VLLM on EC2

The following page presents our findings on testing VLLM in EC2. It shows how to create an EC2 instance and configure VLLM. It also includes a small test on how we can access our VLLM server via LangChain

---

Note: I was unable to install VLLM directly on my Mac, requiring me to set up an EC2 instance for these experiments. The installation process for VLLM appears to depend on the `xformers` library, which can be challenging to install on machines without Nvidia GPUs, particularly Macs.

Encountering an error with the `torch` library during VLLM installation, I was prompted to manually install the `xformers` package.

```
pip wheel --no-cache-dir --use-pep517 "xformers (==0.0.26.post1)"
```

Unfortunately, this step didn't resolve the issue and even introduced additional problems when using Poetry.

---

1 - Configuring EC2

Several Confluence pages cover creating and connecting to AWS EC2 instances, including:

- Creation of a new EC2 instance
- Connecting to EC2 GPU instance
- Connect to a new EC2 instance from Pycharm
- Using the Deep Learning AMI EC2 Instance (UP TO DATE)

However, as a first-time user, I found it challenging to piece together the complete process of creating and connecting to an instance. This led me to use **VSCode** instead of PyCharm for this specific task.

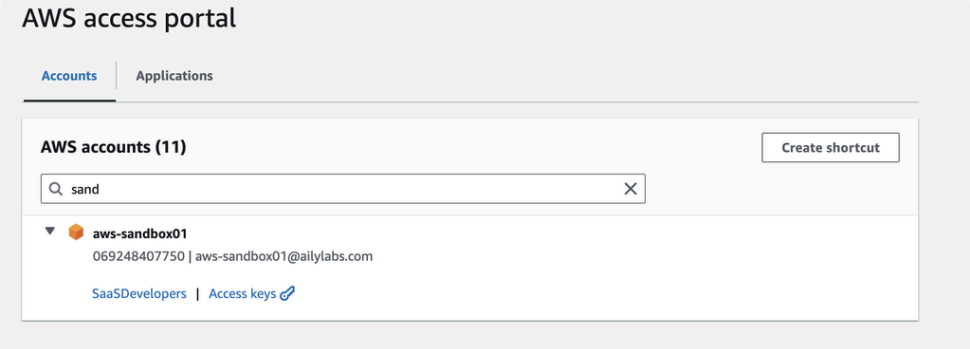
There might be better approaches, and some steps could be improved. This document outlines the process I followed for clarity.

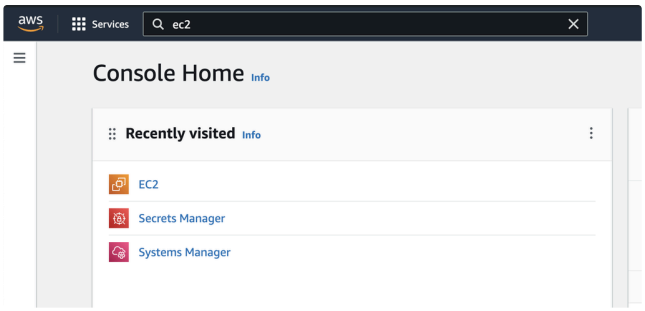
1.1 - Create an EC2 instance

This guide walks you through launching an EC2 instance in the `sandbox01` environment.

1.1.1 - Accessing the EC2 Service

Navigate to the EC2 service within the AWS Management Console. You'll typically find it under "Recently Visited," but you can also search for it if it's your first time.

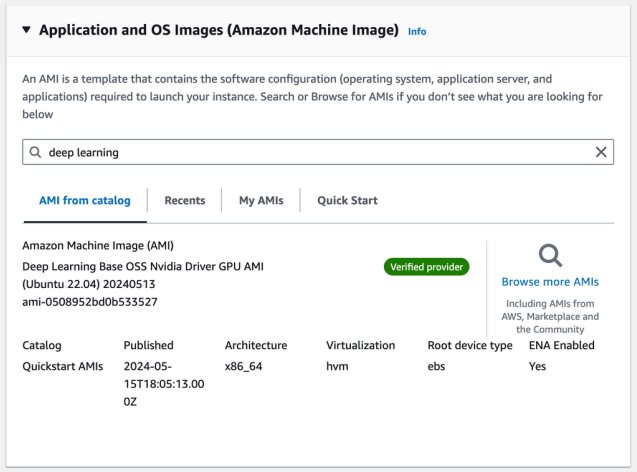




1.1.2 - Choosing an AMI (Machine Image)

Here, you'll decide on the operating system for your instance. AWS offers pre-built Amazon Machine Images (AMIs) or allows you to use a custom image you've created.

