GENERATIVE AI SERIES

Ollama — Build a ChatBot with Langchain, Ollama & Deploy on Docker

Working with Ollama to run models locally, build LLM applications that can be deployed as docker containers.

A B Vijay Kumar · Follow
5 min read · Feb 21, 2024

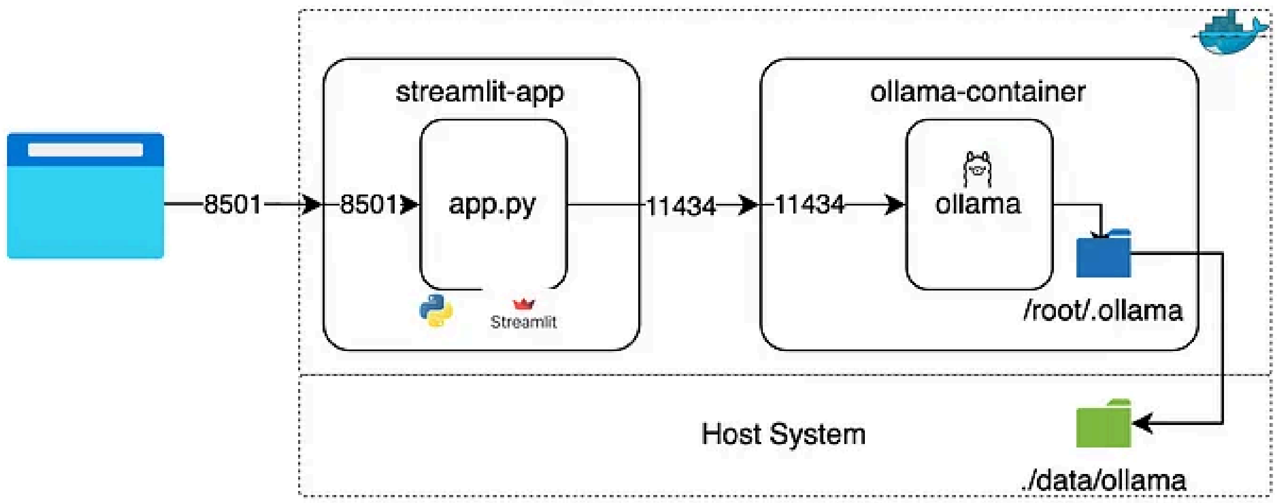
This blog is an ongoing series on GenerativeAI and is a continuation of the previous blogs. In this blog series, we will explore Ollama, and build applications that we can deploy in a distributed architectures using docker.

Ollama is a framework that makes it easy to run powerful language models on your own computer. Please refer to *Ollama — Brings runtime to serve LLMs everywhere. | by A B Vijay Kumar | Feb, 2024 | Medium* for an introduction to Ollama. In this blog we will be building the langchain application and deploying on Docker.

Langchain Chatbot application for Ollama

Let's build the chatbot application using Langshan, to access our model from the Python application, we will be building a simple Steamlit chatbot application. We will be deploying this Python application in a container and will be using Ollama in a different container. We will build the infrastructure using docker-compose. If you do not know how to use docker, or docker-compose, please go through some tutorials on internet, before you go any further.

The following picture shows the architecture of how the containers interact, and what ports they will be accessing.



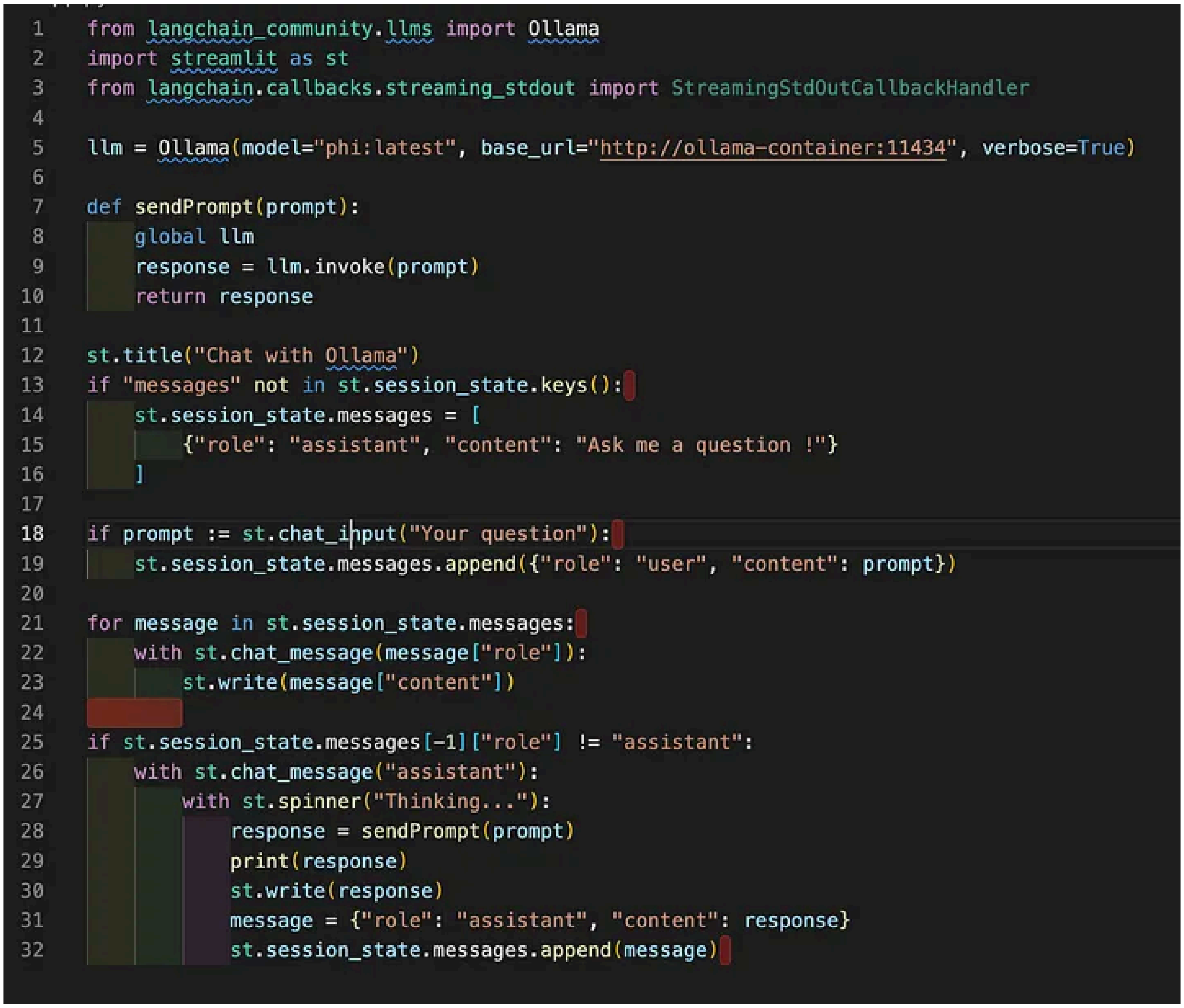
We will build 2 containers,

- Ollama container will be using the host volume to store and load the models (`/root/.ollama` is mapped to the local `./data/ollama`). Ollama container will listen on 11434 (external port, which is internally mapped to 11434)
- Streamlit chatbot application will listen on 8501 (external port, which is internally mapped to 8501).

