MEP 26 -- Embedded Milvus

Current state: ["Under Discussion"]

ISSUE: https://github.com/milvus-io/milvus/issues/15711

PRs: TBD Keywords: embedded, python, deep learning,

Released: Milvus 2.1

Motivation

According to Milvus official document, there are many ways to install and start Milvus on your machine, including:

- Building and starting Milvus from source code at GitHub.
- Installing Milvus with Docker Compose.
- Installing Milvus with Helm.
- Installing Milvus with APT.
- Installing with Debian software package/RPM package.

We have noticed that a large number of data scientists or AI engineers in the Milvus community. Most of them work with Python on a MacBook and a lot of them think that all these ways to install Milvus are yet still too complicated. They also don't want to maintain a Milvus process (no matter in which form) in their operation systems.
The Milvus community has introduced pymilvus Python SDK in the past, but it still requires that a Milvus instance (whichever form) is already running on your machine.
For a very long time the community has been asking for a Milvus mutation that is extremely easy to use, just like pysqlite or pyrocksdb. Take pyrocksdb as an example:

```
>>> import rocksdb
>>> db = rocksdb.DB("test.db", rocksdb.Options(create_if_missing=True))
>>> db.put(b'a', b'data')
>>> print db.get(b'a')
b'data'
```

Summary

We introduce embedded Milvus in this MEP.
With embedded Milvus, you just need a clean environment with Python installed. You can then just do:

```
$ pip install pymilvus # Install pymilvus.
$ python3
>>> From ... import milvus # Milvus is brought up here.
>>> hello_milvus = Collection(…)
>>> insert_result = hello_milvus.insert(...)
>>> hello_milvus.create_index(…)
>>> hello_milvus.load(...)
>>> search_result = hello_milvus.search(…)
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
>>> exit() # Milvus is shut down, but all data lives on.
```

The code piece above is pretty self-explanatory.
You don't need any Milvus server pre-installed. And of course you don't need to keep any Milvus process running in the meantime.
The embedded Milvus starts and exits whenever you wish it to, but all data and logs persist.
We believe this embedded Milvus version makes Milvus a real "DB for AI" as it would make Milvus extremely easy to use for data scientists and AI engineers, etc.

Detailed Design

Milvus as a Library

Up until version 2.0, Milvus is a typical go project published as a go binary. Milvus also has a typical go code structure, as shown below:

```
milvus
 cmd // Entrance of Milvus binary. (the main() function)
 internal // Milvus code.
 configs
 deployments
 docs
 scripts
 ...
```

The first thing an embedded Milvus needs is to publish Milvus as a library.
Following the go code structure convention, we just need to add another layer to Milvus named pkg where we can export Milvus as a library:

```
milvus
 cmd // Entrance of Milvus binary. (the main() function)
 internal // Milvus code.
 pkg
  embedded // Where embedded Milvus code lies.
  ...
 configs
 deployments
 docs
 scripts
 ...
```

Running Go From Python

The goal is to use go code in Python. One option is to use CGO. We could export (*//export*)go code as a C library `libmilvus.so` and have Python to import this library:

```
milvus: build-cpp print-build-info
        @echo "Building Milvus ..."
        @mkdir -p $(INSTALL_PATH) && go env -w CGO_ENABLED="1" && GO111MODULE=on $(GO) build \
                -ldflags="-X 'main.BuildTags=$(BUILD_TAGS)' -X 'main.BuildTime=$(BUILD_TIME)' -X 'main.
GitCommit=$(GIT_COMMIT)' -X 'main.GoVersion=$(GO_VERSION)'" \
                -buildmode=c-shared-o $(INSTALL_PATH)/libmilvus.so $(PWD)/pkg/embedded/embeddedmilvus.go 1>/dev
/null
```

And in Python we could:

```
import ctypes

libmilvus = ctypes.cdll.LoadLibrary('./bin/libmilvus.so')

embedded_milvus = libmilvus.embedded_milvus()
embedded_milvus() # Milvus main() equivalent.
```

The Milvus service usually takes tens of seconds to fully start (which we will optimize later). It is a good idea to keep a background thread with running Milvus who should always stand ready to answer user's calls.

