Gebo.ai configuration manual (0.9.9.6-TECH-PREVIEW/SNAPSHOT)

Enterprise open source retrieve augmented generation, chatbot and agents platform.

Sommario

Gebo.ai “monolithic version” on premise installation structure

This document lists both already developed and under development integrations.

The “monolithic version” is called so contraposed to the “Microservices version” that will be available soon, even if suboptimal “Monolithic” software in this case is easier to install and already tested for installations with up to 10000 documents indexed and retrieved in the chat context.

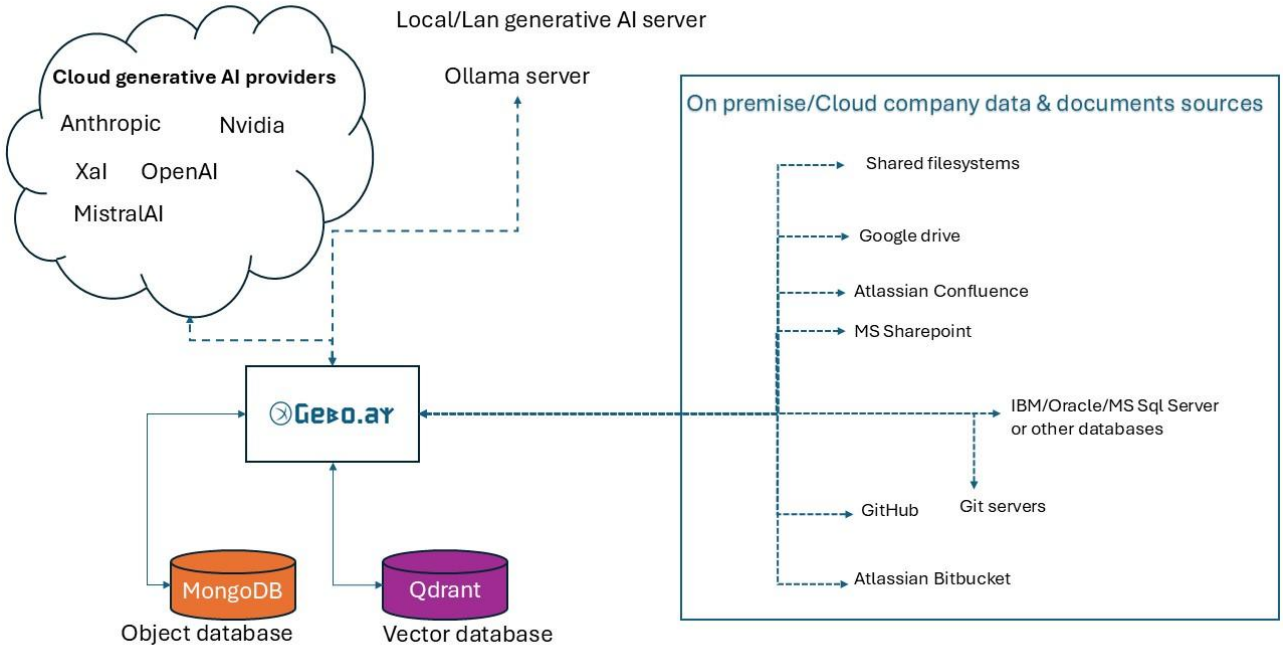

Figure 1Gebo.ai on premise installation components and 3rd party integration schema

Docker compose installation

Important prerequisites: **if you plan to use the docker-compose installation ensure to run a “UTF-8” locale like en_US.UTF-8 localization or it_IT.UTF-8 (for Italian) this is mandatorly if you plan to mount windows filesystem shares to be accessible to Gebo.ai.**

On ubuntu server for example run:

#sudo apt-get install language-pack-en

#sudo locale-gen en_US.UTF-8

#update-locale LANG=en_US.UTF-8

The provided docker compose **docker-compose.yml** configuration files works without any required modification.

It setups both MongDB, Qdrant, Gebo.ai setting up their connectivity and exporting the port 12999 where the gebo.ai application is bound.

The docker-compose configuration let the 3 applications run as a service, so if in your system dockerd or the containerized subsystem is running as a service with automatic startup at system boot, you are fully ready on erogating this service on your local network.

It is very productive to create an "up and running" quite performant installation but contains security settings like mongo password and qdrant api key that we suggest to change for production use (go to the security considerations section to check these topics).

1st Starting up the application

The fastest way to obtain an up & running environment is copying the provided docker-compose.yml in a folder and run:

user@localhost#docker-compose up -d

2nd point your browser to the <server address>:12999 port

This will start the application setup in your browser, the absolute mandatory step here is to register an administrator user with a password.

All others configuration steps now are possible with a system that will be already having a restrict access to an administrator user that will be able to configure new users and all the other technical details of the system.