Before we start coding, lets setup a Python virtual environment.

```
python3 -m venv ./ollama-langchain-venv
source ./ollama-langchain-venv/bin/activate
```

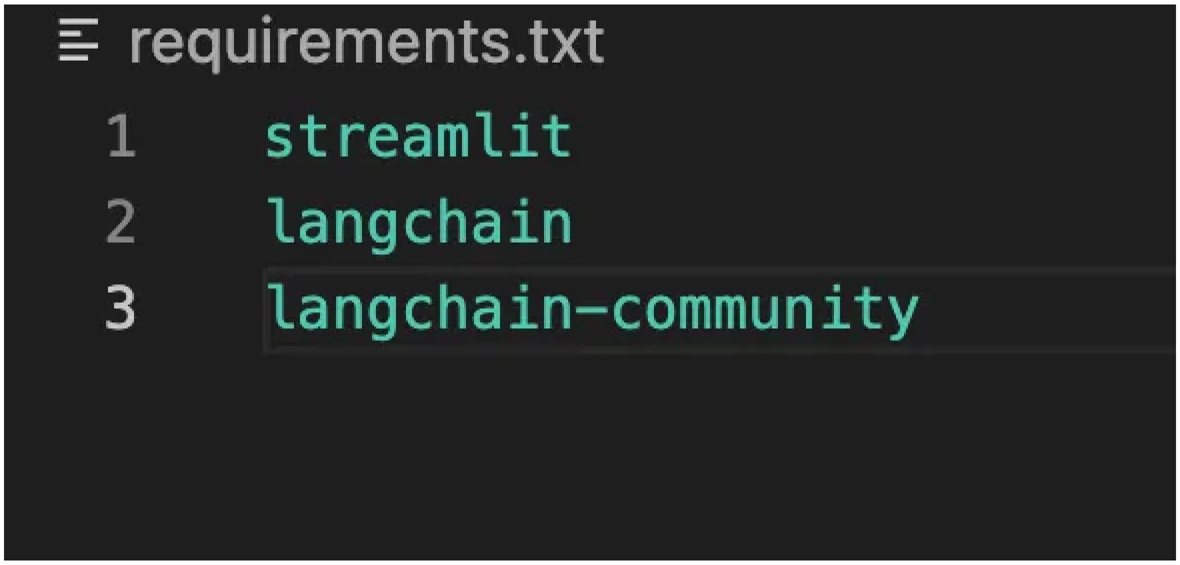
The following is the source code for streamlit application.



```python
from langchain_community.llms import Ollama
import streamlit as st
from langchain.callbacks.streaming_stdout import StreamingStdOutCallbackHandler

llm = Ollama(model="phi:latest", base_url="http://ollama-container:11434", verbose=True)

def sendPrompt(prompt):
    global llm
    response = llm.invoke(prompt)
    return response

st.title("Chat with Ollama")
if "messages" not in st.session_state.keys():
    st.session_state.messages = [
        {"role": "assistant", "content": "Ask me a question !"}
    ]

if prompt := st.chat_input("Your question"):
    st.session_state.messages.append({"role": "user", "content": prompt})

for message in st.session_state.messages:
    with st.chat_message(message["role"]):
        st.write(message["content"])

if st.session_state.messages[-1]["role"] != "assistant":
    with st.chat_message("assistant"):
        with st.spinner("Thinking..."):
            response = sendPrompt(prompt)
            print(response)
            st.write(response)
            message = {"role": "assistant", "content": response}
            st.session_state.messages.append(message)
```

This is very similar source code as I have built in my previous blogs. You can refer to my other blog *Retrieval Augmented Generation(RAG) — Chatbot for documents with LlamaIndex | by A B Vijay Kumar | Feb, 2024 | Medium* for details on how this code works. The main difference is we are using `Ollama` and calling the model through Ollama Langchain library (which is part of `langchain_community`)

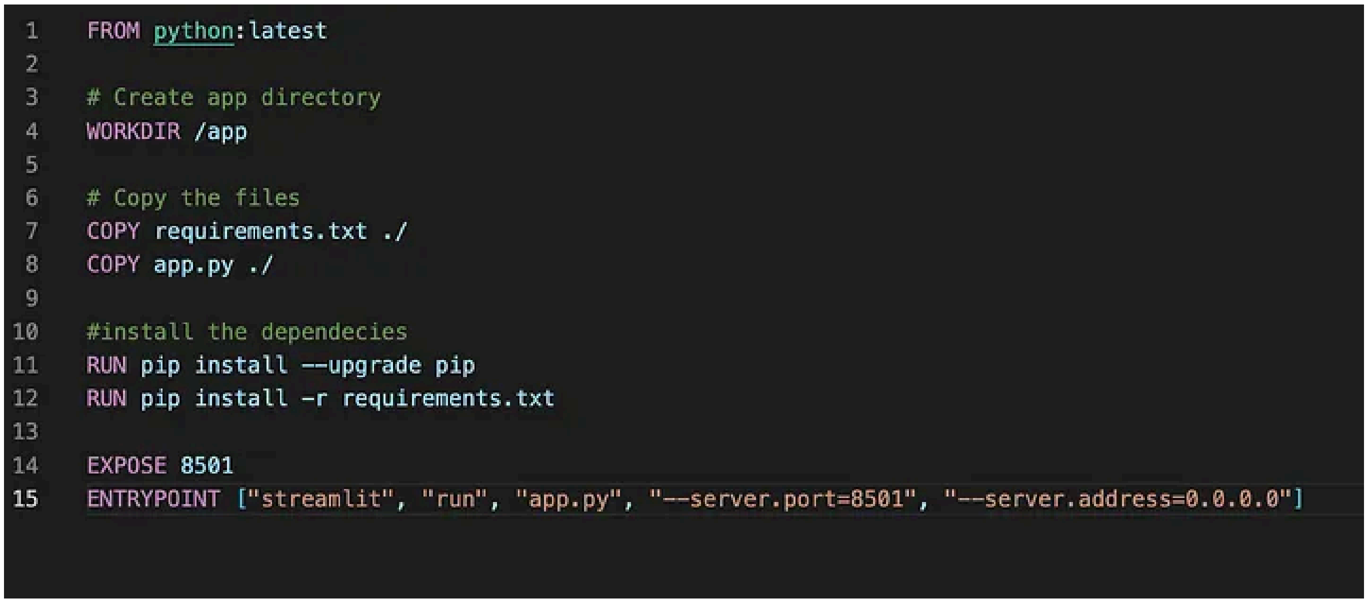
Let's define the dependencies in requirement.txt.



