Milvus v2.0.0

Contents

About Milvus

What is Milvus

Introduction

This page aims to give you an overview of Milvus by answering several questions. After reading this page, you will learn what Milvus is and how it works, as well as the key concepts, why use Milvus, supported indexes and metrics, example applications, the architecture, and relevant tools.

What is Milvus vector database?

Milvus was created in 2019 with a singular goal: store, index, and manage massive embedding vectors generated by deep neural networks and other machine learning (ML) models.

As a database specifically designed to handle queries over input vectors, it is capable of indexing vectors on a trillion scale. Unlike existing relational databases which mainly deal with structured data following a pre-defined pattern, Milvus is designed from the bottom-up to handle embedding vectors converted from unstructured data.

As the Internet grew and evolved, unstructured data became more and more common, including emails, papers, IoT sensor data, Facebook photos, protein structures, and much more. In order for computers to understand and process unstructured data, these are converted into vectors using embedding techniques. Milvus stores and indexes these vectors. Milvus is able to analyze the correlation between two vectors by calculating their similarity distance. If the two embedding vectors are very similar, it means that the original data sources are similar as well.

Key concepts

In case you are new to the world of vector database and similarity search, read the following explanation of key concepts to gain a better understanding.

Learn more about Milvus glossary.

Unstructured data

Unstructured data, including images, video, audio, and natural language, is information that doesn't follow a predefined model or manner of organization. This data type accounts for ~80% of the world's data, and can be converted into vectors using various artificial intelligence (AI) and machine learning (ML) models.

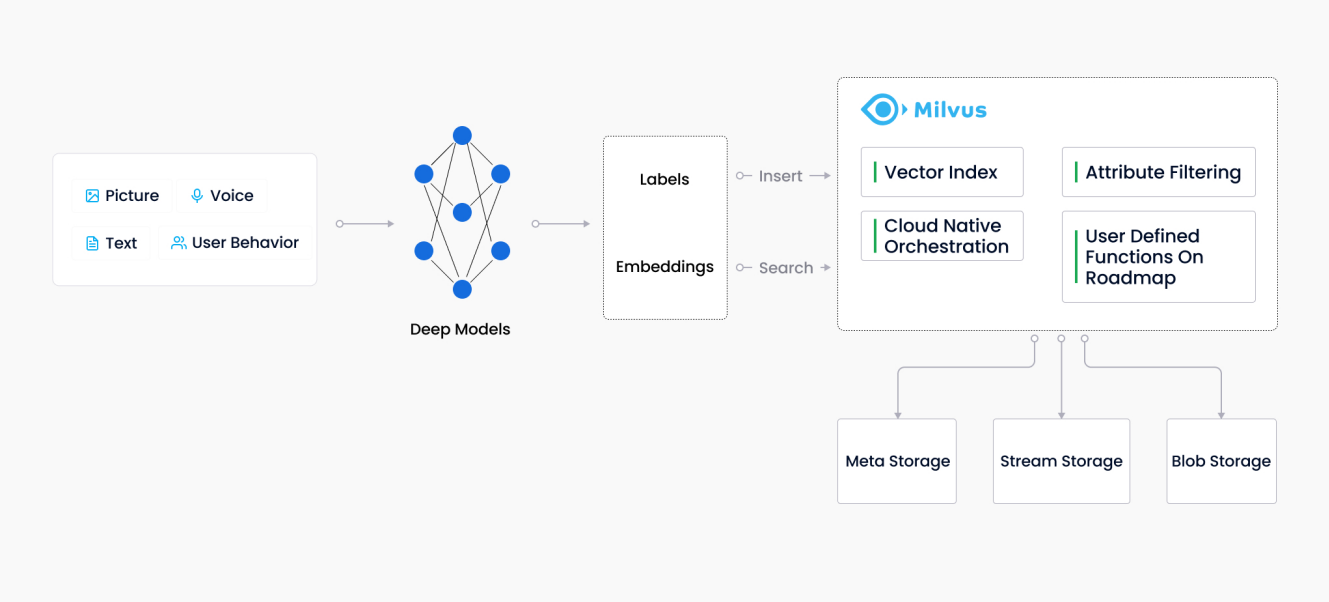


Figure 1: Workflow

Embedding vectors

An embedding vector is a feature abstraction of unstructured data, such as emails, IoT sensor data, Instagram photos, protein structures, and much more. Mathematically speaking, an embedding vector is an array of floating-point numbers or binaries. Modern embedding techniques are used to convert unstructured data to embedding vectors.

Vector similarity search

Vector similarity search is the process of comparing a vector to a database to find vectors that are most similar to the query vector. Approximate nearest neighbor (ANN) search algorithms are used to accelerate the searching process. If the two embedding vectors are very similar, it means that the original data sources are similar as well.

Why Milvus?

- High performance when conducting vector search on massive datasets.
- A developer-first community that offers multi-language support and toolchain.
- Cloud scalability and high reliability even in the event of a disruption.
- Hybrid search achieved by pairing scalar filtering with vector similarity search.

What indexes and metrics are supported?

Indexes are an organization unit of data. You must declare the index type and similarity metric before you can search or query inserted entities. If you do not specify an index type, Milvus will operate brute-force search by default.

Index types

Most of the vector index types supported by Milvus use approximate nearest neighbors search (ANNS), including:

- FLAT: FLAT is best suited for scenarios that seeks perfectly accurate and exact search results on a small, million-scale dataset.
- IVF_FLAT: IVF_FLAT is a quantization-based index and is best suited for scenarios that seeks an ideal balance between accuracy and query speed.
- IVF_SQ8: IVF_SQ8 is a quantization-based index and is best suited for scenarios that seeks a significant reduction on disk, CPU, and GPU memory consumption as these resources are very limited.

- IVF_PQ: IVF_PQ is a quantization-based index and is best suited for scenarios that seeks high query speed even at the cost of accuracy.
- HNSW: HNSW is a graph-based index and is best suited for scenarios that has a high demand for search efficiency.
- ANNOY: ANNOY is a tree-based index and is best suited for scenarios that seeks a high recall rate.

See Vector Index for more details.

Similarity metrics

In Milvus, similarity metrics are used to measure similarities among vectors. Choosing a good distance metric helps improve classification and clustering performance significantly. Depending on the input data forms, specific similarity metric is selected for optimal performance.

The metrics that are widely used for floating point embeddings include:

- Euclidean distance (L2): This metric is generally used in the field of computer vision (CV).
- Inner product (IP): This metric is generally used in the field of natural language processing (NLP). The metrics that are widely used for binary embeddings include:
- Hamming: This metric is generally used in the field of natural language processing (NLP).
- Jaccard: This metric is generally used in the field of molecular similarity search.
- Tanimoto: This metric is generally used in the field of molecular similarity search.
- Superstructure: This metric is generally used to search for similar superstructure of a molecule.
- Substructure: This metric is generally used to search for similar substructure of a molecule.

See Similarity Metrics for more information.

Example applications

Milvus makes it easy to add similarity search to your applications. Example applications of Milvus include:

- Image similarity search: Images made searchable and instantaneously return the most similar images from a massive database.
- Video similarity search: By converting key frames into vectors and then feeding the results into Milvus, billions of videos can be searched and recommended in near real time.
- Audio similarity search: Quickly query massive volumes of audio data such as speech, music, sound effects, and surface similar sounds.
- Molecular similarity search: Blazing fast similarity search, substructure search, or superstructure search for a specified molecule.
- Recommender system: Recommend information or products based on user behaviors and needs.
- Question answering system: Interactive digital QA chatbot that automatically answers user questions.
- DNA sequence classification: Accurately sort out the classification of a gene in milliseconds by comparing similar DNA sequence.
- Text search engine: Help users find the information they are looking for by comparing keywords against a database of texts.

See Milvus tutorials and Milvus Adopters for more Milvus application scenarios.

How is Milvus designed?

As a cloud-native vector database, Milvus 2.0 separates storage and computation by design. To enhance elasticity and flexibility, all components in Milvus 2.0 are stateless.

The system breaks down into four levels:

- Access layer: The access layer is composed of a group of stateless proxies and serves as the front layer of the system and endpoint to users.
- Coordinator service: The coordinator service assigns tasks to the worker nodes and functions as the system's brain.
- Worker nodes: The worker nodes function as arms and legs and are dumb executors that follow instructions from the coordinator service and execute user-triggerd DML/DDL commands.

- Storage: Storage is the bone of the system, and is responsible for data persistence. It comprises meta storage, log broker, and object storage.

For more information, see Architecture Overview.

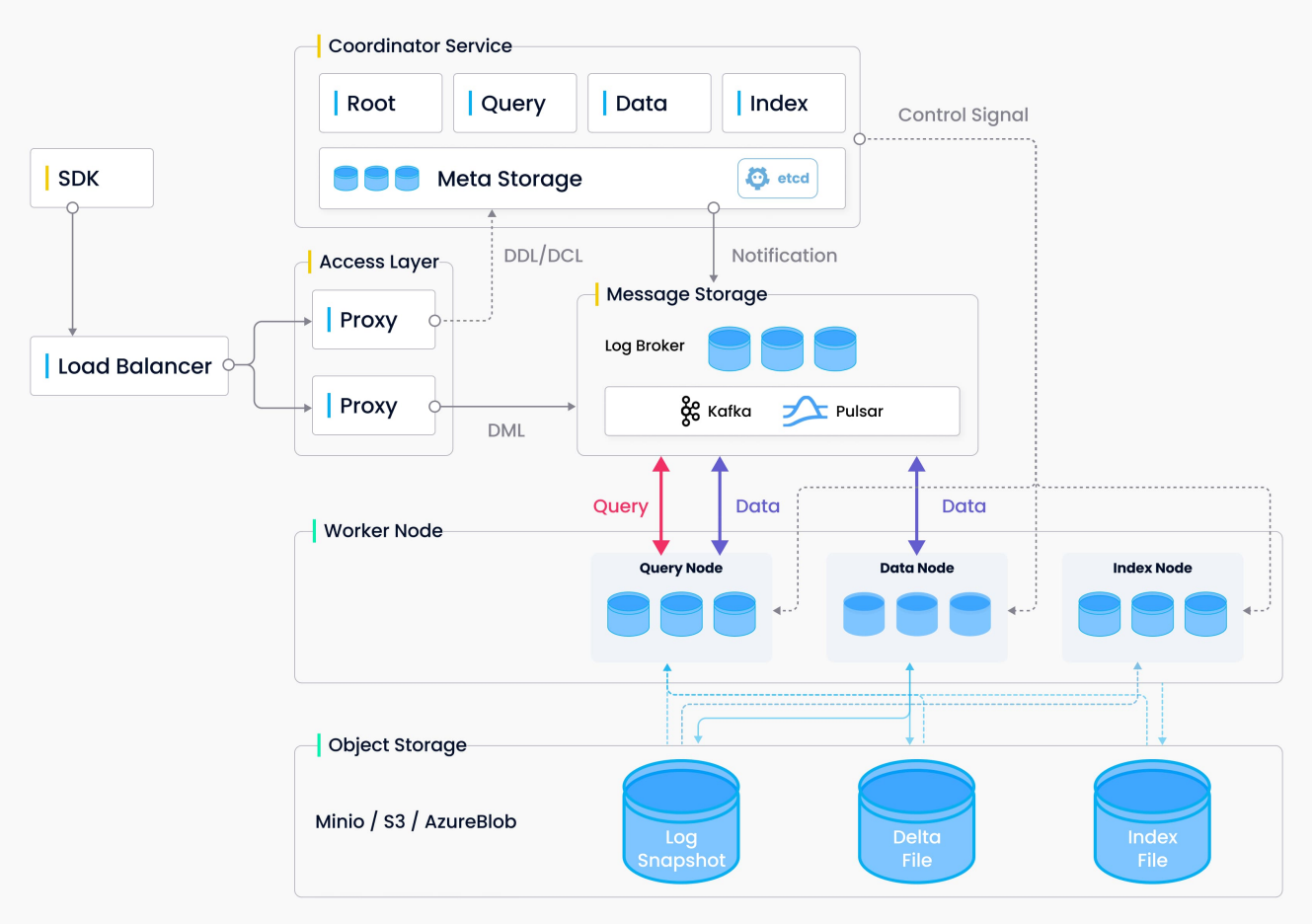


Figure 2: Architecture

Developer tools

Milvus is supported by rich APIs and tools to facilitate DevOps.

API access

Milvus has client libraries wrapped on top of the Milvus API that can be used to insert, delete, and query data programmatically from application code:

- PyMilvus
- Node.js SDK
- Go SDK

We are working on enabling more new client libraries. If you would like to contribute, go to the corresponding repository of the Milvus Project.

Milvus ecosystem tools

The Milvus ecosystem provides helpful tools including:

- Milvus CLI

- Attu, a graphical management system for Milvus.
- MilvusDM (Milvus Data Migration), an open-source tool designed specifically for importing and exporting data with Milvus.
- Milvus sizing tool, which helps you estimate the raw file size, memory size, and stable disk size needed for a specified number of vectors with various index types.

What's next

- Get started with a 3-minute tutorial:
  - Hello Milvus
- Install Milvus for your testing or production environment:
  - Installation Prerequisites
  - Install Milvus Standalone
  - Install Milvus Cluster
- If you're interested in diving deep into the design details of Milvus:
  - Read about Milvus architecture

What is New in Milvus 2.0

We recommend that you try out Milvus 2.0. Here is why:

Design concepts

As our next-generation cloud-native vector database, Milvus 2.0 is built around the following three principles:

Cloud-native first: We believe that only architectures supporting storage and computing separation can scale on demand and take full advantage of the cloud's elasticity. We'd also like to bring your attention to the microservice design of Milvus 2.0, which features read and write separation, incremental and historical data separation, and CPU-intensive, memory-intensive, and IO-intensive task separation. Microservices help optimize allocation of resources for the ever-changing heterogeneous workload.

Logs as data: In Milvus 2.0, the log broker serves as the system's backbone: All data insert and update operations must go through the log broker, and worker nodes execute CRUD operations by subscribing to and consuming logs. This design reduces system complexity by moving core functions such as data persistence and flashback down to the storage layer, and log pub-sub make the system even more flexible and better positioned for future scaling.

Unified batch and stream processing: Milvus 2.0 implements the unified Lambda architecture, which integrates the processing of the incremental and historical data. Compared with the Kappa architecture, Milvus 2.0 introduces log backfill, which stores log snapshots and indexes in the object storage to improve failure recovery efficiency and query performance. To break unbounded (stream) data down into bounded windows, Milvus embraces a new watermark mechanism, which slices the stream data into multiple message packs according to write time or event time, and maintains a timeline for users to query by time.

Product highlights

The costs of running a database involve not only runtime resource consumption, but also the potential learning costs and the operational and maintenance costs. Practically speaking, the more user-friendly a database is, the more likely it is going to save such potential costs. From Milvus' calendar day one, ease of use is always put on the top of our list, and the latest Milvus 2.0 has quite a few to offer in the way of reducing such costs.

Always online

Data reliability and service sustainability are the basic requirements for a database, and our strategy is "fail cheap, fail small, and fail often".
- "Fail cheap" refers to storage and computing separation, which makes the handling of node failure recovery straightforward and at a low cost. - "Fail small" refers to the "divide and conquer" strategy, which simplifies the design complexity by having each coordinator service handle only a small portion of read/write/incremental/historical data. - "Fail often" refers to the introduction of chaos testing, which uses fault injection in a testing environment to simulate situations such as hardware failures and dependency failures and accelerate bug discovery.

Hybrid search between scalar and vector data

To leverage the synergy between structured and unstructured data, Milvus 2.0 supports both scalar and vector data and enables hybrid search between them. Hybrid search helps users find the approximate nearest neighbors that match filter criteria. Currently, Milvus supports relational operations such as EQUAL, GREATER THAN, and LESS THAN, and logical operations such as NOT, AND, OR, and IN.

Tunable consistency

As a distributed database abiding by the PACELC theorem, Milvus 2.0 has to trade off between consistency and availability & latency. In most scenarios, overemphasizing data consistency in production can be overkill because allowing a small portion of data to be invisible has little impact on the overall recall but can significantly improve the query performance. Still, we believe that consistency levels, such as strong, bounded staleness, and session, have their own unique application. Therefore, Milvus supports tunable consistency at the request level. Taking testing as an example, users may require strong consistency to ensure test results are absolutely correct.

Time travel

Data engineers often need to do data rollback to fix dirty data and code bugs. Traditional databases usually implement data rollback through snapshots or even data retrain. This could bring excessive overhead and maintenance costs. Milvus maintains a timeline for all data insert and delete operations, and users can specify a timestamp in a query to retrieve a data view at a specified point in time. With time travel, Milvus can also implement a lightweight data backup or data clone.

ORM Python SDK:

Object-relational mapping (ORM) allows users to focus more on the upper-level business model than on the underlying data model, making it easier for developers to manage relations between collections, fields, and programs. To close the gap between proof of concept (PoC) for AI algorithms and production deployment, we engineered the object-relational mapping PyMilvus APIs, which can work with an embedded library, a standalone deployment, a distributed cluster, or even a cloud service. With a unified set of APIs, we provide users with a consistent user experience and reduce code migration or adaptation costs.

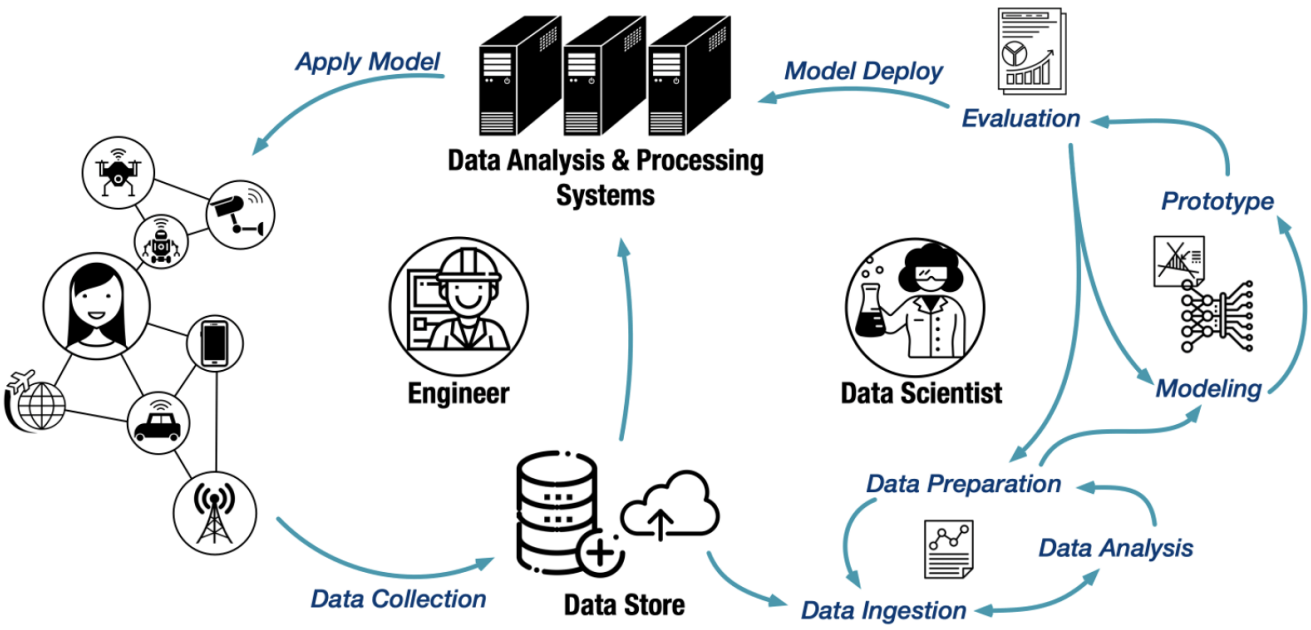


Figure 3: ORM_Python_SDK

Support tools

- Attu is Milvus' graphical user interface offering practical functionalities such as cluster state management, meta management, and data query. The source code of Attu will also be open sourced as an independent project. We are looking for more contributors to join this effort.
- Milvus CLI is Milvus' command-line interface based on Milvus Python SDK, supporting database connection, data operations, and data export/import.
- Out-of-box experience (OOBE), faster deployment: Milvus 2.0 can be deployed using helm or Docker Compose.
- Milvus 2.0 uses Prometheus, an open-source time-series database, to store performance and monitor data, and Grafana, an open observability platform, for metrics visualization.

Milvus 2.0 vs. 1.x: Cloud-native, distributed architecture, highly scalable, and more

Milvus 2.0

Milvus 1.x

```
<tr>
    <th>Architecture</th>
    <td>Cloud native</td>
    <td>Shared storage</td>
</tr>
```

Scalability

500+ nodes

1 to 32 read nodes with only one write node

Durability

Object storage service (OSS)

Distributed file system (DFS)

Local disk

Network file system (NFS)

Availability

99.9%

99%

Data consistency

Three levels of consistency:

Strong

Bounded Staleness

Session

Consistent prefix

Eventual consistency

Data types supported

Vectors

Fixed-length scalars

String and text (in planning)

Vectors

Basic operations supported

Data insertion

Data deletion (in planning)

Data query

Approximate nearest neighbor (ANN) Search

Recurrent neural network (RNN) search (in planning)

Data insertion

Data deletion

Approximate nearest neighbor (ANN) Search

Advanced features

Scalar filtering

Time Travel

Multi-site deployment and multi-cloud integration

Data management tools

Mishards

Milvus DM

Index types and libraries

Faiss

Annoy

Hnswlib

RNSG

ScaNN (in planning)

On-disk index (in planning)

Faiss

Annoy

Hnswlib

RNSG

SDKs

Python

Node.js

Go (in planning)

Java (in planning)

C++ (in planning)

Python

Java

Go

RESTful

C++

Release status

Release candidate. A stable version will be released in August.

Long-term support (LTS)

Milvus Adopters

Trusted by over 1,000 organizations worldwide, Milvus is used in almost all industries. The following table lists companies that have adopted Milvus in production.

Company	Industry	User story
ambeRoad	Software	
Beike	Real estate	Making With Milvus: AI-Infused Proptech for Personalized Real Estate Search
Deepset.ai	Software	Semantic Search with Milvus, Knowledge Graph QA, Web Crawlers and more!
DXY.cn	Medical	
eBay	Online retail	
EnjoyMusic Technology	Entertainment	Item-based Collaborative Filtering for Music Recommender System
Getir	Instant grocery	
Hewlett-Packard (HP)	Information technology	
IOMED	Clinical research	
iYUNDONG	Sports	Extracting Event Highlights Using iYUNDONG Sports App
Juicedata	Software	Building a Milvus Cluster Based on JuiceFS
Kingsoft	Software	Building an AI-Powered Writing Assistant for WPS Office
Line Plus	Technology, software	
Lucidworks	Software	Build Semantic Search at Speed
Meetsocial Group	Marketing	
Miao Health	Health technology	
Mozat	Online social	Building a Wardrobe and Outfit Planning App with Milvus
Opera	Software	
Shopee	Online retail	
Smartnews	Media	
Sohu	Internet	Building an Intelligent News Recommendation System Inside Sohu News App
The Cleveland Museum of Art	Arts	ArtLens AI: Share Your View
Tokopedia	Online retail	How we used semantic search to make our search 10x smarter
TrendMicro	Software	Making with Milvus: Detecting Android Viruses in Real Time for Trend Micro
Ufoto	Software	
Vipshop	Online retail	Building a Personalized Product Recommender System with Vipshop and Milvus
Vova	Online retail	Building a Search by Image Shopping Experience with VOVA and Milvus

Company	Industry	User story
Walmart	Retail	
Xiaomi	Consumer electronics	Making with Milvus: AI-Powered News Recommendation Inside Xiaomi’ s Mobile Browser

Milvus Limits

Milvus is committed to providing the best vector databases to power AI applications and vector similarity search. However, the team is continuously working to bring in more features and the best utilities to enhance user experience. This page lists out some known limitations that the users may encounter when using Milvus.

Length of a resource name

Resource	Limit
Collection	255 characters
Field	255 characters
Index	255 characters
Partition	255 characters

Naming rules

The name of a resource can contain numbers, letters, dollar signs ($), and underscores (_). A resource name must start with a letter or an underscore (_).

Number of resources

Resource	Limit
Collection	65,536
Connection / proxy	65,536

Number of resources in a collection

Resource	Limit
Partition	4,096
Shard	256
Field	256
Index	1
Entity	unlimited

Length of a string

Data type	Limit
VARCHAR	65,535 characters

VARCHAR and more string data types will be supported in future releases of Milvus.

Dimensions of a vector

Property	Limit
Dimension	32,768

Input and Output per RPC

Operation	Limit
Insert	512 MB
Search	512 MB
Query	512 MB

Load limits

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

Search limits

Vectors	Limit
topk (number of the most similar result to return)	16,384
nq (number of the search requests)	16,384

Releases

Release Notes

Find out what' s new in Milvus! This page summarizes information about new features, improvements, known issues, and bug fixes in each release. You can find the release notes for each released version after v2.0.0-RC1 in this section. We suggest that you regularly visit this page to learn about updates.

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

Release date: 2021-12-31

Compatibility

Milvus version

Python SDK version

Java SDK version

Go SDK version

Node.js SDK version

2.0.0-PreGA

2.0.0rc9

2.0.0

Coming soon

1.0.20

Milvus 2.0.0-PreGA is the preview release of Milvus 2.0. It now supports entity deletion by primary key and data compaction to purge deleted data. We also introduce a load balancing mechanism into Milvus to distribute the memory usage of each query node evenly. Some critical issues are fixed in this release, including cleanup of dropped collection data, wrong distance calculation of Jaccard distance, and several bugs that cause system hang and memory leakage.

It should be noted that Milvus 2.0.0-PreGA is NOT compatible with previous versions of Milvus 2.0 because of some changes made to data codec format and RocksMQ data format.

Features

- Deleting entity: Milvus now supports deleting entities through primary keys. Whereas Milvus relies on append-only storage, it only supports logical deletion, id est, Milvus inserts a deletion mark on the entities to cover actual data so that no search or query will return the marked entities. Therefore, it should be noted that overusing deletion may cause search performance to plummet and storage usage to surge. See Delete entities for more instruction.
- Compaction: Compaction mechanism purges the deleted or expired entities in binlogs to save storage space. It is a background task that is triggered by data coord and executed by data node.
- Automatic Loadbalance #9481：Loadbalance mechanism distributes segments evenly across query nodes to balance the memory usage of the cluster. It can be triggered either automatically or by users.
- Handoff #9481：Handoff mechanism refers to that, when a growing segment is sealed, query node waits until the segment is built with index by index node and then loads the segment into memory for search or query.

Improvements

- #12199 Parallelizes executions between segments to improve the search performance.
- #11373 Allows batch consumption of messages in RocksMQ internal loop to improve the system efficiency.
- #11665 Postpones the execution of handoff until index creation is completed.

Bug fixes

- Data is not cleared on etcd, Pulsar, and MinIO when a collection is dropped:
  - #12191 Clears the metadata of the dropped segment on etcd.
  - #11554 Adds garbage collector for data coord.
  - #11552 Completes procedure of dropping collection in data node.
  - #12227 Removes all index when dropping collection.
  - #11436 Changes the default retentionSizeInMB to 8192 (8GB).
- #11901 Wrong distances calculation caused by properties of different metric types.
- #12511 Wrong similarity correlation caused by properties of different metric types.
- #12225 RocksMQ produce hang when do search repeatedly
- #12255 RocksMQ server does not close when standalone exits.
- #12281 Error when dropping alias.
- #11769 Update serviceableTime incorrectly.
- #11325 Panic when reducing search results.
- #11248 Parameter guarantee_timestamp is not working.

Other Enhancements

- #12351 Changes proxy default RPC transfer limitation.
- #12055 Reduces memory cost when loading from MinIO.
- #12248 Supports more deployment metrics.
- #11247 Adds getNodeInfoByID and getSegmentInfoByNode function for cluster.
- #11181 Refactors segment allocate policy on query coord.

v2.0.0-RC8

Release date: 2021-11-5

Compatibility

Milvus version

Python SDK version

Java SDK version

Go SDK version

Node.js SDK version

2.0.0-RC8

2.0.0rc8

Coming soon

Coming soon

2.0.0

Milvus 2.0.0-RC8 is the last release candidate of Milvus 2.0. It supports handoff task, primary key deduplication and search by Time Travel functionalities. The mean time to recovery (MTTR) has also been greatly reduced with the enhancement of timetick mechanism. We had run stress test on 2.0.0-RC8 with 10M datasets, and both standalone and distributed cluster survived for 84 hours.

Current deduplication feature does not guarantee overwriting the old data copies when data with duplicate primary keys (`pk`) are inserted. Therefore, which data copy will return when queried remains unknown behavior. This limitation will be fixed in future releases.

Improvements

- Failure Recovery speed:
  - #10737 Fixes Session checker for proxy.
  - #10723 Fixes seek query channel error.
  - #10907 Fixes `LatestPosition` option conflict with earliest patch.
  - #10616 Removes Common YAML.
  - #10771 Changes `SeekPosition` to the earliest of all segments.
  - #10651 Fixes query coord set seek position error.
  - #9543 Initializes global sealed segments and seek query channel when `AddQueryChannel`.
  - #9684 Skips re-consuming timetick MsgStream when data coord restarts.
- Refactor meta snapshot:
  - #10288 Reduces information saved in `SnapshotMeta`.
  - #10703 Fixes failure when creating meta table because of compatibility issue.
  - #9778 Simplifies `meta_snapshot` interface.
- #10563 Changes default balance policy.
- #10730 Returns segment state when getting query segment information.
- #10534 Supports reading MinIO configuration from environment variables.
- #10114 Sets default `gracefulTime` to `0`.

- #9860 Hides `liveChn` into `sessionutil` and fix liveness initialization order.
- #7115 Uses etcd to watch channel on data node.
- #7606 Makes `knowhere` compile independently.

Features

- Handoff:
  - #10330 Adds `handoffTask`.
  - #10084 Broadcasts `sealedSegmentChangeInfo` to `queryChannel`.
  - #10619 Fixes removing segment when query node receives `segmentChangeInfo`.
  - #10045 Watches `changeInfo` in query node.
  - #10011 Updates excluded segments info when receiving `changeInfo`.
  - #9606 Adds initialization information for `AddQueryChannelRequest`.
  - #10619 Fixes removing segment when query node receives `segmentChangeInfo`.
- Primary Deduplication:
  - #10834 Removes primary key duplicated query result in query node.
  - #10355 Removes duplicated search results in proxy.
  - #10117 Removes duplicated search results in segcore reduce.
  - #10949 Uses primary key only to check search result duplication.
  - #10967 Removes primary key duplicated query result in proxy.
- Auto-flush:
  - #10659 Adds `injectFlush` method for `flushManager` interface.
  - #10580 Adds injection logic for `FlushManager`.
  - #10550 Merges automatic and manual flush with same segment ID.
  - #10539 Allows flushed segments to trigger flush process.
  - #10197 Adds a timed flush trigger mechanism.
  - #10142 Applies flush manager logic in data node.
  - #10075 Uses single signal channel to notify flush.
  - #9986 Adds flush manager structure.
- #10173 Adds binlog iterators.
- #10193 Changes bloom filter use primary key.
- #9782 Adds `allocIDBatch` for data node allocator.

