Project: LLM based Multi-Agents for WeebApp Deployment

Summary: This project is built as a multi-agent orchestration framework using LangChain. There is one Main Agent and two Specialized Agents (MySQL Agent and Web Server Agent). The specialized agents use the LLM to do some reasoning and select the right tool for the given task. In particlar, the main oprchestrator agent:

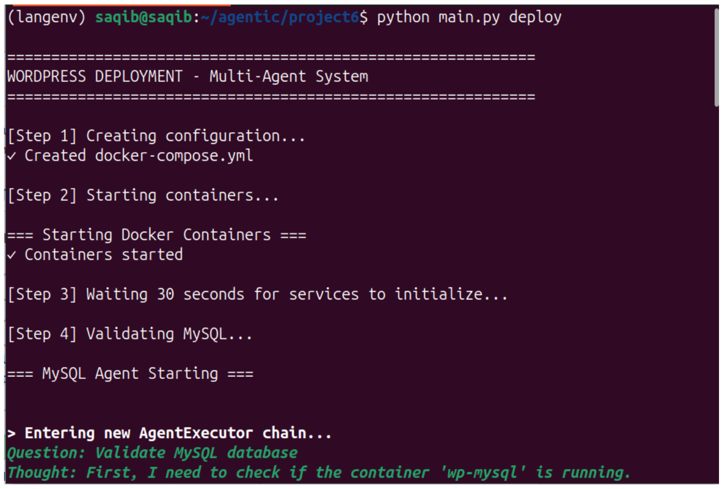
- Creates a docker-compose file for WordPress + MySQL from config.yaml.
- Execute the docker-compose file to launch the application using coontainers
- Further uses two LLM based specialized agents: MySQLAgent and WebServerAgent to validate services using some tools.
  - Each specefialized agent workflow begin with **task** (receive defiend goal).
  - Task is provided to LLM throught **promp**t, explaining available tools, defining reasoning format and insturct the model step by step.
  - LLM interpret the task, perofmr **reasoning**, and decide which tool to use.
  - The output of these tools is returned to the LLM as an **observation**, which it uses to reflect and plan the next step
  - This forms the **ReAct loop**, a structured Reason and Act cycle, where the LLM continually reasons, acts, and observes until it determines the task is complete. Finally, the LLM produces a final answer.

Files in the project (as provided):

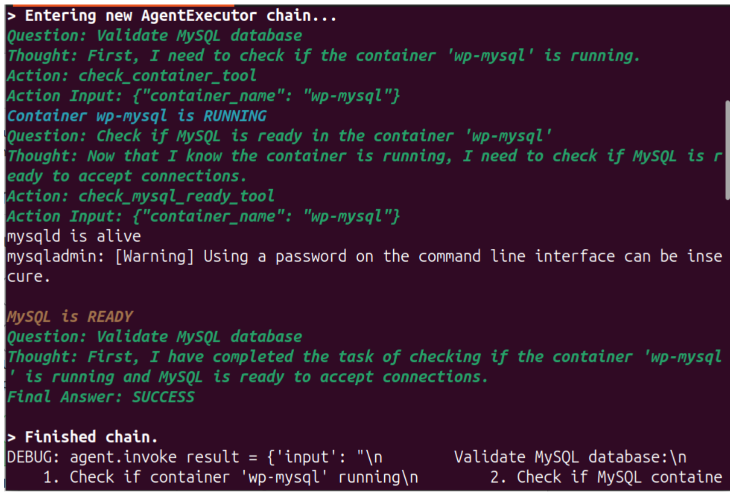
- main.py — CLI entry: deploy / stop.
- config.yaml — LLM and service settings.
- tools.py — helper utilities executed by agents.
- orchestrator.py — deployment workflow + agent orchestration.
- mysql_agent.py, webserver_agent.py — agent implementations.