requirements.txt

```
streamlit
langchain
langchain-community
```

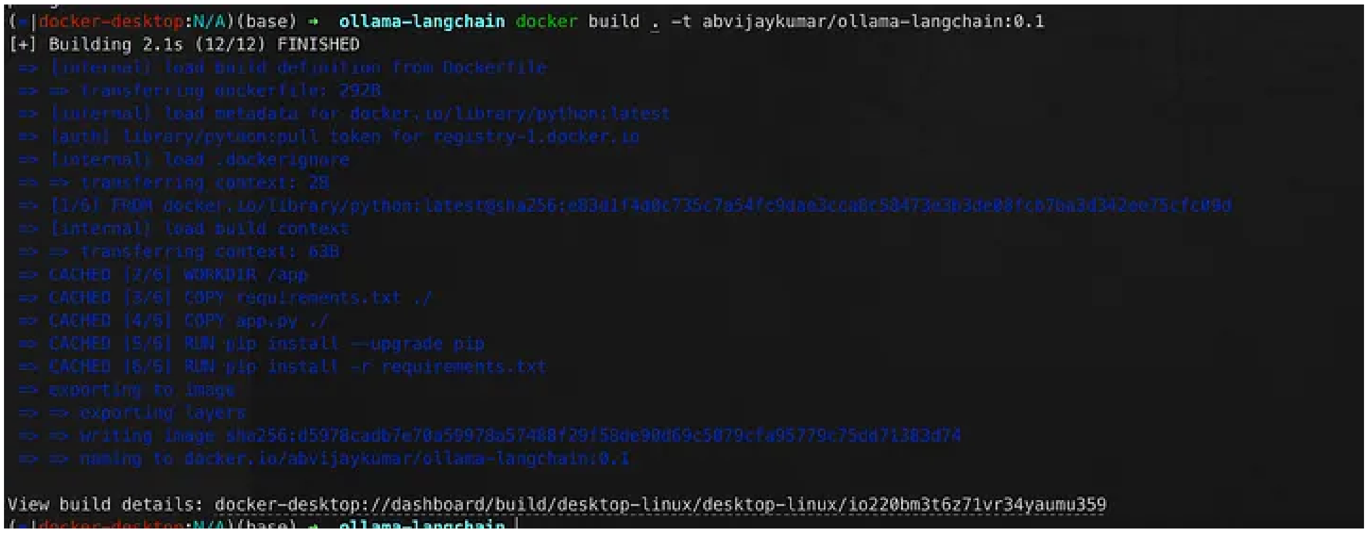
Let's now define a Dockerfile to build the docker image of the Streamlit application.



```
FROM python:latest

# Create app directory
WORKDIR /app

# Copy the files
COPY requirements.txt ./
COPY app.py ./

#install the dependecies
RUN pip install --upgrade pip
RUN pip install -r requirements.txt

EXPOSE 8501
ENTRYPOINT ["streamlit", "run", "app.py", "--server.port=8501", "--server.address=0.0.0.0"]
```

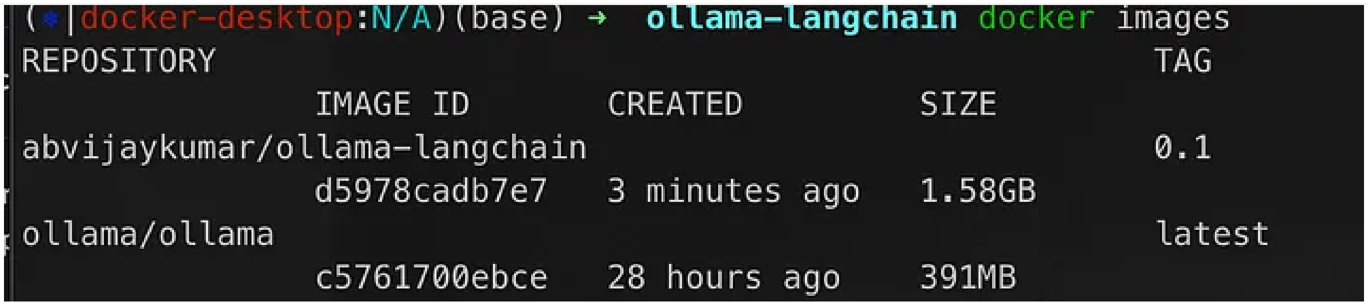
We are using the python docker image, as the base image, and creating a working directory called `/app`. We are then copying our application files there, and running the `pip installs` to install all the dependencies. We are then exposing the port 8501 and starting the `streamlit` application.

We can build the docker image using `docker build` command, as shown below.



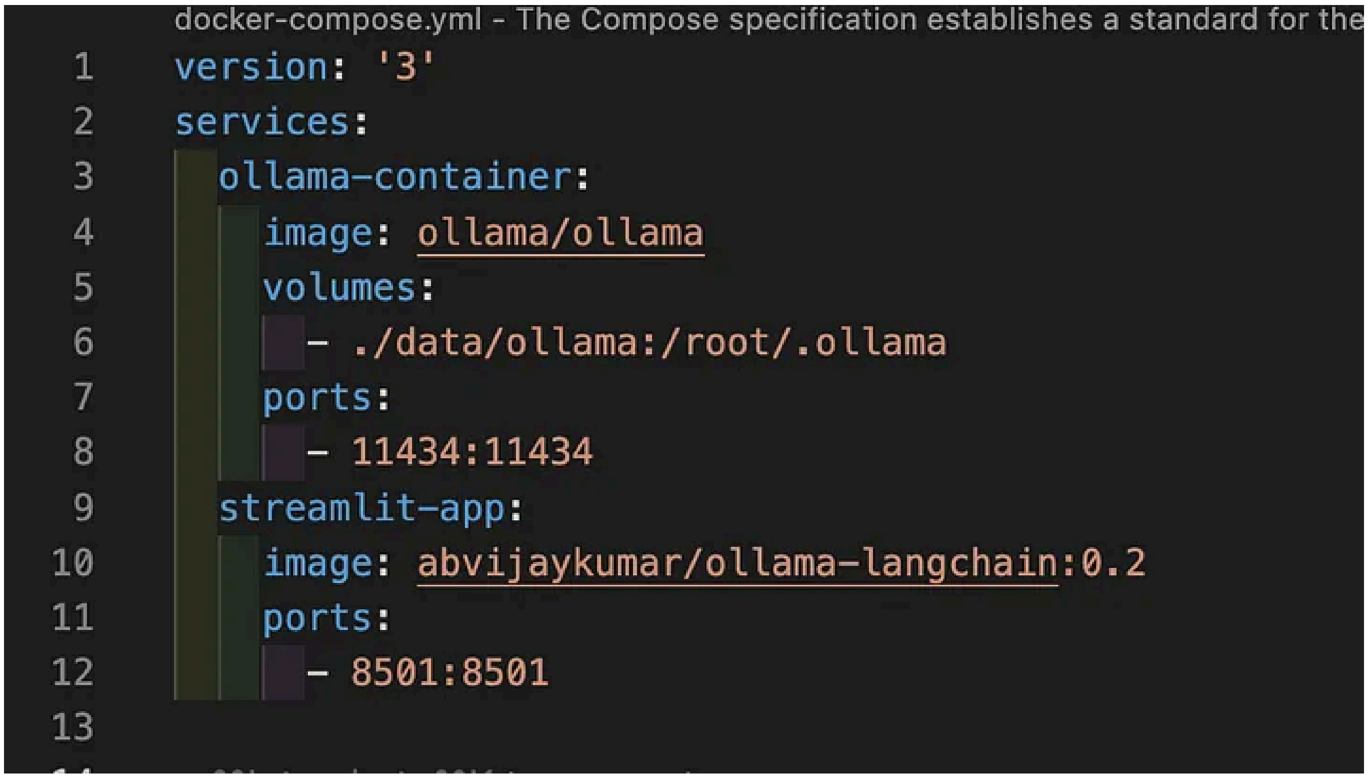
```
( |docker-desktop:N/A)(base) →  ollama-langchain docker build . -t abvijaykumar/ollama-langchain:0.1
[+] Building 2.1s (12/12) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 292B
 => [internal] load metadata for docker.io/library/python:latest
 => [auth] library/python:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/6] FROM docker.io/library/python:latest@sha256:e83d1f4d0c735c7a54fc9dae3cca8c58473a3b3de08fcb7ba3d342ee75cfc09d
 => [internal] load build context
 => => transferring context: 63B
 => CACHED [2/6] WORKDIR /app
 => CACHED [3/6] COPY requirements.txt ./
 => CACHED [4/6] COPY app.py ./
 => CACHED [5/6] RUN pip install --upgrade pip
 => CACHED [6/6] RUN pip install -r requirements.txt
 => exporting to image
 => => exporting layers
 => => writing image sha256:d5978cadb7e70a59978a57488f29f58de90d69c5079cfa95779c75dd71383d74
 => => naming to docker.io/abvijaykumar/ollama-langchain:0.1

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/io220bm3t6z71vr34yaumu359
( |docker-desktop:N/A)(base) →  ollama-langchain
```

