Technical Architecture for Text Generation Using Local LLM and AWS Deployment via Docker

# Overview

This document outlines the technical architecture and steps for deploying a local LLM-based text generation system on an AWS VM using Docker. The system allows text generation through a locally hosted language model and can be accessed via an API hosted by **FastAPI**.

# Key Components:

1. **Text Generation with Local LLM**: A large language model (LLM) that performs text generation tasks based on input prompts.
2. **FastAPI**: A modern web framework for building APIs that will interact with the LLM.
3. **Docker**: Used to containerize the entire setup for easy deployment and portability.
4. **AWS VM (EC2)**: A virtual machine hosted on AWS where the Docker container will be deployed.
5. **Uvicorn**: An ASGI server for running FastAPI.

# System Design

1. **Text Generation Flow**:
   * **Input Prompt**: The user sends an input prompt via a POST request to a FastAPI endpoint.
   * **LLM Integration**: The FastAPI app communicates with the locally deployed LLM to process the prompt and generate text.
   * **Response**: The generated text is sent back to the user as a response to the API call.
2. **Deployment Architecture**:
   * **Docker** will containerize the entire application, including FastAPI, LLM, and all dependencies.
   * The container will be deployed on an **AWS EC2 instance** with a public IP, allowing users to access the API over the internet.
   * **Uvicorn** will serve the FastAPI app, listening on port 8000.

# Detailed Architecture

**1. Local LLM Setup**

* The LLM (e.g., GPT-3 or another local model) will be set up on the AWS EC2 instance. The model is pre-trained and capable of understanding the input prompt, processing it, and generating meaningful text.
* **Frameworks Used**:
  + Transformers (by HuggingFace) or a custom model.
  + PyTorch or TensorFlow for model inference.

**2. FastAPI Setup**

* **FastAPI** is used to create an HTTP API that accepts prompts from users and returns generated text.
* **Endpoint**: /generate-story accepts a POST request with a JSON body containing the prompt and optional max\_tokens to limit the generated text's length.

Example Request Body:

{

"prompt": "Tell me a story about AI",

"max\_tokens": 3000

}

**3. Dockerization**

* **Dockerfile**:
  + A Dockerfile will be created to build a Docker image that contains all the necessary dependencies, including FastAPI, Uvicorn, and the LLM model.
  + The Docker image will allow the app to run seamlessly in any environment, from local machines to cloud servers like AWS.

Example Dockerfile:

Example Dockerfile:

FROM python:3.9-slim

# Set the working directory

WORKDIR /app

# Copy the requirements file

COPY requirements.txt /app/

# Install dependencies

RUN pip install --no-cache-dir -r requirements.txt

# Copy the application code

COPY . /app/

# Expose port 8000

EXPOSE 8000

# Start the FastAPI app using Uvicorn

CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]

**4. AWS EC2 Setup**

* **EC2 Instance**:
  + A **t3.micro** instance can be used for light loads, but scaling may be necessary depending on usage.
  + The instance is set up with a public IP (16.171.197.158) and running Ubuntu.

Steps:

* + **Launch EC2 instance**: Use the AWS Console or CLI to create a new instance.
  + **Connect to EC2**: Use SSH to access the instance from your local machine.
  + **Install Docker**: On the EC2 instance, install Docker if not already installed:

sudo apt-get update

sudo apt-get install docker.io

sudo systemctl start docker

sudo systemctl enable docker

* + **Build and Run Docker Container**:

docker build -t local-llm-app .

docker run -d -p 8000:8000 local-llm-app

**5. Testing and Access**

* Once deployed, the FastAPI app will be accessible at the instance's public IP on port 8000.  
  Example: http://16.171.197.158:8000/generate-story

**6. Security Considerations**

* **AWS Security Groups**:
  + Ensure the security group for your EC2 instance allows inbound traffic on port 8000.
  + Limit access to only trusted IP addresses or use a load balancer for more advanced setups.
* **Environment Variables**: Store sensitive keys such as API keys or model parameters securely using .env files and dotenv.

**7. Scaling**

* **Horizontal Scaling**: To handle more traffic, the application can be replicated across multiple EC2 instances behind a load balancer.
* **Container Orchestration**: Consider using **Amazon ECS** or **EKS** for easier scaling and management of Docker containers.

**Dockerfile**

FROM python:3.9-slim

# Set the working directory

WORKDIR /app

# Copy the requirements file

COPY requirements.txt /app/

# Install dependencies

RUN pip install --no-cache-dir -r requirements.txt

# Copy the application code

COPY . /app/

# Expose port 8000

EXPOSE 8000

# Start the FastAPI app using Uvicorn

CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]

**Requirements.txt**

fastapi==0.68.0

uvicorn==0.15.0

torch==1.9.0

transformers==4.10.0

**Deployment Steps**

1. **Build the Docker Image**:

docker build -t local-llm-app .

1. **Run the Docker Container**:

docker run -d -p 8000:8000 local-llm-app

1. **Access the FastAPI App**:

* Open the browser or use curl to test the /generate-story endpoint.