Bug fixes

- Incorrect collection loading behavior if there is not enough memory:
  - #10796 Fixes get container mem usage.
  - #10800 Uses `TotalInactiveFile` in `GetContainerMemUsed`.
  - #10603 Increases compatibility for `EstimateMemorySize` interface.
  - #10363 Adds `cgroups` to get container memory and check index memory in segment loader.
  - #10294 Uses proto size to calculate request size.
  - #9688 Estimates memory size with descriptor event.
  - #9681 Fixes the way that binlog stores the original memory size.
  - #9628 Stores original memory size of binlog file to extra information.
- Size of etcd-related request is too large:
  - #10909 Fixes too many operations in `txn` request when saving `segmentInfo`.
  - #10812 Fixes too large request when loading segment.
  - #10768 Fixes too large request when loading collection.
  - #10655 Splits watch operations into many transactions.

  - #10587 Compacts `multiSegmentChangeInfo` to a single info.
  - #10425 Trims `segmentinfo` binlog for `VChaninfo` usage.
  - #10340 Fixes `multiSave childTask` failed to etcd.
  - #10310 Fixes error when assigning load segment request.
  - #10125 Splits large `loadSegmentReq` to multiple small requests.
- System panics:
  - #10832 Adds query `mutex` to fix crash with panic.
  - #10821 Index node finishes the task before index coord changed the meta.
  - #10182 Fixes panic when flushing segment.
  - #10681 Fixes query coord panic when upgrading `querychannelInfo`.
- RocksMQ-related issues:
  - #10367 Stops retention gracefully.
  - #9828 Fixes retention data race.
  - #9933 Changes retention ticker time to 10 minutes.
  - #9694 Deletes messages before deleting metadata in rocksmq retention.
  - #11029 Fixes rocksmq `SeekToLatest`.
  - #11057 Fixes `SeekToLatest` memory leakage and remove redundant logic.
  - #11081 Fixes rocksdb retention ts not set.
  - #11083 Adds topic lock for rocksmq `Seek`.
  - #11076 Moves topic lock to the front of final delete in retention expired cleanup.
- #10751 `loadIndex` keep retrying when `indexFilePathInfo` gets empty list.
- #10583 `ParseHybridTs` returns type to INT64.
- #10599 Delete message hash error.
- #10314 Index building task mistakenly canceled by index coord by mistake.
- #9701 Incorrect `CreateAlias/DropAlias/AlterAlias` implementation.
- #9573 Timeout when data coord saves binlog.
- #9788 Watch Channel canceled due to revision compacted.
- #10994 Index node does not balances load.
- #11152 Search is wrong when using Time Travel without filtering condition and call `num_entities`.
- #11249 #11277 Release collection block in query node.
- #11222 Incorrect empty retrieve result handling.

v2.0.0-RC7

Release date: 2021-10-11

Compatibility

Milvus version

Python SDK version

Java SDK version

Go SDK version

Node.js SDK version

2.0.0-RC7

2.0.0rc7

Coming soon

Coming soon

2.0.0

Milvus 2.0.0-RC7 is a preview version of Milvus 2.0. It supports collection alias, shares `msgstream` on physical channel, and changes the default MinIO and Pulsar dependencies to cluster version. Several resource leaks and deadlocks were fixed.

It should be noted that Milvus 2.0.0-RC7 is NOT compatible with previous versions of Milvus 2.0 because of some changes made to storage format.

Improvements

- #8215 Adds max number of retries for `interTask` in query coord.
- #9459 Applies collection start position.
- #8721 Adds Node ID to Log Name.
- #8940 Adds streaming segments memory to used memory in `checkLoadMemory`.
- #8542 Replaces `proto.MarshalTextString` with `proto.Marshal`.
- #8770 Refactors flowgraph and related invocation.
- #8666 Changes CMake version.
- #8653 Updates `getCompareOpType`.
- #8697 #8682 #8657 Applies collection start position when opening segment.
- #8608 Changes segment replica structure.
- #8565 Refactors buffer size calculation.
- #8262 Adds `segcore` logger.
- #8138 Adds `BufferData` in `insertBufferNode`.
- #7738 Implements allocating `msgstream` from pool when creating collections.
- #8054 Improves codes in `insertBufferNode`.
- #7909 Upgrades `pulsar-client-go` to 0.6.0.
- #7913 Moves segcore rows_per_chunk configuration to query_node.yaml.
- #7792 Removes `ctx` from `LongTermChecker`.
- #9269 Changes `==` to `is` when comparing to None in expression.
- #8159 Make `FlushSegments` async.
- #8278 Refactor rocksmq close logic and improve codecov.
- #7797 Uses definitional type instead of raw type.

Features

- #9579 Uses replica memory size and `cacheSize` in `getSystemInfoMetrics`.
- #9556 Adds `ProduceMark` interface to return message ID.
- #9554 Supports `LoadPartial` interface for DataKV.
- #9471 Supports `DescribeCollection` by collection ID.
- #9451 Stores index parameters to descriptor event.
- #8574 Adds a `round_decimal` parameter for precision control to search function.
- #8947 Rocksmq supports `SubscriptionPositionLatest`.
- #8919 Splits blob into several string rows when index file is large.

- #8914 Binlog parser tool supports index files.
- #8514 Refactors the index file format.
- #8765 Adds `cacheSize` to prevent OOM in query node.
- #8673 #8420 #8212 #8272 #8166 Supports multiple Milvus clusters sharing Pulsar and MinIO.
- #8654 Adds `BroadcastMark` for `Msgstream` returning Message IDs.
- #8586 Adds Message ID return value into producers.
- #8408 #8363 #8454 #8064 #8480 Adds session liveness check.
- #8264 Adds description event extras.
- #8341 Replaces `MarshalTextString` with `Marshal` in root coord.
- #8228 Supports healthz check API.
- #8276 Initializes the SIMD type when initializing an index node.
- #7967 Adds knowhere.yaml to support knowhere configuration.
- #7974 Supports setting max task number of task queue.
- #7948 #7975 Adds `suffixSnapshot` to implement SnapshotKV.
- #7942 Supports configuring SIMD type.
- #7814 Supports bool field filter in search and query expression.
- #7635 Supports setting segcore rows_per_chunk via configuration file.

Bug fixes

- #9572 Rocksdb does not delete the end key after `DeleteRange` is called.
- #8735 Acked infomation takes up memory resources.
- #9454 Data race in query service.
- #8850 SDK raises error with a message about index when dropping collection by alias.
- #8930 Flush occasionally gets stuck when `SaveBinlogPath` fails due to instant buffer removal from `insertBuf`.
- #8868 Trace log catches the wrong file name and line number.
- #8844 `SearchTask` result is nil.
- #8835 Root coord crashes because of bug in pulsar-client-go.
- #8780 #8268 #7255 Collection alias-related issues.
- #8744 Rocksdb_kv error process.
- #8752 Data race in mqconsumer.
- #8686 Flush after auto-flush will not finish.
- #8564 #8405 #8743 #8798 #9509 #8884 rocksdb memory leak.
- #8671 Objects are not removed in MinIO when dropped.
- #8050 #8545 #8567 #8582 #8562 tsafe-related issues.
- #8137 Time goes backward because TSO does not load last timestamp.
- #8461 Potential data race in data coord.
- #8386 Incomplete logic when allocating dm channel to data node.
- #8206 Incorrect reduce algorithm in proxy search task.
- #8120 Potential data race in root coord.

- #8068 Query node crashes when query result is empty and optional `retrieve_ret_` is not initialized.
- #8060 Query task panicking.
- #8091 Data race in proxy gRPC client.
- #8078 Data race in root coord gRPC client.
- #7730 Topic and ConsumerGroup remain after `CloseRocksMQ`.
- #8188 Logic error in releasing collections.

v2.0.0-RC6

Release date: 2021-09-10

Compatibility

Milvus version

Python SDK version

Java SDK version

Go SDK version

Node.js SDK version

2.0.0-RC6

2.0.0rc6

Coming soon

Coming soon

2.0.0

Milvus 2.0.0-RC6 is a preview version of Milvus 2.0. It supports specifying shard number when creating collections, and query by expression. It exposes more cluster metrics through API. In RC6 we increase the unit test coverage to 80%. We also fixed a series of issues involving resource leakage, system panic, etc.

Improvements

- Increases unit test coverage to 80%.

Features

- #7482 Supports specifying shard number when creating a collection.
- #7386 Supports query by expression.
- Exposes system metrics through API:
  - #7400 Proxy metrics integrate with other coordinators.
  - #7177 Exposes metrics of data node and data coord.
  - #7228 Exposes metrics of root coord.
  - #7472 Exposes more detailed metrics information.
  - #7436 Supports caching the system information metrics.

Bug fixes

- #7434 Query node OOM if loading a collection that beyond the memory limit.
- #7678 Standalone OOM when recovering from existing storage.
- #7636 Standalone panic when sending message to a closed channel.
- #7631 Milvus panic when closing flowgraph.
- #7605 Milvus crashed with panic when running nightly CI tests.
- #7596 Nightly cases failed because rootcoord disconnected with etcd.
- #7557 Wrong search result returned when the term content in expression is not in order.
- #7536 Incorrect `MqMsgStream` Seek logic.
- #7527 Dataset's memory leak in `knowhere` when searching.

- #7444 Deadlock of channels time ticker.
- #7428 Possible deadlock when `MqMsgStream` broadcast fails.
- #7715 Query request overwritten by concurrent operations on the same slice.

v2.0.0-RC5

Release date: 2021-08-30

Compatibility

Milvus version

Python SDK version

Java SDK version

Go SDK version

Node.js SDK version

2.0.0-RC5

2.0.0rc5

Coming soon

Coming soon

2.0.0

Milvus 2.0.0-RC5 is a preview version of Milvus 2.0. It supports message queue data retention mechanism and etcd data cleanup, exposes cluster metrics through API, and prepares for delete operation support. RC5 also made great progress on system stability. We fixed a series of resource leakage, operation hang and the misconfiguration of standalone Pulsar under Milvus cluster.

Improvements

- #7226 Refactors data coord allocator.
- #6867 Adds connection manager.
- #7172 Adds a seal policy to restrict the lifetime of a segment.
- #7163 Increases the timeout for gRPC connection when creating index.
- #6996 Adds a minimum interval for segment flush.
- #6590 Saves binlog path in `SegmentInfo`.
- #6848 Removes `RetrieveRequest` and `RetrieveTask`.
- #7102 Supports vector field as output.
- #7075 Refactors `NewEtcdKV` API.
- #6965 Adds channel for data node to watch etcd.
- #7066 Optimizes search reduce logics.
- #6993 Enhances the log when parsing gRPC recv/send parameters.
- #7331 Changes context to correct package.
- #7278 Enables etcd auto compaction for every 1000 revision.
- #7355 Clean `fmt.Println`in util/flowgraph.

Features

- #7112 #7174 Imports an embedded etcdKV (part 1).
- #7231 Adds a segment filter interface.
- #7157 Exposes metrics of index coord and index nodes.
- #7137 #7157 Exposes system topology information by proxy.
- #7113 #7157 Exposes metrics of query coord and query nodes.
- #7134 Allows users to get vectors using memory instead of local storage.
- #6617 Supports retention for rocksmq.
- #7303 Adds query node segment filter.
- #7304 Adds `delete` API into proto.
- #7261 Adds delete node.

- #7268 Constructs Bloom filter when inserting.

Bug fixes

- #7272 #7352 #7335 Failure to start new docker container with existing volumes if index was created: proxy is not healthy.
- #7243 Failure to create index in a new version of Milvus for data that were inserted in an old version.
- #7253 Search gets empty results after releasing a different partition.
- #7244 #7227 Proxy crashes when receiving empty search results.
- #7203 Connection gets stuck when gRPC server is down.
- #7188 Incomplete unit test logics.
- #7175 Unspecific error message returns when calculating distances using collection IDs without loading.
- #7151 Data node flowgraph does not close caused by missing `DropCollection`.
- #7167 Failure to load IVF_FLAT index.
- #7123 Timestamp go back for `timeticksync`.
- #7140 `calc_distance` returns wrong results for binary vectors when using TANIMOTO metrics.
- #7143 The state of memory and etcd is inconsistent if KV operation fails.
- #7141 #7136 Index building gets stuck when the index node pod is frequently killed and pulled up.
- #7119 Pulsar `msgStream` may get stuck when subscribed with the same topic and sub name.
- #6971 Exception occurs when searching with index (HNSW).
- #7104 Search gets stuck if query nodes only load sealed segment without watching insert channels.
- #7085 Segments do not auto flush.
- #7074 Index nodes wait for index coord to start to complete.
- #7061 Segment allocation does not expire if data coord does not receive timetick message from data node.
- #7059 Query nodes get producer leakage.
- #7005 Query nodes do not return error to query coord when `loadSegmentInternal` fails.
- #7054 Query nodes return incorrect IDs when `topk` is larger than `row_num`.
- #7053 Incomplete allocation logics.
- #7044 Lack of check on unindexed vectors in memory before retriving vectors in local storage.
- #6862 Memory leaks in flush cache of data node.
- #7346 Query coord container exited in less than 1 minute when re-installing Milvus cluster.
- #7339 Incorrect expression boundary.
- #7311 Collection nil when adding query collection.
- #7266 Flowgraph released incorrectly.
- #7310 Excessive timeout when searching after releasing and loading a partition.
- #7320 Port conflicts between embedded etcd and external etcd.
- #7336 Data node corner cases.

v2.0.0-RC4

Release date: 2021-08-13

Compatibility

Milvus version	Python SDK version	Java SDK version	Go SDK version
2.0.0-RC4	2.0.0rc4	Coming soon	Coming soon

Milvus 2.0.0-RC4 is a preview version of Milvus 2.0. It mainly focuses on fixing stability issues, it also offers functionalities to retrieve vector data from object storage and specify output field by wildcard matching.

Improvements

- #6984 #6772 #6704 #6652 #6536 #6522 Unit test improvements.
- #6859 Increases the `MaxCallRecvMsgSize` and `MaxCallSendMsgSize` of gRPC client.
- #6796 Fixes MsgStream exponential retry.
- #6897 #6899 #6681 #6766 #6768 #6597 #6501 #6477 #6478 #6935 #6871 #6671 #6682 Log improvements.

- #6440 Refactors segment manager.
- #6421 Splits raw vectors to several smaller binlog files when creating index.
- #6466 Separates the idea of query and search.
- #6505 Changes `output_fields` to `out_fields_id` for RetrieveRequest.
- #6427 Refactors the logic of assigning tasks in index coord.
- #6529 #6599 Refactors the snapshot of timestamp statistics.
- #6692 #6343 Shows/Describes collections/partitions with created timestamps.
- #6629 Adds the WatchWithVersion interface for etcdKV.
- #6666 Refactors expression executor to use single bitsets.
- #6664 Auto creates new segments when allocating rows that exceeds the maximum number of rows per segment.
- #6786 Refactors `RangeExpr` and `CompareExpr`.
- #6497 Looses the lower limit of dimension when searching on a binary vector field.

Features

- #6706 Supports reading vectors from disk.
- #6299 #6598 Supports query vector field.
- #5210 Extends the grammar of Boolean expressions.
- #6411 #6650 Supports wildcards and wildcard matching on search/query output fields.
- #6464 Adds a vector chunk manager to support vector file local storage.
- #6701 Supports data persistence with docker compose deployments.
- #6767 Adds a Grafana dashboard .json file for Milvus.

Bug fixes

- #5443 `CalcDistance` returns wrong results when fetching vectors from collection.
- #7004 Pulsar consumer causes goroutine leakage.
- #6946 Data race occurs when a flow graph `close()` immediately after `start()`.
- #6903 Uses proto marshal instead of marshalTextString in querycoord to avoid crash triggered by unknown field name crash.
- #6374 #6849 Load collection failure.
- #6977 Search returns wrong limit after a partition or collection is dropped.
- #6515 #6567 #6552 #6483 Data node BackGroundGC does not work and causes memory leak.
- #6943 The MinIOKV `GetObject` method does not close client and causes goroutine leaking per call.
- #6370 Search is stuck due to wrong semantics offered by load partition.
- #6831 Data node crashes in meta service.
- #6469 Search binary results are wrong with metrics of Hamming when limit (topK) is bigger than the quantity of inserted entities.
- #6693 Timeout caused by segment race condition.
- #6097 Load hangs after frequently restarting query node within a short period of time.
- #6464 Data sorter edge cases.
- #6419 Milvus crashes when inserting empty vectors.

- #6477 Different components repeatedly create buckets in MinIO.
- #6377 Query results get incorrect global sealed segments from etcd.
- #6499 TSO allocates wrong timestamps.
- #6501 Channels are lost after data node crashes.
- #6527 Task info of `watchQueryChannels` can't be deleted from etcd.
- #6576 #6526 Duplicate primary field IDs are added when retrieving entities.
- #6627 #6569 `std::sort` does not work properly to filter search results when the distance of new record is NaN.
- #6655 Proxy crashes when retrieve task is called.
- #6762 Incorrect created timestamp of collections and partitions.
- #6644 Data node failes to restart automatically.
- #6641 Failure to stop data coord when disconnecting with etcd.
- #6621 Milvus throws an exception when the inserted data size is larger than the segment.
- #6436 #6573 #6507 Incorrect handling of time synchronization.
- #6732 Failure to create IVF_PQ index.

v2.0.0-RC2

Release date: 2021-07-13

Compatibility

Milvus version	Python SDK version	Java SDK version	Go SDK version
2.0.0-RC2	2.0.0rc2	Coming soon	Coming soon

Milvus 2.0.0-RC2 is a preview version of Milvus 2.0. It fixes stability and performance issues and refactors code for node and storage management.

Improvements

- #6356 Refactors code for cluster in data coordinator.
- #6300 Refactors code for meta management in data coordinator.
- #6289 Adds `collectionID` and `partitionID` to `SegmentIndexInfo`.
- #6258 Clears the corresponding `searchMsgStream` in proxy when calling `releaseCollection()`.
- #6227 Merges codes relating to retrieve and search in query node.
- #6196 Adds candidate management for data coordinator to manage data node cluster.
- #6188 Adds Building Milvus with Docker Docs.

Features

- #6386 Adds the `fget_objects()` method for loading files from MinIO to the local device.
- #6253 Adds the `GetFlushedSegments()` method in data coordinator.
- #6213 Adds the `GetIndexStates()` method.

Bug fixes

- #6184 Search accuracy worsens when dataset gets larger.
- #6308 The server crashes if the KNNG in NSG is not full.
- #6212 Search hangs after restarting query nodes.
- #6265 The server does not check node status when detecting nodes are online.
- #6359 #6334 An error occurs when compiling Milvus on CentOS

v2.0.0-RC1

Release date: 2021-06-28

Compatibility

Milvus version	Python SDK version	Java SDK version	Go SDK version
2.0.0-RC1	2.0.0rc1	Coming soon	Coming soon

Milvus 2.0.0-RC1 is the preview version of 2.0. It introduces Golang as the distributed layer development language and a new cloud-native distributed design. The latter brings significant improvements to scalability, elasticity, and functionality.

Architecture

Milvus 2.0 is a cloud-native vector database with storage and computation separated by design. All components in this refactored version of Milvus are stateless to enhance elasticity and flexibility.

The system breaks down into four levels:

- Access layer
- Coordinator service
- Worker nodes
- Storage

Access layer: The front layer of the system and endpoint to users. It comprises peer proxies for forwarding requests and gathering results.

Coordinator service: The coordinator service assigns tasks to the worker nodes and functions as the system's brain. It has four coordinator types: root coord, data coord, query coord, and index coord.

Worker nodes: Worker nodes are dumb executors that follow the instructions from the coordinator service. There are three types of worker nodes, each responsible for a different job: data nodes, query nodes, and index nodes.

Storage: The cornerstone of the system that all other functions depend on. It has three storage types: meta storage, log broker, and object storage. Kudos to the open-source communities of etcd, Pulsar, MinIO, and RocksDB for building this fast, reliable storage.

> For more information about how the system works, see Milvus 2.0 Architecture.

New Features

SDK

- Object-relational mapping (ORM) PyMilvus

  The PyMilvus APIs operate directly on collections, partitions, and indexes, helping users focus on the building of an effective data model rather than the detailed implementation.

Core Features

- Hybrid Search between scalar and vector data

  Milvus 2.0 supports storing scalar data. Operators such as GREATER, LESS, EQUAL, NOT, IN, AND, and OR can be used to filter scalar data before a vector search is conducted. Currently supported data types include bool, int8, int16, int32, int64, float, and double. Support for string/VARBINARY data will be offered in a later version.

- Match query

  Unlike the search operation, which returns similar results, the match query operation returns exact matches. Match query can be used to retrieve vectors by primary keys or by condition.

- Tunable consistency

  Distributed databases make tradeoffs between consistency and availability/latency. Milvus offers four consistency levels (from strongest to weakest): strong, bounded staleness, session, and consistent prefix. You can define your own read consistency by specifying the read timestamp. As a rule of thumb, the weaker the consistency level, the higher the availability and the higher the performance.

- Time travel

  Time travel allows you to access historical data at any point within a specified time period, making it possible to query data in the past, restore, and backup.

Miscellaneous

- Supports installing Milvus 2.0 with Helm or Docker-compose.
- Compatibility with Prometheus and Grafana for monitoring and alerts.
- Milvus Insight

  Milvus Insight is a graphical management system for Milvus. It features visualization of cluster states, meta management, data queries and more. Milvus Insight will eventually be open sourced.

Breaking Changes

Milvus 2.0 uses an entirely different programming language, data format, and distributed architecture compared with previous versions. This means prior versions of Milvus cannot be upgraded to 2.x. However, Milvus 1.x is receiving long-term support and data migration tools will be made available as soon as possible.

Specific breaking changes include:

- JAVA, Go, or C++ SDK is not yet supported.
- Delete or update is not yet supported.
- PyMilvus-ORM does not support force flush.
- Data format is incompatible with all prior versions.
- Mishards is deprecated because Milvus 2.0 is distributed and sharding middleware is no longer necessary.
- Local file system and distributed system storage are not yet supported.

Get Started

Prerequisites

Environment Checklist

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

Install with Docker ComposeInstall on Kubernetes

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.

Component	Requirement	Recommendation	Note
RAM			The size of RAM depends on the data volume.
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.

Install Milvus

Milvus Standalone

Install Milvus Standalone

This topic describes how to install Milvus standalone with Docker Compose or on Kubernetes.

Check the requirements for hardware and software prior to your installation.

If you run into image loading errors while installing, you can Install Milvus Offline.

You can also build Milvus from source code at GitHub.

Docker ComposeHelmAPT or YUM

Download an installation file

Download `milvus-standalone-docker-compose.yml` directly or with the following command, and save it as `docker-compose.yml`.

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

Start Milvus

```
$ 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
```

Check the status of the containers.

```
$ sudo docker-compose ps
```

After Milvus standalone starts, three running docker containers appear including two dependencies and one Milvus service.

```
      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
```

Stop Milvus

To stop Milvus standalone, run:

```
sudo docker-compose down
```

To delete data after stopping Milvus, run:

```
sudo rm -rf  volumes
```

What’s next

Having installed Milvus, you can:

- Check Hello Milvus to run an example code with different SDKs to see what Milvus can do.
- Learn the basic operations of Milvus:
  - Connect to Milvus server
  - Conduct a vector search
  - Conduct a hybrid search
- Explore MilvusDM, an open-source tool designed for importing and exporting data in Milvus.
- Monitor Milvus with Prometheus.

Milvus Cluster

Install Milvus Cluster

This topic describes how to install Milvus cluster with Docker Compose or on Kubernetes.

Check the requirements for hardware and software prior to your installation.

If you run into image loading errors while installing, you can Install Milvus Offline.

You can also build Milvus from source code at GitHub.

Docker ComposeHelmMilvus Operator

Download an installation file

Download `milvus-cluster-docker-compose.yml` directly or with the following command, and save it as `docker-compose.yml`.

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

Start Milvus

```
$ 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
```

Check the status of the containers.

```
$ sudo docker ps
```

After Milvus cluster starts, 11 running docker containers appear including three dependencies and eight Milvus services.

```
      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
```

Stop Milvus

To stop Milvus cluster, run $ sudo docker-compose down.

To delete data after stopping Milvus, run $ sudo rm -rf volumes.

What’s next

Having installed Milvus, you can:

- Check Hello Milvus to run an example code with different SDKs to see what Milvus can do.
- Learn the basic operations of Milvus:
  - Connect to Milvus server
  - Conduct a vector search
  - Conduct a hybrid search
- Explore MilvusDM, an open-source tool designed for importing and exporting data in Milvus.
- Monitor Milvus with Prometheus.

Install Offline

Install Milvus Offline

This topic describes how to install Milvus in an offline environment.

Installation of Milvus might fail due to image loading errors. You can install Milvus in an offline environment to avoid such problem.

Install with Docker ComposeInstall on Kubernetes

Download files and images

To install Milvus offline, you need to pull and save all images in an online environment first, and then transfer them to the target host and load them manually.

1. Download an installation file.

- 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
```

2. Download requirement and script files.

```
$ 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
```

3. Pull and save images.

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

The images are stored in the /images folder.

4. Load the images.

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

Install Milvus offline

Having transferred the images to the target host, run the following command to install Milvus offline.

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

Uninstall Milvus

To uninstall Milvus, run the following command.

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

What' s next

Having installed Milvus, you can:

- Check Hello Milvus to run an example code with different SDKs to see what Milvus can do.
- Learn the basic operations of Milvus:
  - Connect to Milvus server
  - Conduct a vector search
  - Conduct a hybrid search
- Scale your Milvus cluster.
- Explore MilvusDM, an open-source tool designed for importing and exporting data in Milvus.

- Monitor Milvus with Prometheus.

Install SDKs

Install Milvus SDK

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

Run Milvus using Python

This topic describes how to run Milvus using Python.

Through running the example code we provided, you will have a primary understanding of what Milvus is capable of.

Preparations

- Milvus 2.0.0
- Python 3 (3.71 or later)
- PyMilvus 2.0.0

Download example code

Download `hello_milvus.py` directly or with the following command.

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

Scan the example code

The example code performs the following steps.

- Imports a PyMilvus package:

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

- Connects to a server:

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

- Creates a 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)
```

- Inserts vectors in the collection:

```
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)
```

- Builds indexes on the entities:

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

- Loads the collection to memory and performs a vector similarity search:

```
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
```

- Performs a vector query:

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

- Performs a hybrid search:

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

- Deletes entities by their primary keys:

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

- Drops the collection:

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

Run the example code

Execute the following command to run the example code.

```
$ python3 hello_milvus.py
```

The returned results and query latency are shown as follows:

```
=== 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` ===
```

Congratulations! You have started Milvus standalone and performed your first vector similarity search.

User Guide

Manage Milvus Connections

This topic describes how to connect to and disconnect from a Milvus server.

Ensure to connect to Milvus server before any operations.

Below example connects to a Milvus server with host as `localhost` and port as `19530` and disconnects from it.

Connect to a Milvus server

Construct a Milvus connection. Ensure to connect to Milvus server before any operations.

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
```

Parameter

Description

alias

Alias of the Milvus connection to construct.

host

IP address of the Milvus server.

port

Port of the Milvus server.

Parameter

Description

address

Address of the Milvus connection to construct.

Parameter

Description

ctx

Context to control API invocation process.

addr

Address of the Milvus connection to construct.

Parameter

Description

Host

IP address of the Milvus server.

Port

Port of the Milvus server.

Option

Description

-h (Optional)

The host name. The default is “127.0.0.1”.

-p (Optional)

The port number. The default is “19530”.

-a (Optional)

The alias name of the Milvus link. The default is “default”.

-D (Optional)

Flag to disconnect from the Milvus server specified by an alias. The default alias is “default”.

Return

A Milvus connection created by the passed parameters.

Raises

NotImplementedError: If handler in connection parameters is not GRPC.

ParamError: If pool in connection parameters is not supported.

Exception: If server specified in parameters is not ready, we cannot connect to server.

Disconnect from a Milvus server

Disconnect from a Milvus server.

Python Java GO Node.js CLI

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

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

```
milvusClient.Close()
```

```
milvusClient.close()
```

```
connect -D
```

Parameter

Description

alias

Alias of the Milvus server to disconnect from.

Limits

The maximum number of connections is 65,536.

What's next

Having connected to a Milvus server, you can:

- Create a collection
- Manage data
- Build a vector index
- Conduct a vector search
- Conduct a hybrid search

For advanced operations, check:

- PyMilvus API reference
- Node.js API reference

Manage Collections

Create a Collection

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.

Milvus supports setting consistency level while creating a collection (only on PyMilvus currently). In this example, the consistency level of the collection is set as "strong", meaning Milvus will read the most updated data view at the exact time point when a search or query request comes. By default, a collection created without specifying

the consistency level is set with bounded consistency level, under which Milvus reads a less updated data view (usually several seconds earlier) when a search or query request comes. Besides collection creation, you can also set the consistency level specifically for search or query (only on PyMilvus currently). For other consistency level supported by Milvus, see Guarantee Timestamp in Search Requests.

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

```
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"
```

```
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> i
```

```
    <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 strong consistency level and 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,
    consistency_level="strong"
    )
```

```
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]

consistency_level (optional)

Consistency level of the collection to create.

Strong

Bounded

Session

Eventually

Customized

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

Check Collection Information

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 the name and schema of the collection.
  DescribeCollectionParam.newBuilder()
    .withCollectionName("book")
    .build()
);
DescCollResponseWrapper wrapperDescribeCollection = new DescCollResponseWrapper(respDescribeCollection.getS
System.out.println(wrapperDescribeCollection);

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

```
describe collection -c book
```

Parameter

Description

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.

primary_field

The primary field of the collection.

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.

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

Drop a Collection

Drop a collection

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

Manage Collection Alias

Collection Alias

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

Load Collection

Load a Collection

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.

Constraints

- Error will be returned at the attempt to load partition(s) when the parent collection is already loaded. Future releases will support releasing partitions from a loaded collection, and (if needed) then loading some other partition(s).
- "Load successfully" will be returned at the attempt to load the collection that is already loaded.
- Error will be returned at the attempt to load the collection when the child partition(s) is/are already loaded. Future releases will support loading the collection when some of its partitions are already loaded.
- Loading different partitions in a same collection via separate RPCs is not allowed.

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

Release Collection

Release a Collection

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.

Constraints

- Releasing the collection that is successfully loaded is allowed.
- Releasing the collection is allowed when its partition(s) are loaded.
- Error will be returned at the attempt to release partition(s) when the parent collection is already loaded. Future releases will support releasing partitions from a loaded collection, and loading the collection when its partition(s) are released.