You should be able to check if the Docker image is built, using `docker images`



```
(⎈|docker-desktop:N/A)(base) ➜  ollama-langchain docker images
REPOSITORY                                                  TAG
                  IMAGE ID       CREATED          SIZE
abvijaykumar/ollama-langchain                               0.1
                  d5978cadb7e7   3 minutes ago    1.58GB
ollama/ollama                                               latest
                  c5761700ebce   28 hours ago     391MB
```

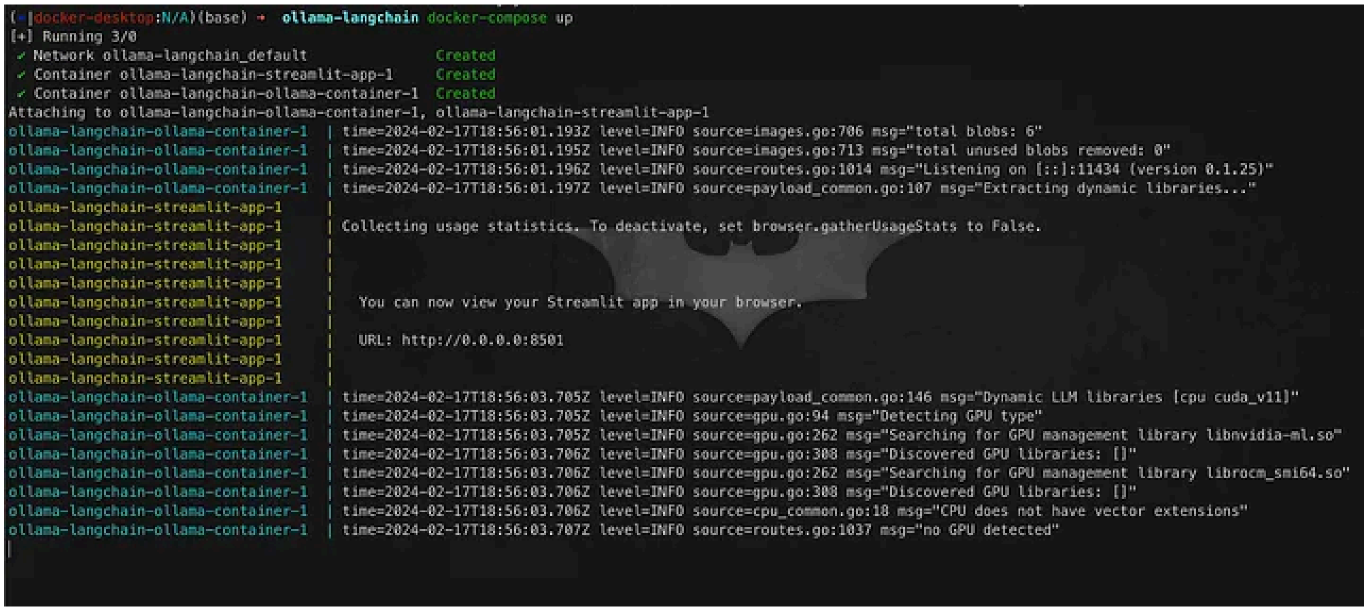
Let's now build a docker-compose configuration file, to define the network of the Streamlit application and the Ollama container, so that they can interact with each other. We will also be defining the various port configurations, as shown in the picture above. For Ollama, we will also be mapping the volume, so that whatever models are pulled, are persisted.



docker-compose.yml - The Compose specification establishes a standard for the

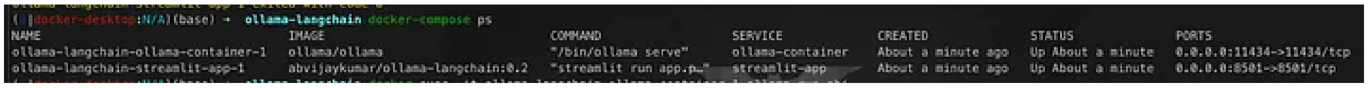
```yaml
version: '3'
services:
  ollama-container:
    image: ollama/ollama
    volumes:
      - ./data/ollama:/root/.ollama
    ports:
      - 11434:11434
  streamlit-app:
    image: abvijaykumar/ollama-langchain:0.2
    ports:
      - 8501:8501

```

We can bring up the applications by running the docker-compose up command, once you execute `docker-compose up`, you should be able to see that both the containers start running, as shown in the screenshot below.



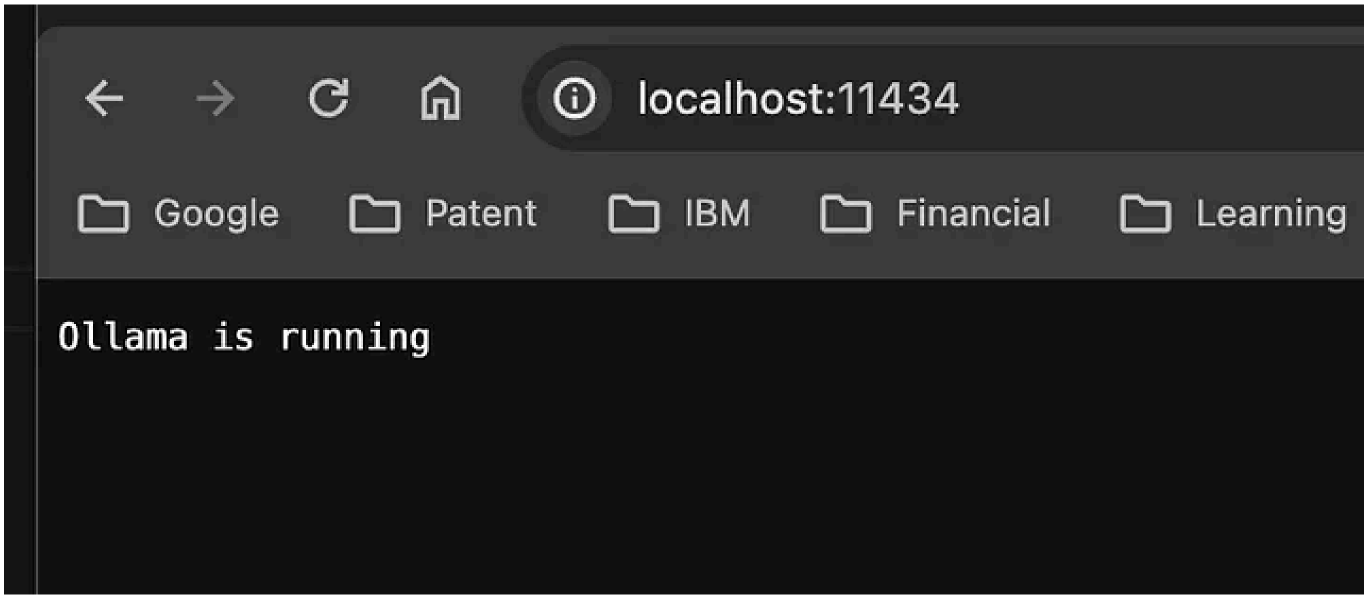
```
( |docker-desktop:N/A)(base) → ollama-langchain docker-compose up
[+] Running 3/0
 ✓ Network ollama-langchain_default                 Created
 ✓ Container ollama-langchain-streamlit-app-1       Created
 ✓ Container ollama-langchain-ollama-container-1    Created
Attaching to ollama-langchain-ollama-container-1, ollama-langchain-streamlit-app-1
ollama-langchain-ollama-container-1  | time=2024-02-17T18:56:01.193Z level=INFO source=images.go:706 msg="total blobs: 6"
ollama-langchain-ollama-container-1  | time=2024-02-17T18:56:01.195Z level=INFO source=images.go:713 msg="total unused blobs removed: 0"
ollama-langchain-ollama-container-1  | time=2024-02-17T18:56:01.196Z level=INFO source=routes.go:1014 msg="Listening on [::]:11434 (version 0.1.25)"
ollama-langchain-ollama-container-1  | time=2024-02-17T18:56:01.197Z level=INFO source=payload_common.go:107 msg="Extracting dynamic libraries..."
ollama-langchain-streamlit-app-1     |
ollama-langchain-streamlit-app-1     | Collecting usage statistics. To deactivate, set browser.gatherUsageStats to False.
ollama-langchain-streamlit-app-1     |
ollama-langchain-streamlit-app-1     |
ollama-langchain-streamlit-app-1     |
ollama-langchain-streamlit-app-1     |   You can now view your Streamlit app in your browser.
ollama-langchain-streamlit-app-1     |
ollama-langchain-streamlit-app-1     |   URL: http://0.0.0.0:8501
ollama-langchain-streamlit-app-1     |
ollama-langchain-streamlit-app-1     |
ollama-langchain-ollama-container-1  | time=2024-02-17T18:56:03.705Z level=INFO source=payload_common.go:146 msg="Dynamic LLM libraries [cpu cuda_v11]"
ollama-langchain-ollama-container-1  | time=2024-02-17T18:56:03.705Z level=INFO source=gpu.go:94 msg="Detecting GPU type"
ollama-langchain-ollama-container-1  | time=2024-02-17T18:56:03.705Z level=INFO source=gpu.go:262 msg="Searching for GPU management library libnvidia-ml.so"
ollama-langchain-ollama-container-1  | time=2024-02-17T18:56:03.706Z level=INFO source=gpu.go:308 msg="Discovered GPU libraries: []"
ollama-langchain-ollama-container-1  | time=2024-02-17T18:56:03.706Z level=INFO source=gpu.go:262 msg="Searching for GPU management library librocm_smi64.so"
ollama-langchain-ollama-container-1  | time=2024-02-17T18:56:03.706Z level=INFO source=gpu.go:308 msg="Discovered GPU libraries: []"
ollama-langchain-ollama-container-1  | time=2024-02-17T18:56:03.706Z level=INFO source=cpu_common.go:18 msg="CPU does not have vector extensions"
ollama-langchain-ollama-container-1  | time=2024-02-17T18:56:03.707Z level=INFO source=routes.go:1037 msg="no GPU detected"
```