For this example, we'll use the Ubuntu AMI for Deep Learning. This AMI benefits from including open-source NVIDIA drivers, as opposed to the proprietary ones typically included in AWS Deep Learning AMIs.



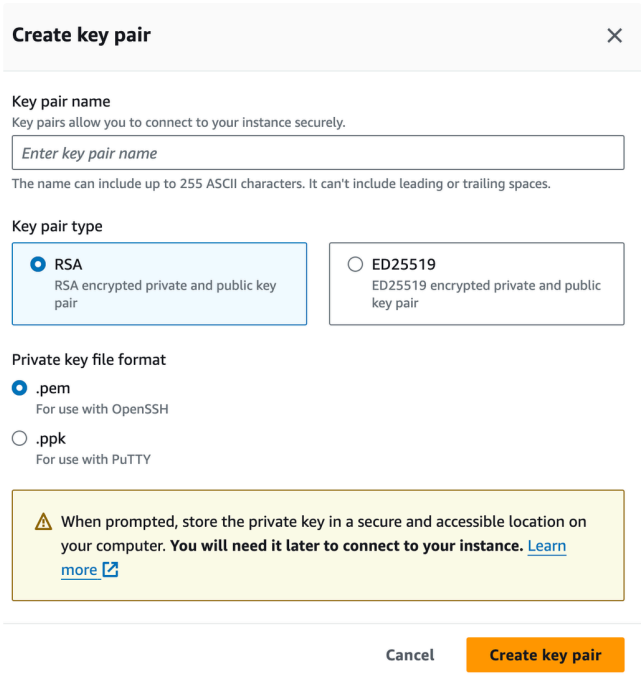
1.1.2 - Instance Type Selection

Select an appropriate instance type based on your workload requirements. In this case, a g5.2xlarge instance is chosen. However, be aware that this type can experience availability issues. For testing with smaller models, a g4dn.2xlarge might be a more suitable and readily available optio

1.1.4 - Setting Up SSH Access

During instance creation, you'll have the option to generate a new SSH key pair for secure remote access. Click to generate the corresponding .pem file, which you'll need to store securely.

Note: You can reuse the same key pair for multiple instances for convenience.

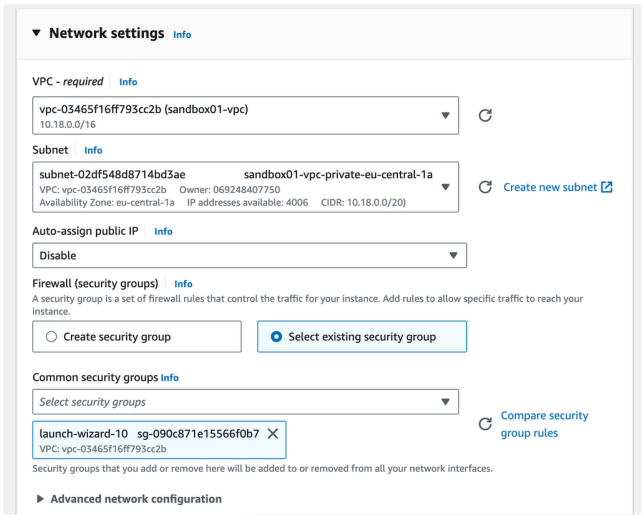


1.1.5 - Security Configuration

Define the network access rules for your instance. Here, we'll only allow connections on port 22 for SSH access. This port can be used for tunneling to expose other ports (like port 8000) for testing purposes.

Security best practice: For enhanced security, the infrastructure team recommends selecting one of the available subnets within your VPC. In this example, you can choose either `sandbox01-vpc-private-eu-central-1a` or `sandbox01-vpc-private-eu-central-1b`.

