GEN AI 인텐시브 과정

강사 장철원

Section 0
코스소개

DAY1	DAY2	DAY3	DAY4	DAY5	DAY6	DAY7	DAY8
LLM Basic Concept	Transformers paper review	Transformers LangChain LangGraph			LLM service develop	Final Project	

목차

GEN AI 인텐시브 과정

Section 1. 실무 적용

Section 1-1. 모델 성능

모델의 중요성

LLM 좋은 모델 사용 -> 성능 향상

라이브러리 설치

```
azureuser@b2b28-ML:~$ pyenv activate py3_11_9

(py3_11_9) azureuser@b2b28-ML:~$ pip install langchain-mcp-adapters

(py3_11_9) azureuser@b2b28-ML:~$ pip install nest_asyncio
```

mcp_server.py

```
(py3_11_9) azureuser@b2b28-ML:~/work/jupyter/3_llm_service$ vim mcp_server.py
```

LangChain mcp adapter를 활용한 MCP 실습

mcp_server.py

```python
import uuid
from typing import Dict
from mcp.server.fastmcp import FastMCP
from langchain_chroma import Chroma
from langchain_openai import AzureOpenAIEmbeddings, AzureChatOpenAI
from langchain_core.chat_history import InMemoryChatMessageHistory

AZURE_ENDPOINT = "https://b2b28-md6zczaj-eastus2.cognitiveservices.azure.com/"
AZURE_API_KEY = "API키"
API_VERSION = "2024-12-01-preview"
EMBEDDING_DEPLOYMENT = "text-embedding-3-small"
CHAT_DEPLOYMENT = "gpt-4.1-mini"

azure_llm = AzureChatOpenAI(
    api_key=AZURE_API_KEY,
    azure_endpoint=AZURE_ENDPOINT,
    azure_deployment=CHAT_DEPLOYMENT,
    openai_api_version=API_VERSION,
    temperature=0.3,
    max_tokens=800
)

embedding_model = AzureOpenAIEmbeddings(
    azure_endpoint=AZURE_ENDPOINT,
    azure_deployment=EMBEDDING_DEPLOYMENT,
    api_key=AZURE_API_KEY,
    openai_api_version=API_VERSION
)

retriever = Chroma(
    persist_directory="./chromaDB",
    embedding_function=embedding_model
).as_retriever(search_kwargs={"k": 3})

chat_histories: Dict[str, InMemoryChatMessageHistory] = {}

def get_chat_history(session_id: str):
    if session_id not in chat_histories:
        chat_histories[session_id] = InMemoryChatMessageHistory()
    return chat_histories[session_id]

mcp = FastMCP("Agents")

@mcp.tool()
def rag_tool(question: str, session_id: str) -> Dict:
    chat_history = get_chat_history(session_id)
    history_text = "\n".join([m.content for m in chat_history.messages])
    docs = retriever.invoke(question)
    top_docs = docs[:3]
    context = "\n".join(doc.page_content for doc in top_docs)
    full_context = f"{history_text}\n\n{context}" if history_text else context

    prompt = f"""다음은 문맥입니다:\n{full_context}\n\n질문: {question}\n\n답변:"""
    result = azure_llm.invoke(prompt).content

    chat_history.add_user_message(question)
    chat_history.add_ai_message(result)

    return {
        "answer": result,
        "references": [doc.page_content for doc in top_docs]
    }

@mcp.tool()
def summarize_tool(text: str) -> str:
    prompt = f"다음 텍스트를 한국어로 요약해줘:\n\n{text}"
    return azure_llm.invoke(prompt).content

@mcp.tool()
def rephrase_tool(text: str) -> str:
    prompt = f"다음 문장을 정중하고 예의 바르게 바꿔줘:\n\n{text}"
    return azure_llm.invoke(prompt).content

if __name__ == "__main__":
    mcp.run(transport="stdio")
```

```python
import uuid
from typing import Dict
from mcp.server.fastmcp import FastMCP
from langchain_chroma import Chroma
from langchain_openai import AzureOpenAIEmbeddings, AzureChatOpenAI
from langchain_core.chat_history import InMemoryChatMessageHistory

AZURE_ENDPOINT = "https://b2b28-md6zczaj-eastus2.cognitiveservices.azure.com/"
AZURE_API_KEY = "API키"
API_VERSION = "2024-12-01-preview"
EMBEDDING_DEPLOYMENT = "text-embedding-3-small"
CHAT_DEPLOYMENT = "gpt-4.1-mini"

azure_llm = AzureChatOpenAI(
    api_key=AZURE_API_KEY,
    azure_endpoint=AZURE_ENDPOINT,
    azure_deployment=CHAT_DEPLOYMENT,
    openai_api_version=API_VERSION,
    temperature=0.3,
    max_tokens=800
)

embedding_model = AzureOpenAIEmbeddings(
    azure_endpoint=AZURE_ENDPOINT,
    azure_deployment=EMBEDDING_DEPLOYMENT,
    api_key=AZURE_API_KEY,
    openai_api_version=API_VERSION
)

retriever = Chroma(
    persist_directory="./chromaDB",
    embedding_function=embedding_model
).as_retriever(search_kwargs={"k": 3})

chat_histories: Dict[str, InMemoryChatMessageHistory] = {}

def get_chat_history(session_id: str):
    if session_id not in chat_histories:
        chat_histories[session_id] = InMemoryChatMessageHistory()
    return chat_histories[session_id]
```

mcp_server.py

```python
import uuid
from typing import Dict
from mcp.server.fastmcp import FastMCP
from langchain_chroma import Chroma
from langchain_openai import AzureOpenAIEmbeddings, AzureChatOpenAI
from langchain_core.chat_history import InMemoryChatMessageHistory

AZURE_ENDPOINT = "https://b2b28-md6zczaj-eastus2.cognitiveservices.azure.com/"
AZURE_API_KEY = "API키"
API_VERSION = "2024-12-01-preview"
EMBEDDING_DEPLOYMENT = "text-embedding-3-small"
CHAT_DEPLOYMENT = "gpt-4.1-mini"

azure_llm = AzureChatOpenAI(
    api_key=AZURE_API_KEY,
    azure_endpoint=AZURE_ENDPOINT,
    azure_deployment=CHAT_DEPLOYMENT,
    openai_api_version=API_VERSION,
    temperature=0.3,
    max_tokens=800
)

embedding_model = AzureOpenAIEmbeddings(
    azure_endpoint=AZURE_ENDPOINT,
    azure_deployment=EMBEDDING_DEPLOYMENT,
    api_key=AZURE_API_KEY,
    openai_api_version=API_VERSION
)

retriever = Chroma(
    persist_directory="./chromaDB",
    embedding_function=embedding_model
).as_retriever(search_kwargs={"k": 3})

chat_histories: Dict[str, InMemoryChatMessageHistory] = {}

def get_chat_history(session_id: str):
    if session_id not in chat_histories:
        chat_histories[session_id] = InMemoryChatMessageHistory()
    return chat_histories[session_id]

mcp = FastMCP("Agents")

@mcp.tool()
def rag_tool(question: str, session_id: str) -> Dict:
    chat_history = get_chat_history(session_id)
    history_text = "\n".join([m.content for m in chat_history.messages])
    docs = retriever.invoke(question)
    top_docs = docs[:3]
    context = "\n".join(doc.page_content for doc in top_docs)
    full_context = f"{history_text}\n\n{context}" if history_text else context

    prompt = f"""다음은 문맥입니다:\n{full_context}\n\n질문: {question}\n\n답변:"""
    result = azure_llm.invoke(prompt).content

    chat_history.add_user_message(question)
    chat_history.add_ai_message(result)

    return {
        "answer": result,
        "references": [doc.page_content for doc in top_docs]
    }

@mcp.tool()
def summarize_tool(text: str) -> str:
    prompt = f"다음 텍스트를 한국어로 요약해줘:\n\n{text}"
    return azure_llm.invoke(prompt).content

@mcp.tool()
def rephrase_tool(text: str) -> str:
    prompt = f"다음 문장을 정중하고 예의 바르게 바꿔줘:\n\n{text}"
    return azure_llm.invoke(prompt).content

if __name__ == "__main__":
    mcp.run(transport="stdio")
```

```python
mcp = FastMCP("Agents")

@mcp.tool()
def rag_tool(question: str, session_id: str) -> Dict:
    chat_history = get_chat_history(session_id)
    history_text = "\n".join([m.content for m in chat_history.messages])
    docs = retriever.invoke(question)
    top_docs = docs[:3]
    context = "\n".join(doc.page_content for doc in top_docs)
    full_context = f"{history_text}\n\n{context}" if history_text else context

    prompt = f"""다음은 문맥입니다:\n{full_context}\n\n질문: {question}\n\n답변:"""
    result = azure_llm.invoke(prompt).content

    chat_history.add_user_message(question)
    chat_history.add_ai_message(result)

    return {
        "answer": result,
        "references": [doc.page_content for doc in top_docs]
    }

@mcp.tool()
def summarize_tool(text: str) -> str:
    prompt = f"다음 텍스트를 한국어로 요약해줘:\n\n{text}"
    return azure_llm.invoke(prompt).content

@mcp.tool()
def rephrase_tool(text: str) -> str:
    prompt = f"다음 문장을 정중하고 예의 바르게 바꿔줘:\n\n{text}"
    return azure_llm.invoke(prompt).content

if __name__ == "__main__":
    mcp.run(transport="stdio")
```

mcp_client.py

```
(py3_11_9) azureuser@b2b28-ML:~/work/jupyter/3_llm_service$ vim mcp_client.py
```

mcp_client.py

```python
import asyncio
import uuid
from mcp import ClientSession, StdioServerParameters
from mcp.client.stdio import stdio_client
from langchain_mcp_adapters.tools import load_mcp_tools
from langgraph.prebuilt import create_react_agent
from langchain_openai import AzureChatOpenAI

AZURE_ENDPOINT = "https://b2b28-md6zczaj-eastus2.cognitiveservices.azure.com/"
AZURE_API_KEY = "APIKey"
API_VERSION = "2024-12-01-preview"
CHAT_DEPLOYMENT = "gpt-4.1-mini"

model = AzureChatOpenAI(
    api_key=AZURE_API_KEY,
    azure_endpoint=AZURE_ENDPOINT,
    azure_deployment=CHAT_DEPLOYMENT,
    openai_api_version=API_VERSION,
    temperature=0.3,
)

async def main():
    session_id = str(uuid.uuid4())

    server_params = StdioServerParameters(
        command="python",
        args=["mcp_server.py"],  # MCP 서버 경로
    )

    async with stdio_client(server_params) as (read, write):
        async with ClientSession(read, write) as session:
            await session.initialize()
            tools = await load_mcp_tools(session)
            agent = create_react_agent(model, tools)

            while True:
                question = input("\n질문을 입력해주세요: ").strip()
                if not question:
                    break

                if "요약" in question:
                    tool_prompt = f"Use summarize_tool to summarize:\n\n{question}"
                elif "정중" in question or "공손" in question or "예의" in question:
                    tool_prompt = f"Use rephrase_tool to rephrase:\n\n{question}"
                else:
                    tool_prompt = f"Use rag_tool to answer:\nquestion: {question}\nsession_id: {session_id}"

                response = await agent.ainvoke({"messages": tool_prompt})

                print("\n[에이전트 응답]")
                print(response)

                cont = input("\n계속하시겠습니까? (예/아니오): ").strip()
                if not cont.startswith("예"):
                    break

if __name__ == "__main__":
    asyncio.run(main())
```

```python
import asyncio
import uuid
from mcp import ClientSession, StdioServerParameters
from mcp.client.stdio import stdio_client
from langchain_mcp_adapters.tools import load_mcp_tools
from langgraph.prebuilt import create_react_agent
from langchain_openai import AzureChatOpenAI

AZURE_ENDPOINT = "https://b2b28-md6zczaj-eastus2.cognitiveservices.azure.com/"
AZURE_API_KEY = "APIKey"
API_VERSION = "2024-12-01-preview"
CHAT_DEPLOYMENT = "gpt-4.1-mini"

model = AzureChatOpenAI(
    api_key=AZURE_API_KEY,
    azure_endpoint=AZURE_ENDPOINT,
    azure_deployment=CHAT_DEPLOYMENT,
    openai_api_version=API_VERSION,
    temperature=0.3,
)
```