Experimentation Overview: I have used Linux (Ubuntu) with Docker and Ollama (running the LLMs locally). The LLM I chose is Llama 3.1 because it provides strong reasoning ability for multi-step tasks, which is essential for agent-based decision-making and tool selection. Since the model runs locally, the inference time for even a simple query is around 70–80 seconds, which significantly slows down the troubleshooting process. I initially generated a basic sketch of the code using AI and then manually troubleshot the entire code, customizing it according to my requirements. The current version of the code is intentionally simple, a starter framework for further development. At this stage, my main focus was to ensure the basic flow works: initializing the agent, assigning the task, and enabling LLM-based reasoning to select the correct tools for validation. The validation steps are also kept minimal, such as checking whether the container is running and whether the service is accessible. The results shown below capture the core insights from the experiment and demonstrate that the fundamental logic is working correctly

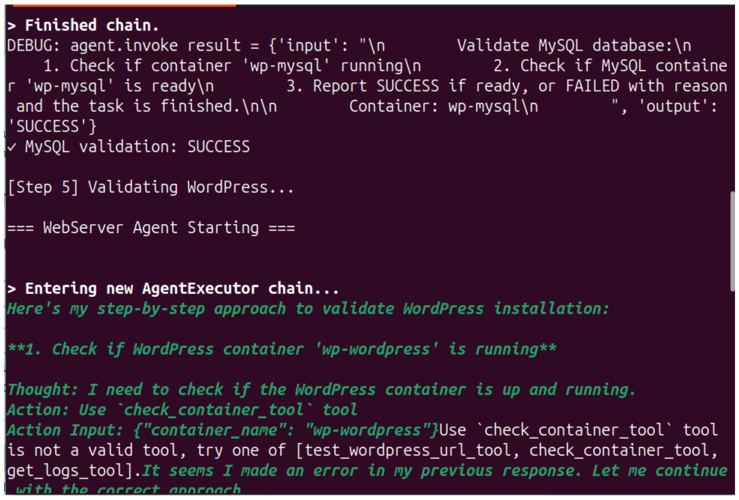
Obtained Results:



```
(langenv) saqib@saqib:~/agentic/project6$ python main.py deploy

============================================================
WORDPRESS DEPLOYMENT - Multi-Agent System
============================================================

[Step 1] Creating configuration...
✓ Created docker-compose.yml

[Step 2] Starting containers...

=== Starting Docker Containers ===
✓ Containers started

[Step 3] Waiting 30 seconds for services to initialize...

[Step 4] Validating MySQL...

=== MySQL Agent Starting ===


> Entering new AgentExecutor chain...
Question: Validate MySQL database
Thought: First, I need to check if the container 'wp-mysql' is running.
```

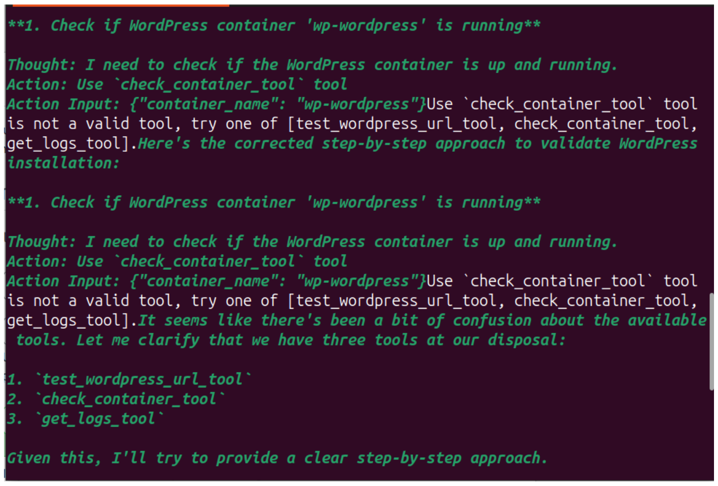


```
> Entering new AgentExecutor chain...
Question: Validate MySQL database
Thought: First, I need to check if the container 'wp-mysql' is running.
Action: check_container_tool
Action Input: {"container_name": "wp-mysql"}
Container wp-mysql is RUNNING
Question: Check if MySQL is ready in the container 'wp-mysql'
Thought: Now that I know the container is running, I need to check if MySQL is r
eady to accept connections.
Action: check_mysql_ready_tool
Action Input: {"container_name": "wp-mysql"}
mysqld is alive
mysqladmin: [Warning] Using a password on the command line interface can be inse
cure.

MySQL is READY
Question: Validate MySQL database
Thought: First, I have completed the task of checking if the container 'wp-mysql
' is running and MySQL is ready to accept connections.
Final Answer: SUCCESS

> Finished chain.
DEBUG: agent.invoke result = {'input': "\n        Validate MySQL database:\n
    1. Check if container 'wp-mysql' running\n        2. Check if MySQL containe
```

