Rahul Deshmukh – G01521128
Rudra Bedekar –G01517024
Shatakshi Rathore - G01545382

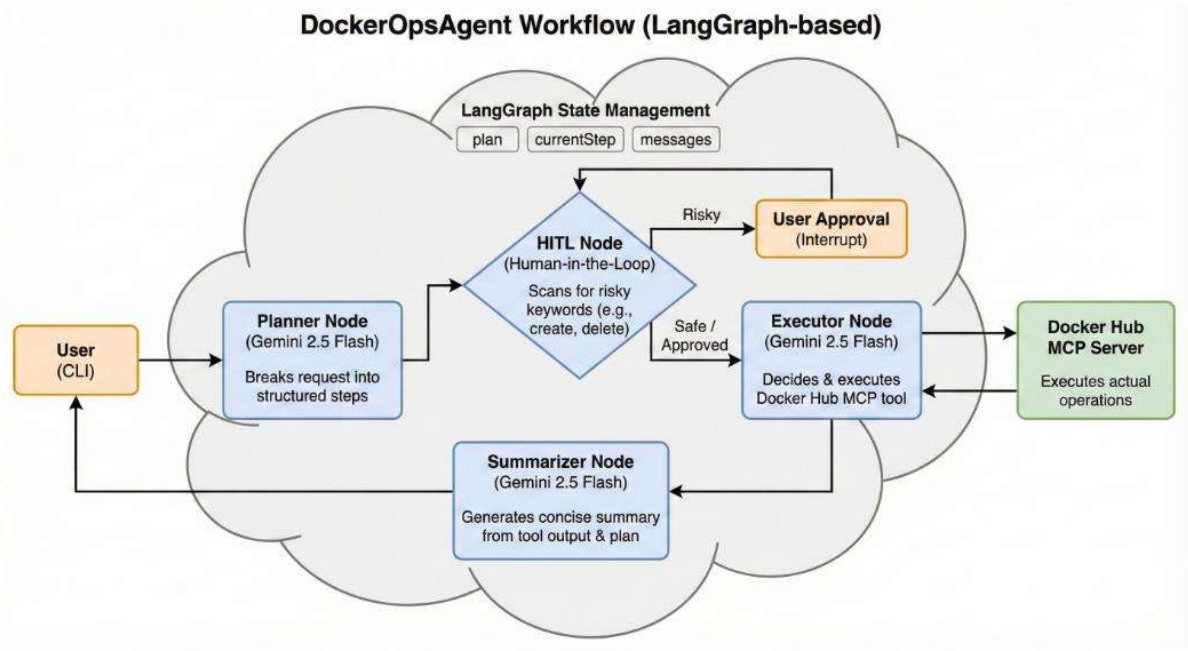
System Overview

DockerOpsAgent is a conversational assistant that uses LangGraph to handle Docker Hub tasks without requiring custom CLI scripts or manual UI navigation. When the user types a natural-language command into the terminal, the agent turns it into an ordered sequence of Docker operations, executes those operations through MCP tools running inside a Docker container, and returns a concise, human-readable result. Planning is handled by Gemini, and each node is orchestrated by LangGraph, producing a predictable and well-structured workflow for tasks like listing repositories, inspecting images, and checking usage metadata.

Every state-changing Docker Hub operation is protected by an integrated safety layer. Before any change is performed, the agent pauses execution and initiates a human-in-the-loop approval phase whenever a request involves creating, modifying, or deleting a repository. This ensures that high-impact actions are explicitly reviewed and confirmed, while normal read-only queries remain fully autonomous and fast.

System Architecture Explanation

The LLM-powered planner (Gemini) transforms commands sent by the user via the CLI into an ordered plan of Docker steps. The HITL node intercepts the workflow and waits for user approval if the plan includes any potentially dangerous action. Once approved, the executor uses Docker Hub MCP tools inside the Docker container to carry out the requested operations. The summarizer, also backed by Gemini, converts raw tool results into a clean, user-friendly response for the CLI. LangGraph maintains the agent's state across nodes, enabling pause-and-resume behavior and reliable multi-step execution.



