed segment that you expect to transfer and the `nodeID` of the query node that you expect to transfer the segment to.

Python Java GO Node.js CLI

```python
from pymilvus import utility
utility.get_query_segment_info("book")
```

```go
// This function is under active development on the GO client.
```

```java
milvusClient.getQuerySegmentInfo(
    GetQuerySegmentInfoParam.newBuilder()
        .withCollectionName("book")
        .build());
```

```
await dataManager.getQuerySegmentInfo({
    collectionName: "book",
});
```

```
show query_segment -c book
```

Parameter

Description

collection_name

Name of the collection to check the segment information.

Parameter

Description

collectionName

Name of the collection to check the segment information.

Parameter

Description

CollectionName

Name of the collection to check the segment information.

Option

Description

-c

Name of the collection to check the segment information.

Transfer segment

Transfer the sealed segment(s) with the `segmentID` and the `nodeID` of the current query node and new query node(s).

Python Java GO Node.js CLI

```
utility.load_balance(
    src_node_id=3,
    dst_node_ids=[4],
    sealed_segment_ids=[431067441441538050]
)
```

```
// This function is under active development on the GO client.
```

```
milvusClient.loadBalance(LoadBalanceParam.newBuilder()
                .withSourceNodeID(3L)
                .addDestinationNodeID(4L)
                .addSegmentID(431067441441538050L)
                .build());
```

```
await dataManager.loadBalance({
    src_nodeID: 3,
    dst_nodeIDs: [4],
    sealed_segmentIDs: [431067441441538050]
});
```

```
load_balance -s 3 -d 4 -ss 431067441441538050
```

Parameter

Description

src_node_id

ID of the query node you want to transfer segment(s) from.

dst_node_ids (Optional)

ID(s) of the query node(s) you want to transfer segment(s) to. Milvus transfers segment(s) to other query nodes automatically if this parameter is left blank.

sealed_segment_ids (Optional)

ID(s) of the segment(s) you want to transfer. Milvus transfers all sealed segment(s) in the source query node to other query nodes automatically if this parameter is left blank.

Parameter

Description

src_nodeID

ID of the query node you want to transfer segment(s) from.

dst_nodeIDs (Optional)

ID(s) of the query node(s) you want to transfer segment(s) to. Milvus transfers segment(s) to other query nodes automatically if this parameter is left blank.

sealed_segmentIDs (Optional)

ID(s) of the segment(s) you want to transfer. Milvus transfers all sealed segment(s) in the source query node to other query nodes automatically if this parameter is left blank.

Parameter

Description

SourceNodeID

ID of the query node you want to transfer segment(s) from.

DestinationNodeID (Optional)

ID(s) of the query node(s) you want to transfer segment(s) to. Milvus transfers segment(s) to other query nodes automatically if this parameter is left blank.

SegmentID (Optional)

ID(s) of the segment(s) you want to transfer. Milvus transfers all sealed segment(s) in the source query node to other query nodes automatically if this parameter is left blank.

Option

Description

-s

ID of the query node you want to transfer segment(s) from.

-d (Multiple)

ID(s) of the query node(s) you want to transfer segment(s) to.

-ss (Multiple)

ID(s) of the segment(s) you want to transfer.

What’s next

- Learn more basic operations of Milvus:
  - Insert data into Milvus
  - Create a partition
  - Build an index for vectors
  - Conduct a vector search
  - Conduct a hybrid search
- Explore API references for Milvus SDKs:
  - PyMilvus API reference
  - Node.js API reference

# 部署与运维

## 配置 Milvus

Configure Milvus

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to configure your Milvus.

In current release, all parameters take effect only after being configured at the startup of Milvus.

Docker ComposeHelm

Download a configuration file

Download `milvus.yaml` directly or with the following command.

```
$ wget https://raw.githubusercontent.com/milvus-io/milvus/v2.0.0/configs/milvus.yaml
```

Modify the configuration file

Configure your Milvus instance ro suit your application scenarios by adjusting corresponding parameters in `milvus.yaml`.

Check the following links for more information about each parameter.

Sorted by:

Components or dependencies Configuration purposes

Dependencies

Components

etcd

MinIO or S3

Pulsar

RocksMQ

Root coord

Proxy

Query coord

Query node

Index coord

Index node

Data coord

Data node

Local storage

Log

Message channel

Common

Knowhere

Purpose

Parameters

Performance tuning

queryNode.gracefulTime

rootCoord.minSegmentSizeToEnableIndex

dataCoord.segment.maxSize

dataCoord.segment.sealProportion

dataNode.flush.insertBufSize

queryCoord.autoHandoff

queryCoord.autoBalance

localStorage.enabled

Data and meta

common.retentionDuration

rocksmq.retentionTimeInMinutes

dataCoord.enableCompaction

dataCoord.enableGarbageCollection

dataCoord.gc.dropTolerance

Administration

log.level

log.file.rootPath

log.file.maxAge

minio.accessKeyID

minio.secretAccessKey

Download an installation file

Download the installation file for Milvus standalone or cluster, and save it as `docker-compose.yml`.

You can also simply run the following command.

```
### For Milvus standalone
$ wget https://github.com/milvus-io/milvus/releases/download/v2.0.0/milvus-standalone-docker-compose.yml -

### For Milvus cluster
$ wget https://github.com/milvus-io/milvus/releases/download/v2.0.0/milvus-cluster-docker-compose.yml -O d
```

Modify the installation file

In `docker-compose.yml`, add a `volumes` section under each Milvus component, i.e. root coord, data coord, data node, query coord, query node, index coord, index node, and proxy.

Map the local path to your `milvus.yaml` file onto the corresponding docker container paths to the configuration files `/milvus/configs/milvus.yaml` under all `volumes` sections.

```
...
proxy:
    container_name: milvus-proxy
    image: milvusdb/milvus:v2.0.0-rc7-20211011-d567b21
    command: ["milvus", "run", "proxy"]
    volumes:       ### Add a volumes section.
      - /local/path/to/your/milvus.yaml:/milvus/configs/milvus.yaml   ### Map the local path to the contai
    environment:
      ETCD_ENDPOINTS: etcd:2379
      MINIO_ADDRESS: minio:9000
      PULSAR_ADDRESS: pulsar://pulsar:6650
    ports:
      - "19530:19530"
...
```

Data is stored in the /volumes folder according to the default configuration in docker-compose.yml. To change the folder to store data, edit docker-compose.yml or run $ export DOCKER_VOLUME_DIRECTORY=.

Start Milvus

Having finished modifying the configuration file and installation file, you can then start Milvus.

```
$ sudo docker-compose up -d
```

What’s next

- If you want to learn how to monitor the Milvus services and create alerts:
  - Learn Monitor Milvus 2.0 with Prometheus Operator on Kubernetes
  - Learn Visualize Milvus Metrics in Grafana.

## 配置集群资源

在 Kubernetes 上配置 Milvus 集群资源

本文将介绍如何在 Kubernetes 上配置 Milvus 集群资源。

在生产环境中，通常应当依据机器工作量及机器类型相应为 Milvus 集群配置资源。你可以在集群运行时更新资源配置，但我们建议在部署集群前先设置参数。

1. 查看可用资源

运行指令 `kubectl describe nodes` 查看整个 Kubernetes 集群可为已创建实例分配的资源。

2. 配置资源

使用 Helm 为 Milvus 集群组件分配内存与 CPU 资源。

使用 Helm 升级资源配置时，正在运行的 pod 将执行滚动更新。

你可以通过以下两种方式来配置资源:

- 使用命令配置资源
- 设置 YAML 文件以配置资源

使用命令配置资源

如使用 `--set` 指令更新资源配置，必须配置每一个 Milvus 组件的资源变量。

单机版 Milvus 分布式版 Milvus

```
helm upgrade my-release milvus/milvus --reuse-values --set standalone.resources.limits.cpu=2 --set standal
```

```
helm upgrade my-release milvus/milvus --reuse-values --set dataNode.resources.limits.cpu=2 --set dataNode.
```

设置 YAML 文件以配置资源

你还可以通过设置 `resources.yaml` 文件中的参数 `resources.requests` 和 `resources.limits` 来分配 CPU 和内存资源。

```yaml
dataNode:
  resources:
    limits:
      cpu: "4"
      memory: "16Gi"
    requests:
      cpu: "1"
      memory: "4Gi"
queryNode:
  resources:
    limits:
      cpu: "4"
      memory: "16Gi"
    requests:
      cpu: "1"
      memory: "4Gi"
```

3. 应用新配置

运行如下指令以在 Milvus 集群中应用新配置。

```
helm upgrade my-release milvus/milvus --reuse-values -f resources.yaml
```

如未设置 resources.limits 参数，pod 会消耗所有可用 CPU 及内存资源。因此，为避免资源过度配置，请设置好 resources.requests 及 resources.limits 参数。

更多资源管理内容，详见 Kubernetes 文档 for more information about managing resources.

更多内容

- 你可能还想了解如何
  - 对 Milvus 集群进行扩缩容
  - 升级 Milvus 2.0
- 在云端部署 Milvus 集群:
  - 使用 Terraform 及 Ansible 在 AWS 上部署 Milvus
  - 使用 Terraform 在 Amazon EKS 上部署 Milvus
  - 使用 Kubernetes 在 GCP 上部署 Milvus
  - 使用 Kubernentes 在 Microsoft Azure 上部署 Milvus

## 云端部署

AWS 部署

Amazon EC2

在 EC2 部署 Milvus 集群

本文介绍如何使用 Terraform 和 Ansible 在 Amazon EC2上部署 Milvus 集群。

预置集群

介绍如何使用 Terraform 预置 Milvus 集群。Terraform 是一个基础架构即代码 (IaC) 软件工具。使用 Terraform，你可以通过使用声明性配置文件来预置基础设施。

先决条件

- 安装和配置 Terraform
- 安装和配置 AWS CLI

准备配置

你可以在 Google 云端硬盘 下载模板配置文件。

- `main.tf` 这个文件包含了用于预置 Milvus 集群的配置。
- `variables.tf` 这个文件允许快速编辑用于设置或更新 Milvus 集群的变量。
- `output.tf` 和 `inventory.tmpl`

  这些文件存储 Milvus 集群的元数据。本问中使用的元数据是每个节点实例的 `public_ip`，每个节点实例的 `private_ip` 和所有 EC2 实例 ID。

准备 variables.tf

本节描述 `variables.tf` 文件包含的配置。

- 节点数量

  下面的模板声明一个 `index_count` 变量，用于设置索引节点的数量。

  index_count 的值必须大于等于 1。

  ```
  variable "index_count" {
    description = "Amount of index instances to run"
    type        = number
    default     = 5
  }
  ```

- 节点类型的实例类型

  下面的模板声明了一个 `index_ec2_type` 变量，用于设置索引节点的实例类型。

  ```
  variable "index_ec2_type" {
    description = "Which server type"
    type        = string
    default     = "c5.2xlarge"
  }
  ```

- 访问权限

  下面的模板声明一个 `key_name` 变量和一个 `my_ip` 变量。`key_name` 变量表示 AWS 访问密钥。`my_ip` 变量表示安全组的 IP 地址范围。

  ```
  variable "key_name" {
    description = "Which aws key to use for access into instances, needs to be uploaded already"
    type        = string
    default     = ""
  }
  ```

```
variable "my_ip" {
  description = "my_ip for security group. used so that ansible and terraform can ssh in"
  type        = string
  default     = "x.x.x.x/32"
}
```

准备 main.tf

本节描述 `main. txt` 文件包含的配置。

- 云提供商和区域

  下面的模板使用 `us-east-2` 区域。更多信息请参见可用区域。

```
provider "aws" {
  profile = "default"
  region  = "us-east-2"
}
```

- 安全组

  下面的模板声明了一个安全组，该安全组允许来自 CIDR 地址范围的流量，该地址范围由在 `variables.tf` 中声明的 `my_ip` 表示。

```
resource "aws_security_group" "cluster_sg" {
  name        = "cluster_sg"
  description = "Allows only me to access"
  vpc_id      = aws_vpc.cluster_vpc.id

  ingress {
    description      = "All ports from my IP"
    from_port        = 0
    to_port          = 65535
    protocol         = "tcp"
    cidr_blocks      = [var.my_ip]
  }

  ingress {
    description      = "Full subnet communication"
    from_port        = 0
    to_port          = 65535
    protocol         = "all"
    self             = true
  }

  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }

  tags = {
    Name = "cluster_sg"
  }
}
```

- 虚拟私有云

  下面的模板在 Milvus 集群中指定里 CIDR 块为 10.0.0.0/24 的虚拟私有云。

```
resource "aws_vpc" "cluster_vpc" {
  cidr_block = "10.0.0.0/24"
  tags = {
    Name = "cluster_vpc"
  }
}

resource "aws_internet_gateway" "cluster_gateway" {
  vpc_id = aws_vpc.cluster_vpc.id

  tags = {
    Name = "cluster_gateway"
  }
}
```

- 子网 (可选)

  下面的模板声明了一个子网，其流量被路由到互联网网关。此时子网的 CIDR 块大小与虚拟私有云的 CIDR 块大小相同。

```
resource "aws_subnet" "cluster_subnet" {
  vpc_id                  = aws_vpc.cluster_vpc.id
  cidr_block              = "10.0.0.0/24"
  map_public_ip_on_launch = true

  tags = {
    Name = "cluster_subnet"
  }
}

resource "aws_route_table" "cluster_subnet_gateway_route" {
  vpc_id       = aws_vpc.cluster_vpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.cluster_gateway.id
  }

  tags = {
    Name = "cluster_subnet_gateway_route"
  }
}

resource "aws_route_table_association" "cluster_subnet_add_gateway" {
  subnet_id      = aws_subnet.cluster_subnet.id
  route_table_id = aws_route_table.cluster_subnet_gateway_route.id
}
```

- 节点实例 (节点)

  下面的模板声明一个 MinIO 节点实例。`main.tf` 模板文件声明了 11 个节点类型。对于某些节点类型，需要设置 `root_block_device`。有关更多信息，请参见 EBS、临时块设备和根块设备。

```
resource "aws_instance" "minio_node" {
  count         = var.minio_count
  ami           = "ami-0d8d212151031f51c"
  instance_type = var.minio_ec2_type
  key_name      = var.key_name
  subnet_id     = aws_subnet.cluster_subnet.id
```

```
    vpc_security_group_ids = [aws_security_group.cluster_sg.id]

    root_block_device {
      volume_type = "gp2"
      volume_size = 1000
    }

    tags = {
      Name = "minio-${count.index + 1}"
    }
  }
```

应用配置

1. 打开一个终端，导航到存储 `main.tf` 的文件夹。
2. 运行 `terraform init` 命令初始化配置。
3. 要应用配置，运行 `terraform apply` 并在提示时输入 `yes`。

你现在已经使用 Terraform 预置了 Milvus 集群。

启动集群

本节描述如何使用 Ansible 启动你预置完的 Milvus 集群。Ansible 是一个配置管理工具，用于自动化云配置和配置管理。

先决条件

- 安装和配置 Ansible

准备配置

你可以在 Google 云端硬盘下载模板配置文件。

- `yaml_files` 文件夹中的文件

这个文件夹存储每个节点类型的 Jinja2 文件。Ansible 使用了 Jinja2 模板。有关 Jinja2 的更多信息请参见介绍。

- `playbook.yaml`

该文件在特定的节点集上执行一组任务。该模板首先在 Milvus 集群的所有节点实例上安装 Docker 和 Docker Compose。

Playbook 是按照从上到下的顺序运行的。在每个 Play 中，任务也从上到下依次运行。

```
- name: All Servers
  hosts: etcd_ips_public:pulsar_ips_public:minio_ips_public:data_ips_public:index_ips_public:query_ips_pub
  remote_user: ec2-user
  become: true
  tags:
    - start
  tasks:
  - name: Install docker
    ansible.builtin.yum:
      name: docker
      state: present
  - name: Run docker
    ansible.builtin.service:
      name: docker
      state: started

  - name: Install or upgrade docker-compose
    get_url:
      url : "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-Linux-x86_64"
```

```
      dest: /usr/local/bin/docker-compose
      mode: 'a+x'
      force: yes
  - name: Create symbolic link for docker-compose
    file:
      src: "/usr/local/bin/docker-compose"
      dest: "/usr/bin/docker-compose"
      state: link
```

在所有节点实例上安装 Docker 和 Docker Compose 后，`playbook.yaml` 按顺序启动所有节点实例的容器。

```
- name: etcd
  hosts: etcd_ips_public
  remote_user: ec2-user
  become: true
  tags:
    - start

  tasks:
  - name: Copy etcd config
    ansible.builtin.template:
      src: ./yaml_files/etcd.j2
      dest: /home/ec2-user/docker-compose.yml
      owner: ec2-user
      group: wheel
      mode: '0644'

  - name: Run etcd node
    shell: docker-compose up -d
    args:
      chdir: /home/ec2-user/
```

应用配置

1. 打开终端，导航到存放 `playbook.yaml` 的文件夹。
2. 运行 `ansible-playbook -i inventory playbook.yaml --tags "start"`。
3. 如果成功，将启动所有节点实例。

现在你已经使用 Ansible 启动了一个 Milvus 集群。

停止节点

当不再需要 Milvus 集群时，可以停止所有节点。

确保 terraform 二进制文件在你的 PATH 上可用。

1. 运行 `terraform destroy` 并在提示时输入 `yes`。
2. 如果成功，则停止所有节点实例。

更多内容

如果你想学习如何在其他云上部署 Milvus: - 在 EKS 部署 Milvus 集群 - 在 GCP 部署 Milvus 集群 - 在 Azure 部署 Milvus 集群

Amazon EKS

在 EKS 部署 Milvus 集群

本文介绍在 Amazon EKS 上部署 Milvus 集群的操作步骤。

本文假设你对 AWS 访问管理有基本的了解。如果你不熟悉它，请参阅 AWS Identity and Access Management 文档。

先决条件

所需软件

- Terraform
- Helm
- kubectl
- AWS CLI 版本 2

云安全

- EKS，EC2，和 S3 的访问权限
- 访问密钥
- 秘密访问密钥

部署集群

你可以在 Google 云端硬盘下载模板配置文件。

1. 预置 Milvus 集群。更多信息请参见预置 Milvus 集群。
2. Milvus 集群预置完成后，使用集群的区域和名称运行以下命令。

   ```
   aws eks --region ${aws-region} update-kubeconfig --name ${cluster-name}
   ```

3. 创建 kubectl 文件，运行命令 `kubectl get svc`。如果成功，将在输出中显示一个集群。

   ```
   NAME          TYPE      CLUSTER-IP    EXTERNAL-IP                                  PORT(S)            A
   kubernetes       ClusterIP   172.20.0.1    <none>                                       443/TCP
   ```

4. 运行以下命令启动预置的 Milvus 集群。使用 S3 作为存储时，需要访问密钥和 S3 存储桶。

   ```
   helm upgrade --install --set cluster.enabled=true --set externalS3.enabled=true --set externalS3.host=
   ```

5. 再次运行 `kubectl get svc` 以获取负载平衡器的 IP 地址，将它用作 Milvus 集群的 IP 地址。

运行 kubectl get pods 查看集群中正在运行的 Pod。

伸缩集群

当前，Milvus 集群仅支持手动伸缩。运行以下命令以修改不同类型的节点实例的数量。

有关数据节点、索引节点、查询节点和代理的更多信息，请参阅存储/计算分离。

```
helm upgrade --install --set cluster.enabled=true --set dataNode.replicas=1 --set indexNode.replicas=1 --s
```

运行以上命令后，运行 `kubectl get pods` 以查看新创建的节点实例。

更多内容

如果你想学习如何在其他云上部署 Milvus: - 在 EC2 部署 Milvus 集群 - 在 GCP 部署 Milvus 集群 - 在 Azure 部署 Milvus 集群

GCP 部署

在 GCP 部署 Milvus 集群

本文介绍在谷歌云端平台 (GCP) 上部署 Milvus 集群的操作步骤。

先决条件

确定您想要使用的项目。如果您不确定要使用哪一个，联系你的 GCP 管理员创建一个新的项目。更多信息请参见创建和管理项目。本文中使用的项目名为 milvus-testing-nonprod。在命令中用你的项目名称替换它。

所需软件

- Cloud SDK
- kubectl
- Helm

你也可以使用 Cloud Shell，它预装了 GCP SDK、kubectl 和 Helm。

安装完 Cloud SDK 后，请确保您的身份验证是正确的。

设置网络

在为 Milvus 创建防火墙规则之前，需要先创建 VPC 网络。

如果已经创建了 VPC 网络，请直接阅读为 Milvus 创建防火墙规则。

创建 VPC

打开终端，执行如下命令创建 VPC 网络。

用你的项目名称替换 milvus-testing-nonprod 。

```
gcloud compute networks create milvus-network --project=milvus-testing-nonprod --subnet-mode=auto --mtu=14
```

执行如下命令创建允许 ICMP、内部、RDP 和 SSH 流量的防火墙规则。

```
gcloud compute firewall-rules create milvus-network-allow-icmp --project=milvus-testing-nonprod --network=
```

```
gcloud compute firewall-rules create milvus-network-allow-internal --project=milvus-testing-nonprod --netw
```

```
gcloud compute firewall-rules create milvus-network-allow-rdp --project=milvus-testing-nonprod --network=p
```

```
gcloud compute firewall-rules create milvus-network-allow-ssh --project=milvus-testing-nonprod --network=p
```

为 Milvus 创建防火墙规则

创建防火墙规则以允许被 Milvus 使用的 `19530` 端口上的传入流量。

```
gcloud compute --project=milvus-testing-nonprod firewall-rules create allow-milvus-in --description="Allow
```

预置 Kubernetes 集群

我们使用 Google Kubernetes Engine (GKE) 来提供一个 K8s 集群。在本文中，我们将创建一个具有两个节点的集群。节点位于 `use-west1-a` 地区，使用 `e2-standard-4` 机器类型，并使用 `cos_containerd` 节点映像。

根据需要修改以上选项。

选择机器类型

在本文中，我们使用 `e2-standard-4` 机器类型，它有四个 vCPU 和 16GB 内存。

你可以根据需要选择机器类型。但是，我们建议选择至少有 16GB 内存的机器类型，以确保稳定性。

```
gcloud beta container --project "milvus-testing-nonprod" clusters create "milvus-cluster-1" --zone "us-west
```

创建集群可能需要几分钟。创建集群后，运行以下命令获取集群的凭据。

```
gcloud container clusters get-credentials milvus-cluster-1
```

上述命令指向集群中的 `kubectl`。

部署 Milvus

集群准备完成后，可以安装 Milvus。如果您切换到另一个终端，请再次运行以下命令获取凭据。

```
gcloud container clusters get-credentials milvus-cluster-1
```

1. 运行以下命令添加 Milvus 的 Helm chart 仓库。

```
helm repo add milvus https://milvus-io.github.io/milvus-helm/
```

2. 运行以下命令更新 Milvus 的 Helm chart。

```
helm repo update
```

3. 运行以下命令安装 Milvus。

本节使用 my-release 发布版本名称。将其替换为你的发布版本名称。

```
helm install my-release milvus/milvus --set service.type=LoadBalancer
```

启动 Pod 可能需要几分钟。执行 kubectl get services 查看服务。如果成功，服务列表如下所示。

GCP

EXTERNAL-IP 列中的 34.145.26.89 为负载均衡器的 IP 地址。该 IP 地址用于连接 Milvus。

使用 Google 云端存储

Google 云端存储 (GCS) 是 Google Cloud 版本的 Amazon S3。

MinIO GCS 网关允许访问 GCS。本质上，MinIO GCS 网关通过使用 API 转换和转发所有到 GCS 的连接。你可以使用 MinIO GCS 网关代替 MinIO 服务器。

设置变量

在使用 MinIO GCS Gateway 之前设置变量。根据需要修改默认值。

Secrets

MinIO GCS Gateway 需要 GCS 服务账户凭据和 MinIO 服务账户凭据才能访问 GCS 资源。将账户凭据存储在 K8s Secret 中。Secret 必须包含以下数据。

- `accesskey`: MinIO 服务账户的访问密钥。
- `secretkey`: MinIO 服务账户的秘密访问密钥。
- `gcs_key.json`: 包含 GCS 服务帐户凭据的 JSON 文件。

下面的示例创建一个名为 `mysecret` 的 Secret，其中包含 `accesskey=minioadmin`、`secretkey=minioadmin` 和 `gcs_key.json=/home/credentials.json`。

```
$ kubectl create secret generic mysecret --from-literal=accesskey=minioadmin --from-literal=secretkey=mini
```

如果你选择的 accesskey 和 secretkey 值不是默认的 minioadmin/minioadmin，你需要更新 minio.accessKey 和 minio.secretKey 的元数据变量。

元数据

下表列出了可以配置的元数据。| 选项 | 描述 | 默认值 | |:—|:—|:—| |`minio.gcsgateway.enabled`| 设置值为 `true` 启用 MinIO GCS 网关。|`false`| |`minio.gcsgateway.projectId`|GCP 项目的 ID。|`""`| |`minio.existingSecret`| 先前定义的 Secret 的名称。|`""`| |`externalGcs.bucketName`| 要使用的 GCS 存储桶的名称。与 S3/MinIO bucket 不同，GCS 存储桶必须是全局唯一的。|`""`|

下表列出了您可能希望保留为默认值的元数据。

| 选项 | 描述 | 默认值 |
| --- | --- | --- |
| `minio.gcsgateway.replicas` | 用于网关的复制节点的数量。建议使用一个，因为 MinIO 不能很好地支持多个副本。 | `1` |
| `minio.gcsgateway.gcsKeyJson` | GCS 服务帐户凭据的文件路径。不要修改默认值。 | `/etc/credentials/gcs_key.json` |

继续使用所有预定义的 MinIO 元数据变量。

下面的例子安装了一个名为 `my-release` 的 chart。

```
$ helm install my-release milvus/milvus --set minio.existingSecret=mysecret --set minio.gcsgateway.enabled=
```

更多内容

如果你想学习如何在其他云上部署 Milvus 集群: - 在 EC2 上部署 Milvus 集群 - 在 EKS 上部署 Milvus 集群 - 在 Azure 上部署 Milvus 集群

Azure 部署

在 Azure 部署 Milvus 集群

本文介绍使用 Azure Kubernetes 服务 (AKS) 和 Azure 门户预置和创建集群。

先决条件

确保你的 Azure 项目已经正确设置，并且你可以访问你想要使用的资源。如果你不确定你的访问权限，请与你的管理员联系。

所需软件

- Azure CLI
- kubectl
- Helm

或者，你可以使用 Cloud Shell，它预装了 Azure CLI 、kubectl 和 Helm。

在安装 Azure CLI 之后，请确保你经过了正确的身份验证。

预置 AKS 群集

1. 登录 Azure 门户。
2. 在 Azure 门户菜单上或在“主页”中，选择“创建资源”。
3. 选择“容器”> “Kubernetes 服务”。
4. 在“基本信息”页面上，配置以下选项:

- 项目详细信息:
  - 订阅：联系你组织的 Azure 管理员，以确定你应该使用哪个订阅。
  - 资源组：联系你组织的 Azure 管理员，以确定应该使用哪个资源组。
- 群集详细信息:
  - Kubernetes 集群名称：输入集群名称。
  - 区域：选择区域。
  - 可用性区域：根据需要选择可用性区域。对于生产集群，建议选择多个可用性区域。
- 主节点池:
  - 节点大小：我们建议你选择内存至少为 16GB 的虚拟机，但是你可以根据需要选择虚拟机大小。
  - 缩放方法：选择缩放方法。
  - 节点数范围：选择节点数范围。
- 节点池:
  - 启用虚拟节点：选中复选框启用虚拟节点。

  - 启动虚拟机规模集：我们建议你选择“启用”。
- 网络：
  - 网络配置：我们建议你选择 `Kubenet`。
  - DNS 前缀：输入 DNS 名称前缀。
  - 流量路由：
    * 负载平衡器： `Standard`
    * HTTP 应用程序路由：不需要。

5. 验证完成后，依次单击“查看 + 创建”、“创建”。创建 AKS 群集需要几分钟时间。

使用 Helm 部署 Milvus

集群创建完成后，使用 Helm 将 Milvus 安装到集群中。

连接集群

1. 导航到你在 Kubernetes 服务中创建的集群，并单击它。
2. 在左侧导航窗格中，单击“概述”。
3. 在弹出的“概述”页面中，单击“连接”，查看资源组和订阅。Azure

设置订阅和凭据

你可以使用 Azure Cloud Shell 执行以下步骤。

1. 运行以下命令设置订阅。

```
az account set --subscription EXAMPLE-SUBSCRIPTION-ID
```

2. 运行以下命令下载凭据并配置 Kubernetes CLI 以使用它们。

```
az aks get-credentials --resource-group YOUR-RESOURCE-GROUP --name YOUR-CLUSTER-NAME
```

对下列过程使用相同的 shell。如果切换到其他 shell，请重新执行上述命令。

部署 Milvus

1. 运行以下命令添加 Milvus 的 Helm chart 仓库。

```
helm repo add milvus https://milvus-io.github.io/milvus-helm/
```

2. 执行以下命令更新你的 Milvus 的 Helm chart。

```
helm repo update
```

3. 运行如下命令安装 Milvus。

本文档以 my-release 作为发布版本名称。将其替换为你的发布版本名称。

```
helm install my-release milvus/milvus --set service.type=LoadBalancer
```

启动 Pod 可能需要几分钟。执行 `kubectl get services` 命令查看服务。如果成功，服务列表如下所示。

Results

EXTERNAL-IP 下的 20.81.111.155 的为负载均衡器的 IP 地址。默认 Milvus 端口为 19530。

使用 Azure Blob 存储

Azure Blob 存储是 Azure 版本的 Amazon S3。

MinIO Azure 网关允许访问 Azure。本质上，MinIO Azure 网关通过使用 API 转换和转发所有到 Azure 的连接。你可以使用 MinIO Azure 网关代替 MinIO 服务器。

设置变量

在使用 MinIO Azure Gateway 之前设置变量。根据需要修改默认值。

元数据

下表列出了可以配置的元数据。

| 选项 | 描述 | 默认值 |
| --- | --- | --- |
| `minio.azuregateway.enabled` | 设置值为 `true` 启用 MinIO Azure 网关。 | `false` |
| `minio.accessKey` | MinIO 访问密钥。 | `""` |
| `minio.secretKey` | MinIO 秘密访问密钥。 | `""` |
| `externalAzure.bucketName` | 要使用的 Azure 存储桶的名称。与 S3/MinIO 存储桶不同，Azure 存储桶必须是全局唯一的。 | `""` |

下表列出了你可能希望保留为默认值的元数据。

| 选项 | 描述 | 默认 |
| --- | --- | --- |
| `minio.azuregateway.replicas` | 用于网关的复制节点的数量。我们建议你使用一个，因为 MinIO 不能很好地支持多个副本。 | `1` |

继续使用所有预定义的 MinIO 元数据变量。

下面的例子安装了一个名为 `my-release` 的图表。

```
helm install my-release ./milvus --set service.type=LoadBalancer --set minio.persistence.enabled=false --s
```

更多内容

如果你想学习如何在其他云上部署 Milvus: - 在 EC2 上部署 Milvus 集群 - 在 EKS 上部署 Milvus 集群 - 在 GCP 上部署 Milvus 集群

## 配置 S3 存储

为 Milvus 配置 S3 存储

Milvus 支持 Amazon Simple Storage Service（Amazon S3）作为存储引擎，实现日志数据和索引文件的数据持久化。本篇文档将展示如何为 Milvus 配置 S3 存储。

使用 Docker Compose 配置 S3

如需通过 Docker Compose 为 Milvus 配置 S3，你需要更改 milvus/configs 目录下的 milvus.yaml 文件中的 MinIO/S3 配置。

鉴于 MinIO 兼容 S3，你可以直接在 `minio` 部分下配置 S3 参数。

```yaml
minio:
  address: <your_s3_endpoint>
  port: <your_s3_port>
  accessKeyID: <your_s3_access_key_id>
  secretAccessKey: <your_s3_secret_access_key>
  useSSL: <true/false>
  bucketName: "<your_bucket_name>"
```

更多细节参考 MinIO 或 S3 配置。

所有参数设置在 Milvus 启动时生效。

使用 Kubernetes 配置 S3

对于 Kubernetes 上的 Milvus 集群，你可以在启动 Milvus 的命令行中配置参数，也可以在启动前通过 milvus-helm 库中 /charts/milvus 目录下的 values.yml 文件配置参数。

以下是 Helm Charts 安装的 S3 配置项：

| 参数 | 说明 | 注释 |
| --- | --- | --- |
| externalS3.enabled | 启用或禁用外部 S3 | true/false |
| externalS3.host | 外部 S3 节点 | |
| externalS3.port | 外部 S3 端口 | |
| externalS3.accessKey | 外部 S3 给用户授权访问的密钥 ID | |
| externalS3.secretKey | 外部 S3 加密字符串 | |
| externalS3.bucketName | 外部 S3 存储桶名 | |
| minio.enabled | 启用或禁用 MinIO | true/false |

使用命令行配置 S3

使用以下命令启动 Milvus 并配置 S3：

```
helm install <your_release_name> milvus/milvus --set cluster.enabled=true --set externalS3.enabled=true --
```

使用 values.yaml 文件配置 S3

在 values.yaml 文件中配置 `minio` 部分：

```
minio:
  enabled: false
```

在 values.yaml 文件中配置 `externalS3` 部分：

```
externalS3:
  enabled: true
  host: "<your_s3_endpoint>"
  port: "<your_s3_port>"
  accessKey: "<your_s3_access_key_id>"
  secretKey: "<your_s3_secret_key>"
  useSSL: <true/false>
  bucketName: "<your_bucket_name>"
```

上述部分配置完成后，运行以下命令：

```
helm install <your_release_name> milvus/milvus -f values.yaml
```

## 配置依赖

Configure Dependencies with Milvus Operator

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

Milvus cluster depends on components including object storage, etcd, and Pulsar. This topic introduces how to configure these dependencies when you install Milvus with Milvus Operator.