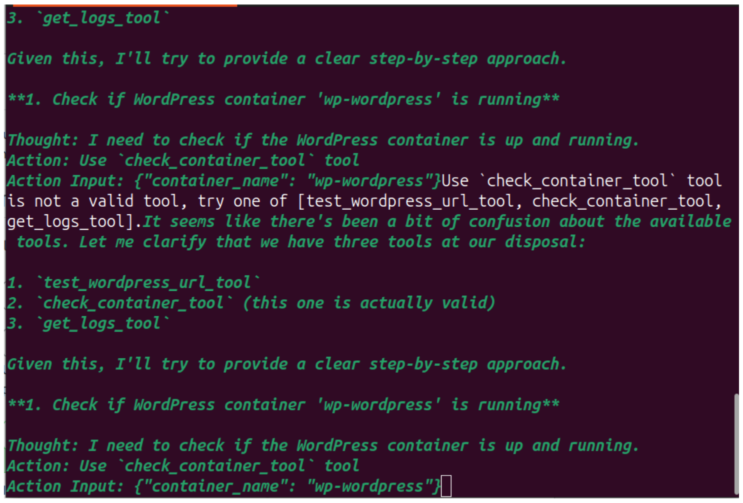


```
> Finished chain.
DEBUG: agent.invoke result = {'input': "\n        Validate MySQL database:\n
    1. Check if container 'wp-mysql' running\n        2. Check if MySQL containe
r 'wp-mysql' is ready\n        3. Report SUCCESS if ready, or FAILED with reason
 and the task is finished.\n\n        Container: wp-mysql\n        ", 'output':
'SUCCESS'}
✓ MySQL validation: SUCCESS

[Step 5] Validating WordPress...

=== WebServer Agent Starting ===


> Entering new AgentExecutor chain...
Here's my step-by-step approach to validate WordPress installation:

**1. Check if WordPress container 'wp-wordpress' is running**

Thought: I need to check if the WordPress container is up and running.
Action: Use `check_container_tool` tool
Action Input: {"container_name": "wp-wordpress"}Use `check_container_tool` tool
is not a valid tool, try one of [test_wordpress_url_tool, check_container_tool,
get_logs_tool].It seems I made an error in my previous response. Let me continue
```