Section

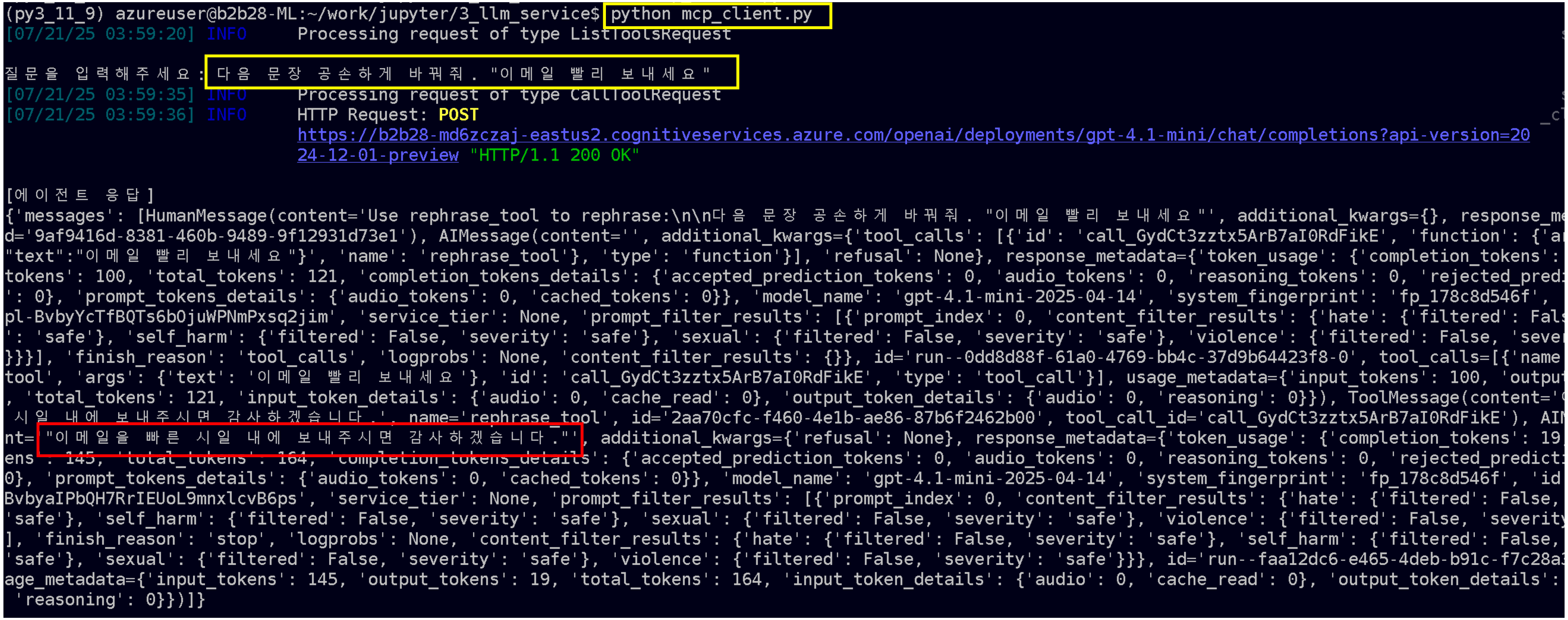
LangChain mcp adapter를 활용한 MCP 실습

mcp_client.py

```python
import asyncio
import uuid
from mcp import ClientSession, StdioServerParameters
from mcp.client.stdio import stdio_client
from langchain_mcp_adapters.tools import load_mcp_tools
from langgraph.prebuilt import create_react_agent
from langchain_openai import AzureChatOpenAI

AZURE_ENDPOINT = "https://b2b28-md6zczaj-eastus2.cognitiveservices.azure.com/"
AZURE_API_KEY = "APIKey"
API_VERSION = "2024-12-01-preview"
CHAT_DEPLOYMENT = "gpt-4.1-mini"

model = AzureChatOpenAI(
    api_key=AZURE_API_KEY,
    azure_endpoint=AZURE_ENDPOINT,
    azure_deployment=CHAT_DEPLOYMENT,
    openai_api_version=API_VERSION,
    temperature=0.3,
)

async def main():
    session_id = str(uuid.uuid4())

    server_params = StdioServerParameters(
        command="python",
        args=["mcp_server.py"],  # MCP 서버 경로
    )

    async with stdio_client(server_params) as (read, write):
        async with ClientSession(read, write) as session:
            await session.initialize()
            tools = await load_mcp_tools(session)
            agent = create_react_agent(model, tools)

            while True:
                question = input("\n질문을 입력해주세요: ").strip()
                if not question:
                    break

                if "요약" in question:
                    tool_prompt = f"Use summarize_tool to summarize:\n\n{question}"
                elif "정중" in question or "공손" in question or "예의" in question:
                    tool_prompt = f"Use rephrase_tool to rephrase:\n\n{question}"
                else:
                    tool_prompt = f"Use rag_tool to answer:\nquestion: {question}\nsession_id: {session_id}"

                response = await agent.ainvoke({"messages": tool_prompt})

                print("\n[에이전트 응답]")
                print(response)

                cont = input("\n계속하시겠습니까? (예/아니오): ").strip()
                if not cont.startswith("예"):
                    break

if __name__ == "__main__":
    asyncio.run(main())
```

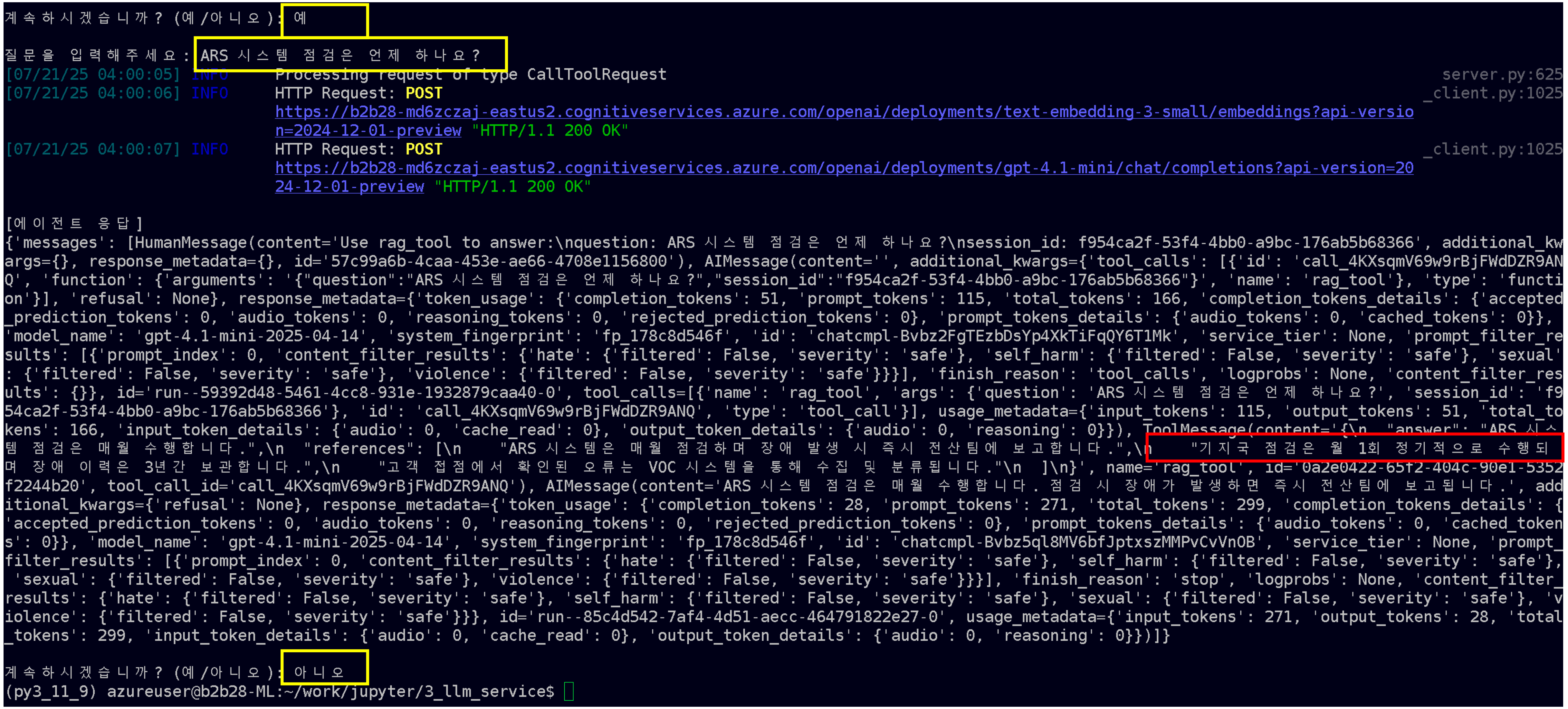
```python
async def main():
    session_id = str(uuid.uuid4())

    server_params = StdioServerParameters(
        command="python",
        args=["mcp_server.py"],  # MCP 서버 경로
    )

    async with stdio_client(server_params) as (read, write):
        async with ClientSession(read, write) as session:
            await session.initialize()
            tools = await load_mcp_tools(session)
            agent = create_react_agent(model, tools)

            while True:
                question = input("\n질문을 입력해주세요: ").strip()
                if not question:
                    break

                if "요약" in question:
                    tool_prompt = f"Use summarize_tool to summarize:\n\n{question}"
                elif "정중" in question or "공손" in question or "예의" in question:
                    tool_prompt = f"Use rephrase_tool to rephrase:\n\n{question}"
                else:
                    tool_prompt = f"Use rag_tool to answer:\nquestion: {question}\nsession_id: {session_id}"

                response = await agent.ainvoke({"messages": tool_prompt})

                print("\n[에이전트 응답]")
                print(response)

                cont = input("\n계속하시겠습니까? (예/아니오): ").strip()
                if not cont.startswith("예"):
                    break

if __name__ == "__main__":
    asyncio.run(main())
```

실행



```
(py3_11_9) azureuser@b2b28-ML:~/work/jupyter/3_llm_service$ python mcp_client.py
[07/21/25 03:59:20] INFO     Processing request of type ListToolsRequest

질문을 입력해주세요: 다음 문장 공손하게 바꿔줘. "이메일 빨리 보내세요"
[07/21/25 03:59:35] INFO     Processing request of type CallToolRequest
[07/21/25 03:59:36] INFO     HTTP Request: POST
                             https://b2b28-md6zczaj-eastus2.cognitiveservices.azure.com/openai/deployments/gpt-4.1-mini/chat/completions?api-version=20
                             24-12-01-preview "HTTP/1.1 200 OK"

[에이전트 응답]
{'messages': [HumanMessage(content='Use rephrase_tool to rephrase:\n\n다음 문장 공손하게 바꿔줘. "이메일 빨리 보내세요"', additional_kwargs={}, response_me
d='9af9416d-8381-460b-9489-9f12931d73e1'), AIMessage(content='', additional_kwargs={'tool_calls': [{'id': 'call_GydCt3zztx5ArB7aI0RdFikE', 'function': {'ar
"text":"이메일 빨리 보내세요"}', 'name': 'rephrase_tool'}, 'type': 'function'}], 'refusal': None}, response_metadata={'token_usage': {'completion_tokens':
tokens': 100, 'total_tokens': 121, 'completion_tokens_details': {'accepted_prediction_tokens': 0, 'audio_tokens': 0, 'reasoning_tokens': 0, 'rejected_predi
': 0}, 'prompt_tokens_details': {'audio_tokens': 0, 'cached_tokens': 0}}, 'model_name': 'gpt-4.1-mini-2025-04-14', 'system_fingerprint': 'fp_178c8d546f',
pl-BvbyYcTfBQTs6bOjuWPNmPxsq2jim', 'service_tier': None, 'prompt_filter_results': [{'prompt_index': 0, 'content_filter_results': {'hate': {'filtered': Fals
': 'safe'}, 'self_harm': {'filtered': False, 'severity': 'safe'}, 'sexual': {'filtered': False, 'severity': 'safe'}, 'violence': {'filtered': False, 'sever
}}}], 'finish_reason': 'tool_calls', 'logprobs': None, 'content_filter_results': {}}, id='run--0dd8d88f-61a0-4769-bb4c-37d9b64423f8-0', tool_calls=[{'name
tool', 'args': {'text': '이메일 빨리 보내세요'}, 'id': 'call_GydCt3zztx5ArB7aI0RdFikE', 'type': 'tool_call'}], usage_metadata={'input_tokens': 100, 'output
, 'total_tokens': 121, 'input_token_details': {'audio': 0, 'cache_read': 0}, 'output_token_details': {'audio': 0, 'reasoning': 0}}), ToolMessage(content='
 시일 내에 보내주시면 감사하겠습니다.', name='rephrase_tool', id='2aa70cfc-f460-4e1b-ae86-87b6f2462b00', tool_call_id='call_GydCt3zztx5ArB7aI0RdFikE'), AIM
nt='"이메일을 빠른 시일 내에 보내주시면 감사하겠습니다."', additional_kwargs={'refusal': None}, response_metadata={'token_usage': {'completion_tokens': 19,
ens': 145, 'total_tokens': 164, 'completion_tokens_details': {'accepted_prediction_tokens': 0, 'audio_tokens': 0, 'reasoning_tokens': 0, 'rejected_predict
0}, 'prompt_tokens_details': {'audio_tokens': 0, 'cached_tokens': 0}}, 'model_name': 'gpt-4.1-mini-2025-04-14', 'system_fingerprint': 'fp_178c8d546f', 'id
BvbyaIPbQH7RrIEUoL9mnxlcvB6ps', 'service_tier': None, 'prompt_filter_results': [{'prompt_index': 0, 'content_filter_results': {'hate': {'filtered': False,
'safe'}, 'self_harm': {'filtered': False, 'severity': 'safe'}, 'sexual': {'filtered': False, 'severity': 'safe'}, 'violence': {'filtered': False, 'severity
], 'finish_reason': 'stop', 'logprobs': None, 'content_filter_results': {'hate': {'filtered': False, 'severity': 'safe'}, 'self_harm': {'filtered': False,
'safe'}, 'sexual': {'filtered': False, 'severity': 'safe'}, 'violence': {'filtered': False, 'severity': 'safe'}}}, id='run--faa12dc6-e465-4deb-b91c-f7c28a
age_metadata={'input_tokens': 145, 'output_tokens': 19, 'total_tokens': 164, 'input_token_details': {'audio': 0, 'cache_read': 0}, 'output_token_details':
 'reasoning': 0}})]}
```