Finally, choose an existing security group to avoid creating unnecessary duplicates. The `launch-wizard-10` group should suffice for this instance.

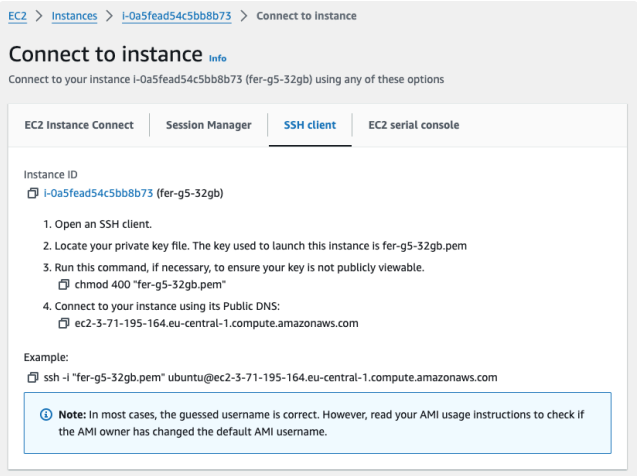


1.2 - Establishing an SSH Connection to the EC2 Instance

1.2.1 - Initiating the Connection

Once the EC2 instance has been launched using the "Start Instance" option, follow these steps to establish an SSH connection:

1. **Locate the Instance:** Navigate to the EC2 Instances dashboard within the AWS Management Console. Identify the instance you want to connect to and click on its ID.
2. **Access the Connection Panel:** On the instance's details page, locate the "Connect to instance" option and click on it.



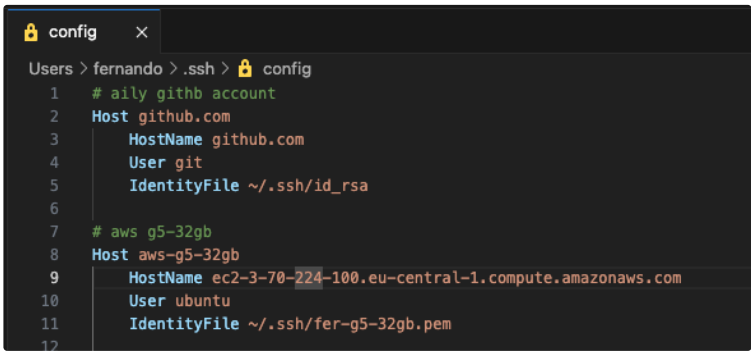
1.2.2 - Addressing Dynamic DNS Changes

It's important to note that the EC2 instance's public DNS address tends to change with each instance launch. This necessitates revisiting the "Connect to instance" panel each time you want to establish a connection. Unfortunately, to the best of my knowledge there's no fixed address that can be configured in the `.ssh/config` file.

1.2.3 - Connecting via VSCode

VSCode offers a convenient alternative for establishing SSH connections to remote servers. Here's how to utilize VSCode for this purpose:

1. **Install the Remote-SSH Extension:** Begin by installing the "Remote - SSH" extension in VSCode.
2. **Initiate Connection:** Launch the extension and click on the "Connect to Host" button.
3. **Configure Connection Details:** Choose one of two options:
   a. **Modify the Existing HostName:** Edit the existing HostName entry in your `.ssh/config` file to reflect the latest public DNS address of the EC2 instance.
   b. **Create a New Connection:** If you prefer to maintain separate configurations, create a new entry in your `.ssh/config` file using the updated public DNS address.