you should be able to see the containers running by executing `docker-compose ps` command as shown below.



```
( |docker-desktop:N/A)(base) → ollama-langchain docker-compose ps
NAME                                  IMAGE                                 COMMAND                 SERVICE            CREATED               STATUS              PORTS
ollama-langchain-ollama-container-1   ollama/ollama                         "/bin/ollama serve"     ollama-container   About a minute ago   Up About a minute   0.0.0.0:11434->11434/tcp
ollama-langchain-streamlit-app-1      abvijaykumar/ollama-langchain:0.2     "streamlit run app.p…"  streamlit-app      About a minute ago   Up About a minute   0.0.0.0:8501->8501/tcp
```

We should be able to check, if ollama is running by calling http://localhost:11434, as shown in the screenshot below.

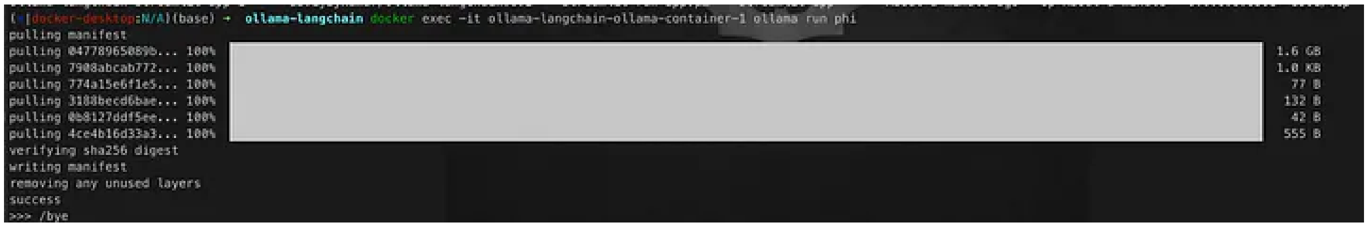


```
Ollama is running
```

Let's now download the required model, by logging into the docker container using the docker exec command as shown below.

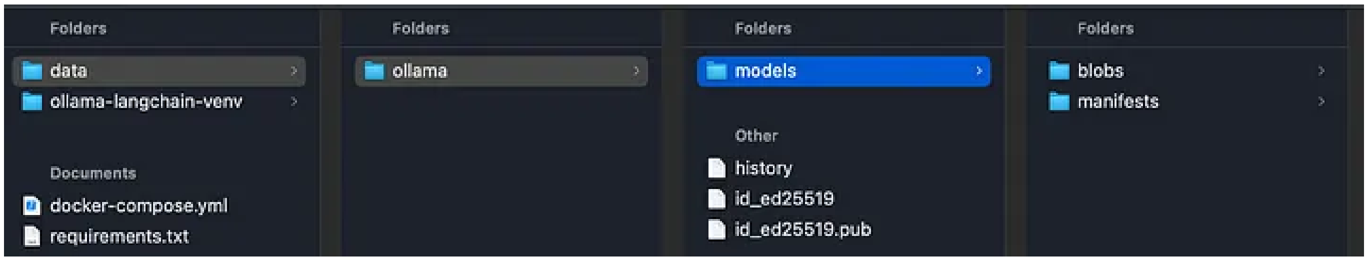
```
docker exec -it ollama-langchain-ollama-container-1 ollama run phi
```

Since we are using the model phi, we are pulling that model and testing it by running it. you can see the screenshot below, where the phi model is downloaded and will start running (since we are using `-it` flag we should be able to interact and test with sample prompts)