1. LLM (Gemini 2.5)

Understanding user requests from the CLI and transforming them into a structured Docker process is the responsibility of the LLM. It generates an orderly plan, such as check repository, verify namespace, create repository, and establish visibility, in contrast to a standard chatbot. As a result, the model operates more like a planner than a conversational agent and guarantees that all actions on Docker Hub adhere to a precise, predictable order rather than vague directions.

2. Tools

The agent uses tools to carry out genuine tasks. Operating within a Docker container, the Docker Hub MCP server offers authorized features such as repository listing, tag inspection, and tag creation. The agent never makes direct connections to Docker Hub; instead, it transmits commands via the MCP tool layer, which manages execution, security, and API logic. As a result, the model only plans and never directly interacts with credentials, keeping Docker activities safe, contained, and technically isolated from the LLM.

3. Memory

LangGraph manages the agent's memory instead of a vector database. The system saves the current step and execution state each time the workflow passes through a node (planner, HITL, executor, or summarizer). Because this state is preserved, the user can

resume later without losing context if the workflow pauses for approval. This removes the need for long-term semantic storage and makes the agent feel consistent and dependable. The only memory retained is operational state, which is precisely what a tool-driven agent needs.

4. HITL (Human-in-the-Loop)

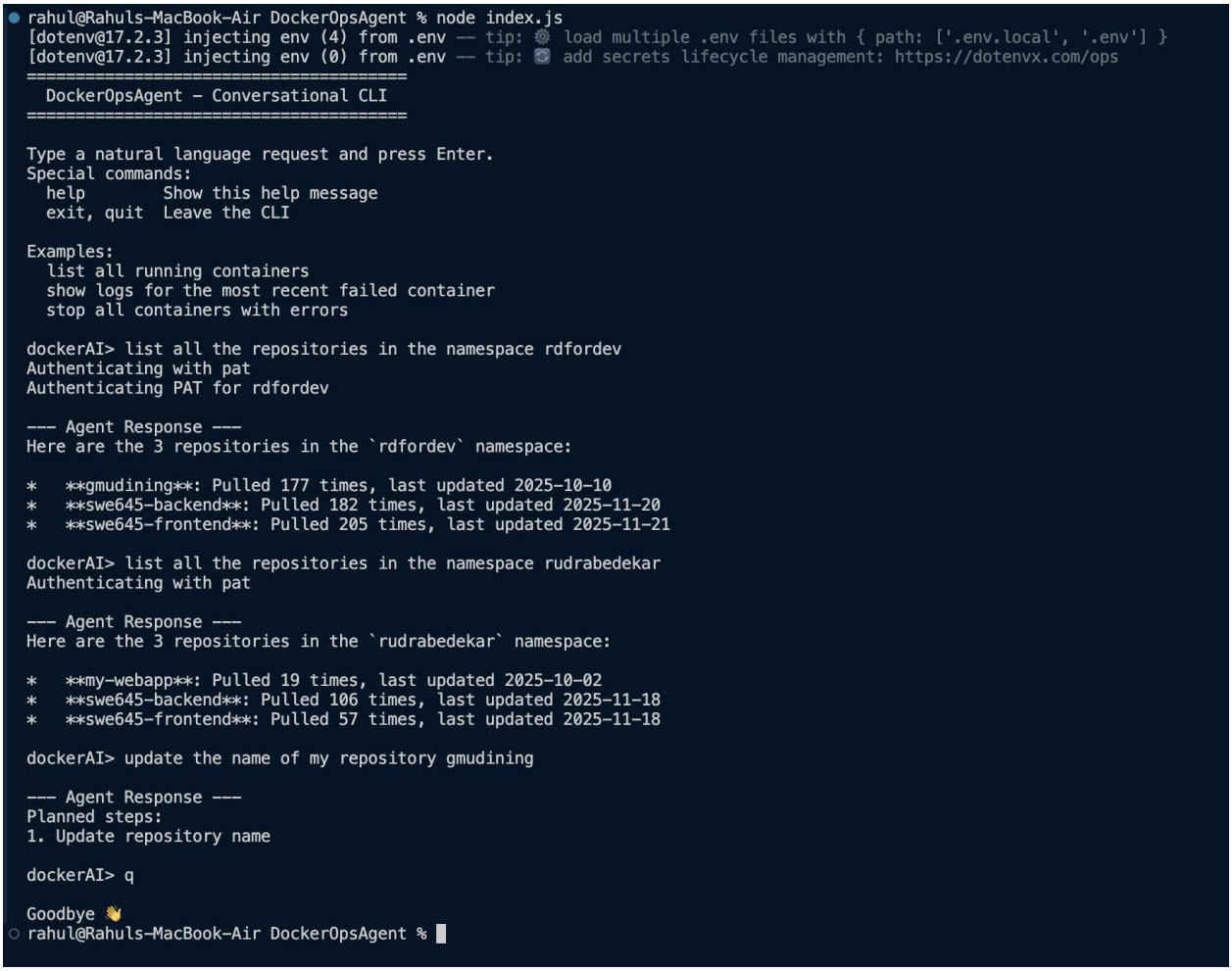
For commands that are harmful or alter state, the HITL node serves as a safety gate. When the plan includes actions such as create, update, delete, or change visibility, the agent automatically halts execution and asks the user to approve or reject the operation. Read-only actions (like listing repositories or viewing metadata) run immediately, but any operation that can modify Docker Hub requires explicit human confirmation. As a result, the agent stays highly autonomous for safe queries, while ensuring that no Docker changes occur without the user's consent.