This topic assumes that you have deployed Milvus Operator.

See Deploy Milvus Operator for more information.

You need to specify a configuration file for using Milvus Operator to start a Milvus cluster.

```
kubectl apply -f https://raw.githubusercontent.com/milvus-io/milvus-operator/main/config/samples/milvusclu
```

You only need to edit the code template in `milvuscluster_default.yaml` to configure third-party dependencies. The following sections introduce how to configure object storage, etcd, and Pulsar respectively.

### Configure object storage

A Milvus cluster uses MinIO or S3 as object storage to persist large-scale files, such as index files and binary logs. Add required fields under `spec.dependencies.storage` to configure object storage.

`storage` supports `external` and `inCluster`.

#### External object storage

`external` indicates using an external object storage service.

Fields used to configure an external object storage service include:

- `external`: A `true` value indicates that Milvus uses an external storage service.
- `type`: Specifies whether Milvus uses S3 or MinIO as object storage.
- `secretRef`: The secret reference that the object storage service uses.
- `endpoint`: The endpoint of the object storage service.

#### Example

The following example configures an external object storage service.

```yaml
kind: MilvusCluster

metadata:

  name: my-release

  labels:

    app: milvus

spec:

  dependencies: ### Optional

    storage: ### Optional

      ### Whether (=true) to use an existed external storage as specified in the field endpoints or

      ### (=false) create a new storage inside the same kubernetes cluster for milvus.

      external: true ### Optional default=false

      type: "MinIO" ### Optional ("MinIO", "S3") default:="MinIO"

      ### Secret reference of the storage if it has

      secretRef: mySecret ### Optional

      ### The external storage endpoint if external=true

      endpoint: "storageEndpoint"

  components: {}
```

```
  config: {}
```

Internal object storage

`inCluster` indicates when a Milvus cluster starts, a MinIO service starts automatically in the cluster.

A Milvus cluster only supports using MinIO as the internal object storage service.

Example

The following example configures an internal MinIO service.

```
apiVersion: milvus.io/v1alpha1

kind: MilvusCluster

metadata:

  name: my-release

  labels:

    app: milvus

spec:

  dependencies:

    storage: #

      external: false

      type: "MinIO" ### Optional ("MinIO", "S3") default:="MinIO"

      inCluster:

        ### deletionPolicy of storage when the milvus cluster is deleted

        deletionPolicy: Retain ### Optional ("Delete", "Retain") default="Retain"

        ### When deletionPolicy="Delete" whether the PersistantVolumeClaim shoud be deleted when the stora

        pvcDeletion: false

        values:

          resources:

             limits:

              cpu: '2'

              memory: 6Gi

            requests:

              cpu: 100m
```

```
            memory: 512Mi
        statefulset:
          replicaCount: 6
  components: {}
  config: {}
```

In this example, inCluster.deletionPolicy defines a deleletion policy for data. inCluster.values.resources defines the compute resources that MinIO uses. inCluster.values.statefulset.replicaCount defines the number of replicas of MinIO on each drive.

Find the complete configuration items to configure an internal MinIO service in values.yaml. Add configuration items as needed under storage.inCluster.values as shown in the preceding example.

Assuming that the configuration file is named `milvuscluster.yaml`, run the following command to apply the configuration.

```
kubectl apply -f milvuscluster.yaml
```

If my-release is an existing Milvus cluster, milvuscluster.yaml overwrites its configuration. Otherwise, a new Milvus cluster is created.

Configure etcd

etcd stores metadata of components in a Milvus cluster. Add required fields under `spec.dependencies.etcd` to configure etcd.

`etcd` supports `external` and `inCluster`.

Fields used to configure an external etcd service include:

- `external`: A `true` value indicates that Milvus uses an external etcd service.
- `endpoints`: The endpoints of etcd.

External etcd

Example

The following example configures an external etcd service.

```
kind: MilvusCluster
metadata:
  name: my-release
  labels:
    app: milvus

spec:
  dependencies: ### Optional
    etcd: ### Optional
      ### Whether (=true) to use an existed external etcd as specified in the field endpoints or
```

```
      ### (=false) create a new etcd inside the same kubernetes cluster for milvus.
      external: true ### Optional default=false
      ### The external etcd endpoints if external=true
      endpoints:
      - 192.168.1.1:2379
  components: {}
  config: {}
```

Internal etcd

`inCluster` indicates when a Milvus cluster starts, an etcd service starts automatically in the cluster.

Example

The following example configures an internal etcd service.

```
apiVersion: milvus.io/v1alpha1
kind: MilvusCluster
metadata:
  name: my-release
  labels:
    app: milvus
spec:
  dependencies:
    etcd:
      inCluster:
        values:
          replicaCount: 5
          resources:
            limits:
              cpu: '4'
              memory: 8Gi
            requests:
```

```
            cpu: 200m
            memory: 512Mi
  components: {}
  config: {}
```

The preceding example specifies the number of replicas as 5 and limits the compute resources for etcd.

Find the complete configuration items to configure an internal etcd service in values.yaml. Add configuration items as needed under etcd.inCluster.values as shown in the preceding example.

Assuming that the configuration file is named `milvuscluster.yaml`, run the following command to apply the configuration.

```
kubectl apply -f milvuscluster.yaml
```

Configure Pulsar

Pulsar manages logs of recent changes, outputs stream logs, and provides log subscriptions. Add required fields under `spec.dependencies.pulsar` to configure Pulsar. `pulsar` supports `external` and `inCluster`.

External Pulsar

`external` indicates using an external Pulsar service. Fields used to configure an external Pulsar service include:

- `external`: A `true` value indicates that Milvus uses an external Pulsar service.
- `endpoints`: The endpoints of Pulsar.

Example

The following example configures an external Pulsar service.

```
apiVersion: milvus.io/v1alpha1
kind: MilvusCluster
metadata:
  name: my-release
  labels:
    app: milvus
spec:
  dependencies: ### Optional
    pulsar: ### Optional
      ### Whether (=true) to use an existed external pulsar as specified in the field endpoints or
      ### (=false) create a new pulsar inside the same kubernetes cluster for milvus.
      external: true ### Optional default=false
      ### The external pulsar endpoints if external=true
```

```
      endpoints:
      - 192.168.1.1:6650
  components: {}
  config: {}
```

Internal Pulsar

`inCluster` indicates when a Milvus cluster starts, a Pulsar service starts automatically in the cluster.

Example

The following example configures an internal Pulsar service.

```
apiVersion: milvus.io/v1alpha1
kind: MilvusCluster
metadata:
  name: my-release
  labels:
    app: milvus
spec:
  dependencies:
    pulsar:
      inCluster:
        values:
          components:
            autorecovery: false
          zookeeper:
            replicaCount: 1
          bookkeeper:
            replicaCount: 1
            resoureces:
              limit:
                cpu: '4'
              memory: 8Gi
```

```
      requests:
        cpu: 200m
        memory: 512Mi
    broker:
      replicaCount: 1
      configData:
        ### Enable `autoSkipNonRecoverableData` since bookkeeper is running
        ### without persistence
        autoSkipNonRecoverableData: "true"
        managedLedgerDefaultEnsembleSize: "1"
        managedLedgerDefaultWriteQuorum: "1"
        managedLedgerDefaultAckQuorum: "1"
    proxy:
      replicaCount: 1
components: {}
config: {}
```

This example specifies the numbers of replicas of each component of Pulsar, the compute resources of Pulsar BookKeeper, and other configurations.

Find the complete configuration items to configure an internal Pulsar service in values.yaml. Add configuration items as needed under pulsar.inCluster.values as shown in the preceding example.

Assuming that the configuration file is named `milvuscluster.yaml`, run the following command to apply the configuration.

```
kubectl apply -f milvuscluster.yaml
```

What’s next

If you want to learn how to configure dependencies with `milvus.yaml`, see System Configuration.

## 扩缩容

Scale a Milvus Cluster

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

Milvus supports horizontal scaling of its components. This means you can either increase or decrease the number of worker nodes of each type according to your own need.

This topic describes how to scale out and scale in a Milvus cluster. We assume that you have already installed a Milvus cluster before scaling. Also, we recommend familiarizing yourself with the Milvus architecture before you begin.

This tutorial takes scaling out three query nodes as an example. To scale out other types of nodes, replace `queryNode` with the corresponding node type in the command line.

What is horizontal scaling?

Horizontal scaling includes scaling out and scaling in.

Scaling out

Scaling out refers to increasing the number of nodes in a cluster. Unlike scaling up, scaling out does not require you to allocate more resources to one node in the cluster. Instead, scaling out expands the cluster horizontally by adding more nodes.

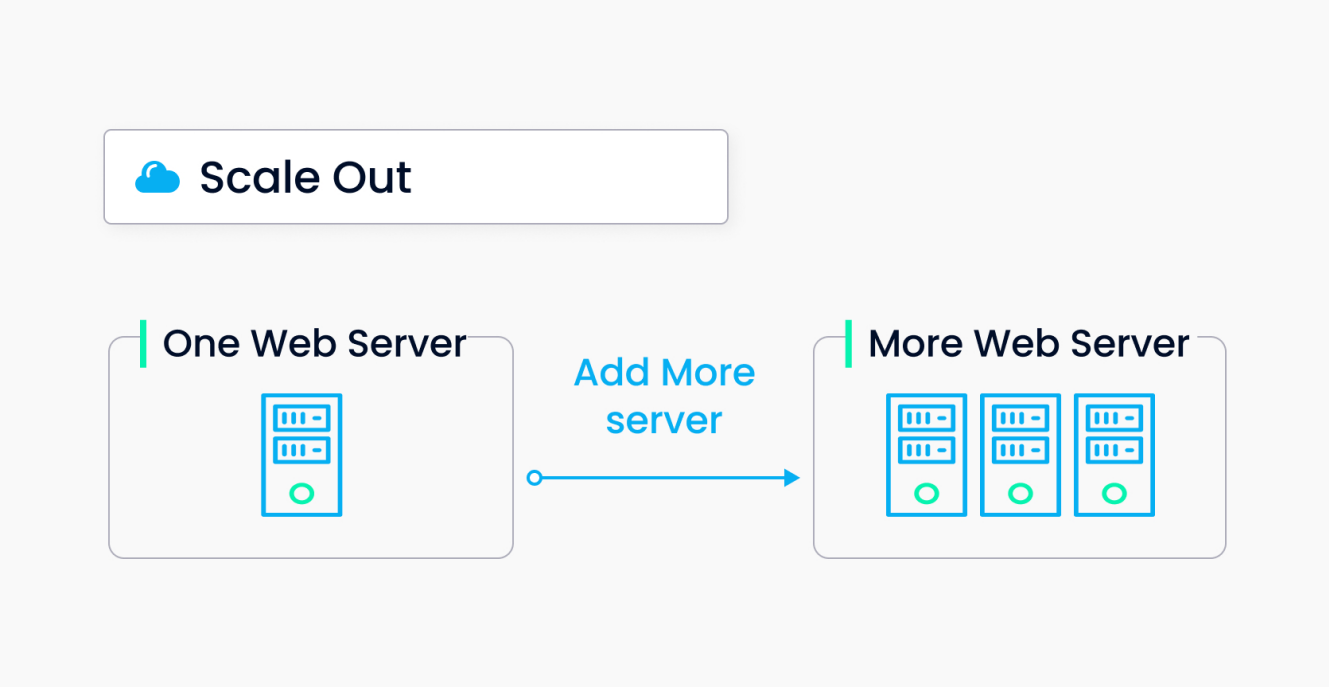



Figure 4: Scaleout

According to the Milvus architecture, stateless worker nodes include query node, data node, index node, and proxy. Therefore, you can scale out these type of nodes to suit your business needs and application scenarios. You can either scale out the Milvus cluster manually or automatically.

Generally, you will need to scale out the Milvus cluster you created if it is over-utilized. Below are some typical situations where you may need to scale out the Milvus cluster: - The CPU and memory utilization is high for a period of time. - The query throughput becomes higher. - Higher speed for indexing is required. - Massive volumes of large datasets need to be processed. - High availability of the Milvus service needs to be ensured.

Scaling in

Scaling in refers to decreasing the number of nodes in a cluster. Generally, you will need to scale in the Milvus cluster you created if it is under-utilized. Below are some typical situations where you need to scale in the Milvus cluster: - The CPU and memory utilization is low for a period of time. - The query throughput becomes lower. - Higher speed for indexing is not required. - The size of the dataset to be processed is small.

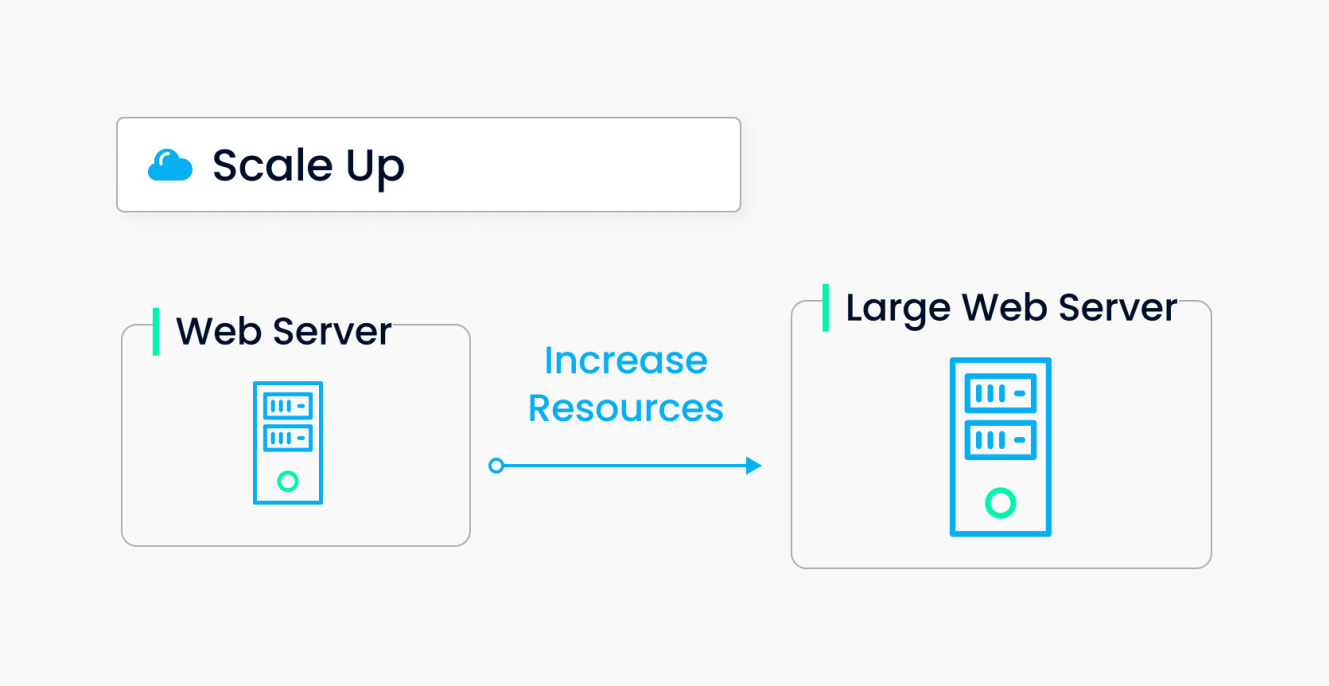



Figure 5: Scaleup

We do not recommend reducing the number of workers nodes dramatically. For example, if there are five data nodes in the cluster, we recommend reducing one data node at a time to ensure service availability. If the service is available after the first attempt of scaling in, you can continue to further reduce the number of the data node.

Prerequisites

Run `kubectl get pods` to get a list of the components and their working status in the Milvus cluster you created.

```
NAME                                          READY   STATUS      RESTARTS   AGE
my-release-etcd-0                             1/1     Running     0          1m
my-release-milvus-datacoord-7b5d84d8c6-rzjml  1/1     Running     0          1m
my-release-milvus-datanode-665d4586b9-525pm   1/1     Running     0          1m
my-release-milvus-indexcoord-9669d5989-kr5cm  1/1     Running     0          1m
my-release-milvus-indexnode-b89cc5756-xbpbn   1/1     Running     0          1m
my-release-milvus-proxy-7cbcc8ffbc-4jn8d      1/1     Running     0          1m
my-release-milvus-pulsar-6b9754c64d-4tg4m     1/1     Running     0          1m
my-release-milvus-querycoord-75f6c789f8-j28bg 1/1     Running     0          1m
my-release-milvus-querynode-7c7779c6f8-pnjzh  1/1     Running     0          1m
my-release-milvus-rootcoord-75585dc57b-cjh87  1/1     Running     0          1m
my-release-minio-5564fbbddc-9sbgv             1/1     Running     0          1m
```

Milvus only supports adding the worker nodes and does not support adding the coordinator components.

Scale a Milvus cluster

You can scale in your Milvus cluster either manually or automatically. If autoscaling is enabled, the Milvus cluster will shrink or expand automatically when CPU and memory resources consumption reaches the value you have set.

Manual scaling

Scaling out

Run `helm upgrade my-release milvus/milvus --set queryNode.replicas=3 --reuse-values` to manually scale out the query node.

If successful, three running pods on the query node are added as shown in the following example.

```
NAME                                          READY   STATUS    RESTARTS   AGE
my-release-etcd-0                             1/1     Running   0          2m
my-release-milvus-datacoord-7b5d84d8c6-rzjml  1/1     Running   0          2m
my-release-milvus-datanode-665d4586b9-525pm   1/1     Running   0          2m
my-release-milvus-indexcoord-9669d5989-kr5cm  1/1     Running   0          2m
my-release-milvus-indexnode-b89cc5756-xbpbn   1/1     Running   0          2m
my-release-milvus-proxy-7cbcc8ffbc-4jn8d      1/1     Running   0          2m
my-release-milvus-pulsar-6b9754c64d-4tg4m     1/1     Running   0          2m
my-release-milvus-querycoord-75f6c789f8-j28bg 1/1     Running   0          2m
my-release-milvus-querynode-7c7779c6f8-czq9f  1/1     Running   0          5s
my-release-milvus-querynode-7c7779c6f8-jcdcn  1/1     Running   0          5s
my-release-milvus-querynode-7c7779c6f8-pnjzh  1/1     Running   0          2m
my-release-milvus-rootcoord-75585dc57b-cjh87  1/1     Running   0          2m
my-release-minio-5564fbbddc-9sbgv             1/1     Running   0          2m
```

Scaling in

Run `helm upgrade my-release milvus/milvus --set queryNode.replicas=1 --reuse-values` to scale in the query node.

If successful, three running pods on the query node are reduced to one as shown in the following example.

```
NAME                                          READY   STATUS    RESTARTS   AGE
my-release-etcd-0                             1/1     Running   0          2m
my-release-milvus-datacoord-7b5d84d8c6-rzjml  1/1     Running   0          2m
my-release-milvus-datanode-665d4586b9-525pm   1/1     Running   0          2m
my-release-milvus-indexcoord-9669d5989-kr5cm  1/1     Running   0          2m
my-release-milvus-indexnode-b89cc5756-xbpbn   1/1     Running   0          2m
my-release-milvus-proxy-7cbcc8ffbc-4jn8d      1/1     Running   0          2m
my-release-milvus-pulsar-6b9754c64d-4tg4m     1/1     Running   0          2m
my-release-milvus-querycoord-75f6c789f8-j28bg 1/1     Running   0          2m
my-release-milvus-querynode-7c7779c6f8-pnjzh  1/1     Running   0          2m
my-release-milvus-rootcoord-75585dc57b-cjh87  1/1     Running   0          2m
my-release-minio-5564fbbddc-9sbgv             1/1     Running   0          2m
```

Autoscaling

Run the following command to enable autoscaling for query node. You also need to configure the value for CPU and memory resource to trigger autoscaling.

```
helm upgrade my-release milvus/milvus --set queryNode.autoscaling.enabled=true --reuse-values
```

What's next

- If you want to learn how to monitor the Milvus services and create alerts:
  - Learn Monitor Milvus 2.0 with Prometheus Operator on Kubernetes
- If you are ready to deploy your cluster on clouds:
  - Learn how to Deploy Milvus on AWS with Terraform and Ansible
  - Learn how to Deploy Milvus on Amazon EKS with Terraform
  - Learn how to Deploy Milvus Cluster on GCP with Kubernetes
  - Learn how to Deploy Milvus on Microsoft Azure With Kubernetes
- If you are looking for instructions on how to allocate resources:
  - Allocate Resources on Kubernetes

## 升级 Milvus 2.0

使用 Helm Chart 升级 Milvus 2.0 版本

本文将介绍如何使用 Helm Chart 升级 Milvus 2.0 版本。本文以 Milvus 2.0.0-RC7 升级至 2.0.0-RC8 为例。

目前，暂不暂不支持使用 Helm Chart 实现单机版 Milvus 与分布式版 Milvus 之间的升级转换。Milvus 2.0.0-RC7 与此前版本不兼容，因此不支持从此前版本升级至 2.0.0-RC7 版本。

升级单机版 Milvus

步骤 1. 查看 Milvus 版本

运行指令 `$ helm list`，查看 Milvus app 版本. 你可以看到返回结果中显示 Milvus `APP VERSION` 为 2.0.0-rc7。

```
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS
my-release      default         1               2021-11-08 17:12:44.678247 +0800 CST    deployed
```

步骤 2. 查看运行中的 pod

运行指令 `$ kubectl get pods`，查看运行中的 pod。你可以看到如下结果：

```
NAME                                          READY   STATUS    RESTARTS   AGE
my-release-etcd-0                             1/1     Running   0          84s
my-release-milvus-standalone-75c599fffc-6rwlj   1/1     Running   0          84s
my-release-minio-744dd9586f-qngzv             1/1     Running   0          84s
```

步骤 3. 查看 image tag

查看 `my-release-milvus-standalone-75c599fffc-6rwlj` pod 的 image tag。可以看到你所使用的单机版 Milvus 版本为 2.0.0-RC7。

```
$ kubectl get pods my-release-milvus-standalone-75c599fffc-6rwlj -o=jsonpath='{$.spec.containers[0].image}
```

```
milvusdb/milvus:v2.0.0-rc7-20211011-d567b21
```

步骤 4. 检查所有可用 app 版本

运行如下指令，检查所有可用 app 版本。

```
$ helm repo update
$ helm search repo milvus --versions
```

```
NAME                    CHART VERSION        APP VERSION             DESCRIPTION
milvus/milvus           2.3.3                2.0.0-rc.8              Milvus is an open-source vector databas
milvus/milvus           2.3.2                2.0.0-rc.8              Milvus is an open-source vector databas
milvus/milvus           2.3.1                2.0.0-rc.8              Milvus is an open-source vector databas
milvus/milvus           2.3.0                2.0.0-rc.8              Milvus is an open-source vector databas
milvus/milvus           2.2.6                2.0.0-rc.7              Milvus is an open-source vector databas
milvus/milvus           2.2.5                2.0.0-rc.7              Milvus is an open-source vector databas
milvus/milvus           2.2.4                2.0.0-rc.7              Milvus is an open-source vector databas
milvus/milvus           2.2.3                2.0.0-rc.7              Milvus is an open-source vector databas
milvus/milvus           2.2.2                2.0.0-rc.7              Milvus is an open-source vector databas
milvus/milvus           2.2.1                2.0.0-rc.6              Milvus is an open-source vector databas
milvus/milvus           2.2.0                2.0.0-rc.6              Milvus is an open-source vector databas
```

步骤 5. 升级单机版 Milvus

1. 运行如下指令，将单机版 Milvus 2.0.0-RC7 升级至 2.0.0-RC8。

```
$ helm repo update
$ helm upgrade my-release milvus/milvus --set cluster.enabled=false --set etcd.replicaCount=1 --set minio.
```

2. 再次运行指令 `$ helm list`，查看当前 Milvus app 版本。可以看到当前单机版 Milvus 已升级至 2.0.0-RC8。

```
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS
my-release      default         2               2021-11-08 17:15:46.530627 +0800 CST    deployed
```

3. 运行指令 `$ kubectl get pods`，查看当前运行中的 pod。你可以看到如下结果：

```
NAME                                          READY   STATUS    RESTARTS   AGE
my-release-etcd-0                             1/1     Running   0          3m32s
my-release-milvus-standalone-6967454987-72r55 1/1     Running   0          22s
my-release-minio-744dd9586f-qngzv             1/1     Running   0          3m32s
```

升级单机版 Milvus 时，原有的 pod 将被删除。因此，Milvus 服务可能会暂时中断。

4. 运行如下指令，查看当前 image tag 版本。可以看到，当前版本为 v2.0.0-rc8。

```
$ kubectl get pods my-release-milvus-standalone-6967454987-72r55 -o=jsonpath='{$.spec.containers[0].image}
```

```
milvusdb/milvus:v2.0.0-rc8-20211104-d1f4106
```

升级分布式版 Milvus

步骤 1. 查看 Milvus 版本

运行指令 `$ helm list`，查看 Milvus app 版本. 你可以看到返回结果中显示 Milvus `APP VERSION` 为 2.0.0-rc7。

```
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS
my-release      default         1               2021-11-08 17:21:13.511069 +0800 CST    deployed
```

步骤 2. 查看运行中的 pod

运行指令 `$ kubectl get pods`，查看运行中的 pod。你可以看到如下结果：

```
NAME                                          READY   STATUS      RESTARTS   AGE
my-release-etcd-0                             1/1     Running     0          5m40s
my-release-etcd-1                             1/1     Running     0          5m40s
my-release-etcd-2                             1/1     Running     0          5m40s
my-release-milvus-datacoord-c99d7dfdf-mjghl   1/1     Running     0          5m40s
my-release-milvus-datanode-69cccf85d8-9r8ph   1/1     Running     0          5m40s
my-release-milvus-indexcoord-64f7d548fb-46hn8 1/1     Running     0          5m40s
my-release-milvus-indexnode-57b96d9cc7-gvmvl  1/1     Running     0          5m40s
my-release-milvus-proxy-6664d564f9-pwqn9      1/1     Running     0          5m40s
my-release-milvus-querycoord-59767cb88c-n5416 1/1     Running     0          5m40s
my-release-milvus-querynode-847ccdf855-78mnz  1/1     Running     0          5m40s
my-release-milvus-rootcoord-597bd9f565-2jgzq  1/1     Running     0          5m40s
my-release-minio-0                            1/1     Running     0          5m40s
my-release-minio-1                            1/1     Running     0          5m40s
my-release-minio-2                            1/1     Running     0          5m40s
my-release-minio-3                            1/1     Running     0          5m40s
my-release-pulsar-autorecovery-869bffb7b8-g4cbh 1/1   Running     0          5m40s
my-release-pulsar-bastion-7c659df966-86b5s    1/1     Running     0          5m40s
my-release-pulsar-bookkeeper-0                1/1     Running     0          5m40s
my-release-pulsar-bookkeeper-1                1/1     Running     0          3m54s
my-release-pulsar-broker-864775f5ff-zlnfx     1/1     Running     0          5m40s
my-release-pulsar-proxy-86bcdbbb4c-24kcj      2/2     Running     0          5m40s
my-release-pulsar-zookeeper-0                 1/1     Running     0          5m40s
my-release-pulsar-zookeeper-1                 1/1     Running     0          5m20s
my-release-pulsar-zookeeper-2                 1/1     Running     0          5m5s
my-release-pulsar-zookeeper-metadata-hw5xt    0/1     Completed   0          5m40s
```

步骤 3. 查看 image tag

查看 my-release-milvus-proxy-6664d564f9-pwqn9 pod 的 image tag。可以看到你所使用的分布式版 Milvus 版本为 2.0.0-RC7。

```
$ kubectl get pods my-release-milvus-proxy-6664d564f9-pwqn9 -o=jsonpath='{$.spec.containers[0].image}'

milvusdb/milvus:v2.0.0-rc7-20211011-d567b21
```

步骤 4. 检查所有可用 app 版本

运行如下指令，检查所有可用 app 版本。

```
$ helm repo update
$ helm search repo milvus --versions
```

```
NAME                    CHART VERSION        APP VERSION              DESCRIPTION
milvus/milvus           2.3.3                2.0.0-rc.8               Milvus is an open-source vector databas
milvus/milvus           2.3.2                2.0.0-rc.8               Milvus is an open-source vector databas
milvus/milvus           2.3.1                2.0.0-rc.8               Milvus is an open-source vector databas
milvus/milvus           2.3.0                2.0.0-rc.8               Milvus is an open-source vector databas
milvus/milvus           2.2.6                2.0.0-rc.7               Milvus is an open-source vector databas
milvus/milvus           2.2.5                2.0.0-rc.7               Milvus is an open-source vector databas
milvus/milvus           2.2.4                2.0.0-rc.7               Milvus is an open-source vector databas
milvus/milvus           2.2.3                2.0.0-rc.7               Milvus is an open-source vector databas
milvus/milvus           2.2.2                2.0.0-rc.7               Milvus is an open-source vector databas
milvus/milvus           2.2.1                2.0.0-rc.6               Milvus is an open-source vector databas
milvus/milvus           2.2.0                2.0.0-rc.6               Milvus is an open-source vector databas
```

步骤 5. 升级分布式版 Milvus

1. 运行如下指令，将分布式版 Milvus 2.0.0-RC7 升级至 2.0.0-RC8。

```
$ helm repo update
$ helm upgrade my-release milvus/milvus
```

2. 再次运行指令 $ helm list，查看当前 Milvus app 版本。可以看到当前单机版 Milvus 已升级至 2.0.0-RC8。

```
NAME                NAMESPACE           REVISION            UPDATED                                         STATUS
my-release          default             2                   2021-11-08 17:29:07.815765 +0800 CST            deployed
```

3. 行指令 $ kubectl get pods，查看当前运行中的 pod。你可以看到如下结果:

```
NAME                                            READY   STATUS    RESTARTS   AGE
my-release-etcd-0                               1/1     Running   0          71s
my-release-etcd-1                               1/1     Running   0          2m34s
my-release-etcd-2                               1/1     Running   0          3m41s
my-release-milvus-datacoord-76d55548b6-zl4kj    1/1     Running   0          3m45s
my-release-milvus-datanode-5b9774cc75-dhn7j     1/1     Running   0          3m45s
my-release-milvus-indexcoord-96549bfff-r9m99    1/1     Running   0          3m45s
my-release-milvus-indexnode-f7c9b444b-vjqnm     1/1     Running   0          3m44s
my-release-milvus-proxy-5685bbc546-v6scq        1/1     Running   0          3m44s
my-release-milvus-querycoord-5fcd65544-8m6lb    1/1     Running   0          3m44s
my-release-milvus-querynode-5b76d575f6-2szfj    1/1     Running   0          3m44s
my-release-milvus-rootcoord-8668f8c46b-9nss2    1/1     Running   0          3m44s
my-release-minio-0                              1/1     Running   0          11m
my-release-minio-1                              1/1     Running   0          11m
my-release-minio-2                              1/1     Running   0          11m
my-release-minio-3                              1/1     Running   0          11m
my-release-pulsar-autorecovery-869bffb7b8-g4cbh 1/1     Running   0          11m
my-release-pulsar-bastion-7c659df966-86b5s      1/1     Running   0          11m
my-release-pulsar-bookkeeper-0                  1/1     Running   0          11m
```

```
my-release-pulsar-bookkeeper-1                    1/1     Running   0          9m55s
my-release-pulsar-broker-864775f5ff-zlnfx         1/1     Running   0          11m
my-release-pulsar-proxy-86bcdbbb4c-24kcj          2/2     Running   0          11m
my-release-pulsar-zookeeper-0                     1/1     Running   0          11m
my-release-pulsar-zookeeper-1                     1/1     Running   0          11m
my-release-pulsar-zookeeper-2                     1/1     Running   0          11m
```

4. 运行如下指令，查看当前 image tag 版本。可以看到，当前版本为 v2.0.0-rc8。

```
$ kubectl get pods my-release-milvus-proxy-5685bbc546-v6scq -o=jsonpath='{$.spec.containers[0].image}'

milvusdb/milvus:v2.0.0-rc8-20211104-d1f4106
```

更多内容

- 你可能还想了解：
  - 对 Milvus 集群进行扩缩容
- 如果你想要在云端部署分布式版 Milvus：
  - 使用 Terraform 及 Ansible 在 AWS 上部署 Milvus
  - 使用 Terraform 在 Amazon EKS 上部署 Milvus
  - 使用 Kubernetes 在 GCP 上部署 Milvus
  - 使用 Kubernentes 在 Microsoft Azure 上部署 Milvus

## 监控与报警

监控框架概述

Milvus monitoring framework overview

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic explains how Milvus uses Prometheus to monitor metrics and Grafana to visualize metrics and create alerts.