실행



```
계속하시겠습니까? (예/아니오): 예

질문을 입력해주세요: ARS 시스템 점검은 언제 하나요?
[07/21/25 04:00:05] INFO     Processing request of type CallToolRequest                          server.py:625
[07/21/25 04:00:06] INFO     HTTP Request: POST                                                 _client.py:1025
                             https://b2b28-md6zczaj-eastus2.cognitiveservices.azure.com/openai/deployments/text-embedding-3-small/embeddings?api-versio
                             n=2024-12-01-preview "HTTP/1.1 200 OK"
[07/21/25 04:00:07] INFO     HTTP Request: POST                                                 _client.py:1025
                             https://b2b28-md6zczaj-eastus2.cognitiveservices.azure.com/openai/deployments/gpt-4.1-mini/chat/completions?api-version=20
                             24-12-01-preview "HTTP/1.1 200 OK"

[에이전트 응답]
{'messages': [HumanMessage(content='Use rag_tool to answer:\nquestion: ARS 시스템 점검은 언제 하나요?\nsession_id: f954ca2f-53f4-4bb0-a9bc-176ab5b68366', additional_kw
args={}, response_metadata={}, id='57c99a6b-4caa-453e-ae66-4708e1156800'), AIMessage(content='', additional_kwargs={'tool_calls': [{'id': 'call_4KXsqmV69w9rBjFWdDZR9AN
Q', 'function': {'arguments': '{"question":"ARS 시스템 점검은 언제 하나요?","session_id":"f954ca2f-53f4-4bb0-a9bc-176ab5b68366"}', 'name': 'rag_tool'}, 'type': 'functi
on'}], 'refusal': None}, response_metadata={'token_usage': {'completion_tokens': 51, 'prompt_tokens': 115, 'total_tokens': 166, 'completion_tokens_details': {'accepted
_prediction_tokens': 0, 'audio_tokens': 0, 'reasoning_tokens': 0, 'rejected_prediction_tokens': 0}, 'prompt_tokens_details': {'audio_tokens': 0, 'cached_tokens': 0}},
'model_name': 'gpt-4.1-mini-2025-04-14', 'system_fingerprint': 'fp_178c8d546f', 'id': 'chatcmpl-Bvbz2FgTEzbDsYp4XkTiFqQY6T1Mk', 'service_tier': None, 'prompt_filter_re
sults': [{'prompt_index': 0, 'content_filter_results': {'hate': {'filtered': False, 'severity': 'safe'}, 'self_harm': {'filtered': False, 'severity': 'safe'}, 'sexual'
: {'filtered': False, 'severity': 'safe'}, 'violence': {'filtered': False, 'severity': 'safe'}}}], 'finish_reason': 'tool_calls', 'logprobs': None, 'content_filter_res
ults': {}}, id='run--59392d48-5461-4cc8-931e-1932879caa40-0', tool_calls=[{'name': 'rag_tool', 'args': {'question': 'ARS 시스템 점검은 언제 하나요?', 'session_id': 'f9
54ca2f-53f4-4bb0-a9bc-176ab5b68366'}, 'id': 'call_4KXsqmV69w9rBjFWdDZR9ANQ', 'type': 'tool_call'}], usage_metadata={'input_tokens': 115, 'output_tokens': 51, 'total_to
kens': 166, 'input_token_details': {'audio': 0, 'cache_read': 0}, 'output_token_details': {'audio': 0, 'reasoning': 0}}), ToolMessage(content='{\n  "answer": "ARS 시스
템 점검은 매월 수행합니다.",\n  "references": [\n    "ARS 시스템은 매월 점검하며 장애 발생 시 즉시 전산팀에 보고합니다.",\n    "기지국 점검은 월 1회 정기적으로 수행되
며 장애 이력은 3년간 보관합니다.",\n    "고객 접점에서 확인된 오류는 VOC 시스템을 통해 수집 및 분류됩니다."\n  ]\n}', name='rag_tool', id='0a2e0422-65f2-404c-90e1-5352
f2244b20', tool_call_id='call_4KXsqmV69w9rBjFWdDZR9ANQ'), AIMessage(content='ARS 시스템 점검은 매월 수행합니다. 점검 시 장애가 발생하면 즉시 전산팀에 보고됩니다.', add
itional_kwargs={'refusal': None}, response_metadata={'token_usage': {'completion_tokens': 28, 'prompt_tokens': 271, 'total_tokens': 299, 'completion_tokens_details': {
'accepted_prediction_tokens': 0, 'audio_tokens': 0, 'reasoning_tokens': 0, 'rejected_prediction_tokens': 0}, 'prompt_tokens_details': {'audio_tokens': 0, 'cached_token
s': 0}}, 'model_name': 'gpt-4.1-mini-2025-04-14', 'system_fingerprint': 'fp_178c8d546f', 'id': 'chatcmpl-Bvbz5ql8MV6bfJptxszMMPvCvVnOB', 'service_tier': None, 'prompt_
filter_results': [{'prompt_index': 0, 'content_filter_results': {'hate': {'filtered': False, 'severity': 'safe'}, 'self_harm': {'filtered': False, 'severity': 'safe'},
 'sexual': {'filtered': False, 'severity': 'safe'}, 'violence': {'filtered': False, 'severity': 'safe'}}}], 'finish_reason': 'stop', 'logprobs': None, 'content_filter_
results': {'hate': {'filtered': False, 'severity': 'safe'}, 'self_harm': {'filtered': False, 'severity': 'safe'}, 'sexual': {'filtered': False, 'severity': 'safe'}, 'v
iolence': {'filtered': False, 'severity': 'safe'}}}, id='run--85c4d542-7af4-4d51-aecc-464791822e27-0', usage_metadata={'input_tokens': 271, 'output_tokens': 28, 'total
_tokens': 299, 'input_token_details': {'audio': 0, 'cache_read': 0}, 'output_token_details': {'audio': 0, 'reasoning': 0}})]}

계속하시겠습니까? (예/아니오): 아니오
(py3_11_9) azureuser@b2b28-ML:~/work/jupyter/3_llm_service$
```

MCP

Section 2. MCP 실습

Section 2-3. ReAct Agent 기반 실습

ReAct

Reasoning + Acting

질문의 의미를 분석하고,
어떤 도구를 사용할지 결정

자신이 결정한
도구를 사용

mcp_server.py(변동X)

```python
import uuid
from typing import Dict
from mcp.server.fastmcp import FastMCP
from langchain_chroma import Chroma
from langchain_openai import AzureOpenAIEmbeddings, AzureChatOpenAI
from langchain_core.chat_history import InMemoryChatMessageHistory

AZURE_ENDPOINT = "https://b2b28-md6zczaj-eastus2.cognitiveservices.azure.com/"
AZURE_API_KEY = "API키"
API_VERSION = "2024-12-01-preview"
EMBEDDING_DEPLOYMENT = "text-embedding-3-small"
CHAT_DEPLOYMENT = "gpt-4.1-mini"

azure_llm = AzureChatOpenAI(
    api_key=AZURE_API_KEY,
    azure_endpoint=AZURE_ENDPOINT,
    azure_deployment=CHAT_DEPLOYMENT,
    openai_api_version=API_VERSION,
    temperature=0.3,
    max_tokens=800
)

embedding_model = AzureOpenAIEmbeddings(
    azure_endpoint=AZURE_ENDPOINT,
    azure_deployment=EMBEDDING_DEPLOYMENT,
    api_key=AZURE_API_KEY,
    openai_api_version=API_VERSION
)

retriever = Chroma(
    persist_directory="./chromaDB",
    embedding_function=embedding_model
).as_retriever(search_kwargs={"k": 3})

chat_histories: Dict[str, InMemoryChatMessageHistory] = {}

def get_chat_history(session_id: str):
    if session_id not in chat_histories:
        chat_histories[session_id] = InMemoryChatMessageHistory()
    return chat_histories[session_id]

mcp = FastMCP("Agents")

@mcp.tool()
def rag_tool(question: str, session_id: str) -> Dict:
    chat_history = get_chat_history(session_id)
    history_text = "\n".join([m.content for m in chat_history.messages])
    docs = retriever.invoke(question)
    top_docs = docs[:3]
    context = "\n".join(doc.page_content for doc in top_docs)
    full_context = f"{history_text}\n\n{context}" if history_text else context

    prompt = f"""다음은 문맥입니다:\n{full_context}\n\n질문: {question}\n\n답변:"""
    result = azure_llm.invoke(prompt).content

    chat_history.add_user_message(question)
    chat_history.add_ai_message(result)

    return {
        "answer": result,
        "references": [doc.page_content for doc in top_docs]
    }

@mcp.tool()
def summarize_tool(text: str) -> str:
    prompt = f"다음 텍스트를 한국어로 요약해줘:\n\n{text}"
    return azure_llm.invoke(prompt).content

@mcp.tool()
def rephrase_tool(text: str) -> str:
    prompt = f"다음 문장을 정중하고 예의 바르게 바꿔줘:\n\n{text}"
    return azure_llm.invoke(prompt).content

if __name__ == "__main__":
    mcp.run(transport="stdio")
```

```python
import uuid
from typing import Dict
from mcp.server.fastmcp import FastMCP
from langchain_chroma import Chroma
from langchain_openai import AzureOpenAIEmbeddings, AzureChatOpenAI
from langchain_core.chat_history import InMemoryChatMessageHistory

AZURE_ENDPOINT = "https://b2b28-md6zczaj-eastus2.cognitiveservices.azure.com/"
AZURE_API_KEY = "API키"
API_VERSION = "2024-12-01-preview"
EMBEDDING_DEPLOYMENT = "text-embedding-3-small"
CHAT_DEPLOYMENT = "gpt-4.1-mini"

azure_llm = AzureChatOpenAI(
    api_key=AZURE_API_KEY,
    azure_endpoint=AZURE_ENDPOINT,
    azure_deployment=CHAT_DEPLOYMENT,
    openai_api_version=API_VERSION,
    temperature=0.3,
    max_tokens=800
)

embedding_model = AzureOpenAIEmbeddings(
    azure_endpoint=AZURE_ENDPOINT,
    azure_deployment=EMBEDDING_DEPLOYMENT,
    api_key=AZURE_API_KEY,
    openai_api_version=API_VERSION
)

retriever = Chroma(
    persist_directory="./chromaDB",
    embedding_function=embedding_model
).as_retriever(search_kwargs={"k": 3})

chat_histories: Dict[str, InMemoryChatMessageHistory] = {}

def get_chat_history(session_id: str):
    if session_id not in chat_histories:
        chat_histories[session_id] = InMemoryChatMessageHistory()
    return chat_histories[session_id]
```

