Getting Started with AutoGPT: Self-Hosting Guide

Introduction

This guide will help you setup the server and builder for the project.

> **Warning**
>
> **DO NOT FOLLOW ANY OUTSIDE TUTORIALS AS THEY WILL LIKELY BE OUT OF DATE**

Prerequisites

To setup the server, you need to have the following installed:

- Node.js
- Docker
- Git

Checking if you have Node.js & NPM installed

We use Node.js to run our frontend application.

If you need assistance installing Node.js: https://nodejs.org/en/download/

NPM is included with Node.js, but if you need assistance installing NPM: https://docs.npmjs.com/downloading-and-installing-node-js-and-npm

You can check if you have Node.js & NPM installed by running the following command:

```
node -v
npm -v
```

Once you have Node.js installed, you can proceed to the next step.

Checking if you have Docker & Docker Compose installed

Docker containerizes applications, while Docker Compose orchestrates multi-container Docker applications.

If you need assistance installing docker: https://docs.docker.com/desktop/

Docker-compose is included in Docker Desktop, but if you need assistance installing docker compose: https://docs.docker.com/compose/install/

You can check if you have Docker installed by running the following command:

```
docker -v
docker compose -v
```

Once you have Docker and Docker Compose installed, you can proceed to the next step.

Raspberry Pi 5 Specific Notes

On Raspberry Pi 5 with Raspberry Pi OS, the default 16K page size will cause issues with the `supabase-vector` container (expected: 4K).
To fix this, edit `/boot/firmware/config.txt` and add:

```
kernel=kernel8.img
```

Then reboot. You can check your page size with:

```
getconf PAGESIZE
```

`16384` means 16K (incorrect), and `4096` means 4K (correct). After adjusting, `docker compose up -d --build` should work normally.
See supabase/supabase #33816 for additional context.

Setup

Cloning the Repository

The first step is cloning the AutoGPT repository to your computer. To do this, open a terminal window in a folder on your computer and run:

```
git clone https://github.com/Significant-Gravitas/AutoGPT.git
```

Floating button icon

If you get stuck, follow this guide.

Once that's complete you can continue the setup process.

Running the backend services

To run the backend services, follow these steps:

- Navigate to the `autogpt_platform` directory inside the AutoGPT folder:

```
cd AutoGPT/autogpt_platform
```

- Copy the `.env.example` file to `.env` in `autogpt_platform`:

```
cp .env.example .env
```

  This command will copy the `.env.example` file to `.env` in the `supabase` directory. You can modify the `.env` file to add your own environment variables.

- Run the backend services:

```
docker compose up -d --build
```

  This command will start all the necessary backend services defined in the `docker-compose.combined.yml` file in detached mode.

Running the frontend application

To run the frontend application open a new terminal and follow these steps:

- Navigate to `frontend` folder within the `autogpt_platform` directory:

```
cd frontend
```

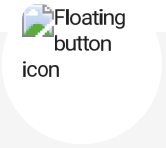
- Copy the `.env.example` file available in the `frontend` directory to `.env` in the same directory:

```
cp .env.example .env
```

You can modify the `.env` within this folder to add your own environment variables for the frontend application.

- Run the following command:

```
corepack enable
pnpm install
pnpm dev
```



This command will enable corepack, install the necessary dependencies with pnpm, and start the frontend application in development mode.

Checking if the application is running

You can check if the server is running by visiting http://localhost:3000 in your browser.

Notes:

By default the application for different services run on the following ports:

Frontend UI Server: 3000 Backend Websocket Server: 8001 Execution API Rest Server: 8006

Additional Notes

You may want to change your encryption key in the `.env` file in the `autogpt_platform/backend` directory.

To generate a new encryption key, run the following command in python:

```
from cryptography.fernet import Fernet;Fernet.generate_key().decode()
```

Or run the following command in the `autogpt_platform/backend` directory:

```
poetry run cli gen-encrypt-key
```

Then, replace the existing key in the `autogpt_platform/backend/.env` file with the new one.

> **Note**
>
> *The steps below are an alternative to Running the backend services*

Alternate Steps

AutoGPT Agent Server (OLD) This is an initial project for creating the next generation of agent execution, which is an AutoGPT agent server. The agent server will enable the creation of composite multi-agent systems that utilize AutoGPT agents and other non-agent components as its primitives. ##### Docs You can access the docs for the [AutoGPT Agent Server here](https://docs.agpt.co/#1-autogpt-server). ##### Setup We use the Poetry to manage the dependencies. To set up the project, follow these steps inside this directory: 0. Install Poetry

```
pip install poetry
```

1. Configure Poetry to use .venv in your project directory

```
  poetry config virtualenvs.in-project true
```

2. Enter the poetry shell

```
poetry shell
```

3. Install dependencies

```
poetry install
```

4. Copy .env.example to .env

```
cp .env.example .env
```

5. Generate the Prisma client

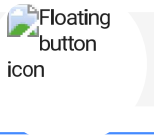
```
poetry run prisma generate
```

> In case Prisma generates the client for the global Python installation instead of the virtual environment, the current mitigation is to just uninstall the global Prisma package: > >

```
pip uninstall prisma
```