Prometheus in Milvus

Prometheus is an open-source monitoring and alerting toolkit for Kubernetes implementations. It collects and stores metrics as time-series data. This means that metrics are stored with timestamps when recorded, alongside with optional key-value pairs called labels. Currently Milvus uses the following components of Prometheus: - Prometheus endpoint to pull data from endpoints set by exporters. - Prometheus operator to effectively manage Prometheus monitoring instances. - Kube-prometheus to provide easy to operate end-to-end Kubernetes cluster monitoring.

Grafana in Milvus

Grafana is a visualizing stack. It features a dashboard that can help you visualize all the data and metrics you need. With the Grafana dashboard, you can query, understand, and analyze your data.

What’s next

After learning about the basic workflow of monitoring and alerting, learn: - Deploy monitoring services - Visualize Milvus metrics - Create an alert

部署监控

Deploying Monitoring Services on Kubernetes

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to use Prometheus to deploy monitoring services for a Milvus cluster on Kubernetes.

## Monitor metrics with Prometheus

Metrics are indicators providing information about the running status of your system. For example, with metrics, you can understand how much memory or CPU resources are consumed by a data node in Milvus. Being aware of the performance and status of the components in your Milvus cluster makes you well-informed and hence making better decisions and adjusting resource allocation in a more timely manner.

Generally, metrics are stored in a time series database (TSDB), like Prometheus, and the metrics are recorded with a timestamp. In the case of monitoring Milvus services, you can use Prometheus to pull data from endpoints set by exporters. Prometheus then exports metrics of each Milvus component at `http://<component-host>:9091/metrics`.

However, you might have several replicas for one component, which makes manual configuration of Prometheus too complicated. Therefore, you can use Prometheus Operator, an extension to Kubernetes, for automated and effective management of Prometheus monitoring instances. Using Prometheus Operator saves you the trouble of manually adding metric targets and service providers.

The ServiceMonitor Custom Resource Definition (CRD) enables you to declaratively define how a dynamic set of services are monitored. It also allows selecting which services to monitor with the desired configuration using label selections. With Prometheus Operator, you can introduce conventions specifying how metrics are exposed. New services can be automatically discovered following the convention you set without the need for manual reconfiguration.

The following image illustrates Prometheus workflow.




## Prerequisites

This tutorial uses kube-prometheus to save you the trouble of installing and manually configuring each monitoring and alerting component.

Kube-prometheus collects Kubernetes manifests, Grafana dashboards, and Prometheus rules combined with documentation and scripts.

Before deploying monitoring services, you need to create a monitoring stack by using the configuration in the kube-prometheus manifests directory.

```
$ git clone https://github.com/prometheus-operator/kube-prometheus.git
$ cd ### to the local path of the repo
$ kubectl create -f manifests/setup
$ until kubectl get servicemonitors --all-namespaces ; do date; sleep 1; echo ""; done
$ kubectl create -f manifests/
```

To delete a stack, run `kubectl delete --ignore-not-found=true -f manifests/ -f manifests/setup`.

## Deploy monitoring services on Kubernetes

1. Access the dashboards

You can access Prometheus via `http://localhost:9090`, and Grafana at `http://localhost:3000`.

```
$ kubectl --namespace monitoring port-forward svc/prometheus-k8s 9090
$ kubectl --namespace monitoring port-forward svc/grafana 3000
```

2. Enable ServiceMonitor

The ServiceMonitor is not enabled for Milvus Helm by default. After installing the Prometheus Operator in the Kubernetes cluster, you can enable it by adding the parameter `metrics.serviceMontior.enabled=true`.

```
$ helm install my-release milvus/milvus --set metrics.serviceMonitor.enabled=true
```

When the installation completes, use `kubectl` to check the ServiceMonitor resource.

```
$ kubectl get servicemonitor
```

```
NAME                            AGE
my-release-milvus               54s
```

What’s next

- If you have deployed monitoring services for the Milvus cluster, you might also want to learn to:
  - Visualize Milvus metrics in Grafana
  - Create an Alert for Milvus Services
  - Adjust your resource allocation
- If you are looking for information about how to scale a Milvus cluster:
  - Learn scale a Milvus cluster
- If you are interested in upgrading the Milvus 2.0 version,
  - Read the upgrading guide

可视化监控指标

Visualize Milvus metrics in Grafana

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic describes how to visualize Milvus metrics using Grafana.

As described in the monitoring guide, metrics contain useful information such as how much memory is used by a specific Milvus component. Monitoring metrics helps you better understand Milvus performance and its running status so that you can adjust resource allocation timely.

Visualization is a chart showing the change of resource usage across time, which makes it easier for you to quickly see and notice the changes to resource usage especially when an event occurs.

This tutorial uses Grafana, an open-source platform for time-series analytics, to visualize various performance metrics of Milvus.

Prerequisites

You need to configure Prometheus to monitor and collect metrics before using Grafana to visualize the metrics. If the setup is successful, you can access Grafana at `http://localhost:3000`. Or you can also access Grafana using the default Grafana `user:password` of `admin:admin`.

Visualize metrics using Grafana

1. Download and import dashboard

Download and import Milvus dashboard from the JSON file.

```
wget https://raw.githubusercontent.com/milvus-io/milvus/master/deployments/monitor/grafana/milvus-dashboar
```

Download_and_import

2. View metrics

Select the Milvus instance you want to monitor. Then you can see the Milvus components panel.

Select_instance

Grafana_panel

What’s next

- If you have set Grafana to visualize Milvus metrics, you might also want to:
  - Learn how to create an alert for Milvus services
  - Adjust your resource allocation
  - Scale out or scale in a Milvus cluster
- If you are interested in upgrading the Milvus 2.0 version,
  - Read the upgrading guide

创建报警

Create an Alert for Milvus Services

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic introduces the alert mechanism for Milvus services and explains why, when, and how to create alerts in Milvus.

By creating alerts, you can receive notifications when the value of a specific metric exceeds the threshold you have predefined.

For example, you create an alert and set 80 MB as the maximum value for memory usage by Milvus components. If the actual usage exceeds the predefined number, you will receive alerts reminding you that the memory usage by Milvus component surpasses 80 MB. Upon the alert, you can then adjust the allocation of resources accordingly and timely to ensure service availability.

Scenarios for creating alerts.

Below are some common scenarios where you need to create an alert for.

- CPU or memory usage by Milvus components is too high.
- Milvus component pods are running low on disk space.
- Milvus component pods are restarting too frequently.

The following metrics are available for alerting configuration:

| Metric | Description | Unit of measure |
|---|---|---|
| CPU Usage | CPU usage by Milvus components that is indicated by the running time of CPU. | Second |
| Memory | Memory resources consumed by Milvus components. | MB |
| Goroutines | Concurrent executing activities in GO language. | / |
| OS Threads | Threads, or lightweight processes in an operating system. | / |
| Process Opened Fds | The current number of used file descriptors. | / |

Set up alerts

This guide takes the example of creating an alert for the memory usage of Milvus components. To create other types of alerts, please adjust your commands accordingly. If you encounter any problems during the process, feel free to ask in the Milvus forum or initiate a discussion on Slack.

Prerequisites

This tutorial assumes that you have Grafana installed and configured. If not, we recommend reading the monitoring guide.

1. Add a new query

To add an alert for the memory usage of Milvus components, edit the Memory panel. Then, add a new query with the metric: `process_resident_memory_bytes{app_kubernetes_io_name="milvus", app_kubernetes_io_instance=~"my-relea namespace="default"}`

Alert_metric

2. Save the dashboard

Save the dashboard, and wait for a few minutes to see the alert.

Alert_dashboard

Grafana alert query does not support template variables. Therefore, you should add a second query without any template variables in the labels. The second query is named as "A" by default. You can rename it by clicking on the dropdown.

Alert_query

3. Add alert notifications

To receive alert notifications, add a "notification channel". Then, specify the channel in the field "Send to".

Alert_notification

If the alert is successfully created and triggered, you will receive the notification as shown in the screenshot below.

Notification_message

To delete an alert, go to the "Alert" panel and click the delete button.

Delete_alert

What's next

- If you need to start monitoring services for Milvus:
  - Read the monitoring guide
  - Learn how to visualize monitoring metrics
- If you have created alerts for memory usage by Milvus components:
  - Learn how to allocate resources
- If you are looking for information about how to scale a Milvus cluster:
  - Learn scale a Milvus cluster

## 数据迁移

HDF5 至 Milvus

将 HDF5 文件数据导入 Milvus

本文将介绍如何使用开源工具 MilvusDM 将 HDF5 文件数据导入 Milvus，实现数据迁移。

数据迁移前提

在迁移数据前，你需要先 安装 MilvusDM。

1. 下载 YAML 文件

下载 M2H.yaml 文件。

```
$ wget https://raw.githubusercontent.com/milvus-io/milvus-tools/main/yamls/M2H.yaml
```

2. 设置参数

需要设置的参数包括:

| 参数 | 说明 | 示例 |
| --- | --- | --- |
| milvus_version | Milvus 版本。 | 2.0.0 |
| data_path | HDF5 文件路径。data_path 及 data_dir 两个参数中只能配置一个。 | - /Users/zilliz/float_1.h5 - /Users/zilliz/float_2.h5 |
| data_dir | HDF5 文件目录。data_path 及 data_dir 两个参数中只能配置一个。 | '/Users/zilliz/Desktop/HDF5_data' |
| dest_host | 目标 Milvus 服务器地址。 | '127.0.0.1' |
| dest_port | 目标 Milvus 服务器端口。 | 19530 |
| mode | 数据迁移模式，包括 skip、append 及 overwrite。该参数仅在指定 collction 名称存在于 Milvus 中时生效。 | 'append' |
| dest_collection_name | 导入数据的 collection 名称。 | 'test_float' |
| dest_partition_name （可选参数） | 导入数据的 partition 名称。 | 'partition_1' |
| collection_parameter | collection 相关信息，包括向量维度、索引文件大小、相似度计算方式等。 | "dimension: 512 index_file_size: 1024 metric_type: 'HAMMING' " |

如下两个参数配置示例仅供参考。示例 1 中设置了 data_path 参数。示例 2 中设置了 data_dir 参数。你可以根据需求，从参数 data_path 及 data_dir 中选择一个进行配置。

示例 1

```
H2M:
  milvus-version: 2.0.0
  data_path:
    - /Users/zilliz/float_1.h5
    - /Users/zilliz/float_2.h5
  data_dir:
  dest_host: '127.0.0.1'
  dest_port: 19530
  mode: 'overwrite'        ### 'skip/append/overwrite'
  dest_collection_name: 'test_float'
  dest_partition_name: 'partition_1'
  collection_parameter:
    dimension: 128
    index_file_size: 1024
    metric_type: 'L2'
```

示例 2

```
H2M:
  milvus_version: 2.0.0
  data_path:
  data_dir: '/Users/zilliz/HDF5_data'
  dest_host: '127.0.0.1'
  dest_port: 19530
```

```
  mode: 'append'        ### 'skip/append/overwrite'
  dest_collection_name: 'test_binary'
  dest_partition_name:
  collection_parameter:
    dimension: 512
    index_file_size: 1024
    metric_type: 'HAMMING'
```

3. 将 HDF5 文件数据导入 Milvus

运行 MilvusDM，通过如下指令将 HDF5 文件数据导入 Milvus。

```
$ milvusdm --yaml H2M.yaml
```

更多内容

- 如果你想要将其他格式的数据导入 Milvus，你可以：
  - 了解如何 将 Faiss 数据导入 Milvus。
- 如果你想要了解如何将 Milvus 1.x 数据迁移至 Milvus 2.0，
  - 详见 版本迁移。
- 如果你想要了解更多有关数据迁移工具详情，
  - 阅读 MilvusDM 简介。

Faiss 至 Milvus

将 Faiss 数据导入 Milvus

本文将介绍如何使用开源工具 MilvusDM 将 Faiss 数据导入 Milvus，实现数据迁移。

数据迁移前提

在迁移数据前，你需要先 安装 MilvusDM。

1. 下载 YAML 文件

下载 `F2M.yaml` 文件。

```
$ wget https://raw.githubusercontent.com/milvus-io/milvus-tools/main/yamls/F2M.yaml
```

2. 设置参数

需要设置的参数包括：

| 参数 | 说明 | 示例 |
|---|---|---|
| `milvus_version` | Milvus 版本。 | 2.0.0 |
| `data_path` | HDF5 文件路径。`data_path` 及 `data_dir` 两个参数中只能配置一个。 | ‘/home/user/data/faiss.index’ |
| `data_dir` | HDF5 文件目录。`data_path` 及 `data_dir` 两个参数中只能配置一个。 | ‘/Users/zilliz/Desktop/HDF5_data’ |
| `dest_host` | 目标 Milvus 服务器地址。 | ‘127.0.0.1’ |
| `dest_port` | 目标 Milvus 服务器端口。 | 19530 |
| `mode` | 数据迁移模式，包括 `skip`、`append` 及 `overwrite`。该参数仅在指定 collction 名称存在于 Milvus 中时生效。 | ‘append’ |
| `dest_collection_name` | 导入数据的 collection 名称。 | ‘test’ |
| `dest_partition_name` (optional) | 导入数据的 partition 名称。 | ‘partition’ |

| 参数 | 说明 | 示例 |
| --- | --- | --- |
| collection_parameter | collection 相关信息，包括向量维度、索引文件大小、相似度计算方式等。 | “dimension: 512<br>index_file_size: 1024<br>metric_type: ‘HAMMING’ ” |

示例

如下参数配置示例仅供参考。

```
F2M:
  milvus_version: 2.0.0
  data_path: '/home/data/faiss1.index'
  dest_host: '127.0.0.1'
  dest_port: 19530
  mode: 'append'
  dest_collection_name: 'test'
  dest_partition_name: ''
  collection_parameter:
    dimension: 256
    index_file_size: 1024
    metric_type: 'L2'
```

3. 将 Faiss 数据导入 Milvus

运行 MilvusDM，通过如下指令将 Faiss 数据导入 Milvus。

```
$ milvusdm --yaml F2M.yaml
```

更多内容

- 如果你想要将其他格式的数据导入 Milvus，你可以:
  - 了解如何 将 HDF5 文件数据导入 Milvus。
- 如果你想要了解如何将 Milvus 1.x 数据迁移至 Milvus 2.0，
  - 详见 版本迁移。
- 如果你想要了解更多有关数据迁移工具详情，
  - 阅读 MilvusDM 简介。

Milvus 1.x 至 2.0

Milvus 版本迁移

本文将介绍如何使用开源工具 MilvusDM 将 Milvus 1.x 数据导入 Milvus 2.0，实现版本迁移。

MilvusDM 不支持将数据从单机版 Milvus 迁移至分布式版 Milvus。

数据迁移前提

在迁移 Milvus 版本前，你需要先 安装 MilvusDM。

1. 下载 YAML 文件

下载 M2M.yaml 文件。

```
$ wget https://raw.githubusercontent.com/milvus-io/milvus-tools/main/yamls/M2M.yaml
```

2. 设置参数

需要设置的参数包括:

| 参数 | 说明 | 示例 |
| --- | --- | --- |
| milvus_version | Milvus 版本。 | 2.0.0 |
| data_path | HDF5 文件路径。data_path 及 data_dir 两个参数中只能配置一个。 | - /Users/zilliz/float_1.h5 - /Users/zilliz/float_2.h5 |
| data_dir | HDF5 文件目录。data_path 及 data_dir 两个参数中只能配置一个。 | '/Users/zilliz/Desktop/HDF5_data' |
| dest_host | 目标 Milvus 服务器地址。 | '127.0.0.1' |
| dest_port | 目标 Milvus 服务器端口。 | 19530 |
| mode | 数据迁移模式，包括 skip、append 及 overwrite。该参数仅在指定 collction 名称存在于 Milvus 中时生效。 | 'append' |
| dest_collection_name | 导入数据的 collection 名称。 | 'test_float' |
| dest_partition_name (optional) | 导入数据的 partition 名称。 | 'partition_1' |
| collection_parameter | collection 相关信息，包括向量维度、索引文件大小、相似度计算方式等。 | "dimension: 512 index_file_size: 1024 metric_type: 'HAMMING' " |

如下两个参数配置示例仅供参考。示例 1 中设置了 mysql_parameter 参数。如未在 Milvus 1.x 版本中使用 MySQL 管理向量 ID，参考示例 2。

示例 1

```
M2M:
  milvus_version: 2.0.0
  source_milvus_path: '/home/user/milvus'
  mysql_parameter:
    host: '127.0.0.1'
    user: 'root'
    port: 3306
    password: '123456'
    database: 'milvus'
  source_collection: ### specify the 'partition_1' and 'partition_2' partitions of the 'test' collection.
    test:
      - 'partition_1'
      - 'partition_2'
  dest_host: '127.0.0.1'
  dest_port: 19530
  mode: 'skip' ### 'skip/append/overwrite'
```

示例 2

```
M2M:
  milvus_version: 2.0.0
  source_milvus_path: '/home/user/milvus'
  mysql_parameter:
  source_collection: ### specify the collection named 'test'
    test:
  dest_host: '127.0.0.1'
  dest_port: 19530
  mode: 'skip' ### 'skip/append/overwrite'
```

3. 将 Milvus 1.x 数据导入 Milvus 2.0

运行 MilvusDM，通过如下指令将 Milvus 1.x 数据导入 Milvus 2.0。

```
$ milvusdm --yaml M2M.yaml
```

更多内容

- 如果你想要将其他格式的数据导入 Milvus，你可以:
  - 了解如何 将 Faiss 数据导入 Milvus。
  - 了解如何 将 HDF5 文件数据导入 Milvus。
- 如果你想要了解更多有关数据迁移工具详情，
  - 阅读 MilvusDM 简介。

# 开发工具

## Milvus CLI

简介

什么是 Milvus_CLI?

Milvus 命令行接口 (CLI) 是一个支持数据库连接、数据操作、以及数据导入和导出的命令行工具。基于 Milvus Python SDK，它允许使用交互式命令行提示符通过终端执行命令。

推荐版本

在下面的表格中，你可以找到根据你正在使用的 Milvus 版本推荐的 PyMilvus 和 Milvus 版本。

| Milvus | PyMilvus | Milvus_CLI |
|---|---|---|
| 1.0.x | 1.0.1 | x |
| 1.1.x | 1.1.2 | x |
| 2.0.0-RC1 | 2.0.0rc1 | x |
| 2.0.0-RC2 | 2.0.0rc2 | 0.1.3 |
| 2.0.0-RC4 | 2.0.0rc4 | 0.1.4 |
| 2.0.0-RC5 | 2.0.0rc5 | 0.1.5 |
| 2.0.0-RC6 | 2.0.0rc6 | 0.1.6 |
| 2.0.0-RC7 | 2.0.0rc7 | 0.1.7 |
| 2.0.0-RC8 | 2.0.0rc8 | 0.1.8 |
| 2.0.0-RC9 | 2.0.0rc9 | 0.1.9 |

由于存储格式的变化，Milvus 2.0.0-RC7 及更高版本不能向 2.0.0-RC6 及更低版本兼容。

当前版本

Milvus_CLI 的当前版本是 0.1.9。如果想查找已安装的版本，并查看是否需要更新，请运行 `milvus_cli --version` 命令。

安装

安装 Milvus_CLI

本文将介绍如何安装 Milvus_CLI。

通过 PyPI 安装

我们建议你从 PyPI 安装 Milvus_CLI。

安装前提

- 安装 Python 3.8.5 或更高版本
- 安装 pip

使用 pip 安装

安装 Milvus_CLI。

```
pip install milvus-cli
```

使用 Docker 安装

你也可以使用 Docker 安装 Milvus_CLI。

安装前提

需要 Docker 19.03 或更高版本。

通过 Docker 镜像安装

```
$ docker run -it zilliz/milvus_cli:latest
```

从源代码安装

1. 下载 `milvus_cli` 仓库到本地。

```
git clone https://github.com/zilliztech/milvus_cli.git
```

2. 进入 `milvus_cli` 文件夹。

```
cd milvus_cli
```

3. 安装 Milvus。

```
python -m pip install --editable .
```

你也可以从这里下载包含源代码的 tarball 压缩文件并运行如下命令来安装 Milvus_CLI。

```
python -m pip install milvus_cli-<version>.tar.gz
```

命令

Milvus_CLI 命令参考

Milvus 命令行接口 (CLI) 是一个支持数据库连接、数据操作、以及数据导入和导出的命令行工具。基于 Milvus Python SDK，它允许使用交互式命令行提示符通过终端执行命令。本文介绍了所有支持的命令及其参数，还包括了一些示例供你参考。

calc distance

计算两个向量数组之间的距离。

语法

```
calc distance
```

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| help | n/a | 显示用法信息。 |

示例

根据提示输入需要的信息，计算两个向量数组之间的距离。

```
milvus_cli > calc distance

Import left operator vectors from existing collection? [y/N]: n

The vector's type (float_vectors, bin_vectors): float_vectors

Left vectors:
    [[0.083, 0.992, 0.931, 0.433, 0.93, 0.706, 0.668, 0.481, 0.255, 0.088, 0.121, 0.701, 0.935, 0.142, 0.0

Import right operator vectors from existing collection? [y/N]: n

The vector's type (float_vectors, bin_vectors): float_vectors

Right vectors:
    [[0.518, 0.034, 0.786, 0.251, 0.04, 0.247, 0.55, 0.595, 0.638, 0.957, 0.303, 0.023, 0.007, 0.712, 0.84

Supported metric type. Default is "L2" (L2, IP, HAMMING, TANIMOTO) [L2]:
L2

sqrt [False]: True

Timeout(optional) []:

======
Return type:
Assume the vectors_left: L_1, L_2, L_3
Assume the vectors_right: R_a, R_b
Distance between L_n and R_m we called "D_n_m"
The returned distances are arranged like this:
[[D_1_a, D_1_b],
[D_2_a, D_2_b],
[D_3_a, D_3_b]]

Note: if some vectors do not exist in collection, the returned distance is "-1.0"
======

Result:

[[3.625464916229248, 3.234992742538452, 3.568333148956299, 3.694913148880005], [2.556027889251709, 2.89012
```

calc mkts_from_hybridts

根据现有的混合时间戳、timedelta 和增量时间间隔生成混合时间戳。

语法

```
calc mkts_from_hybridts -h (int) -m (float)
```

参数

| 参数 | 全名 | 描述 |
|---|---|---|
| -h | hybridts | 用于生成新的混合时间戳的原始混合时间戳。非负整数，取值范围为 0 ~ 18446744073709551615。 |
| -m | milliseconds | 增量间隔，以毫秒为单位。 |
| help | n/a | 显示使用命令的帮助。 |

calc mkts_from_unixtime

根据 Unix Epoch 时间、timedelta 和增量时间间隔生成混合时间戳。

语法

```
calc mkts_from_unixtime -e (float) -m (float)
```

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| -e | epoch | 用于生成混合时间戳的已知 Unix 时间戳。Unix epoch 是自 1970 年 1 月 1 日（UTC/GMT 午夜）以来经过的秒数。 |
| -m | milliseconds | 增量间隔，以毫秒为单位。 |
| help | n/a | 显示使用命令的帮助。 |

calc hybridts_to_unixtime

将混合时间戳转换为 UNIX 时间戳，忽略逻辑部分。

语法

```
calc hybridts_to_unixtime -h (int)
```

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| -h | hybridts | 要转换为 UNIX 时间戳的已知混合时间戳。非负整数，取值范围为 0 ~ 18446744073709551615。 |
| help | n/a | 显示使用命令的帮助。 |

clear

清除屏幕。

语法

```
clear
```

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| help | n/a | 显示用法信息。 |

connect

连接 Milvus。

语法

```
connect [-h (text)] [-p (int)] [-a (text)] [-D]
```

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| -h | host | （可选）主机名。默认是 “127.0.0.1”。 |
| -p | port | （可选）端口号。默认是” 19530”。 |
| -a | alias | （可选）Milvus 链接的别名。默认是” default”。 |
| -D | disconnect | （可选）从别名指定的 Milvus 服务器断开连接的开关。默认别名为” default”。 |
| help | n/a | 显示用法信息。 |

示例

```
milvus_cli > connect -h 127.0.0.1 -p 19530 -a default
```

create alias

指定集合的唯一别名。

集合可以有多个别名。但是，别名最多只能对应一个集合。

语法

```
create alias -c (text) -a (text) [-A] [-t (float)]
```

参数

| 参数 | 全名 | 描述 |
|---|---|---|
| -c | collection-name | 集合的名称。 |
| -a | alias-name | 别名。 |
| -A | alter | （可选）将别名转移到指定的集合的开关。 |
| -t | timeout | （可选）允许的远程过程调用的最大持续时间 (以秒为单位)。如果不传递此参数，客户端将一直等待直到服务 |
| help | n/a | 显示用法信息。 |

示例

示例 1

为 car 集合创建 carAlias1 和 carAlias2 别名。

```
milvus_cli > create alias -c car -a carAlias1 -a carAlias2
```

示例 2

示例 2 基于示例 1。

将 carAlias1 和 carAlias2 别名从 car 集合转移到 car2 集合。

```
milvus_cli > create alias -c car2 -A -a carAlias -a carAlias2
```

create collection

创建一个集合。

语法

```
create collection -c (text) -f (text) -p (text) [-a] [-d (text)]
```

参数

| 参数 | 全名 | 描述 |
|---|---|---|
| -c | collection-name | 集合的名称。 |
| -f | schema-field | （多个）使用 `<fieldName>:<dataType>:<dimOfVector/desc>` 格式表示的字段规范。 |
| -p | schema-primary-field | 主键字段的名称。 |
| -a | schema-auto-id | （可选）自动生成 ID 的开关。 |
| -d | schema-描述 | （可选）集合的描述。 |
| help | n/a | 显示用法信息。 |

示例

```
milvus_cli > create collection -c car -f id:INT64:primary_field -f vector:FLOAT_VECTOR:128 -f color:INT64:
```

create partition

创建一个分区。

语法

```
create partition -c (text) -p (text) [-d (text)]
```

参数

| 参数 | 全名 | 描述 |
|---|---|---|
| -c | collection | 集合的名称。 |
| -p | partition | 分区的名称。 |
| -d | description | （可选）分区的描述。 |
| help | n/a | 显示用法信息。 |

示例

```
milvus_cli > create partition -c car -p new_partition -d test_add_partition
```

create index

为字段创建一个索引。

目前，一个集合最多支持一个索引。

语法

```
create index
```

参数

| 参数 | 全名 | 描述 |
|---|---|---|
| help | n/a | 显示用法信息。 |

示例

根据提示输入需要的信息，为字段创建索引。

```
milvus_cli > create index

Collection name (car, car2): car2

The name of the field to create an index for (vector): vector

Index type (FLAT, IVF_FLAT, IVF_SQ8, IVF_PQ, RNSG, HNSW, ANNOY): IVF_FLAT

Index metric type (L2, IP, HAMMING, TANIMOTO): L2

Index params nlist: 2

Timeout []:
```

delete alias

删除别名。

语法

```
delete alias -a (text) [-t (float)]
```

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| -a | alias-name | 别名。 |
| -t | timeout | （可选）允许的远程过程调用的最大持续时间 (以秒为单位)。如果不传递此参数，客户端将一直等待直到服务器响 |
| help | n/a | 显示用法信息。 |

delete collection

删除集合。

语法

```
delete collection -c (text) [-t (float)]
```

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| -c | collection | 待删除集合的名称。 |
| -t | timeout | （可选）允许的远程过程调用的最大持续时间 (以秒为单位)。如果不传递此参数，客户端将一直等待直到服务器响应 |
| help | n/a | 显示用法信息。 |

示例

```
milvus_cli > delete collection -c car
```

delete entities

删除实体。

语法

```
delete entities -c (text) [-p (text)] [-t (float)]
```

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| -c | collection | 集合的名称。 |
| -p | partition | （可选）实体所属的分区的名称。 |
| -t | timeout | （可选）允许的远程过程调用的最大持续时间 (以秒为单位)。如果不传递此参数，客户端将一直等待直到服务器响应 |
| help | n/a | 显示用法信息。 |

示例

```
milvus_cli > delete entities -c car

The expression to specify entities to be deleted, such as "film_id in [0, 1 ]": film_id in [ 0, 1 ]

You are trying to delete the entities of collection. This action cannot be undone!

Do you want to continue? [y/N]: y
```

delete partition

删除分区。

语法

```
delete partition -c (text) -p (text) [-t (float)]
```

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| -c | collection | 待删除分区所属集合的名称。 |
| -t | timeout | （可选）允许的远程过程调用的最大持续时间 (以秒为单位)。如果不传递此参数，客户端将一直等待直到服务器响应 |
| -p | partition | 待删除分区的名称。 |
| help | n/a | 显示用法信息。 |

示例

```
milvus_cli > delete partition -c car -p new_partition
```

delete index

删除索引和相应的索引文件。

目前，一个集合最多支持一个索引。

语法

```
delete index -c (text) [-t (float)]
```

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| -c | collection | 集合的名称。 |
| -t | timeout | （可选）允许的远程过程调用的最大持续时间 (以秒为单位)。如果不传递此参数，客户端将一直等待直到服务器响应 |
| help | n/a | 显示用法信息。 |

示例

```
milvus_cli > delete index -c car
```

describe collection

显示集合的详细信息。

语法

```
describe collection -c (text)
```

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| -c | collection | 集合的名称。 |
| help | n/a | 显示用法信息。 |

示例

```
milvus_cli > describe collection -c test_collection_insert
```

describe partition

显示分区的详细信息。

语法

```
describe partition -c (text) -p (text)
```

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| -c | collection | 分区所属的集合的名称。 |
| -p | partition | 分区的名称。 |
| help | n/a | 显示用法信息。 |