mcp_server.py(변동X)

```
import uuid
from typing import Dict
from mcp.server.fastmcp import FastMCP
from langchain_chroma import Chroma
from langchain_openai import AzureOpenAIEmbeddings, AzureChatOpenAI
from langchain_core.chat_history import InMemoryChatMessageHistory

AZURE_ENDPOINT = "https://b2b28-md6zczaj-eastus2.cognitiveservices.azure.com/"
AZURE_API_KEY = "API키"
API_VERSION = "2024-12-01-preview"
EMBEDDING_DEPLOYMENT = "text-embedding-3-small"
CHAT_DEPLOYMENT = "gpt-4.1-mini"

azure_llm = AzureChatOpenAI(
    api_key=AZURE_API_KEY,
    azure_endpoint=AZURE_ENDPOINT,
    azure_deployment=CHAT_DEPLOYMENT,
    openai_api_version=API_VERSION,
    temperature=0.3,
    max_tokens=800
)

embedding_model = AzureOpenAIEmbeddings(
    azure_endpoint=AZURE_ENDPOINT,
    azure_deployment=EMBEDDING_DEPLOYMENT,
    api_key=AZURE_API_KEY,
    openai_api_version=API_VERSION
)

retriever = Chroma(
    persist_directory="./chromaDB",
    embedding_function=embedding_model
).as_retriever(search_kwargs={"k": 3})

chat_histories: Dict[str, InMemoryChatMessageHistory] = {}

def get_chat_history(session_id: str):
    if session_id not in chat_histories:
        chat_histories[session_id] = InMemoryChatMessageHistory()
    return chat_histories[session_id]

mcp = FastMCP("Agents")

@mcp.tool()
def rag_tool(question: str, session_id: str) -> Dict:
    chat_history = get_chat_history(session_id)
    history_text = "\n".join([m.content for m in chat_history.messages])
    docs = retriever.invoke(question)
    top_docs = docs[:3]
    context = "\n".join(doc.page_content for doc in top_docs)
    full_context = f"{history_text}\n\n{context}" if history_text else context

    prompt = f"""다음은 문맥입니다:\n{full_context}\n\n질문: {question}\n\n답변:"""
    result = azure_llm.invoke(prompt).content

    chat_history.add_user_message(question)
    chat_history.add_ai_message(result)

    return {
        "answer": result,
        "references": [doc.page_content for doc in top_docs]
    }

@mcp.tool()
def summarize_tool(text: str) -> str:
    prompt = f"다음 텍스트를 한국어로 요약해줘:\n\n{text}"
    return azure_llm.invoke(prompt).content

@mcp.tool()
def rephrase_tool(text: str) -> str:
    prompt = f"다음 문장을 정중하고 예의 바르게 바꿔줘:\n\n{text}"
    return azure_llm.invoke(prompt).content

if __name__ == "__main__":
    mcp.run(transport="stdio")
```

```
mcp = FastMCP("Agents")

@mcp.tool()
def rag_tool(question: str, session_id: str) -> Dict:
    chat_history = get_chat_history(session_id)
    history_text = "\n".join([m.content for m in chat_history.messages])
    docs = retriever.invoke(question)
    top_docs = docs[:3]
    context = "\n".join(doc.page_content for doc in top_docs)
    full_context = f"{history_text}\n\n{context}" if history_text else context

    prompt = f"""다음은 문맥입니다:\n{full_context}\n\n질문: {question}\n\n답변:"""
    result = azure_llm.invoke(prompt).content

    chat_history.add_user_message(question)
    chat_history.add_ai_message(result)

    return {
        "answer": result,
        "references": [doc.page_content for doc in top_docs]
    }

@mcp.tool()
def summarize_tool(text: str) -> str:
    prompt = f"다음 텍스트를 한국어로 요약해줘:\n\n{text}"
    return azure_llm.invoke(prompt).content

@mcp.tool()
def rephrase_tool(text: str) -> str:
    prompt = f"다음 문장을 정중하고 예의 바르게 바꿔줘:\n\n{text}"
    return azure_llm.invoke(prompt).content

if __name__ == "__main__":
    mcp.run(transport="stdio")
```

mcp_client.py

```
(py3_11_9) azureuser@b2b28-ML:~/work/jupyter/3_llm_service$ vim mcp_react_client.py
```

mcp_react_client.py

```python
import asyncio
import uuid
from mcp import ClientSession, StdioServerParameters
from mcp.client.stdio import stdio_client
from langchain_mcp_adapters.tools import load_mcp_tools
from langgraph.prebuilt import create_react_agent
from langchain_openai import AzureChatOpenAI

# Azure 설정
AZURE_ENDPOINT = "https://b2b28-md6zczaj-eastus2.cognitiveservices.azure.com/"
AZURE_API_KEY = “APIKey"
API_VERSION = "2024-12-01-preview"
CHAT_DEPLOYMENT = "gpt-4.1-mini"

model = AzureChatOpenAI(
    api_key=AZURE_API_KEY,
    azure_endpoint=AZURE_ENDPOINT,
    azure_deployment=CHAT_DEPLOYMENT,
    openai_api_version=API_VERSION,
    temperature=0.3,
    max_tokens=800
)

async def main():
    session_id = str(uuid.uuid4())

    # MCP 서버 실행 파라미터
    server_params = StdioServerParameters(
        command="python",
        args=["mcp_server.py"],  # MCP 툴 서버 경로
    )

    async with stdio_client(server_params) as (read, write):
        async with ClientSession(read, write) as session:
            await session.initialize()
            tools = await load_mcp_tools(session)

            # React Agent 생성 (툴 목록 제공)
            agent = create_react_agent(model, tools)

            while True:
                user_input = input("\n질문을 입력해주세요: ").strip()
                if not user_input:
                    break

                # React Agent에게 메시지 전달
                messages = [
                    {"role": "user", "content": f"{user_input}\n(session_id: {session_id})"}
                ]
                result = await agent.ainvoke({"messages": messages})

                print("\n[에이전트 응답]")
                print(result)

                cont = input("\n계속하시겠습니까? (예/아니오): ").strip()
                if not cont.startswith("예"):
                    break

if __name__ == "__main__":
    asyncio.run(main())
```

```python
import asyncio
import uuid
from mcp import ClientSession, StdioServerParameters
from mcp.client.stdio import stdio_client
from langchain_mcp_adapters.tools import load_mcp_tools
from langgraph.prebuilt import create_react_agent
from langchain_openai import AzureChatOpenAI

# Azure 설정
AZURE_ENDPOINT = "https://b2b28-md6zczaj-eastus2.cognitiveservices.azure.com/"
AZURE_API_KEY = “APIKey"
API_VERSION = "2024-12-01-preview"
CHAT_DEPLOYMENT = "gpt-4.1-mini"

model = AzureChatOpenAI(
    api_key=AZURE_API_KEY,
    azure_endpoint=AZURE_ENDPOINT,
    azure_deployment=CHAT_DEPLOYMENT,
    openai_api_version=API_VERSION,
    temperature=0.3,
    max_tokens=800
)
```

Section

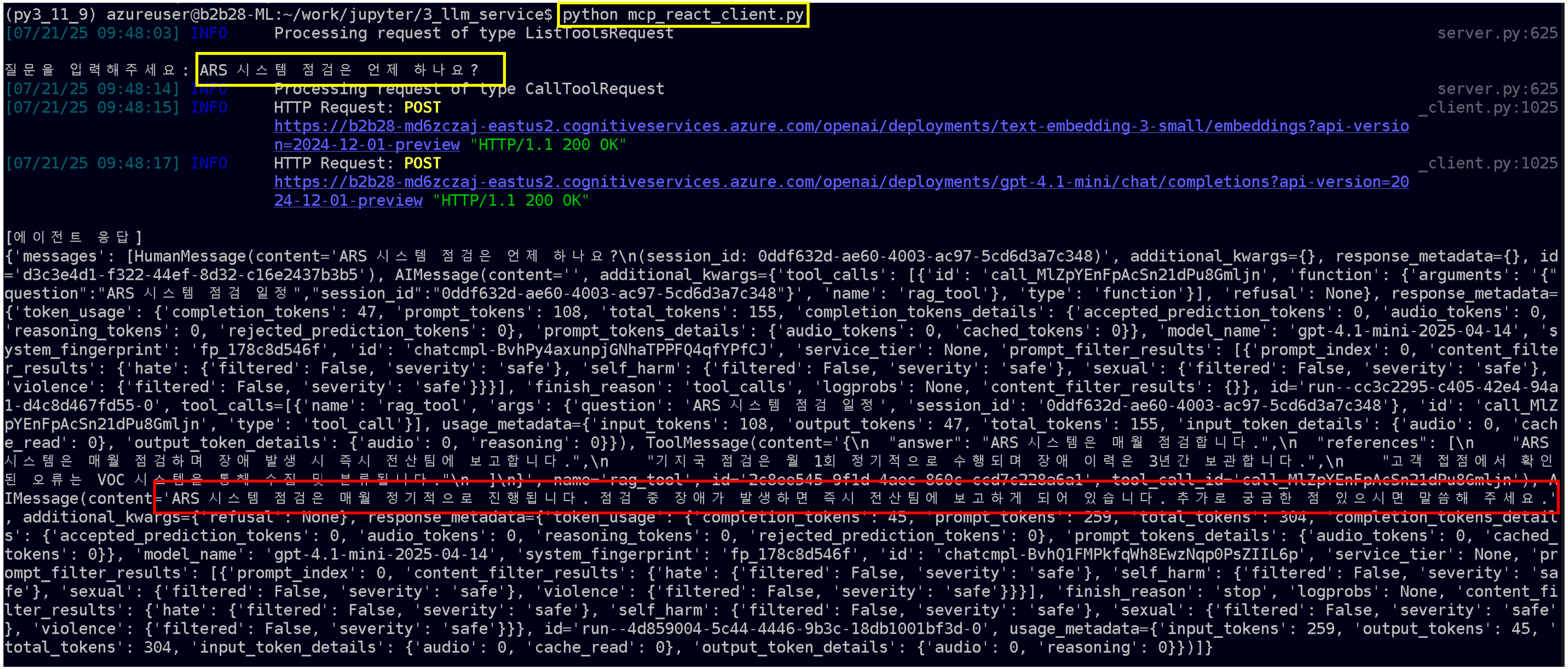
ReAct Agent 기반 실습

mcp_react_client.py

```python
import asyncio
import uuid
from mcp import ClientSession, StdioServerParameters
from mcp.client.stdio import stdio_client
from langchain_mcp_adapters.tools import load_mcp_tools
from langgraph.prebuilt import create_react_agent
from langchain_openai import AzureChatOpenAI

# Azure 설정
AZURE_ENDPOINT = "https://b2b28-md6zczaj-eastus2.cognitiveservices.azure.com/"
AZURE_API_KEY = “APIKey"
API_VERSION = "2024-12-01-preview"
CHAT_DEPLOYMENT = "gpt-4.1-mini"

model = AzureChatOpenAI(
    api_key=AZURE_API_KEY,
    azure_endpoint=AZURE_ENDPOINT,
    azure_deployment=CHAT_DEPLOYMENT,
    openai_api_version=API_VERSION,
    temperature=0.3,
    max_tokens=800
)

async def main():
    session_id = str(uuid.uuid4())

    # MCP 서버 실행 파라미터
    server_params = StdioServerParameters(
        command="python",
        args=["mcp_server.py"],  # MCP 툴 서버 경로
    )

    async with stdio_client(server_params) as (read, write):
        async with ClientSession(read, write) as session:
            await session.initialize()
            tools = await load_mcp_tools(session)

            # React Agent 생성 (툴 목록 제공)
            agent = create_react_agent(model, tools)

            while True:
                user_input = input("\n질문을 입력해주세요: ").strip()
                if not user_input:
                    break

                # React Agent에게 메시지 전달
                messages = [
                    {"role": "user", "content": f"{user_input}\n(session_id: {session_id})"}
                ]
                result = await agent.ainvoke({"messages": messages})

                print("\n[에이전트 응답]")
                print(result)

                cont = input("\n계속하시겠습니까? (예/아니오): ").strip()
                if not cont.startswith("예"):
                    break

if __name__ == "__main__":
    asyncio.run(main())
```