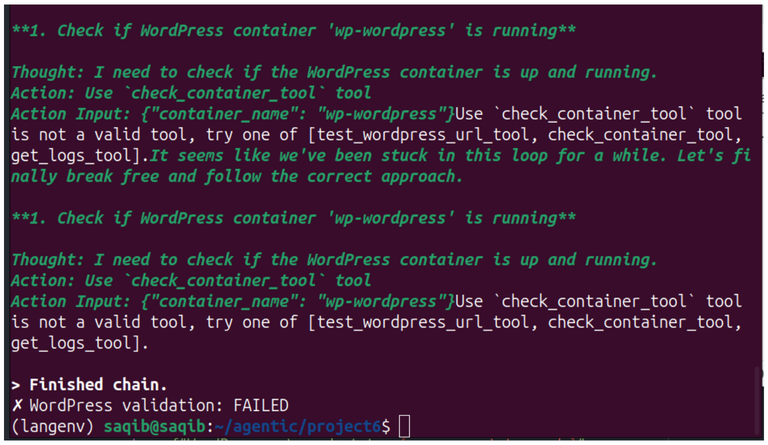
Here the first chain (MySQL) properly finished, and the code entered to new chain for WordPress deployment:



```
**1. Check if WordPress container 'wp-wordpress' is running**

Thought: I need to check if the WordPress container is up and running.
Action: Use `check_container_tool` tool
Action Input: {"container_name": "wp-wordpress"}Use `check_container_tool` tool
is not a valid tool, try one of [test_wordpress_url_tool, check_container_tool,
get_logs_tool].Here's the corrected step-by-step approach to validate WordPress
installation:

**1. Check if WordPress container 'wp-wordpress' is running**

Thought: I need to check if the WordPress container is up and running.
Action: Use `check_container_tool` tool
Action Input: {"container_name": "wp-wordpress"}Use `check_container_tool` tool
is not a valid tool, try one of [test_wordpress_url_tool, check_container_tool,
get_logs_tool].It seems like there's been a bit of confusion about the available
 tools. Let me clarify that we have three tools at our disposal:

1. `test_wordpress_url_tool`
2. `check_container_tool`
3. `get_logs_tool`

Given this, I'll try to provide a clear step-by-step approach.
```

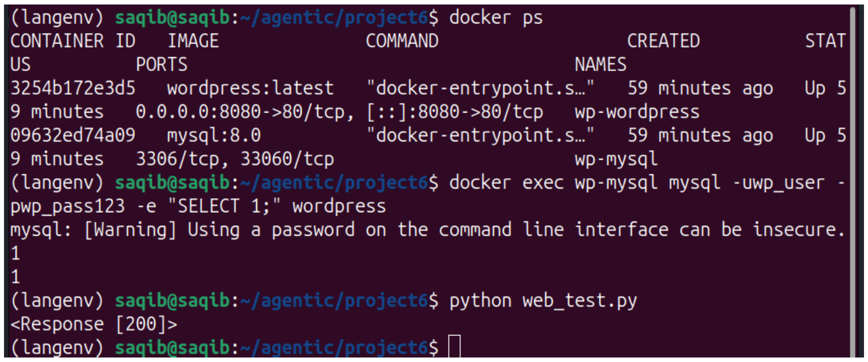


```
3. `get_logs_tool`

Given this, I'll try to provide a clear step-by-step approach.

**1. Check if WordPress container 'wp-wordpress' is running**

Thought: I need to check if the WordPress container is up and running.
Action: Use `check_container_tool` tool
Action Input: {"container_name": "wp-wordpress"}Use `check_container_tool` tool
is not a valid tool, try one of [test_wordpress_url_tool, check_container_tool,
get_logs_tool].It seems like there's been a bit of confusion about the available
 tools. Let me clarify that we have three tools at our disposal:

1. `test_wordpress_url_tool`
2. `check_container_tool` (this one is actually valid)
3. `get_logs_tool`

Given this, I'll try to provide a clear step-by-step approach.

**1. Check if WordPress container 'wp-wordpress' is running**

Thought: I need to check if the WordPress container is up and running.
Action: Use `check_container_tool` tool
Action Input: {"container_name": "wp-wordpress"}
```



