An introduction to Milvus - the world's most popular open-source vector database.

@Milvus.io

Vector Database 101 - Introduction to Milvus

Introduction

Hey there - welcome back to Milvus tutorials. In the previous tutorial, we took a quick tour of vector databases and listed the features an ideal vector database should implement. We then compared vector databases to vector search libraries[1] and vector search plugins[2]. Through example code, we found that neither vector search libraries nor vector search plugins fulfill all of the features required to store, index, and search across large datasets of unstructured data. This prompted us to go over some of the technical challenges vector database developers face.

Milvus history

Milvus development began in 2018 at *Zilliz*, making it the world's first open-source vector database. Milvus' initial conception was infrastructure which could be used to build and scale search applications; as such, Milvus was initially intended to be a Google/Bing for unstructured data. Although vector indices and search strategies were prevalent at that time, vector databases were still a relatively unknown concept. During this process, the Milvus community discovered that Milvus had the potential to be significantly more than what we originally intended it to be.

As we developed Milvus, we refined the concept as a combination of search, storage, and indexing, making it a full-fledged managed database. In November of 2019, we open-sourced Milvus under the Apache 2.0 license and released to the general public as the first widely available vector database solution. Milvus during its v0.6 release was mostly single-instance and supported only a handful of indexes.

In March 2020, Milvus joined the LF AI & Data Foundation, a nonprofit organization under the broader Linux Foundation umbrella. With help from the LF AI & Data Foundation, the Milvus community has been able to reposition itself as a *database for the AI era*, while also helping engage with the broader open-source community. While with LF AI & Data, Milvus continues to receive constant updates from both Zilliz as well as the broader open-source community. More about LF AI & Data here.

That same year, our first [full-length academic paper]https://www.cs.purdue.edu/homes/csjgwang/pubs/SIGMOD21_Mi
was accepted into ACM SIGMOD 2021 - one of the world's premier database conferences. We also

[1] These libraries include ANNOY, FAISS, ScaNN, DiskANN, and others.

[2] We defined the term "vector search plugin" as an enhancement or add-on for traditional databases and/or search systems meant to store and index structured data.

began working on a fully distributed, cloud-native version of Milvus, aptly named *Milvus 2.0*. We'll dive deeper into Milvus 2.0 in the next section.

Milvus is one of the first and foremost vector databases most developers think of when discussing vector databases. A big reason for this is Milvus' rich history - a continuously evolving history that we hope you will one day be a part of!

Milvus 2.x

We'll continue the topic of Milvus history by discussing Milvus 1.0 (1.x) and 2.0 (2.x) along with the major differences between them.

Milvus 1.0 was released in March 2021 as the first major Milvus release. This version was built atop the previous releases, and supported a number of similarity metrics (Euclidean/L2 distance, Hamming distance, Jaccard similarity, etc) along with multiple ANN indexes (FAISS, HNSW, ANNOY, and standard inverted indicies). Horizontal scaling was accomplished through a feature called *Mishards*, and storage was achieved through either local storage or NFS. Milvus 1.0 also supported fast indexing and querying through general purpose computing processors used by machine learning engineers, including NVidia GPUs and Xilinx FPGAs.

The first version of Milvus 2.0 was released in June 2021, the same month that Milvus became a graduate of the LF AI & Data Foundation. Unlike Milvus 1.x, Milvus 2.x's architecture is a *fully cloud-native* architecture, scalable to hundreds of individual nodes with a target availability/uptime of 99.9%. Also in contrast with Milvus 1.x, Milvus 2.x incorporates multiple *levels of data consistency*, enabling maximum flexibility when it comes to application development. Milvus 2.x also supports a number of other advanced features, such as multi-cloud integration, an administrative console (via Zilliz's Attu), and a feature called *time travel* (no, Milvus is not a time machine). These accomplishments along with the community's impact in the field of machine learning and vector databases was recognized in VLDB 2022, another of the field's top-tier academic conferences. You can read our VLDB 2022 paper ***LINK COMING SOON***.

If you're looking for a vector database to use for your application, we strongly recommend Milvus 2.x. Milvus 2.x is a completely novel vector database system/solution (when compared with Milvus 1.x) - it is highly available, scalable, and distributed. Due to these architectural advancements, Milvus 1.x has been officially deprecated.

Touring Milvus's architecture

Now that we've covered Milvus' history as well as how Milvus 2.x differs from Milvus 1.x, let's take some time to go over Milvus 2.x's architecture. Just as a quick refresher, here are the key features a vector database should implement[3]: - Scalability and tunability - Multi-tenancy and data isolation - A complete suite of APIs - An intuitive user interface/administrative console

As we move through each of the individual components within Milvus 2.0 (see the diagram below), we'll see how these design choices are enable Milvus to implement each of these key features.

We're going to get fairly technical here, so fasten your seat belts and let's dive right in.

Access layer

As the name suggests, the access layer is responsible for communication with the outside world, helping implement the "a complete suite of APIs" feature. When Milvus receives a request, it first gets forwarded to the access layer, where a proxy within the layer is tapped to process client connections and carry out static verification + dynamic checks before forwarding the request to the appropriate service. Once the downstream service completes execution, it returns relevant data back to the access layer; the access layer then forwards the returned content back to the end user.

Within the access layer, proxies are essentially stateless containers which provide a unified front to the outside world through load-balanced components (Nginx, Kubernetes Ingress, NodePort, and LVS). Milvus uses a massive parallel processing (MPP) architecture, where proxies return results gathered from worker nodes after global aggregation and post-processing.