示例

```
milvus_cli > describe partition -c test_collection_insert -p _default
```

describe index

显示索引的详细信息。

目前，一个集合最多支持一个索引。

语法

```
describe index -c (text)
```

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| -c | collection | 集合的名称。 |
| help | n/a | 显示用法信息。 |

exit

关闭命令行窗口。

语法

```
exit
```

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| help | n/a | 显示用法信息。 |

help

显示命令的用法信息。

语法

```
help <command>
```

命令

| 命令 | 描述 |
| --- | --- |
| calc | 计算两个向量数组之间的距离、mkts_from_hybridts、mkts_from_unixtime、或 hybridts_to_unixtime。 |
| clear | 清除屏幕。 |
| connect | 连接到 Milvus。 |
| create | 创建集合、分区、索引或别名。 |
| delete | 删除集合、分区、索引、实体或别名。 |
| describe | 描述集合、分区或索引。 |
| exit | 关闭命令行窗口。 |
| help | 显示命令的用法信息。 |
| import | 将数据导入分区。 |
| list | 列出集合、分区或索引。 |
| load | 加载集合或分区。 |
| load_balance | 在查询节点上执行负载均衡。 |
| query | 显示与您输入的所有条件匹配的查询结果。 |
| release | 释放集合或分区。 |
| search | 执行向量相似搜索或混合搜索。 |
| show | 显示当前集合、实体加载的进度、实体索引的进度或段信息。 |
| version | 显示 Milvus_CLI 的版本信息。 |

import

将本地或者远程数据导入分区。

语法

```
import -c (text)[-p (text)][-t (float)] <file_path>
```

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| -c | collection | 将数据插入到的集合的名称。 |
| -p | partition | （可选）将数据插入到的分区的名称。不传递这个参数表示选择“_default”分区。 |
| -t | timeout | （可选）允许的远程过程调用的最大持续时间 (以秒为单位)。如果不传递此参数，客户端将一直等待直到服务器响应 |
| help | n/a | 显示用法信息。 |

示例 1

以导入本地 CSV 文件为例。

```
milvus_cli > import -c car 'examples/import_csv/vectors.csv'

Reading csv file...  [####################################]  100%

Column names are ['vector', 'color', 'brand']

Processed 50001 lines.

Insert successfully.
   -------------------------  ------------------   Total insert entities:                  50000   Total
```

示例 2

以导入远程 CSV 文件为例。

```
milvus_cli > import -c car 'https://raw.githubusercontent.com/milvus-
io/milvus_cli/main/examples/import_csv/vectors.csv'
```

```
Reading file from remote URL.

Reading csv file...  [###################################]  100%

Column names are ['vector', 'color', 'brand']

Processed 50001 lines.

Inserting ...

Insert successfully.

-------------------------  ------------------
Total insert entities:                  50000
Total collection entities:             150000
Milvus timestamp:           428849214449254403
-------------------------  ------------------
```

list collections

列出所有的集合。

语法

```
list collections [-t (float)][-l (boolean)]
```

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| -t | timeout | （可选）允许的远程过程调用的最大持续时间 (以秒为单位)。如果不传递此参数，客户端将一直等待直到服务器 |
| -l | show-loaded | （可选）仅显示已加载的集合。 |
| help | n/a | 显示用法信息。 |

list indexes

列出集合的所有索引。

目前，一个集合最多支持一个索引。

语法

```
list indexes -c (text)
```

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| -c | collection | 集合的名称。 |
| help | n/a | 显示用法信息。 |

list partitions

列出集合的所有分区。

语法

```
list partitions -c (text)
```

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| -c | collection | 集合的名称。 |
| help | n/a | 显示用法信息。 |

load

将集合或分区从硬盘加载到内存中。

语法

```
load -c (text) [-p (text)]
```

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| -c | collection | 分区所属的集合的名称。 |
| -p | partition | (可选/多个) 分区的名称。 |
| help | n/a | 显示用法信息。 |

load_balance

通过从源查询节点向目的查询节点转移 segment 实现负载均衡。

语法

```
load_balance -s (int) -d (int) -ss (int) [-t (int)]
```

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| -s | src-node-id | 要均衡的源查询节点 ID。 |
| -d | dst-node-id | （多个）要转移 segment 的目的查询节点 ID。 |
| -ss | sealed-segment-ids | （多个）要转移的密封 segment 的 ID。 |
| -t | timeout | （可选）超时时间，单位为秒。 |
| help | n/a | 显示用法信息。 |

query

显示与您输入的所有条件匹配的查询结果。

语法

```
query
```

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| help | n/a | 显示用法信息。 |

示例

示例 1

根据提示输入需要的信息，执行查询。

```
milvus_cli > query

Collection name: car

The query expression: id in [ 428960801420883491, 428960801420883492,
428960801420883493 ]

Name of partitions that contain entities(split by "," if multiple) []:
default

A list of fields to return(split by "," if multiple) []: color, brand

timeout []:

Guarantee timestamp. This instructs Milvus to see all operations performed before a provided timestamp. If
Graceful time. Only used in bounded consistency level. If graceful_time is set, PyMilvus will use current
Travel timestamp. Users can specify a timestamp in a search to get results based on a data view at a speci
```

示例 2

根据提示输入需要的信息，执行查询。

```
milvus_cli > query

Collection name: car

The query expression: id > 428960801420883491

Name of partitions that contain entities(split by "," if multiple) []:
default

A list of fields to return(split by "," if multiple) []: id, color,
brand

timeout []:

Guarantee timestamp. This instructs Milvus to see all operations performed before a provided timestamp. If
Graceful time. Only used in bounded consistency level. If graceful_time is set, PyMilvus will use current
Travel timestamp. Users can specify a timestamp in a search to get results based on a data view at a speci
```

release

从内存释放集合或分区。

语法

```
release -c (text) [-p (text)]
```

参数

| 参数 | 全名 | 描述 |
|---|---|---|
| -c | collection | 分区所属的集合的名称。 |
| -p | partition | (多选/多个) 分区的名称。 |
| help | n/a | 显示用法信息。 |

search

执行向量相似搜索或混合搜索。

语法

search

参数

| 参数 | 全名 | 描述 |
| --- | --- | --- |
| help | n/a | 显示用法信息。 |

示例

示例 1

根据提示输入需要的信息，在 CSV 文件上执行搜索。

```
milvus_cli > search

Collection name (car, test_collection): car

The vectors of search data(the length of data is number of query (nq), the dim of every vector in data mus
out headers): examples/import_csv/search_vectors.csv

The vector field used to search of collection (vector): vector

Metric type: L2

Search parameter nprobe's value: 10

The max number of returned record, also known as topk: 2

The boolean expression used to filter attribute []: id > 0

The names of partitions to search (split by "," if multiple) ['_default'] []: _default

timeout []:

Guarantee Timestamp(It instructs Milvus to see all operations performed before a provided timestamp. If no

Travel Timestamp(Specify a timestamp in a search to get results based on a data view) [0]:
```

示例 2

根据提示输入需要的信息，在被索引的集合上执行搜索。

```
milvus_cli > search

Collection name (car, test_collection): car

The vectors of search data(the length of data is number of query (nq), the dim of every vector in data mus
    [[0.71, 0.76, 0.17, 0.13, 0.42, 0.07, 0.15, 0.67, 0.58, 0.02, 0.39, 0.47, 0.58, 0.88, 0.73, 0.31, 0.23

The vector field used to search of collection (vector): vector

Metric type: L2

Search parameter nprobe's value: 10

The specified number of decimal places of returned distance [-1]: 5

The max number of returned record, also known as topk: 2
```

```
The boolean expression used to filter attribute []: id > 0

The names of partitions to search (split by "," if multiple) ['_default'] []: _default

timeout []:

Guarantee Timestamp(It instructs Milvus to see all operations performed before a provided timestamp. If no

Travel Timestamp(Specify a timestamp in a search to get results based on a data view) [0]:
```

示例 3

根据提示输入需要的信息，在未被索引的集合上执行搜索。

```
milvus_cli > search

Collection name (car, car2): car

The vectors of search data(the length of data is number of query (nq), the dim of every vector in data mus

The vector field used to search of collection (vector): vector

The specified number of decimal places of returned distance [-1]: 5

The max number of returned record, also known as topk: 2

The boolean expression used to filter attribute []:

The names of partitions to search (split by "," if multiple) ['_default'] []:

timeout []:

Guarantee Timestamp(It instructs Milvus to see all operations performed before a provided timestamp. If no

Travel Timestamp(Specify a timestamp in a search to get results based on a data view) [0]:
```

show connection

显示当前连接。

语法

```
show connection [-a]
```

参数

| 参数 | 全名 | 描述 |
|---|---|---|
| -a | all | （可选）显示所有连接的开关。 |
| help | n/a | 显示用法信息。 |

show index_progress

显示为实体索引的进度。

语法

```
show index_progress -c (text) [-i (text)]
```

参数

| 参数 | 全名 | 描述 |
|---|---|---|
| -c | collection | 实体所属的集合的名称。 |
| -i | index | （可选）索引的名称。 |
| help | n/a | 显示用法信息。 |

show loading_progress

显示实体加载的进度。

语法

```
show loading_progress -c (text) [-p (text)]
```

参数

| 参数 | 全名 | 描述 |
|---|---|---|
| -c | collection | 实体所属的集合的名称。 |
| -p | partition | （可选/多个）加载分区的名称。 |
| help | n/a | 显示用法信息。 |

show query_segment

显示集合的段信息。

语法

```
show query_segment -c (text) [-t (float)]
```

参数

| 参数 | 全名 | 描述 |
|---|---|---|
| -c | collection-name | 集合的名称。 |
| -t | timeout | （可选）允许的远程过程调用的最大持续时间 (以秒为单位)。如果不传递此参数，客户端将一直等待直到服务 |
| help | n/a | 显示用法信息。 |

version

显示 Milvus_CLI 的版本信息。

语法

```
version
```

参数

| 参数 | 全名 | 描述 |
|---|---|---|
| help | n/a | 显示用法信息。 |

你可以直接在终端中检查 Milvus_CLI 的版本，示例如下。在本例中，milvus_CLI——version 作为命令。

示例

```
$ milvus_cli --version
Milvus_CLI v0.1.7
```

## MilvusDM

MilvusDM 简介

MilvusDM 是一款针对 Milvus 研发的开源数据迁移工具，支持 Milvus 数据传输以及数据文件的导入与导出。开发者使用 MilvusDM 可以提升数据管理效率，降低运维成本。MilvusDM 可以通过指定 Milvus 中的集合或分区，帮助用户更智能地迁移所需数据。

MilvusDM 目前支持以下数据传输通道：

- Milvus 至 Milvus: 支持 Milvus 实例之间的数据迁移
- Faiss 至 Milvus: 将未压缩的 Faiss 文件导入 Milvus
- HDF5 至 Milvus: 将 HDF5 格式的文件导入 Milvus
- Milvus 至 HDF5: 将 Milvus 数据批量备份为 HDF5 格式的本地文件

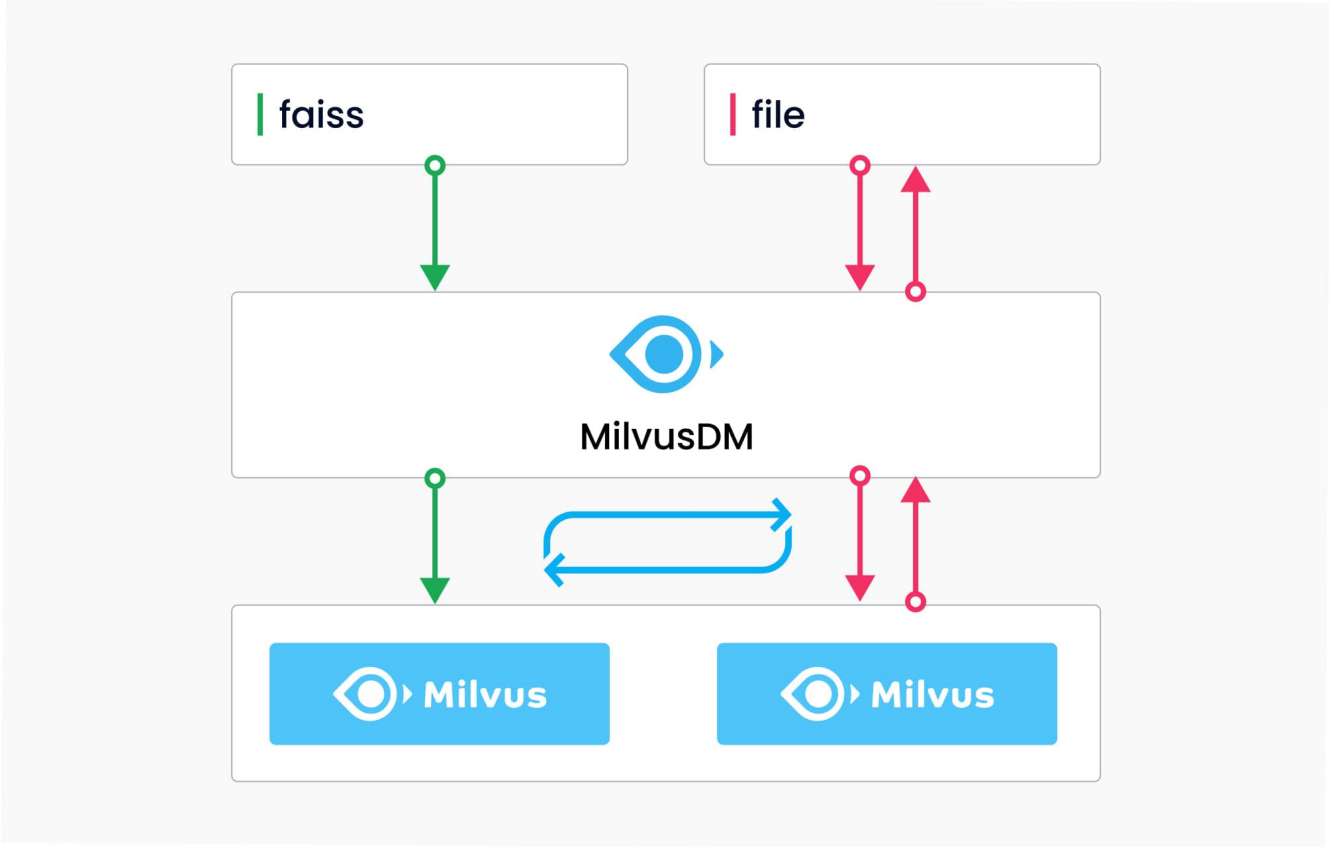



Figure 6: MilvusDM

如需安装 MilvusDM，运行：

```
pip3 install pymilvusdm
```

MilvusDM 文件结构

使用 MilvusDM 时会根据传入的.yaml 文件执行对应的任务，如下图所示：

MilvusDM 文件结构：

- pymilvusdm
  - core
    - milvus_client.py：执行 Milvus 客户端相关的操作
    - read_data.py：读取本地 HDF5 格式的数据文件（如果有读取其他文件格式的需求，可在此处添加代码）
    - read_faiss_data.py：读取 Faiss 的数据文件

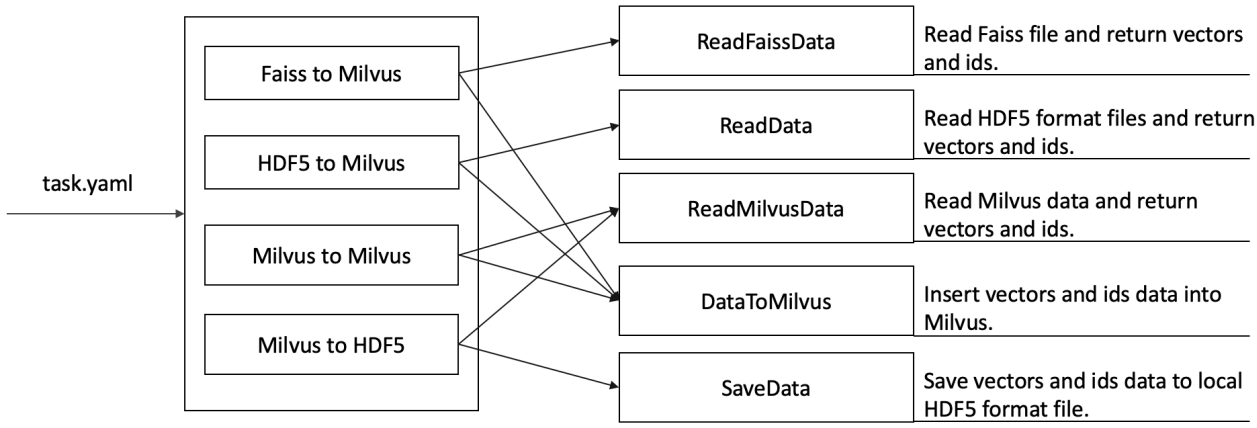



Figure 7: File structure

    * read_milvus_data.py：读取 Milvus 的数据文件
    * read_milvus_meta.py：读取 Milvus 的元数据
    * data_to_milvus.py：根据.yaml 文件配置参数，建立集合或分区，并将向量和对应的 ID 导入 Milvus
    * save_data.py：将读取到的数据保存为 HDF5 格式的文件
    * write_logs.py：在执行操作时写 `debug/info/error` 日志
  - faiss_to_milvus.py：实现将 Faiss 文件数据导入 Milvus
  - hdf5_to_milvus.py：实现将 HDF5 格式的文件数据导入 Milvus
  - milvus_to_milvus.py：实现将 Milvus 实例间的数据迁移
  - milvus_to_hdf5.py：实现将 Milvus 的数据导出为 HDF5 格式的文件
  - main.py：根据.yaml 文件执行相关任务
  - setting.py：执行代码时的相关配置参数
- setup.py：将 pymilvusdm 打包并上传到 PyPI（Python Package Index）

开发计划

我们计划在下个版本的 MilvusDM 中添加以下功能:

- 支持将 Faiss 的 binary 数据文件导入 Milvus
- Milvus 至 Milvus 时支持指定黑白名单
- Milvus 至 Milvus 时支持将多个集合或分区的数据合并导入至一个集合中
- 支持 Milvus 数据备份和数据恢复 我们十分欢迎大家为开源项目 MilvusDM 贡献代码。你可以通过代码文件结构了解 MilvusDM 工具的设计构思。如有新的数据迁移需求，你还可以通过修改源码，为社区贡献代码。

MilvusDM 在 GitHub 上开源。

欢迎贡献代码 ✍，也请给本项目点 star ☆

安装 MilvusDM

MilvusDM 是一款针对 Milvus 研发的开源数据迁移工具，支持 Milvus 数据传输以及数据文件的导入与导出。本文将介绍如何安装 MilvusDM。

pymilvusdm2.0 可用于将数据从 Milvus(0.10.x or 1.x) 版本 迁移到 Milvus2.x 版本。

安装前提

在安装 MilvusDM 之前请确保你的操作系统和软件符合以下要求。

| 操作系统 | 版本 |
| --- | --- |
| CentOS | 7.5 或以上版本 |
| Ubuntu LTS | 18.04 或以上版本 |

| 软件 | 版本 |
| --- | --- |
| Milvus | 0.10.x or 1.x or 2.x |
| Python3 | 3.7 或以上版本 |
| pip3 | 和 Python 版本对应. |

安装 MilvusDM

1. 将如下内容添加到 ~/.bashrc :

```
export MILVUSDM_PATH='/home/$user/milvusdm'
export LOGS_NUM=0
```

- `MILVUSDM_PATH`：该变量定义了 MilvusDM 工作目录，用于保存 MilvusDM 生成的日志和数据。该变量的值默认为 `/home/$user/milvusdm`。
- `LOGS_NUM`：MilvusDM 每日生成一个日志文件。使用该变量定义需要保存的日志文件数量。该变量的默认值为 0，所有日志都会保存。

2. 配置环境变量：

```
$ source ~/.bashrc
```

3. 使用 `pip` 安装 MilvusDM:

```
$ pip3 install pymilvusdm==2.0
```

## Attu

简介

Attu

Attu 是 Milvus 的高效开源管理工具. 它具有直观的图形用户界面 (GUI)，可让您轻松与数据库进行交互。只需点击几下，您就可以可视化集群状态、管理元数据、执行数据查询等等。

Attu 是 Zilliz 下属的开源项目。

Attu_overview

特点

Attu 正在快速开发中，每周都会添加新功能。每当新功能准备就绪时，我们都会发布新版本。

以下是我们必须提供的功能：

- Milvus 集群统计信息一目了然。

view_cluster_statistics

- 以简单直接的方式浏览、查询和管理集合。

manage_collections

- 只需单击几下即可执行 CRUD 或批量操作。

attu_operations

- 创建矢量索引。

attu_create_index

- 以全新的方式进行向量相似度搜索。

attu_conduct_search

- 新的代码模式为您提供更好的用户体验。

code_mode

阅读安装 Attu 以了解更多。

贡献

Attu 是一个开源项目。欢迎所有贡献者。在提交代码前，请阅读我们的贡献指南。

如果您发现错误或想要请求新功能，请创建一个 GitHub Issue，并确保其他人没有创建相同的问题。

安装 Attu

这篇文章将描述如何安装 Attu - 一个开源的 MIlvus 管理工具。

Docker Compose 安装 Helm 安装安装包安装

先决条件

- 已安装 Milvus 单机版 或者 分布式版。
- Docker 版本 19.03 或者更新的版本。

Attu 只支持 Milvus 2.x。

运行 Attu

```
docker run -p 8000:3000 -e HOST_URL=http://{ your machine IP }:8000 -e MILVUS_URL={your machine IP}:19530 
```

一旦你成功运行了 Attu docker, 在浏览器输入 `http://{ your machine IP }:8000`，然后点击 Connect 按钮连接 Milvus。

Attu_install

欢迎成为贡献者

Attu 是一个开源项目。所有人都欢迎成为贡献者。在提交代码前，可以参考贡献导读。

如果你发现一个 bug 或者想添加新功能，请创建一个 GitHub Issue，同时请确认仓库中不存在类似的 issue。

Attu Overview 页面

这个篇文章将描述 Attu 的首页。

Attu 主要有 Overview 页面、Collection 页面、Vector Search 页面、和 System View 页面，对应左侧导航栏的四个图标。

Overview 页面主要列出了以下信息:

1. 所有加载到内存的 collection 数量
2. 所有 collection 数量
3. 所有导入到 Milvus 的数据
4. Loaded For Search 卡片：一个交互式快捷面板，允许你执行向量搜索或者释放 Collection

Attu Overview

管理 Collection

使用 Attu 管理 Collections

这片文章将会描述如何使用 Attu 管理 collections。

创建 collection

1. 点击左侧导航的 Collection 标签，然后点击 Create Collection。如下图所示，将会出现 Create Collection 对话框。

Create Collection dialog box

2. 填写相应表单，这个例子将创建名为 test 的 collection，这个 collection 有一个主键字段，一个向量字段，一个标量字段。你也可以继续添加需要的标量字段。

Create Collection dialog box

3. 点击 Create 就可以创建我们第一个 collection。

Create Collection dialog box

删除 collection

1. 在表格里，选中需要删除的 collection。
2. 点击 Trash 图标，如下图所示，将会出现 Delete Collection 对话框。
3. 输入 `delete` 以确认删除操作。
4. 点击 Delete 将会删除选中的所有 collection。

删除操作是不可回滚的。

Delete Collection dialog box

加载 collection

1. 鼠标悬浮在需要加载的 collection 行上，Load 图标将会出现在该行末尾。

Load Collection

2. 点击 Load 图标，将会出现 Load Collection 对话框。
3. 点击对话框中的 Load 按钮。

Load Collection

4. 加载 collection 数据到缓存中可能需要一些时间。当加载完成后，Status 列将会显示 Loaded For Search。

Load Collection

释放 collection

1. 鼠标悬浮在需要加载的 collection 行上，Release 图标将会出现在该行末尾。

Release Collection

2. 点击 Release 图标，将会出现 Release Collection 对话框。
3. 点击对话框中的 Release 按钮以释放 collection。
4. 如果释放成功，Status 列将显示 Unloaded。

Release Collection

查看 collection 的 schema

1. 点击数据行的 collection name，跳转到 collection 详情页面。
2. 在详情页点击 Schema，就可以看到 schema 的相关信息。

schema 的属性包含：

- Field Name
- Field Type
- Dimension (适用于向量字段)
- Index Type (适用于向量字段)
- Index Parameters (适用于向量字段)
- Description

Collection Schema

管理 Partition

使用 Attu 管理 Partitions

这个篇文章将描述 Attu 如何管理 partition。

Milvus 会在创建 collection 成功以后，自动创建一个默认的，不可删除的 partition。

创建 partition

1. 在 Collection 详情页面，点击 Partitions 页签。
2. 在 Partitions 页签，点击 Create Partition ，如下图所示，将会出现 Create Partition 对话框。
3. 在 Create Partition 对话框中，在 Name 输入框内，输入新的 partition 名称。
4. 点击 Create 按钮，创建一个 partition。

Create Partition

如果创建成功，新的 partition 会出现在 Partitions 页面中。

Create Partition

这样就可以选择任意一个 partition 来存储数据。

删除 partition

1. 选择需要删除的 partition。
2. 点击 Trash 图标，如下图所示，将会出现 Delete Partition 对话框。
3. 输入 `delete` 确认删除操作。
4. 点击 Delete 删除 partition。

Delete Partition

管理数据

使用 Attu 管理数据

这片文章将会描述如何使用 Attu 管理数据。

导入数据

这个例子将导入 20,000 行数据。导入数据是追加数据，而不是覆盖数据。

1. 在 Collection 页面，点击 Import Data 按钮。如下图所示，将会出现 Import Data 对话框。

Import Data

2. 在 Collection 下拉列表框中，选择需要的 collection 来导入数据。
3. 在 Partition 下拉列表框中，选择需要的 partition 来导入数据。
4. 点击 Choose CSV File 选择 CSV 文件。

需要保证 CSV 文件符合以下条件:

列名与 collection 中 schema 的字段名完全一致。

上传的 CSV 文件需要小于 150MB，以及行的总数小于 100,000。

5. 上传了一个合法的 CSV 文件以后，便可以点击 Next。

Import Data

6. 在新的对话框中，你可以选择在下拉列表框中选择列名。

我们推荐在 CSV 文件的首行设置你的列名。

Import Data

7. 在确认完列名与 collection 的字段一致后，点击 Import Data 将会导入 CSV 文件中的数据至 Milvus。导入数据可能需要一段时间。

Import Data

8. 导入成功后，collection 的 Entity Count 列将会更新为导入数据的总数。同时在 Partition 页面，导入数据时选择的 partition 的 Entity Count 列也会相应的更新。这个更新可能会需要一些时间。

Import Data

导出数据

1. 在 Collection 页面，点击 Data Query 页签。进入 Data Query 页面，输入你需要的 query 条件，然后点击 Query 就能得到符合条件的所有数据。
2. 点击 Download 图标，将会下载这些数据, 作为 CSV 文件保存到本地。

Export Data

管理索引

使用 Attu 管理索引

这片文章将会描述如何使用 Attu 管理索引。

创建索引

这个例子将会创建 IVF_FLAT 索引，使用 Euclidean 距离作为相似性度量，以及 `nlist` 值为 `1024`。

1. 在 Collection 页面，点击 Schema 页签。然后在 Schema 页面，点击 CREATE INDEX，将会出现 Create Index 对话框。
2. 在 Create Index 对话框中，在 Index Type 下拉列表框中，选择 IVF_FLAT，在 Metric Type 下拉列表框中，选择 L2，最后在 `nlist` 输入框中输入 `1024`。
3. 你可以点击 View Code 开关，将会看到 Code View 页面。你可以在这个页面中看到相应的 Python 或者 Node.js 代码。
4. 点击 Create 创建索引。

如果创建成功，你创建的索引类型将会出现在向量字段的 Index Type 列中。

Create Index

Create Index

删除索引

1. 点击 Index Type 列中的 Trash 图标，将会出现 Delete Index 对话框。
2. 输入 `delete` 来确认删除操作，然后点击 Delete 来删除索引。

如果删除成功，CREATE INDEX 按钮将会出现在 Index Type 列中。

Delete Index

结构化匹配

使用 Attu 获取数据

这个篇文章将描述 Attu 如何获取数据。

获取数据

1. 点击需要获取数据的 collection，就可以跳转到相应到详情页面。
2. 在 Data Query 页面，点击 Filter 标签，将会出现 Advanced Filter 对话框。
3. 通过表单，选择你需要的筛选条件，例如 color > 10 && color < 20。点击 Apply 确认筛选条件。

Query Data

1. 点击 Query，获取所有符合条件的数据。

Query Data

删除数据

1. 选择需要删除的数据，然后点击 Trash 图标。
2. 在 Delete entities 对话框中，输入 `delete` 确认删除操作。
3. 点击 Delete 删除选中的数据。

Delete Data

删除成功后，通过 Query 获取已删除的数据，结果将不会包含已删除的数据。

Delete Data

搜索数据

使用 Attu 进行向量搜索

这个篇文章将描述 Attu 如何进行向量搜索。

向量相似度搜索

在常规向量相似度搜索的基础上，还可以进行时间旅行搜索的混合搜索。

加载 collection 到缓存中

Milvus 内的所有 CRUD 操作都在内存中执行。在进行向量相似性搜索之前，需要将 collection 加载到内存中。查看加载 collection 了解更多。

Search Data

设置搜索参数

1. 在 Choose collection and field 区域的下拉列表框中选择要搜索的 collection 和向量字段。
2. 在 Enter vector value 字段中，输入一个向量，其维度与要搜索的目标向量具有相同的所选字段维度。
3. 在 Set search parameters 区域，指定索引的特定参数和其他与搜索相关的参数。

Search Data

带有高级过滤器的混合搜索（可选）

单击 Advanced Filter，将出现 Advanced Filter 对话框。您可以使用 AND 或 OR 运算符将多个条件组合成一个复合条件。过滤器表达式会随着条件的任何更改而自动更新。查看 boolean expression rule 了解更多。

Search Data

时间旅行搜索（可选）

Milvus 为所有数据插入和删除操作维护一个时间线。它允许用户在向量搜索中，指定时间戳以检索指定时间点的数据视图。

1. 单击 Time Travel，然后在出现的对话框中选择一个时间点。

Search Data

2. 在 TopK 下拉列表框中，指定要返回的搜索结果数。

3. 单击 Search 根据搜索条件检索最相似的向量。

Search Data Search Data

系统状态

使用 Attu 监控系统状态

此话题讨论如何使用 Attu 监控 Milvus 系统。

系统视图

点击左侧导航面板上的系统视图图标进入系统视图页面。

系统视图

系统视图仪表盘由以下卡片组成:

- 磁盘：展示存储空间占用情况。
- 内存：展示内存占用情况。
- Qps：展示最近 10 次 QPS。每个节点代表一个时间节点。鼠标悬浮上节点可以查看时间节点上 QPS 详情。
- 延迟: 展示最近 10 次延迟。每个节点代表一个时间节点。鼠标悬浮上节点可以查看时间节点上延迟详情。

系统视图 系统视图

- 拓扑图：展示当前运行 Milvus 实例系统结构。点击 Milvus 节点或者 Coordinator 节点后，右侧信息卡片会展示选中节点相关信息。
- 信息：展示选中节点的硬件，系统，配置信息。

节点列表视图

所有被同一父层 Coordinator 节点下管理的子节点都将出现在列表中。子节点可以通过 CPU 核心数、CPU 内核使用、磁盘使用及内存使用衡量指标排序。

节点列表视图

列表右侧是一张迷你版的拓扑图展示算中节点和父层 Coordinator 节点关系。迷你拓扑图下方是相关信息卡片。

点击向下箭头收起节点列表视图。

Attu FAQ

Attu 为什么报网络错误?

答:请确认在执行 `docker run` 命令时,传入了正确的 `HOST_URL` 值。你也可以在浏览器输入 `{HOST_URL}/api/v1/healthy` 来确认 Attu 的服务状态。

为什么 Attu 连接不上 Milvus?

答：请确保 Milvus 和 Attu 在同一网络。

我该如何在 k8s 中使用 Attu?