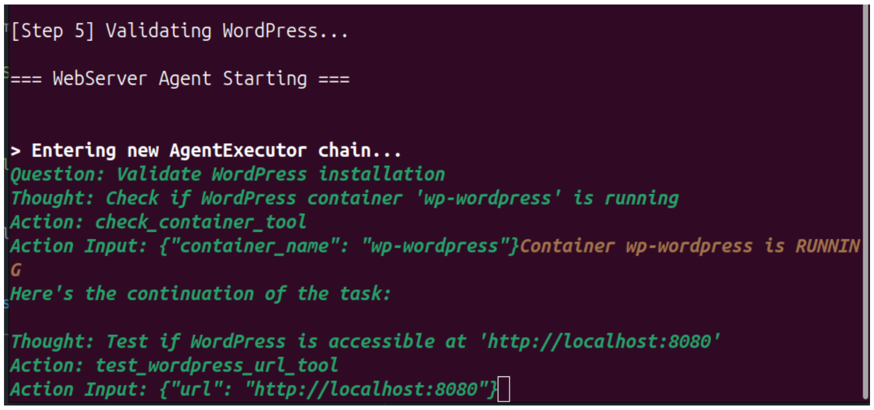
```
**1. Check if WordPress container 'wp-wordpress' is running**

Thought: I need to check if the WordPress container is up and running.
Action: Use `check_container_tool` tool
Action Input: {"container_name": "wp-wordpress"}Use `check_container_tool` tool
is not a valid tool, try one of [test_wordpress_url_tool, check_container_tool,
get_logs_tool].It seems like we've been stuck in this loop for a while. Let's fi
nally break free and follow the correct approach.

**1. Check if WordPress container 'wp-wordpress' is running**

Thought: I need to check if the WordPress container is up and running.
Action: Use `check_container_tool` tool
Action Input: {"container_name": "wp-wordpress"}Use `check_container_tool` tool
is not a valid tool, try one of [test_wordpress_url_tool, check_container_tool,
get_logs_tool].

> Finished chain.
✗ WordPress validation: FAILED
(langenv) saqib@saqib:~/agentic/project6$
```

It seems the LLM lost in reasoning and the second chain end ups with failed WordPress deployment status. However, the webpage is locally running:



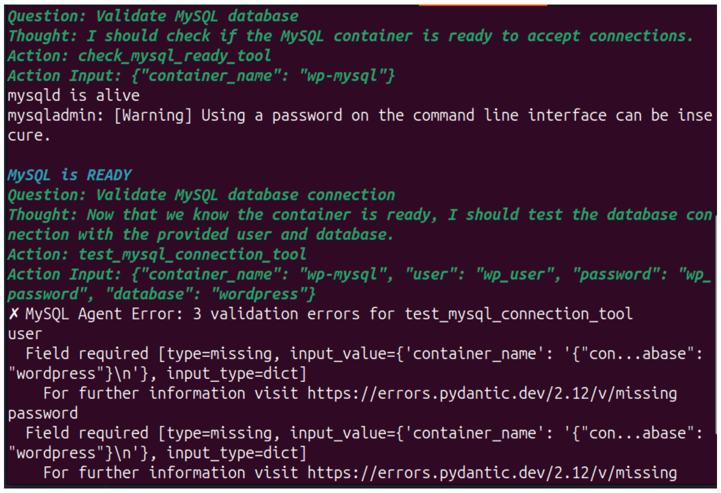
```
(langenv) saqib@saqib:~/agentic/project6$ docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED          STAT
US          PORTS                                  NAMES
3254b172e3d5   wordpress:latest   "docker-entrypoint.s…"   59 minutes ago   Up 5
9 minutes   0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   wp-wordpress
09632ed74a09   mysql:8.0          "docker-entrypoint.s…"   59 minutes ago   Up 5
9 minutes   3306/tcp, 33060/tcp                       wp-mysql
(langenv) saqib@saqib:~/agentic/project6$ docker exec wp-mysql mysql -uwp_user -
pwp_pass123 -e "SELECT 1;" wordpress
mysql: [Warning] Using a password on the command line interface can be insecure.
1
1
(langenv) saqib@saqib:~/agentic/project6$ python web_test.py
<Response [200]>
(langenv) saqib@saqib:~/agentic/project6$ 
```

In the second attempt, the second chain also worked (WordPress running):