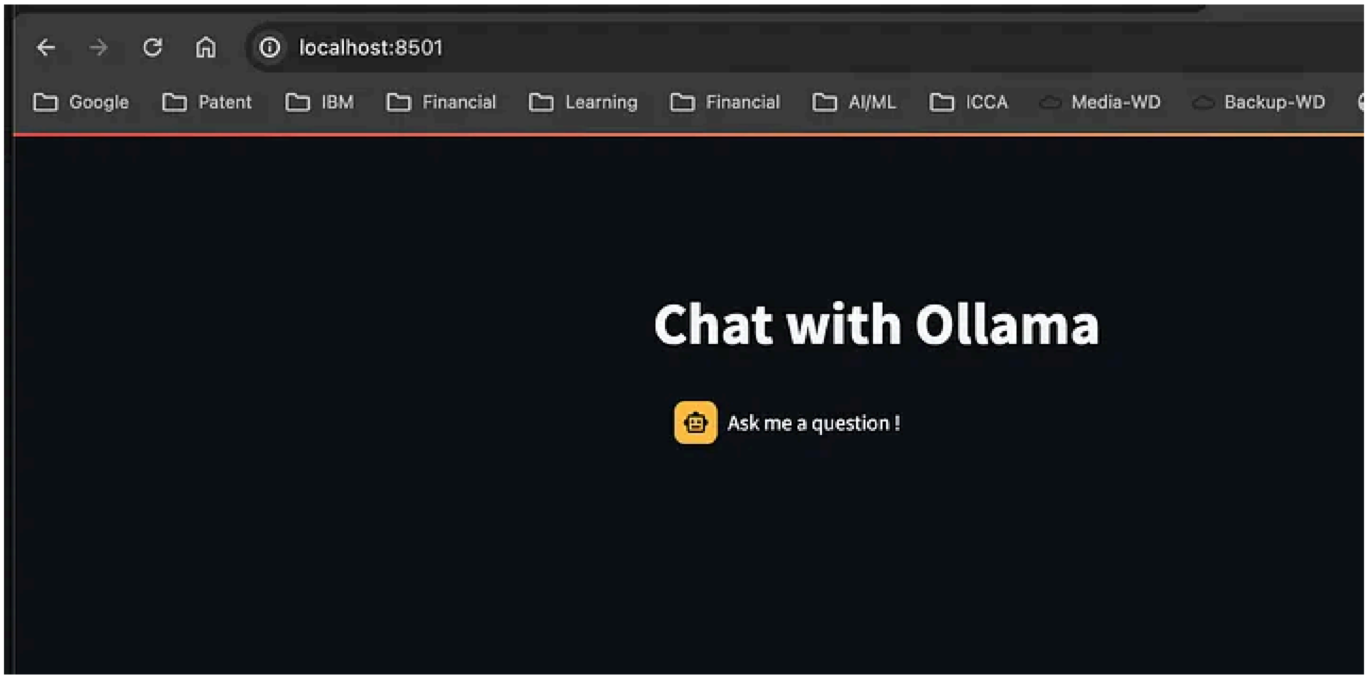


```
(⎈|docker-desktop:N/A)(base) → ollama-langchain docker exec -it ollama-langchain-ollama-container-1 ollama run phi
pulling manifest
pulling 04778965089b... 100%                                                    1.6 GB
pulling 7908abcab772... 100%                                                    1.0 KB
pulling 774a15e6f1e5... 100%                                                      77 B
pulling 3188becd6bae... 100%                                                     132 B
pulling 0b8127ddf5ee... 100%                                                      42 B
pulling 4ce4b16d33a3... 100%                                                     555 B
verifying sha256 digest
writing manifest
removing any unused layers
success
>>> /bye
```

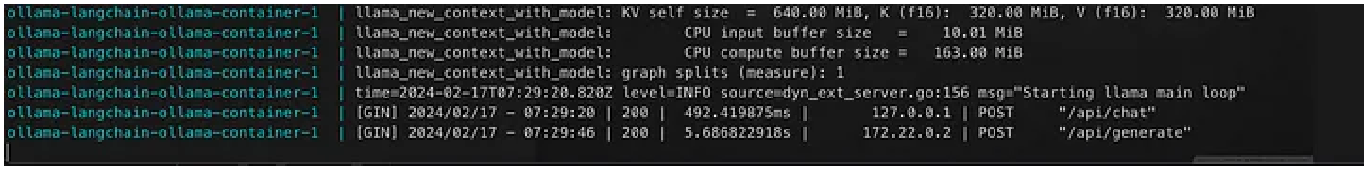
you should be able to see the downloaded model files and manifests in your local folder `./data/ollama` (which is internally mapped to `/root/.ollama` for the container, which is where Ollama looks for the downloaded models to serve)



Lets now run access our streamlit application by opening http://localhost:8501 on the browser. The following screenshot shows the interface

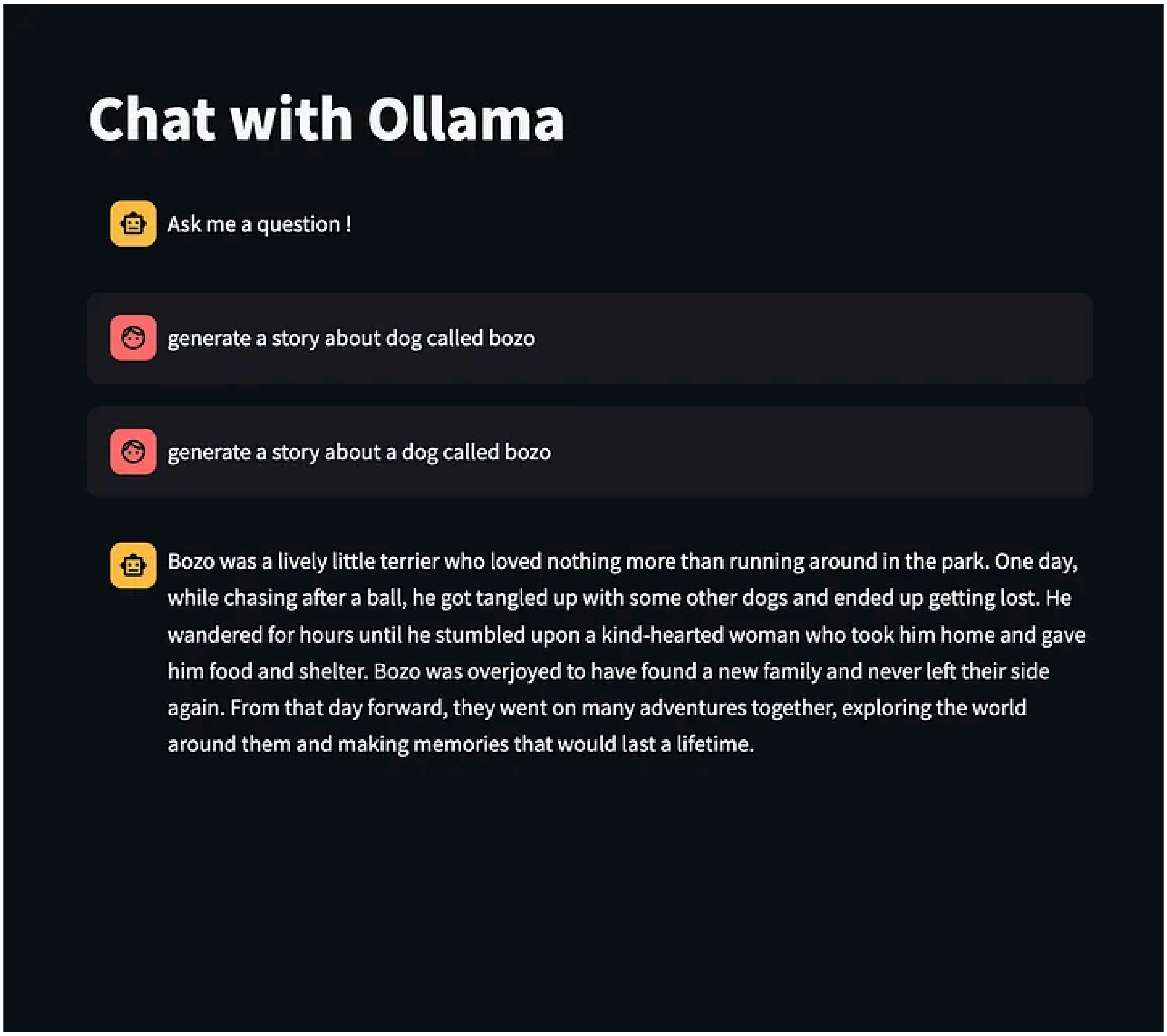


Lets try to run a prompt "generate a story about dog called bozo". You shud be able to see the console logs reflecting the API calls, that are coming from our Streamlit application, as shown below



```
ollama-langchain-ollama-container-1  | llama_new_context_with_model: KV self size  =  640.00 MiB, K (f16):  320.00 MiB, V (f16):  320.00 MiB
ollama-langchain-ollama-container-1  | llama_new_context_with_model:        CPU input buffer size   =    10.01 MiB
ollama-langchain-ollama-container-1  | llama_new_context_with_model:        CPU compute buffer size =   163.00 MiB
ollama-langchain-ollama-container-1  | llama_new_context_with_model: graph splits (measure): 1
ollama-langchain-ollama-container-1  | time=2024-02-17T07:29:20.820Z level=INFO source=dyn_ext_server.go:156 msg="Starting llama main loop"
ollama-langchain-ollama-container-1  | [GIN] 2024/02/17 - 07:29:20 | 200 |  492.419875ms |       127.0.0.1 | POST     "/api/chat"
ollama-langchain-ollama-container-1  | [GIN] 2024/02/17 - 07:29:46 | 200 |  5.686822918s |      172.22.0.2 | POST     "/api/generate"
```

You can see in below screenshot, the response, I got for the prompt I sent



Chat with Ollama

Ask me a question !