答：请参考使用 Helm Chart 安装 Attu。

# 参考手册

## 系统架构

概述

Milvus 系统架构概述

Milvus 是一款云原生向量数据库，它具备高可用、高性能、易拓展的特点，用于海量向量数据的实时召回。本文主要描绘了 Milvus 的宏观设计，包括数据模型、架构设计以及关键路径。

Milvus 基于 FAISS、Annoy、HNSW 等向量搜索库构建，核心是解决稠密向量相似度检索的问题。建议在阅读本文前，先了解向量检索的 基本概念。

在向量检索库的基础上，Milvus 支持数据分区分片、数据持久化、增量数据摄取、标量向量混合查询、time travel 等功能，同时大幅优化了向量检索的性能，可满足任何向量检索场景的应用需求。我们推荐用户使用 Kubernetes 部署 Milvus，以获得最佳可用性和弹性。

Milvus 采用共享存储架构，存储计算完全分离，计算节点支持横向扩展。

从架构上来看，Milvus 遵循数据流和控制流分离，整体分为了 四个层次，分别为接入层（access layer）、协调服务（coordinator service）、执行节点（worker node）和存储层（storage）。各个层次相互独立，独立扩展和容灾。

Architecture_diagram

更多 Milvus 架构细节，参考 存储计算分离 以及 主要组件。

存储计算分离

存储/计算分离

从架构上来看，Milvus 遵循数据流和控制流分离，整体分为了四个层次，分别为接入层（access layer）、协调服务（coordinator service）、执行节点（worker node）和存储层（storage）。各个层次相互独立，独立扩展和容灾。

接入层

接入层由一组无状态 proxy 组成，是整个系统的门面，对外提供用户连接的 endpoint。接入层负责验证客户端请求并减少返回结果。

- Proxy 本身是无状态的，一般通过负载均衡组件（Nginx、Kubernetes Ingress、NodePort、LVS）对外提供统一的访问地址并提供服务。
- 由于 Milvus 采用大规模并行处理（MPP）架构，proxy 会先对执行节点返回的中间结果进行全局聚合和后处理后再返回至客户端。

协调服务

协调服务是系统的大脑，负责向执行节点分配任务。它承担的任务包括集群拓扑节点管理、负载均衡、时间戳生成、数据声明和数据管理等。

协调服务共有四种角色:

Root coordinator（root coord）

负责处理数据定义语言（DDL）和数据控制语言（DCL）请求，比如创建或删除 collection、partition、index 等，同时负责维护中心授时服务 TSO 和时间窗口的推进。

Query coordinator (query coord)

负责管理 query node 的拓扑结构和负载均衡以及 growing segment 到 sealed segment 的切换流程（handoff）。

Data coordinator (data coord)

负责管理 data node 的拓扑结构，维护数据的元信息以及触发 flush、compact 等后台数据操作。

Index coordinator (index coord)

负责管理 index node 的拓扑结构，构建索引构和维护索引元信息。

执行节点

执行节点是系统的四肢，负责完成协调服务下发的指令和 proxy 发起的数据操作语言（DML）命令。由于采取了存储计算分离，执行节点是无状态的，可以配合 Kubernetes 快速实现扩缩容和故障恢复。执行节点分为三种角色：

Query node

Query node 通过订阅消息存储（log broker）获取增量日志数据并转化为 growing segment，基于对象存储加载历史数据，提供标量 + 向量的混合查询和搜索功能。

Data node

Data node 通过订阅消息存储获取增量日志数据，处理更改请求，并将日志数据打包存储在对象存储上实现日志快照持久化。

Index node

Index node 负责执行索引构建任务。Index node 不需要常驻于内存，可以通过 serverless 的模式实现。

存储服务

存储服务是系统的骨骼，负责 Milvus 数据的持久化，分为元数据存储（meta store）、消息存储（log broker）和对象存储（object storage）三个部分。

元数据存储

负责存储元信息的快照，比如 collection schema 信息、节点状态信息、消息消费的 checkpoint 等。元信息存储需要极高的可用性、强一致和事务支持，因此 etcd 是这个场景下的不二选择。除此之外，etcd 还承担了服务注册和健康检查的职责。

对象存储

负责存储日志的快照文件、标量/向量索引文件以及查询的中间处理结果。Milvus 采用 MinIO 作为对象存储，另外也支持部署于 AWS S3 和 Azure Blob 这两大最广泛使用的低成本存储。但是由于对象存储访问延迟较高，且需要按照查询计费，因此 Milvus 未来计划支持基于内存或 SSD 的缓存池，通过冷热分离的方式提升性能以降低成本。

消息存储

消息存储是一套支持回放的发布订阅系统，用于持久化流式写入的数据，以及可靠的异步执行查询、事件通知和结果返回。执行节点宕机恢复时，通过回放消息存储保证增量数据的完整性。目前分布式 Milvus 依赖 Pulsar 作为消息存储，Milvus standalone 依赖 RocksDB 作为消息存储。消息存储也可以替换为 Kafka、Pravega 等流式存储。

整个 Milvus 围绕日志为核心来设计，遵循日志即数据的准则，因此在 2.0 版本中没有维护物理上的表，而是通过日志持久化和日志快照来保证数据的可靠性。

Log_mechanism

日志系统作为系统的主干，承担了数据持久化和解耦的作用。通过日志的发布一订阅机制，Milvus 将系统的读、写组件解耦。一个极致简化的模型如上图所示，整个系统主要由两个角　构成，分别是消息存储（log broker）（负责维护“日志序列”）与“　志订阅者”。其中的“　志序列”记录了所有改变库表状态的操作，“日志订阅者”通过订阅日志序列更新本地数据，以只读副本的　式提供服务。发布一订阅机制的出现也给系统预留了很大的拓展空间，便于 change data capture（CDC）、全球部署等功能的拓展。

更多 Milvus 架构细节，参考 主要组件。

主要组件

Milvus 支持两种部署模式，单机模式（standalone）和分布式模式（cluster）。两种模式具备完全相同的能力，用户可以根据数据规模、访问量等因素选择适合自己的模式。Standalone 模式部署的 Milvus 暂时不支持在线升级为 cluster 模式。

单机版 Milvus

单机版 Milvus 包括三个组件:

- Milvus 负责提供系统的核心功能。
- Etcd 是元数据引擎，用于管理 Milvus 内部组件的元数据访问和存储，例如 proxy、index node 等。
- MinIO 是存储引擎，负责维护 Milvus 的数据持久化。

Standalone_architecture

分布式版 Milvus

分布式版 Milvus 由八个微服务组件和三个第三方依赖组成，每个微服务组件可使用 Kubernetes 独立部署。

微服务组件

- Root coord
- Proxy
- Query coord
- Query node
- Index coord
- Index node
- Data coord
- Data node

第三方依赖

- etcd 负责存储集群中各组件的元数据信息。
- MinIO 负责处理集群中大型文件的数据持久化，如索引文件和全二进制日志文件。
- Pulsar 负责管理近期更改操作的日志，输出流式日志及提供日志订阅服务。

Distributed_architecture

更多 Milvus 架构细节，参考 存储计算分离。

关键路径

本文介绍了 Milvus 系统中数据写入、索引构建以及数据查询的具体实现。

数据写入

用户可以为每个 collection 指定 shard 数量，每个 shard 对应一个虚拟通道 vchannel。如下图所示，在 log broker 内，每个 vchanel 被分配了一个对应的物理通道 pchannel。Proxy 基于主键哈希决定输入的增删请求进入哪个 shard。

由于没有复杂事务，DML 的检查与确认　作被提前至 proxy。对于所有的增删请求，proxy 会先通过请求位于 root coord 的 TSO 中心授时模块获取时间戳。这个时间戳决定了数据最终可见和相互覆盖的顺序。除了分配时间戳，proxy 也为每行数据分配全局唯一的 primary key。Primary key 以及 entity 所处的 segmentID 均从 data coord 批量获取，批量有助于提升系统的吞吐，降低 data coord 的负载。

Channels 1

除增删类操作之外，数据定义类操作也会写　　志序列。由于数据定义类操作出现的频率很低，系统只为其分配　路 channel。

Channels 2

Vchannel 由消息存储底层引擎的物理节点承担。不同 vchannel 可以被调度到不同的物理节点，但每个 channel 在物理上不再进 步拆分，因此单个 vchannel 不会跨多个物理节点。当 collection 写入出现瓶颈时，通常需要关注两个问题：一是 log broker 节点负载是否过高，需要扩容；二是 shard 是否足够多，保证每个 log broker 的负载足够均衡。

Write log sequence

上图总结了 志序列的写 过程中涉及的四个组件：proxy、log broker、data node 和对象存储。整体共四部分 作：DML 请求的检查与确认、日志序列的发布—订阅、流式 志到日志快照的转换、日志快照的持久化存储。在 Milvus 2.0 中，对这四部分 作进行了解耦，做到同类型节点之间的对等。面向不同的 库负载，特别是大规模 波动的流式负载，各环节的系统组件可以做到独 的弹性伸缩。

索引构建

构建索引的任务由 index node 执 。为了避免数据更新导致的索引频繁重复构建，Milvus 将 collection 分成了更 的粒度，即 segment，每个 segment 对应自己的独 的索引。

Index building

Milvus 可以对每个向量列、标量列和主键列构建索引。索引构建任务的输 与输出都是对象存储。Index node 拉取 segment 中需要构建索引的日志快照，在内存中进 数据与元信息的反序列化，构建索引。索引构建完成后，将索引结构序列化并写回对象存储。

对向量构建索引的过程属于计算密集、访存密集的负载类型，主要操作是向量运算与矩阵运算。由于被索引的数据维度过高，难以通过传统的树形结构进 高效索引。目前较为成熟的技术是基于聚类或图来表示 维稠密向量的近邻关系。无论哪种索引类型，都涉及到大规模向量数据的多次迭代计算，如寻找聚类、图遍历的收敛状态。

与传统的索引操作相比，向量计算需要充分利 SIMD 加速。 前，Milvus 内置的引擎 持 SSE、AVX2、AVX512 等 SIMD 指令。向量索引任务具备突发性、高资源消耗等特点，其弹性能力对于成本格外重要。未来 Milvus 会继续探索异构计算和 serverless 架构，持续优化索引构建的成本。

同时，Milvus 支持标量过滤和主键查询功能。为了实现高效率的标量查询，Milvus 构建了 Bloom filter index、hash index、tree index 和 inverted index。未来 Milvus 会逐渐完善索引类型，提供 bitmap index、rough index 等更多外部索引能力。

数据查询

数据查询指在一个指定 collection 中查找与目标向量最近邻的 k 个向量或满足距离范围的全部向量的过程。结果返回满足条件的向量及其对应的 primary key 和 field。

Data query

一个 collection 中的数据被分为多个 segment，query node 以 segment 为粒度加载索引。查询请求会广播到全部的 query node，所有 query node 并发执行查询。每个 query node 各自对本地的 segment 进行剪枝并搜索符合条件的数据，同时将各 segment 结果进行聚合返回。

上述过程中 query node 并不感知其他 query node 的存在，每个 query node 只需要完成两件任务：首先是响应 query coord 的调度，加载/卸载 segment；其次是根据本地的 segment 响应查询请求。Proxy 负责将每个 query node 返回的数据进行全局聚合返回给客户端。

Handoff

Query node 中的 segment 只存在两种状态——growing 和 sealed——分别对应增量数据和历史数据。对于 growing segment，query node 通过订阅 vchannel 获取数据的近期更新。当 data coord 已经 flush 完该 segment 的所有数据，会通知 query coord 进行 handoff 操作，将增量数据转换为历史数据。Sealed segment 的索引由 index node 构建完成后会被 query node 自动加载。对于 sealed segment，query coord 会综合考虑内存使用、CPU 开销、segment 数目等因素，尽可能均匀分配给所有的 query node。

## 系统配置

Milvus System Configurations Checklist

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic introduces the general sections of the system configurations in Milvus.

Milvus maintains a considerable number of parameters that configure the system. Each configuration has a default value, which can be used directly. You can modify these parameters flexibly so that Milvus can better serve your application. See Configure Milvus for more information.

In current release, all parameters take effect only after being configured at the startup of Milvus.

Sections

For the convenience of maintenance, Milvus classifies its configurations into 17 sections based on its components, dependencies, and general usage.

**etcd**

etcd is the metadata engine supporting Milvus' metadata storage and access.

Under this section, you can configure etcd endpoints, relevant key prefixes, etc.

See etcd-related Configurations for detailed description for each parameter under this section.

**minio**

Milvus supports MinIO and Amazon S3 as the storage engine for data persistence of insert log files and index files. Whereas MinIO is the de facto standard for S3 compatibility, you can configure S3 parameters directly under MinIO section.

Under this section, you can configure MinIO or S3 address, relevant access keys, etc.

See MinIO-related Configurations for detailed description for each parameter under this section.

**pulsar**

Pulsar is the underlying engine supporting Milvus cluster's reliable storage and publication/subscription of message streams.

Under this section, you can configure Pulsar address, the message size, etc.

See Pulsar-related Configurations for detailed description for each parameter under this section.

**rocksmq**

RocksMQ is the underlying engine supporting Milvus standalone's reliable storage and publication/subscription of message streams. It is implemented on the basis of RocksDB.

Under this section, you can configure message size, retention time and size, etc.

See RocksMQ-related Configurations for detailed description for each parameter under this section.

**rootCoord**

Root coordinator (root coord) handles data definition language (DDL) and data control language (DCL) requests, manages TSO (timestamp Oracle), and publishes time tick messages.

Under this section, you can configure root coord address, index building threshold, etc.

See Root Coordinator-related Configurations for detailed description for each parameter under this section.

`proxy`

Proxy is the access layer of the system and endpoint to users. It validates client requests and reduces the returned results.

Under this section, you can configure proxy port, system limits, etc.

See Proxy-related Configurations for detailed description for each parameter under this section.

`queryCoord`

Query coordinator (query coord) manages topology and load-balancing of the query nodes, and handoff operation from growing segments to sealed segments.

Under this section, you can configure query coord address, auto handoff, auto load-balancing, etc.

See Query coordinator-related Configurations for detailed description for each parameter under this section.

`queryNode`

Query node performs hybrid search of vector and scalar data on both incremental and historical data.

Under this section, you can configure query node port, graceful time, etc.

See Query Node-related Configurations for detailed description for each parameter under this section.

`indexCoord`

Index coordinator (index coord) manages topology of the index nodes, and maintains index metadata.

Under this section, you can configure index coord address, etc.

See Index Coordinator-related Configurations for detailed description for each parameter under this section.

`indexNode`

Index node builds indexes for vectors.

Under this section, you can configure index node port, etc.

See Index Node-related Configurations for detailed description for each parameter under this section.

`dataCoord`

Data coordinator (data coord) manages the topology of data nodes, maintains metadata, and triggers flush, compact, and other background data operations.

Under this section, you can configure data coord address, segment settings, compaction, garbage collection, etc.

See Data Coordinator-related Configurations for detailed description for each parameter under this section.

`dataNode`

Data node retrieves incremental log data by subscribing to the log broker, processes mutation requests, and packs log data into log snapshots and stores them in the object storage.

Under this section, you can configure data node port, etc.

See Data Node-related Configurations for detailed description for each parameter under this section.

`localStorage`

Milvus stores the vector data in local storage during search or query to avoid repetitive access to MinIO or S3 service.

Under this section, you can enable local storage, and configure the path, etc.

See Local Storage-related Configurations for detailed description for each parameter under this section.

`log`

Using Milvus generates a collection of logs. By default, Milvus uses logs to record information at debug or even higher level for standard output (stdout) and standard error (stderr).

Under this section, you can configure the system log output.

See Log-related Configurations for detailed description for each parameter under this section.

`msgChannel`

Under this section, you can configure the message channel name prefixes and component subscription name prefixes.

See Message Channel-related Configurations for detailed description for each parameter under this section.

`common`

Under this section, you can configure the default names of partition and index, and the Time Travel (data retention) span of Milvus.

See Common Configurations for detailed description for each parameter under this section.

`knowhere`

Knowhere is the search engine of Milvus.

Under this section, you can configure the default SIMD instruction set type of the system.

See Knowhere-related Configurations for detailed description for each parameter under this section.

Frequently used parameters

Below list some frequently used parameters categorized in accordance with the purposes of modification.

Performance tuning

The following parameters control the system behaviors that influence the performance of index creation and vector similarity search.

queryNode.gracefulTime

rootCoord.minSegmentSizeToEnableIndex

dataCoord.segment.maxSize

dataCoord.segment.sealProportion

dataNode.flush.insertBufSize

queryCoord.autoHandoff

queryCoord.autoBalance

localStorage.enabled

Data and metadata retention

The following parameters control the retention of data and metadata.

common.retentionDuration

rocksmq.retentionTimeInMinutes

dataCoord.enableCompaction

dataCoord.enableGarbageCollection

dataCoord.gc.dropTolerance

Administration

The following parameters control the log output and object storage access.

log.level

log.file.rootPath

log.file.maxAge

minio.accessKeyID

minio.secretAccessKey

What’s next

- Learn how to configure Milvus before installation.
- Learn more about the installation of Milvus:
  - Install Milvus Standalone
  - Install Milvus Cluster ## 距离计算方式

Milvus 基于不同的距离计算方式比较向量间的距离。选择合适的距离计算方式能极大地提高数据分类和聚类性能。

以下表格列出了 Milvus 目前支持的距离计算方式与数据格式、索引类型之间的兼容关系。

浮点型向量 二值型向量

距离计算方式

索引类型

欧氏距离 (L2)

内积 (IP)

FLAT

IVF_FLAT

IVF_SQ8

IVF_PQ

HNSW

IVF_HNSW

RHNSW_FLAT

RHNSW_SQ

RHNSW_PQ

ANNOY

距离计算方式

索引类型

杰卡德距离 (Jaccard) 谷本距离 (Tanimoto) 汉明距离 (Hamming)

BIN_FLAT

BIN_IVF_FLAT

超结构 (superstructure) 子结构 (substructure)

BIN_FLAT

欧氏距离（L2）

欧氏距离计算的是两点之间最短的直线距离。

欧氏距离的计算公式为：

$$d(\boldsymbol{a},\boldsymbol{b}) = d(\boldsymbol{b},\boldsymbol{a}) = \sqrt{\sum_{i=1}^{n}(b_i - a_i)^2}$$

Figure 8: euclidean

其中 a = (a1, a2,⋯, an) 和 b = (b1, b2,⋯, bn) 是 n 维欧氏空间中的两个点。

欧氏距离是最常用的距离计算方式之一，应用广泛，适合数据完整，数据量纲统一的场景。

内积（IP）

两条向量内积距离的计算公式为：

$$p(A,B) = A \cdot B = \sum_{i=1}^{n} a_i \times b_i$$

Figure 9: ip

假设有 A 和 B 两条向量，则 `||A||` 与 `||B||` 分别代表 A 和 B 归一化后的值。

内积更适合计算向量的方向而不是大小。

如需使用点积计算向量相似度，则必须对向量作归一化处理。处理后点积与余弦相似度等价。

假设 X’ 是向量 X 的归一化向量：

$$X' = (x'_1, x'_2, \ldots, x'_n), X' \in^n$$

Figure 10: normalize

两者之间的关系为：

杰卡德距离

杰卡德相似系数计算数据集之间的相似度，计算方式为：数据集交集的个数和并集个数的比值。计算公式可以表示为：

杰卡德距离是用来衡量两个数据集差异性的一种指标，被定义为 1 减去杰卡德相似系数。对于二值变量，杰卡德距离等价于谷本系数。

$$x'_i = \frac{x_i}{\| X \|} = \frac{x_i}{\sqrt{\sum_{i=1}^{n} (x_i)^2}}$$

Figure 11: normalization

$$J(A,B) = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Figure 12: Jaccard similarity coefficient

$$d_j(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

Figure 13: Jaccard distance

杰卡德距离适合字符串相似性度量。

谷本距离

对于二值变量，谷本距离公式可表示为：

$$d_t = \frac{A \cdot B}{|A|^2 + |B|^2 - A \cdot B}$$

Figure 14: tanimoto distance

在 Milvus 中，谷本距离仅支持二值变量。

值域从 0 到正无穷。

对于二值变量，谷本系数等价于杰卡德距离：

$$T(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

Figure 15: tanimoto coefficient

对于二值变量，谷本系数值域为 0 到 +1（+1 的相似度最高）

汉明距离

汉明距离计算二进制字符串之间的距离。两个等长字符串之间的汉明距离定义为将其中一个变为另外一个所需要作的最小替换次数。

比如，假设有两条字符串 1101 1001 和 1001 1101。比较时，如果字符相同用 0 表示，如果字符不同则用 1 表示。

$11011001 \oplus 10011101 = 01000100$

所以以上两条字符串之间的汉明距离为 2。

超结构

超结构主要用来计算某化学结构与其超结构的相似度。值越小则相似度越大。Milvus 目前只返回距离为 0 的结果。

超结构的公式可表示为：

$$1 - \frac{N_{A\&B}}{N_A}$$

Figure 16: superstructure

其中

- 分子式 B 是分子式 A 的超结构。

- NA 表示分子式 A 的化学指纹中二进制位的数量。
- NB 表示分子式 B 的化学指纹中二进制位的数量。
- NAB 表示分子式 A 和 B 的化学指纹中共有的二进制位的数量。

子结构

子结构主要用来计算某化学结构与其子结构的相似度。值越小则相似度越大。Milvus 目前只返回距离为 0 的结果。

子结构的公式可表示为:

$$1 - \frac{N_{A\&B}}{N_B}$$

Figure 17: substructure

其中

- 分子式 B 是分子式 A 的子结构。
- NA 表示分子式 A 的化学指纹中二进制位的数量。
- NB 表示分子式 B 的化学指纹中二进制位的数量。
- NAB 表示分子式 A 和 B 的化学指纹中共有的二进制位的数量。

常见问题

为什么向量距离计算方式是内积时，搜索出来的 top1 不是目标向量本身?

向量距离计算方式用内积时，如果向量未归一化，会出现这样的情况。

什么是归一化? Milvus 中为什么有时候需要归一化?

归一化指的是通过数学变换将向量的模长变为 1 的过程。如需使用点积计算向量相似度，则必须对向量作归一化处理。处理后点积与余弦相似度等价。

可参阅文章 向量搜索的简明数学基础。

为什么欧氏距离和内积在计算向量相似度时的结果不一致?

如果欧氏距离和内积返回不一致的结果，需要检查数据是否已经归一化。如果没有，请先对数据进行归一化。理论上可以证明，对于未归一化的数据，欧氏距离和内积的结果是不一致的。

## 索引概述

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

创建索引是一个组织数据的过程，是向量数据库实现快速查询百万、十亿、甚至万亿级数据集所依赖的一个巨大组成部分。

加速向量查询

相似性搜索引擎的工作原理是将输入的对象与数据库中的对象进行比较，找出与输入最相似的对象。索引是有效组织数据的过程，极大地加速了对大型数据集的查询，在相似性搜索的实现中起着重要作用。对一个大规模向量数据集创建索引后，查询可以被路由到最有可能包含与输入查询相似的向量的集群或数据子集。在实践中，这意味着要牺牲一定程度的准确性来加快对真正的大规模向量数据集的查询。

为提高查询性能，你可以为每个向量字段指定一种索引类型。目前，一个向量字段仅支持一种索引类型。切换索引类型时，Milvus 自动删除之前的索引。

索引创建机制

当 `create_index` 方法被调用时，Milvus 会同步为这个字段的现有数据创建索引。Segment 是 Milvus 中储存数据的最小单位。在建立索引时，Milvus 为每个 Segment 单独创建索引文件。

默认设定下，Milvus 不会对插入的数据少于 1024 行的 segment 创建索引。如修改此项参数，需修改 milvus.yaml 中的 rootCoord.minSegmentSizeToEnableIndex 配置项。

根据应用场景选择索引

Milvus 目前支持的向量索引类型大都属于 ANNS（Approximate Nearest Neighbors Search，近似最近邻搜索）。ANNS 的核心思想是不再局限于只返回最精确的结果项，而是仅搜索可能是近邻的数据项，即以牺牲可接受范围内的精度的方式提高检索效率。

关于索引和向量距离计算方法的选择，请访问 距离计算方式。

根据实现方式，ANNS 向量索引可分为四大类：

- 基于树的索引
- 基于图的索引
- 基于哈希的索引
- 基于量化的索引

下表将目前 Milvus 支持的索引进行了归类：

Milvus 支持的索引

索引分类

适用场景

FLAT

N/A

查询数据规模小

需要 100% 的召回率。

IVF_FLAT

基于量化的索引

高速查询

要求高召回率

IVF_SQ8

基于量化的索引

高速查询

磁盘和内存资源有限

查询召回率低于 IVF_FLAT

IVF_PQ

基于量化的索引

超高速查询

磁盘和内存资源有限

可以接受偏低的查询召回率

HNSW

基于图的索引

高速查询

要求尽可能高的召回率

内存空间大

IVF_HNSW

Quantization-and-graph-based index

High-speed query

Requires a recall rate as high as possible

Large memory resources

RHNSW_FLAT

Quantization-and-graph-based index

High-speed query

Requires a recall rate as high as possible

Large memory resources

RHNSW_SQ

Quantization-and-graph-based index

High-speed query

Limited memory resources

Accepts minor compromise in recall rate

RHNSW_PQ

Quantization-and-graph-based index

Very high-speed query

Limited memory resources

Accepts substantial compromise in recall rate

ANNOY

基于树的索引

低维向量空间

索引概览

FLAT

对于需要 100% 召回率且数据规模相对较小（百万级）的向量相似性搜索应用，FLAT 索引是一个很好的选择。FLAT 是指对向量进行原始文件存储，是唯一可以保证精确的检索结果的索引。FLAT 的结果也可以用于对照其他召回率低于 100% 的索引产生的结果。

FLAT 之所以精准是因为它采取了详尽查询的方法，即对于每个查询，目标输入都要与数据集中的每个向量进行比较。因此 FLAT 是列表中查询速度最慢的，而且不适合查询大量的向量数据。Milvus 中没有 FLAT 索引的参数，使用它不需要数据训练，也不需要占用额外的磁盘空间。

- 查询参数

| 参数 | 说明 | 取值范围 |
| --- | --- | --- |
| `metric_type` | [可选] 距离计算方式 | 详见 目前支持的距离计算方式。 |

IVF_FLAT

IVF_FLAT 它通过聚类方法把空间里的点划分至 `nlist` 个单元，然后比较目标向量与所有单元中心的距离，选出 `nprobe` 个最近单元。然后比较这些被选中单元里的所有向量，得到最终的结果，极大地缩短了查询时间。

通过调整 `nprobe`，可以找到特定场景下查询准确性和查询速度之间的理想平衡。IVF_FLAT 性能测试结果表明，随着目标输入向量的数量（`nq`）和需要检索的集群数量（`nprobe`）的增加，查询时间也急剧增加。

IVF_FLAT 是最基础的 IVF 索引，存储在各个单元中的数据编码与原始数据一致。

- 建索引参数

| 参数 | 说明 | 取值范围 |
|---|---|---|
| `nlist` | 聚类单元数 | [1, 65536] |

- 查询参数

| 参数 | 说明 | 取值范围 |
|---|---|---|
| `nprobe` | 查询取的单元数 | [1, 65536] |

IVF_SQ8

由于 IVF_FLAT 未对原始的向量数据做任何压缩，IVF_FLAT 索引文件的大小与原始数据文件大小相当。例如 sift-1b 数据集原始数据文件的大小为 476 GB，生成的 IVF_FLAT 索引文件大小有 470 GB 左右，若将全部索引文件加载进内存，就需要 470 GB 的内存资源。

当磁盘或内存、显存资源有限时，IVF_SQ8 是一个更好的选择。它通过对向量进行标量量化（scalar quantization），能把原始向量中每个 FLOAT（4 字节）转为 UINT8（1 字节），从而可以把磁盘及内存、显存资源的消耗量减少为原来的 1/4 至 1/3。同样以 sift-1b 数据集为例，生成的 IVF_SQ8 索引文件只有 140 GB。

- 建索引参数

| 参数 | 说明 | 取值范围 |
|---|---|---|
| `nlist` | 聚类单元数 | [1, 65536] |

- 查询参数

| 参数 | 说明 | 取值范围 |
|---|---|---|
| `nprobe` | 查询取的单元数 | [1, nlist] |

IVF_PQ

`PQ`（Product Quantization，乘积量化）会将原来的高维向量空间均匀分解成 `m` 个低维向量空间的笛卡尔积，然后对分解得到的低维向量空间分别做矢量量化。最终每条向量会存储在 `m` × `nbits` 个 bit 位里。乘积量化能将全样本的距离计算转化为到各低维空间聚类中心的距离计算，从而大大降低算法的时间复杂度。

IVF_PQ 是先对向量做乘积量化，然后进行 IVF 索引聚类。其索引文件甚至可以比 IVF_SQ8 更小，不过同样地也会导致查询时的精度损失。

不同版本的建索引参数和查询参数设置不同，请根据使用的 Milvus 版本查看相应的参数信息。

- 建索引参数

| 参数 | 说明 | 取值范围 |
| --- | --- | --- |
| `nlist` | 聚类单元数 | [1, 65536] |
| `m` | 乘积量化因子个数 | $\dim \equiv 0 \pmod{m}$ |
| `nbits` | 分解后每个低维向量的存储位数 (可选) | [1, 16]（默认 8） |

- 查询参数

| 参数 | 说明 | 取值范围 |
| --- | --- | --- |
| `nprobe` | 查询取的单元数 | [1, nlist] |

HNSW

HNSW（Hierarchical Small World Graph）是一种基于图的索引算法。它会为一张图按规则建成多层导航图，并让越上层的图越稀疏，结点间的距离越远；越下层的图越稠密，结点间的距离越近。搜索时从最上层开始，找到本层距离目标最近的结点后进入下一层再查找。如此迭代，快速逼近目标位置。

为了提高性能，HNSW 限定了每层图上结点的最大度数 `M` 。此外，建索引时可以用 `efConstruction`，查询时可以用 `ef` 来指定搜索范围。

- 建索引参数

| 参数 | 说明 | 取值范围 |
| --- | --- | --- |
| `M` | 结点的最大度数 | [4, 64] |
| `efConstruction` | 搜索范围 | [8, 512] |

- 查询参数

| 参数 | 说明 | 取值范围 |
| --- | --- | --- |
| `ef` | 搜索范围 | [`top_k`, 32768] |

IVF_HNSW

IVF_HNSW is an indexing algorithm based on IVF_FLAT and HNSW. Using HNSW indexing algorithm as quantizer, this index type builds the multi-layer navigation structure with the `nlist` cluster units divided by IVF_FLAT indexing algorithm, so that it can approach the target position quickly.

- Index building parameters

| Parameter | Description | Range |
| --- | --- | --- |
| `nlist` | Number of cluster units | [1, 65536] |
| `M` | Maximum degree of the node | [4, 64] |
| `efConstruction` | Search scope | [8, 512] |

- Search parameters

| Parameter | Description | Range |
| --- | --- | --- |
| `nprobe` | Number of units to query | [1, nlist] |
| `ef` | Search scope | [`top_k`, 32768] |

RHNSW_FLAT

RHNSW_FLAT (Refined Hierarchical Small World Graph) is a refined indexing algorithm based on HNSW. This index type optimizes the data storage solution of HNSW and thereby reduces the storage consumption.

- Index building parameters

| Parameter | Description | Range |
|---|---|---|
| `M` | Maximum degree of the node | [4, 64] |
| `efConstruction` | Search scope | [8, 512] |

- Search parameters

| Parameter | Description | Range |
|---|---|---|
| `ef` | Search scope | [`top_k`, 32768] |

RHNSW_SQ

RHNSW_SQ (Refined Hierarchical Small World Graph and Scalar Quantization) is a refined indexing algorithm based on HNSW. This index type performs scalar quantization on vector data on the basis of HNSW and thereby substantially reduces the storage consumption.

- Index building parameters

| Parameter | Description | Range |
|---|---|---|
| `M` | Maximum degree of the node | [4, 64] |
| `efConstruction` | Search scope | [8, 512] |

- Search parameters

| Parameter | Description | Range |
|---|---|---|
| `ef` | Search scope | [`top_k`, 32768] |

RHNSW_PQ

RHNSW_SQ (Refined Hierarchical Small World Graph and Product Quantization) is a refined indexing algorithm based on HNSW. This index type performs product quantization on vector data on the basis of HNSW and thereby significantly reduces the storage consumption.

- Index building parameters

| Parameter | Description | Range |
|---|---|---|
| `M` | Maximum degree of the node | [4, 64] |
| `efConstruction` | Search scope | [8, 512] |
| `PQM` | Number of factors of product quantization | dim $\equiv 0 \pmod{\texttt{PQM}}$ |

- Search parameters

| Parameter | Description | Range |
|---|---|---|
| `ef` | Search scope | [`top_k`, 32768] |

Annoy

Annoy（Approximate Nearest Neighbors Oh Yeah）是一种用超平面把高维空间分割成多个子空间，并把这些子空间以树型结构存储的索引方式。

在查询时，Annoy 会顺着树结构找到距离目标向量较近的一些子空间，然后比较这些子空间里的所有向量（要求比较的向量数不少于 `search_k` 个）以获得最终结果。显然，当目标向量靠近某个子空间的边缘时，有时需要大大增加搜索的子空间数以获得高召回率。因此，Annoy 会使用 `n_trees` 次不同的方法来划分全空间，并同时搜索所有划分方法以减少目标向量总是处于子空间边缘的概率。

- 建索引参数

| 参数 | 说明 | 取值范围 |
|---|---|---|
| `n_trees` | 空间划分的方法数 | $[1, 1024]$ |

- 查询参数

| 参数 | 说明 | 取值范围 |
|---|---|---|
| `search_k` | 搜索的结点数。`-1` 表示用全数据量的 5% | $\{\text{-1}\} \cup [\texttt{top\_k}, \text{n} \times \texttt{n\_trees}]$ |

常见问题

Milvus 中 FLAT 索引和 IVF_FLAT 索引的原理比较?