1.3 - Configure the EC2 instance

1.3.1 - Update System Packages

Install the essential development dependencies for Python and other tools:

```
sudo apt update
sudo apt install make build-essential libssl-dev zlib1g-dev \
libbz2-dev libreadline-dev libsqlite3-dev wget curl llvm \
libncursesw5-dev xz-utils tk-dev libxml2-dev libxmlsec1-dev libffi-dev liblzma-dev
```

1.3.2 - Install Python with Pyenv

We'll install Python using Pyenv, which allows managing multiple Python versions on your system. Here are the steps:

- **Create the `~/.python-version` file**: This file tells Pyenv the default Python version to use. Create it in your home directory and add a single line containing the desired version:

```
3.10.6
```

- **Install Pyenv and virtualenv:** Execute the following command in your terminal:

```
curl https://pyenv.run | bash
```

- **Add Pyenv and Poetry to your shell configuration file** (e.g., `~/.bashrc`):

```
# Pyenv
export PYENV_ROOT="$HOME/.pyenv"
command -v pyenv >/dev/null || export PATH="$PYENV_ROOT/bin:$PATH"
eval "$(pyenv init -)"
eval "$(pyenv virtualenv-init -)"

# Poetry (assuming installation in ~/.local/bin)
export PATH="$HOME/.local/bin:$PATH"
```

- **Reload your shell configuration:**

```
source ~/.bashrc
```

- **Install the desired Python Version with Pyenv:**

```
pyenv install 3.10.6
```

- **Reload your shell configuration again**

```
source ~/.bashrc
```

1.3.4 - Install Poetry

Install it using the following commands:

```
python3 -m pip install --user pipx  # Installs pipx for managing Poetry globally
python3 -m pipx ensurepath         # Adds pipx's bin directory to your PATH
pipx install poetry                  # Installs Poetry
```

2 - Exploring VLLM Capabilities

VLLM simplifies LLM interaction through its built-in HTTP server, which mimics OpenAI's Completions and Chat API. This allows you to use the same API calls as OpenAI's models, making it incredibly easy to integrate VLLM with Langchain.

Key features:

- **Multi-Model Support:** VLLM's server can host multiple models simultaneously. You can list available models with `curl http://localhost:8000/v1/models`.
- **Access Control:** VLLM-OpenAI allows restricting access to your model using API keys. This ensures only authorized users can interact with it.
- **Complete OpenAI Compatibility:** VLLM-OpenAI offers all functionalities supported by OpenAI's API, including asynchronous, streaming, and batch processing.

2.1 - GitHub repositories

To test VLLM's capabilities, we created two small GitHub repositories

- fernandorodriguez-aily/vllm-server

- fernandorodriguez-aily/vllm-client

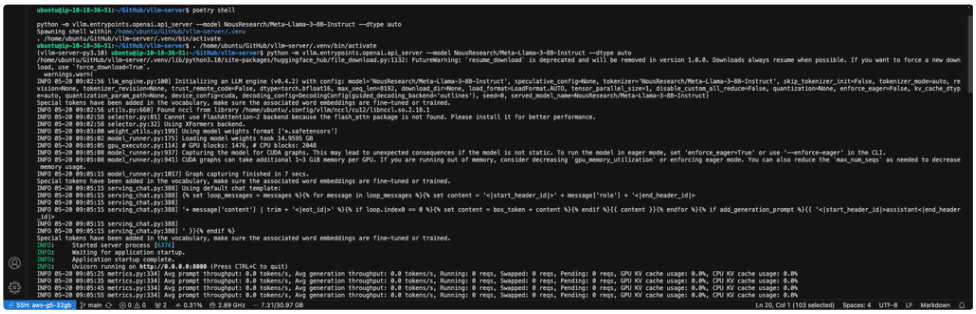
We used separate repositories due to dependency conflicts. Additionally, VLLM dependencies currently prevent running the server on macOS. Therefore, the recommended approach is to run the server on an EC2 instance and the client on your Mac.

2.2 - Start VLLM server

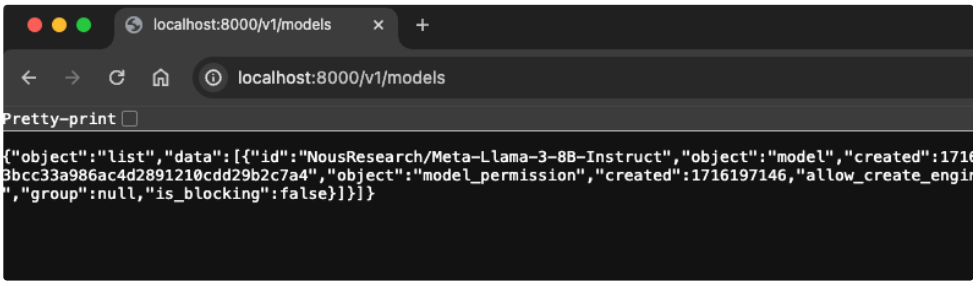
Once the `vllm-server` is installed, we can start it by running:

```
poetry shell
python -m vllm.entrypoints.openai.api_server --model NousResearch/Meta-Llama-3-8B-Instruct --dtype auto
```

