Prerequisites

- A running Milvus instance. The following options are available:
  - Milvus Standalone: Docker, Operator, Helm,DEB/RPM, Docker Compose.
  - Milvus Cluster: Operator, Helm.
- If required, an API key for the EmbeddingModel to generate the embeddings stored by the `MilvusVectorStore`.

Dependencies

There has been a significant change in the Spring AI auto-configuration, starter modules' artifact names. Please refer to the upgrade notes for more information.

Then add the Milvus VectorStore boot starter dependency to your project:

```
<dependency>
        <groupId>org.springframework.ai</groupId>
        <artifactId>spring-ai-starter-vector-store-milvus</artifactId>
```

```
</dependency>
```

or to your Gradle `build.gradle` build file.

```
dependencies {
    implementation 'org.springframework.ai:spring-ai-starter-vector-store-milvus'
}
```

> Refer to the Dependency Management section to add the Spring AI BOM to your build file. Refer to the Artifact Repositories section to add Maven Central and/or Snapshot Repositories to your build file.

The vector store implementation can initialize the requisite schema for you, but you must opt-in by specifying the `initializeSchema` boolean in the appropriate constructor or by setting `…initialize-schema=true` in the `application.properties` file.

> this is a breaking change! In earlier versions of Spring AI, this schema initialization happened by default.

The Vector Store, also requires an `EmbeddingModel` instance to calculate embeddings for the documents. You can pick one of the available EmbeddingModel Implementations.

To connect to and configure the `MilvusVectorStore`, you need to provide access details for your instance. A simple configuration can either be provided via Spring Boot's `application.yml`

```
spring:
        ai:
                vectorstore:
                        milvus:
                                client:
                                        host: "localhost"
                                        port: 19530
                                        username: "root"
                                        password: "milvus"
                                databaseName: "default"
                                collectionName: "vector_store"
                                embeddingDimension: 1536
                                indexType: IVF_FLAT
                                metricType: COSINE
```

Check the list of configuration parameters to learn about the default values and configuration options.

Now you can Auto-wire the Milvus Vector Store in your application and use it

```
@Autowired VectorStore vectorStore;

// ...
```

```
List <Document> documents = List.of(
    new Document("Spring AI rocks!! Spring AI rocks!! Spring AI rocks!! Spring AI rocks!! Spring AI rocks!!", Map.of("meta1", "meta1")),
    new Document("The World is Big and Salvation Lurks Around the Corner"),
    new Document("You walk forward facing the past and you turn back toward the future.", Map.of("meta2", "meta2")));

// Add the documents to Milvus Vector Store
vectorStore.add(documents);

// Retrieve documents similar to a query
List<Document> results = this.vectorStore.similaritySearch(SearchRequest.builder().query("Spring").topK(5).build());
```

Manual Configuration

Instead of using the Spring Boot auto-configuration, you can manually configure the `MilvusVectorStore`. To add the following dependencies to your project:

```
<dependency>
        <groupId>org.springframework.ai</groupId>
        <artifactId>spring-ai-milvus-store</artifactId>
</dependency>
```

Refer to the Dependency Management section to add the Spring AI BOM to your build file.

To configure MilvusVectorStore in your application, you can use the following setup:

```
        @Bean
        public VectorStore vectorStore(MilvusServiceClient milvusClient, EmbeddingModel embeddingModel) {
                return MilvusVectorStore.builder(milvusClient, embeddingModel)
                                .collectionName("test_vector_store")
                                .databaseName("default")
                                .indexType(IndexType.IVF_FLAT)
                                .metricType(MetricType.COSINE)
                                .batchingStrategy(new TokenCountBatchingStrategy())
                                .initializeSchema(true)
                                .build();
        }

        @Bean
        public MilvusServiceClient milvusClient() {
                return new MilvusServiceClient(ConnectParam.newBuilder()
                        .withAuthorization("minioadmin", "minioadmin")
                        .withUri(milvusContainer.getEndpoint())
                        .build());
        }
```

Metadata filtering

You can leverage the generic, portable metadata filters with the Milvus store.

For example, you can use either the text expression language:

```
vectorStore.similaritySearch(
    SearchRequest.builder()
    .query("The World")
    .topK(TOP_K)
    .similarityThreshold(SIMILARITY_THRESHOLD)
    .filterExpression("author in ['john', 'jill'] && article_type == 'blog'").build());
```

or programmatically using the `Filter.Expression` DSL:

```
FilterExpressionBuilder b = new FilterExpressionBuilder();

vectorStore.similaritySearch(SearchRequest.builder()
    .query("The World")
    .topK(TOP_K)
    .similarityThreshold(SIMILARITY_THRESHOLD)
    .filterExpression(b.and(
        b.in("author","john", "jill"),
        b.eq("article_type", "blog")).build()).build());
```

These filter expressions are converted into the equivalent Milvus filters.

Using MilvusSearchRequest

MilvusSearchRequest extends SearchRequest, allowing you to use Milvus-specific search parameters such as native expressions and search parameter JSON.

```
MilvusSearchRequest request = MilvusSearchRequest.milvusBuilder()
    .query("sample query")
    .topK(5)
    .similarityThreshold(0.7)
    .nativeExpression("metadata[\"age\"] > 30") // Overrides
filterExpression if both are set
    .filterExpression("age <= 30") // Ignored if nativeExpression is
set
    .searchParamsJson("{\"nprobe\":128}")
    .build();
List results = vectorStore.similaritySearch(request);
```

This allows greater flexibility when using Milvus-specific search features.