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

Manage Partitions

Create a Partition

Create a Partitions

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

Check Partition Information

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

Drop a Partition

Drop Partitions

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

Load a Partition

This topic describes how to load a partition to memory. Loading partitions instead of the whole collection to memory can significantly reduce the memory usage. 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(["novel"])
```

```
### Or you can load a partition with the partition as an object
from pymilvus import Partition
partition = Partition("novel")       ### Get an existing partition.
partition.load()
```

```
await milvusClient.partitionManager.loadPartitions({
  collection_name: "book",
  partition_names: ["novel"],
});
```

```
err := milvusClient.LoadPartitions(
  context.Background(),   // ctx
  "book",                 // CollectionName
  []string{"novel"},      // partitionNames
  false                   // async
)
if err != nil {
  log.Fatal("failed to load partitions:", err.Error())
}
```

```
milvusClient.loadPartitions(
  LoadPartitionsParam.newBuilder()
          .withCollectionName("book")
          .withPartitionNames(["novel"])
          .build()
);
```

```
load -c book -p novel
```

Parameter

Description

partition_name

Name of the partition.

Parameter

Description

collection_name

Name of the collection to load partitions from.

partition_names

List of names of the partitions to load.

Parameter

Description

ctx

Context to control API invocation process.

CollectionName

Name of the collection to load partitions from.

partitionNames

List of names of the partitions 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 partitions from.

PartitionNames

List of names of the partitions to load.

Option

Description

-c

Name of the collection to load partitions from.

-p (Multiple)

The name of the partition to load.

Constraints

- Error will be returned at the attempt to load partition(s) when the parent collection is already loaded. Future releases will support releasing partitions from a loaded collection, and (if needed) then loading some other partition(s).
- "Load successfully" will be returned at the attempt to load the collection that is already loaded.
- Error will be returned at the attempt to load the collection when the child partition(s) is/are already loaded. Future releases will support loading the collection when some of its partitions are already loaded.
- Loading different partitions in a same collection via separate RPCs is not allowed.

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

Release a Partition

Load a Partition

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

Python Java GO Node.js CLI

```
from pymilvus import Partition
partition = Partition("novel")       ### Get an existing partition.
partition.release()
```

```
await milvusClient.partitionManager.releasePartitions({
    collection_name: "book",
    partition_names: ["novel"],
 });
```

```
err := milvusClient.ReleasePartitions(
  context.Background(),   // ctx
  "book",                 // CollectionName
```

```
  []string{"novel"}       // partitionNames
)
if err != nil {
  log.Fatal("failed to release partitions:", err.Error())
}
```

```
milvusClient.releasePartitions(
  ReleasePartitionsParam.newBuilder()
    .withCollectionName("book")
    .withPartitionNames(["novel"])
    .build()
);
```

```
release -c book -p novel
```

Parameter

Description

partition_name

Name of the partition.

Parameter

Description

collection_name

Name of the collection to release partitions.

partition_names

List of names of the partitions to load.

Parameter

Description

ctx

Context to control API invocation process.

CollectionName

Name of the collection to release partitions.

partitionNames

List of names of the partitions to release.

Parameter

Description

CollectionName

Name of the collection to release partition.

PartitionNames

List of names of the partitions to release.

Option

Description

-c

Name of the collection to release partition.

-p (Multiple)

The name of the partition to release.

Constraints

- Error will be returned at the attempt to load partition(s) when the parent collection is already loaded. Future releases will support releasing partitions from a loaded collection, and (if needed) then loading some other partition(s).
- "Load successfully" will be returned at the attempt to load the collection that is already loaded.
- Error will be returned at the attempt to load the collection when the child partition(s) is/are already loaded. Future releases will support loading the collection when some of its partitions are already loaded.
- Loading different partitions in a same collection via separate RPCs is not allowed.

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

Manage Data

Insert Data

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

Delete Entities

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

Compact Data

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

Manage Indexes

Build an Index

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

Drop an Index

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

Search and Query

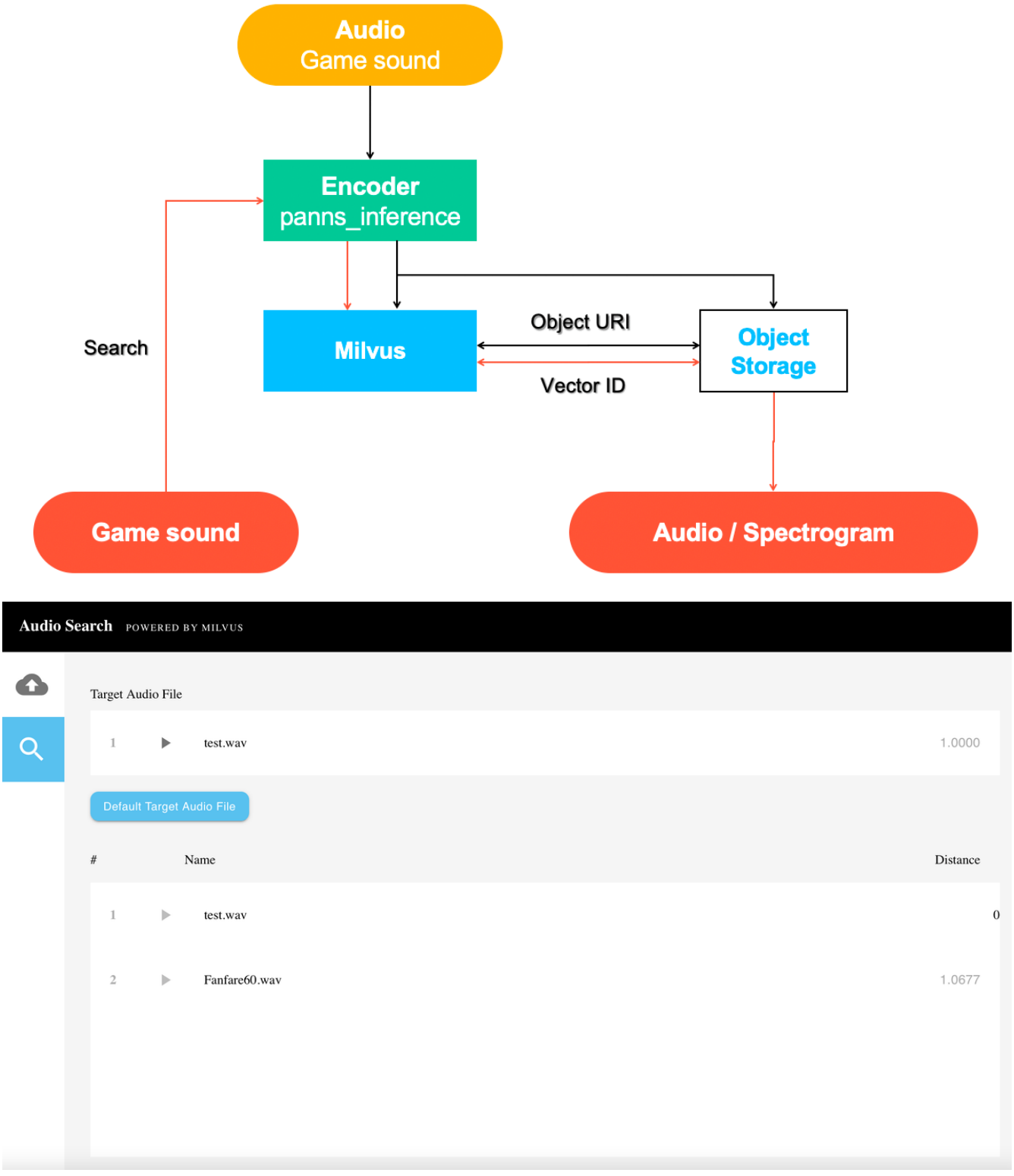
Search

Audio Similarity Search

This tutorial demonstrates how to use Milvus, the open-source vector database to build an audio similarity search system. - Open Jupyter notebook - Quick deploy The ML model and third-party software used include: - PANNs (Large-Scale Pretrained Audio Neural Networks) - MySQL

Speech, music, sound effects, and other types of audio search makes it possible to quickly query massive volumes of audio data and surface similar sounds. Applications of audio similarity search systems include identifying similar sound effects, minimizing IP infringement, and more. Audio retrieval can be used to search and monitor online media in real-time to crack down on infringement of intellectual property rights. It also assumes an important role in the classification and statistical analysis of audio data.

In this tutorial, you will learn how to build an audio similarity search system that can return similar sound clips. The uploaded audio clips are converted into vectors using PANNs. These vectors are stored in Milvus which automatically generates a unique ID for each vector. Then users can conduct a vector similarity search in Milvus and query the audio clip data path corresponding to th unique vector ID returned by Milvus.



Query

Conduct a Vector Query

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.

Milvus supports setting consistency level specifically for a search or query (only on PyMilvus currently). The consistency level set in the search or query requests overwrites the one set while creating the collection. In this example, the consistency level of the search request is set as "strong", meaning Milvus will read the most updated data view at the exact time point when a search or query request comes. Without specifying the consistency level during a search or query, Milvus adopts the original consistency level of the collection.

Python Java GO Node.js CLI

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

```
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.

consistency_level (optional)

Consistency level of the query.

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

Hybrid Search

Conduct a Hybrid Search

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 distance
```

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

Search with Time Travel

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

Load Balance

Balance Query Load

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

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

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

```
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

Administration Guide

Configure Milvus

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 to 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.

Allocate Resources

Allocate Resources on Kubernetes

This topic describes how to allocate resources to a Milvus cluster on Kubernetes.

Generally, the resources you allocate to a Milvus cluster in production should be proportionate to the machine workload. You should also consider the machine type when allocating resources. Although you can update the configurations when the cluster is running, we recommend setting the values before deploying the cluster.

1. View available resources

Run `kubectl describe nodes` to view the available resources on the instances that you have provisioned.

2. Allocate resources

Use Helm to allocate CPU and memory resources to Milvus components.

Using Helm to upgrade resources will cause the running pods to perform rolling update.

There are two ways to allocate resources:

- Use the commands
- Set the parameters in the `YAML` file

Allocate resources with commands

You need to set the resource variables for each Milvus component if you use `--set` to update the resource configurations.

Milvus standalone Milvus cluster

```
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.
```

Allocate resources by setting configuration file

You can also allocate CPU and memory resources by specifying the parameters `resources.requests` and `resources.limits` in the `resources.yaml` file.

```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. Apply configurations

Run the following command to apply the new configurations to your Milvus cluster.

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

If resources.limits is not specified, the pods will consume all the CPU and memory resources available. Therefore, ensure to specify resources.requests and resources.limits to avoid overallocation of resources when other running tasks on the same instance require more memory consumption.

See Kubernetes documentation for more information about managing resources.

What' s next

- You might also want to learn how to:
  - Scale a Milvus cluster
  - Upgrade your Milvus instance
- 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

Deploy on Clouds

AWS

Amazon EC2

Deploy a Milvus Cluster on EC2

This topic describes how to deploy a Milvus cluster on Amazon EC2 with Terraform and Ansible.

Provision a Milvus cluster

This section describes how to use Terraform to provision a Milvus cluster.

Terraform is an infrastructure as code (IaC) software tool. With Terraform, you can provision infrastructure by using declarative configuration files.

Prerequisites

- Install and configure Terraform
- Install and configure AWS CLI

Prepare configuration

You can download template configuration files at Google Drive.

- `main.tf`

  This file contains the configuration for provisioning a Milvus cluster.

- `variables.tf`

  This file allows quick editing of variables used to set up or update a Milvus cluster.

- `output.tf` and `inventory.tmpl`

  These files store the metadata of a Milvus cluster. The metadata used in this topic is the `public_ip` for each node instance, `private_ip` for each node instance, and all EC2 instance IDs.

Prepare variables.tf

This section describes the configuration that a `variables.tf` file that contains.

- Number of nodes

  The following template declares an `index_count` variable used to set the number of index nodes.

  The value of index_count must be greater than or equal to one.

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

- Instance type for a node type

  The following template declares an `index_ec2_type` variable used to set the instance type for index nodes.

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

- Access permission

  The following template declares a `key_name` variable and a `my_ip` variable. The `key_name` variable represents the AWS access key. The `my_ip` variable represents the IP address range for a security group.

```
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"
}
```

Prepare main.tf

This section describes the configurations that a `main.tf` file that contains.

- Cloud provider and region

  The following template uses the `us-east-2` region. See Available Regions for more information.

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

- Security group

  The following template declares a security group that allows incoming traffic from the CIDR address range represented by `my_ip` declared in `variables.tf`.

  ```
  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"
  }
}
```

- VPC

  The following template specifies a VPC with the 10.0.0.0/24 CIDR block on a Milvus cluster.

```
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"
  }
}
```

- Subnets (Optional)

  The following template declares a subnet whose traffic is routed to an internet gateway. In this case, the size of the subnet's CIDR block is the same as the VPC's CIDR block.

```
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
}
```

- Node instances (Nodes)

  The following template declares a MinIO node instance. The `main.tf` template file declares nodes of 11 node types. For some node types, you need to set `root_block_device`. See EBS, Ephemeral, and Root Block Devices

for more information.

```
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}"
  }
}
```

Apply the configuration

1. Open a terminal and navigate to the folder that stores `main.tf`.
2. To initialize the configuration, run `terraform init`.
3. To apply the configuration, run `terraform apply` and enter `yes` when prompted.

You have now provisioned a Milvus cluster with Terraform.

Start the Milvus cluster

This section describes how to use Ansible to start the Milvus cluster that you have provisioned.

Ansible is a configuration management tool used to automate cloud provisioning and configuration management.

Prerequisites

- Install and configure Ansible

Prepare configuration

You can download template configuration files at Google Drive.

- Files in the `yaml_files` folder

  This folder stores Jinja2 files for each node type. Ansible uses Jinja2 templating. See Introduction for more information about Jinja2.

-
  ```
  This file performs a set of tasks on specific sets of nodes. The template begins with installing Docke

  <div class="alert note">A playbook runs in sequence from top to bottom. Within each play, tasks also r

  ```playbook.yaml
  - name: All Servers
    hosts: etcd_ips_public:pulsar_ips_public:minio_ips_public:data_ips_public:index_ips_public:query_ips
    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
```

After Docker and Docker Compose are installed on all node instances, `playbook.yaml` starts containers for all node instances in sequence.

```
- 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/
```

Apply the configuration

1. Open a terminal and navigate to the folder that stores `playbook.yaml`.
2. Run `ansible-playbook -i inventory playbook.yaml --tags "start"`.
3. If successful, all node instances start.

You have now started a Milvus cluster with Ansible.

Stop nodes

You can stop all nodes after you do not need a Milvus cluster any longer.

Ensure that the terraform binary is available on your PATH.

1. Run `terraform destroy` and enter `yes` when prompted.
2. If successful, all node instances are stopped.

What's next

If you want to learn how to deploy Milvus on other clouds: - Deploy a Milvus Cluster on EKS - Deploy Milvus Cluster on GCP with Kubernetes - Guide to Deploying Milvus on Microsoft Azure With Kubernetes

Amazon EKS

Deploy a Milvus Cluster on EKS

This topic describes how to deploy a Milvus cluster on Amazon EKS.

This topic assumes that you have a basic understanding of AWS access management. If you're not familiar with it, see AWS Identity and Access Management Documentation.

Prerequisites

Software requirements

- Terraform
- Helm
- kubectl
- AWS CLI version 2

Cloud security

- Access to EKS, EC2, and S3
- Access key ID
- Security access key

Deploy a Milvus cluster

You can download template configuration files at Google Drive.

1. Provision a Milvus cluster. See Provision a Milvus cluster for more information.
2. After a Milvus cluster is provisioned, run the following command with a region and name for the cluster.

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

3. Create a kubeconfig file and run `kubectl get svc`. If successful, a cluster appears in the output.

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

4. Run the following command to start the Milvus cluster that you have provisioned. The access key and an S3 bucket are required to use S3 as storage.

```
helm upgrade --install --set cluster.enabled=true --set externalS3.enabled=true --set externalS3.host='s3.
```

5. Run `kubectl get svc` again to retrieve the IP address of the load balancer and use it as the IP address of the Milvus cluster.

Run kubectl get pods to view the running pods on the cluster.

Scale the Milvus cluster

Currently, a Milvus cluster can only be scaled manually. Run the following command to modify the numbers of node instances with different types.

See Storage/Computing Disaggregation for more information about the data node, index node, query node, and proxy.

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

After running the preceding command, you can run `kubectl get pods` to view the newly created node instances.

What's next

If you want to learn how to deploy Milvus on other clouds: - Deploy a Milvus Cluster on EC2 - Deploy Milvus Cluster on GCP with Kubernetes - Guide to Deploying Milvus on Microsoft Azure With Kubernetes

GCP

Deploy a Milvus Cluster on GCP

This topic describes how to deploy a Milvus cluster on Google Cloud Platform (GCP).

Prerequisites

Determine the Google Cloud project that you want to work with. If you are not sure which one to use, ask your GCP administrators to create a new one. See Creating and managing projects for more information. The project used in this topic is named milvus-testing-nonprod. Replace it with your project name in commands.

Software requirements

- Cloud SDK
- kubectl
- Helm

Alternatively, you can use Cloud Shell which has the GCP SDK, kubectl, and Helm preinstalled.

After you install the Cloud SDK, ensure that you are properly authenticated.

Set up network

Ensure that you create a virtual private cloud (VPC) before creating a firewall rule for Milvus. If you already have a VPC that you want to use, proceed to Create a firewall rule for Milvus.

Create a VPC

Open a terminal and run the following command to create a VPC.

Replace milvus-testing-nonprod with your project name.

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

Run the following commands to create firewall rules to allow ICMP, internal, RDP, and SSH traffic.

```
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
```

Create a firewall rule for Milvus

Create a firewall rule to allow incoming traffic on the `19530` port used by Milvus.

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

Provision a Kubernetes cluster

We use Google Kubernetes Engine (GKE) to provision a K8s cluster. In this topic, we create a cluster that has two nodes. The nodes are in the `use-west1-a` zone, are with the `e2-standard-4` machine type, and use the `cos_containerd` node image.

Modify the preceding options as needed.

Select a machine type

In this topic, we use the `e2-standard-4` machine type, which has 4 vCPUs and 16 GB of memory.

You can select machine types as you need. However, we recommend that you select machine types that have a minimum of 16 GB of memory to ensure stability.

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

Creating a cluster might take several minutes. After the cluster is created, run the following command to fetch credentials for the cluster.

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

The preceding command points `kubectl` at the cluster.

Deploy Milvus

After provisioning a cluster, you can deploy Milvus. If you switch to a different terminal, run the following command again to fetch credentials.

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

1. Run the following command to add the Milvus Helm chart repository.

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

2. Run the following command to update your Milvus Helm chart.

```
helm repo update
```

3. Run the following command to deploy Milvus.

This topic uses the my-release release name. Replace it with your release name.

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

Starting pods might take several minutes. Run kubectl get services to view services. If successful, a list of services is shown as follows.

GCP

34.145.26.89 in the the EXTERNAL-IP column is the IP address of the load balancer. The IP address is used to connect to Milvus.

Use Google Cloud Storage

Google Cloud Storage (GCS) is Google Cloud's version of AWS Simple Storage Service (S3).

MinIO GCS Gateway allows accessing GCS. Essentially, MinIO GCS Gateway translates and forwards all connections to GCS by using APIs. You can use MinIO GCS Gateway instead of a MinIO server.

Set variables

Set variables before you use MinIO GCS Gateway. Modify the default values as needed.

Secrets

To access GCS resources, MinIO GCS Gateway requires both GCP service account credentials and MinIO credentials. Store the credentials in a K8s secret. The credentials are listed as follows.

- `accesskey`: The MinIO access key.
- `secretkey`: The MinIO secret key.
- `gcs_key.json`: The GCP service account credentials file.

The following example creates a secret named `mysecret` with `accesskey=minioadmin`, `secretkey=minioadmin`, and `gcs_key.json` using the `/home/credentials.json` path.

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

If you choose accesskey and secretkey values other than the default minioadmin/minioadmin, you need to update the minio.accessKey and minio.secretKey metadata variables as well.

Metadata

The following table lists the metadata that you can configure. |Option|Description|Default| |:—|:—|:—| |`minio.gcsgateway.enable` the value to `true` to enable MinIO GCS Gateway.|`false`| |`minio.gcsgateway.projectId`|The ID of the GCP project.|`""`| |`minio.existingSecret`|The name of the previously defined secret.|`""`| |`externalGcs.bucketName`|The name of the GCS bucket to use. Unlike an S3/MinIO bucket, a GCS bucket must be globally unique.|`""`|

The following table lists the metadata that you might want to leave as default. |Option|Description|Default| |:—|:—|:—| |`minio.gcsgateway.replicas`|The number of replica nodes to use for the gateway. We recommend that you use one because MinIO does not support well for more than one replica.|`1`| |`minio.gcsgateway.gcsKeyJson`|The file path to GCS service account access credentials file. Do not modify the default value.|`/etc/credentials/gcs_key.json`|

Continue to use all normal MinIO metadata variables.

The following example installs a chart named `my-release`.

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

What’s next

If you want to learn how to deploy Milvus on other clouds: - Deploy a Milvus Cluster on EC2 - Deploy a Milvus Cluster on EKS - Deploy a Milvus Cluster on Azure

Azure

Deploy Milvus on Azure with AKS

This topic describes how to provision and create a cluster with Azure Kubernetes Service (AKS) and the Azure portal.

Prerequisites

Ensure that your Azure project has been set up properly and you have access to the resources that you want to use. Contact your administrators if you are not sure about your access permission.

Software requirements

- Azure CLI
- kubectl
- Helm

Alternatively, you can use the Cloud Shell which has the Azure CLI, kubectl, and Helm preinstalled.

After you install the Azure CLI, ensure that you are properly authenticated.

Provision a Kubernetes cluster

1. Log on to the Azure portal.
2. On the Azure portal menu or from the Home page, select Create a resource.
3. Select Containers > Kubernetes Service.
4. On the Basics page, configure the following options:

- Project details:
  - Subscription: Contact your organization' s Azure Administrator to determine which subscription you should use.
    - Resource group: Contact your organization' s Azure Administrator to determine which resource group you should use.
- Cluster details:
  - Kubernetes cluster name: Enter a cluster name.
  - Region: Select a region.
  - Availability zones: Select availability zones as you need. For production clusters, we recommend that you select multiple availability zones.
- Primary node pool:
  - Node size: We recommend that you choose VMs with a minimum of 16 GB of RAM, but you can select virtual machine sizes as you need.
  - Scale method: Choose a scale method.
  - Node count range: Select a range for the number of nodes.
- Node pools:
  - Enable virtual nodes: Select the checkbox to enable virtual nodes.
  - Enable virtual machine scale sets: We recommend that you choose `enabled`.
- Networking:
  - Network configuration: We recommend that you choose `Kubenet`.
  - DNS name prefix: Enter a DNS name prefix.
  - Traffic Routing:
    - Load balancer: `Standard`.
    - HTTP application routing: Not required.

5. After configuring the options, click Review + create and then Create when validation completes. It takes a few minutes to create the cluster.

Deploy Milvus with Helm

After the cluster is created, install Milvus on the cluster with Helm.

Connect to the cluster

1. Navigate to the cluster that you have created in Kubernetes services and click it.
2. On the left-side navigation pane, click `Overview`.
3. On the Overview page that appears, click Connect to view the resource group and subscription. Azure

Set a subscription and credentials

You can use Azure Cloud Shell to perform the following procedures.

1. Run the following command to set your subscription.

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

2. Run the following command to download credentials and configure the Kubernetes CLI to use them.

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

Use the same shell for the following procedures. If you switch to another shell, run the preceding commands again.

Deploy Milvus

1. Run the following command to add the Milvus Helm chart repository.

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

2. Run the following command to update your Milvus Helm chart.

```
helm repo update
```

3. Run the following command to install Milvus.

This topic uses my-release as the release name. Replace it with your release name.

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

Starting pods might take several minutes. Run `kubectl get services` to view services. If successful, a list of services is shown as follows.

Results

20.81.111.155 in the the EXTERNAL-IP column is the IP address of the load balancer. The default Milvus port is 19530.

Using Azure Blob Storage

Azure Blob Storage is Azure' s version of AWS Simple Storage Service (S3).

MinIO Azure Gateway allows accessing Azure. Essentially, MinIO Azure Gateway translates and forwards all connections to Azure by using APIs. You can use MinIO Azure Gateway instead of a MinIO server.

Set variables

Set variables before you use MinIO Azure Gateway. Modify the default values as needed.

Metadata

The following table lists the metadata that you can configure.

Option	Description	Default
`minio.azuregateway.enabled`	Set the value to `true` to enable MinIO Azure Gateway.	`false`
`minio.accessKey`	The MinIO access key.	`""`
`minio.secretKey`	The MinIO secret key.	`""`
`externalAzure.bucketName`	The name of the Azure bucket to use. Unlike an S3/MinIO bucket, an Azure bucket must be globally unique.	`""`

The following table lists the metadata that you might want to leave as default.

Option	Description	Default
`minio.azuregateway.replicas`	The number of replica nodes to use for the gateway. We recommend that you use one because MinIO does not support well for more than one replica.	`1`

Continue to use all predefined MinIO metadata variables.

The following example installs a chart named `my-release`.

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

What’s next

If you want to learn how to deploy Milvus on other clouds: - Deploy a Milvus Cluster on EC2 - Deploy a Milvus Cluster on EKS - Deploy a Milvus Cluster on GCP

Storage

Set Up Storage

Milvus supports using Amazon Simple Storage Service (S3) as persistent storage for log and index files. This topic describes how to set up S3 for Milvus.

You can set up S3 with Docker Compose or on K8s.

Set up with Docker Compose

1. Configure S3

MinIO is compatible with S3. To set up S3 with Docker Compose, provide your values for the minio section in the milvus.yaml file on the milvus/configs path.

```
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>"
```

See MinIO/S3 Configurations for more information.

2. Run Milvus

Run the following command to start Milvus that uses the S3 configurations.

```
docker-compose up
```

Configurations only take effect after Milvus starts. See Start Milvus for more information.

Set up on K8s

For Milvus clusters on K8s, you can configure S3 in the same command that starts Milvus. Alternatively, you can configure S3 using the values.yml file on the /charts/milvus path in the milvus-helm repository before you start Milvus.

The following table lists the keys for configuring S3 in the YAML file. | Key | Description | Value | | ———— | ———————| ———————| | externalS3.enabled | Enables or disables S3. | true/false | | externalS3.host | The endpoint to access S3. | | | externalS3.port | The port to access S3. | | | externalS3.accessKey | The

access key ID for S3. | | | externalS3.secretKey | The secret access key for S3. | | | externalS3.bucketName | The name of the S3 bucket. | | | minio.enabled | Enables or disables MinIO. | true/false |

Using the YAML file

1. Configure the minio section in the values.yaml file.

```
minio:
  enabled: false
```

2. Configure the externalS3 section using your values in the values.yaml file.

```
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>"
```

3. After configuring the preceding sections and saving the values.yaml file, run the following command to install Milvus that uses the S3 configurations.

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

Using a command

To install Milvus and configure S3, run the following command using your values.

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

What's next

If you want to learn how to use storage from other cloud providers: - Use Google Cloud Storage - Use Azure Blob Storage

Configure Dependencies

Configure Dependencies with Milvus Operator

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.

```
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.

```yaml
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.

```yaml
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.

Scale a Milvus Cluster

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.

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.

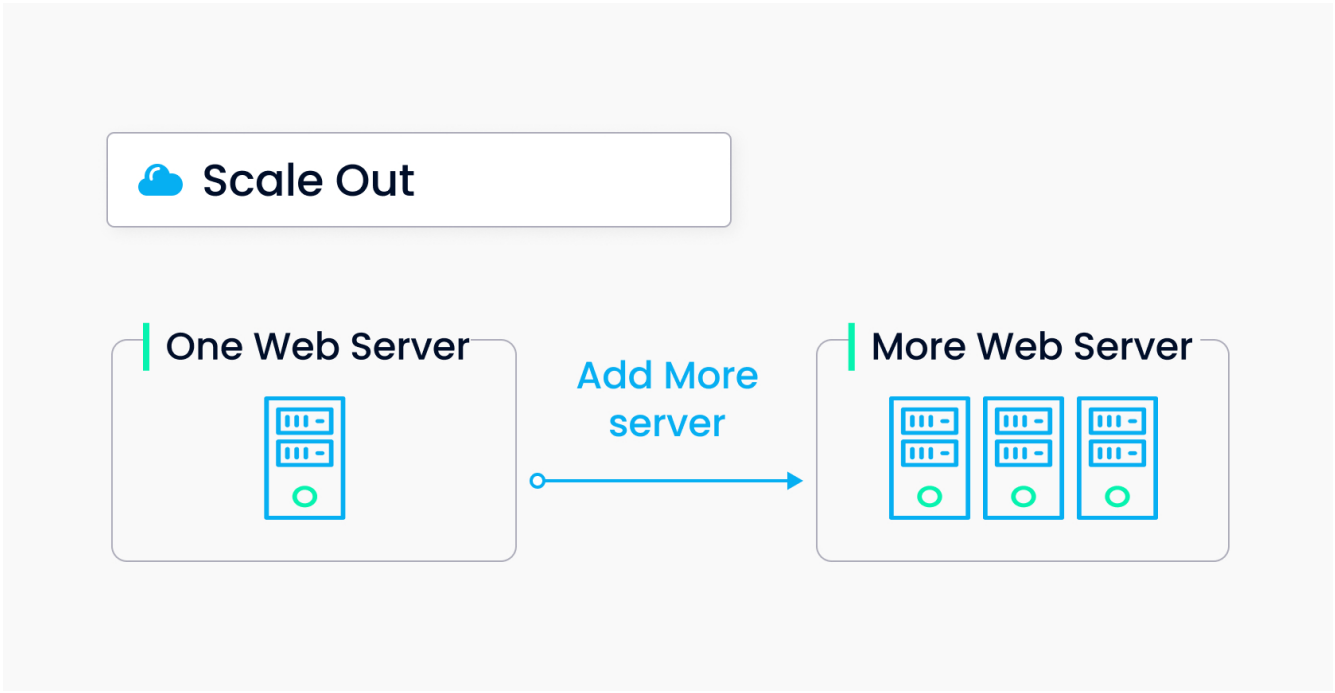


Figure 4: Scaleout

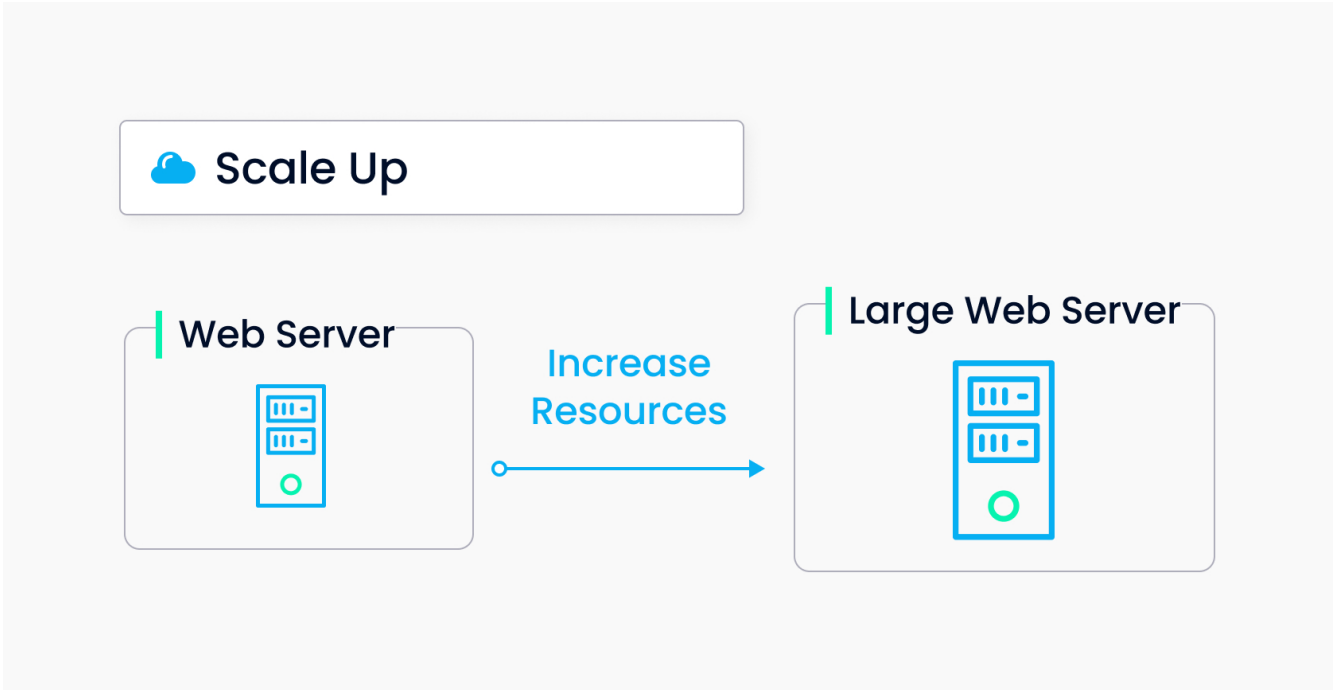


Figure 5: Scaleup

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.

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

Upgrade

Upgrade Milvus Using Helm Chart

This topic describes how to upgrade Milvus 2.0 with Helm Chart using the example of upgrading from Milvus 2.0.0-RC7 to 2.0.0-RC8.

Helm Chart does not support upgrading from Milvus 2.0 standalone to Milvus 2.0 cluster or vice versa. Milvus 2.0.0-RC7 is not compatible with earlier RC versions. Therefore, you cannot upgrade from prior versions to 2.0.0-RC7.

Upgrade Milvus standalone

Step 1. Check the Milvus version

Run `$ helm list` to check your Milvus app version. You can see the `APP VERSION` is 2.0.0-rc7.