Preparation from scratch of an Ubuntu server 24.04 system to run the docker-compose distribution of Gebo.ai

If you require to prepare a system from scratch to use Gebo.ai follow the instructions:

- Install Ubuntu server 24.04 LTS
- Adjust the character encoding of your system
    - Install language pack with:"**sudo apt-get install language-pack-en**" (or other language pack of your local language or company filesystems locales)
    - Run "**locale-gen en_US.UTF-8**"
    - Run "**sudo update-locale LANG=en_US.UTF-8**"
- Install and configure docker
    - Run "**sudo apt-get install docker.io**"
    - Run "**sudo apt-get install docker-compose**"
    - Run "**sudo service docker start**"
- Create gebo.ai user to run docker-compose system:
    - check docker group gid in /etc/group (110)

    - create gebo.ai user with group docker and bourn shell as a shell, home directory /home/gebo.ai with: “**sudo useradd gebo.ai -g 110 -m -s /bin/bash**”
    - create a gebo.ai use password with: “**sudo passwd gebo.ai**”
- Launching gebo.ai infrastructure
    - Download docker-compose.yml from https://gebo.ai/ in /home/gebo.ai
    - Switch user to gebo.ai “**sudo su – gebo.ai**”
    - Launching the infrastructure: “**sudo docker-compose up -d**”

Mounting windows shared filesystems (SMB CIFS) to be visible to your gebo.ai linux server:

If you plan to host gebo.ai in a linux server or in a linux server using docker-compose (see preceiding chapters), please ensurie that your linux service ha an extensive support for eventually present UTF-8 characters encoding in file names (view preceiding chapters).

To successfully let your windows shared filesystem being visible to your gebo.ai server let’s mount them with the following commands:

#mount //<server name>/<share path> /mnt/windows-share -t cifs -o username=<username>,password=<password>,uid=<uid nr>,gid=<gid nr>,iocharset=utf8

Or

#mount //<server name>/<share path> /mnt/windows-share -t cifs -o username=<username>,password=<password>,uid=<uid nr>,gid=<gid nr>,iocharset=utf8,codepage=cp850

You can statically provide the reference to your company’s windows shares to your /etc/fstab configuration file and configure those mounted volumes to be visible to your gebo.ai system.

“Monolithic version” base installation components & integration with 3rd party systems.

The gebo.ai monolithic version software is an application that runs on linux/unix/windows and on containerized systems using java virtual machines 17 or higher (preferable the 21 or 22 versions) runtimes.

Minimum software stack for the installation

- Jvm 21 or 22 (already shipped in bundle with the geboai/gebo.ai docker container).
- A reachable **MongoDB** installation (also a common edition is OK)

- A reachable Vector database, preferably **QDRANT** that the more tested one.

MongoDB is the only part of the infrastructure that must be configured dealing directly with the software configuration files, almost all the other integrations or required infrastructure parts can be configured from the administrative UI.

Connectivity with cloud generative AI providers or local generative AI server

The software can use the following cloud or on-premise generative AI providers/servers, you require at least to configure 2 default models for the software:

- a "chat large language model" (like GPT4o/O1/Claude 3.X/Llama 3.X or others)→ this is the chatbot that will perform generative AI session elaboration and output.
- an "embedding model" like OpenAi text-embedding-3-large or NVIDIA/NV-Embed-v2 or others. This model is responsible for transforming document contents on their semantic numerical vector representation to support data retrieval.

Supported providers (and some supported models examples)

From networking/technical point of view those of the following cloud providers chosen must be configured to have full accessibility from the server where Gebo.ai is installed on. In other words if you chose to use OpenAI or Xai or other suppliers, at least https networking with those domains have to be permitted on the company firewalls.

All the cloud or on premise generative AI providers or servers can be directly configured using the software administrative UI, inserting directly in the user interface api key or other company subscription credentials.

The credentials management uses cryptographic algorithms to let those access informations being securely stored and unreadable even if the database is hacked (go to the security considerations sections).

- OpenAi, chat models: gpt4o / O1 / O3, embedding models: text-embedding-3-large (suggested, best multilanguage)
- Antrophic, chat models: Claude 3.X
- MistralAI, chat models: Mistral Large/Codestral/Mistral mini embedding models: Mistral Embed (English only).
- Xai, chat models: Grok, embedding models: embedding-beta
- Groq provider, chat models: Meta Llama 3.1 3.2 3.3 8b/70b
- Nvidia Nim/Cloud platform, chat models: https://build.nvidia.com/explore/reasoning embedding models: https://build.nvidia.com/explore/retrieval
- Supported soon: Google AI infrastructures.
- Other infrastructures compatible with OpenAI API can be tested with the generical OpenAI API connection profile.

Supported local/lan generative AI server: ollama server can run Meta Llama 3.X and a long list of chat models and various open source embedding models, for a realistic use invesing in powerful Nvidia GPUS is a must.