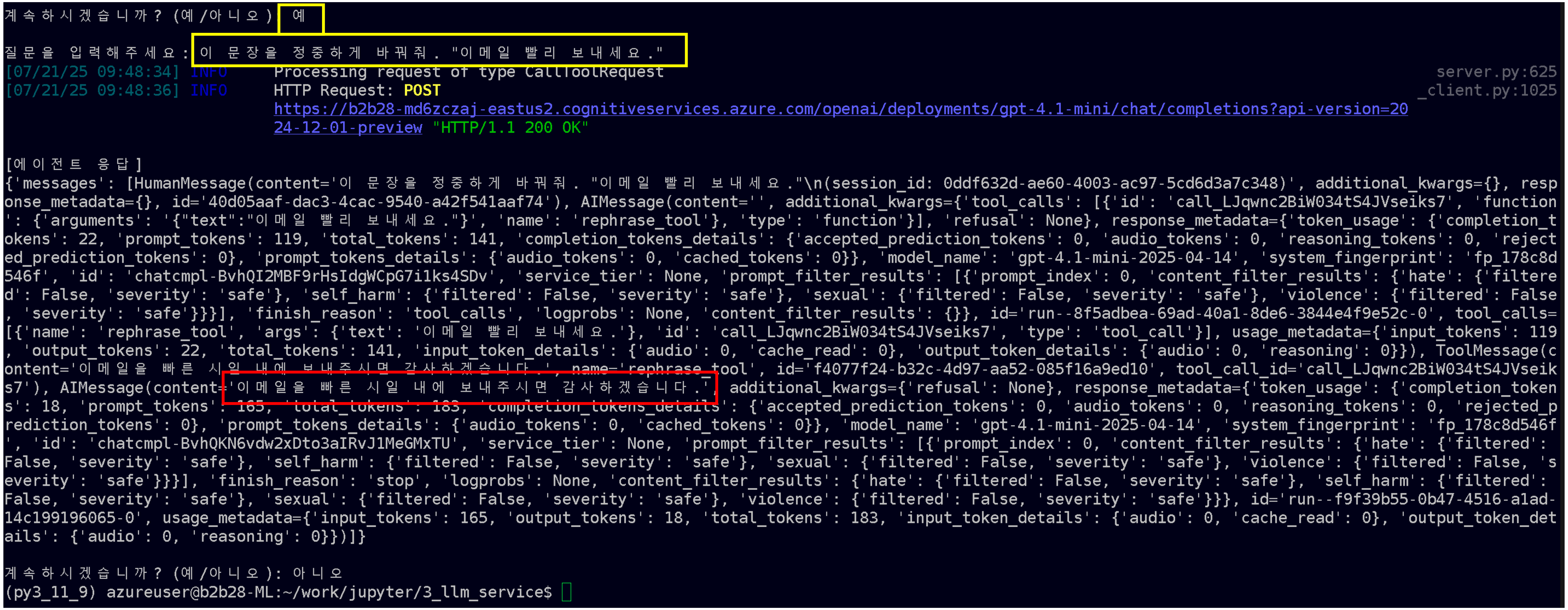
```python
async def main():
    session_id = str(uuid.uuid4())

    # MCP 서버 실행 파라미터
    server_params = StdioServerParameters(
        command="python",
        args=["mcp_server.py"],  # MCP 툴 서버 경로
    )

    async with stdio_client(server_params) as (read, write):
        async with ClientSession(read, write) as session:
            await session.initialize()
            tools = await load_mcp_tools(session)

            # React Agent 생성 (툴 목록 제공)
            agent = create_react_agent(model, tools)

            while True:
                user_input = input("\n질문을 입력해주세요: ").strip()
                if not user_input:
                    break

                # React Agent에게 메시지 전달
                messages = [
                    {"role": "user", "content": f"{user_input}\n(session_id: {session_id})"}
                ]
                result = await agent.ainvoke({"messages": messages})

                print("\n[에이전트 응답]")
                print(result)

                cont = input("\n계속하시겠습니까? (예/아니오): ").strip()
                if not cont.startswith("예"):
                    break

if __name__ == "__main__":
    asyncio.run(main())
```

실행(1)



```
(py3_11_9) azureuser@b2b28-ML:~/work/jupyter/3_llm_service$ python mcp_react_client.py
[07/21/25 09:48:03] INFO     Processing request of type ListToolsRequest                          server.py:625

질문을 입력해주세요: ARS 시스템 점검은 언제 하나요?
[07/21/25 09:48:14] INFO     Processing request of type CallToolRequest                           server.py:625
[07/21/25 09:48:15] INFO     HTTP Request: POST                                                 _client.py:1025
                             https://b2b28-md6zczaj-eastus2.cognitiveservices.azure.com/openai/deployments/text-embedding-3-small/embeddings?api-versio
                             n=2024-12-01-preview "HTTP/1.1 200 OK"
[07/21/25 09:48:17] INFO     HTTP Request: POST                                                 _client.py:1025
                             https://b2b28-md6zczaj-eastus2.cognitiveservices.azure.com/openai/deployments/gpt-4.1-mini/chat/completions?api-version=20
                             24-12-01-preview "HTTP/1.1 200 OK"

[에이전트 응답]
{'messages': [HumanMessage(content='ARS 시스템 점검은 언제 하나요?\n(session_id: 0ddf632d-ae60-4003-ac97-5cd6d3a7c348)', additional_kwargs={}, response_metadata={}, id
='d3c3e4d1-f322-44ef-8d32-c16e2437b3b5'), AIMessage(content='', additional_kwargs={'tool_calls': [{'id': 'call_MlZpYEnFpAcSn21dPu8Gmljn', 'function': {'arguments': '{"
question":"ARS 시스템 점검 일정","session_id":"0ddf632d-ae60-4003-ac97-5cd6d3a7c348"}', 'name': 'rag_tool'}, 'type': 'function'}], 'refusal': None}, response_metadata=
{'token_usage': {'completion_tokens': 47, 'prompt_tokens': 108, 'total_tokens': 155, 'completion_tokens_details': {'accepted_prediction_tokens': 0, 'audio_tokens': 0,
'reasoning_tokens': 0, 'rejected_prediction_tokens': 0}, 'prompt_tokens_details': {'audio_tokens': 0, 'cached_tokens': 0}}, 'model_name': 'gpt-4.1-mini-2025-04-14', 's
ystem_fingerprint': 'fp_178c8d546f', 'id': 'chatcmpl-BvhPy4axunpjGNhaTPPFQ4qfYPfCJ', 'service_tier': None, 'prompt_filter_results': [{'prompt_index': 0, 'content_filte
r_results': {'hate': {'filtered': False, 'severity': 'safe'}, 'self_harm': {'filtered': False, 'severity': 'safe'}, 'sexual': {'filtered': False, 'severity': 'safe'},
'violence': {'filtered': False, 'severity': 'safe'}}}], 'finish_reason': 'tool_calls', 'logprobs': None, 'content_filter_results': {}}, id='run--cc3c2295-c405-42e4-94a
1-d4c8d467fd55-0', tool_calls=[{'name': 'rag_tool', 'args': {'question': 'ARS 시스템 점검 일정', 'session_id': '0ddf632d-ae60-4003-ac97-5cd6d3a7c348'}, 'id': 'call_MlZ
pYEnFpAcSn21dPu8Gmljn', 'type': 'tool_call'}], usage_metadata={'input_tokens': 108, 'output_tokens': 47, 'total_tokens': 155, 'input_token_details': {'audio': 0, 'cach
e_read': 0}, 'output_token_details': {'audio': 0, 'reasoning': 0}}), ToolMessage(content='{\n  "answer": "ARS 시스템은 매월 점검합니다.",\n  "references": [\n    "ARS
시스템은 매월 점검하며 장애 발생 시 즉시 전산팀에 보고합니다.",\n    "기지국 점검은 월 1회 정기적으로 수행되며 장애 이력은 3년간 보관합니다.",\n    "고객 접점에서 확인
된 오류는 VOC 시스템을 통해 수집 및 분류됩니다."\n  ]\n}', name='rag_tool', id='2c8ec545-9f1d-4aec-860c-ccd7c228a6a1', tool_call_id='call_MlZpYEnFpAcSn21dPu8Gmljn'), A
IMessage(content='ARS 시스템 점검은 매월 정기적으로 진행됩니다. 점검 중 장애가 발생하면 즉시 전산팀에 보고하게 되어 있습니다. 추가로 궁금한 점 있으시면 말씀해 주세요.'
, additional_kwargs={'refusal': None}, response_metadata={'token_usage': {'completion_tokens': 45, 'prompt_tokens': 259, 'total_tokens': 304, 'completion_tokens_detail
s': {'accepted_prediction_tokens': 0, 'audio_tokens': 0, 'reasoning_tokens': 0, 'rejected_prediction_tokens': 0}, 'prompt_tokens_details': {'audio_tokens': 0, 'cached_
tokens': 0}}, 'model_name': 'gpt-4.1-mini-2025-04-14', 'system_fingerprint': 'fp_178c8d546f', 'id': 'chatcmpl-BvhQ1FMPkfqWh8EwzNqp0PsZIIL6p', 'service_tier': None, 'pr
ompt_filter_results': [{'prompt_index': 0, 'content_filter_results': {'hate': {'filtered': False, 'severity': 'safe'}, 'self_harm': {'filtered': False, 'severity': 'sa
fe'}, 'sexual': {'filtered': False, 'severity': 'safe'}, 'violence': {'filtered': False, 'severity': 'safe'}}}], 'finish_reason': 'stop', 'logprobs': None, 'content_fi
lter_results': {'hate': {'filtered': False, 'severity': 'safe'}, 'self_harm': {'filtered': False, 'severity': 'safe'}, 'sexual': {'filtered': False, 'severity': 'safe'
}, 'violence': {'filtered': False, 'severity': 'safe'}}}, id='run--4d859004-5c44-4446-9b3c-18db1001bf3d-0', usage_metadata={'input_tokens': 259, 'output_tokens': 45, '
total_tokens': 304, 'input_token_details': {'audio': 0, 'cache_read': 0}, 'output_token_details': {'audio': 0, 'reasoning': 0}})]}
```