```
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS
my-release      default         1               2021-11-08 17:12:44.678247 +0800 CST    deployed
```

Step 2. Check the running pods

Run `$ kubectl get pods` to check the running pods. You can see the following output.

```
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
```

Step 3. Check the image tag

Check the image tag for the pod `my-release-milvus-standalone-75c599fffc-6rwlj`. You can see the release of your Milvus standalone is 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
```

Step 4. Check available app versions

Run the following commands to see all available app versions.

```
$ 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 databa
milvus/milvus       2.3.2                2.0.0-rc.8               Milvus is an open-source vector databa
milvus/milvus       2.3.1                2.0.0-rc.8               Milvus is an open-source vector databa
milvus/milvus       2.3.0                2.0.0-rc.8               Milvus is an open-source vector databa
milvus/milvus       2.2.6                2.0.0-rc.7               Milvus is an open-source vector databa
milvus/milvus       2.2.5                2.0.0-rc.7               Milvus is an open-source vector databa
milvus/milvus       2.2.4                2.0.0-rc.7               Milvus is an open-source vector databa
milvus/milvus       2.2.3                2.0.0-rc.7               Milvus is an open-source vector databa
milvus/milvus       2.2.2                2.0.0-rc.7               Milvus is an open-source vector databa
milvus/milvus       2.2.1                2.0.0-rc.6               Milvus is an open-source vector databa
milvus/milvus       2.2.0                2.0.0-rc.6               Milvus is an open-source vector databa
```

Step 5. Upgrade

1. Run the following commands to upgrade your Milvus standalone from 2.0.0-RC7 to 2.0.0-RC8.

```
$ helm repo update
$ helm upgrade my-release milvus/milvus --set cluster.enabled=false --set etcd.replicaCount=1 --set minio.r
```

2. Run `$ helm list` again to check your Milvus app version. You can see your Milvus standalone has been upgraded to 2.0.0-RC8.

```
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS
my-release      default         2               2021-11-08 17:15:46.530627 +0800 CST    deployed
```

3. Run `$ kubectl get pods` to check the new pods. You can see the following output.

```
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
```

When upgrading your Milvus standalone, old pods will be deleted. Therefore, the service may be offline for a short period of time.

4. Run the following command to check the new image version. You can see it is v2.0.0-rc8 now.

```
$ kubectl get pods my-release-milvus-standalone-6967454987-72r55 -o=jsonpath='{$.spec.containers[0].image}
```

```
milvusdb/milvus:v2.0.0-rc8-20211104-d1f4106
```

Upgrade Milvus cluster

Step 1. Check the Milvus version

Run `$ helm list` to check your Milvus app version. You can see the `APP VERSION` is 2.0.0-rc7.

```
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS
my-release      default         1               2021-11-08 17:21:13.511069 +0800 CST    deployed
```

Step 2. Check the running pods

Run `$ kubectl get pods` to check the running pods. You can see the following output.

```
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
```

Step 3. Check the image tag

Check the image tag for the pod `my-release-milvus-proxy-6664d564f9-pwqn9`. You can see the release of your Milvus cluster is 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
```

Step 4. Check available app versions

Run the following commands to see all available app versions.

```
$ 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 databa
milvus/milvus        2.3.0                2.0.0-rc.8              Milvus is an open-source vector databa
milvus/milvus        2.2.6                2.0.0-rc.7              Milvus is an open-source vector databa
milvus/milvus        2.2.5                2.0.0-rc.7              Milvus is an open-source vector databa
milvus/milvus        2.2.4                2.0.0-rc.7              Milvus is an open-source vector databa
milvus/milvus        2.2.3                2.0.0-rc.7              Milvus is an open-source vector databa
milvus/milvus        2.2.2                2.0.0-rc.7              Milvus is an open-source vector databa
milvus/milvus        2.2.1                2.0.0-rc.6              Milvus is an open-source vector databa
milvus/milvus        2.2.0                2.0.0-rc.6              Milvus is an open-source vector databa
```

Step 5. Upgrade

1. Run the following commands to upgrade your Milvus cluster from 2.0.0-RC7 to 2.0.0-RC8.

```
$ helm repo update
$ helm upgrade my-release milvus/milvus
```

2. Run `$ helm list` again to check your Milvus version. You can see your Milvus cluster has been upgraded to 2.0.0-RC8.

```
NAME              NAMESPACE        REVISION        UPDATED                                 STATUS
my-release        default          2               2021-11-08 17:29:07.815765 +0800 CST    deployed
```

3. Run `$ kubectl get pods` to check the new pods. You can see the following output.

```
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. Run the following command to check the new image version. You can see it is v2.0.0-rc8 now.

```
$ kubectl get pods my-release-milvus-proxy-5685bbc546-v6scq -o=jsonpath='{$.spec.containers[0].image}'

milvusdb/milvus:v2.0.0-rc8-20211104-d1f4106
```

What's next

- You might also want to learn how to:
  - Scale a Milvus cluster

- 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

Monitor and Alert

Monitoring Architecture

Milvus monitoring framework overview

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

Deploy Monitoring Services

Deploying Monitoring Services on Kubernetes

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.

Prometheus_architecture

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

Visualize Milvus Metrics

Visualize Milvus Metrics in Grafana

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

Create an Alert

Create an Alert for Milvus Services

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

Migrate

HDF5 to Milvus

Migrate Data from HDF5 to Milvus

This topic describes how to import data in HDF5 files into Milvus using MilvusDM, an open-source tool specifically designed for Milvus data migration.

Prerequisites

You need to install MilvusDM before migrating Milvus data.

1. Download YAML file

Download the `M2H.yaml` file.

```
$ wget https://raw.githubusercontent.com/milvus-io/milvus-tools/main/yamls/M2H.yaml
```

2. Set the parameters

Configuration parameters include:

Parameter	Description	Example
`milvus_version`	Version of Milvus.	2.0.0
`data_path`	Path to the HDF5 files. Set either `data_path` or `data_dir`.	- /Users/zilliz/float_1.h5 - /Users/zilliz/float_2.h5
`data_dir`	Directory of the HDF5 files. Set either `data_path` or `data_dir`.	'/Users/zilliz/Desktop/HDF5_data'
`dest_host`	Milvus server address.	'127.0.0.1'
`dest_port`	Milvus server port.	19530
`mode`	Mode of migration, including `skip`, `append`, and `overwrite`. This parameter works only when the specified collection name exists in the Milvus library.	'append'
`dest_collection_name`	Name of the collection to import data to.	'test_float'
`dest_partition_name` (optional)	Name of the partition to import data to.	'partition_1'
`collection_parameter`	Collection-specific information including vector dimension, index file size, and similarity metric.	"dimension: 512 index_file_size: 1024 metric_type: 'HAMMING' "

The following two examples of configuration are for your reference. The first example sets the parameter `data_path` while the second sets `data_dir`. You can set either `data_path` or `data_dir` according to your need.

Example 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'
```

Example 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. Migrate data from HDF5 to Milvus

Run MilvusDM to import data in HDF5 files into Milvus with the following command.

```
$ milvusdm --yaml H2M.yaml
```

What's next

- If you are interested in migrating data in other forms into Milvus,
  - Learn how to Migrate Data from Faiss to Milvus.
- If you are looking for information about how to migrate data from Milvus 1.x to Milvus 2.0,
  - Learn version migration.
- If you are interested in learning more about the data migration tool,
  - Read the overview of MilvusDM.

Faiss to Milvus

Migrate Data from Faiss to Milvus

This topic describes how to import data from Faiss to Milvus using MilvusDM, an open-source tool specifically designed for Milvus data migration.

Prerequisites

You need to install MilvusDM before migrating Milvus data.

1. Download YAML file

Download the `F2M.yaml` file.

```
$ wget https://raw.githubusercontent.com/milvus-io/milvus-tools/main/yamls/F2M.yaml
```

2. Set the parameters

Configuration parameters include:

Parameter	Description	Example
`milvus_version`	Version of Milvus.	2.0.0
`data_path`	Path to the data in Faiss.	'/home/user/data/faiss.index'
`data_dir`	Directory of the HDF5 files. Set either `data_path` or `data_dir`.	'/Users/zilliz/Desktop/HDF5_data'
`dest_host`	Milvus server address.	'127.0.0.1'
`dest_port`	Milvus server port.	19530
`mode`	Mode of migration, including `skip`, `append`, and `overwrite`. This parameter works only when the specified collection name exists in the Milvus library.	'append'
`dest_collection_name`	Name of the collection to import data to.	'test'
`dest_partition_name` (optional)	Name of the partition to import data to.	'partition'
`collection_parameter`	Collection-specific information including vector dimension, index file size, and similarity metric.	"dimension: 512 index_file_size: 1024 metric_type: 'HAMMING' "

Example

The following example of configuration is for your reference.

```
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. Migrate data from Faiss to Milvus

Run MilvusDM to import data from Faiss to Milvus with the following command.

```
$ milvusdm --yaml F2M.yaml
```

What's next

- If you are interested in migrating data in other forms into Milvus,
  - Learn how to Migrate Data from HDF5 to Milvus.
- If you are looking for information about how to migrate data from Milvus 1.x to Milvus 2.0,
  - Learn version migration.
- If you are interested in learning more about the data migration tool,
  - Read the overview of MilvusDM.

Milvus 1.x to 2.0

Version Migration

This topic describes how to migrate data from Milvus 1.x to Milvus 2.0 using MilvusDM, an open-source tool specifically designed for Milvus data migration.

MilvusDM does not support migrating data from Milvus 2.0 standalone to Milvus 2.0 cluster.

Prerequisites

You need to install MilvusDM before migrating Milvus data.

1. Download YAML file

Download the `M2M.yaml` file.

```
$ wget https://raw.githubusercontent.com/milvus-io/milvus-tools/main/yamls/M2M.yaml
```

2. Set the parameters

Configuration parameters include:

Parameter	Description	Example
`milvus_version`	Version of Milvus.	2.0.0
`data_path`	Path to the HDF5 files. Set either `data_path` or `data_dir`.	- /Users/zilliz/float_1.h5 - /Users/zilliz/float_2.h5
`data_dir`	Directory of the HDF5 files. Set either `data_path` or `data_dir`.	'/Users/zilliz/Desktop/HDF5_data'
`dest_host`	Milvus server address.	'127.0.0.1'
`dest_port`	Milvus server port.	19530
`mode`	Mode of migration, including `skip`, `append`, and `overwrite`. This parameter works only when the specified collection name exists in the Milvus library.	'append'
`dest_collection_name`	Name of the collection to import data to.	'test_float'
`dest_partition_name` (optional)	Name of the partition to import data to.	'partition_1'
`collection_parameter`	Collection-specific information including vector dimension, index file size, and similarity metric.	"dimension: 512 index_file_size: 1024 metric_type: 'HAMMING' "

The following two examples of configuration are for your reference. The first example involves setting `mysql_parameter`. If you do not use MySQL for managing vector IDs in Milvus 1.x, refer to the second example.

Example 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'
```

Example 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. Migrate data from Milvus to Milvus

Run MilvusDM to import data from Milvus 1.x to Milvus 2.0 with the following command.

```
$ milvusdm --yaml M2M.yaml
```

What's next

- If you are interested in migrating data in other forms into Milvus,
  - Learn how to Migrate Data from Faiss to Milvus.
  - Learn how to Migrate from HDF5 to Milvus.
- If you are interested in learning more about the data migration tool,
  - Read the overview of MilvusDM.

Tools

Milvus CLI

Overview

Milvus Command-Line Interface

Milvus Command-Line Interface (CLI) is a command-line tool that supports database connection, data operations, and import and export of data. Based on Milvus Python SDK, it allows the execution of commands through a terminal using interactive command-line prompts.

Recommended version

In the following table, you can find the recommended versions of PyMilvus and Milvus_CLI according to the version of Milvus that you use.

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

Milvus	PyMilvus	Milvus_CLI
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

Milvus 2.0.0-RC7 and later are not backward compatible with 2.0.0-RC6 and earlier due to changes made to storage formats.

Current version

The current version of Milvus_CLI is 0.1.9. To find your installed version and see if you need to update, run `milvus_cli --version`.

Installation

Install Milvus_CLI

This topic describes how to install Milvus_CLI.

Install from PyPI

You can install Milvus_CLI from PyPI.

Prerequisites

- Install Python 3.8.5 or later
- Install pip

Install via pip

Run the following command to install Milvus_CLI.

```
pip install milvus-cli
```

Install with Docker

You can instal Milvus_CLI with docker.

Prerequisites

Docker 19.03 or later is required.

Install based on Docker image

```
$ docker run -it zilliz/milvus_cli:latest
```

Install from source code

1. Run the following command to download a `milvus_cli` repository.

```
git clone https://github.com/zilliztech/milvus_cli.git
```

2. Run the following command to enter the `milvus_cli` folder.

```
cd milvus_cli
```

3. Run the following command to install Milvus_CLI.

```
python -m pip install --editable .
```

Alternatively, you can install Milvus_CLI from a compressed tarball (`.tar.gz` file). Download a tarball and run `python -m pip install milvus_cli-<version>.tar.gz`.

Install from an .exe file

This installation method only applies to Windows.

Download an .exe file from GitHub and run it to install Milvus_CLI. If successful, `milvus_cli-<version>.exe` pops up as shown in the following figure.

Milvus_CLI

Commands

Milvus_CLI Command Reference

Milvus Command-Line Interface (CLI) is a command-line tool that supports database connection, data operations, and import and export of data.

This topic introduces all supported commands and the corresponding options. Some examples are also included for your reference.

calc distance

Calculates the distance between two vector arrays.

Syntax

```
calc distance
```

Options

Option	Full name	Description
help	n/a	Displays help for using the command.

Example

To calculate the distance between two vector arrays and be prompted for the required input:

```
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

Generates a hybrid timestamp based on an existing hybrid timestamp, timedelta, and incremental time interval.

Syntax

```
calc mkts_from_hybridts -h (int) -m (float)
```

Options

Option	Full name	Description
-h	hybridts	The original hybrid timestamp used to generate a new hybrid timestamp. A non-negative integer that ranges from 0 to 18446744073709551615.
-m	milliseconds	The incremental interval in milliseconds.
help	n/a	Displays help for using the command.

calc mkts_from_unixtime

Generates a hybrid timestamp based on the Unix Epoch time, timedelta, and incremental time interval.

Syntax

```
calc mkts_from_unixtime -e (float) -m (float)
```

Options

Option	Full name	Description
-e	epoch	The known Unix timestamp used to generate a hybrid timestamp. The Unix epoch is the number of seconds that have elapsed since January 1, 1970 (midnight UTC/GMT).
-m	milliseconds	The incremental interval in milliseconds.
help	n/a	Displays help for using the command.

calc hybridts_to_unixtime

Converts a hybrid timestamp to the UNIX timestamp ignoring the logic part.

Syntax

```
calc hybridts_to_unixtime -h (int)
```

Options

Option	Full name	Description
-h	hybridts	The known hybrid timestamp to be converted to a UNIX timestamp. A non-negative integer that ranges from 0 to 18446744073709551615.
help	n/a	Displays help for using the command.

clear

Clears the screen.

Syntax

```
clear
```

Options

Option	Full name	Description
help	n/a	Displays help for using the command.

connect

Connects to Milvus.

Syntax

```
connect [-h (text)] [-p (int)] [-a (text)] [-D]
```

Options

Option	Full name	Description
-h	host	(Optional) The host name. The default is "127.0.0.1".
-p	port	(Optional) The port number. The default is "19530".
-a	alias	(Optional) The alias name of the Milvus link. The default is "default".
-D	disconnect	(Optional) Flag to disconnect from the Milvus server specified by an alias. The default alias is "default".
help	n/a	Displays help for using the command.

Example

```
milvus_cli > connect -h 127.0.0.1 -p 19530 -a default
```

create alias

Specifies unique aliases for a collection.

A collection can have multiple aliases. However, an alias corresponds to a maximum of one collection.

Syntax

```
create alias -c (text) -a (text) [-A] [-t (float)]
```

Options

Option	Full name	Description
-c	collection-name	The name of the collection.
-a	alias-name	The alias.
-A	alter	(Optional) Flag to transfer the alias to a specified collection.
-t	timeout	(Optional) The maximum allowed duration in seconds of an RPC call. Not passing this option indicates that the client keeps waiting until the server responds or an error occurs.
help	n/a	Displays help for using the command.

Examples

Example 1

The following example creates the carAlias1 and carAlias2 aliases for the car collection.

```
milvus_cli > create alias -c car -a carAlias1 -a carAlias2
```

Example 2

Example 2 is based on Example 1.

The following example transfers the carAlias1 and carAlias2 aliases from the car collection to the car2 collection.

```
milvus_cli > create alias -c car2 -A -a carAlias -a carAlias2
```

create collection

Creates a collection.

Syntax

```
create collection -c (text) -f (text) -p (text) [-a] [-d (text)]
```

Options

Option	Full name	Description
-c	collection-name	The nam of the collection.
-f	schema-field	(Multiple) The field schema in the `<fieldName>:<dataType>:<dimOfVector/desc>` format.
-p	schema-primary-field	The name of the primary key field.
-a	schema-auto-id	(Optional) Flag to generate IDs automatically.
-d	schema-description	(Optional) The description of the collection.
help	n/a	Displays help for using the command.

Example

```
milvus_cli > create collection -c car -f id:INT64:primary_field -f vector:FLOAT_VECTOR:128 -f color:INT64:
```

create partition

Creates a partition.

Syntax

```
create partition -c (text) -p (text) [-d (text)]
```

Options

Option	Full name	Description
-c	collection-name	The name of the collection.
-p	partition	The partition name.
-d	description	(Optional) The description of the partition.
help	n/a	Displays help for using the command.

Example

```
milvus_cli > create partition -c car -p new_partition -d test_add_partition
```

create index

Creates an index for a field.

Currently, a collection supports a maximum of one index.

Syntax

```
create index
```

Options

Option	Full name	Description
help	n/a	Displays help for using the command.

Example

To create an index for a field and be prompted for the required input:

```
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

Deletes an alias.

Syntax

```
delete alias -a (text) [-t (float)]
```

Options

Option	Full name	Description
-a	alias-name	The alias.
-t	timeout	(Optional) The maximum allowed duration in seconds of an RPC call. Not passing this option indicates that the client keeps waiting until the server responds or an error occurs.
help	n/a	Displays help for using the command.

delete collection

Deletes a collection.

Syntax

```
delete collection -c (text) [-t (float)]
```

Options

Option	Full name	Description
-c	collection-name	The name of the collection to be deleted.
-t	timeout	(Optional) The maximum allowed duration in seconds of an RPC call. Not passing this option indicates that the client keeps waiting until the server responds or an error occurs.
help	n/a	Displays help for using the command.

Example

```
milvus_cli > delete collection -c car
```

delete entities

Deletes entities.

Syntax

```
delete entities -c (text) -p (text) [-t (float)]
```

Options

Option	Full name	Description
-c	collection-name	The name of the collection that entities to be deleted belongs to.

Option	Full name	Description
-p	partition	(Optional) The name of the partition to be deleted.
-t	timeout	(Optional) The maximum allowed duration in seconds of an RPC call. Not passing this option indicates that the client keeps waiting until the server responds or an error occurs.
help	n/a	Displays help for using the command.

Example

```
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

Deletes a partition.

Syntax

```
delete partition -c (text) -p (text) [-t (float)]
```

Options

Option	Full name	Description
-c	collection-name	The name of the collection that the partition to be deleted belongs to.
-t	timeout	(Optional) The maximum allowed duration in seconds of an RPC call. Not passing this option indicates that the client keeps waiting until the server responds or an error occurs.
-p	partition	The name of the partition to be deleted.
help	n/a	Displays help for using the command.

Example

```
milvus_cli > delete partition -c car -p new_partition
```

delete index

Deletes an index and the corresponding index files.

Currently, a collection supports a maximum of one index.

Syntax

```
delete index -c (text) [-t (float)]
```

Options

Option	Full name	Description
-c	collection-name	The name of the collection.
-t	timeout	(Optional) The maximum allowed duration in seconds of an RPC call. Not passing this option indicates that the client keeps waiting until the server responds or an error occurs.
help	n/a	Displays help for using the command.

Example

```
milvus_cli > delete index -c car
```

describe collection

Shows the detailed information of a collection.

Syntax

```
describe collection -c (text)
```

Options

Option	Full name	Description
-c	collection-name	The name of the collection.
help	n/a	Displays help for using the command.

Example

```
milvus_cli > describe collection -c test_collection_insert
```

describe partition

Shows the detailed information of a partition.

Syntax

```
describe partition -c (text) -p (text)
```

Options

Option	Full name	Description
-c	collection-name	The name of the collection that the partition belongs to.
-p	partition	The name of the partition.
help	n/a	Displays help for using the command.

Example

```
milvus_cli > describe partition -c test_collection_insert -p _default
```

describe index

Shows the detailed information of an index.

Currently, a collection supports a maximum of one index.

Syntax

```
describe index -c (text)
```

Options

Option	Full name	Description
-c	collection-name	The name of the collection.
help	n/a	Displays help for using the command.

exit

Closes the command line window.

Syntax

```
exit
```

Options

Option	Full name	Description
help	n/a	Displays help for using the command.

help

Displays help for using a command.

Syntax

```
help <command>
```

Commands

Command	Description
calc	Calculates the distance between two vector arrays, mkts_from_hybridts, mkts_from_unixtime, or hybridts_to_unixtime.
clear	Clears the screen.
connect	Connects to Milvus.
create	Creates a collection, partition, index, or alias.
delete	Deletes a collection, partition, index, entity, or alias.
describe	Describes a collection, partition, or index.
exit	Closes the command line window.
help	Displays help for using a command.
import	Imports data into a partition.
list	Lists collections, partitions, or indexes.
load	Loads a collection or partition.
load_balance	Performs load balancing on a query node.
query	Shows query results that match all the criteria that you enter.
release	Releases a collection or partition.
search	Performs a vector similarity search or hybrid search.
show	Shows the current collection, progress of entity loading, progress of entity indexing, or segment information.

Command	Description
version	Shows the version of Milvus_CLI.

import

Imports local or remote data into a partition.

Syntax

```
import -c (text)[-p (text)][-t (float)] <file_path>
```

Options

Option	Full name	Description
-c	collection-name	The name of the collection that the data is inserted into.
-p	partition	(Optional) The name of the partition that the data is inserted into. Not passing this partition option indicates choosing the "_default" partition.
-t	timeout	(Optional) The maximum allowed duration in seconds of an RPC call. Not passing this option indicates that the client keeps waiting until the server responds or an error occurs.
help	n/a	Displays help for using the command.

Example 1

The following example imports a local CSV file.

```
milvus_cli > import -c car 'examples/import_csv/vectors.csv'

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
-------------------------  ------------------
```

Example 2

The following example imports a remote CSV file.

```
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

Lists all collections.

Syntax

```
list collections [-t (float)][-l (boolean)]
```

Options

Option	Full name	Description
-t	time-out	(Optional) The maximum allowed duration in seconds of an RPC call. Not passing this option indicates that the client keeps waiting until the server responds or an error occurs.
-l	show-loaded	(Optional) Shows the loaded collections only.
help	n/a	Displays help for using the command.

list indexes

Lists all indexes for a collection.

Currently, a collection supports a maximum of one index.

Syntax

```
list indexes -c (text)
```

Options

Option	Full name	Description
-c	collection-name	The name of the collection.
help	n/a	Displays help for using the command.

list partitions

Lists all partitions of a collection.

Syntax

```
list partitions -c (text)
```

Options

Option	Full name	Description
-c	collection-name	The name of the collection.
help	n/a	Displays help for using the command.

load

Loads a collection or partition from hard drive space into RAM.

Syntax

```
load -c (text) [-p (text)]
```

Options

Option	Full name	Description
-c	collection-name	The name of the collection that the partition belongs to.
-p	partition	(Optional/Multiple) The name of the partition.
help	n/a	Displays help for using the command.

load_balance

Performs load balancing by transferring segments from a source query node to a destination one.

Syntax

```
load_balance -s (int) -d (int) -ss (int) [-t (int)]
```

Options

Option	Full name	Description
-s	src-node-id	The ID of the source query node to be balanced.
-d	dst-node-id	(Multiple) The ID of the destination query node to transfer segments to.
-ss	sealed-segment-ids	(Multiple) The ID of the sealed segment to be transferred.
-t	timeout	(Optional) The timeout in seconds.
help	n/a	Displays help for using the command.

query

Shows query results that match all the criteria that you enter.

Syntax

```
query
```

Options

Option	Full name	Description
help	n/a	Displays help for using the command.

Example

Example 1

To perform a query and be prompted for the required input:

```
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
```

Example 2

To perform a query and be prompted for the required input:

```
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

Releases a collection or partition from RAM.

Syntax

```
release -c (text) [-p (text)]
```

Options

Option	Full name	Description
-c	collection-name	The name of the collection that the partition belongs to.
-p	partition	(Optional/Multiple) The name of the partition.
help	n/a	Displays help for using the command.

search

Performs a vector similarity search or hybrid search.

Syntax

`search`

Options

Option	Full name	Description
help	n/a	Displays help for using the command.