> > Then run the generation again. The path *should* look something like this: > `/pypoetry/virtualenvs/backend-TQIRSwR6-py3.12/bin/prisma` 6. Migrate the database. Be careful because this deletes current data in the database.

```
docker compose up db -d
poetry run prisma migrate deploy
```



Starting the AutoGPT server without Docker

To run the server locally, start in the autogpt_platform folder:

```
cd ..
```

Run the following command to run database in docker but the application locally:

```
docker compose --profile local up deps --build --detach
cd backend
poetry run app
```

Starting the AutoGPT server with Docker

Run the following command to build the dockerfiles:

```
docker compose build
```

Run the following command to run the app:

```
docker compose up
```

Run the following to automatically rebuild when code changes, in another terminal:

```
docker compose watch
```

Run the following command to shut down:

```
docker compose down
```

If you run into issues with dangling orphans, try:

```
docker compose down --volumes --remove-orphans && docker-compose up --force-
recreate --renew-anon-volumes --remove-orphans
```

📌 Windows Installation Note

When installing Docker on Windows, it is **highly recommended** to select **WSL 2** inst  er-V. Using Hyper-V can cause compatibility issues with Supabase, leading to the `supal` container being marked as **unhealthy**.

Steps to enable WSL 2 for Docker:

1. Install WSL 2.
2. Ensure that your Docker settings use WSL 2 as the default backend:
3. Open **Docker Desktop**.
4. Navigate to **Settings > General**.
5. Check **Use the WSL 2 based engine**.
6. Restart **Docker Desktop.**

Already Installed Docker with Hyper-V?

If you initially installed Docker with Hyper-V, you **don't need to reinstall** it. You can switch to WSL 2 by following these steps: 1. Open **Docker Desktop**. 2. Go to **Settings > General**. 3. Enable **Use the WSL 2 based engine**. 4. Restart Docker.

🚨 **Warning:** Enabling WSL 2 may **erase your existing containers and build history**. If you have important containers, consider backing them up before switching.

For more details, refer to Docker's official documentation.

Development

Formatting & Linting

Auto formatter and linter are set up in the project. To run them:

Install:

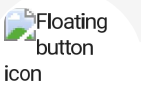
```
poetry install --with dev
```

Format the code:

```
poetry run format
```

Lint the code:

```
poetry run lint
```



Testing

To run the tests:

```
poetry run test
```

To update stored snapshots after intentional API changes:

```
pytest --snapshot-update
```

Project Outline

The current project has the following main modules:

blocks

This module stores all the Agent Blocks, which are reusable components to build a graph that represents the agent's behavior.

data

This module stores the logical model that is persisted in the database. It abstracts the database operations into functions that can be called by the service layer. Any code that interacts with Prisma objects or the database should reside in this module. The main models are: * `block`: anything related to the block used in the graph * `execution`: anything related to the execution graph execution * `graph`: anything related to the graph, node, and its relations

execution

This module stores the business logic of executing the graph. It currently has the following main modules: * `manager`: A service that consumes the queue of the graph execution and executes the graph. It contains both pieces of logic. * `scheduler`: A service that triggers scheduled graph execution based on a cron expression. It pushes an execution request to the manager.

server

This module stores the logic for the server API. It contains all the logic used for the API that allows the client to create, execute, and monitor the graph and its execution. This API service interacts with other services like those defined in `manager` and `scheduler`.

utils

This module stores utility functions that are used across the project. Currently, it h
modules: * `process`: A module that contains the logic to spawn a new process. * `s`
module that serves as a parent class for all the services in the project.

Service Communication

Currently, there are only 3 active services:

- AgentServer (the API, defined in `server.py`)
- ExecutionManager (the executor, defined in `manager.py`)
- Scheduler (the scheduler, defined in `scheduler.py`)

The services run in independent Python processes and communicate through an IPC. A communication layer (`service.py`) is created to decouple the communication library from the implementation.

Currently, the IPC is done using Pyro5 and abstracted in a way that allows a function decorated with `@expose` to be called from a different process.

Adding a New Agent Block

To add a new agent block, you need to create a new class that inherits from `Block` and provides the following information: * All the block code should live in the `blocks` (`backend.blocks`) module. * `input_schema`: the schema of the input data, represented by a Pydantic object. * `output_schema`: the schema of the output data, represented by a Pydantic object. * `run` method: the main logic of the block. * `test_input` & `test_output`: the sample input and output data for the block, which will be used to auto-test the block. * You can mock the functions declared in the block using the `test_mock` field for your unit tests. * Once you finish creating the block, you can test it by running `poetry run pytest -s test/block/test_block.py`.

June 6, 2025 July 29, 2024