It will then wait for requests, showing the average token throughput:



We can test that it works by accessing http://localhost:8000/v1/models on a browser:



It will show currently deployed models (in our case, `NousResearch/Meta-Llama-3-8B-Instruct`)

2.3 - Interacting with VLLM-OpenAI Server

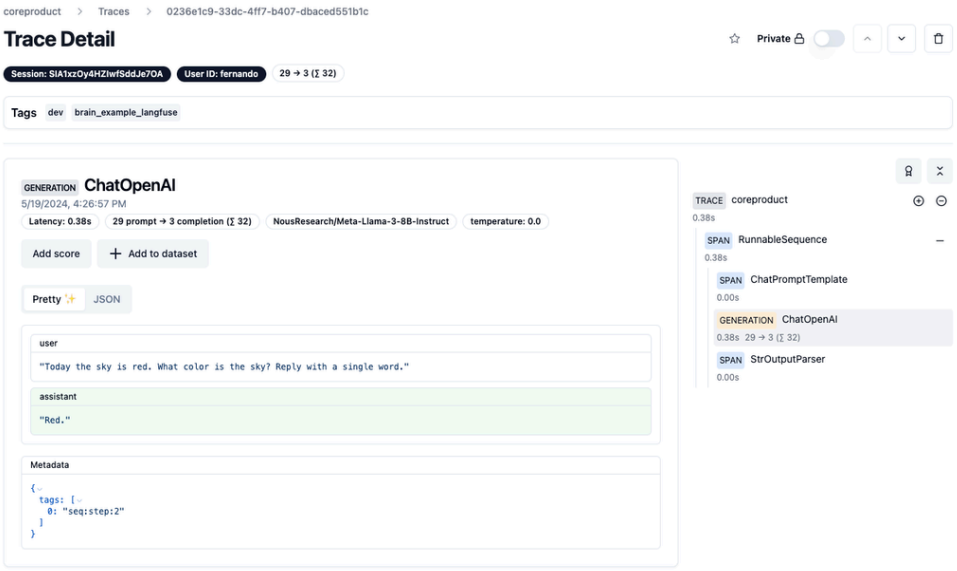
The `vllm-client` repository (and accompanying documentation) demonstrates two ways to interact with a VLLM-OpenAI server:

1. **Using the OpenAI Library:** This method offers the advantage of utilizing extra VLLM-specific parameters beyond the traditional OpenAI API. These parameters are included in the `extra_body` section of the request.
2. **Using LangChain with ChatOpenAI:** This approach follows the standard LangChain method but, as far as we have seen, it doesn't currently leverage VLLM's extra features.
   a. We plan on incorporating this functionality to the aily-brain repository

Here is an example of we what see in the server when a request is directed to it:

2.4 - Langfuse integration

From our small experiments, VLLM is compatible with our current Langfuse infrastructure:



3 - Next Steps: Production-Ready VLLM Deployment

While deploying VLLM on an EC2 instance is a valuable approach for initial testing and experimentation, for production environments, a more robust and scalable solution is recommended. Here's how we can take this to the next level:

- **Dockerize the VLLM Server:** Containerizing the server with Docker offers several benefits. It packages the server with all its dependencies, simplifying deployment across different environments. Docker also streamlines server management and replication.
- **Deploy in Kubernetes:** Leveraging Kubernetes as an orchestration platform allows us to manage multiple VLLM server instances efficiently. Kubernetes provides features such as automatic scaling, load balancing, and self-healing, ensuring a highly available and reliable LLM service.

References

Llama 2 and Llama 3.1 Hardware Requirements: GPU, CPU, RAM

How We Saved 10s of Thousands of Dollars Deploying Low Cost Open Source AI Technologies At Scale with Kubernetes