Examples

Example 1

To perform a search on a csv file and be prompted for the required input:

```
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]:
```

Example 2

To perform a search on an indexed collection and be prompted for the required input:

```
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]:
```

Example 3

To perform a search on a non-indexed collection and be prompted for the required input:

```
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

Shows the current connection.

Syntax

```
show connection [-a]
```

Options

Option	Full name	Description
-a	all	(Optional) Flag to show all connections.
help	n/a	Displays help for using the command.

show index_progress

Shows the progress of entity indexing.

Syntax

```
show index_progress -c (text) [-i (text)]
```

Options

Option	Full name	Description
-c	collection-name	The name of the collection that the entities belong to.
-i	index	(Optional) The name of the index.
help	n/a	Displays help for using the command.

show loading_progress

Shows the progress of entity loading.

Syntax

```
show loading_progress -c (text) [-p (text)]
```

Options

Option	Full name	Description
-c	collection-name	The name of the collection that the entities belong to.
-p	partition	(Optional/Multiple) The name of the loading partition.
help	n/a	Displays help for using the command.

show query_segment

Shows the segment information of a collection.

Syntax

```
show query_segment -c (text) [-t (float)]
```

Options

Option	Full name	Description
-c	collection-name	The name of the collection.
-t	timeout	(Optional) The maximum allowed duration in seconds of an RPC call. Not passing this option indicates that the client keeps waiting until the server responds or an error occurs.
help	n/a	Displays help for using the command.

version

Shows the version of Milvus_CLI.

Syntax

```
version
```

Options

Option	Full name	Description
help	n/a	Displays help for using the command.

You can also check the version of Milvus_CLI in a shell as shown in the following example. In this case, milvus_cli version acts as a command.

Example

```
$ milvus_cli --version
Milvus_CLI v0.1.7
```

MilvusDM

Overview

MilvusDM (Milvus Data Migration) is an open-source tool designed specifically for importing and exporting data with Milvus. MilvusDM allows you to migrate data in a specific collection or partition. To substantially improve data management efficiency and reduce DevOps costs, MilvusDM supports the following migration channels:

- Milvus to Milvus: Migrates data between instances of Milvus.
- Faiss to Milvus: Imports unzipped data from Faiss to Milvus.
- HDF5 to Milvus: Imports HDF5 files into Milvus.
- Milvus to HDF5: Saves the data in Milvus as HDF5 files.

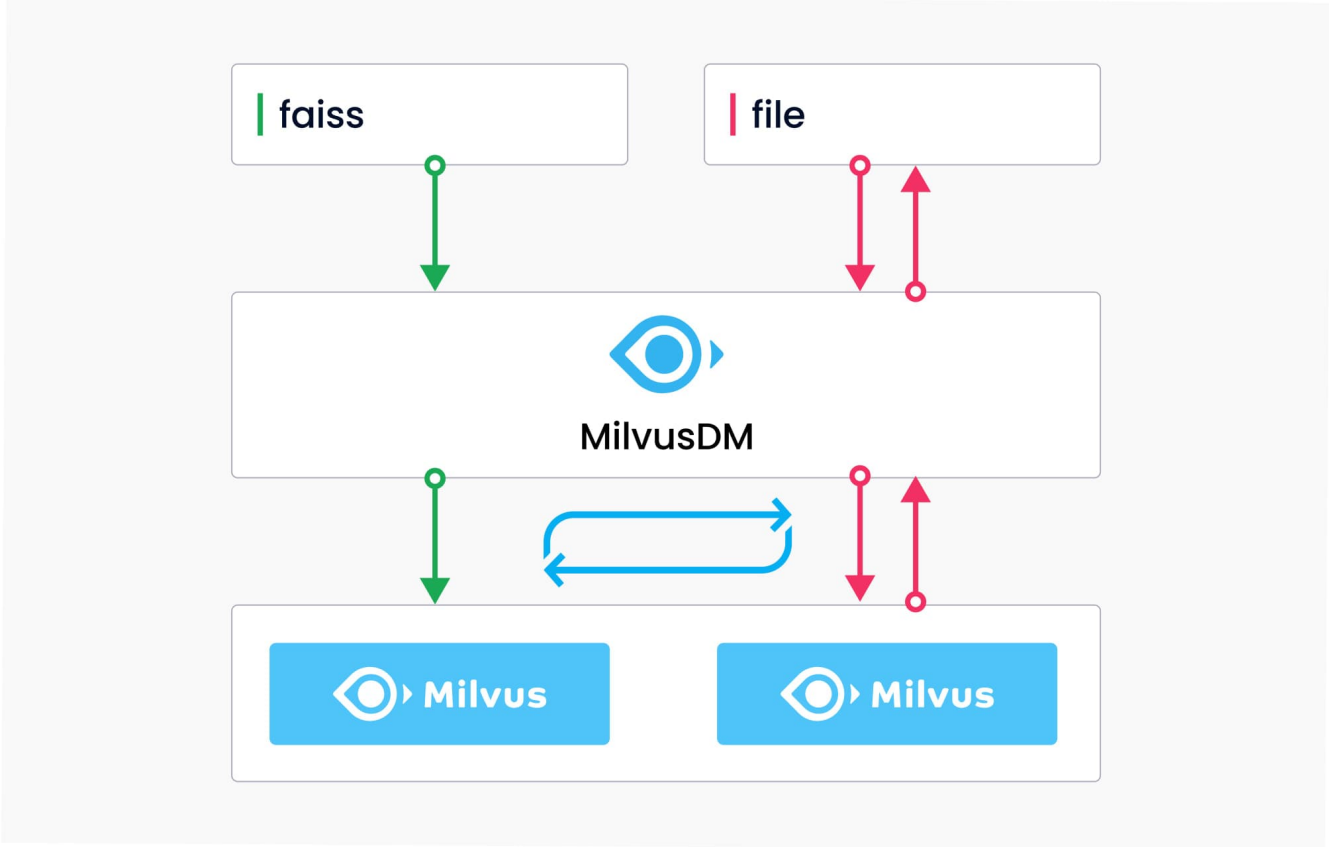


Figure 6: MilvusDM

MilvusDM is hosted on GitHub. To install MilvusDM, run:

```
pip3 install pymilvusdm
```

MilvusDM File Structure

The flow chart below shows how MilvusDM performs different tasks according to the .yaml file it receives:

MilvusDM file structure:

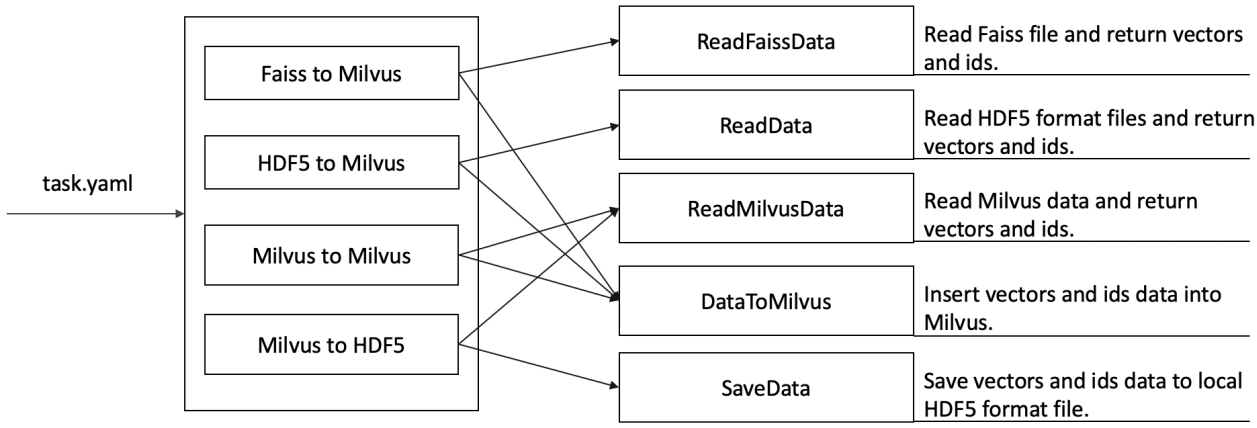


Figure 7: File structure

- pymilvusdm
  - core
    - milvus_client.py: Performs client operations in Milvus.
    - read_data.py: Reads the HDF5 files on your local drive. (Add your code here to support reading data files in other formats.)
    - read_faiss_data.py: Reads Faiss data files.
    - read_milvus_data.py: Reads Milvus data files.
    - read_milvus_meta.py: Reads Milvus metadata.
    - data_to_milvus.py: Creates collections or partitions as specified in .yaml files and imports vectors and the corresponding IDs into Milvus.
    - save_data.py: Saves data as HDF5 files.
    - write_logs.py: Writes `debug`/`info`/`error` logs during runtime.
  - faiss_to_milvus.py: Imports Faiss data into Milvus.
  - hdf5_to_milvus.py: Imports HDF5 files into Milvus.
  - milvus_to_milvus.py: Migrates data from a source Milvus to a target Milvus.
  - milvus_to_hdf5.py: Saves Milvus data as HDF5 files.
  - main.py: Executes tasks as specified by the received .yaml file.
  - setting.py: Stores configurations for MilvusDM operation.
- setup.py: Creates and uploads pymilvusdm file packages to PyPI (Python Package Index). ### Enhancement Plan In future releases, MilvusDM will provide more new features including MilvusDump and MilvusRestore to support exporting all Milvus data, restoring data in specified collections and partitions, resuming interrupted download and more.

The MilvusDM project is open sourced on GitHub. Any contribution to the project is welcome. Give it a star ☆, and feel free to file an issue or submit your own code!

Installation

Install MilvusDM

MilvusDM is an open-source tool designed specifically for importing and exporting data with Milvus. This page shows you how to install MilvusDM.

The pymilvusdm2.0 is used for migrating data from Milvus(0.10.x or 1.x) to Milvus2.x.

Before you begin

Ensure you meet the requirements for operating system and software before installing MilvusDM.

Operating system	Supported versions
CentOS	7.5 or higher
Ubuntu LTS	18.04 or higher

Software	Version
Milvus	0.10.x or 1.x or 2.x
Python3	3.7 or higher
pip3	Should be in correspondence to the Python version.

Install MilvusDM

1. Add the following two lines to the ~/.bashrc file:

```
export MILVUSDM_PATH='/home/$user/milvusdm'
export LOGS_NUM=0
```

- `MILVUSDM_PATH`: This parameter defines the working path of MilvusDM. Logs and data generated by MilvusDM will be stored in this path. The default value is `/home/$user/milvusdm`.
- `LOGS_NUM`: MilvusDM generates one log file each day. This parameter defines the number of log files you would like to save. The default value is 0, which means all log files are saved.

2. Configure environment variables:

```
$ source ~/.bashrc
```

3. Install MilvusDM with `pip`:

```
$ pip3 install pymilvusdm==2.0
```

Attu

Overview

Attu

Attu is an efficient open-source management tool for Milvus. It features an intuitive graphical user interface (GUI), allowing you to easily interact with your databases. With just a few clicks, you can visualize your cluster status, manage metadata, perform data queries, and much more.

Attu is an open source project of Zilliz.

Attu_overview

Features

Attu is under rapid development and new features are added on a weekly basis. We will release a new version every time when a new feature is ready.

Below is the features we have to offer:

- View milvus cluster statistics in a glance. (TBD)

view_cluster_statistics

- Browse, query, and manage collections in a simple and straightforward way.

manage_collections

- Perform CRUD or bulk operations with just a few clicks.

attu_operations

- Create vector index instantly.

attu_create_index

- Conduct vector searches in a brand new way.

attu_conduct_search

- New code-mode provides a better user experience for you.

code_mode

Learn more about how to install Attu.

Contribution

Attu is an open-source project. All contributions are welcome. Pleae read our Contribute guide before making contributions.

If you find a bug or want to request a new feature, please create a GitHub Issue, and make sure that the same issue has not been created by someone else.

Installation

Install Attu

This topic describes how to install Attu, an efficient open-source management tool for Milvus.

Install with Docker ComposeInstall with Helm ChartInstall with Package

Prerequisites

- Milvus installed on your local device or cluster.
- Docker 19.03 or later

Attu only supports Milvus 2.x.

Start an Attu instance

```
docker run -p 8000:3000 -e HOST_URL=http://{ your machine IP }:8000 -e MILVUS_URL={your machine IP}:19530 
```

Once you start the docker, visit `http://{ your machine IP }:8000` in your browser, and click Connect to enter the Attu service.

Attu_install

Contribution

Attu is an open-source project. All contributions are welcome. Please read our Contribute guide before making contributions.

If you find a bug or want to request a new feature, please create a GitHub Issue, and make sure that the same issue has not been created by someone else.

Attu Overview Page

This topic describes the Home page of Attu.

Attu consists of Overview page, Collection page, Vector Search page, and System View page, corresponding to the four icons on the left-side navigation pane respectively.

The Overview page lists the following information:

1. Loaded Collections: the number of loaded collections.
2. All Collections: the total number of collections.
3. Data: the total number of entities.

4. Loaded For Search card: an interactive shortcut panel that allows you to perform a vector search on or release the collection.

Attu Overview

Manage Collections

Manage Collections with Attu

This topic describes how to manage collections with Attu.

Create a collection

1. Click the Collection icon on the left-side navigation pane and then click Create Collection. The Create Collection dialog box appears as shown below.

Create Collection dialog box

2. Enter the required information. This example creates a collection named `test` with a primary key field, a vector field, and a scalar field. You can add scalar fields as needed.

Create Collection dialog box

3. Click Create to create a collection.

Create Collection dialog box

Delete a collection

1. Tick the collection you want to delete in the data grid.
2. Click the Trash icon and the Delete Collection dialog box appears as shown below.
3. Type `delete` to confirm the deletion.
4. Click Delete to delete the collection.

Deleting a collection is irreversible.

Delete Collection dialog box

Load a collection

1. Hover on the collection you want to load, the Load icon appears on the right end of the entry.

Load Collection

2. Click the Load icon and the Load Collection dialog box appears.
3. Click Load in the Load Collection dialog box.

Load Collection

4. Loading a collection might take a while. If successful, Loaded For Search appears in the Status column.

Load Collection

Release a collection

1. Hover on the loaded collection you want to release, the Release icon appears on the right end of the entry.

Release Collection

2. Click the Release icon and the Release Collection dialog box appears.
3. Click Release in the Release Collection dialog box.
4. If successful, the Status becomes Unloaded in the Status column.

Release Collection

View the schema of a collection

1. Click the name of the collection that you want to view the schema of, and the corresponding detail page appears.
2. Click Schema on the detail page, which lists the information of all fields.

Attributes of a schema include:

- Field Name
- Field Type
- Dimension (Only applies to vector fields)
- Index Type (Only applies to vector fields)
- Index Parameters (Only applies to vector fields)
- Collection description

Collection Schema

Manage Partitions

Manage Partitions with Attu

This topic describes how to manage partitions with Attu.

Milvus creates a partition automatically after a collection is created, which cannot be deleted.

Create a partition

1. Click Partitions tab on the Collection page.
2. Click Create Partition on the Partitions tab page, and the Create Partition dialog box appears as shown below.
3. In the Create Partition dialog box, enter the new partition name in the Name field.
4. Click Create to create a partition.

Create Partition

If successful, the new partition appears on the Partitions tab page.

Create Partition

Choose the Default partition or the newly created partition to store imported data as needed.

Delete a partition

1. Tick the partition you want to delete.
2. Click the Trash icon and the Delete Partition dialog box appears as shown below.
3. Type `delete` to confirm the deletion.
4. Click Delete to delete the partition.

Delete Partition

Manage Data

Manage Data with Attu

This topic describes how to manage data with Attu.

Import data

This example imports 20,000 rows of data. Importing data appends data instead of overwriting data.

1. Click Import Data on the Collection page. The Import Data dialog box appears as shown below.

Import Data

2. Select the collection you want to import data to in the Collection dropdown list.
3. Select the partition you want to import data to in the Partition dropdown list.

4. Click Choose CSV File and choose a CSV file.

Ensure that the CSV file meets the following criteria:

Column names are the same as the field names specified in the schema;

The file is smaller than 150MB and the row count is less than 100,000.

5. After a legal CSV file is selected, you can then proceed by clicking Next.

Import Data

6. On the new dialog box, you can match the field names by clicking the corresponding cells in the dropdown lists.

We recommend making the headers (column names) as the first row in your CSV file.

Import Data

7. After confirming the column names corresponding to the field names, click Import Data to import the CSV file into Milvus. Importing data might take a while.

Import Data

8. If successful, the row count status updates in the Entity Count column of the collection. On the corresponding Partition tab page, the row count status updates in the Entity Count column of the partition your imported data in. It might take a while for the entity count to update.

Import Data

Export Data

1. Click Data Query on the Collection page. On the Data Query tab page, enter query conditions in the field and then click Query to retrieve all query results that match your query conditions.
2. Click the Download icon to download the query results as a CSV file.

Export Data

Manage Index

Manage Index with Attu

This topic describes how to manage an index with Attu.

Create indexes

This example builds an IVF_FLAT index with Euclidean distance as the similarity metrics and an `nlist` value of `1024`.

1. Click Schema on the Collection page. On the Schema tab page, click CREATE INDEX and the Create Index dialog box appears.
2. In the Create Index dialog box, select IVF_FLAT from the Index Type dropdown list, select L2 from the Metric Type dropdown list, and enter `1024` in the `nlist` field.
3. (Optional) Turn on View Code and the Code View page appears. You can check the code in Python or Node.js as you want.
4. Click Create to create the index.

If successful, the type of the index you created appears in the Index Type column for the vector field.

Create Index

Create Index

Delete indexes

1. Click the Trash icon in the Index Type column and the Delete Index dialog box appears.
2. Enter `delete` to confirm the deletion and click Delete to delete the indexes.

If successful, CREATE INDEX button appears in the Index Type column.

Delete Index

Query Data

Query Data with Attu

This topic describes how to query data with Attu.

Query data with advanced filters

1. Click the entry of the collection that you want to query data in, and the corresponding detail page appears.
2. On the Data Query tab page, click the Filter icon and the Advanced Filter dialog box appears.
3. Specify a complicated query condition such as color > 10 && color < 20 by using the Field Name dropdown lists, the Logic dropdown lists, the Value fields, and the AND operator. Then click Apply Filter to apply the query condition.

Query Data

3. Click Query to retrieve all query results that match the query condition.

Query Data

Query data with Time Travel

TBD (Not supported yet)

Delete data

1. Tick the entities you want to delete and click the Trash icon.
2. Type `delete` to confirm the deletion in the Delete entities dialog box.
3. Click Delete to delete the selected entities.

Delete Data

You can perform a query to retrieve the deleted entities. No results will be returned if the deletion is successful.

Delete Data

Search Data

Search Data with Attu

This topic describes how to search data with Attu.

Conduct a vector similarity search

On the basis of the regular vector similarity search, you can perform hybrid search of search with Time Travel.

Load the collection to memory

All CRUD operations within Milvus are executed in memory. Load the collection to memory before conducting a vector similarity search. See Load a collection for more instruction.

Search Data

Set search parameters

1. Select the collection and the vector field you wish to search in in the dropdown lists of the Choose collection and field section.
2. In the Enter vector value field, enter a vector (or vectors) with the same dimensions of the selected field as the target vector(s) to search with.
3. In the Set search parameters section, specify the specific parameter(s) to the index and other search-related parameters.

Search Data

Hybrid search with advanced filters (optional)

Click Advanced Filter and the Advanced Filter dialog box appears. You can use the AND or OR operators to combine multiple conditions into a compound condition. The filter expression updates automatically with any changes to the conditions. See boolean expression rule for more information.

Search Data

Search with Time Travel (optional)

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.

1. Click Time Travel, and select a time point in the dialog box that appears.

Search Data

2. Specify the number of search results to return in the TopK dropdown list.
3. Click Search to retrieve the nearest search results, which indicate the most similar vectors.

Search Data Search Data

System Status

Monitor System with Attu

This topic describes how to monitor your Milvus system with Attu.

System view

Click the System View icon on the left-side navigation pane to visit the System View page.

System View

The System View dashboard consists of the following tiles:

- Disk: Shows the usage of storage space.
- Memory: Shows the usage of memory.
- Qps: Shows the last 10 monitored QPS. Each node represents a time point. Hover on a node to view the QPS monitored at the time point.
- Latency: Shows the last 10 monitored latency. Each node represents a time point. Hover on a node to view the latency monitored at the time point.

System View System View

- Topology: Shows the structure of your Milvus instance. Click Milvus node or a coordinator node and the corresponding information appears in the Info tile on the right.
- Info: Shows the hardware, system, and configuration information of the selected node.

Node list view

All nodes managed by their parent coordinator node are listed in the table. You can sort the nodes by metrics including CPU Core Count, CPU Core Usage, Disk Usage, and Memory Usage.

Node List View

Right to the table is a mini topology showing the selected node and its coordinator node. Under the mini topology is a tile showing relevant information.

Click the down arrow to collapse rows in the table.

Attu FAQ

Why is Attu throwing a network error?

A: Check whether you have assigned a correct value to `HOST_URL` in the `docker run` command. Alternatively, you can enter `{HOST_URL}/api/v1/healthy` in the address bar of your browser to check the network status of Attu.

Why did Attu fail to connect to Milvus?

A: Ensure that Milvus and Attu are on the same network.

How do I use Attu with K8s?

A: You can install Attu while deploying Milvus with Helm.

Reference

Architecture

Overview

Milvus Architecture Overview

Built on top of popular vector search libraries including Faiss, Annoy, HNSW, and more, Milvus was designed for similarity search on dense vector datasets containing millions, billions, or even trillions of vectors. Before proceeding, familiarize yourself with the basic principles of embedding retrieval.

Milvus also supports data sharding, data persistence, streaming data ingestion, hybrid search between vector and scalar data, time travel, and many other advanced functions. The platform offers performance on demand and can be optimized to suit any embedding retrieval scenario. We recommend deploying Milvus using Kubernetes for optimal availability and elasticity.

Milvus adopts a shared-storage architecture featuring storage and computing disaggregation and horizontal scalability for its computing nodes. Following the principle of data plane and control plane disaggregation, Milvus comprises four layers: access layer, coordinator service, worker node, and storage. These layers are mutually independent when it comes to scaling or disaster recovery.

Architecture_diagram

For more details about Milvus' architecture, see Computing/Storage Disaggregation and Main Components.

Storage/Computing

Storage/Computing Disaggregation

Following the principle of data plane and control plane disaggregation, Milvus comprises four layers that are mutually independent in terms of scalability and disaster recovery.

Access layer

Composed of a group of stateless proxies, the access layer is the front layer of the system and endpoint to users. It validates client requests and reduces the returned results:

- Proxy is in itself stateless. It provides a unified service address using load balancing components such as Nginx, Kubernetes Ingress, NodePort, and LVS.
- As Milvus employs a massively parallel processing (MPP) architecture, the proxy aggregates and post-process the intermediate results before returning the final results to the client.

Coordinator service

The coordinator service assigns tasks to the worker nodes and functions as the system's brain. The tasks it takes on include cluster topology management, load balancing, timestamp generation, data declaration, and data management.

There are four coordinator types: root coordinator (root coord), data coordinator (data coord), query coordinator (query coord), and index coordinator (index coord).

Root coordinator (root coord)

Root coord handles data definition language (DDL) and data control language (DCL) requests, such as create or delete collections, partitions, or indexes, as well as manage TSO (timestamp Oracle) and time ticker issuing.

Query coordinator (query coord)

Query coord manages topology and load balancing for the query nodes, and handoff from growing segments to sealed segments.

Data coordinator (data coord)

Data coord manages topology of the data nodes, maintains metadata, and triggers flush, compact, and other background data operations.

Index coordinator (index coord)

Index coord manages topology of the index nodes, builds index, and maintains index metadata.

Worker nodes

The arms and legs. Worker nodes are dumb executors that follow instructions from the coordinator service and execute data manipulation language (DML) commands from the proxy. Worker nodes are stateless thanks to separation of storage and computation, and can facilitate system scale-out and disaster recovery when deployed on Kubenetes. There are three types of worker nodes:

Query node

Query node retrieves incremental log data and turn them into growing segments by subscribing to the log broker, loads historical data from the object storage, and runs hybrid search between vector and scalar data.

Data node

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.

Index node

Index node builds indexes. Index nodes do not need to be memory resident, and can be implemented with the serverless framework.

Storage

Storage is the bone of the system, responsible for data persistence. It comprises meta storage, log broker, and object storage.

Meta storage

Meta storage stores snapshots of metadata such as collection schema, node status, and message consumption checkpoints. Storing metadata demands extremely high availability, strong consistency, and transaction support, so Milvus chose etcd for meta store. Milvus also uses etcd for service registration and health check.

Object storage

Object storage stores snapshot files of logs, index files for scalar and vector data, and intermediate query results. Milvus uses MinIO as object storage and can be readily deployed on AWS S3 and Azure Blob, two of the world's most popular, cost-effective storage services. However, object storage has high access latency and charges by the number of queries. To improve its performance and lower the costs, Milvus plans to implement cold-hot data separation on a memory- or SSD-based cache pool.

Log broker

The log broker is a pub-sub system that supports playback. It is responsible for streaming data persistence, execution of reliable asynchronous queries, event notification, and return of query results. It also ensures integrity of the incremental data when the worker nodes recover from system breakdown. Milvus cluster uses Pulsar as log broker; Milvus standalone uses RocksDB as log broker. Besides, the log broker can be readily replaced with streaming data storage platforms such as Kafka and Pravega.

Milvus is built around log broker and follows the "log as data" principle, so Milvus 2.0 does not maintain a physical table but guarantees data reliability through logging persistence and snapshot logs.

Log_mechanism

The log broker is the backbone of Milvus 2.0. It is responsible for data persistence and read-write disaggregation, thanks to its innate pub-sub mechanism. The above illustration shows a simplified depiction of the mechanism, where the system is divided into two roles, log broker (for maintaining the log sequence) and log subscriber. The former records all operations that change collection states; the latter subscribes to the log sequence to update the local data and provides services in the form of read-only copies. The pub-sub mechanism also makes room for system extendability in terms of change data capture (CDC) and globally-distributed deployment.

For more details about Milvus' architecture, see Main Components.

Main Components

There are two modes for running Milvus: Standalone and Cluster. These two modes share the same features. You can choose a mode that best fits your dataset size, traffic data, and more. For now, Milvus standalone cannot be "online" upgraded to Milvus cluster.

Milvus standalone

Milvus standalone includes three components:

- Milvus: The core functional component.
- etcd: The metadata engine, which accesses and stores metadata of Milvus' internal components, including proxies, index nodes, and more.
- MinIO: The storage engine, which is responsible for data persistence for Milvus.

Standalone_architecture

Milvus cluster

Milvus cluster includes eight microservice components and three third-party dependencies. All microservices can be deployed on Kubernetes, independently from each other.

Microservice components

- Root coord
- Proxy
- Query coord
- Query node
- Index coord
- Index node
- Data coord
- Data node

Third-party dependencies

- etcd: Stores metadata for various components in the cluster.
- MinIO: Responsible for data persistence of large files in the cluster, such as index and binary log files.
- Pulsar: Manages logs of recent mutation operations, outputs streaming log, and provides log publish-subscribe services.

Distributed_architecture

For more details about Milvus' architecture, see Computing/Storage Disaggregation.

Data Processing

This article provides a detailed description of the implementation of data insertion, index building, and data query in Milvus.

Data insertion

You can specify a number of shards for each collection in Milvus, each shard corresponding to a virtual channel (vchannel). As the following figure shows, Milvus 2.0 assigns each vchannel in the log broker a physical channel (pchannel). Any incoming insert/delete request is routed to shards based on the hash value of primary key.

Validation of DML requests is moved forward to proxy because Milvus does not have complicated transactions. Proxy would request a timestamp for each insert/delete request from TSO (Timestamp Oracle), which is the timing module that colocates with the root coordinator. With the older timestamp being overwritten by the newer one, timestamps are used to determine the sequence of data requests being processed. Proxy retrieves information in batches from data coord including entities' segments and primary keys to increase overall throughput and avoid overburdening the central node.

Channels 1

Both DML (data manipulation language) operations and DDL (data definition language) operations are written to the log sequence, but DDL operations are only assigned one channel because of their low frequency of occurrence.

Channels 2

Vchannels are maintained in the underlying log broker nodes. Each channel is physically indivisible and available for any but only one node. When data ingestion rate reaches bottleneck, consider two things: Whether the log broker node is overloaded and needs to be scaled, and whether there are sufficient shards to ensure load balance for each node.

Write log sequence

The above diagram encapsulates four components involved in the process of writing log sequence: proxy, log broker, data node, and object storage. The process involves four tasks: validation of DML requests, publication-subscription of log sequence, conversion from streaming log to log snapshots, and persistence of log snapshots. The four tasks

are decoupled from each other to make sure each task is handled by its corresponding node type. Nodes of the same type are made equal and can be scaled elastically and independently to accommodate various data loads, massive and highly fluctuating streaming data in particular.

Index building

Index building is performed by index node. To avoid frequent index building for data updates, a collection in Milvus is divided further into segments, each with its own index.

Index building

Milvus supports building index for each vector field, scalar field and primary field. Both the input and output of index building engage with object storage: The index node loads the log snapshots to index from a segment (which is in object storage) to memory, deserializes the corresponding data and metadata to build index, serializes the index when index building completes, and writes it back to object storage.

Index building mainly involves vector and matrix operations and hence is computation- and memory-intensive. Vectors cannot be efficiently indexed with traditional tree-based indexes due to their high-dimensional nature, but can be indexed with techniques that are more mature in this subject, such as cluster- or graph-based indexes. Regardless its type, building index involves massive iterative calculations for large-scale vectors, such as Kmeans or graph traverse.

Unlike indexing for scalar data, building vector index has to take full advantage of SIMD (single instruction, multiple data) acceleration. Milvus has innate support for SIMD instruction sets, e.g., SSE, AVX2, and AVX512. Given the "hiccup" and resource-intensive nature of vector index building, elasticity becomes crucially important to Milvus in economical terms. Future Milvus releases will further explorations in heterogeneous computing and serverless computation to bring down the related costs.

Besides, Milvus also supports scalar filtering and primary field query. It has inbuilt indexes to improve query efficiency, e.g., Bloom filter indexes, hash indexes, tree-based indexes, and inverted indexes, and plans to introduce more external indexes, e.g., bitmap indexes and rough indexes.

Data query

Data query refers to the process of searching a specified collection for k number of vectors nearest to a target vector or for all vectors within a specified distance range to the vector. Vectors are returned together with their corresponding primary key and fields.

Data query

A collection in Milvus is split into multiple segments, and the query nodes loads indexes by segment. When a search request arrives, it is broadcast to all query nodes for a concurrent search. Each node then prunes the local segments, searches for vectors meeting the criteria, and reduces and returns the search results.

Query nodes are independent from each other in a data query. Each node is responsible only for two tasks: Load or release segments following the instructions from query coord; conduct a search within the local segments. And proxy is responsible for reducing search results from each query node and returning the final results to the client.

Handoff

There are two types of segments, growing segments (for incremental data), and sealed segments (for historical data). Query nodes subscribe to vchannel to receive recent updates (incremental data) as growing segments. When a growing segment reaches a predefined threshold, data coord seals it and index building begins. Then a handoff operation initiated by query coord turns incremental data to historical data. Query coord will distribute sealed segments evenly among all query nodes according to memory usage, CPU overhead, and segment number.

System Configurations

Milvus System Configurations Checklist

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 ## Similarity Metrics

In Milvus, similarity metrics are used to measure similarities among vectors. Choosing a good distance metric helps improve the classification and clustering performance significantly.

The following table shows how these widely used similarity metrics fit with various input data forms and Milvus indexes.

Floating point embeddings Binary embeddings

Similarity Metrics

Index Types

Euclidean distance (L2)

Inner product (IP)