실행(2)



```
계속하시겠습니까? (예/아니오) 예

질문을 입력해주세요: 이 문장을 정중하게 바꿔줘. "이메일 빨리 보내세요."
[07/21/25 09:48:34] INFO     Processing request of type CallToolRequest                                  server.py:625
[07/21/25 09:48:36] INFO     HTTP Request: POST                                                       _client.py:1025
                             https://b2b28-md6zczaj-eastus2.cognitiveservices.azure.com/openai/deployments/gpt-4.1-mini/chat/completions?api-version=20
                             24-12-01-preview "HTTP/1.1 200 OK"

[에이전트 응답]
{'messages': [HumanMessage(content='이 문장을 정중하게 바꿔줘. "이메일 빨리 보내세요."\n(session_id: 0ddf632d-ae60-4003-ac97-5cd6d3a7c348)', additional_kwargs={}, resp
onse_metadata={}, id='40d05aaf-dac3-4cac-9540-a42f541aaf74'), AIMessage(content='', additional_kwargs={'tool_calls': [{'id': 'call_LJqwnc2BiW034tS4JVseiks7', 'function
': {'arguments': '{"text":"이메일 빨리 보내세요."}', 'name': 'rephrase_tool'}, 'type': 'function'}], 'refusal': None}, response_metadata={'token_usage': {'completion_t
okens': 22, 'prompt_tokens': 119, 'total_tokens': 141, 'completion_tokens_details': {'accepted_prediction_tokens': 0, 'audio_tokens': 0, 'reasoning_tokens': 0, 'reject
ed_prediction_tokens': 0}, 'prompt_tokens_details': {'audio_tokens': 0, 'cached_tokens': 0}}, 'model_name': 'gpt-4.1-mini-2025-04-14', 'system_fingerprint': 'fp_178c8d
546f', 'id': 'chatcmpl-BvhQI2MBF9rHsIdgWCpG7i1ks4SDv', 'service_tier': None, 'prompt_filter_results': [{'prompt_index': 0, 'content_filter_results': {'hate': {'filtere
d': False, 'severity': 'safe'}, 'self_harm': {'filtered': False, 'severity': 'safe'}, 'sexual': {'filtered': False, 'severity': 'safe'}, 'violence': {'filtered': False
, 'severity': 'safe'}}}], 'finish_reason': 'tool_calls', 'logprobs': None, 'content_filter_results': {}}, id='run--8f5adbea-69ad-40a1-8de6-3844e4f9e52c-0', tool_calls=
[{'name': 'rephrase_tool', 'args': {'text': '이메일 빨리 보내세요.'}, 'id': 'call_LJqwnc2BiW034tS4JVseiks7', 'type': 'tool_call'}], usage_metadata={'input_tokens': 119
, 'output_tokens': 22, 'total_tokens': 141, 'input_token_details': {'audio': 0, 'cache_read': 0}, 'output_token_details': {'audio': 0, 'reasoning': 0}}), ToolMessage(c
ontent='이메일을 빠른 시일 내에 보내주시면 감사하겠습니다.', name='rephrase_tool', id='f4077f24-b32c-4d97-aa52-085f16a9ed10', tool_call_id='call_LJqwnc2BiW034tS4JVseik
s7'), AIMessage(content='이메일을 빠른 시일 내에 보내주시면 감사하겠습니다.', additional_kwargs={'refusal': None}, response_metadata={'token_usage': {'completion_token
s': 18, 'prompt_tokens': 165, 'total_tokens': 183, 'completion_tokens_details': {'accepted_prediction_tokens': 0, 'audio_tokens': 0, 'reasoning_tokens': 0, 'rejected_p
rediction_tokens': 0}, 'prompt_tokens_details': {'audio_tokens': 0, 'cached_tokens': 0}}, 'model_name': 'gpt-4.1-mini-2025-04-14', 'system_fingerprint': 'fp_178c8d546f
', 'id': 'chatcmpl-BvhQKN6vdw2xDto3aIRvJ1MeGMxTU', 'service_tier': None, 'prompt_filter_results': [{'prompt_index': 0, 'content_filter_results': {'hate': {'filtered':
False, 'severity': 'safe'}, 'self_harm': {'filtered': False, 'severity': 'safe'}, 'sexual': {'filtered': False, 'severity': 'safe'}, 'violence': {'filtered': False, 's
everity': 'safe'}}}], 'finish_reason': 'stop', 'logprobs': None, 'content_filter_results': {'hate': {'filtered': False, 'severity': 'safe'}, 'self_harm': {'filtered':
False, 'severity': 'safe'}, 'sexual': {'filtered': False, 'severity': 'safe'}, 'violence': {'filtered': False, 'severity': 'safe'}}}, id='run--f9f39b55-0b47-4516-a1ad-
14c199196065-0', usage_metadata={'input_tokens': 165, 'output_tokens': 18, 'total_tokens': 183, 'input_token_details': {'audio': 0, 'cache_read': 0}, 'output_token_det
ails': {'audio': 0, 'reasoning': 0}})]}

계속하시겠습니까? (예/아니오): 아니오
(py3_11_9) azureuser@b2b28-ML:~/work/jupyter/3_llm_service$
```

GEN AI 인텐시브 과정

Section 1. 실무 적용

Section 1-2. 벡터 DB

FAISS 기초

[2]:
```
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
```

[3]:
```
# 임베딩 모델 설정
embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2")
```

[4]:
```
# 저장할 텍스트 데이터
texts = ["사과는 빨갛다", "바다는 파랗다", "개나리는 노랗다"]
```

[5]:
```
# FAISS에 저장
faiss_store = FAISS.from_texts(texts, embedding=embedding_model)
```

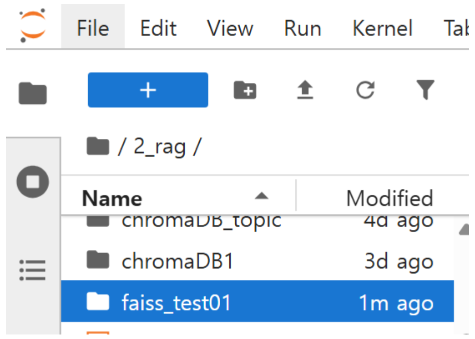
[6]:
```
# 검색
query = "딸기는 빨갛다"
result = faiss_store.similarity_search(query, k=2)
```

[7]:
```
# 검색 결과
for doc in result:
    print(doc.page_content)
```

```
사과는 빨갛다
개나리는 노랗다
```

[8]:
```
# 로컬 저장 & 불러오기
faiss_store.save_local("./faiss_test01")
```

[10]:
```
loaded_store = FAISS.load_local(
    folder_path = "./faiss_test01",
    embeddings = embedding_model,
    allow_dangerous_deserialization=True
)
```

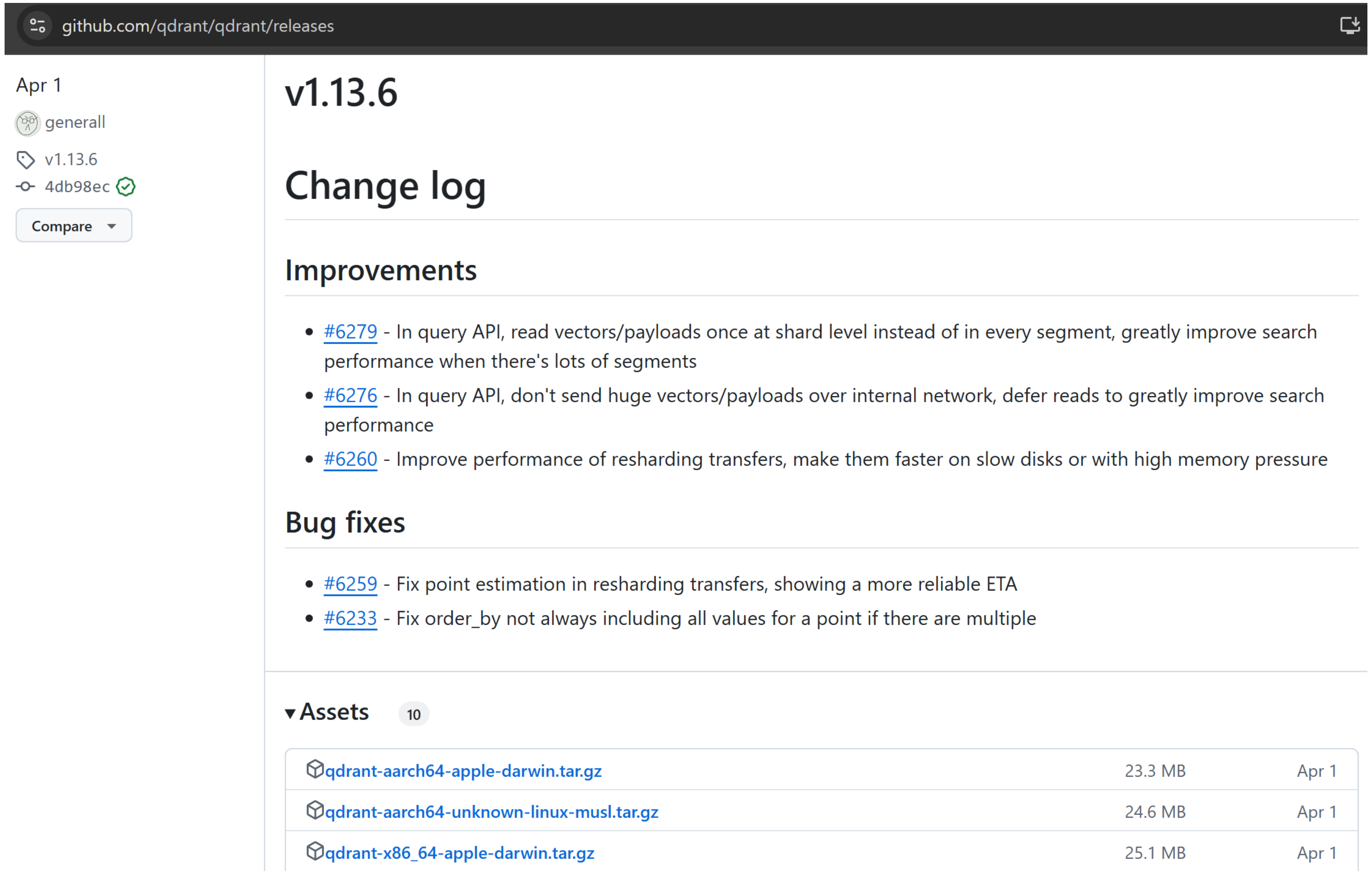


/ 2_rag /

Name	Modified
chromaDB_topic	4d ago
chromaDB1	3d ago
faiss_test01	1m ago

Qdrant 버전 확인

https://github.com/qdrant/qdrant/releases



github.com/qdrant/qdrant/releases

Apr 1
generall
v1.13.6
4db98ec
Compare

v1.13.6

Change log

Improvements

- #6279 - In query API, read vectors/payloads once at shard level instead of in every segment, greatly improve search performance when there's lots of segments
- #6276 - In query API, don't send huge vectors/payloads over internal network, defer reads to greatly improve search performance
- #6260 - Improve performance of resharding transfers, make them faster on slow disks or with high memory pressure

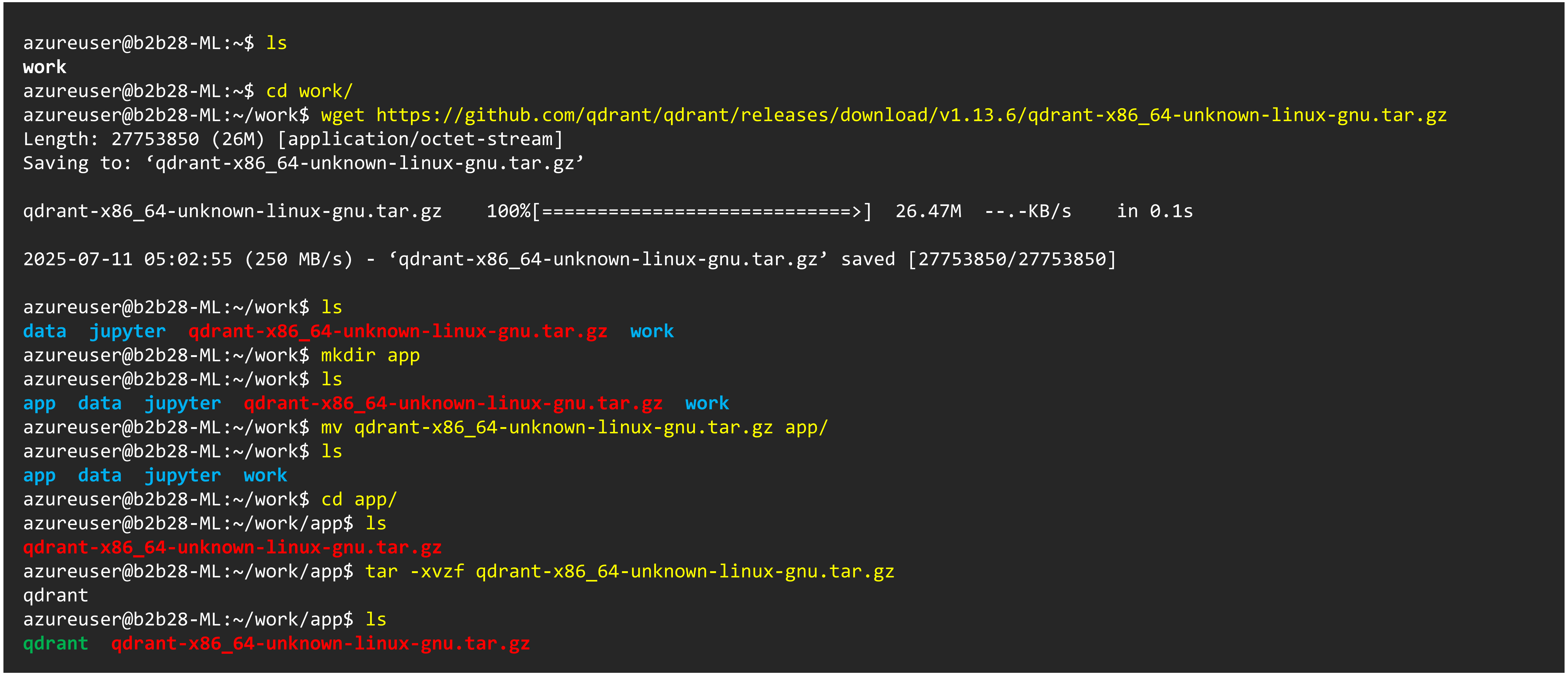
Bug fixes

- #6259 - Fix point estimation in resharding transfers, showing a more reliable ETA
- #6233 - Fix order_by not always including all values for a point if there are multiple