Description of the Agent's Use Case and Goals

Instead of using manual dashboard clicks or Docker CLI instructions, this agent is made to handle Docker Hub chores using natural language. Simple commands, such as listing repositories or verifying tags, can be typed by the user, and the Docker Hub MCP tools operating inside a container take care of the planning and execution. The agent completes the workflow independently and returns a brief result to the terminal for regular tasks like checking repository details, pull counts, or picture metadata.

Making Docker Hub operations simpler, safer, and more organized without sacrificing user control is the main objective. The agent requires human consent for state-changing tasks, such as creating or modifying repositories, but it is autonomous for read-only actions. By avoiding unintentional modifications, this balance enables the system to minimize repetitive tasks and execution overhead. By integrating conversational input, organized planning, regulated tool execution, and an integrated approval layer for sensitive tasks, the agent streamlines daily Docker Hub management.

Example run or transcript



Due to its access to the logged-in user namespace, the initial listing necessitated authentication. Because Docker Hub permits anonymous read access for public images and the target namespace contains publicly accessible repositories, the second listing did not require credentials.

Challenges Faced and Lessons Learned

Creating a trustworthy safety checkpoint for creating and updating activities was one of the challenges. Careful placement of the HITL node and consistent verb classification were necessary for the agent to identify dangerous actions and halt execution without disrupting the workflow. Running the MCP server in a Docker container presented another challenge because the tools were inaccessible whenever Docker was not in use. It also required fine-tuning to ensure seamless planner-to-executor communication, particularly when the LLM periodically generated extra steps or incomplete tool calls.

The main takeaway from this was that, particularly when controlling real infrastructure, an agent must behave predictably rather than imaginatively. Because the agent simply needed

to remember where it halted rather than retain long-term histories, keeping memory lightweight with LangGraph state monitoring was adequate. The HITL gate demonstrated that a modest level of human monitoring may preserve safety without losing automation, while the clear distinction between the planning, approval, and execution nodes helped prevent unintentional changes and made the system easier to debug.

Github Link - https://github.com/Rahul-1611/DockerOpsAgent

Recording - https://gmuedu-my.sharepoint.com/:v:/g/personal/rdeshmu_gmu_edu/IQDakrq8lYZhQYTcVQVJW5eyAQ3nGrbaKq5II4IupgZPyuI?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJTdHJlYW1XZWJBcHAiLCJyZWZlcnJhbFZpZXciOiJTaGFyZURpYWxvZy1MaW5rIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXcifX0%3D&e=kWMpp2