```
<td class="tg-0pky" rowspan="2"><ul><li>FLAT</li><li>IVF_FLAT</li><li>IVF_SQ8</li><li>IVF_PQ</li><li>HNSW<
```

Distance Metrics

Index Types

Jaccard

Tanimoto

Hamming

BIN_FLAT

BIN_IVF_FLAT

Superstructure

Substructure

BIN_FLAT

Euclidean distance (L2)

Essentially, Euclidean distance measures the length of a segment that connects 2 points.

The formula for Euclidean distance is as follows:

where a = (a1, a2,⋯, an) and b = (b1, b2,⋯, bn) are two points in n-dimensional Euclidean space

It’s the most commonly used distance metric and is very useful when the data is continuous.

Inner product (IP)

The IP distance between two embeddings are defined as follows:

Where A and B are embeddings, `||A||` and `||B||` are the norms of A and B.

IP is more useful if you are more interested in measuring the orientation but not the magnitude of the vectors.

$$d(\boldsymbol{a},\boldsymbol{b}) = d(\boldsymbol{b},\boldsymbol{a}) = \sqrt{\sum_{i=1}^{n}(b_i - a_i)^2}$$

Figure 8: euclidean

$$p(A,B) = A \cdot B = \sum_{i=1}^{n} a_i \times b_i$$

Figure 9: ip

If you use IP to calculate embeddings similarities, you must normalize your embeddings. After normalization, the inner product equals cosine similarity.

Suppose X’ is normalized from embedding X:

$$X' = (x'_1, x'_2, \ldots, x'_n), X' \in^n$$

Figure 10: normalize

The correlation between the two embeddings is as follows:

Jaccard distance

Jaccard similarity coefficient measures the similarity between two sample sets and is defined as the cardinality of the intersection of the defined sets divided by the cardinality of the union of them. It can only be applied to finite sample sets.

Jaccard distance measures the dissimilarity between data sets and is obtained by subtracting the Jaccard similarity coefficient from 1. For binary variables, Jaccard distance is equivalent to the Tanimoto coefficient.

Tanimoto distance

For binary variables, the Tanimoto coefficient is equivalent to Jaccard distance:

In Milvus, the Tanimoto coefficient is only applicable for a binary variable, and for binary variables, the Tanimoto coefficient ranges from 0 to +1 (where +1 is the highest similarity).

For binary variables, the formula of Tanimoto distance is:

The value ranges from 0 to +infinity.

$$x'_i = \frac{x_i}{\| X \|} = \frac{x_i}{\sqrt{\sum_{i=1}^{n} (x_i)^2}}$$

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

$$T(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

Figure 14: tanimoto coefficient

$$d_t = \frac{A \cdot B}{|A|^2 + |B|^2 - A \cdot B}$$

Figure 15: tanimoto distance

Hamming distance

Hamming distance measures binary data strings. The distance between two strings of equal length is the number of bit positions at which the bits are different.

For example, suppose there are two strings, 1101 1001 and 1001 1101.

11011001 ⊕ 10011101 = 01000100. Since, this contains two 1s, the Hamming distance, d (11011001, 10011101) = 2.

Superstructure

The Superstructure is used to measure the similarity of a chemical structure and its superstructure. The less the value, the more similar the structure is to its superstructure. Only the vectors whose distance equals to 0 can be found now.

Superstructure similarity can be measured by:

$$1 - \frac{N_{A\&B}}{N_A}$$

Figure 16: superstructure

Where

- B is the superstructure of A
- NA specifies the number of bits in the fingerprint of molecular A.
- NB specifies the number of bits in the fingerprint of molecular B.
- NAB specifies the number of shared bits in the fingerprint of molecular A and B.

Substructure

The Substructure is used to measure the similarity of a chemical structure and its substructure. The less the value, the more similar the structure is to its substructure. Only the vectors whose distance equals to 0 can be found now.

Substructure similarity can be measured by:

$$1 - \frac{N_{A\&B}}{N_B}$$

Figure 17: substructure

Where

- B is the substructure of A
- NA specifies the number of bits in the fingerprint of molecular A.
- NB specifies the number of bits in the fingerprint of molecular B.
- NAB specifies the number of shared bits in the fingerprint of molecular A and B.

FAQ

Why is the top1 result of a vector search not the search vector itself, if the metric type is inner product?

This occurs if you have not normalized the vectors when using inner product as the distance metric.

What is normalization? Why is normalization needed?

Normalization refers to the process of converting an embedding (vector) so that its norm equals 1. If you use Inner Product to calculate embeddings similarities, you must normalize your embeddings. After normalization, inner product equals cosine similarity.

See Wikipedia for more information.

Why do I get different results using Euclidean distance (L2) and inner product (IP) as the distance metric?

Check if the vectors are normalized. If not, you need to normalize the vectors first. Theoretically speaking, similarities worked out by L2 are different from similarities worked out by IP, if the vectors are not normalized.

Vector Index

Indexing, a process of organizing data, is a huge component of what makes it possible to efficiently query the million-, billion-, or even trillion-vector datasets that vector databases rely on.

Accelerating vector similarity search

Similarity search engines work by comparing an input to the objects in a database to find those that are most similar to the input. Indexing is the process of efficiently organizing data, and it plays a major role in making similarity search useful by dramatically accelerating time-consuming queries on large datasets. After a massive vector dataset is indexed, queries can be routed to clusters, or subsets of data, that are most likely to contain vectors similar to an input query. In practice, this means a certain degree of accuracy is sacrificed to speed up queries on really large vector datasets.

To improve query performance, you can specify an index type for each vector field. Currently, a vector field only supports one index type. Milvus automatically deletes the old index when switching the index type.

Create indexes

When the `create_index` method is called, Milvus synchronously indexes the existing data on vector field. Whereas Milvus stores massive data in segments, it creates an index file for each data segment separately when indexing.

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

Selecting an Index Best Suited for Your Scenario

Most of the vector index types supported by Milvus use approximate nearest neighbors search (ANNS). Compared with accurate retrieval, which is usually very time-consuming, the core idea of ANNS is no longer limited to returning the most accurate result, but only searching for neighbors of the target. ANNS improves retrieval efficiency by sacrificing accuracy within an acceptable range.

To learn how to choose an appropriate metric for an index, see Similarity Metrics.

According to the implementation methods, the ANNS vector index can be divided into four categories:

- Tree-based index
- Graph-based index
- Hash-based index
- Quantization-based index

The following table classifies the indexes that Milvus supports:

Supported index

Classification

Scenario

FLAT

N/A

Relatively small dataset

Requires a 100% recall rate

IVF_FLAT

Quantization-based index

High-speed query

Requires a recall rate as high as possible

IVF_SQ8

Quantization-based index

High-speed query

Limited memory resources

Accepts minor compromise in recall rate

IVF_PQ

Quantization-based index

Very high-speed query

Limited memory resources

Accepts substantial compromise in recall rate

HNSW

Graph-based index

High-speed query

Requires a recall rate as high as possible

Large memory resources

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

Tree-based index

Low-dimensional vectors

Supported vector indexes

FLAT

For vector similarity search applications that require perfect accuracy and depend on relatively small (million-scale) datasets, the FLAT index is a good choice. FLAT does not compress vectors, and is the only index that can guarantee exact search results. Results from FLAT can also be used as a point of comparison for results produced by other indexes that have less than 100% recall.

FLAT is accurate because it takes an exhaustive approach to search, which means for each query the target input is compared to every vector in a dataset. This makes FLAT the slowest index on our list, and poorly suited for querying massive vector data. There are no parameters for the FLAT index in Milvus, and using it does not require data training or additional storage.

- Search parameters

Parameter	Description	Range
`metric_type`	[Optional] The chosen distance metric.	See Supported Metrics.

IVF_FLAT

IVF_FLAT divides vector data into `nlist` cluster units, and then compares distances between the target input vector and the center of each cluster. Depending on the number of clusters the system is set to query (`nprobe`), similarity search results are returned based on comparisons between the target input and the vectors in the most similar cluster(s) only —drastically reducing query time.

By adjusting `nprobe`, an ideal balance between accuracy and speed can be found for a given scenario. Results from the IVF_FLAT performance test demonstrate that query time increases sharply as both the number of target input vectors (`nq`), and the number of clusters to search (`nprobe`), increase.

IVF_FLAT is the most basic IVF index, and the encoded data stored in each unit is consistent with the original data.

- Index building parameters

Parameter	Description	Range
`nlist`	Number of cluster units	[1, 65536]

- Search parameters

Parameter	Description	Range
`nprobe`	Number of units to query	CPU: [1, nlist] GPU: [1, min(2048, nlist)]

IVF_SQ8

IVF_FLAT does not perform any compression, so the index files it produces are roughly the same size as the original, raw non-indexed vector data. For example, if the original 1B SIFT dataset is 476 GB, its IVF_FLAT index files will be slightly larger (~470 GB). Loading all the index files into memory will consume 470 GB of storage.

When disk, CPU, or GPU memory resources are limited, IVF_SQ8 is a better option than IVF_FLAT. This index type can convert each FLOAT (4 bytes) to UINT8 (1 byte) by performing scalar quantization. This reduces disk, CPU, and GPU memory consumption by 70 75%. For the 1B SIFT dataset, the IVF_SQ8 index files require just 140 GB of storage.

- Index building parameters

Parameter	Description	Range
`nlist`	Number of cluster units	[1, 65536]

- Search parameters

Parameter	Description	Range
`nprobe`	Number of units to query	CPU: [1, nlist] GPU: [1, min(2048, nlist)]

IVF_PQ

`PQ` (Product Quantization) uniformly decomposes the original high-dimensional vector space into Cartesian products of `m` low-dimensional vector spaces, and then quantizes the decomposed low-dimensional vector spaces. Instead of calculating the distances between the target vector and the center of all the units, product quantization enables the calculation of distances between the target vector and the clustering center of each low-dimensional space and greatly reduces the time complexity and space complexity of the algorithm.

IVF_PQ performs IVF index clustering before quantizing the product of vectors. Its index file is even smaller than IVF_SQ8, but it also causes a loss of accuracy during searching vectors.

Index building parameters and search parameters vary with Milvus distribution. Select your Milvus distribution first.

- Index building parameters

Parameter	Description	Range
`nlist`	Number of cluster units	[1, 65536]
`m`	Number of factors of product quantization	`dim` $\equiv 0 \pmod{\texttt{m}}$
`nbits`	[Optional] Number of bits in which each low-dimensional vector is stored.	[1, 16] (8 by default)

- Search parameters

Parameter	Description	Range
`nprobe`	Number of units to query	[1, nlist]

HNSW

HNSW (Hierarchical Navigable Small World Graph) is a graph-based indexing algorithm. It builds a multi-layer navigation structure for an image according to certain rules. In this structure, the upper layers are more sparse and the distances between nodes are farther; the lower layers are denser and the distances between nodes are closer. The search starts from the uppermost layer, finds the node closest to the target in this layer, and then enters the next layer to begin another search. After multiple iterations, it can quickly approach the target position.

In order to improve performance, HNSW limits the maximum degree of nodes on each layer of the graph to `M`. In addition, you can use `efConstruction` (when building index) or `ef` (when searching targets) to specify a search range.

- 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]

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) is an index that uses a hyperplane to divide a high-dimensional space into multiple subspaces, and then stores them in a tree structure.

When searching for vectors, ANNOY follows the tree structure to find subspaces closer to the target vector, and then compares all the vectors in these subspaces (The number of vectors being compared should not be less than `search_k`) to obtain the final result. Obviously, when the target vector is close to the edge of a certain subspace, sometimes it is necessary to greatly increase the number of searched subspaces to obtain a high recall rate. Therefore, ANNOY uses `n_trees` different methods to divide the whole space, and searches all the dividing methods simultaneously to reduce the probability that the target vector is always at the edge of the subspace.

- Index building parameters

Parameter	Description	Range
`n_trees`	The number of methods of space division.	[1, 1024]

- Search parameters

Parameter	Description	Range
`search_k`	The number of nodes to search. -1 means 5% of the whole data.	$\{-1\} \cup [$`top_k`, n $\times$ `n_trees`$]$

FAQ

What is the difference between FLAT index and IVF_FLAT index?

IVF_FLAT index divides a vector space into nlist clusters. If you keep the default value of nlist as 16384, Milvus compares the distances between the target vector and the centers of all 16384 clusters to get nprobe nearest clusters. Then Milvus compares the distances between the target vector and the vectors in the selected clusters to get the nearest vectors. Unlike IVF_FLAT, FLAT directly compares the distances between the target vector and each and every vector.

Therefore, when the total number of vectors approximately equals nlist, IVF_FLAT and FLAT has little difference in the way of calculation required and search performance. But as the number of vectors grows to two times, three times, or n times of nlist, IVF_FLAT index begins to show increasingly greater advantages.

See How to Choose an Index in Milvus for more information.

Bibliography

- 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

Properties

```
    <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

A collection schema is the logical definition of a collection. Usually you need to define the field schema before defining a collection schema and creating a collection.

Collection schema properties

Properties

```
    <th>Description</th>
    <th>Note</th>
</tr>
</thead>
<tbody>
<tr>
    <td>field</td>
    <td>Fields in the collection to create</td>
```

```
    <td>Mandatory</td>
</tr>
<tr>
    <td>description</td>
    <td>Description of the collection</td>
    <td>Data type: String.<br/>Optional</td>
</tr>
<tr>
    <td>auto_id</td>
    <td>Whether to enable Automatic ID (primary key) allocation or not</td>
    <td>Data type: Boolean (<code>true</code> or <code>false</code>).<br/>Optional</td>
</tr>
</tbody>
```

Create a collection schema

Define the field schemas before defining a 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
```

Create a collection with the schema specified:

```
from pymilvus import Collection
collection_name1 = "tutorial_1"
collection1 = Collection(name=collection_name1, schema=schema, using='default', shards_num=2)
```

You can define the shard number with shards_num and in which Milvus server you wish to create a collection by specifying the alias in using.

You can also create a collection with Collection.construct_from_dataframe, which automatically generates a collection schema from DataFrame and creates a 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
                                )
```

Boolean Expression Rules

Overview

A predicate expression outputs a boolean value. Milvus conducts scalar filtering by searching with predicates. A predicate expression, when evaluated, returns either TRUE or FALSE. View Python SDK API Reference for instruction on using predicate expressions.

EBNF grammar rules describe boolean expressions rules:

```
Expr = LogicalExpr | NIL
```

```
LogicalExpr = LogicalExpr BinaryLogicalOp LogicalExpr
              | UnaryLogicalOp LogicalExpr
              | "(" LogicalExpr ")"
              | SingleExpr;

BinaryLogicalOp = "&&" | "and" | "||" | "or";

UnaryLogicalOp = "not";

SingleExpr = TermExpr | CompareExpr;

TermExpr = IDENTIFIER "in" ConstantArray;

Constant = INTEGER | 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 = ">" | ">=" | "<" | "<=" | "=="| "!=";
```

The following table lists the description of each symbol mentioned in the above Boolean expression rules.

Notation	Description
=	Definition.
,	Concatenation.
;	Termination.
|	Alternation.
{···}	Repetition.
(···)	Grouping.
NIL	Empty. The expression can be an empty string.
INTEGER	Integers such as 1, 2, 3.
FLOAT	Float numbers such as 1.0, 2.0.
CONST	Integers or float numbers.
IDENTIFIER	Identifier. In Milvus, the IDENTIFIER represents the field name.
LogicalOp	A LogicalOp is a logical operator that supports combining more than one relational operation in one comparison. Returned value of a LogicalOp is either TRUE (1) or FALSE (0). There are two types of LogicalOps, including BinaryLogicalOps and UnaryLogicalOps.

Notation	Description
UnaryLogicalOp	UnaryLogicalOp refers to the unary logical operator “not”.
BinaryLogicalOp	Binary logical operators that perform actions on two operands. In a complex expression with two or more operands, the order of evaluation depends on precedence rules.
ArithmeticOp	An ArithmeticOp, namely an arithmetic operator, performs mathematical operations such as addition and subtraction on operands.
UnaryArithOp	A UnaryArithOp is an arithmetic operator that performs an operation on a single operand. The negative UnaryArithOp changes a positive expression into a negative one, or the other way round.
BinaryArithOp	A BinaryArithOp, namely a binary operator, performs operations on two operands. In a complex expression with two or more operands, the order of evaluation depends on precedence rules.
CmpOp	CmpOp is a relational operator that perform actions on two operands.
CmpOpRestricted	CmpOpRestricted is restricted to “Less than” and “Equal”.
ConstantExpr	ConstantExpr can be a Constant or a BinaryArithOp on two ConstExprs or a UnaryArithOp on a single ConstantExpr. It is defined recursively.
ConstantArray	ConstantArray is wrapped by square brackets, and ConstantExpr can be repeated in the square brackets. ConstArray must include at least one ConstantExpr.
TermExpr	TermExpr is used to check whether the value of an IDENTIFIER appears in a ConstantArray. TermExpr is represented by “in”.
CompareExpr	A CompareExpr, namely comparison expression can be relational operations on two IDENTIFIERs, or relational operations on one IDENTIFIER and one ConstantExpr, or ternary operation on two ConstantExprs and one IDENTIFIER.
SingleExpr	SingleExpr, namely single expression, can be either a TermExpr or a CompareExpr.
LogicalExpr	A LogicalExpr can be a BinaryLogicalOp on two LogicalExprs, or a UnaryLogicalOp on a single LogicalExpr, or a LogicalExpr grouped within parentheses, or a SingleExpr. The LogicalExpr is defined recursively.
Expr	Expr, an abbreviation meaning expression, can be LogicalExpr or NIL.

Operators

Logical operators:

Logical operators perform a comparison between two expressions.

Symbol	Operation	Example	Description
‘and’ &&	and	expr1 && expr2	True if both expr1 and expr2 are true.
‘or’ ||	or	expr1 || expr2	True if either expr1 or expr2 are true.

Binary arithmetic operators:

Binary arithmetic operators contain two operands and can perform basic arithmetic operations and return the corresponding result.

Symbol	Operation	Example	Description
+	Addition	a + b	Add the two operands.
-	Subtraction	a - b	Subtract the second operand from the first operand.
*	Multiplication	a * b	Multiply the two operands.
/	Division	a / b	Divide the first operand by the second operand.
**	Power	a ** b	Raise the first operand to the power of the second operand.
%	Modulo	a % b	Divide the first operand by the second operand and yield the remainder portion.

Relational operators:

Relational operators use symbols to check for equality, inequality, or relative order between two expressions.

Symbol	Operation	Example	Description
<	Less than	a < b	True if a is less than b.
>	Greater than	a > b	True if a is greater than b.
==	Equal	a == b	True if a is equal to b.
!=	Not equal	a != b	True if a is not equal to b.
<=	Less than or equal	a <= b	True if a is less than or equal to b.
>=	Greater than or equal	a >= b	True if a is greater than or equal to b.

Operator precedence and associativity

The following table lists the precedence and associativity of operators. Operators are listed top to bottom, in descending precedence.

Precedence	Operator	Description	Associativity
1	+ -	UnaryArithOp	Left-to-right
2	not	UnaryLogicOp	Right-to-left
3	**	BinaryArithOp	Left-to-right
4	* / %	BinaryArithOp	Left-to-right
5	+ -	BinaryArithOp	Left-to-right
6	< <= > >=	CmpOp	Left-to-right
7	== !=	CmpOp	Left-to-right
8	&& and	BinaryLogicOp	Left-to-right
9	|| or	BinaryLogicOp	Left-to-right

Expressions are normally evaluated from left to right. Complex expressions are evaluated one at a time. The order in which the expressions are evaluated is determined by the precedence of the operators used.

If an expression contains two or more operators with the same precedence, the operator to the left is evaluated first.

For example, 10 / 2 * 5 will be evaluated as (10 / 2) and the result multiplied by 5.

When a lower precedence operation should be processed first, it should be enclosed within parentheses.

For example, 30 / 2 + 8. This is normally evaluated as 30 divided by 2 then 8 added to the result. If you want to divide by 2 + 8, it should be written as 30 / (2 + 8).

Parentheses can be nested within expressions. Innermost parenthetical expressions are evaluated first.

Time Travel

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.

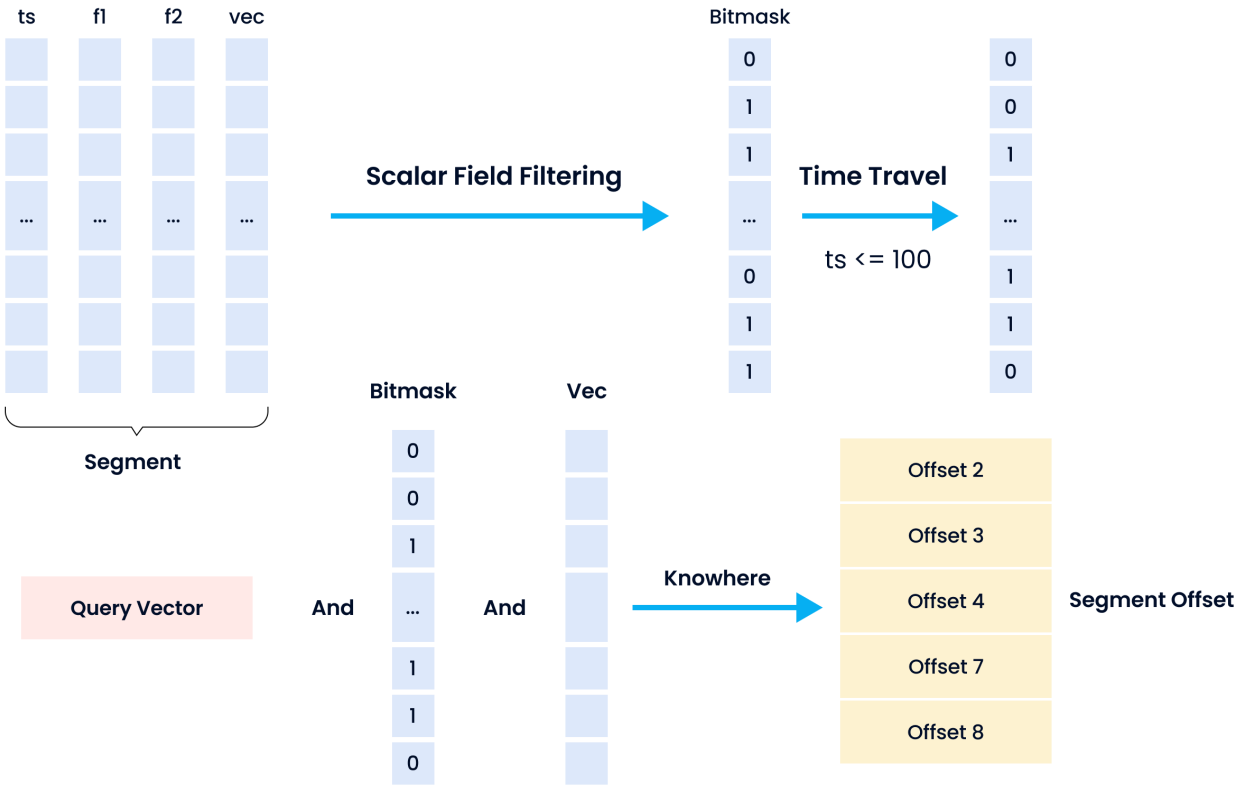


Figure 18: Time_travel

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.

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.

Glossary

Collection

A collection in Milvus is equivalent to a table in a relational database management system (RDBMS). In Milvus, collections are used to store and manage entities.

Dependency

A dependency is a program that another program relies on to work. Milvus' dependencies include etcd (stores meta data), MinIO or S3 (object storage), and Pulsar (manages snapshot logs).

Entity

An entity consists of a group of fields that represent real world objects. Each entity in Milvus is represented by a unique primary key.

You can customize primary keys. If you do not configure manually, Milvus automatically assigns primary keys to entities. If you choose to configure your own customized primary keys, note that Milvus does not support primary key de-duplication for now. Therefore, there can be duplicate primary keys in the same collection.

Field

Fields are the units that make up entities. Fields can be structured data (e.g., numbers, strings) or vectors.

Scalar field filtering is now available in Milvus 2.0!

Log broker

The log broker is a publish-subscribe system that supports playback. It is responsible for streaming data persistence, execution of reliable asynchronous queries, event notification, and return of query results. It also ensures integrity of the incremental data when the worker nodes recover from system breakdown.

Log sequence

The log sequence records all operations that change collection states in Milvus.

Log subscriber

Log subscribers subscribe to the log sequence to update the local data and provides services in the form of read-only copies.

Milvus cluster

In a cluster deployment of Milvus, services are provided by a group of nodes to achieve high availability and easy scalability.

Milvus standalone

In a standalone deployment of Milvus, all operations including data insertion, index building, and vector similarity search are completed in one single process.

Normalization

Normalization refers to the process of converting an embedding (vector) so that its norm equals one. If inner product (IP) is used to calculate embeddings similarities, all embeddings must be normalized. After normalization, inner product equals cosine similarity.

Partition

A partition is a division of a collection. Milvus supports dividing collection data into multiple parts on physical storage. This process is called partitioning, and each partition can contain multiple segments.

PChannel

PChannel stands for physical channel. Each PChannel corresponds to a topic for log storage. A group of 64 PChannels by default will be assigned to store logs that record data insertion, deletion, and update when the Milvus cluster is started.

Schema

Schema is the meta information that defines data type and data property. Each collection has its own collection schema that defines all the fields of a collection, automatic ID (primary key) allocation enablement, and collection description. Also included in collection schemas are field schemas that defines the name, data type, and other properties of a field.

Segment

A segment is a data file automatically created by Milvus for holding inserted data. A collection can have multiple segments and a segment can have multiple entities. During vector similarity search, Milvus scans each segment and returns the search results. A segment can be either growing or sealed. A growing segment keeps receiving the newly inserted data till it is sealed. A sealed segment no longer receives any new data, and will be flushed to the object storage, leaving new data to be inserted into a freshly created growing segment. A growing segment will be sealed either because the number of entities it holds reaches the pre-defined threshold, or because the span of "growing" status exceeds the specified limit.

Sharding

Sharding refers to distributing write operations to different nodes to make the most of the parallel computing potential of a Milvus cluster for writing data. By default, a single collection contains two shards. Milvus adopts a sharding method based on primary key hashing. Milvus' development roadmap includes supporting more flexible sharding methods such as random and custom sharding.

Partitioning works to reduce read load by specifying a partition name, while sharding spreads write load among multiple servers.

Unstructured data

Unstructured data, including images, video, audio, and natural language, is information that doesn't follow a predefined model or manner of organization. This data type accounts for ~80% of the world's data, and can be converted into vectors using various artificial intelligence (AI) and machine learning (ML) models.

VChannel

VChannel stands for logical channel. Each collection will be assigned a group of VChannels for recording data insertion, deletion, and update. VChannels are logically separated but physically share resources.

Embedding Vector

An embedding vector is a feature abstraction of unstructured data, such as emails, IoT sensor data, Instagram photos, protein structures, and much more. Mathematically speaking, an embedding vector is an array of floating-point numbers or binaries. Modern embedding techniques are used to convert unstructured data to embedding vectors.

Vector index

A vector index is a reorganized data structure derived from raw data that can greatly accelerate the process of vector similarity search. Milvus supports several vector index types.

Vector similarity search

Vector similarity search is the process of comparing a vector to a database to find vectors that are most similar to the target search vector. Approximate nearest neighbor (ANN) search algorithms are used to calculate similarity between vectors.

Example Applications

Image Similarity Search

This tutorial demonstrates how to use Milvus, the open-source vector database, to build a reverse image search system. - Open Jupyter notebook - Quick deploy - Try online demo The ML models and third-party software used include: - YOLOv3 - ResNet-50 - MySQL

Major search engines like Google already give users the option to search by image. Additionally, e-commerce platforms have realized the benefits this functionality offers online shoppers, with Amazon incorporating image search into its smartphone applications.

In this tutorial, you will learn how to build a reverse image search system that can detect image patterns and return similar images to the one you upload. To build such an image similarity search system, download PASCAL VOC image set which contains 17125 images of 20 categories. Alternatively, you can prepare your own image datasets. Use YOLOv3 for object detection and ResNet-50 for image feature extraction. After going through the two ML models, images are converted into 256-dimensional vectors. Then store the vectors in Milvus and a unique ID for each vector is automatically generated by Milvus. MySQL is then used to map the vector IDs to the images in the dataset. Whenever you upload a new image to the image search system, it will be converted into a new vector and compared against the vectors previously stored in Milvus. Milvus then returns the IDs of the most similar vectors and you can query the corresponding images in MySQL.

Question Answering System

This tutorial demonstrates how to use Milvus, the open-source vector database, to build a question answering (QA) system. - Open Jupyter notebook - Quick deploy - Try online demo

The ML model and third-party software used include: - BERT - MySQL

Question answering system is a common real world application that belongs to the field of natural language processing. Typical QA systems include online customer service systems, QA chatbots, and more. Most question answering systems can be classified as: generative or retrieval, single-round or multi-round, open-domain or specific question answering systems.

In this tutorial, you will learn how to build a QA system that can link new user questions to massive answers previously stored in the vector database. To build such a chatbot, prepare your own dataset of questions and corresponding answers. Store the questions and answers in MySQL, a relational database. Then use BERT, the machine learning (ML) model for natural language processing (NLP) to convert questions into vectors. These question vectors are stored and indexed in Milvus. When users input a new question, it is converted into a vector by the BERT model as well, and Milvus searches for the most similar question vector to this new vector. The QA system returns the corresponding answer to the most similar questions.

Recommender System

This tutorial demonstrates how to use Milvus, the open-source vector database, to build a recommendation system. - Open Jupyter notebook - Quick deploy

The ML Model and third-party software used include: - PaddlePaddle - Redis or MySQL

The recommender system is a subset of the information filtering system, which can be used in various scenarios including personalized movie, music, product, and feed stream recommendation. Unlike search engines, recommender systems do not require users to accurately describe their needs but discover users' needs and interests by analyzing user behaviors.