把 FLAT 和 IVF_FLAT 做比较，可以这么估算：

已知 IVF_FLAT 索引是把向量分成 nlist 个单元。假设用默认的 nlist = 16384，搜索的时候是先用目标向量和这 16384 个中心点计算距离，得到最近的 nprobe 个单元，再在单元里计算最近向量。而 FLAT 是每条向量和目标向量计算距离。

所以当总的向量条数约等于 nlist 时，两者的计算量相当，性能也差不多。而随着向量条数达到 nlist 的 2 倍、3 倍、n 倍之后，IVF_FLAT 的优势就越来越大。

可参阅 如何选择索引类型。

参考文献


- HNSW：Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs
- Annoy：Nearest neighbors and vector models　part 2　algorithms and data structures


## Schema

Field Schema

A field schema is the logical definition of a field. It is the first thing you need to define before defining a collection schema and creating a collection.

Milvus 2.0 supports only one primary key field in a collection.

Field schema properties

Poperties

```
    <th>Description</th>
    <th>Note</th>
</tr>
</thead>
<tbody>
<tr>
```

```
    <td>name</td>
    <td>Name of the field in the collection to create</td>
    <td>Data type: String.<br/>Mandatory</td>
</tr>
<tr>
    <td>dtype</td>
    <td>Data type of the field</td>
    <td>Mandatory</td>
</tr>
<tr>
    <td>description</td>
    <td>Description of the field</td>
    <td>Data type: String.<br/>Optional</td>
</tr>
<tr>
    <td>is_primary</td>
    <td>Whether to set the field as the primary key field or not</td>
    <td>Data type: Boolean (<code>true</code> or <code>false</code>).<br/>Mandatory for the primary key fi
</tr>
<tr>
    <td>dim</td>
    <td>Dimension of the vector</td>
<td>Data type: Integer &isin;[1, 32768].<br/>Mandatory for the vector field</td>
</tr>
</tbody>
```

Create a field schema

```
from pymilvus import FieldSchema
id_field = FieldSchema(name="id", dtype=DataType.INT64, is_primary=True, description="primary id")
age_field = FieldSchema(name="age", dtype=DataType.INT64, description="age")
embedding_field = FieldSchema(name="embedding", dtype=DataType.FLOAT_VECTOR, dim=128, description="vector"
```

Supported data type

`DataType` defines the kind of data a field contains. Different fields support different data types.

- Primary key field supports:
  - INT8: numpy.int8
  - INT16: numpy.int16
  - INT32: numpy.int32
  - INT64: numpy.int64
- Scalar field supports:
  - BOOL: Boolean (`true` or `false`)
  - INT8: numpy.int8
  - INT16: numpy.int16
  - INT32: numpy.int32
  - INT64: numpy.int64
  - FLOAT: numpy.float32
  - DOUBLE: numpy.double
- Vector field supports:
  - BINARY_VECTOR: Binary vector
  - FLOAT_VECTOR: Float vector

Collection Schema

Collection schema 是 collection 的逻辑定义。通常你需要在定义 collection schema 和 创建 collection 之前定义 field schema。

Collection schema 属性

属性

```
    <th>描述</th>
    <th>备注</th>
</tr>
</thead>
<tbody>
<tr>
    <td>field</td>
    <td>要创建的 collection 中的 field</td>
    <td>强制</td>
</tr>
<tr>
    <td>description</td>
    <td>collection 描述</td>
    <td>数据类型：String。<br/>可选</td>
</tr>
<tr>
    <td>auto_id</td>
    <td>是否启用自动分配 ID（即 primary key）</td>
    <td>数据类型：Boolean（<code>true</code> 或 <code>false</code>）。<br/>可选</td>
</tr>
</tbody>
```

创建 collection schema

先定义 field schema，再定义 collection schema。

```
from pymilvus import FieldSchema, CollectionSchema
id_field = FieldSchema(name="id", dtype=DataType.INT64, is_primary=True, description="primary id")
age_field = FieldSchema(name="age", dtype=DataType.INT64, description="age")
embedding_field = FieldSchema(name="embedding", dtype=DataType.FLOAT_VECTOR, dim=128, description="vector"
schema = CollectionSchema(fields=[id_field, age_field, embedding_field], auto_id=False, description="desc
```

使用指定的 schema 创建 collection：

```
from pymilvus import Collection
collection_name1 = "tutorial_1"
collection1 = Collection(name=collection_name1, schema=schema, using='default', shards_num=2)
```

你可以使用 shards_num 参数定义分片编号，并在 using 中指定 alias 来定义在哪个 Milvus server 中创建 collection。

你也可以使用 `Collection.construct_from_dataframe` 自动从 DataFrame 生成一个 collection schema 并创建一个 collection。

```
import pandas as pd
df = pd.DataFrame({
        "id": [i for i in range(nb)],
        "age": [random.randint(20, 40) for i in range(nb)],
        "embedding": [[random.random() for _ in range(dim)] for _ in range(nb)]
    })
collection, ins_res = Collection.construct_from_dataframe(
                                'my_collection',
                                df,
                                primary_field='id',
                                auto_id=False
                                )
```

## 布尔表达式语法规则

表达式计算结果将输出布尔值——真（TRUE）或假（False）。在搜索向量时，Milvus 通过表达式进行标量过滤。查看 Python SDK API 参考以了解表达式使用说明。

扩展巴科斯范式（EBNF）语法规则中定义了布尔表达式的语法规则。布尔表达式的语法规则如下所示：

```
Expr = LogicalExpr | NIL

LogicalExpr = LogicalExpr BinaryLogicalOp LogicalExpr
              | UnaryLogicalOp LogicalExpr
              | "(" LogicalExpr ")"
              | SingleExpr;

BinaryLogicalOp = "&&" | "and" | "||" | "or";

UnaryLogicalOp = "not";

SingleExpr = TermExpr | CompareExpr;

TermExpr = IDENTIFIER "in" ConstantArray;

Constant = INTERGER | FLOAT

ConstantExpr = Constant
               | ConstantExpr BinaryArithOp ConstantExpr
               | UnaryArithOp ConstantExpr;

ConstantArray = "[" ConstantExpr { "," ConstantExpr } "]";

UnaryArithOp = "+" | "-"

BinaryArithOp = "+" | "-" | "*" | "/" | "%" | "**";

CompareExpr = IDENTIFIER CmpOp IDENTIFIER
              | IDENTIFIER CmpOp ConstantExpr
              | ConstantExpr CmpOp IDENTIFIER
              | ConstantExpr CmpOpRestricted IDENTIFIER CmpOpRestricted ConstantExpr;

CmpOpRestricted = "<" | "<=";

CmpOp = ">" | ">=" | "<" | "<=" | "=="| "!=";
```

上述布尔表达式规则中提及的符号及其含义请见下表。

| 表达式 | 含义 |
| --- | --- |
| = | 定义。 |
| , | 串联。 |
| ; | 终止。 |
| {…} | 重复。 |
| (…) | 分组。 |
| NIL | 空值。表达式可以为空字符串。 |
| INTEGER | 整数，如 1、2、3 等。 |
| FLOAT | 浮点数，如 1.0、2.0。 |
| CONST | 整数或浮点数。 |
| IDENTIFIER | 标识符。在 Milvus 用来表示 Field 名称。 |

| 表达式 | 含义 |
| --- | --- |
| LogicalOp | LogicalOp 包括 BinaryLogicalOp 及 UnaryLogicalOp，支持在一次比较运算中包含多种逻辑关系。LogicalOp 返回结果为 TRUE (1) 或 FALSE (0)。 |
| UnaryLogicalOp | 一元逻辑运算符‘否’。 |
| BinaryLogicalOp | 在运算元数量大于等于 2 的复杂表达式中，运算先后顺序由运算符的优先级决定。 |
| ArithmeticOp | ArithmeticOp 计算数学关系，如加法、减法等。 |
| UnaryArithOp | UnaryArithOp 表达式中仅包含 1 个运算元。一元否定运算符会将表达式中符号的含义从肯定变为否定，或从否定变为肯定。 |
| BinaryArithOp | 在运算元数量大于等于 2 的复杂表达式中，运算先后顺序由运算符的优先级决定。 |
| CmpOp | 关系运算符。 |
| CmpOpRestricted | 仅指代“小于”以及“小于或等于”。 |
| ConstantExpr | ConstantExpr 可用于面向 2 个 ConstantExpr 的常量的 BinaryArithOp，或面向 1 个 ConstantExpr 的 UnaryArithOp。ConstantExpr 的定义方式为递归定义。 |
| ConstantArray | 使用方括号来表示 ConstantArray。ConstantExpr 可在方括号内重复。ConstantArray 必须包含至少 1 个 ConstantExpr。 |
| TermExpr | TermExpr 用于检查 IDENTIFIER 的值是否出现在 ConstantArray 中。TermExpr 由“in”来表示。 |
| CompareExpr | CompareExpr 可用于面向 2 个 IDENTIFIER 或面向 1 个 IDENTIFIER 和 1 个 ConstantExpr 的关系运算。CompareExpr 还可以运用于面向 2 个 ConstantExpr 和 1 个 IDENTIFIER 的三元运算。 |
| SingleExpr | SingleExpr 可以是 TermExpr 或 CompareExpr。 |
| LogicalExpr | LogicalExpr 可以是面向 2 个 LogicalExpr 的 BinaryLogicalOp，或面向 1 个 LogicalExpr 的 UnaryLogicalOp，或是 1 个按括号分组的 LogicalExpr、或是 1 个 SingleExpr。LogicalExpr 的定义方式为递归定义。 |
| Expr | Expr 可以是 LogicalExpr 或 NIL。 |

运算符

逻辑运算

逻辑运算符用于将条件进行关联。它们对这些条件返回的布尔值执行逻辑运算。

| 符号 | 运算 | 示例 | 规则 |
| --- | --- | --- | --- |
| 且运算符（&&） | and | expr1 && expr2 | 若 expr1 和 expr2 两个均为 true，则结果为 true。 |
| 或运算符（\|\|） | or | expr1 \|\| expr2 | 若 expr1 或 expr2 中任意一个为 true，则结果为 true。 |

二元算数运算符

二元算数运算符用于运算两个运算元之间的基本算数关系，并返回相应结果。

| 符号 | 运算 | 示例 | 规则 |
| --- | --- | --- | --- |
| + | 加 | a + b | 计算 a 与 b 的和。 |
| - | 减 | a - b | 计算 a 与 b 的差。 |
| * | 乘 | a * b | 计算 a 与 b 的乘积。 |
| / | 除 | a / b | 计算 a 除以 b 的商。 |

| 符号 | 运算 | 示例 | 规则 |
| --- | --- | --- | --- |
| ** | 幂 | a ** b | 计算 a 的 b 次方。 |
| % | 取模 | a % b | 计算 a 除以 b 得到的余数。 |

关系运算符

关系运算符用于确定两个表达式之间是否存在等于、不等于、大于、小于等特定关系。

| 符号 | 运算 | 示例 | 规则 |
| --- | --- | --- | --- |
| < | 小于 | a < b | 若 a 小于 b，则结果为 true。 |
| > | 大于 | a > b | 若 a 大于 b，则结果为 true。 |
| == | 等于 | a == b | 若 a 等于 b，则结果为 true。 |
| != | 不等于 | a != b | 若 a 不等于 b，则结果为 true。 |
| <= | 小于等于 | a <= b | 若 a 小于等于 b，则结果为 true。 |
| >= | 大于等于 | a >= b | 若 a 大于等于 b，则结果为 true。 |

运算符优先级及关联性

下表为运算符的优先级及关联性。表中运算符优先级按照从高到低排序。

| 优先级 | 符号 | 运算符 | 关联性 |
| --- | --- | --- | --- |
| 1 | + - | 一元算数运算符（UnaryArithOp） | 从左到右 |
| 2 | not | 一元逻辑运算符（UnaryLogicalOp） | 从右到左 |
| 3 | ** | 二元算数运算符（BinaryArithOp） | 从左到右 |
| 4 | * / % | 二元算数运算符（BinaryArithOp） | 从左到右 |
| 5 | + - | 二元算数运算符（BinaryArithOp） | 从左到右 |
| 6 | < <= > >= | 二元算数运算符（BinaryArithOp） | 从左到右 |
| 7 | == != | 比较运算符（CmpOp） | 从左到右 |
| 8 | && and | 二元逻辑运算符（BinaryLogicalOp） | 从左到右 |
| 9 | \|\| or | 二元逻辑运算符（BinaryLogicalOp） | 从左到右 |

表达式运算通常遵循从左到右的计算规则。一次只能计算一种复杂表达式。运算先后顺序由运算符的优先级决定。

如果表达式中包含两个或以上同等优先级的运算符，运算时从左到右进行。

例如，10 / 2 * 5 的运算过程为 (10 / 2) 的商乘以 5。

若需要先处理低优先级的运算，请使用括号。

例如，30 / 2 + 8 的运算过程为 30 除以 2 的商加 8。若你想计算 30 除以 2 加 8 的和 10，则应将表达式写作 30 / (2 + 8).

你可以在表达式中使用括号。运算时先计算最内层括号内的表达式。

## Time Travel

Milvus Docs 需要你的帮助

本文档暂时没有中文版本，欢迎你成为社区贡献者，协助中文技术文档的翻译。你可以通过页面右边的 编辑 按钮直接贡献你的翻译。更多详情，参考 贡献指南。如需帮助，你可以 提交 GitHub Issue。

This topic introduces the Time Travel feature in detail, including how it is designed and how it works in Milvus. See Search with Time Travel for more information about how to use this feature.

Data engineers often need to roll back data to fix dirty data or bugs. Unlike traditional databases that use snapshots or retrain data to achieve data rollback, Milvus maintains a timeline for all data insert or delete operations. Therefore, users can specify the timestamp in a query to retrieve data at a specific point of time, which can significantly reduce maintenance costs.

Design Details

When the proxy receives a data insert or delete request, it also gets a timestamp from root coord. Then, the proxy adds the timestamp as an additional field to the inserted or deleted data. Timestamp is a data field just like primary key (`pk`). Data in the same insert or delete request share the same timestamp. The timestamp field is stored together with other data fields of a collection.

When you load a collection to memory, all data in the collection, including their corresponding timestamps, are loaded into memory.

During a search, if the search request received by the proxy contains the parameter, `travel_timestamp`, the value of this parameter will be passed to segcore, the execution engine which supports concurrent insertion, deletion, query, index loading, monitoring and statistics of a segment data in memory. The segcore filters the search results by timestamp.

Search implementation

Searches with filtering in knowhere is achieved by bitset. Bitset can be applied in the following three aspects:

- Delete data
- Timestamp
- Attribute filtering

When searching in segcore, you can obtain a bitset indicating if the timestamp meets the condition. Then, the segcore combines the timestamp bitset with the other two types of bitsets, data deletion bitset and attribute filtering bitset. Finally, a bitset containing all deletion, attribute filtering, and timestamp information is generated. Then Milvus judges the range of data to query or search based on this bitset.

All CRUD operations within Milvus are executed in memory. Therefore, you need to load collection from disk to memory before searching with Time Travel.

Sealed segment

For sealed segments, you need to call `collection.load()` to load the collection to memory before searching with Time Travel. As an additional field of data, timestamps are also loaded to memory when you call `collection.load()`. When loading, segcore builds an index, `TimestampIndex` , on the timestamp field. The index contains information about the smallest and the largest timestamp of this sealed segment, and the offset, or the row number, of each timestamp in the segment.

When you search with Time Travel, Milvus first filters the sealed segment according to the smallest and largest timestamp in the `TimestampIndex`:

- If the value you set for `travel_timestamp` is greater than the largest timestamp of the segment, this means all the data in this segment meets the requirement. Therefore, the bitset of the data in this segment is marked as 1.
- If the value you set for `travel_timestamp` is smaller than the smallest timestamp of the segment, this means the data in this segment does not meet the requirement. Therefore, the bitset of the data in this segment is marked as 0.
- If the value you set for `travel_timestamp` is between the largest and the smallest timestamp of the segment, Milvus compares the timestamps in the segment one by one, and generates a bitset accordingly. In the bitset, if the data meet the requirement, they are marked with 1, and 0 if they do not.

Growing segment

For growing segments, you do not need to load the collection to memory. All inserted data exists in memory, with the timestamp field attached. Data in growing segments are sorted according to the order of timestamp. When new data are inserted, they are added to the segment in the order of their timestamp. Segment data are organized in segcore memory in the same way.

When you search with Time Travel, Milvus uses binary search to find the first offset, or the row number data, with their timestamp value greater than the value you set for the `travel_timestamp` parameter. Then subsequent operations including filtering and vector similarity search are conducted within this range of offsets.

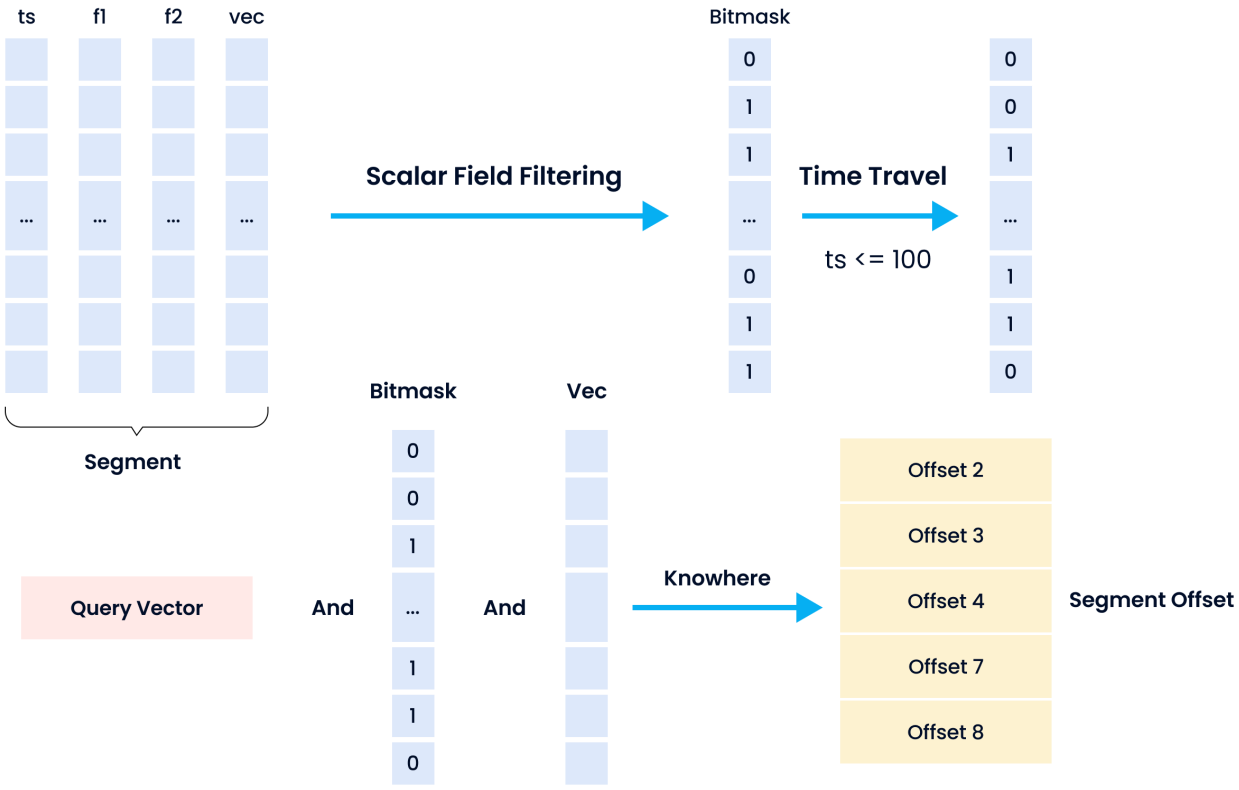



Figure 18: Time_travel

What’s next

After learning how Time Travel works in Milvus, you might also want to:

- Learn how to search with Time Travel
- Learn the architecture of Milvus.
- Understand how data are processed in Milvus.

## Milvus 术语

术语表

Collection

包含一组 entity，可以等价于关系型数据库系统（RDBMS）中的表。

Dependency

依赖是另一个程序赖以工作的程序。Milvus 的依赖包括 etcd (存储元数据)、MinIO 或 S3 (对象存储) 和 Pulsar( 管理快照日志)。

Entity

包含一组 field。field 与实际对象相对应。field 可以是代表对象属性的结构化数据，也可以是代表对象特征的向量。primary key 是用于指代一个 entity 的唯一值。

你可以自定义 primary key，否则 Milvus 将会自动生成 primary key。请注意，目前 Milvus 不支持 primary key 去重，因此有可能在一个 collection 内出现 primary key 相同的 entity。

Field

Entity 的组成部分。Field 可以是结构化数据，例如数字和字符串，也可以是向量。

Milvus 2.0 现已支持标量字段过滤。

Partition

分区是集合的一个分区。Milvus 支持将收集数据划分为物理存储上的多个部分。这个过程称为分区，每个分区可以包含多个段。

PChannel

PChannel 表示物理信道。每个 PChannel 对应一个日志存储主题。默认情况下，将分配一组 64 个 PChannels 来存储记录 Milvus 集群启动时数据插入、删除和更新的日志。

Schema

模式是定义数据类型和数据属性的元信息。每个集合都有自己的集合模式，该模式定义了集合的所有字段、自动 ID (主键) 分配支持以及集合描述。集合模式中还包括定义字段名称、数据类型和其他属性的字段模式。

Segment

Milvus 在数据插入时通过合并数据自动创建的数据文件。一个 collection 可以包含多个 segment。一个 segment 可以包含多个 entity。在搜索中，Milvus 会搜索每个 segment，并返回合并后的结果。

Sharding

Shard 是指将数据写入操作分散到不同节点上，使 Milvus 能充分利用集群的并行计算能力进行写入。默认情况下单个 collection 包含 2 个分片（shard）。目前 Milvus 采用基于主键哈希的分片方式，未来将支持随机分片、自定义分片等更加灵活的分片方式。

Partition 的意义在于通过划定分区减少数据读取，而 shard 的意义在于多台机器上并行写入操作。

VChannel

VChannel 表示逻辑通道。每个集合将分配一组 VChannels，用于记录数据的插入、删除和更新。VChannels 在逻辑上是分开的，但在物理上共享资源。

单机部署

一种 Milvus 的部署方式。在单机部署模式下，数据插入、索引构建、近似搜索等所有操作都在一个进程中完成。

分布式部署

一种 Milvus 的部署方式。在分布式部署模式下，Milvus 服务由一组节点共同提供，可实现高可用和易扩展。

非结构化数据

非结构化数据，包括图像、视频、音频和自然语言，是不遵循预定义模型或组织方式的信息。这种数据类型约占全球数据的 80%，可以通过各种人工智能 (AI) 和机器学习 (ML) 模型转换为矢量。

归一化

归一化指的是通过数学变换将向量的模长变为 1 的过程。如需使用点积计算向量相似度，则必须对向量作归一化处理。处理后点积与余弦相似度等价。

日志代理

日志代理是一个支持回放的发布-订阅系统。它负责流数据持久化、可靠异步查询的执行、事件通知和查询结果的返回。当工作节点从系统崩溃中恢复时，它还确保增量数据的完整性。

日志订阅者

日志订阅方通过订阅日志序列来更新本地数据，并以只读副本的形式提供服务。

日志序列

日志序列记录了在 Milvus 中更改集合状态的所有操作。

索引

索引基于原始数据构建，可以提高对 collection 数据搜索的速度。Milvus 支持多种索引类型。

向量

向量是非结构化数据的特征抽象，比如电子邮件、物联网传感器数据、Instagram 照片、蛋白质结构等等。从数学上讲，向量是一个浮点数或二进制数组。现代嵌入技术将非结构化数据转化为向量。

向量相似性搜索

向量相似度搜索是将一个向量与数据库进行比较，找出与目标搜索向量最相似的向量的过程。近似最近邻 (ANN) 搜索算法用于计算向量之间的相似度。

# Milvus 系统搭建教程

## 图片检索系统

图片相似度检索

本教程将介绍如何使用开源向量数据库 Milvus 搭建图片相似度检索系统。

- 打开 Jupyter notebook
- 快速部署

- 在线体验 本教程中使用到的 ML 模型及第三方软件包括:
- YOLOv3
- ResNet-50
- MySQL

像谷歌这样的大型搜索引擎已经为用户提供了按图片搜索的选项。另外，电商平台已经意识到以图搜图功能可以方便网购者，所以亚马逊在其智能手机应用程序中集成了以图搜图功能。

在本教程中，你将学会如何构建一个图片相似度检索系统。该系统可以检测图案，并返回与你上传的图片相似的其他图片。为了搭建这样一个图片相似度检索系统，请先下载包含 20 个类别、17125 张图片的 PASCAL VOC 图片数据集。然后使用 YOLOv3 进行目标检测、使用 ResNet-50 进行图像特征提取。所有图片通过上述两个机器学习（ML）模型被转换为 256 维的向量。将图片向量存储在 Milvus 中，Milvus 自动为每个向量生成唯一的 ID。然后使用 MySQL 用于存储 向量 ID 及数据集图片间的映射关系。新上传到图片搜索系统中的图片将被转换为新的向量，Milvus 将比较新向量与之前存储在 Milvus 中的所有向量数据的相似度，并返回最相似向量的 ID。随后，你可以在 MySQL 中查询 ID 所对应的图像。

## 智能问答机器人

本教程将介绍如何使用开源向量数据库 Milvus 搭建智能问答（QA）系统。

- 打开 Jupyter notebook
- 快速部署
- 在线体验

本教程中使用到的 ML 模型及第三方软件包括:

- BERT
- MySQL

通过本教程，你将学习到如何搭建一个 QA 系统，用于检索向量数据库中存储的大量问答是否与用户问题相关联。要构建这样的聊天机器人，请提前准备好自己的问题和相应的答案数据集。将问题和答案存储在关系数据库 MySQL 中。然后使用用于自然语言处理 (NLP) 的机器学习 (ML) 模型 BERT 将问题转换为向量。使用 Milvus 存储这些问题向量并构建索引。当用户输入一个新问题时，BERT 模型会将其转换为一个新向量，然后 Milvus 会搜索与这个新向量最相似的问题向量。最终 QA 系统返回最相似问题的相应答案。

## 推荐系统

本教程将介绍如何使用开源向量数据库 Milvus 搭建推荐系统。

- 打开 Jupyter notebook
- 快速部署

本教程中使用到的 ML 模型及第三方软件包括: - PaddlePaddle - Redis or MySQL

推荐系统是一种信息过滤系统，可用于推荐个性化电影、音乐、产品、订阅消息等各种应用场景。与搜索引擎不同，推荐系统不需要用户准确地描述他们的需求，可以通过分析用户行为来发现用户的需求和兴趣。

通过本教程，你将学会如何搭建一个电影推荐系统，可以根据用户的兴趣来推荐电影。要构建这样的推荐系统，首先下载一个与电影相关的数据集。本教程使用 MovieLens 1M。或者你也可以准备自己的数据集，里面应包括用户对电影的评分，用户的特征统计和电影的描述。使用 PaddlePaddle 组合用户 ID 和特征，并将它们转换为 256 维向量。以类似的方式将电影 ID 和特征转换为向量。将电影向量存储在 Milvus 中，并使用用户向量进行相似度搜索。如果用户向量与电影向量相似，Milvus 将返回电影向量及其 ID 作为推荐结果。然后使用存储在 Redis 或 MySQL 中的电影向量 ID 查询电影信息。

## 视频检索系统

视频相似度检索

本教程将介绍如何使用开源向量数据库 Milvus 搭建视频相似度检索系统。

- 打开 Jupyter notebook
- 快速部署 本教程中使用到的 ML 模型及第三方软件包括:
- OpenCV
- ResNet-50
- MySQL