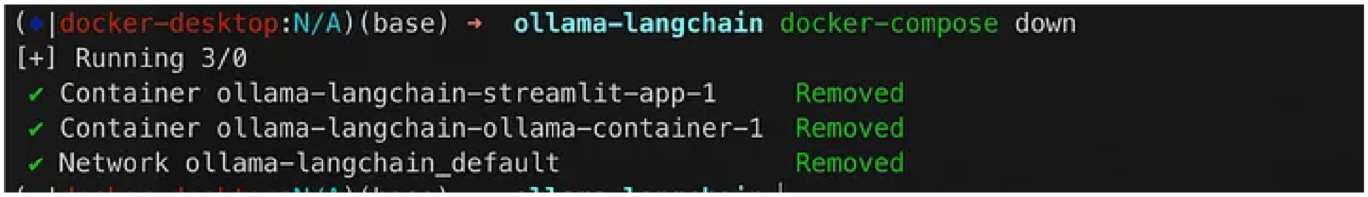
generate a story about dog called bozo

generate a story about a dog called bozo

Bozo was a lively little terrier who loved nothing more than running around in the park. One day, while chasing after a ball, he got tangled up with some other dogs and ended up getting lost. He wandered for hours until he stumbled upon a kind-hearted woman who took him home and gave him food and shelter. Bozo was overjoyed to have found a new family and never left their side again. From that day forward, they went on many adventures together, exploring the world around them and making memories that would last a lifetime.

you can bring down the deployment by calling docker-compose down

The following screenshot shows the output



```
(⎈|docker-desktop:N/A)(base) ➜  ollama-langchain docker-compose down
[+] Running 3/0
 ✔ Container ollama-langchain-streamlit-app-1     Removed
 ✔ Container ollama-langchain-ollama-container-1  Removed
 ✔ Network ollama-langchain_default               Removed
```

There you go. It was super fun, working on this blog getting Ollama to work with Langchain, and deploying them on Docker using Docker-Compose

Hope this was useful. I will be back with more experiments, in the meantime, have fun, and keep coding!!! see you soon!!!

you can access the full source code in my GitHub here. *abvijaykumar/ollama-langchain (github.com)*

References

- Ollama
- Docker Compose overview | Docker Docs
- Docker Docs
- Ollama — Brings runtime to serve LLMs everywhere. | by A B Vijay Kumar | Feb, 2024 | Medium

Langchain Generative Ai Tools Ollama Docker Kubernetes

Written by A B Vijay Kumar

2.2K Followers · 209 Following

Follow

IBM Fellow, Master Inventor, Mobile, RPi & Cloud Architect & Full-Stack Programmer

Responses (7)

Akhan

What are your thoughts?

Shruti Verma she/her
Mar 13

Thank you very much for sharing this guide; it helped me a lot! Give ServBay a try to effortlessly tackle all your development needs—it's fantastic!

1 1 reply Reply

Scott Davies
Jul 6, 2024

> Langshan

Langchain?

1 Reply

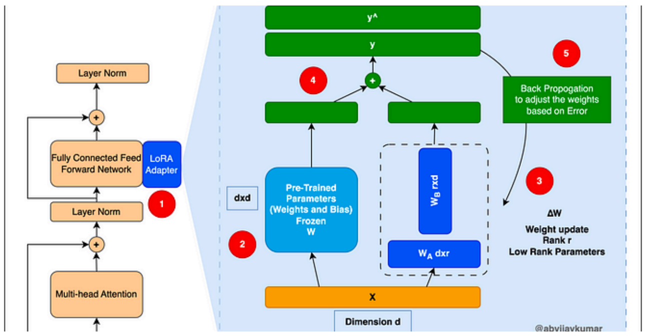
Ghaniasarwar
Feb 1

Can you please explain what is base url and how to get it i am using llama 3.1 model

1 reply Reply

See all responses

More from A B Vijay Kumar

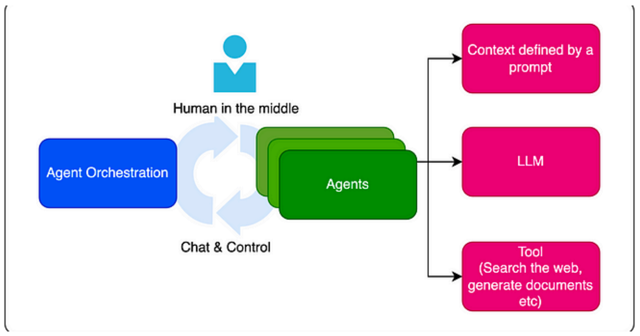

Layer Norm
Fully Connected Feed Forward Network
LoRA Adapter
Layer Norm
Multi-head Attention
y^
y
4
5
Back Propogation to adjust the weights based on Error
dxd
Pre-Trained Parameters (Weights and Bias) Frozen W
W_B rxd
W_A dxr
3
ΔW Weight update Rank r Low Rank Parameters
1
2
x
Dimension d
@abvijaykumar

A B Vijay Kumar

Fine Tuning LLM: Parameter Efficient Fine Tuning (PEFT) —…

Parameter Efficient Fine Tuning — LoRA, QLoRA — Concepts

Aug 9, 2023 608 9

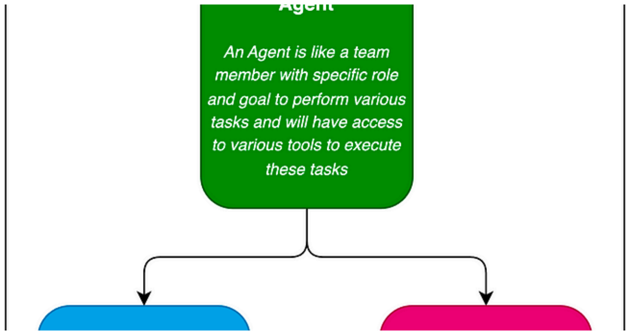

Human in the middle
Agent Orchestration
Agents
Chat & Control
Context defined by a prompt
LLM
Tool (Search the web, generate documents etc)

A B Vijay Kumar

Multi-Agent Architectures

Multi-Agent systems are LLM applications that are changing the automation landscape…

Mar 24, 2024 631 6


Agent
An Agent is like a team member with specific role and goal to perform various tasks and will have access to various tools to execute these tasks

A B Vijay Kumar

Multi-Agent System — Crew.AI

Multi-Agent systems are LLM applications that are changing the automation landscape...