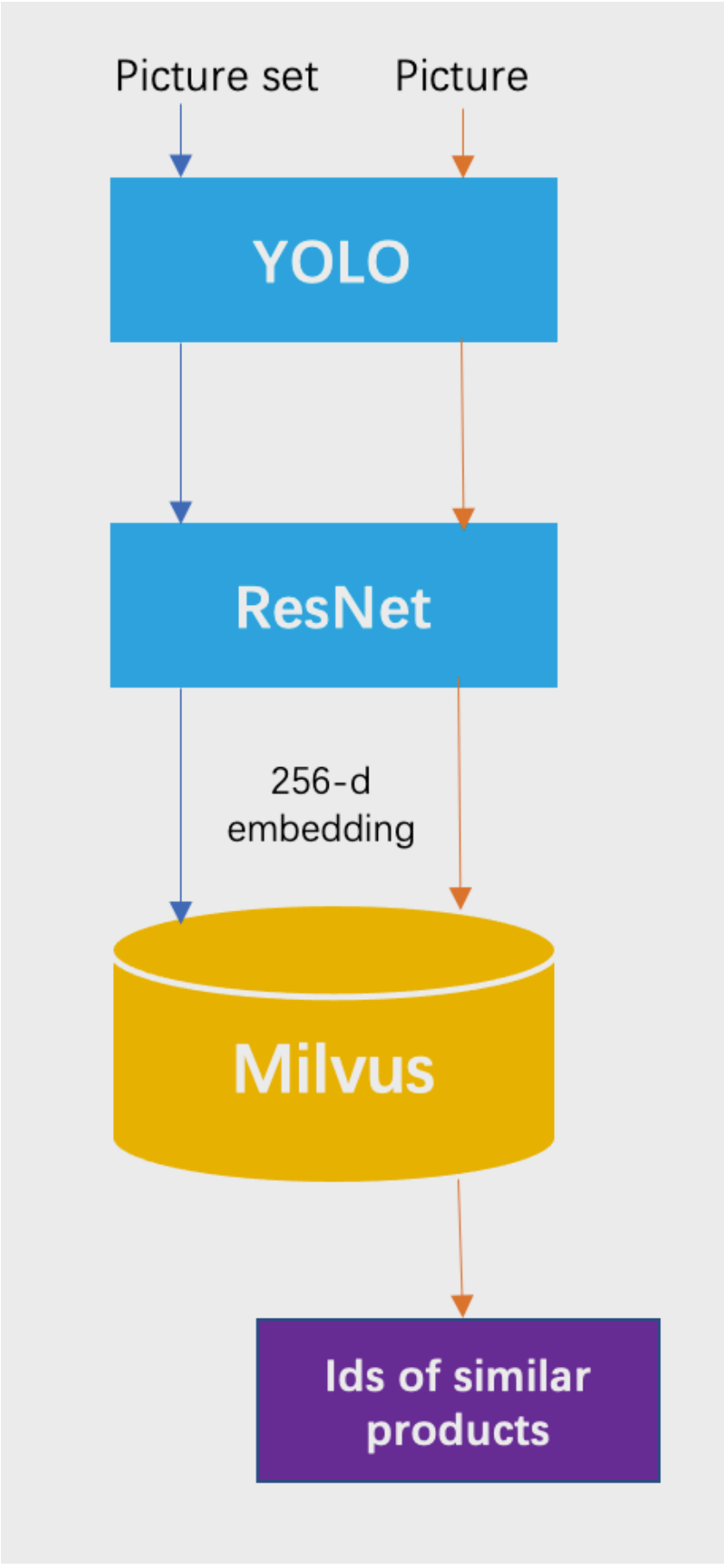


Figure 19: image_search

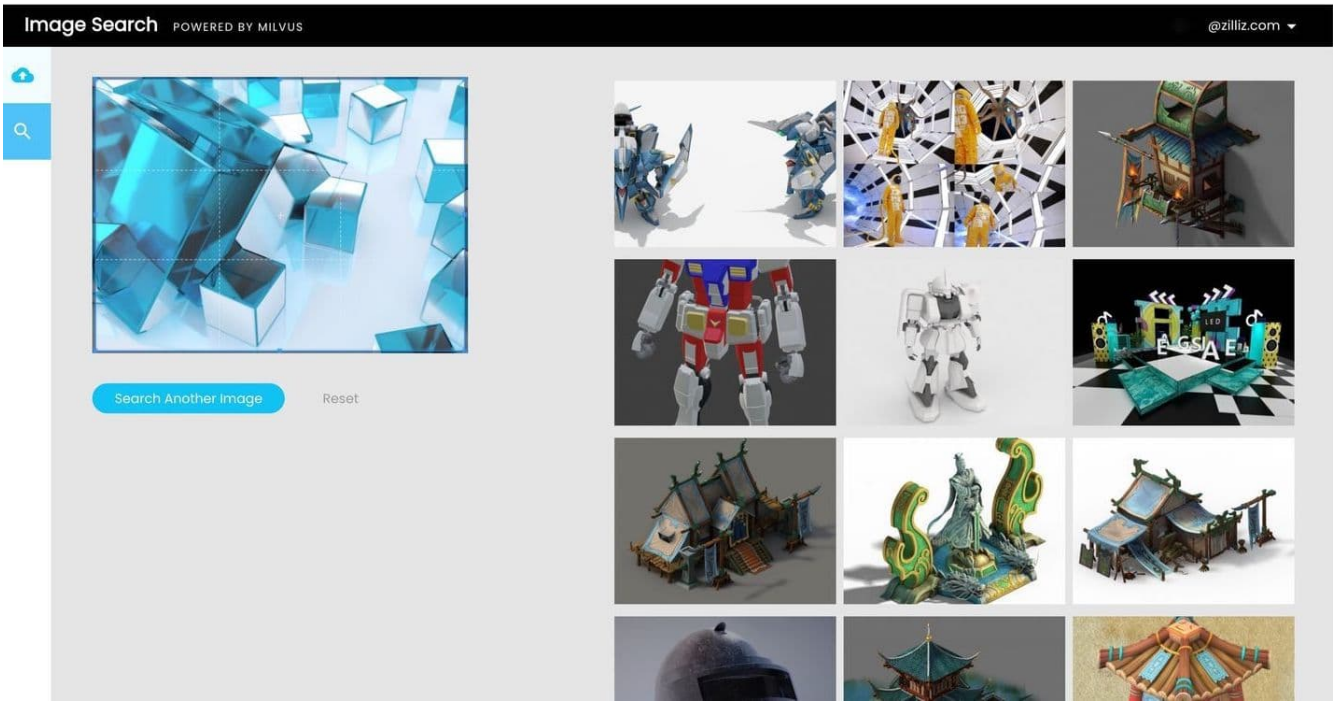


Figure 20: image_search_demo

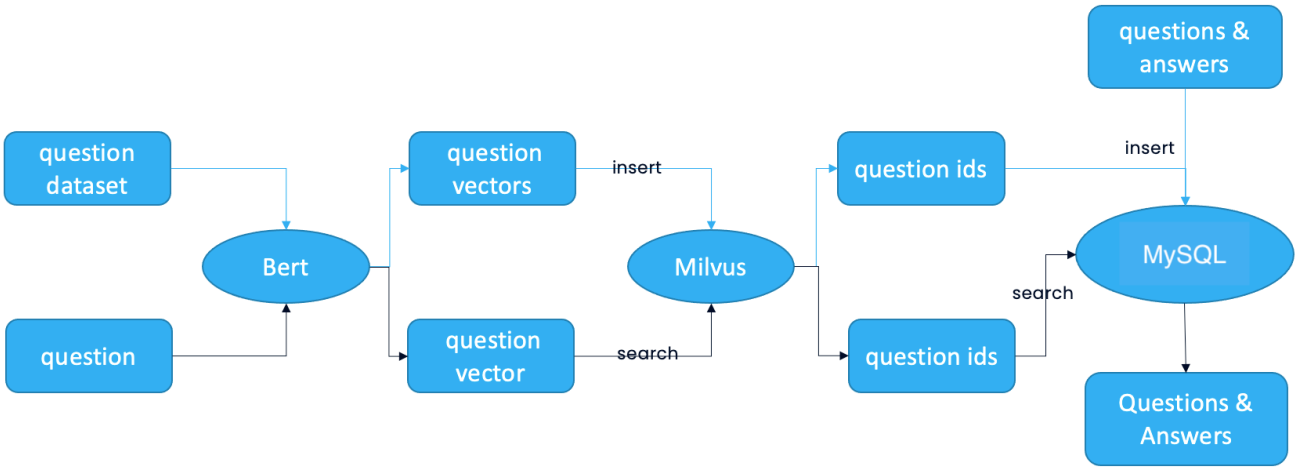


Figure 21: QA_Chatbot

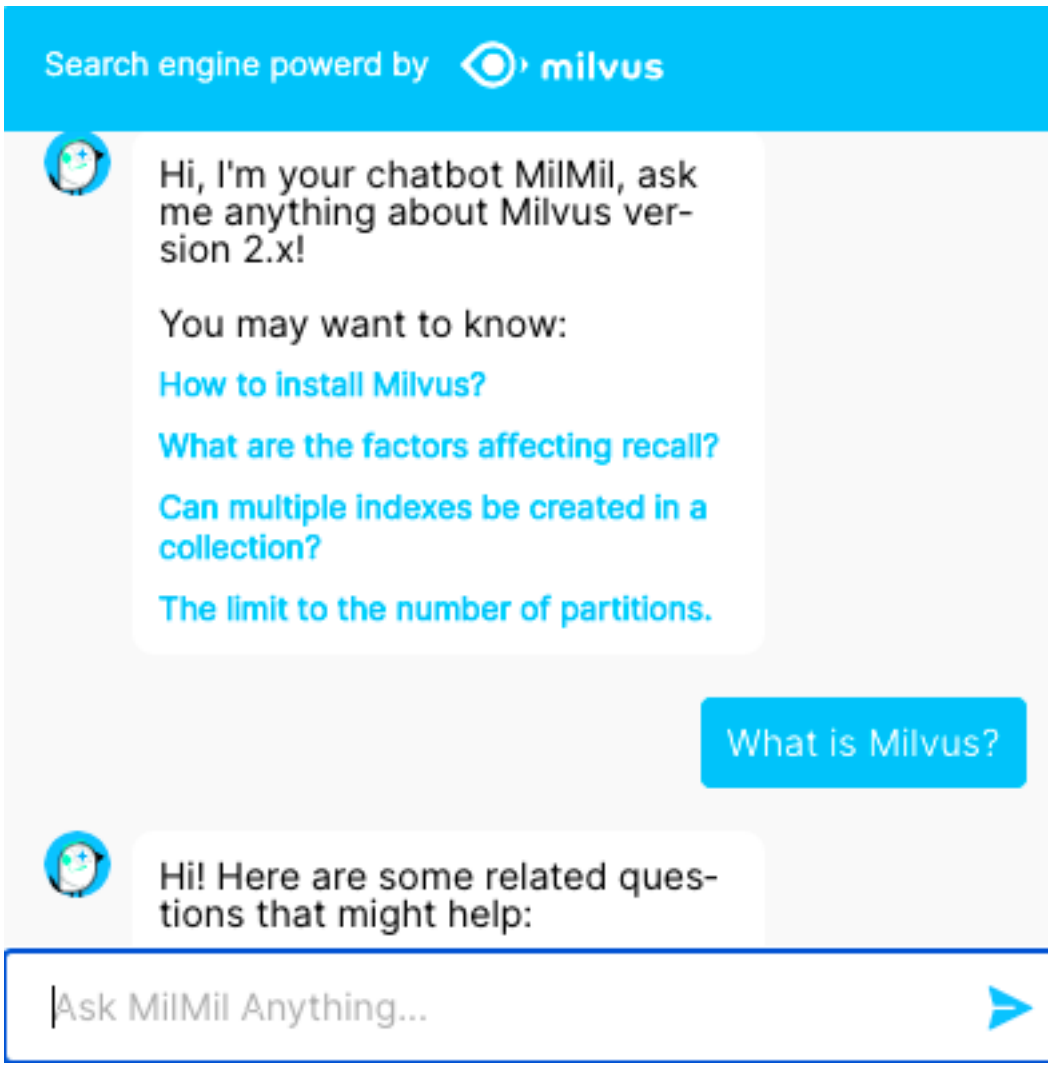


Figure 22: QA_chatbot_demo

In this tutorial, you will learn how to build a movie recommender system that can suggest movies meeting user interests. To build such a recommender system, first download a movie-related dataset. This tutorial uses MovieLens 1M. Alternatively, you can prepare your own datasets, which should include such information as users' ratings of movies, users' demographic characteristics, and movie description. Use PaddlePaddle to combine user IDs and features and convert them into 256-dimensional vectors. Convert movie IDs and features into vectors in a similar way. Store the movie vectors in Milvus and use user vectors for similarity search. If the user vector is similar to a movie vector, Milvus will return the movie vector and its ID as the recommendation result. Then query movie information using the movie vector ID stored in Redis or MySQL.

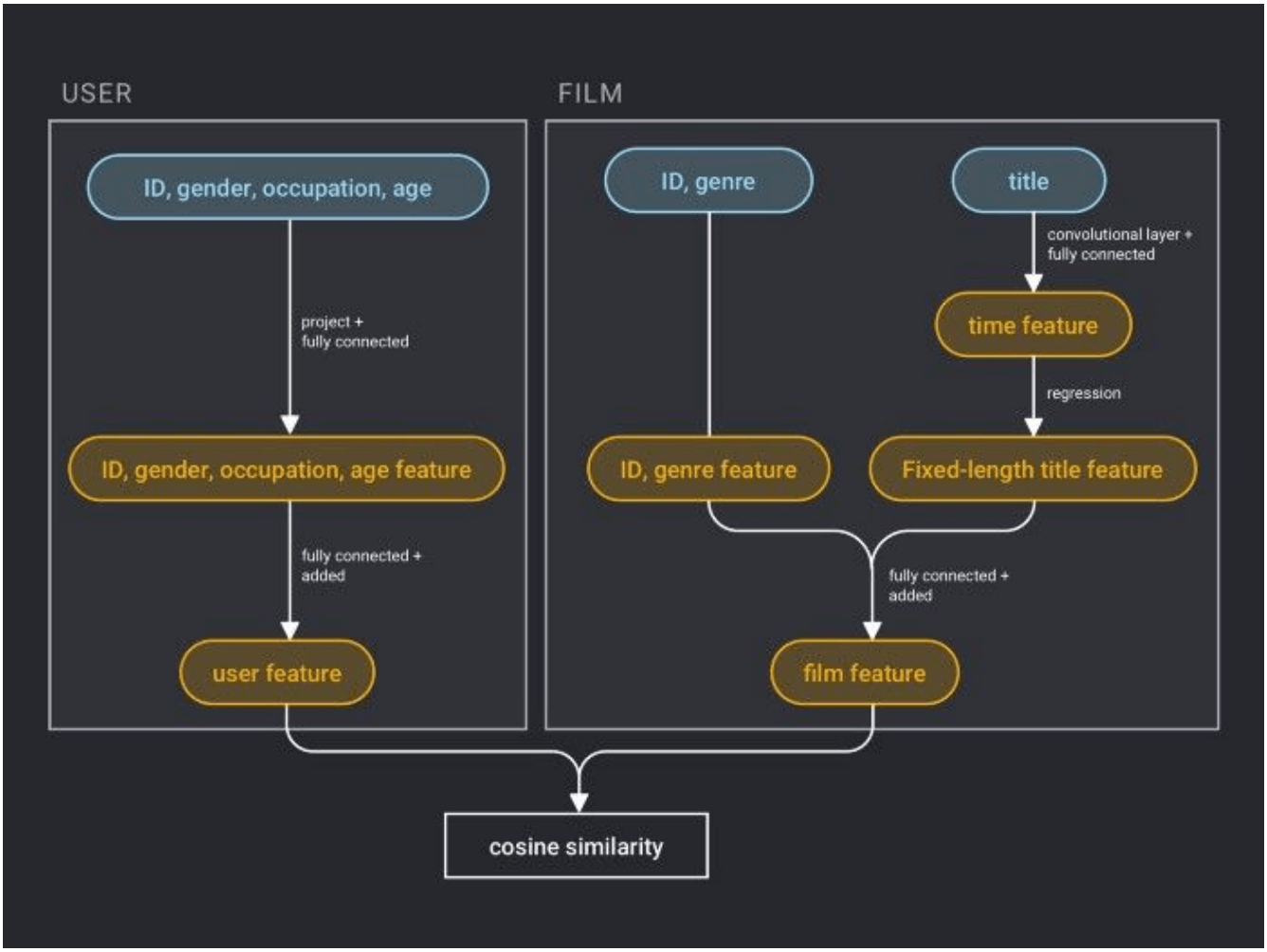


Figure 23: recommender_system

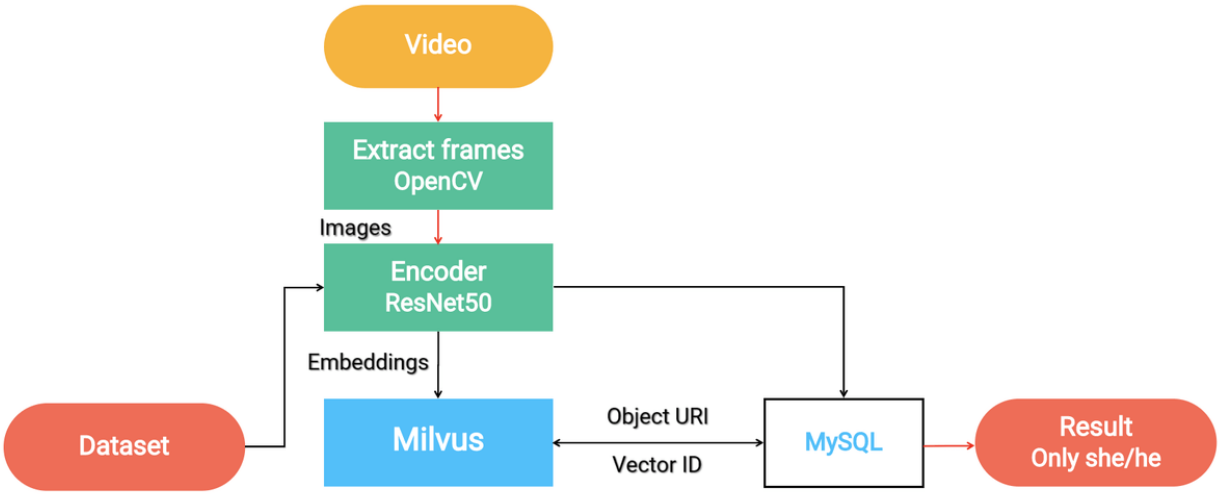
Video Similarity Search

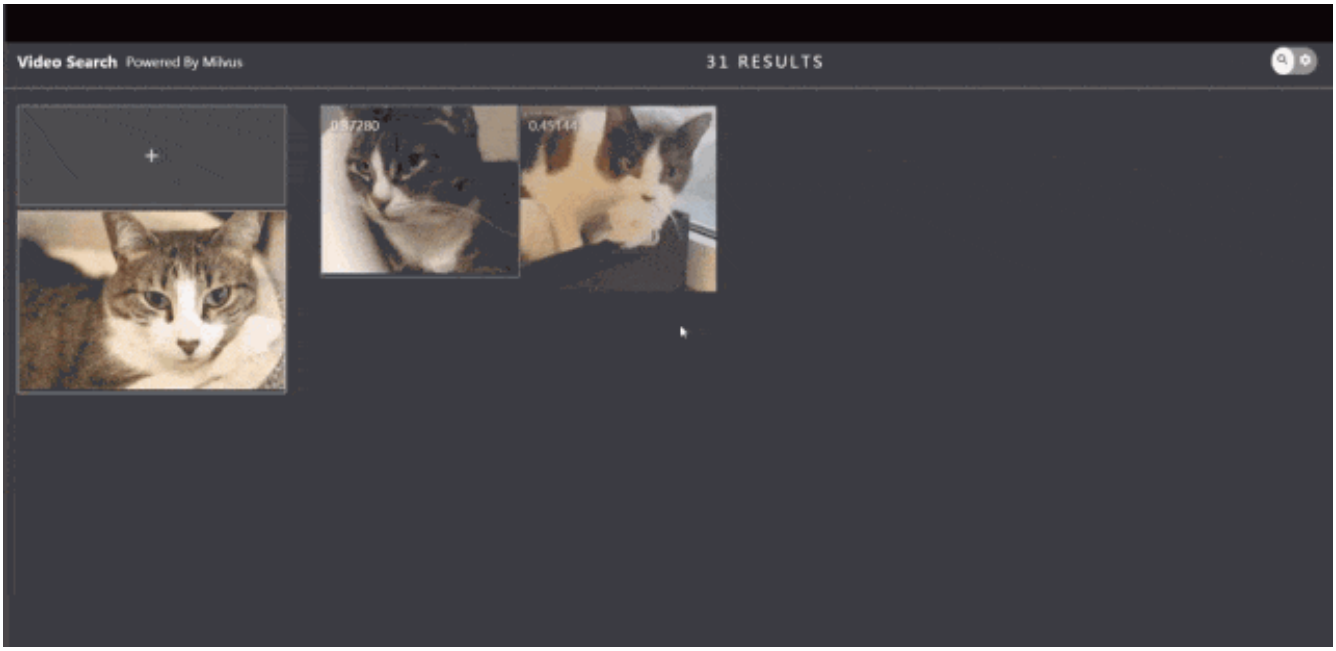
This tutorial demonstrates how to use Milvus, the open-source vector database, to build a video similarity search system. - Open Jupyter notebook - Quick deploy The ML models and third-party software used include: - OpenCV - ResNet-50 - MySQL

Nowadays, after watching a movie or video they like, people can easily take screenshots and share their thoughts by posting on various social networking platforms. When the followers see the screenshots, it can be really difficult for them to tell which movie it is if the movie name is not spelled out explicitly in the post. In order to figure out the name of the movie, people can take advantage of a video similarity search system. By using the system, users can upload an image and get videos or movies that contain key frames similar to the uploaded image.

In this tutorial, you will learn how to build a video similarity search system. This tutorial uses approximately 100 animated gifs on Tumblr to build the system. However, you can also prepare your own video datasets. The system

first uses OpenCV to extract key frames in videos and then obtains feature vectors of each key frame using ResNet-50. All vectors are stored and searched in Milvus, which will return the IDs of similar vectors. Then map the IDs to the corresponding video stored in MySQL.





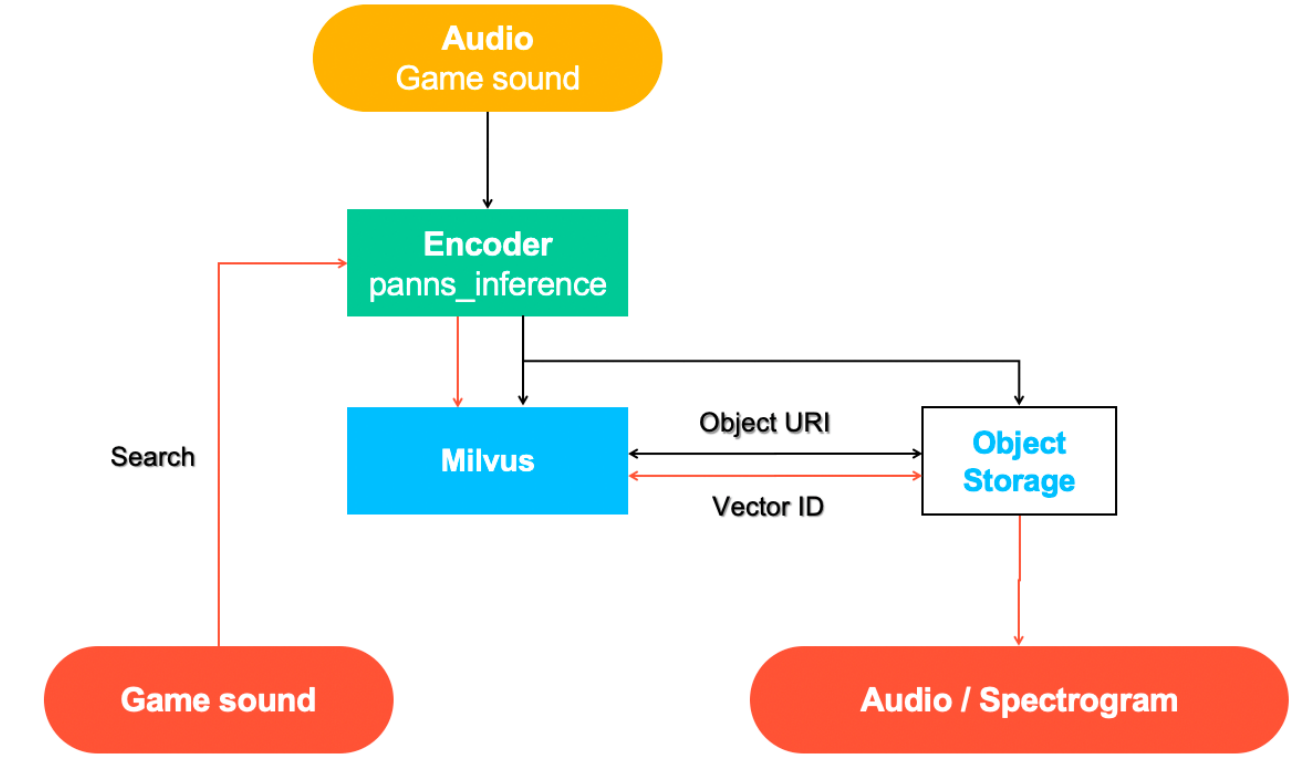
Audio Similarity Search

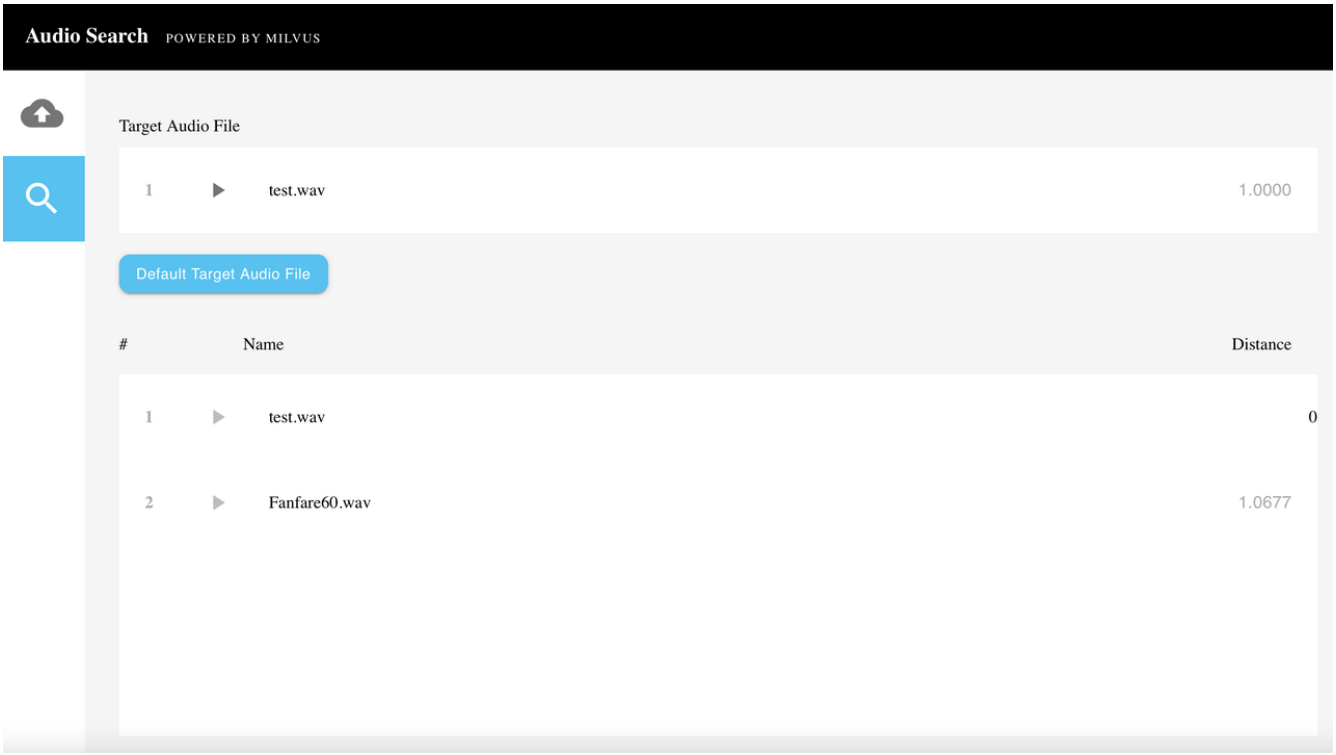
This tutorial demonstrates how to use Milvus, the open-source vector database to build an audio similarity search system. - Open Jupyter notebook - Quick deploy The ML model and third-party software used include: - PANNs (Large-Scale Pretrained Audio Neural Networks) - MySQL

Speech, music, sound effects, and other types of audio search makes it possible to quickly query massive volumes of audio data and surface similar sounds. Applications of audio similarity search systems include identifying similar sound effects, minimizing IP infringement, and more. Audio retrieval can be used to search and monitor online media in real-time to crack down on infringement of intellectual property rights. It also assumes an important role in the classification and statistical analysis of audio data.

In this tutorial, you will learn how to build an audio similarity search system that can return similar sound clips. The uploaded audio clips are converted into vectors using PANNs. These vectors are stored in Milvus which automatically generates a unique ID for each vector. Then users can conduct a vector similarity search in Milvus and

query the audio clip data path corresponding to th unique vector ID returned by Milvus.





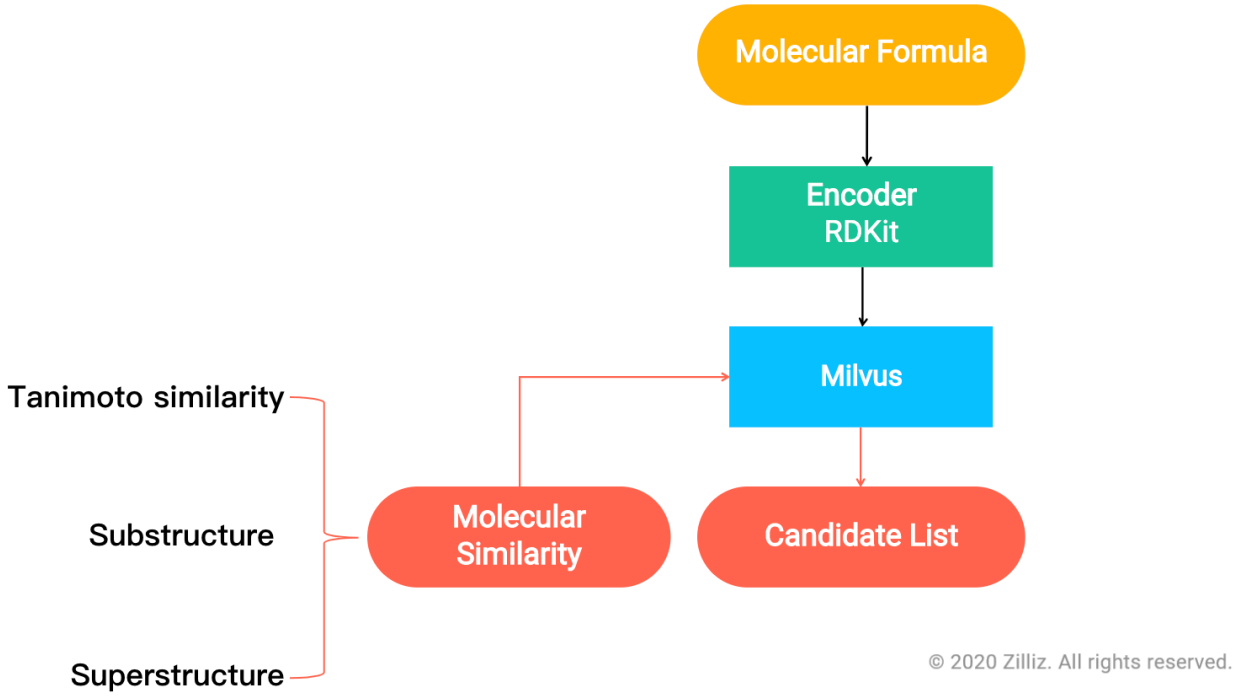
Molecular Similarity Search

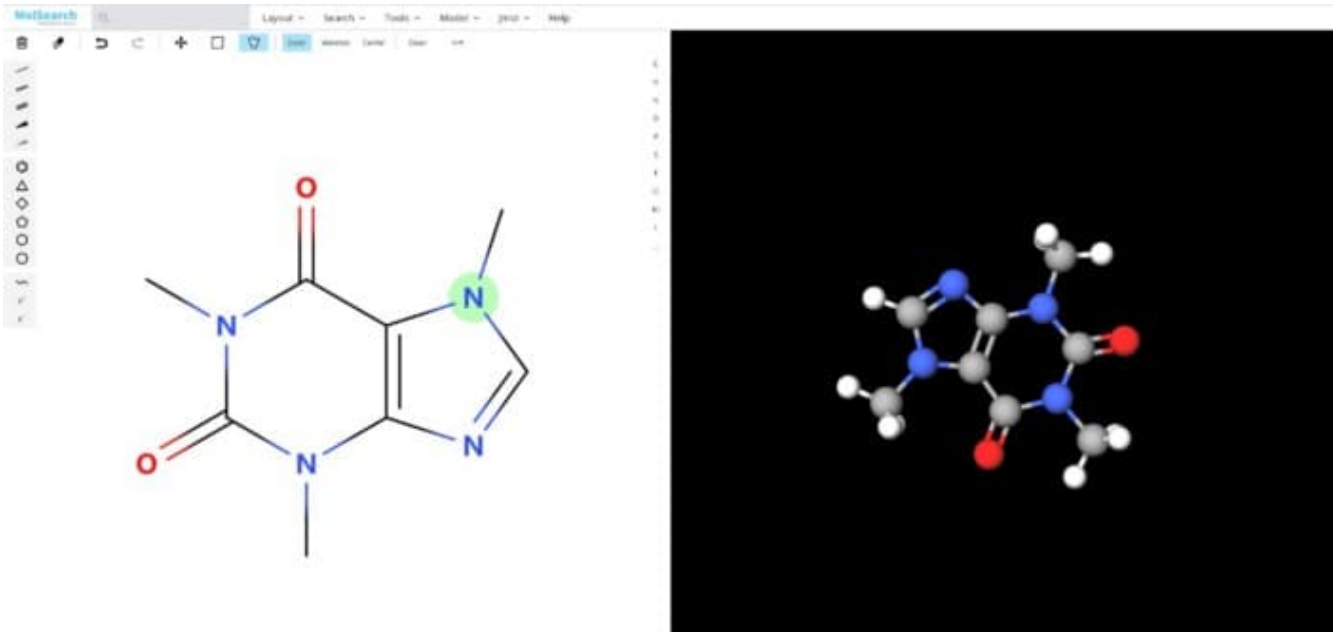
This tutorial demonstrates how to use Milvus, the open-source vector database, to build a molecular similarity search system. - Open Jupyter notebook - Quick deploy - Try demo The third-party software used include: - RDKit

- MySQL

Drug discovery is an important part of new medicine research and development. The process of drug discovery includes target selection and confirmation. When fragments or lead compounds are discovered, researchers usually search for similar compounds in internal or commercial compound libraries in order to discover structure-activity relationship (SAR), compound availability. Ultimately, they will evaluate the potential of the lead compounds to be optimized to candidate compounds. In order to discover available compounds from billion-scale compound libraries, chemical fingerprint is usually retrieved for substructure search and molecule similarity search.