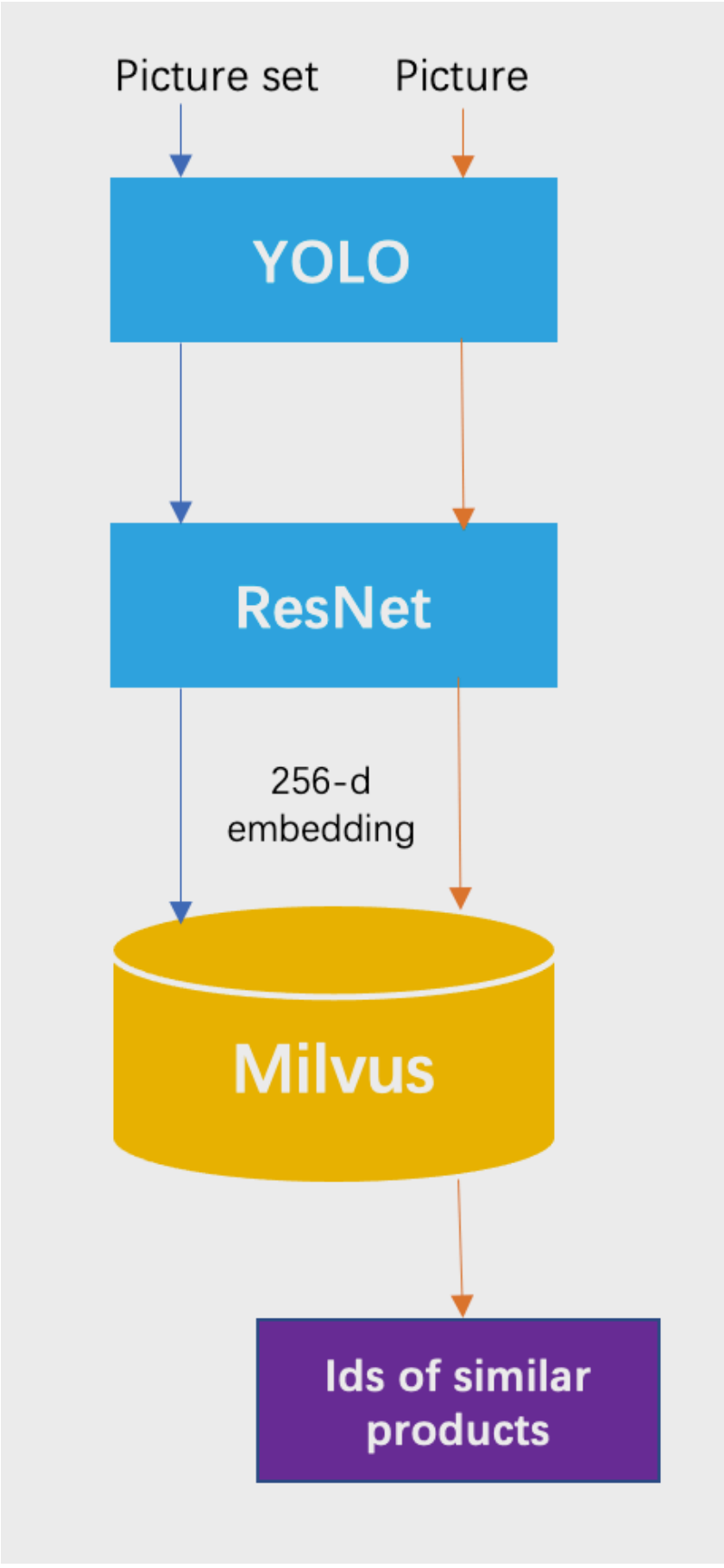



Figure 19: image_search

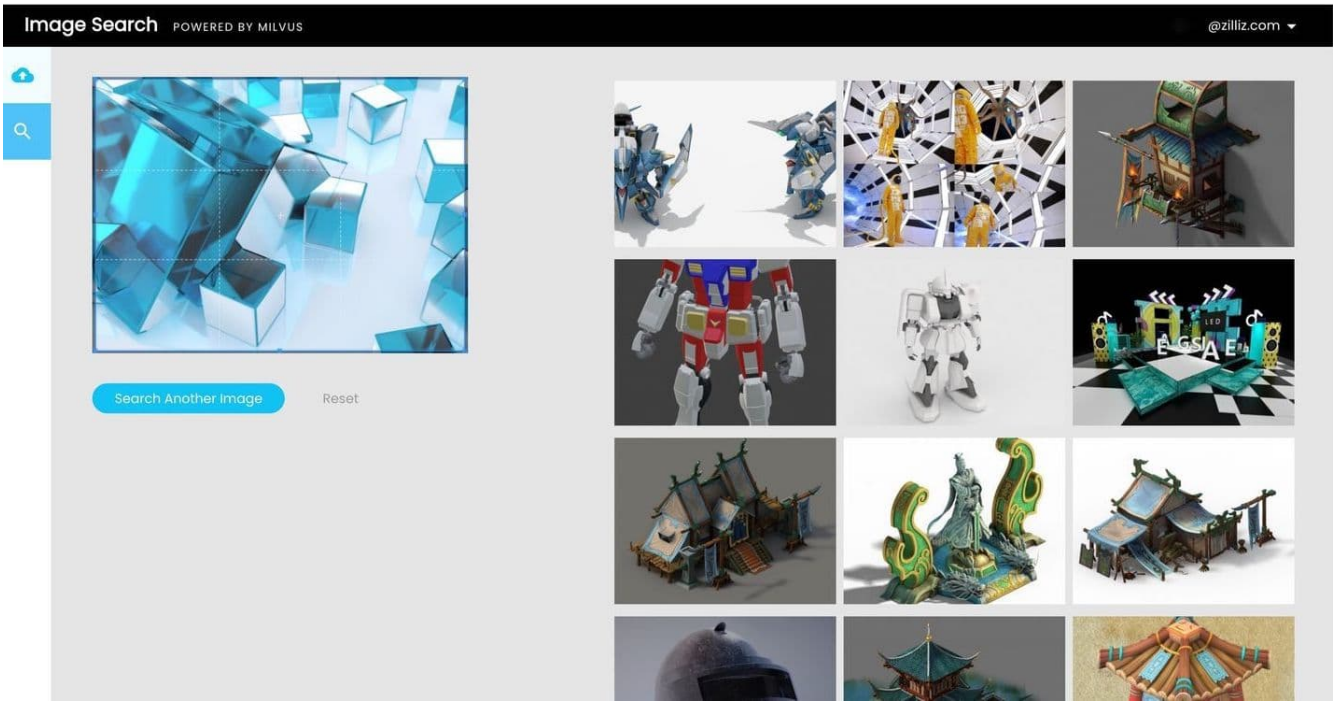



Figure 20: image_search_demo

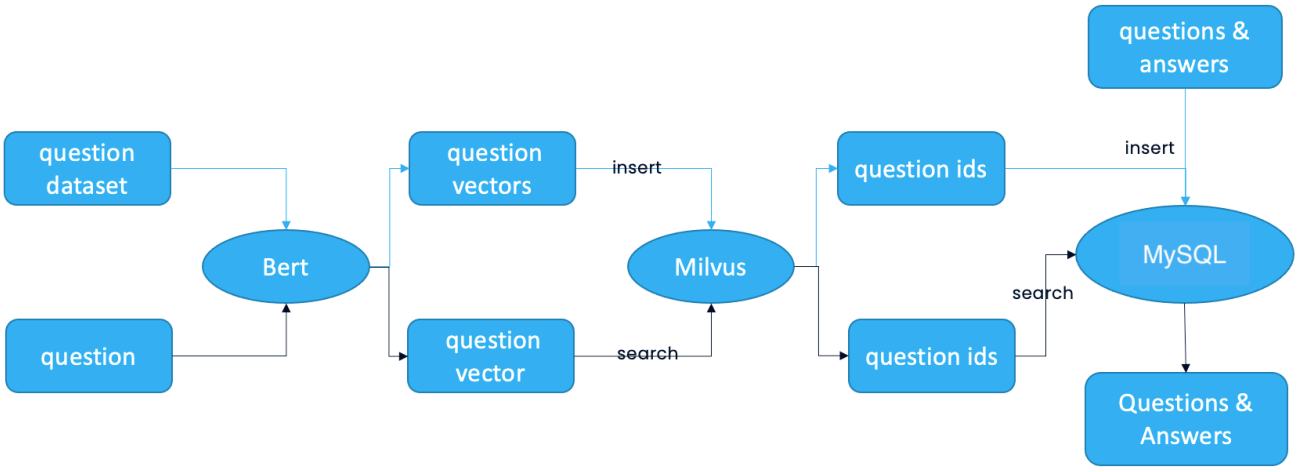



Figure 21: Qa_chatbot

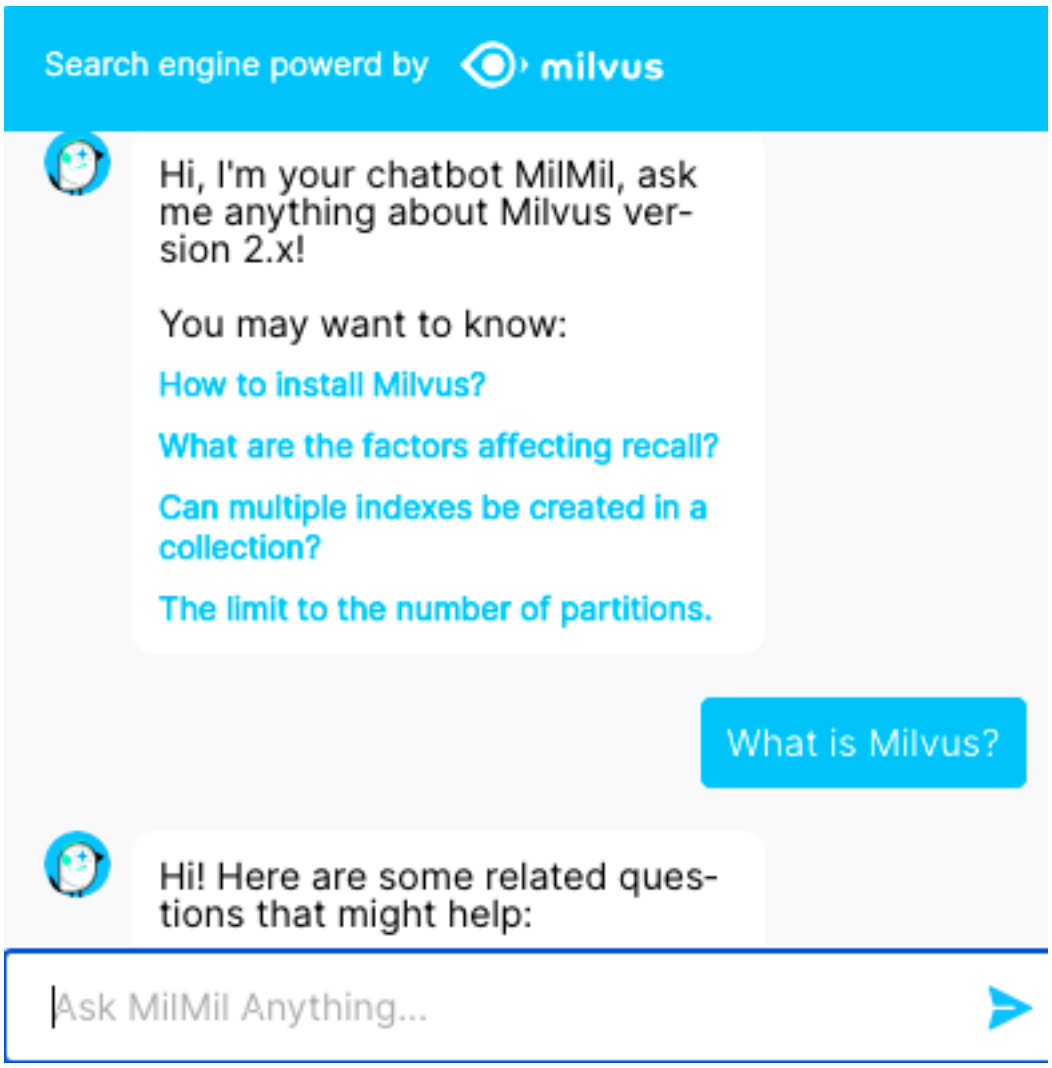



Figure 22: QA_chatbot_demo

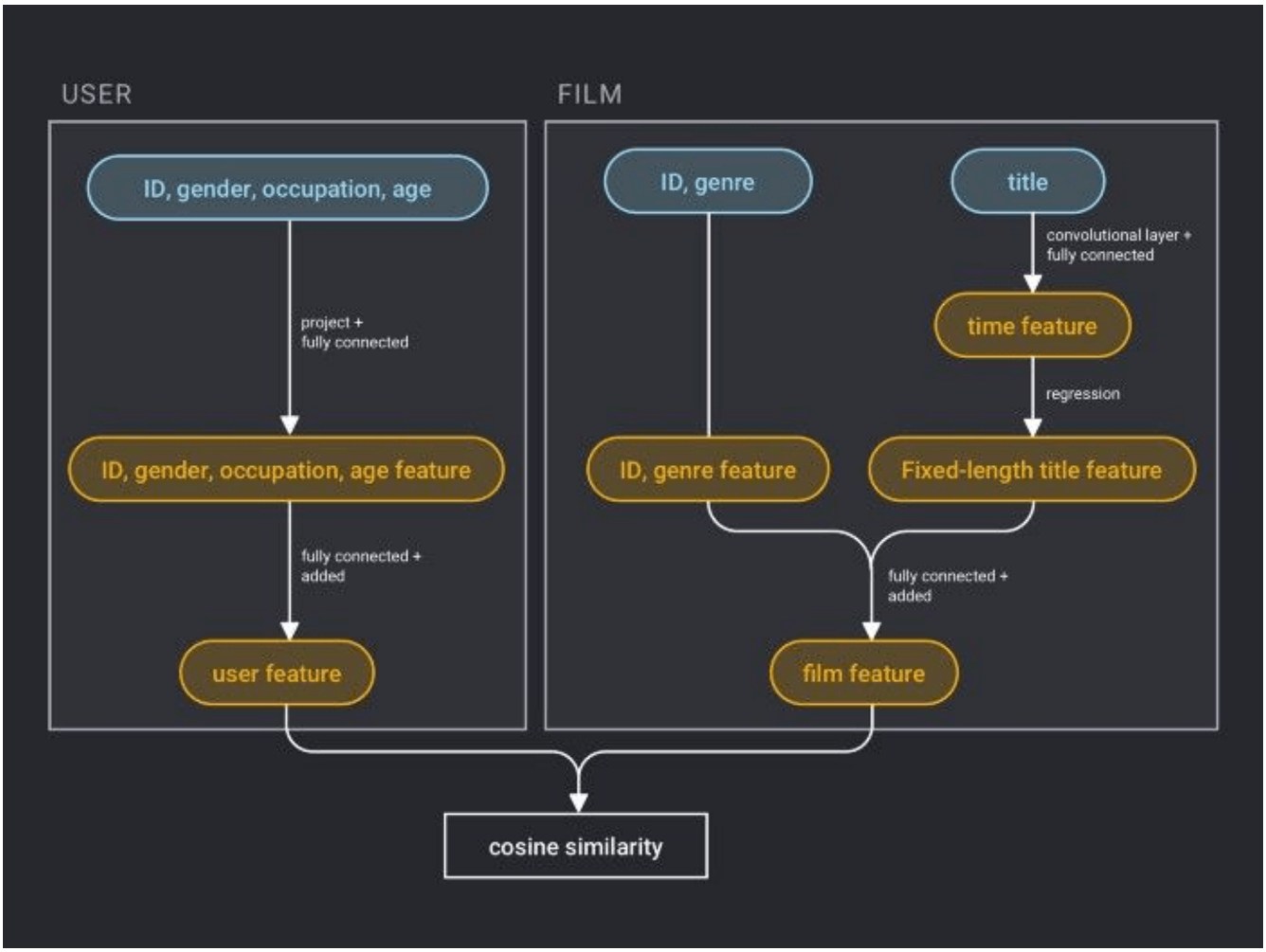
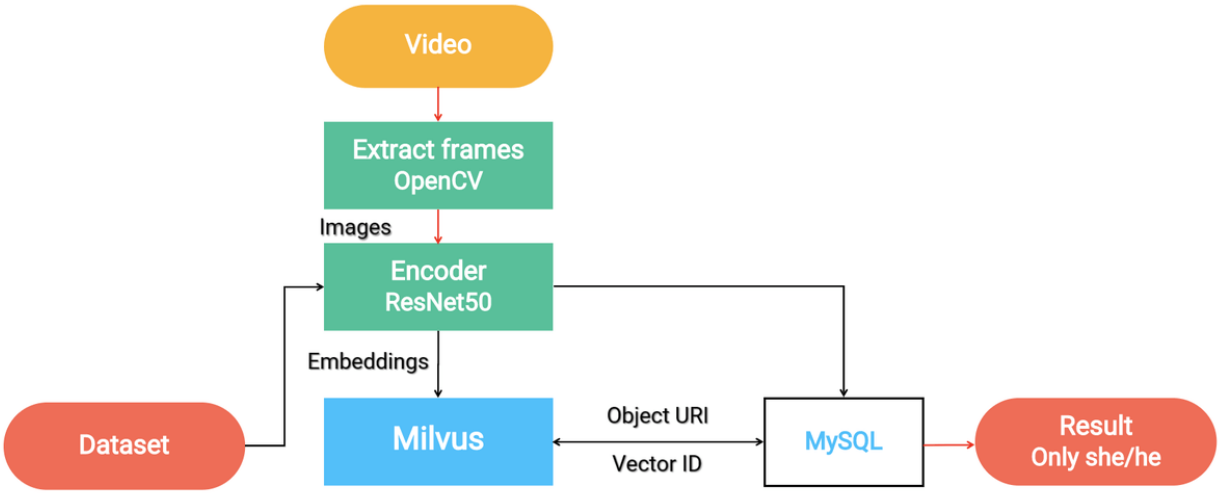
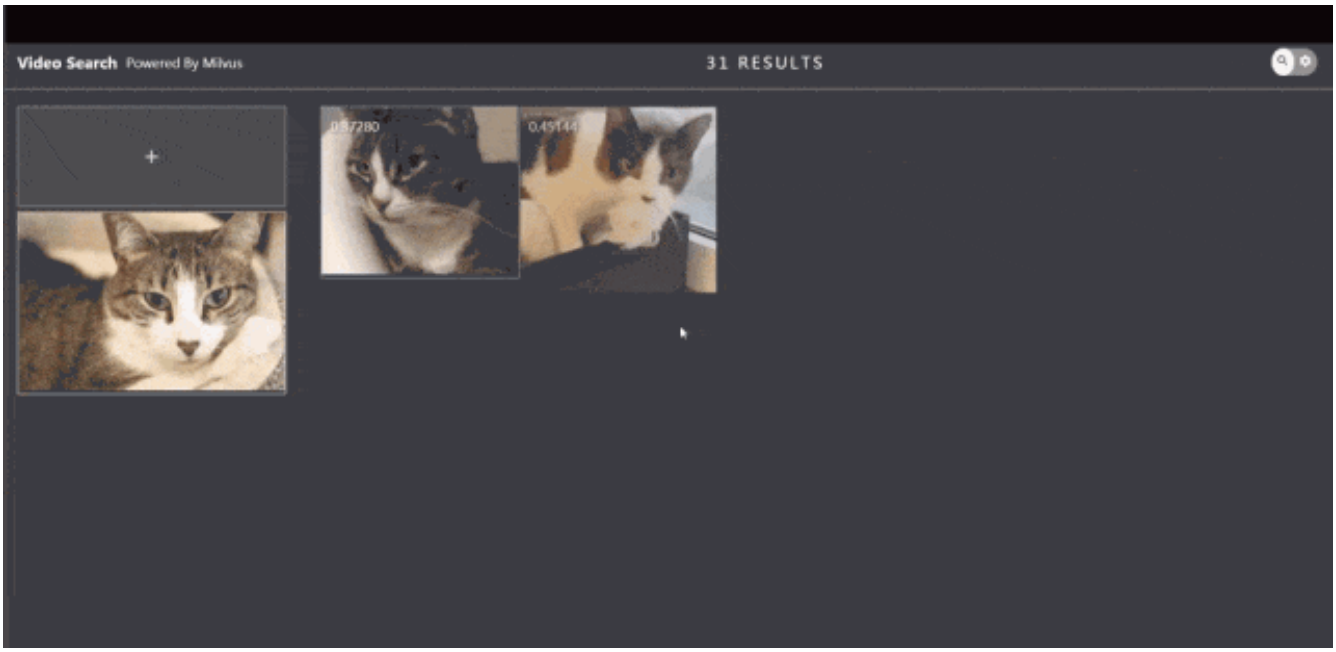



Figure 23: recommender_system

如今，看完了自己喜欢的电影或者视频后，人们喜欢在各种社交网络平台上发帖，通过截图分享他们的想法。当粉丝看到截图之后，如果帖子上没有明确指出电影名称，他们很难分辨出是哪部电影。为了找出电影的名称，人们可以利用视频相似度搜索系统。通过使用该系统，用户可以上传图像并获取包含与上传图像相似的关键帧的视频或电影。

在本教程中，你将学会如何搭建一个视频相似度检索系统。本教程使用的数据集为来自 Tumblr 的约 100 个 gif 动图。当然，你也可以准备自己的视频数据集。该系统首先使用 OpenCV 提取视频中的关键帧，然后使用 ResNet-50 获取每个关键帧的特征向量。使用 Milvus 存储向量并进行向量相似性检索。Milvus 会返回相似向量的 ID。然后通过 MySQL 中存储的映射关系，找到向量 ID 所对应的视频。




## 音频检索系统

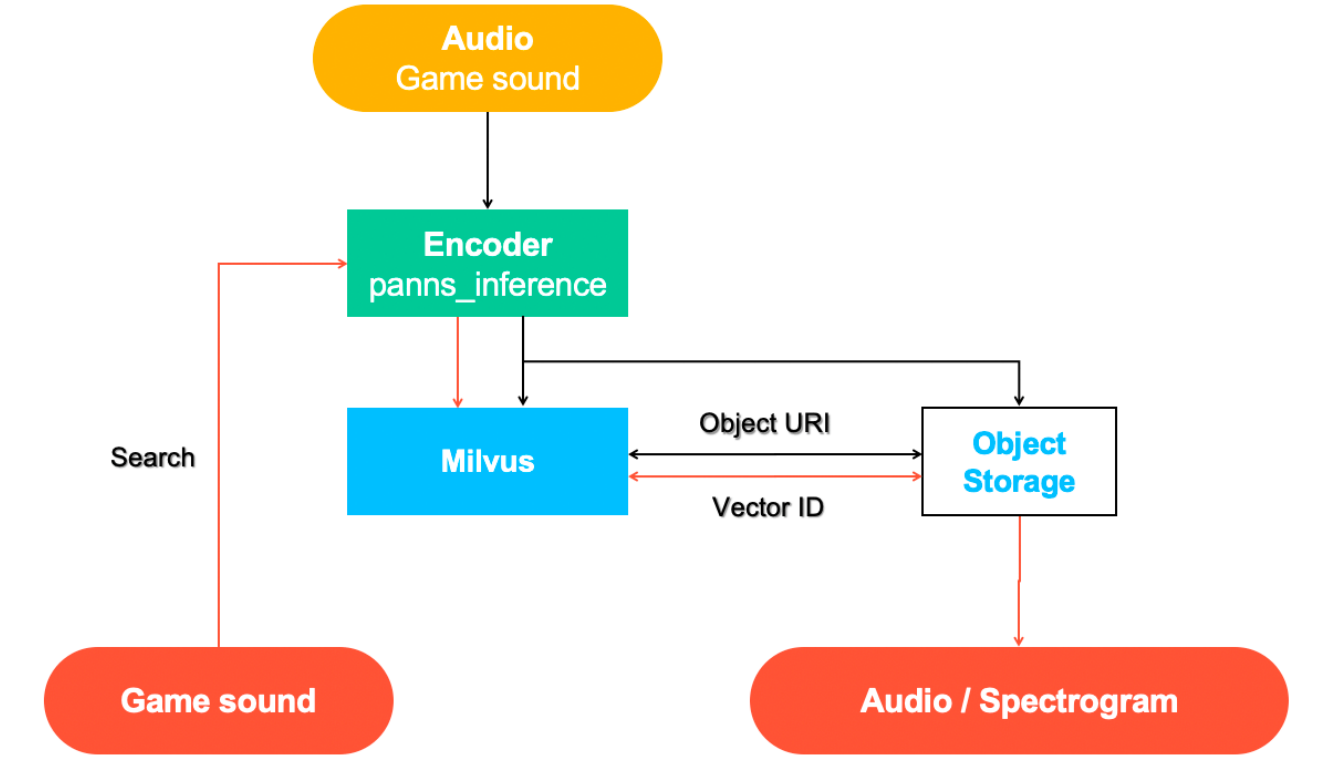
音频相似度检索

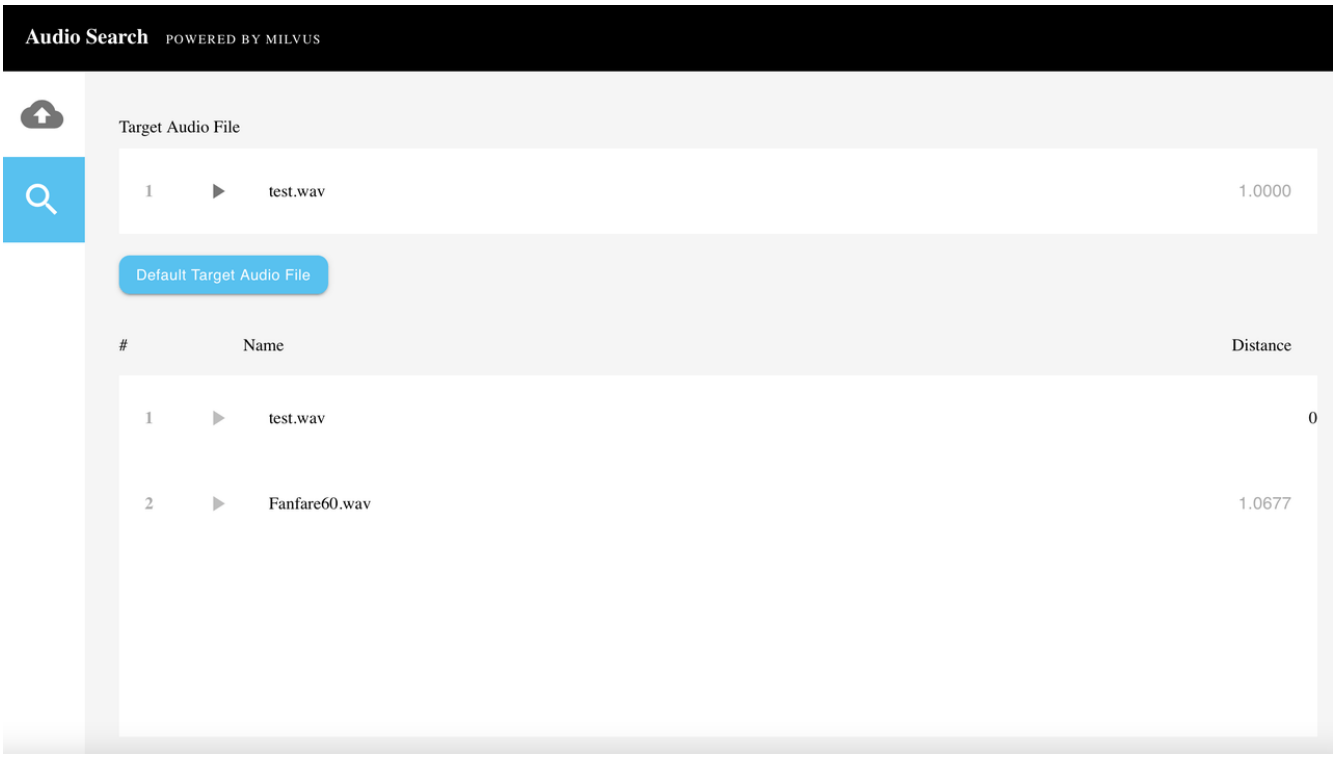
本教程将介绍如何使用开源向量数据库 Milvus 搭建音频相似度检索系统。

- 打开 Jupyter notebook
- 快速部署 本教程中使用到的 ML 模型及第三方软件包括：
- PANNs (大规模预训练音频神经网络)
- MySQL

音频检索（如演讲、音乐、音效等检索）实现了在海量音频数据中查询并找出相似声音片段。音频相似性检索系统可用于识别相似的音效、最大限度减少知识产权侵权等。音频检索还可以用于实时网络媒体的搜索和监控，来打击侵犯知识产权的行为。在音频数据的分类和统计分析中，音频检索也发挥着重要作用。

在本教程中，你将学会如何构建一个音频检索系统，用来检索相似的声音片段。使用 PANNs 将上传的音频片段转换为向量数据，并存储在 Milvus 中。Milvus 自动为每个向量生成唯一的 ID。然后用户就可以在 Milvus 中进行向量相似度搜索，Milvus 返回的检索结果为向量 ID，每个 ID 对应音频片段数据的路径。



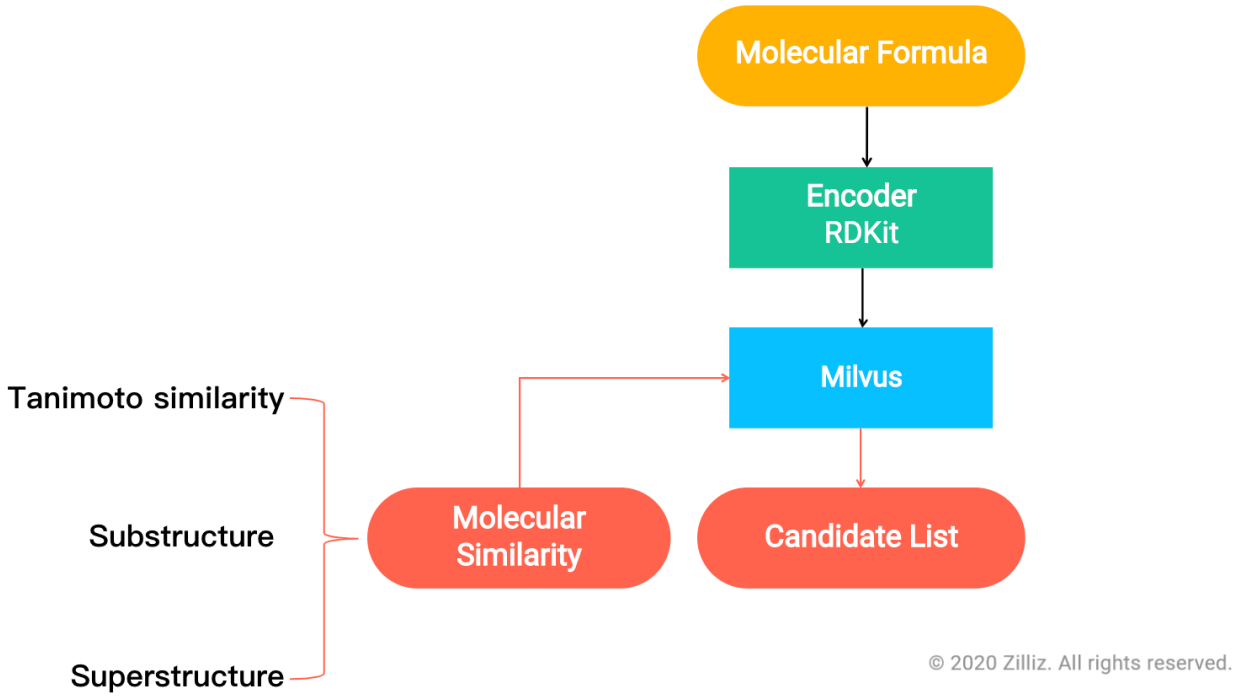
## 分子式检索系统

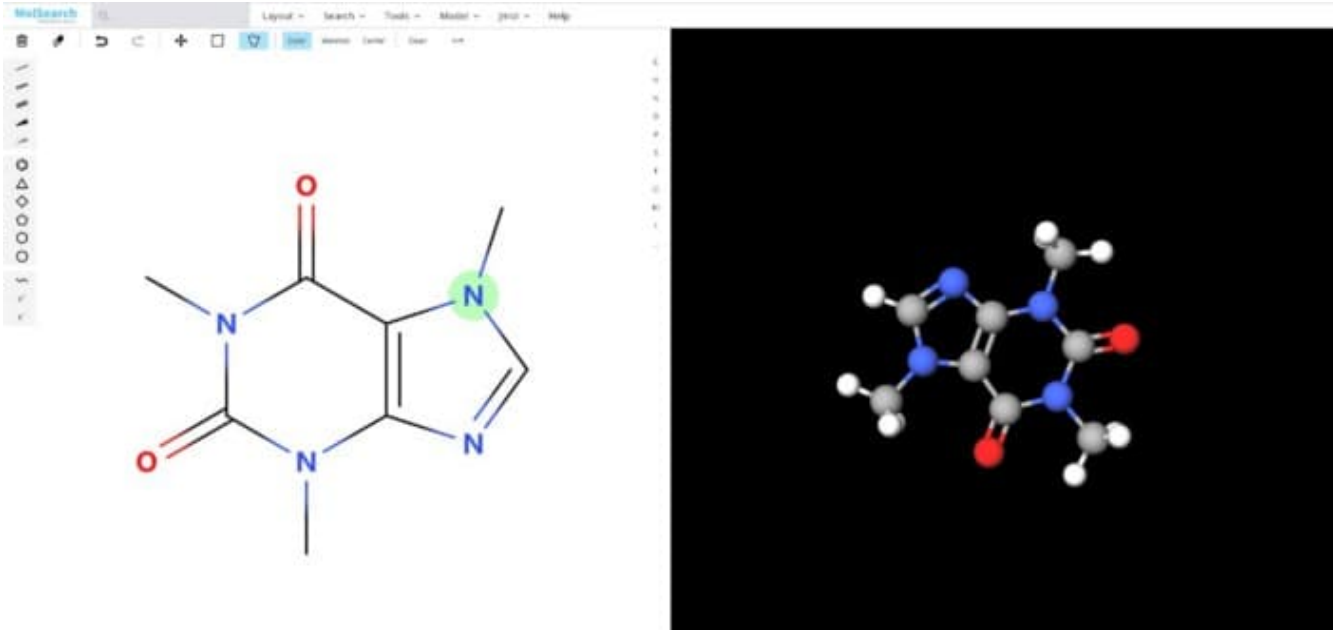
本教程将介绍如何使用开源向量数据库 Milvus 搭建分子式检索系统。

- 打开 Jupyter notebook
- 快速部署
- 在线体验 本教程中使用到的 ML 模型及第三方软件包括:
- RDKit
- MySQL

药物发现是新药研发中的重要一环。药物发现过程包括了靶点选择和确认。当发现片段或先导化合物时，研究人员通常会在内部或商业化合物库中搜索类似的化合物，以发现构效关系 (SAR) 和化合物的可用性。最终，他们将评估先导化合物成为候选化合物的潜力。为了从十亿规模的化合物库中发现可用的化合物，通常检索化学指纹以进行子结构搜索和分子相似性搜索。

通过本教程，你将学习到如何搭建分子式检索系统，该系统可以检索特定分子的子结构，超结构，和相似结构。RDKit 是一个开源化学信息学软件，可以将分子结构转换为向量。然后，向量存储在 Milvus 中，Milvus 可以对向量进行相似度搜索。Milvus 还会自动为每个向量生成一个唯一的 ID。向量 ID 和分子结构的映射存储在 MySQL 中。





## DNA 序列分类模型

DNA 序列分类

本教程将介绍如何使用开源向量数据库 Milvus 搭建 DNA 序列分类模型。- 打开 Jupyter notebook - 快速部署 本教程中使用到的 ML 模型及第三方软件包括：- CountVectorizer - MySQL

DNA 序列是基因溯源、物种鉴定、疾病诊断等领域内的流行概念。各行各业都渴望发现更智能、更高效的研究方法。因此，人工智能的运用在生物和医学领域内备受关注。越来越多的科学家和研究人员开始做出贡献，努力推动机器学习（machine learning）和深度学习（deep learning）在生物信息学领域内的应用。通常，科学家和研究人员会通过增加样本量来提高实验结果的说服力。将基因组学与大数据相结合能够拓展实际应用场景。然而，传统的序列比对存在局限性，不适用于大型数据集的比对。为解决这一问题，可以选择将大型 DNA 序列数据向量化。