Coordinator service

The coordinator service acts as Milvus's central command center, and is responsible for everything from load balancing to data management. The service itself is composed of four coordinators - the root coordinator, query coordinator, data coordinator, and index coordinator. We'll briefly describe each coordinator here - if you're interested in learning more, please check out the Milvus documentatation. - The root coordinator handles data-related requests such as collection, partition, and index creation/deletion requests. The root coordinator also manages global timestamps - all requests are assigned a timestamp the moment the root coordinator receives the request. - The query coordinator administers all query nodes within Milvus. As the name suggests, query nodes are responsible for performing searches using index and delta files. - The data coordinator manages all data nodes within Milvus, maintains metadata, and triggers flush, compact, and other background data operations. More on these operations in a future tutorial. - The index coordinator maintains the index nodes and index metadata, automatically instructing each node to load raw embedding vectors and build/rebuild vector indices when necessary.

Worker nodes

Within Milvus, worker nodes are responsible for execution. Worker nodes are horizontally scalable pods which execute commands from the corresponding coordinator service[4], akin to data nodes in

[3] See our previous tutorial for a more in-depth explanation of each bullet point.

[4] With the exception of the root coordinator, each coordinator service has a corresponding set of worker nodes.

Hadoop. A query request into Milvus, for example, goes first through a proxy in the access layer before reaching the query coordinator. Given the state of the query cluster, the coordinator will then send an appropriate set of control/command signals to the query cluster in order to successfully execute the request.

The coordinators and their corresponding worker nodes together help fullfill the "scalability and tunability" plus "multi-tenancy and data isolation" features for vector databases. Scalability is obvious - as the amount of data, queries, or indexing requirements grows and shrinks, the individual worker node clusters are able to grow and shrink horizontally with system load. By segregating querying from indexing from storage, Milvus has been architected to be highly tunable and can support applications that require varying query/write speeds in addition to varying levels of consistency.

Object storage

The object storage layer helps put the "database" in Milvus, and is responsible for general data persistence. The storage layer is divided into three separate components: - Meta store: Responsible for storing snapshots of meta data such as collection schema, node status, message consumption checkpoints, etc. Milvus relies on `etcd`, a distributed key-value store, for this functionality. `etcd` also helps preform service registration and health checks. - Log broker: The log broker is a pub/sub system that supports playback and is responsible for streaming data persistence, reliable asynchronous query execution, event notifications, and returning query results. When nodes are performing downtime recovery, the log broker ensures the integrity of incremental data through a feature called a log broker playback. Milvus uses Pulsar as its log broker when running in distrubuted mode and RocksDB as its log broker when running in standalone mode. Streaming storage services such as Kafka and Pravega can also be used as log brokers. - Object storage: The object storage layer stores log snapshots, index files, and intermediate query processing results. Milvus supports both AWS S3 and Azure Blob Storage, in addition to MinIO, a lightweight, open-source object storage service. Due to the high access latency and billing per query of object storage services, Milvus will soon support memory/SSD-based cache pools and hot/cold data separation to improve performance and reduce costs.

Future roadmap (endgame/long-term vision)

Milvus has already been tested and trusted by thousands of corporations for use in production systems at scale. By Github statistics, Milvus is by and large the world's most popular open-source vector database, and arguably the world's most advanced as well. As mentioned in the previous sections, our accomplishments in the field of unstructured data processing has already been recognized in the industry's top-tier academic conferences (SIGMOD 2021 and VLDB 2022).

We're a community of technologists at heart. By paying extra attention to components that impact scalability and performance, we architected Milvus to be superior to other vector database solutions[5]. With a number of advanced features (different types of indexes, time travel, multi-cloud object storage), choosing Milvus for your next unstructured data application is a no-brainer. A fully managed solution - the last "missing piece" - is coming soon as well, courtesy of Zilliz.

Our endgame vision for Milvus is to be a *complete* database for unstructured data processing. While Milvus 2.x is already a highly scalable and flexible architecture for querying, indexing, and storage, further improvements can be made to create a complete vector database ecosystem. This includes 1) integrating unstructured data ETL, 2) extending support for Microsoft Azure and Google Cloud Platform, and 3) support for traditional metadata types such as lists and JSON objects. In the upcoming years, the broader Milvus community will continue to improve Milvus' features and functionality. We won't rest until Milvus is *the premier* platform for all things related to unstructured data.

There is no doubt that Milvus has helped revolutionize unstructured data processing, and will continue to do so in the years to come.

Wrapping up

In this tutorial, we provided a brief introduction to Milvus, Milvus' history, as well as the primary differences between Milvus 1.x and Milvus 2.x. We also took a quick tour of the architecture of Milvus 2.x and helped shine some light on how Milvus' architecture allows it to implement all of the required features of vector databases.

In the next several tutorials, we'll provide a series of *Milvus quickstarts*, designed to help you spin up and use a Milvus instance in just a couple of minutes: - In the first tutorial, we'll provide a *Milvus standalone quickstart*, designed to help you get Milvus up and running on a local x86 or ARM/M1 instance. - In the second tutorial, we'll emulate a cluster on a local machine in order to demonstrate how to boot up a *Milvus distributed cluster*. - In the third tutorial, we'll show how to deploy an *on-premise version of Milvus*.

Lastly, we at the Milvus community have provided a short video introducing Milvus in 2.5 minutes, narrated by yours truly. Grab a cup of coffee and enjoy a front-row seat for the video!

---

[5] Such as Vespa, Pinecone, Weaviate, etc.