Importance of `nativeExpression` and `searchParamsJson` in `MilvusSearchRequest`

These two parameters enhance Milvus search precision and ensure optimal query performance:

nativeExpression: Enables additional filtering capabilities using Milvus' native filtering expressions. Milvus Filtering

Example:

```
MilvusSearchRequest request = MilvusSearchRequest.milvusBuilder()
    .query("sample query")
    .topK(5)
    .nativeExpression("metadata['category'] == 'science'")
    .build();
```

searchParamsJson: Essential for tuning search behavior when using IVF_FLAT, Milvus' default index. Milvus Vector Index

By default, `IVF_FLAT` requires `nprobe` to be set for accurate results. If not specified, `nprobe` defaults to `1`, which can lead to poor recall or even zero search results.

Example:

```
MilvusSearchRequest request = MilvusSearchRequest.milvusBuilder()
    .query("sample query")
    .topK(5)
    .searchParamsJson("{\"nprobe\":128}")
    .build();
```

Using `nativeExpression` ensures advanced filtering, while `searchParamsJson` prevents ineffective searches caused by a low default `nprobe` value.

Milvus VectorStore properties

You can use the following properties in your Spring Boot configuration to customize the Milvus vector store.

Property	Description	Default value
spring.ai.vectorstore.milvus.database-name	The name of the Milvus database to use.	default
spring.ai.vectorstore.milvus.collection-name	Milvus collection name to store the vectors	vector_store
spring.ai.vectorstore.milvus.initialize-schema	whether to initialize Milvus' backend	false
spring.ai.vectorstore.milvus.embedding-dimension	The dimension of the vectors to be stored in the Milvus collection.	1536
spring.ai.vectorstore.milvus.index-type	The type of the index to be created for the Milvus collection.	IVF_FLAT

Property	Description	Default value
spring.ai.vectorstore.milvus.metric-type	The metric type to be used for the Milvus collection.	COSINE
spring.ai.vectorstore.milvus.index-parameters	The index parameters to be used for the Milvus collection.	{"nlist":1024}
spring.ai.vectorstore.milvus.id-field-name	The ID field name for the collection	doc_id
spring.ai.vectorstore.milvus.is-auto-id	Boolean flag to indicate if the auto-id is used for the ID field	false
spring.ai.vectorstore.milvus.content-field-name	The content field name for the collection	content
spring.ai.vectorstore.milvus.metadata-field-name	The metadata field name for the collection	metadata
spring.ai.vectorstore.milvus.embedding-field-name	The embedding field name for the collection	embedding
spring.ai.vectorstore.milvus.client.host	The name or address of the host.	localhost
spring.ai.vectorstore.milvus.client.port	The connection port.	19530
spring.ai.vectorstore.milvus.client.uri	The uri of Milvus instance	-
spring.ai.vectorstore.milvus.client.token	Token serving as the key for identification	-

Property	Description	Default value
	and authentication purposes.	
spring.ai.vectorstore.milvus.client.connect-timeout-ms	Connection timeout value of client channel. The timeout value must be greater than zero .	10000
spring.ai.vectorstore.milvus.client.keep-alive-time-ms	Keep-alive time value of client channel. The keep-alive value must be greater than zero.	55000
spring.ai.vectorstore.milvus.client.keep-alive-timeout-ms	The keep-alive timeout value of client channel. The timeout value must be greater than zero.	20000
spring.ai.vectorstore.milvus.client.rpc-deadline-ms	Deadline for how long you are willing to wait for a reply from the server. With a deadline setting, the client will wait when encounter fast RPC fail caused by network fluctuations. The deadline value must be larger than or equal to zero.	0

Property	Description	Default value
spring.ai.vectorstore.milvus.client.client-key-path	The client.key path for tls two-way authentication, only takes effect when "secure" is true	-
spring.ai.vectorstore.milvus.client.client-pem-path	The client.pem path for tls two-way authentication, only takes effect when "secure" is true	-
spring.ai.vectorstore.milvus.client.ca-pem-path	The ca.pem path for tls two-way authentication, only takes effect when "secure" is true	-
spring.ai.vectorstore.milvus.client.server-pem-path	server.pem path for tls one-way authentication, only takes effect when "secure" is true.	-
spring.ai.vectorstore.milvus.client.server-name	Sets the target name override for SSL host name checking, only takes effect when "secure" is True. Note: this value is passed to grpc.ssl_target_name_override	-

Property	Description	Default value
spring.ai.vectorstore.milvus.client.secure	Secure the authorization for this connection, set to True to enable TLS.	false
spring.ai.vectorstore.milvus.client.idle-timeout-ms	Idle timeout value of client channel. The timeout value must be larger than zero.	24h
spring.ai.vectorstore.milvus.client.username	The username and password for this connection.	root
spring.ai.vectorstore.milvus.client.password	The password for this connection.	milvus

Starting Milvus Store

From within the src/test/resources/ folder run:

```
docker-compose up
```

To clean the environment:

```
docker-compose down; rm -Rf ./volumes
```

Then connect to the vector store on http://localhost:19530 or for management http://localhost:9001 (user: `minioadmin`, pass: `minioadmin`)

Troubleshooting

If Docker complains about resources, then execute:

```
docker system prune --all --force --volumes
```

Accessing the Native Client

The Milvus Vector Store implementation provides access to the underlying native Milvus client (`MilvusServiceClient`) through the `getNativeClient()` method:

```
MilvusVectorStore vectorStore =
context.getBean(MilvusVectorStore.class);
Optional<MilvusServiceClient> nativeClient =
vectorStore.getNativeClient();

if (nativeClient.isPresent()) {
    MilvusServiceClient client = nativeClient.get();
    // Use the native client for Milvus-specific operations
}
```

The native client gives you access to Milvus-specific features and operations that might not be exposed through the `VectorStore` interface.