Table of Contents

[1. Introduction 2](#_Toc202798433)

[2. Scope of Work 2](#_Toc202798434)

[2.1 Goals 2](#_Toc202798435)

[2.2 Out of Scope 2](#_Toc202798436)

[3. Architectural Options 2](#_Toc202798437)

[3.1 Option A: Monolithic Container 2](#_Toc202798438)

[3.2 Option B: Microservice Containers 3](#_Toc202798439)

[4. Prerequisites 3](#_Toc202798440)

[4.1 Hardware 3](#_Toc202798441)

[4.2 Software 4](#_Toc202798442)

[4.3 Networking & Accounts 4](#_Toc202798443)

[4.4 Developer Tools 4](#_Toc202798444)

[5. Recommended Approach & Justification 4](#_Toc202798445)

[6. Learning Roadmap 4](#_Toc202798446)

[7. Component Learning & Exploration 5](#_Toc202798447)

[8. Milestones & Timeline (4 Weeks) 5](#_Toc202798448)

[9. Requirements & Risks 5](#_Toc202798449)

[10.System Flow 6](#_Toc202798450)

[11. Next Steps 6](#_Toc202798451)

Scoping Document: vLLM + SLM/LLM Docker Infrastructure Stack

# 1. Introduction

This document outlines the scope, architecture options, learning roadmap, milestones, and timelines for the implementation of a **containerized inference platform** hosting multiple small- and large-scale language models (SLMs + LLMs) via vLLM, exposed through an OpenAI-compatible API.

Your goal is to:

* Understand core concepts of vLLM, Docker orchestration, and API layering.
* Prototype and deploy an end-to-end solution within 4 weeks.
* Demonstrate learning and apply best practices for future solutions.

# 2. Scope of Work

## 2.1 Goals

1. **Containerize vLLM runtime** with support for multiple models (SLMs & LLMs).
2. **Expose an OpenAI-compatible API** (Chat Completions, Embeddings) via FastAPI.
3. **Implement monitoring** (Prometheus) and GPU utilization metrics.
4. **Ensure extendibility**: easy onboarding of new models and scaling.

## 2.2 Out of Scope

* Training or fine-tuning models.
* Advanced multi-node GPU clustering (single-node initially).
* Production hardened security (P0: basic auth, later iterations handle encryption, RBAC).

# 3. Architectural Options

## 3.1 Option A: Monolithic Container

* **Single Docker image** bundles vLLM runtime, models, FastAPI server, and Prometheus exporter.
* **Pros**: Simple, quick to prototype.
* **Cons**: Harder to scale parts independently, larger images, slower CI/CD.

A diagram of a business

AI-generated content may be incorrect.

## 3.2 Option B: Microservice Containers

* **vLLM Worker**: loads models, exposes gRPC or REST endpoints.
* **API Gateway**: FastAPI service implements OpenAI spec, routes requests to vLLM workers.
* A diagram of a work flow

  AI-generated content may be incorrect.**Monitoring Service**: standalone Prometheus exporter container.

# 4. Prerequisites

Here are the **Prerequisites** you’ll need before kicking off your containerized LLM inference project:

## 4.1 Hardware

* **GPU**: NVIDIA‑based with **≥16 GB VRAM** (e.g. RTX 3090/4090, A10G, A100)
* **System RAM**: ≥32 GB
* **Disk**: ≥100 GB SSD (models can be tens of GBs each)

## 4.2 Software

* **Operating System**: Linux (Ubuntu 20.04+ recommended) or Windows/Mac with WSL2
* **Docker Desktop** (or Docker Engine)
* **NVIDIA Container Toolkit** (for --gpus support in Docker)
* **CUDA Toolkit** matching your GPU driver (e.g. CUDA 12.x)
* **Python 3.10+** and **pip**
* **Git** (to clone repos, fetch examples)

## 4.3 Networking & Accounts

* **Internet access** (to download model weights from Hugging Face once)
* **Hugging Face account** (if models require authentication tokens)
* (Optional) **Private registry** or **Docker Hub** credentials if you’ll push images

## 4.4 Developer Tools

* **VS Code** or your favorite IDE/editor
* **Postman** or **curl** (for API testing)
* **Prometheus + Grafana** (for metrics and dashboards)

# 5. Recommended Approach & Justification

We recommend **Option B (Microservice Containers)** because:

1. **Separation of Concerns**: Independently develop, test, and scale inference vs. API logic.
2. **Faster Iteration**: Smaller images mean quicker rebuilds when tweaking one component.
3. **Extendibility**: Future support for multi-node and GPU autoscaling.

# 6. Learning Roadmap

|  |  |  |
| --- | --- | --- |
| **Topic** | **Resources** | **Time Estimate** |
| vLLM Fundamentals | Official vLLM docs, sample notebooks | 2 days |
| Docker & Docker Compose | Docker official tutorial, Compose guide | 2 days |
| FastAPI & OpenAI spec | FastAPI docs, OpenAI API reference | 2 days |
| Prometheus instrumentation | Prometheus Python client docs, examples | 1 day |
| GPU management with NVIDIA | NVIDIA Docker, Nvidia-Smi | 1 day |
| CI/CD for Docker images | GitHub Actions or GitLab CI fundamentals | 2 days |
| Integration & Testing | Pytest, Postman or HTTP for API testing | 1 day |

# 7. Component Learning & Exploration

To implement and understand the architecture effectively, the following hands-on learning steps are recommended for each core component:

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Docs Link** | **Learning Activity** | **CLI/Code Starter** |
| **vLLM** | [GitHub](https://github.com/vllm-project/vllm) | Run basic inference | python3 -m vllm.entrypoints.openai.api\_server --model mistralai/Mistral-7B-Instruct-v0.1 |
| **FastAPI** | [Docs](https://fastapi.tiangolo.com/) | Build /v1/chat/completions | pip install fastapi uvicorn + Hello World |
| **Docker** | [Docker Docs](https://docs.docker.com/) | Create and run containers | docker build -t test . && docker run test |
| **Prometheus** | [Prometheus Docs](https://prometheus.io/docs/introduction/overview/) | Expose and scrape metrics | Add /metrics via prometheus\_client |
| **NVIDIA GPU** | [NVIDIA Toolkit](https://docs.nvidia.com/datacenter/cloud-native/) | Set up GPU containers | nvidia-smi, docker run --gpus all |

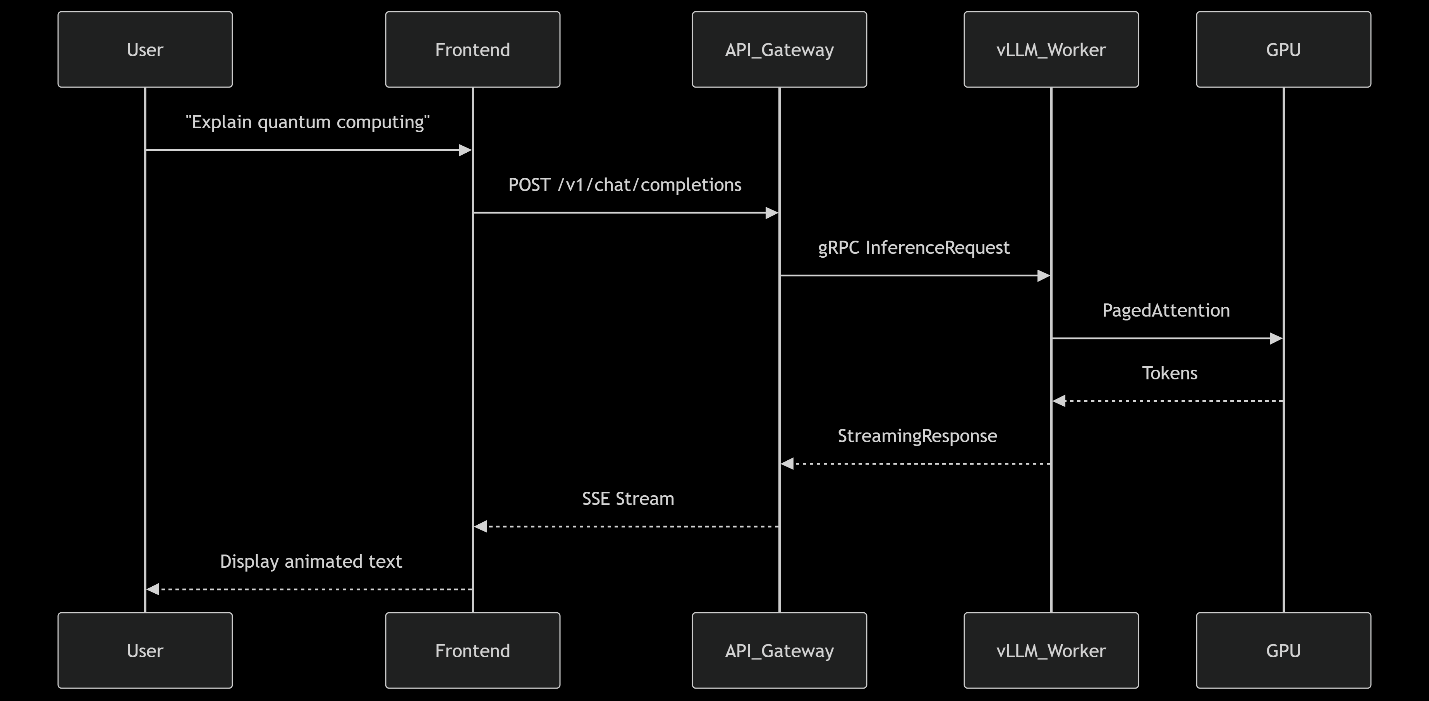
# 8. Milestones & Timeline (4 Weeks)

1. **Week 1: Research & Prototype**
   * Setup vLLM in a local Docker container.
   * Load one small model (e.g., GPT-2) and verify inference.
2. **Week 2: API Integration**
   * Scaffold FastAPI service exposing completion endpoint.
   * Integrate with vLLM worker via REST/gRPC.
3. **Week 3: Monitoring & Metrics**
   * Instrument vLLM and FastAPI with Prometheus exporters.
   * Validate GPU utilization dashboards.
4. **Week 4: Testing & Documentation**
   * End-to-end tests for the OpenAI-compatible API.
   * Write README, deployment scripts, and architecture diagram.

# 9. Requirements & Risks

* **Hardware**: Single GPU-enabled server (e.g., NVIDIA T4 or better).
* **Software**: Docker >=20.x, Python >=3.10, CUDA toolkit matching GPU.
* **Risks**:
  + **Model loading latency**: mitigate via warm-up or pre-loading.
  + **GPU memory constraints**: start with smaller SLMs; monitor usage.
  + **API spec drift**: adhere strictly to OpenAI definitions.

# 10.System Flow



# 11. Next Steps

1. Review this document with your lead and align on milestones.
2. Kick off Week 1 research and share progress by the end of Day 5.
3. Schedule a mid-point demo at the end of Week 2.

*Prepared by: [Hamza Sanaullah]*

*Date: July 7, 2025*