▾Assets 10

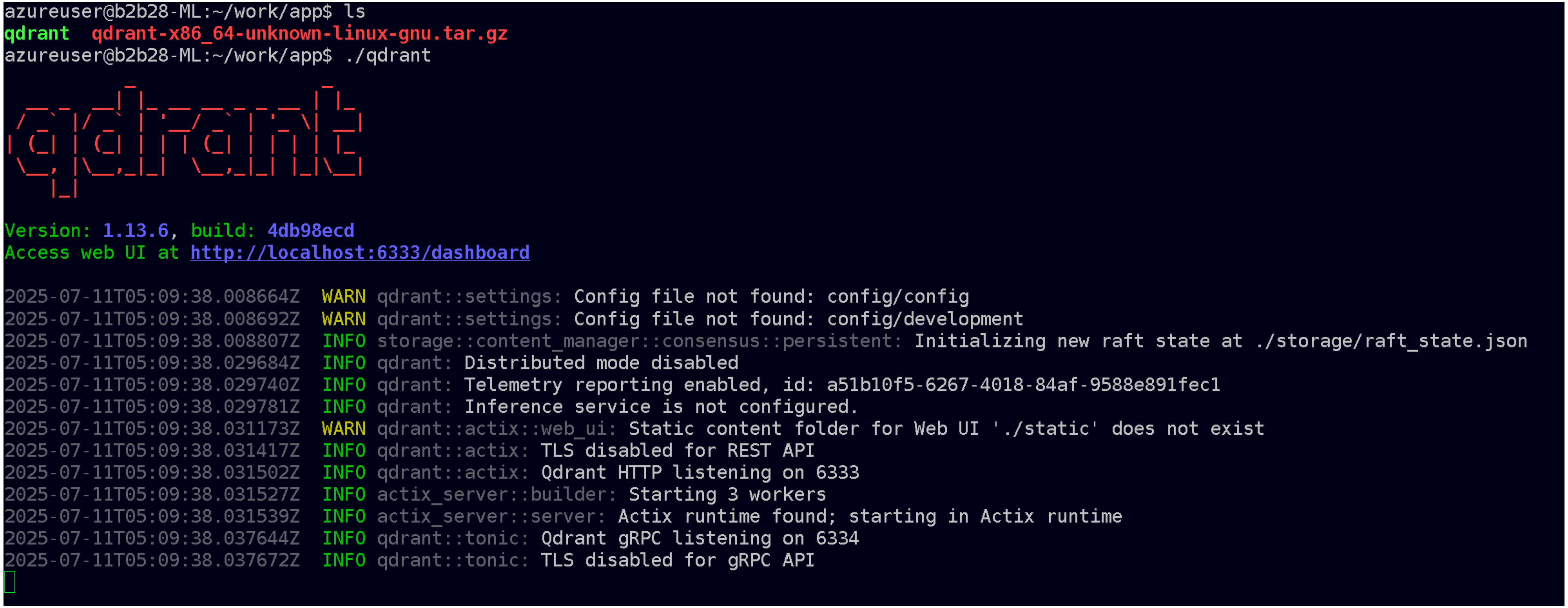
qdrant-aarch64-apple-darwin.tar.gz	23.3 MB	Apr 1
qdrant-aarch64-unknown-linux-musl.tar.gz	24.6 MB	Apr 1
qdrant-x86_64-apple-darwin.tar.gz	25.1 MB	Apr 1

Qdrant 다운로드 및 압축 해제



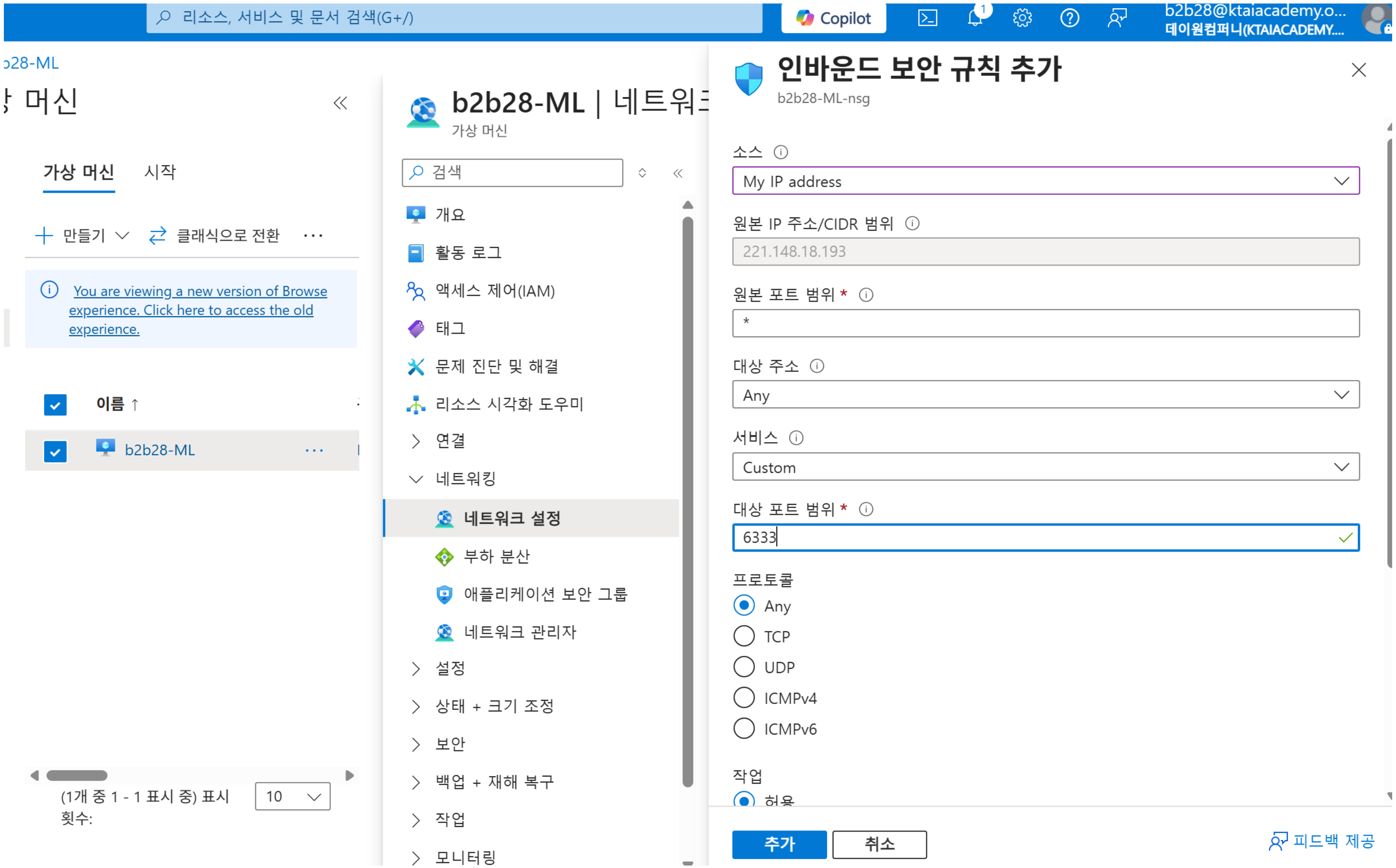
```
azureuser@b2b28-ML:~$ ls
work
azureuser@b2b28-ML:~$ cd work/
azureuser@b2b28-ML:~/work$ wget https://github.com/qdrant/qdrant/releases/download/v1.13.6/qdrant-x86_64-unknown-linux-gnu.tar.gz
Length: 27753850 (26M) [application/octet-stream]
Saving to: ‘qdrant-x86_64-unknown-linux-gnu.tar.gz’

qdrant-x86_64-unknown-linux-gnu.tar.gz    100%[=============================>]  26.47M  --.-KB/s    in 0.1s

2025-07-11 05:02:55 (250 MB/s) - ‘qdrant-x86_64-unknown-linux-gnu.tar.gz’ saved [27753850/27753850]

azureuser@b2b28-ML:~/work$ ls
data  jupyter  qdrant-x86_64-unknown-linux-gnu.tar.gz  work
azureuser@b2b28-ML:~/work$ mkdir app
azureuser@b2b28-ML:~/work$ ls
app  data  jupyter  qdrant-x86_64-unknown-linux-gnu.tar.gz  work
azureuser@b2b28-ML:~/work$ mv qdrant-x86_64-unknown-linux-gnu.tar.gz app/
azureuser@b2b28-ML:~/work$ ls
app  data  jupyter  work
azureuser@b2b28-ML:~/work$ cd app/
azureuser@b2b28-ML:~/work/app$ ls
qdrant-x86_64-unknown-linux-gnu.tar.gz
azureuser@b2b28-ML:~/work/app$ tar -xvzf qdrant-x86_64-unknown-linux-gnu.tar.gz
qdrant
azureuser@b2b28-ML:~/work/app$ ls
qdrant  qdrant-x86_64-unknown-linux-gnu.tar.gz
```

Qdrant 실행



```
azureuser@b2b28-ML:~/work/app$ ls
qdrant  qdrant-x86_64-unknown-linux-gnu.tar.gz
azureuser@b2b28-ML:~/work/app$ ./qdrant
           _                 _
  __ _  __| |_ __ __ _ _ __ | |_
 / _` |/ _` | '__/ _` | '_ \| __|
| (_| | (_| | | | (_| | | | | |_
 \__, |\__,_|_|  \__,_|_| |_|\__|
    |_|

Version: 1.13.6, build: 4db98ecd
Access web UI at http://localhost:6333/dashboard

2025-07-11T05:09:38.008664Z  WARN qdrant::settings: Config file not found: config/config
2025-07-11T05:09:38.008692Z  WARN qdrant::settings: Config file not found: config/development
2025-07-11T05:09:38.008807Z  INFO storage::content_manager::consensus::persistent: Initializing new raft state at ./storage/raft_state.json
2025-07-11T05:09:38.029684Z  INFO qdrant: Distributed mode disabled
2025-07-11T05:09:38.029740Z  INFO qdrant: Telemetry reporting enabled, id: a51b10f5-6267-4018-84af-9588e891fec1
2025-07-11T05:09:38.029781Z  INFO qdrant: Inference service is not configured.
2025-07-11T05:09:38.031173Z  WARN qdrant::actix::web_ui: Static content folder for Web UI './static' does not exist
2025-07-11T05:09:38.031417Z  INFO qdrant::actix: TLS disabled for REST API
2025-07-11T05:09:38.031502Z  INFO qdrant::actix: Qdrant HTTP listening on 6333
2025-07-11T05:09:38.031527Z  INFO actix_server::builder: Starting 3 workers
2025-07-11T05:09:38.031539Z  INFO actix_server::server: Actix runtime found; starting in Actix runtime
2025-07-11T05:09:38.037644Z  INFO qdrant::tonic: Qdrant gRPC listening on 6334
2025-07-11T05:09:38.037672Z  INFO qdrant::tonic: TLS disabled for gRPC API
```

