On this page

Installation

AutoGen is a versatile tool that can be installed and run in Docker or locally using a virtual environment. Below are detailed instructions for both methods.

Option 1: Install and Run AutoGen in Docker

Docker, an indispensable tool in modern software development, offers a compelling solution for AutoGen's setup. Docker allows you to create consistent environments that are portable and isolated from the host OS. With Docker, everything AutoGen needs to run, from the operating system to specific libraries, is encapsulated in a container, ensuring uniform functionality across different systems. The Dockerfiles necessary for AutoGen are conveniently located in the project's GitHub repository at https://github.com/microsoft/autogen/tree/main/samples/dockers.

Pre-configured DockerFiles: The AutoGen Project offers pre-configured Dockerfiles for your use. These Dockerfiles will run as is, however they can be modified to suit your development needs. Please see the README.md file in autogen/samples/dockers

- **autogen_base_img**: For a basic setup, you can use the `autogen_base_img` to run simple scripts or applications. This is ideal for general users or those new to AutoGen.
- **autogen_full_img**: Advanced users or those requiring more features can use `autogen_full_img`. Be aware that this version loads ALL THE THINGS and thus is very large. Take this into consideration if you build your application off of it.

Step 1: Install Docker

- **General Installation**: Follow the official Docker installation instructions. This is your first step towards a containerized environment, ensuring a consistent and isolated workspace for AutoGen.
- **For Mac Users**: If you encounter issues with the Docker daemon, consider using colima. Colima offers a lightweight alternative to manage Docker containers efficiently on macOS.

Step 2: Build a Docker Image

AutoGen now provides updated Dockerfiles tailored for different needs. Building a Docker image is akin to setting the foundation for your project's environment:

- **Autogen Basic (dockerfile.base)**: Ideal for general use, this setup includes common Python libraries and essential dependencies. Perfect for those just starting with AutoGen.

```
docker build -f samples/dockers/Dockerfile.base -t autogen_base_img https://github.com/microsoft/autogen.git
```

- **Autogen Advanced (dockerfile.full)**: Advanced users or those requiring all the things that AutoGen has to offer `autogen_full_img`

```
docker build -f samples/dockers/Dockerfile.full -t autogen_full_img https://github.com/microsoft/autogen.git
```

Step 3: Run AutoGen Applications from Docker Image

Here's how you can run an application built with AutoGen, using the Docker image:

1. **Mount Your Directory**: Use the Docker `-v` flag to mount your local application directory to the Docker container. This allows you to develop on your local machine while running the code in a consistent Docker environment. For example:

   ```
   docker run -it -v $(pwd)/myapp:/home/autogen/autogen/myapp autogen_base_img:latest python /home/autogen/autoge
   ```

   Here, `$(pwd)/myapp` is your local directory, and `/home/autogen/autogen/myapp` is the path in the Docker container where your code will be located.

2. **Mount your code:** Now suppose you have your application built with AutoGen in a main script named `twoagent.py` (example) in a folder named `myapp`. With the command line below, you can mount your folder and run the application in Docker.

```
# Mount the local folder `myapp` into docker image and run the script named "twoagent.py" in the docker.
docker run -it -v `pwd`/myapp:/myapp autogen_img:latest python /myapp/main_twoagent.py
```

3. **Port Mapping**: If your application requires a specific port, use the `-p` flag to map the container's port to your host. For instance, if your app runs on port 3000 inside Docker and you want it accessible on port 8080 on your host machine:

   ```
   docker run -it -p 8080:3000 -v $(pwd)/myapp:/myapp autogen_base_img:latest python /myapp
   ```

   In this command, `-p 8080:3000` maps port 3000 from the container to port 8080 on your local machine.

4. **Examples of Running Different Applications**: Here is the basic format of the docker run command.

```
docker run -it -p {WorkstationPortNum}:{DockerPortNum} -v {WorkStation_Dir}:{Docker_DIR} {name_of_the_image} {bash/
```

- *Simple Script*: Run a Python script located in your local `myapp` directory.

  ```
  docker run -it -v `pwd`/myapp:/myapp autogen_base_img:latest python /myapp/my_script.py
  ```

- *Web Application*: If your application includes a web server running on port 5000.

  ```
  docker run -it -p 8080:5000 -v $(pwd)/myapp:/myapp autogen_base_img:latest
  ```

- *Data Processing*: For tasks that involve processing data stored in a local directory.

  ```
  docker run -it -v $(pwd)/data:/data autogen_base_img:latest python /myapp/process_data.py
  ```

Additional Resources

- For more information on Docker usage and best practices, refer to the official Docker documentation.
- Details on how to use the Dockerfile.dev version can be found on the Contributing

Option 2: Install AutoGen Locally Using Virtual Environment

When installing AutoGen locally, we recommend using a virtual environment for the installation. This will ensure that the dependencies for AutoGen are isolated from the rest of your system.

Option a: venv

You can create a virtual environment with `venv` as below:

```
python3 -m venv pyautogen
source pyautogen/bin/activate
```

The following command will deactivate the current `venv` environment:

```
deactivate
```

Option b: conda

Another option is with `Conda`. You can install it by following this doc, and then create a virtual environment as below:

```
conda create -n pyautogen python=3.10  # python 3.10 is recommended as it's stable and not too old
conda activate pyautogen
```

The following command will deactivate the current `conda` environment:

```
conda deactivate
```

Option c: poetry

Another option is with `poetry`, which is a dependency manager for Python.