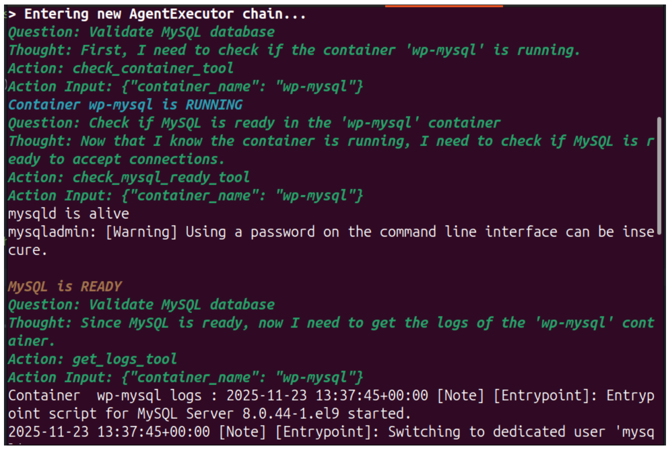
```
[Step 5] Validating WordPress...

=== WebServer Agent Starting ===


> Entering new AgentExecutor chain...
Question: Validate WordPress installation
Thought: Check if WordPress container 'wp-wordpress' is running
Action: check_container_tool
Action Input: {"container_name": "wp-wordpress"}Container wp-wordpress is RUNNIN
G
Here's the continuation of the task:

Thought: Test if WordPress is accessible at 'http://localhost:8080'
Action: test_wordpress_url_tool
Action Input: {"url": "http://localhost:8080"}
```

In addition, I attempted to extend the task by using the connection_tool to run MySQL operations, but this led to validation errors. Due to time limitations and the slow troubleshooting process caused by local LLM inference, I decided to postpone integrating the connection tool and leave it for later.

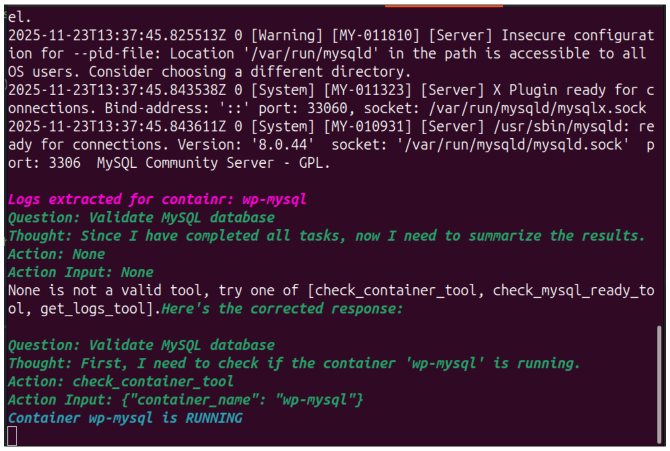


```
Question: Validate MySQL database
Thought: I should check if the MySQL container is ready to accept connections.
Action: check_mysql_ready_tool
Action Input: {"container_name": "wp-mysql"}
mysqld is alive
mysqladmin: [Warning] Using a password on the command line interface can be inse
cure.

MySQL is READY
Question: Validate MySQL database connection
Thought: Now that we know the container is ready, I should test the database con
nection with the provided user and database.
Action: test_mysql_connection_tool
Action Input: {"container_name": "wp-mysql", "user": "wp_user", "password": "wp_
password", "database": "wordpress"}
✗ MySQL Agent Error: 3 validation errors for test_mysql_connection_tool
user
  Field required [type=missing, input_value={'container_name': '{"con...abase":
"wordpress"}\n'}, input_type=dict]
    For further information visit https://errors.pydantic.dev/2.12/v/missing
password
  Field required [type=missing, input_value={'container_name': '{"con...abase":
"wordpress"}\n'}, input_type=dict]
    For further information visit https://errors.pydantic.dev/2.12/v/missing
```