External Dependencies

The Milvus standalone version used to depend on external dependencies, namely MinIO and Etcd. With the Milvus 2.1 release we will have both dependencies removed, by providing options too: (1) replacing MinIO with local disk storage and (2) Replacing Etcd server with embedded Etcd.
To disable MinIO and use local disk storage: [TBD]
To enable embedded Etcd: Toggle `etcd.use.embed` option ON in `milvus.yaml` file.

Logging

Logging should be suppressed during the embedded Milvus run, otherwise your program can easily get flooded with logs. We propose that all logs, no matter which level, should not be printed to the console.

Working with Pymilvus

Pymilvus is an essential part of Milvus and still the most popular SDK. Embedded Milvus works with Pymilvus in the following ways:

- Embedded milvus will be constructed as an independent python package named `milvus` and will be published to The Python Package Index (PyPI).
- The embedded Milvus package includes the pymilvus package.
- A new embedded Milvus version is automatically released when a new Milvus version is released.
- Embedded Milvus package doesn't necessarily have to depend on the latest pymilvus package, but has to make sure that the depending pymilvus is fully compatible with itself.

We believe that constructing embedded Milvus as a separate PyPI package has these advantages:

1. The embedded Milvus package could be as large as ~100MBs while the pymilvus package is ~30MBs. It is too much of a burden if embedded Milvus is put into pymilvus, package size wise.
2. We could make embedded Milvus and pymilvus completely independent. However, this would make version control (amongst Milvus, embedded Milvus, pymilvus) very complicated. Also, doing two `pip install` operations is not a good user experience.

Comparing Embedded Milvus and Pymilvus

Recall that with pymilvus SDK, you will need to:

1. Install Docker.
2. Start a Milvus instance with all its dependencies (MinIO, Etcd and Pulsar if running Milvus cluster) with `docker-compose up`.
3. Install pymilvus with "pip install pymilvus". (what, do you guys not have Python?)
4. From pymilvus, connect to the Milvus instance you just brought up in 2.

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

5. Start playing with Milvus. Enjoy your time.
6. Shut down Milvus with `docker-compose down`.

However, with the embedded Milvus embedded, all you need to do is:

1. Install Pymilvus with "pip install pymilvus".
2. Import embedded Milvus with:

   ```
   from ... import milvus # Milvus will start.
   ```

3. Start playing with Milvus. Again, enjoy your time.
4. `exit()` if you are in Python interactive mode, otherwise your Python script should already finish gracefully.

Best Practice: Embedded Milvus versus Pymilvus

Embedded Milvus and pymilvus are really constructed for different scenarios. You may consider using embedded Milvus when:

- You want to use Milvus but do not want to install Milvus in any way.
- You want to use Milvus but do not want to keep a long running Milvus process in your machine.
- You want to use Milvus but feel too tired to start a Milvus process (even if you know how to do so).

It is suggested that you should NOT use embedded Milvus if:

- You are to use embedded Milvus in your production environment (Please don't).
- You have strict performance needs (embedded Milvus doesn't have the best performance).

Cross Milvus Migration

Embedded Milvus is not designed for production environment deployment. However, we will guarantee that everything you did in embedded Milvus can "run everywhere".

Anything thing you have done in embedded Milvus, all the scripts and code you have written, can be 100% migrated and runnable on other Milvus instances, no matter what form, including Milvus standalone, Milvus cluster and cloud-native Milvus.

Test Plans

- Embedded Milvus should have some unit tests for itself.

- It is suggested to apply a part of (if not all of) Milvus standalone tests on embedded Milvus.

Future Work

- Milvus was not born as a library. We made embedded Milvus running in a background daemon thread. When communicating with embedded Milvus, we are still making gRPC calls. It will be great if these gRPC calls can be replaced with function calls.