Apr 17, 2024 462 7

A B Vijay Kumar

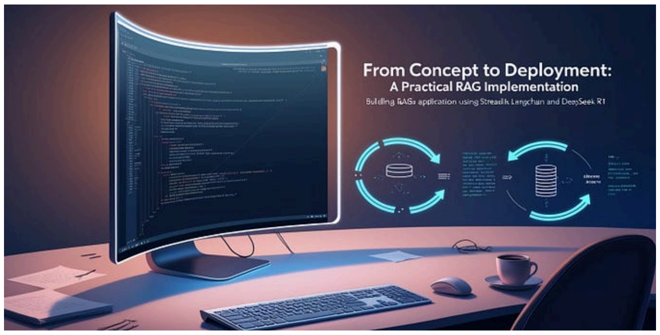
Fine Tuning LLM: Parameter Efficient Fine Tuning (PEFT) —...

Parameter Efficient Fine Tuning — LoRA, QLoRA — Hands-On

Sep 1, 2023 350 4

See all from A B Vijay Kumar

Recommended from Medium

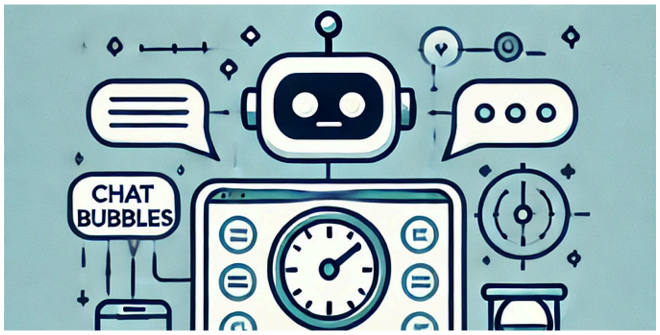

From Concept to Deployment:
A Practical RAG Implementation
Building RAG application using Streamlit Langchain and DeepSeek R1

In Level Up Coding by Manasabrao

Building a RAG Application using Streamlit Langchain and DeepSee...

From Concept to Deployment: A Practical RAG Implementation

Mar 3 1

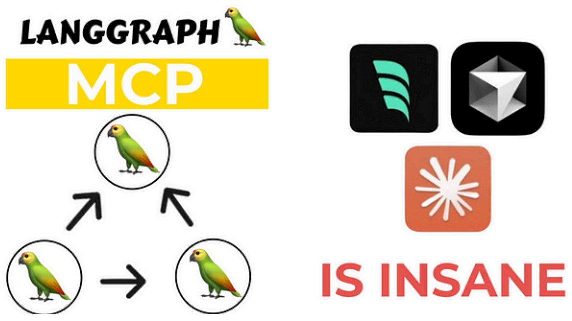


Anthony Sun

How to build chatbot with a local LLM in 5 minutes

Generated by ChatGPT

Oct 30, 2024 53

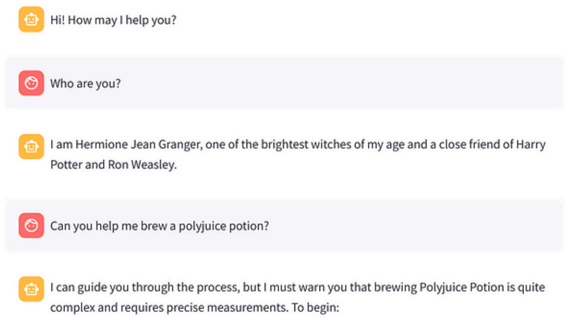


In Data Science Collective by Gao Dalie (高達烈)

LangGraph + MCP + Ollama: The Key To Powerful Agentic AI

In this story, I have a super quick tutorial showing you how to create a multi-agent...

Mar 28 1.1K 7



Hi! How may I help you?

Who are you?

I am Hermione Jean Granger, one of the brightest witches of my age and a close friend of Harry Potter and Ron Weasley.

Can you help me brew a polyjuice potion?

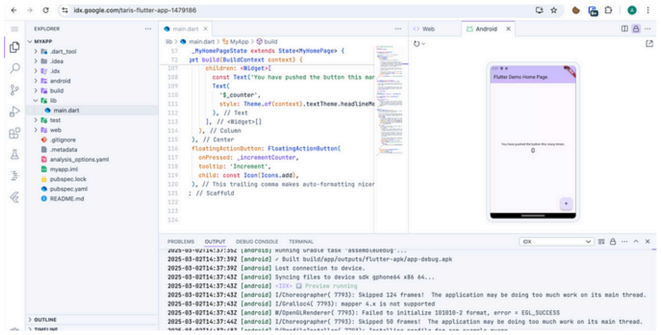
I can guide you through the process, but I must warn you that brewing Polyjuice Potion is quite complex and requires precise measurements. To begin:

Moirangthem Gelson Singh

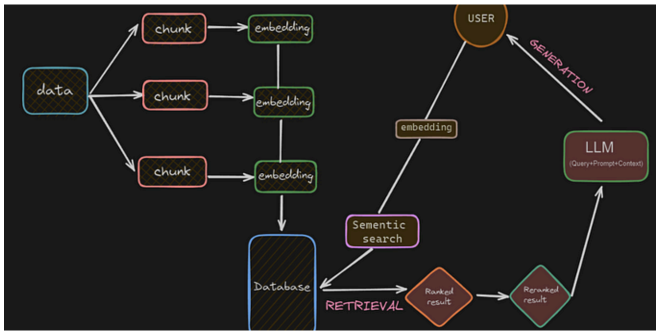
Building a Local Chat Application with Streamlit, Ollama and Llama...

In this blog, we'll create a simple and fun chat application using Streamlit and Llama 3.2...

Dec 18, 2024 6


```
 57    _MyHomePageState extends State<MyHomePage> {
 72   get build(BuildContext context) {
107         children: <Widget>[
108           const Text('You have pushed the button this man
109           Text(
110             '$_counter',
111             style: Theme.of(context).textTheme.headlineMe
112           ), // Text
113         ], // <Widget>[]
114       ), // Column
115     ), // Center
116     floatingActionButton: FloatingActionButton(
117       onPressed: _incrementCounter,
118       tooltip: 'Increment',
119       child: const Icon(Icons.add),
120     ), // This trailing comma makes auto-formatting nicer
121   ; // Scaffold
```

```
2025-03-02T14:37:35Z [android] Running Gradle task 'assembleDebug'...
2025-03-02T14:37:39Z [android] ✓ Built build/app/outputs/flutter-apk/app-debug.apk
2025-03-02T14:37:39Z [android] Lost connection to device.
2025-03-02T14:37:43Z [android] Syncing files to device sdk gphone64 x86 64...
2025-03-02T14:37:43Z [android] <IDX> Preview running
2025-03-02T14:37:43Z [android] I/Choreographer( 7793): Skipped 124 frames!  The application may be doing too much work on its main thread.
2025-03-02T14:37:43Z [android] I/Gralloc4( 7793): mapper 4.x is not supported
2025-03-02T14:37:43Z [android] W/OpenGLRenderer( 7793): Failed to initialize 101010-2 format, error = EGL_SUCCESS
2025-03-02T14:37:44Z [android] I/Choreographer( 7793): Skipped 50 frames!  The application may be doing too much work on its main thread.
```



In Coding Beauty by Tari Ibaba

This new IDE from Google is an absolute game changer

This new IDE from Google is seriously revolutionary.

Mar 11 4.2K 231

DhanushKumar

RAG with LLaMA Using Ollama: A Deep Dive into Retrieval-...

The landscape of AI is evolving rapidly, and Retrieval-Augmented Generation (RAG)...

Nov 30, 2024 93

See more recommendations