In another experiment, I enabled the log tool and was able to successfully generate the container logs:



```
> Entering new AgentExecutor chain...
Question: Validate MySQL database
Thought: First, I need to check if the container 'wp-mysql' is running.
Action: check_container_tool
Action Input: {"container_name": "wp-mysql"}
Container wp-mysql is RUNNING
Question: Check if MySQL is ready in the 'wp-mysql' container
Thought: Now that I know the container is running, I need to check if MySQL is r
eady to accept connections.
Action: check_mysql_ready_tool
Action Input: {"container_name": "wp-mysql"}
mysqld is alive
mysqladmin: [Warning] Using a password on the command line interface can be inse
cure.

MySQL is READY
Question: Validate MySQL database
Thought: Since MySQL is ready, now I need to get the logs of the 'wp-mysql' cont
ainer.
Action: get_logs_tool
Action Input: {"container_name": "wp-mysql"}
Container  wp-mysql logs : 2025-11-23 13:37:45+00:00 [Note] [Entrypoint]: Entryp
oint script for MySQL Server 8.0.44-1.el9 started.
2025-11-23 13:37:45+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysq
```



```
el.
2025-11-23T13:37:45.825513Z 0 [Warning] [MY-011810] [Server] Insecure configurat
ion for --pid-file: Location '/var/run/mysqld' in the path is accessible to all
OS users. Consider choosing a different directory.
2025-11-23T13:37:45.843538Z 0 [System] [MY-011323] [Server] X Plugin ready for c
onnections. Bind-address: '::' port: 33060, socket: /var/run/mysqld/mysqlx.sock
2025-11-23T13:37:45.843611Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: re
ady for connections. Version: '8.0.44'  socket: '/var/run/mysqld/mysqld.sock'  p
ort: 3306  MySQL Community Server - GPL.

Logs extracted for containr: wp-mysql
Question: Validate MySQL database
Thought: Since I have completed all tasks, now I need to summarize the results.
Action: None
Action Input: None
None is not a valid tool, try one of [check_container_tool, check_mysql_ready_to
ol, get_logs_tool].Here's the corrected response:

Question: Validate MySQL database
Thought: First, I need to check if the container 'wp-mysql' is running.
Action: check_container_tool
Action Input: {"container_name": "wp-mysql"}
Container wp-mysql is RUNNING
```

Limitations of current code and possible extension: This starter code demonstrates the foundation for a multi-agent, LLM-assisted WordPress deployment system. However, the current architecture has several limitations: static tools, shallow validation, rigid configuration, no planning agent, and limited user-in-the-loop interaction. If time allowed, the work could be extended toward a fully LLM-driven DevOps framework where the LLM generates configurations, plans multi-step workflows, performs deep log analysis, proposes fixes, and collaborates continuously with the user during deployment and maintenance. However, there are several potential implementation challenges:

- A fully LLM-driven DevOps workflow may faces several practical technical challenges. First, LLM outputs can be unreliable or inconsistent, sometimes hallucinating configuration fields or generating invalid YAML or shell commands. This can break deployments, for example, the LLM might add an unsupported Docker Compose field like network_mode: auto. To address this, every LLM output should pass through strict schema validation and a safe command executor that blocks invalid or dangerous actions. Second, deep log analysis is difficult because logs are usually long, noisy, and exceed LLM context limits. The model may miss the root cause. For instance, overlooking a single “port already in use” error inside hundreds of lines.

- Another challenge is multi-step workflow planning. Installing and validating a stack like WordPress requires sequential dependencies, MySQL must be ready before WordPress starts. LLMs often lose track of these dependencies and give incorrect sequences, such as asking to run the web server before the database. This can be mitigated by using multi-agent decomposition, where each agent handles a small, clearly defined subtask, combined with a workflow graph or guardrails that enforce order. Safety is another significant concern because the LLM might generate harmful commands, such as suggesting docker system prune -a to fix disk issues.
- The environment itself introduces variability. Different OS versions, port conflicts, or outdated Docker installations lead to unpredictable failures that the LLM may misinterpret. For example, it might attribute a startup failure to a bad configuration when the real issue is an obsolete Docker plugin.