Connectivity with company used on premise or cloud systems

1st connectivity with various “content systems” or “file sharing systems”

The software has an administrative area where users with administrative rights can configure integrations with all the Cloud or “On premise” content management systems or control version systems or files/documents sharing systems.

2nd composing knowledge bases

Administrative users will composes huge knowledge bases adding documents chosen from all the available CMS/shared filesystems/versioning systems to be available via retrieve augmented generation while Gebo.ai users interact with chat bots.

The server where Gebo.ai is installed must have access to all of those network resources.

All the content management services/version management services/sharing filesystem services/database services providers or servers can be directly configured using the software administrative UI, inserting directly in the user interface alla required credentials.

The credentials management uses cryptographic algorithms to let those access informations being securely stored and unreadable even if the database is hacked (go to the security considerations sections).

Gebo.ai detailed configuration

The application distributed in the docker environment with an application.yml configuration that already works in the context where the provided docker-compose.yml is used.

The internal path on the container is /opt/gebo.ai/config/application.yml, if necessary it can be extracted from the standard container running:

docker cp <container name>:/opt/gebo.ai/config/application.yml ./application.yml

or it can be edited directly inside the container.

All the following chapters mention configuration keys and values that already exist in this file or that can be added to change the software behavior.

Be careful when handling the yml file format to use a software with proper syntax management like visual code with the yml format plugin. This format requires proper formatting to be accepted by the application.

Use an application.properties format if not familiar with the yaml format, you can simply use this translator: https://mageddo.com/tools/yaml-converter to convert the application.yml to an application.properties file and use this in place of the first once removed.

GEBO_HOME environment variable or system property

When the application is started it requires a GEBO_HOME environment variable (already set in the docker container to /opt/gebo.ai/home) .

This variable must point to an existing folder that is usable to store minimal application internal configurations. It is convenient to backup this folder periodically.

To ensure coherency on persistence using the docker image we suggest to mount this folder to an existing folder on the host system and backup it periodically.

GEBO_WORK_DIRECTORY environment variable or system property or interactive admin UI setting.

The application requires the setting of a “work directory” used to store its working files, It is convenient to backup this folder periodically.

The work directory can be handle in 2 different way:

- Work directory interactively set at gebo.ai interactive setup by the administrative user using the administrator user interface, behavior configured when the configuration key: ***ai.gebo.config.setupConfiguresWorkdir*** is set to true. In this state the work directory path is configured in the MongoDB system.
- Work directory configured via configuration file, behavior set with: ***ai.gebo.config.setupConfiguresWorkdir*** is set to false with **GEBO_WORK_DIRECTORY** environment or system property pointing to the chosen work directory (that is required to be an existing and empty folder).

In the containerized docker distribution the system work with this 2nd type of configuration with work directory being set to /opt/gebo.ai/work folder.

To ensure coherency on persistence using the docker image we suggest to mount this folder to an existing folder on the host system and backup it periodically.

MongoDB connectivity settings

This base configuration is mandatory and required for the application to start, the configuration is in the following application.yml or application.properties configuration keys

ai.gebo.mongodb.enabled: true

ai.gebo.mongodb.databaseName: gebo-ai

ai.gebo.mongodb.connectionString: mongodb://<user>:<pwd>@localhost:27017/gebo-ai?authSource=admin

The connection string grammar reference is standard for MongoDB configurations and is documented here: https://www.mongodb.com/docs/manual/reference/connection-string/

If you are using the docker-compose launch system the provided docker-compose.yml file sets up MongoDB already coherent with these settings shipped with the provided configuration.

MongoDB is used as NOSQL database for configurations, application data and much more, so it must be considered part of the "persistent storage class" of the application, managed taking into account security and under backup management.

QDRANT or other vector database configuration

This base configuration can be provided from the application.yml configuration file, however if is not provided a clean new installation started up without it and let the administrator user configure it interactively in the administration UI "setup" section.

Configuration keys:

ai.gebo.vectorstore.use: QDRANT

ai.gebo.vectorstore.qdrant.host: 127.0.0.1

ai.gebo.vectorstore.qdrant.port: 6334

ai.gebo.vectorstore.qdrant.tls: false

ai.gebo.vectorstore.qdrant.apiKey: <api key value>

The vector database is a mandatory component of the system when we have configured at least one large language model with "embedding" capabilities, Gebo.ai create separate vector databases connection for each "embedding model" and configured Knowledge base, so once configured and your company started to use the retrieve augmented generation system it has to be considered as a part of the "persistent storage class" in the application stack keeping it secure and with managed backup.

Company shared filesystem

Company shared filesystems paths, due to their delicate handling, must first being configured to be reachable from the Gebo.ai software.

If you are using the provided docker-compose.yml file to install/launch the system you need to mount your filesystems under the **/opt/gebo.ai/shares** on the linux docker installation using the standard documer (or integrating it in docker-compose.yml) "volume mounting" commands.