通过本教程，你将学习到如何搭建 DNA 序列分类系统。首先，使用 CountVectorizer 模型提取 DNA 序列特征并转化为向量。然后，将向量存储在 Milvus 中，将向量与 DNA 分类信息的对应关系存储在 MySQL 中。使用 Milvus 进行向量相似度搜索，并从 MySQL 中获取对应的 DNA 分类。

## 文本搜索引擎

本教程将介绍如何使用开源向量数据库 Milvus 搭建一个文本搜索引擎。

- 打开 Jupyter notebook
- 快速部署 本教程中使用到的 ML 模型及第三方软件包括：
- BERT
- MySQL

Milvus 在自然语言处理（NLP）领域的一个主要应用是文本搜索引擎。这是一个很棒的工具，可以帮助用户找到他们正在寻找的信息。它甚至可以显示难以找到的信息。文本搜索引擎将用户输入的关键字或语义与文本数据库进行比较，然后返回满足特定条件的结果。

通过本教程，你将学习到如何构建一个文本搜索引擎。本教程使用 BERT 将文本转换为固定长度的向量。Milvus 作为向量数据库，用于存储和向量相似性搜索。使用 MySQL 存储 Milvus 向量 ID 与文本数据间的映射关系。

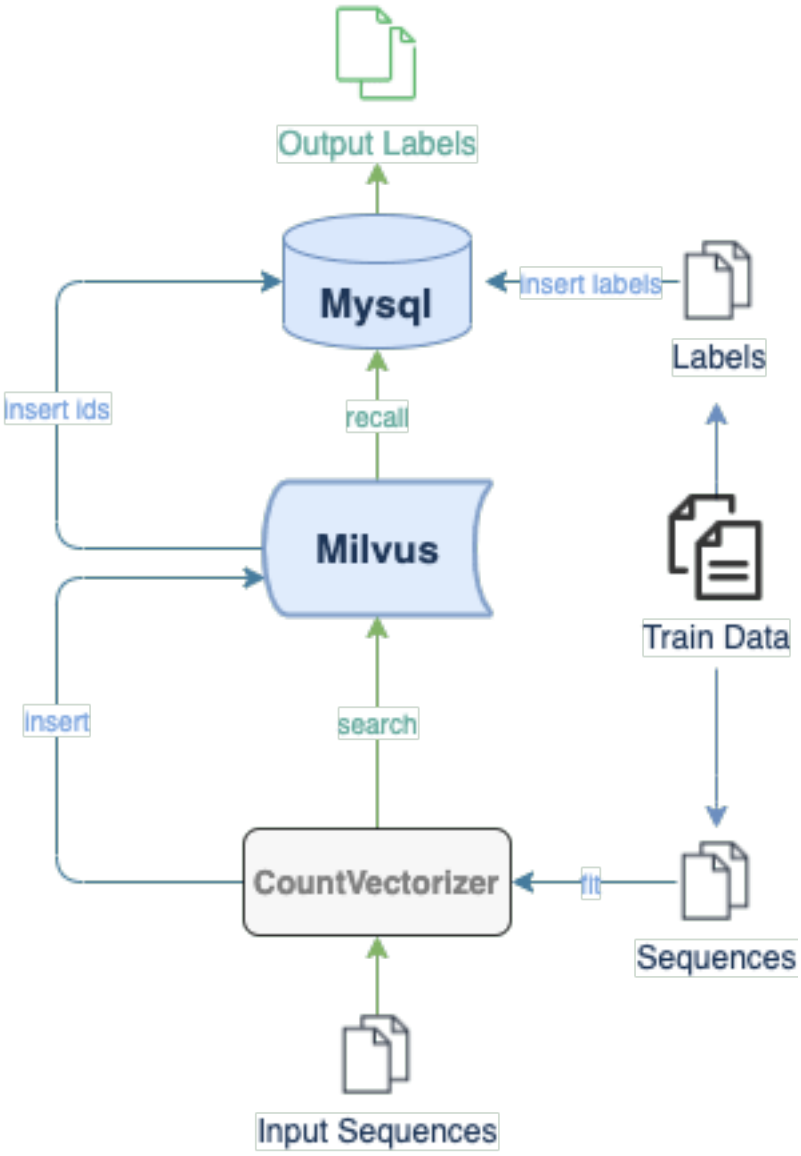
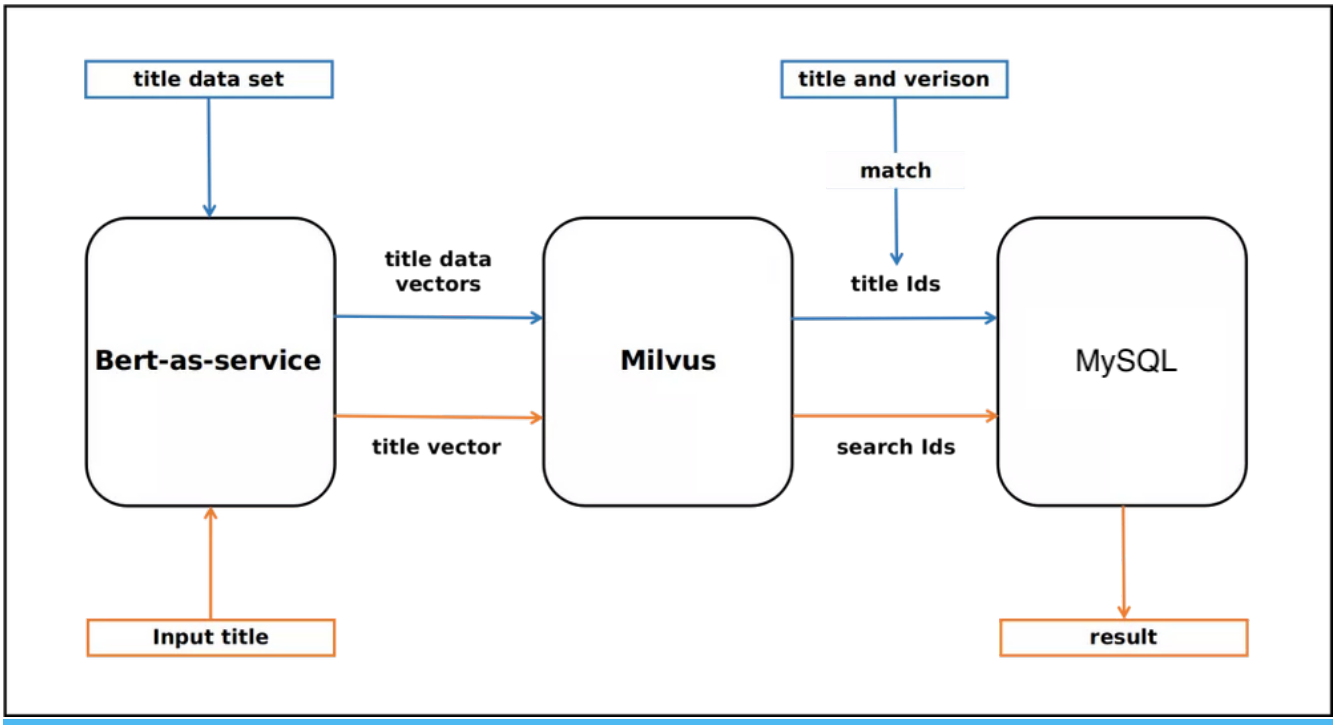



Figure 24: dna




**Search Engine** powered by Milvus  Phelps Eyes Fourth Gold

## Phelps Eyes Fourth Gold

ATHENS (Reuters) - A weary Michael Phelps targeted his fourth Olympic gold medal in Athens, turning his attention on Wednesday to the 200 meters individual medley and settling for the second-fastest overall time in the heats.

Show more

## Indians Mount Charge

The Cleveland Indians pulled within one game of the AL Central lead by beating the Minnesota Twins, 7-1, Saturday night with home runs by Travis Hafner and Victor Martinez.

Show more

## Giddy Phelps Touches Gold for First Time

Michael Phelps won the gold medal in the 400 individual medley and set a world record in a time of 4 minutes 8.26 seconds.

Show more

## Home Depot Likes High Oil

# FAQs

## 性能调优

### 性能常见问题

#### 如何设置 IVF 索引的 `nlist` 和 `nprobe` 参数?

IVF 索引的 `nlist` 值应根据具体使用情况设置。一般来说，建议值为 `4 × sqrt(n)`，其中 `n` 指 segment 最多包含的 entity 条数。

每个 segment 的大小由参数 `datacoord.segment.maxSize` 决定，默认为 512 MB。Segment 内的 entity 条数可通过将 `datacoord.segment.maxSize` 除以每条 entity 的大小估算得出。

`nprobe` 值的选取需要根据数据总量和实际场景在查询性能和准确率之间进行取舍。建议通过多次实验确定合理取值。

以下是使用公开测试数据集 sift50m 针对 `nlist` 和 `nprobe` 的一个测试。以索引类型 IVF_SQ8 为例，测试对比了不同 `nlist/nprobe` 组合的搜索时间和召回率。

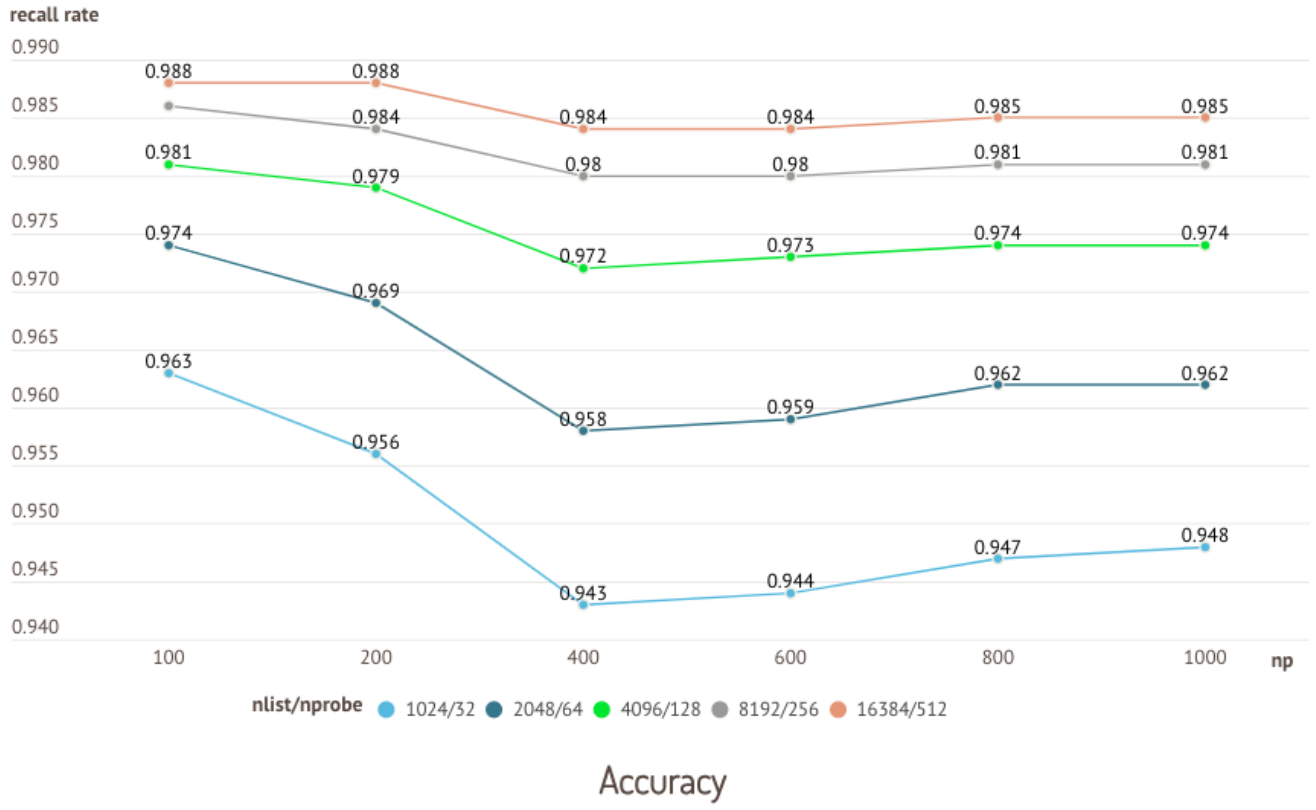



Figure 25: accuracy_nlist_nprobe.png

测试显示，召回率与 `nlist/nprobe` 值呈正相关。

#### 为什么有时小数据集查询时间反而更长?

查询操作在 segment 上进行，有索引时查询性能更高。如果 segment 尚未构建索引，Milvus 将对原始数据进行暴力搜索，大大增加查询时长。

因此，当小数据集（collection）尚未创建索引时，就有可能出现查询时间更长的情况。这是因为其 segment 的大小没有达到 `rootcoord.minSegmentSizeToEnableIndex` 所设定的索引构建阈值。调用 `create_index()` 方法可对已达到该阈值但没有构建索引的 segment 强制构建索引以加快查询速度。

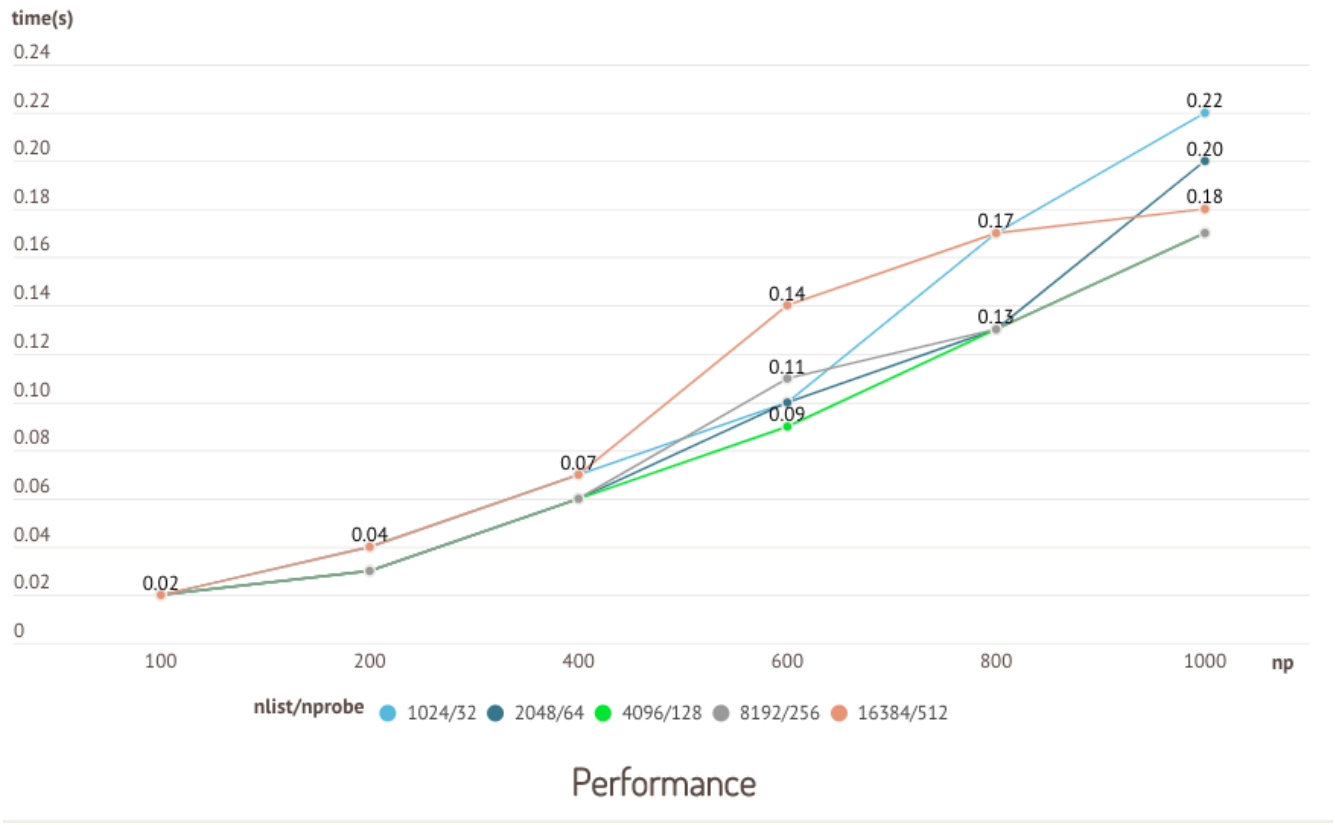



Figure 26: performance_nlist_nprobe.png

CPU 利用率受哪些因素的影响?

Milvus 构建索引和执行查询操作时，CPU 利用率提高。一般来说，除了 ANNOY 索引是单线程运行之外，索引构建都会占用大量 CPU 资源。

向量查询时，CPU 利用率受参数 `nq` 和 `nprobe` 影响，当 `nq` 和 `nprobe` 都较小时，程序并发度较小，故 CPU 利用率不高。

边插入边搜索会影响搜索性能吗?

数据插入本身并不是一个 CPU 密集型操作，但是由于新插入的数据所在 segment 的大小可能还未达到自动创建索引的阈值，在查询时只能对其采用暴搜的方式，查询性能就会降低。

参数 `rootcoord.minSegmentSizeToEnableIndex` 决定了 segment 自动构建索引的阈值，默认值为 1024。更多详情，请见系统配置。

仍有问题没有得到解答?

如果仍有其他问题，你可以：

- 访问我们的 GitHub 主页，与我们分享你的问题和想法，或帮助其他用户。
- 加入我们的 Slack 社区，参与开源社区的讨论交流。

## 产品问题

产品常见问题

Milvus 会收费吗?

Milvus 会坚持开源路线，软件本身不会收取任何费用。

请遵循 Apache 2.0 协议 使用 Milvus 开源项目。

对于不愿意自行搭建和维护 Milvus 分布式实例的用户，Zilliz 也提供全托管式平台 Zilliz Cloud，允许用户根据实际使用服务付费，数据可靠性由云平台负责管理。

Milvus 支持非 x86 平台吗?

Milvus 暂不支持非 x86 平台。

为保证 Milvus 的正常运行，你的 CPU 须支持以下任一指令集：SSE4.2、AVX、AVX2、AVX512。这些均为 x86 平台专用的 SIMD 指令集。

Milvus 可以处理百亿或千亿级数据吗?

理论上来说，Milvus 能够处理的数据规模取决于用户自身的硬件条件，其中有两大关键指标，即系统内存容量和持久化存储空间容量。

- 执行查询操作前，Milvus 需先将所有指定的 collection 或 partition 加载到内存。因此，内存容量决定了 Milvus 可查询数据的上限。
- 执行插入操作时，Milvus 需先将所有的 entity 以及 collection 相关的 schema （当前仅支持 MinIO 作为持久化存储）全部写入持久化存储。因此，持久化存储空间的容量决定了 Milvus 可插入数据的上限。

Milvus 数据存储在哪里?

Milvus 包含两部分数据：用户插入的数据和元数据。

用户插入的数据以增量日志的方式存储在持久化存储上（当前仅支持 MinIO 作为持久化存储），包括向量数据、标量数据以及 collection 相关 schema 等。

每个 Milvus 模块都会产生各自的元数据，存储在 etcd 中。

为什么我在 etcd 找不到向量数据?

etcd 只用于存放 Milvus 系统的元数据，entity 存储在 MinIO 中。

Milvus 的 Python SDK 有连接池吗?

Milvus v0.9.0 及更高版本对应的 Python SDK 有连接池。连接池的连接数量没有上限。

Milvus 是否支持 “边插入边查询”?

支持。插入操作和查询操作由两个相互独立的模块分开执行，因此互不影响。对于客户端，插入数据进入消息队列即意味着该插入操作结束，尽管此时的数据可能还无法被查询到。只有加载到 query node 的数据才能被用户查询到。若插入的 segment 的大小未满足构建索引的阈值（默认值为 512 MB），Milvus 将使用暴搜，这种情况下的查询性能会受到一定影响。

Milvus 允许插入重复 ID 的向量吗?

允许，Milvus 不会对向量 ID（即 primary key）进行去重。

如果插入重复 ID（即 primary key）的向量，Milvus 是否会将其作为数据更新处理?

目前，Milvus 的去重功能无法保证插入新数据会覆盖与其 pk 相同的旧数据。因此，当前版本中基于相同 pk 的结构化匹配的返回结果为未知行为。该限制将在未来版本中修复。

Milvus 中自定义 ID （即 primary key）有没有长度限制?

Entity ID （即 primary key）必须是非负的 64 位整型。

Milvus 中单次插入数据有上限吗?

因 gRPC 限制，单次插入数据不能超过 1024 MB。

搜索指定 partition 时，如果所在的 collection 大小发生变化，是否对查询性能有影响?

不会。如果你在搜索时指定了 partition，Milvus 只会在相应 partition 进行搜索。

如果已指定仅搜索部分 partition，Milvus 会将整个 collection 的数据加载到内存吗?

不会。查询前需先保证数据已加载到内存。

- 如果明确知道当前数据所在 partition，可直接调用 `load_partition()` 方法加载指定 partition 的数据，然后调用 `search()` 方法并指定该 partition。
- 如果不确定数据所在 partition，那么在调用 `search()` 方法前需先调用 `load_collection()` 方法。
- 如果未在查询前加载 collection 或 partition 数据，Milvus 会报错。

Milvus 支持新增向量后再建索引吗?

支持。调用 `create_index()` 方法后，Milvus 会为后续新增向量自动构建索引的任务。每当新增数据量达到一个完整的 segment 时即触发这一任务，Milvus 为新插入的向量构建索引。

新增向量的索引文件与前期构建的索引文件相互独立。

Milvus 中 FLAT 索引和 IVF_FLAT 索引的原理比较?

IVF_FLAT 索引将向量空间分成 `nlist` 个聚类单元。假设以默认值 `nlist` = 16,384 搜索，Milvus 会先比较这 16384 个单元的中心与目标向量之间的距离，得出最近的 `nprobe` 个单元，接着比较这些单元内所有向量距离，得到最接近的向量。

FLAT 则计算每条向量和目标向量之间的距离。

当向量总条数约等于 `nlist` 时，两者的计算量相当，无明显性能差距。然而，随着向量条数达到 `nlist` 的 2 倍、3 倍、n 倍之后，IVF_FLAT 的性能优势就越来越突出。

可参阅 向量索引。

Milvus 的数据落盘逻辑是怎样的?

新增数据写入消息队列后，Milvus 即返回插入成功，表示当前插入操作已经结束，但是此时数据并未落盘。Milvus 系统的 data node 负责将消息队列中的数据以增量日志的方式写入持久化存储；如果调用 `flush()` 方法，也会迫使 data node 立刻将当前消息队列的所有数据写入持久化存储。

什么是归一化? Milvus 中为什么有时候需要归一化?

归一化指通过数学变换将向量的模长变为 1 的过程。如需使用点积计算向量相似度，则必须对向量作归一化处理。处理后点积与余弦相似度等价。

可参阅文章 向量搜索的简明数学基础。

为什么欧氏距离和内积在计算向量相似度时的结果不一致?

根据数学原理，对于已经归一化的向量数据，用欧氏距离和内积分别计算向量相似度，其返回的结果是一致的。

如果用欧氏距离和内积计算向量相似度返回的结果不一致，需要检查向量数据是否已经归一化。

Milvus 对 collection 和 partition 的总数有限制吗?

Milvus 对 collection 数量没有限制，但每个 collection 内的 partition 数量不能超过参数 `master.maxPartitionNum` 所设定的值。

为什么搜索 `topk` 条向量，但召回结果不足 k 条向量?

在 Milvus 支持的索引类型中，IVF_FLAT 和 IVF_SQ8 是基于 k-means 空间划分的分单元搜索算法。空间被分为 `nlist` 个单元，导入的向量被分配存储在基于 `nlist` 划分的文件结构中。Milvus 计算出距离最近的 `nprobe` 个单元，比较目标向量与选定单元中所有向量之间的距离，以返回最终结果。

如果 `nlist` 和 `topk` 比较大，而 `nprobe` 又足够小，就有可能出现 `nprobe` 个单元中的所有向量总数小于 k 的情况，导致返回结果不足 k 条向量。

想要避免这种情况，可以尝试将 `nprobe` 设置为更大值，或者把 `nlist` 和 `topk` 设置为更小值。

详见 向量索引。

Milvus 支持的向量维度的最大值是多少?

Milvus 最多支持 32768 维向量。

Milvus 是否支持 Apple M1 CPU?

当前版本 Milvus 暂不支持 Apple M1 CPU。

Milvus 支持何种 ID （即 primary key）field 数据类型?

在当前版本中，Milvus 仅支持 INT64 数据型。未来的 2.0 正式版将会同时支持 INT64 和 string 数据型。

Milvus 支持扩缩容吗?

支持。你可以通过 Helm Chart 在 Kubernetes 上部署多节点 Milvus 集群。更多相关说明，参考 扩缩容指南。

查询是否在内存中执行？什么是增量数据和历史数据?

是的。当收到查询请求时，Milvus 会将增量数据和历史数据共同加载至内存后进行搜索。增量数据来自尚未达到持久化阈值而缓存在内存中的 growing segment，而历史数据则来自已经持久化在对象存储中的 sealed segment。增量数据和历史数据共同构成了要搜索的整个数据集。

Milvus 2.0 支持并行搜索吗?

支持。对于同一个集合上的查询，Milvus 会并行查询增量数据和历史数据，而对不同集合上的查询则是串行的。但由于历史数据可能是一个非常庞大的数据集，对历史数据的搜索相对更耗时，所以本质上是串行的。Milvus 2.0 的正式版将改进这个问题。

为什么在对应的集合被删除后仍然保留 MinIO 中的数据?

为了方便数据回滚，MinIO 中的数据被设计为保留一定时间。

Milvus 是否支持 Pulsar 以外的消息引擎?

未来 Milvus 2.0 将支持 Kafka。

相似性搜索与结构性匹配有何区别?

在 Milvus 中，向量相似度搜索是通过计算向量相似度以及向量索引的加速来检索向量。与搜索不同，向量结构性匹配是通过标量过滤来匹配检索向量。布尔表达式通过过滤标量 field 或 primary key field，检索并返回所有与之匹配的结果。结构性匹配不涉及相似性计算以及向量索引。

仍有问题没有得到解答?

如果仍有其他问题，你可以：

- 访问我们的 GitHub 主页，与我们分享你的问题和想法，或帮助其他用户。
- 加入我们的 Slack 社区，参与开源社区的讨论交流。

## 运维问题

### 操作常见问题

#### 安装 Milvus 时，从 Docker Hub 拉取镜像失败怎么办?

如果无法从 Docker Hub 拉取镜像，可以尝试添加其它的镜像源。

中国大陆用户可以在文件 /etc/docker/daemon.json 中的`registry-mirrors`组添加国内镜像源地址`"https://registry.docker-cn.`
。

```
{
  "registry-mirrors": ["https://registry.docker-cn.com"]
}
```

#### Milvus 只能使用 Docker 部署吗?

使用 Docker 能高效部署 Milvus，但并不是唯一方式。Milvus 也支持从源码编译安装，但该方法仅支持 Ubuntu 系统（内核版本 18.04 或以上）和 CentOS 系统（内核版本 7 或以上）。详见 从源代码编译 Milvus。

#### 召回率主要受哪些因素影响?

召回率主要受索引类型和查询参数影响。

对于 FLAT 索引，Milvus 会在 collection 内做全量搜索，召回率为 100%。

对于 IVF 索引，`nprobe` 参数决定了搜索范围——`nprobe` 越大，搜索的数据比例越高，召回率也就越高，但查询性能会相应降低。

对于 HNSW 索引，`ef` 参数决定了导航图搜索的广度——`ef` 越大，图上扫描到的结点越多，召回率也就越高，但查询性能会相应降低。

详见 Milvus 索引类型。

#### 为什么配置文件更新后没有生效?

Milvus v2.0 暂不支持运行时动态修改配置文件。配置文件更新后，必须重启 Milvus Docker 让修改生效。

#### 如何得知我的 Milvus 已经成功启动?

如果通过 Docker Compose 启动 Milvus 服务，可运行 `docker ps` 命令观察运行中的 Docker 容器数量，以此判断 Milvus 服务是否已经启动。

- 对于单机版 Milvus，应至少有三个 Docker 容器正在运行，其中一个是 Milvus 服务，其余两个是 etcd 管理和存储服务。详见 安装单机版 Milvus。
- 对于分布式 Milvus，应至少有 12 个 Docker 容器正在运行，其中 9 个是 Milvus 服务，其余三个是基础服务。详见 安装分布式 Milvus。

#### 为什么日志文件时间与系统时间不一致?

日志文件时间与系统时间不一致通常是因为主机未使用 UTC 时间。

Docker 镜像内部的日志文件默认使用 UTC 时间。因此，如果主机未使用 UTC 时间，就会出现日志文件时间与系统时间不一致的情况。

#### 如何确认我的 CPU 支持 Milvus?

Milvus 在构建索引和向量查询时依赖于 CPU 对 SIMD (Single Instruction Multiple Data) 扩展指令集的支持。请确认运行 Milvus 的 CPU 至少支持以下 SIMD 指令集中的一种:

- SSE4.2
- AVX
- AVX2

- AVX512

可以使用 lscpu 命令来判断 CPU 是否支持特定 SIMD 指令集

```
$ lscpu | grep -e sse4_2 -e avx -e avx2 -e avx512
```

详见 CPU 对 SIMD 指令集的支持。

为什么 Milvus 在启动时返回 `Illegal instruction`?

要保证 Milvus 的正常运行，你的 CPU 须支持以下至少一种 SIMD 指令集种：SSE4.2、AVX、AVX2 和 AVX512。如果 Milvus 在启动时返回 `Illegal instruction`，说明当前 CPU 不支持以上任何一种指令集。

详见 CPU 对 SIMD 指令集的支持。

可以在 Windows 上安装 Milvus 吗?

可以。你可以通过源码编译或下载编译完成的二进制在 Windows 上安装 Milvus。

阅读 在 Windows 上运行 Milvus 2.0 学习如何在 Windows 上安装 Milvus。

在 Windows 安装 PyMilvus 报错，如何解决?

不建议在 Windows 安装 PyMilvus。如果必须在 Windows 安装 PyMilvus 且系统报错，你可以尝试在 Conda 环境下安装。阅读安装 SDK 了解如何安装 PyMilvus。

能否在内网离线环境中部署 Milvus 服务?

可以。你可以离线部署 Milvus 服务。阅读 离线安装 Milvus 了解如何在离线环境中部署 Milvus。

Milvus 日志打印在哪里?

Milvus 的日志默认输出在标准输出（standard output）和标准误差（standard error）流中，实际生产中建议用户重定向目录到持久卷已便于问题的排查。重定向日志需要修改 milvus.yaml 中的 `log.file.rootPath` 参数配置。如果你通过 `milvus-helm` chart 部署 Milvus，需要首先通过在安装时设定 `--set log.persistence.enabled=true` 以启用日志持久化。

Milvus 是否支持先建索引再插入数据?

支持。但我们推荐分批次插入数据，每次插入数据量不超过 256 MB，插入完成后统一创建索引。

仍有问题没有得到解答?

如果仍有其他问题，你可以：

- 访问我们的 GitHub 主页，与我们分享你的问题和想法，或帮助其他用户。
- 加入我们的 Milvus Forum 或 Slack 社区，参与开源社区的讨论交流。

## 故障诊断

故障诊断问题

本页列举了使用 Milvus 可能会遇到的常见问题及潜在解决方案，主要分为以下几类：

- 服务启动问题
- 服务运行问题
- API 问题

服务启动问题

服务启动时发生故障会导致服务无法正常启动。可运行以下命令查看相关错误信息：

```
1$ docker logs <your milvus container id>
```

服务运行问题

服务运行期间发生的故障可能导致服务宕机。如遇到此类故障，请先检查系统版本与所使用的客户端版本是否兼容，然后再查询相关错误信息。

API 问题

在 Milvus 服务端和客户端之间调用 API 方法时发生的故障。这类错误信息将以同步或异步的方式返回给客户端。

如有问题无法自行解决，你可以:

- 加入我们的 Slack 社区，获取 Milvus 团队的帮助。
- 在 GitHub 上 创建 issue 并提供相关问题的详细描述。