In this tutorial, you will learn how to build a molecular similarity search system that can retrieve the substructure, superstructure, and similar structure of a particular molecule. RDKit is an open-source cheminformatics software that can convert molecule structures into vectors. Then, the vectors are stored in Milvus and Milvus can perform similarity search on vectors. Milvus also automatically generates a unique ID for each vector. The mapping of vector IDs and structure of molecules are stored in MySQL.





DNA Sequence Classification

This tutorial demonstrates how to use Milvus, the open-source vector database, to build a DNA sequence classification model. - Open Jupyter notebook - Quick deploy The ML model and third-party software used include: - CountVectorizer - MySQL

DNA sequence is a popular concept in gene traceability, species identification, disease diagnosis, and many more areas. Whereas all industries starve for a more intelligent and efficient research method, artificial intelligence has attracted much attention especially from biological and medical domains. More and more scientists and researchers are contributing to machine learning and deep learning in the field of bioinformatics. To make experimental results more convincing, one common option is to increase sample size. The collaboration with big data in genomics brings more possibilities of application in reality. However, the traditional sequence alignment has limitations, making it unsuitable for large datasets. In order to make less trade-off in reality, vectorization is a good choice for a large dataset of DNA sequences.

In this tutorial, you will learn how to build a DNA sequence classification model. This tutorial uses CountVectorizer to extract features of DNA sequences and convert them into vectors. Then, these vectors are stored in Milvus and their corresponding DNA classes are stored in MySQL. Users can conduct a vector similarity search in Milvus and recall the corresponding DNA classification from MySQL.

Text Search Engine

In this tutorial, you will learn how to use Milvus, the open-source vector database, to build a text search engine. - Open Jupyter notebook - Quick deploy The ML model and third-party software used include: - BERT - MySQL

One major application of Milvus in the field of natural language processing (NLP) is text search engine. It is a great tool that can help users find the information they are looking for. It can even surface information that is hard to find. Text search engines compare the keywords or semantics users input against a database of texts, and then return the results that meet certain criteria.

In this tutorial, you will learn how to build a text search engine. This tutorial uses BERT to convert texts into fixed-length vectors. Milvus is used as a vector database for storage and vector similarity search. Then use MySQL to map the vector IDs generated by Milvus to the text data.

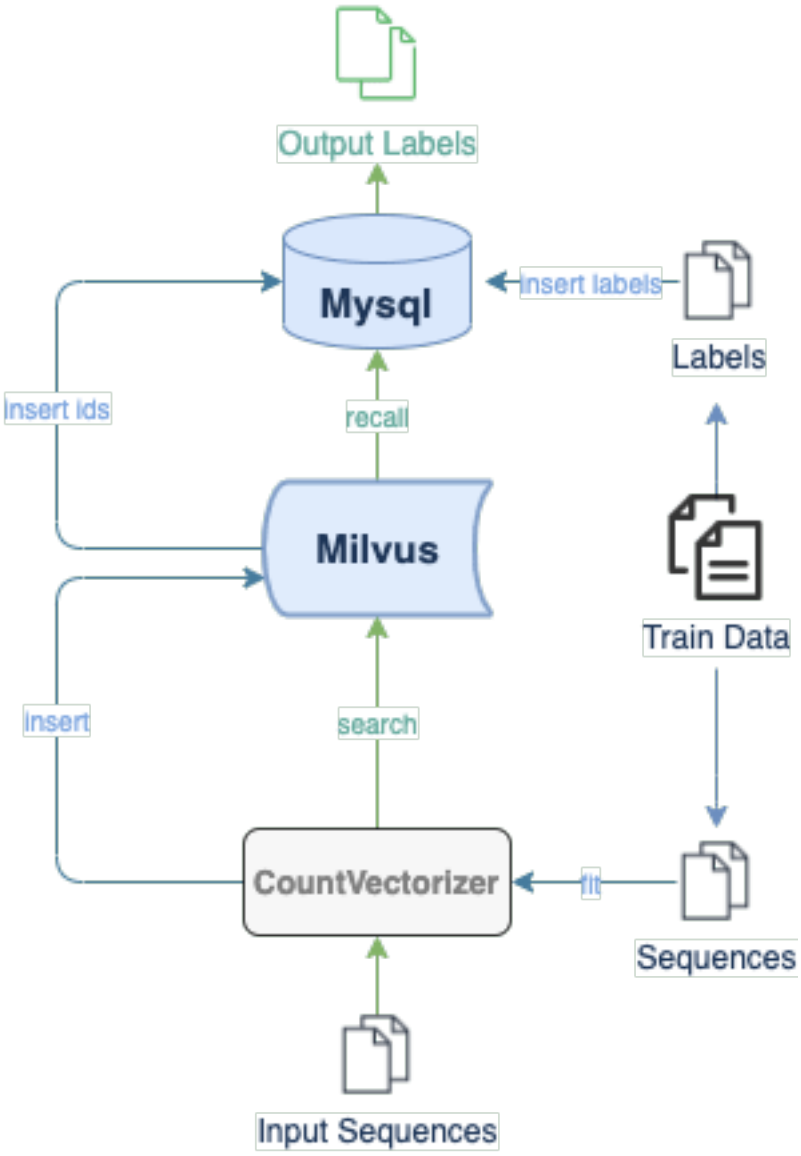
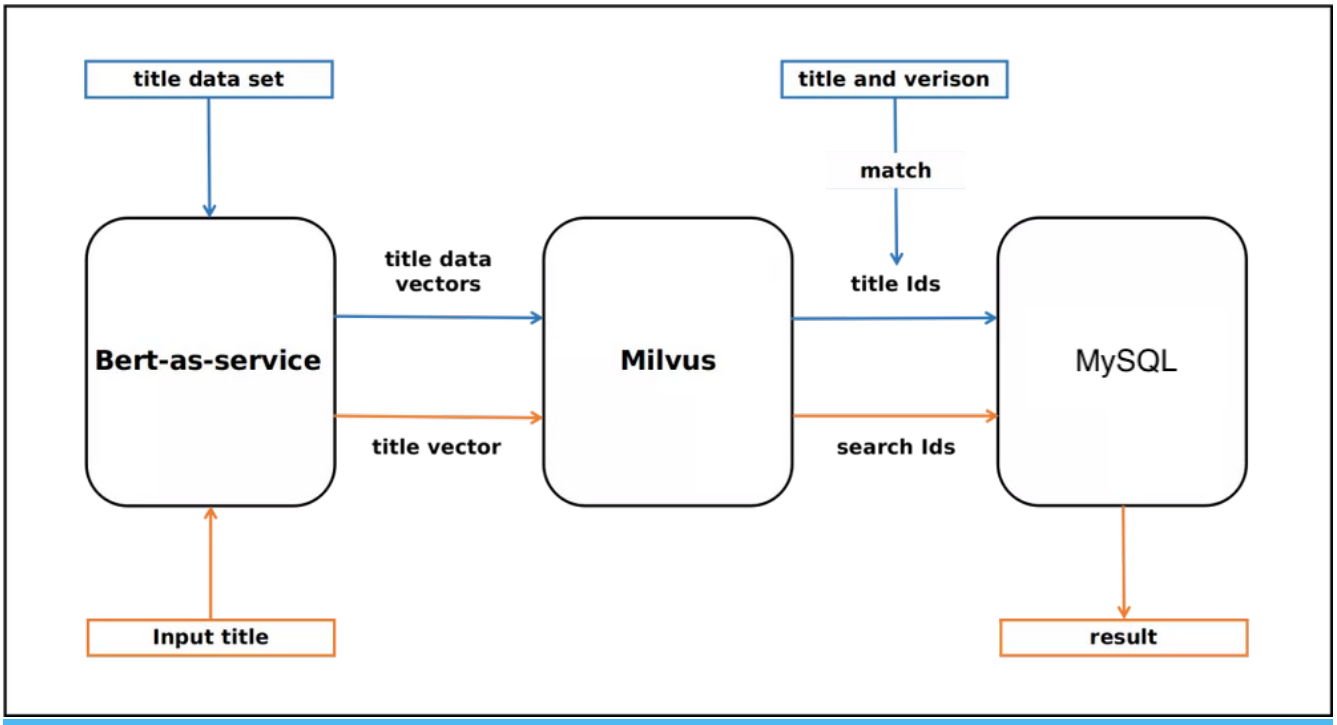


Figure 24: dna



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

Performance FAQs

Performance FAQ

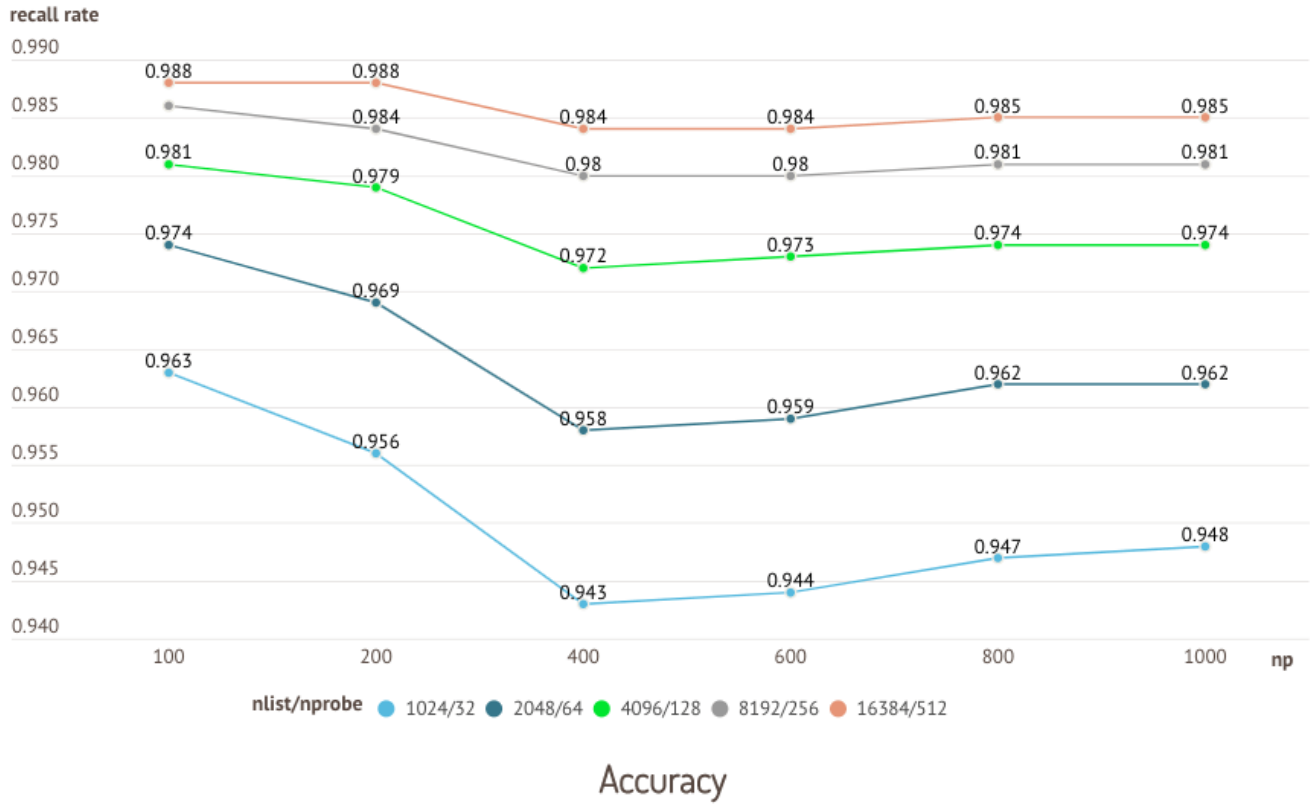
How to set `nlist` and `nprobe` for IVF indexes?

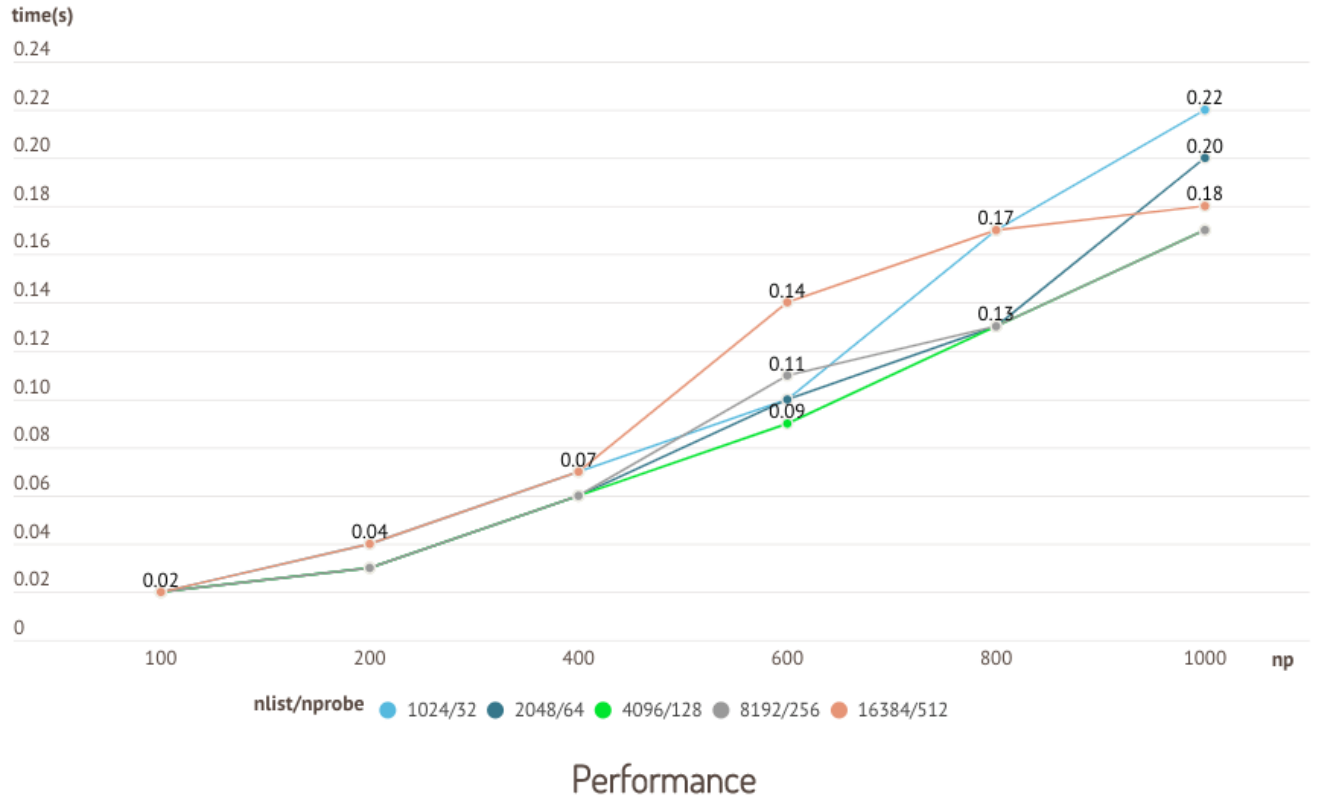
Setting `nlist` is scenario-specific. As a rule of thumb, the recommended value of `nlist` is `4 × sqrt(n)`, where `n` is the total number of entities in a segment.

The size of each segment is determined by the `datacoord.segment.maxSize` parameter, which is set to 512 MB by default. The total number of entities in a segment n can be estimated by dividing `datacoord.segment.maxSize` by the size of each entity.

Setting `nprobe` is specific to the dataset and scenario, and involves a trade-off between accuracy and query performance. We recommend finding the ideal value through repeated experimentation.

The following charts are results from a test running on the sift50m dataset and IVF_SQ8 index, which compares recall and query performance of different `nlist`/`nprobe` pairs.





Performance

Why do queries sometimes take longer on smaller datasets?

Query operations are conducted on segments. indexes reduce the amount of time it takes to query a segment. If a segment has not been indexed, Milvus resorts to brute-force search on the raw data—drastically increasing query time.

Therefore, it usually takes longer to query on a small dataset (collection) because it has not built index. This is because the sizes of its segments have not reached the index-building threshold set by `rootCoord.minSegmentSizeToEnableindex`. Call `create_index()` to force Milvus to index segments that have reached the threshold but not yet been automatically indexed, significantly improving query performance.

What factors impact CPU usage?

CPU usage increases when Milvus is building indexes or running queries. In general, index building is CPU intensive except when using Annoy, which runs on a single thread.

When running queries, CPU usage is affected by `nq` and `nprobe`. When `nq` and `nprobe` are small, concurrency is low and CPU usage stays low.

Does simultaneously inserting data and searching impact query performance?

Insert operations are not CPU intensive. However, because new segments may not have reached the threshold for index building, Milvus resorts to brute-force search—significantly impacting query performance.

The `rootcoord.minSegmentSizeToEnableIndex` parameter determines the index-building threshold for a segment, and is set to 1024 rows by default. See System Configuration for more information.

Still have questions?

You can:

- Check out Milvus on GitHub. Feel free to ask questions, share ideas, and help others.
- Join our Slack Channel to find support and engage with our open-source community.

Product FAQs

Product FAQ

How much does Milvus cost?

Milvus is a 100% free open-source project.

Please adhere to Apache License 2.0 when using Milvus for production or distribution purposes.

Zilliz, the company behind Milvus, also offers a fully managed cloud version of the platform for those that don't want to build and maintain their own distributed instance. Zilliz Cloud automatically maintains data reliability and allows users to pay only for what they use.

Does Milvus support non-x86 architectures?

Milvus cannot be installed or run on non-x86 platforms.

Your CPU must support one of the following instruction sets to run Milvus: SSE4.2, AVX, AVX2, AVX512. These are all x86-dedicated SIMD instruction sets.

What is the maximum dataset size Milvus can handle?

Theoretically, the maximum dataset size Milvus can handle is determined by the hardware it is run on, specifically system memory and storage:

- Milvus loads all specified collections and partitions into memory before running queries. Therefore, memory size determines the maximum amount of data Milvus can query.
- When new entities and and collection-related schema (currently only MinIO is supported for data persistence) are added to Milvus, system storage determines the maximum allowable size of inserted data.

Where does Milvus store data?

Milvus deals with two types of data, inserted data and metadata.

Inserted data, including vector data, scalar data, and collection-specific schema, is stored in persistent storage (for now MinIO only) as incremental log.

Metadata is generated within Milvus. Each Milvus module has its own metadata that is stored in etcd.

Why is there no vector data in etcd?

etcd stores Milvus module metadata; MinIO stores entities.

Does Milvus' Python SDK have a connection pool?

Python SDKs for Milvus v0.9.0 or higher have a connection pool. The number of connections in a connection pool has no upper limit.

Does Milvus support inserting and searching data simultaneously?

Yes. Insert operations and query operations are handled by two separate modules that are mutually independent. From the client's perspective, an insert operation is complete when the inserted data enters the message queue. However, inserted data is unsearchable until it is loaded to the query node. If the segment size does not reach the index-building threshold (512 MB by default), Milvus resorts to brute-force search and query performance may be diminished.

Can vectors with duplicate primary keys be inserted into Milvus?

Yes. Milvus does not check if vector primary keys are duplicates.

When vectors with duplicate primary keys are inserted, does Milvus treat it as an update operation?

No. Milvus does not currently support update operations and does not check if entity primary keys are duplicates. You are responsible for ensuring entity primary keys are unique, and if they aren't Milvus may contain multiple entities with duplicate primary keys.

If this occurs, which data copy will return when queried remains an unknown behavior. This limitation will be fixed in future releases.

What is the maximum length of self-defined entity primary keys?

Entity primary keys must be non-negative 64-bit integers.

What is the maximum amount of data that can be added per insert operation?

An insert operation must not exceed 1,024 MB in size. This is a limit imposed by gRPC.

Does collection size impact query performance when searching in a specific partition?

No. If partitions for a search are specified, Milvus searches the specified partitions only.

Does Milvus load the entire collection when partitions are specified for a search?

No. Milvus v2.0 has varied behavior. Data must be loaded to memory before searching.

- If you know which partitions your data is located in, call `load_partition()` to load the intended partition(s) then specify partition(s) in the `search()` method call.
- If you do not know the exact partitions, call `load_collection()` before calling `search()`.
- If you fail to load collections or partitions before searching, Milvus returns an error.

Can indexes be created after inserting vectors?

Yes. If `create_index()` is called, Milvus builds an index for subsequently inserted vectors. However, Milvus does not build an index until the newly inserted vectors fill an entire segment and the newly created index file is separate from the previous one.

How are the FLAT and IVF_FLAT indexes different?

The IVF_FLAT index divides vector space into list clusters. At the default list value of 16,384, Milvus compares the distances between the target vector and the centroids of all 16,384 clusters to return probe nearest clusters. Milvus then compares the distances between the target vector and the vectors in the selected clusters to get the nearest vectors. Unlike IVF_FLAT, FLAT directly compares the distances between the target vector and every other vector.

When the total number of vectors approximately equals nlist, there is little distance between IVF_FLAT and FLAT in terms of calculation requirements and search performance. However, as the number of vectors exceeds nlist by a factor of two or more, IVF_FLAT begins to demonstrate performance advantages.

See Vector Index for more information.

How does Milvus flush data?

Milvus returns success when inserted data is loaded to the message queue. However, the data is not yet flushed to the disk. Then Milvus' data node writes the data in the message queue to persistent storage as incremental logs. If `flush()` is called, the data node is forced to write all data in the message queue to persistent storage immediately.

What is normalization? Why is normalization needed?

Normalization refers to the process of converting a vector so that its norm equals 1. If inner product is used to calculate vector similarity, vectors must be normalized. After normalization, inner product equals cosine similarity.

See Wikipedia for more information.

Why do Euclidean distance (L2) and inner product (IP) return different results?

For normalized vectors, Euclidean distance (L2) is mathematically equivalent to inner product (IP). If these similarity metrics return different results, check to see if your vectors are normalized

Is there a limit to the total number of collections and partitions in Milvus?

There is no limit on the number of collections. However, the number of partitions in each collection must not exceed the value set by the parameter `master.maxPartitionNum`.

Why do I get fewer than k vectors when searching for `topk` vectors?

Among the indexes that Milvus supports, IVF_FLAT and IVF_SQ8 implement the k-means clustering method. A data space is divided into `nlist` clusters and the inserted vectors are distributed to these clusters. Milvus then selects the `nprobe` nearest clusters and compares the distances between the target vector and all vectors in the selected clusters to return the final results.

If `nlist` and `topk` are large and nprobe is small, the number of vectors in the nprobe clusters may be less than `k`. Therefore, when you search for the `topk` nearest vectors, the number of returned vectors is less than `k`.

To avoid this, try setting `nprobe` larger and `nlist` and `k` smaller.

See Vector Index for more information.

What is the maximum vector dimension supported in Milvus?

Milvus can manage vectors with up to 32,768 dimensions.

Does Milvus support Apple M1 CPU?

Current Milvus release does not support Apple M1 CPU.

What data types does Milvus support on the primary key field?

In current release, Milvus only support INT64 on primary key field. Both INT64 and string will be supported in the formal release of Milvus 2.0.0.

Is Milvus scalable?

Yes. You can deploy Milvus cluster with multiple nodes via Helm Chart on Kubernetes. Refer to Scale Guide for more instruction.

Does the query perform in memory? What are incremental data and historical data?

Yes. When a query request comes, Milvus searches both incremental data and historical data by loading them into memory. Incremental data are data in the growing segments, which are buffered in memory before they reach the threshold to be persisted in storage engine, while historical data are from the sealed segments that are stored in the object storage. Incremental data and historical data together constitute the whole dataset to search.

Is Milvus 2.0 available for concurrent search?

Yes. For queries on the same collection, Milvus concurrently searches the incremental and historical data. However, queries on different collections are conducted in series. Whereas the historical data can be an extremely huge dataset, searches on the historical data are relatively more time-consuming and essentially performed in series. The formal release of Milvus 2.0 will improve this issue.

Why does the data in MinIO remain after the corresponding collection is dropped?

Data in MinIO is designed to remain for a certain period of time for the convenience of data rollback.

Does Milvus support message engines other than Pulsar?

Future release of Milvus 2.0 will support Kafka.

What's the diference between a search and a query?

In Milvus, a vector similarity search retrieves vectors based on similarity calculation and vector index acceleration. Unlike a vector similarity search, a vector query retrieves vectors via scalar filtering based on boolean expression. The boolean expression filters on scalar fields or the primary key field, and it retrieves all results that match the filters. In a query, neither similarity metrics nor vector index is involved.

Still have questions?

You can:

- Check out Milvus on GitHub. You're welcome to raise questions, share ideas, and help others.
- Join our Slack community to find support and engage with our open-source community.

Operational FAQs

Operational FAQ

What if I failed to pull the Milvus Docker image from Docker Hub?

If you failed to pull the Milvus Docker image from Docker Hub, try adding other registry mirrors.

Users from Mainland China can add the URL "https://registry.docker-cn.com" to the registry-mirrors array in /etc.docker/daemon.json.

```
{
  "registry-mirrors": ["https://registry.docker-cn.com"]
}
```

Is Docker the only way to install and run Milvus?

Docker is an efficient way to deploy Milvus, but not the only way. You can also deploy Milvus from source code. This requires Ubuntu (18.04 or higher) or CentOS (7 or higher). See Building Milvus from Source Code for more information.

What are the main factors affecting recall?

Recall is affected mainly by index type and search parameters.

For FLAT index, Milvus takes an exhaustive scan within a collection, with a 100% return.

For IVF indexes, the nprobe parameter determines the scope of a search within the collection. Increasing nprobe increases the proportion of vectors searched and recall, but diminishes query performance.

For HNSW index, the ef parameter determines the breadth of the graph search. Increasing ef increases the number of points searched on the graph and recall, but diminishes query performance.

For more information, see Vector Indexing.

Why did my changes to the configuration files not take effect?

Milvus v2.0 does not support modification to configuration files during runtime. You must restart Milvus Docker for configuration file changes to take effect.

How do I know if Milvus has started successfully?

If Milvus is started using Docker Compose, run `docker ps` to observe how many Docker containers are running and check if Milvus services started correctly.

- For Milvus standalone, you should be able to observe at least three running Docker containers, one being the Milvus service and the other two being etcd management and storage service. For more information, see Installing Milvus Standalone.
- For Milvus Cluster, you should be able to observe at least twelve running Docker containers, nine for the Milvus service and three for basic services. For more information, see Installing Milvus Cluster.

Why is the time in the log files different from the system time?

The time difference is usually due to the fact that the host machine does not use Coordinated Universal Time (UTC).

The log files inside the Docker image use UTC by default. If your host machine does not use UTC, this issue may occur.

How do I know if my CPU supports Milvus?

Milvus' computing operations depend on CPU's support for SIMD (Single Instruction, Multiple Data) extension instruction set. Whether your CPU supports SIMD extension instruction set is crucial to index building and vector similarity search within Milvus. Ensure that your CPU supports at least one of the following SIMD instruction sets:

- SSE4.2
- AVX
- AVX2
- AVX512

Run the lscpu command to check if your CPU supports the SIMD instruction sets mentioned above:

```
$ lscpu | grep -e sse4_2 -e avx -e avx2 -e avx512
```

Why does Milvus return `illegal instruction` during startup?

Milvus requires your CPU to support a SIMD instruction set: SSE4.2, AVX, AVX2, or AVX512. CPU must support at least one of these to ensure that Milvus operates normally. An `illegal instruction` error returned during startup suggests that your CPU does not support any of the above four instruction sets.

See CPU's support for SIMD Instruction Set.

Can I install Milvus on Windows?

Yes. You can install Milvus on Windows either by compiling from source code or from a binary package.

See Run Milvus 2.0 on Windows to learn how to install Milvus on Windows.

I got an error when installing pymilvus on Windows. What shall I do?

It is not recommended to install PyMilvus on Windows. But if you have to install PyMilvus on Windows but got an error, try installing it in a Conda environment. See Install Milvus SDK for more information about how to install PyMilvus in the Conda environment.

Can I deploy Milvus when disconnected from the Internet?

Yes. You can install Milvus in an offline environment. See Install Milvus Offline for more information.

Where can I find the logs generated by Milvus?

The Milvus log is printed to stout (standard output) and stderr (standard error) by default, however we highly recommend redirecting your log to a persistent volume in production. To do so, update `log.file.rootPath` in milvus.yaml. And if you deploy Milvus with `milvus-helm` chart, you also need to enable log persistence first via `--set log.persistence.enabled=true`.

If you didn't change the config, using kubectl logs or docker logs CONTAINER can also help you to find the log.

Can I create index for a segment before inserting data into it?

Yes, you can. But we recommend inserting data in batches, each of which should not exceed 256 MB, before indexing each segment.

Still have questions?

You can:

- Check out Milvus on GitHub. Feel free to ask questions, share ideas, and help others.
- Join our Milvus Forum or Slack Channel to find support and engage with our open-source community.

Troubleshooting

This page lists common issues that may occur when running Milvus, as well as possible troubleshooting tips. Issues on this page fall into the following categories:

- Boot issues
- Runtime issues
- API issues

Boot issues

Boot errors are usually fatal. Run the following command to view error details:

```
$ docker logs <your milvus container id>
```

Runtime issues

Errors that occur during runtime may cause service breakdown. To troubleshoot this issue, check compatibility between the server and your client before moving forward.

API issues

These issues occur during API method calls between the Milvus server and your client. They will be returned to the client synchronously or asynchronously.

If you need help solving a problem, feel free to:

- Join our Slack channel and reach out for support from the Milvus team.
- File an Issue on GitHub that includes details about your problem.