Poetry is a tool for dependency management and packaging in Python. It allows you to declare the libraries your project depends on and it will manage (install/update) them for you. Poetry offers a lockfile to ensure repeatable installs, and can build your project for distribution.

You can install it by following this doc, and then create a virtual environment as below:

```
poetry init
poetry shell

poetry add pyautogen
```

The following command will deactivate the current `poetry` environment:

```
exit
```

Now, you're ready to install AutoGen in the virtual environment you've just created.

Python

AutoGen requires **Python version >= 3.8, < 3.13**. It can be installed from pip:

```
pip install pyautogen
```

`pyautogen<0.2` requires `openai<1`. Starting from pyautogen v0.2, `openai>=1` is required.

Migration guide to v0.2

openai v1 is a total rewrite of the library with many breaking changes. For example, the inference requires instantiating a client, instead of using a global class method. Therefore, some changes are required for users of `pyautogen<0.2`.

- `api_base` -> `base_url`, `request_timeout` -> `timeout` in `llm_config` and `config_list`. `max_retry_period` and `retry_wait_time` are deprecated. `max_retries` can be set for each client.
- MathChat is unsupported until it is tested in future release.
- `autogen.Completion` and `autogen.ChatCompletion` are deprecated. The essential functionalities are moved to `autogen.OpenAIWrapper`:

```
from autogen import OpenAIWrapper
client = OpenAIWrapper(config_list=config_list)
response = client.create(messages=[{"role": "user", "content": "2+2="}])
print(client.extract_text_or_completion_object(response))
```

- Inference parameter tuning and inference logging features are currently unavailable in `OpenAIWrapper`. Logging will be added in a future release. Inference parameter tuning can be done via `flaml.tune`.
- `seed` in autogen is renamed into `cache_seed` to accommodate the newly added `seed` param in openai chat completion api. `use_cache` is removed as a kwarg in `OpenAIWrapper.create()` for being automatically decided by `cache_seed`: int | None. The difference between autogen's `cache_seed` and openai's `seed` is that:
  - autogen uses local disk cache to guarantee the exactly same output is produced for the same input and when cache is hit, no openai api call will be made.
  - openai's `seed` is a best-effort deterministic sampling with no guarantee of determinism. When using openai's `seed` with `cache_seed` set to None, even for the same input, an openai api call will be made and there is no guarantee for getting exactly the same output.

Optional Dependencies

- **Docker**

Even if you install AutoGen locally, we highly recommend using Docker for code execution.

To use docker for code execution, you also need to install the python package `docker`:

```
pip install docker
```

You might want to override the default docker image used for code execution. To do that set `use_docker` key of `code_execution_config` property to the name of the image. E.g.:

```
user_proxy = autogen.UserProxyAgent(
    name="agent",
    human_input_mode="TERMINATE",
    max_consecutive_auto_reply=10,
    code_execution_config={"work_dir":"_output", "use_docker":"python:3"},
    llm_config=llm_config,
    system_message="""Reply TERMINATE if the task has been solved at full satisfaction.
Otherwise, reply CONTINUE, or the reason why the task is not solved yet."""
)
```

- **blendsearch**

`pyautogen<0.2` offers a cost-effective hyperparameter optimization technique EcoOptiGen for tuning Large Language Models. Please install with the [blendsearch] option to use it.

```
pip install "pyautogen[blendsearch]<0.2"
```

Example notebooks:

Optimize for Code Generation

Optimize for Math

- **retrievechat**

`pyautogen` supports retrieval-augmented generation tasks such as question answering and code generation with RAG agents. Please install with the [retrievechat] option to use it.

```
pip install "pyautogen[retrievechat]"
```

RetrieveChat can handle various types of documents. By default, it can process plain text and PDF files, including formats such as 'txt', 'json', 'csv', 'tsv', 'md', 'html', 'htm', 'rtf', 'rst', 'jsonl', 'log', 'xml', 'yaml', 'yml' and 'pdf'. If you install unstructured (`pip install "unstructured[all-docs]"`), additional document types such as 'docx', 'doc', 'odt', 'pptx', 'ppt', 'xlsx', 'eml', 'msg', 'epub' will also be supported.

You can find a list of all supported document types by using `autogen.retrieve_utils.TEXT_FORMATS`.

Example notebooks:

Automated Code Generation and Question Answering with Retrieval Augmented Agents

Group Chat with Retrieval Augmented Generation (with 5 group member agents and 1 manager agent)

Automated Code Generation and Question Answering with Qdrant based Retrieval Augmented Agents

- **Teachability**

To use Teachability, please install AutoGen with the [teachable] option.

```
pip install "pyautogen[teachable]"
```

Example notebook: Chatting with a teachable agent

- **Large Multimodal Model (LMM) Agents**

We offered Multimodal Conversable Agent and LLaVA Agent. Please install with the [lmm] option to use it.

```
pip install "pyautogen[lmm]"
```

Example notebooks:

LLaVA Agent

- **mathchat**

`pyautogen<0.2` offers an experimental agent for math problem solving. Please install with the [mathchat] option to use it.

```
pip install "pyautogen[mathchat]<0.2"
```

Example notebooks:

Using MathChat to Solve Math Problems

Edit this page

Previous
« Getting Started

Next
Multi-agent Conversation Framework »

Community

Discord

Twitter