포트 열기



소스: My IP address

원본 IP 주소/CIDR 범위: 221.148.18.193

원본 포트 범위 *: *

대상 주소: Any

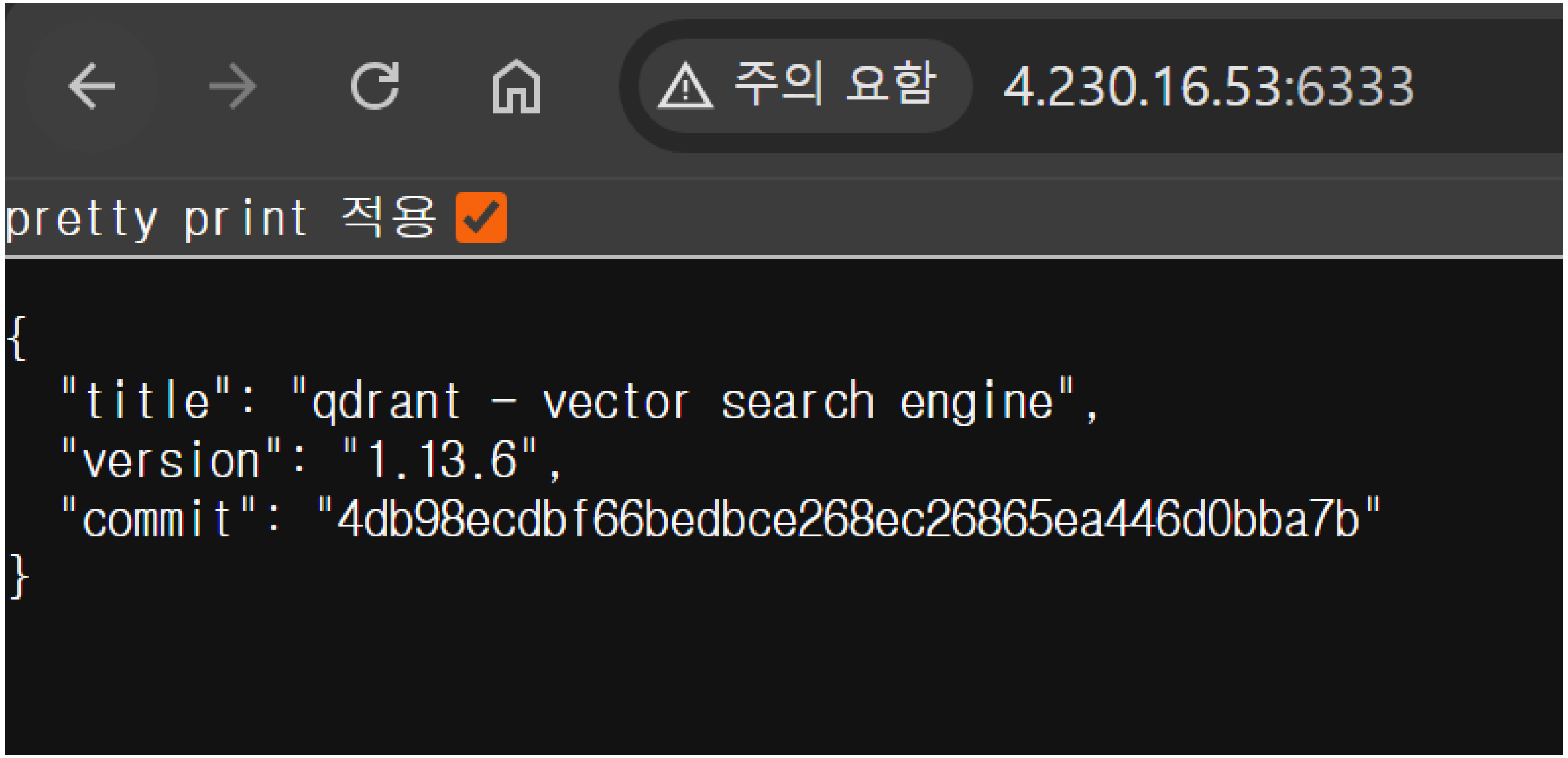
서비스: Custom

대상 포트 범위 *: 6333

프로토콜: [✓ Any] TCP UDP ICMPv4 ICMPv6

작업: [✓ 허용]

접속 확인



```
{
  "title": "qdrant - vector search engine",
  "version": "1.13.6",
  "commit": "4db98ecdbf66bedbce268ec26865ea446d0bba7b"
}
```

Qdrant 클라이언트 설치

```
azureuser@b2b28-ML:~$ pyenv activate py3_11_9

(py3_11_9) azureuser@b2b28-ML:~$ pip install qdrant-client

(py3_11_9) azureuser@b2b28-ML:~$ pip install -U langchain-qdrant
```

라이브러리 불러오기 & 연결 & 임베딩 모델

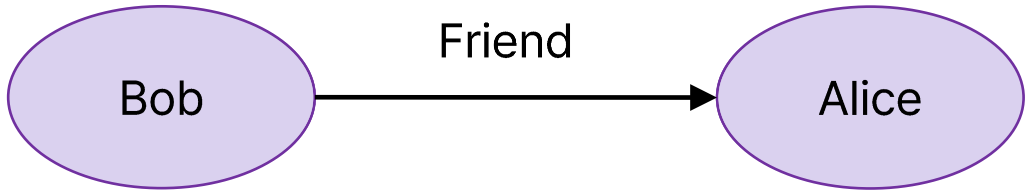
```
from qdrant_client import QdrantClient
from qdrant_client.models import Distance, VectorParams
from langchain_qdrant import QdrantVectorStore
from langchain_huggingface import HuggingFaceEmbeddings
from langchain.schema import Document
```

```
# 1. Qdrant 클라이언트 연결
client = QdrantClient(host="localhost", port=6333)
```

```
# 2. 임베딩 모델
embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2")
```

GraphDB의 개념

- 데이터 간의 관계를 중점덕으로 다루는 데이터베이스
  - 데이터를 노드(node)와 엣지(edge)라는 구조로 저장
    - 노드(node): 개체(ex: 사람, 도시, 제품 등)
    - 엣지(edge): 관계(ex: 친구, 가족, 위치, 연결 등)
    - 속성(property): 노드나 엣지의 추가정보



```
{
  "node": {
    "name": "Alice",
    "type": "Person"
  },
  "edge": {
    "type": "FRIEND",
    "since": 2015
  },
  "target": {
    "name": "Bob",
    "type": "Person"
  }
}
```

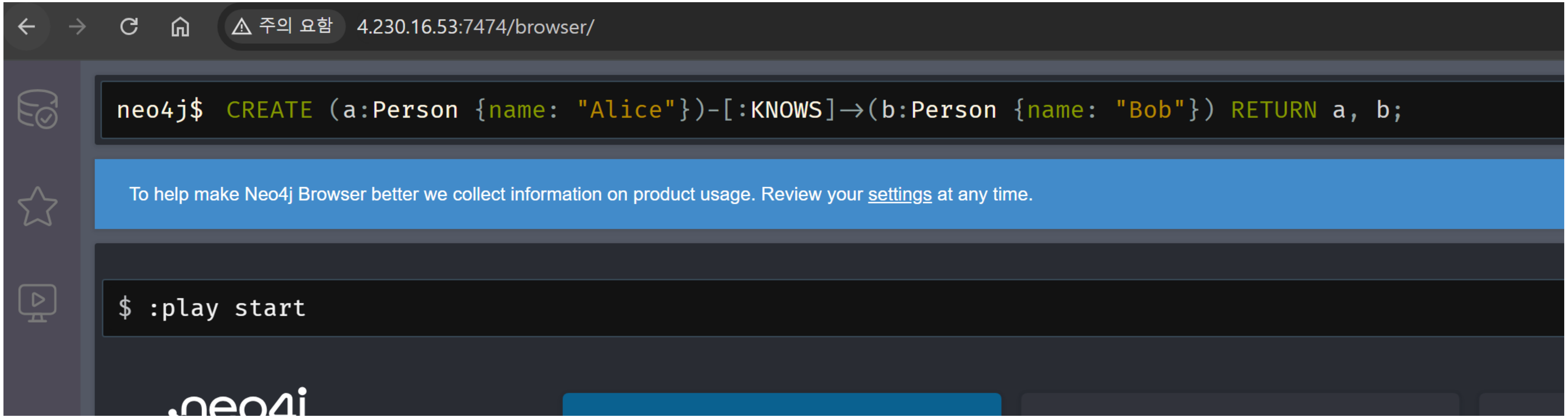
GraphDB의 필요성

- RDB(관계형 데이터베이스)의 한계
  - 테이블 조인(join)이 너무 많아져서 느림
  - 관계가 깊을수록 증가하는 복잡도
  - 유연한 스키마 설계 어려움
- GraphDB의 장점
  - 관계를 직접 표현 가능
  - 빠른 탐색 속도
  - 동적으로 확장 가능한 구조

Section

Neo4j 설치

기본 사용법



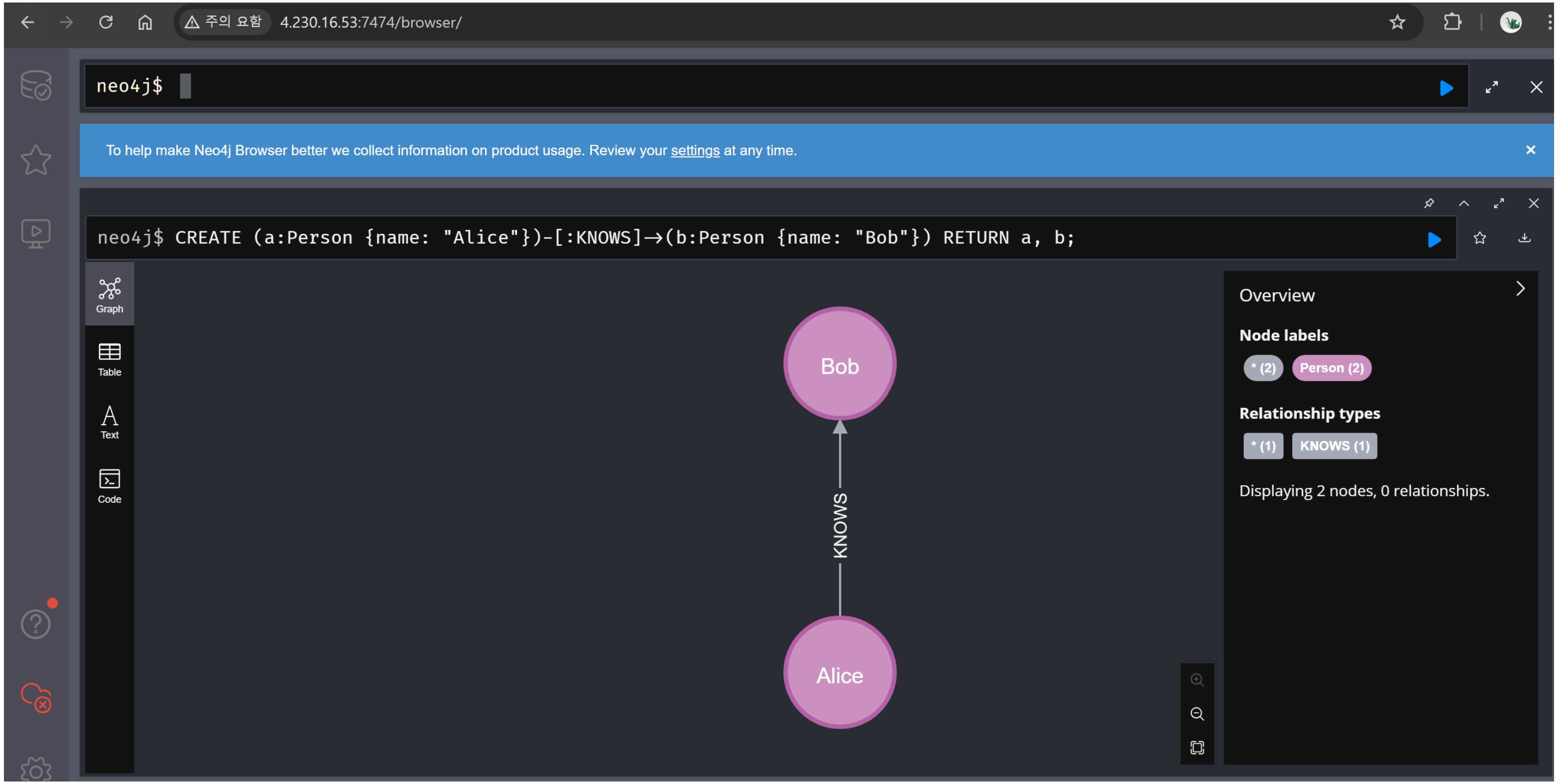
주의 요함 4.230.16.53:7474/browser/

```
neo4j$ CREATE (a:Person {name: "Alice"})-[:KNOWS]→(b:Person {name: "Bob"}) RETURN a, b;
```

To help make Neo4j Browser better we collect information on product usage. Review your settings at any time.

```
$ :play start
```

기본 사용법



```
neo4j$ CREATE (a:Person {name: "Alice"})-[:KNOWS]→(b:Person {name: "Bob"}) RETURN a, b;
```

To help make Neo4j Browser better we collect information on product usage. Review your settings at any time.

Overview

Node labels

* (2) Person (2)

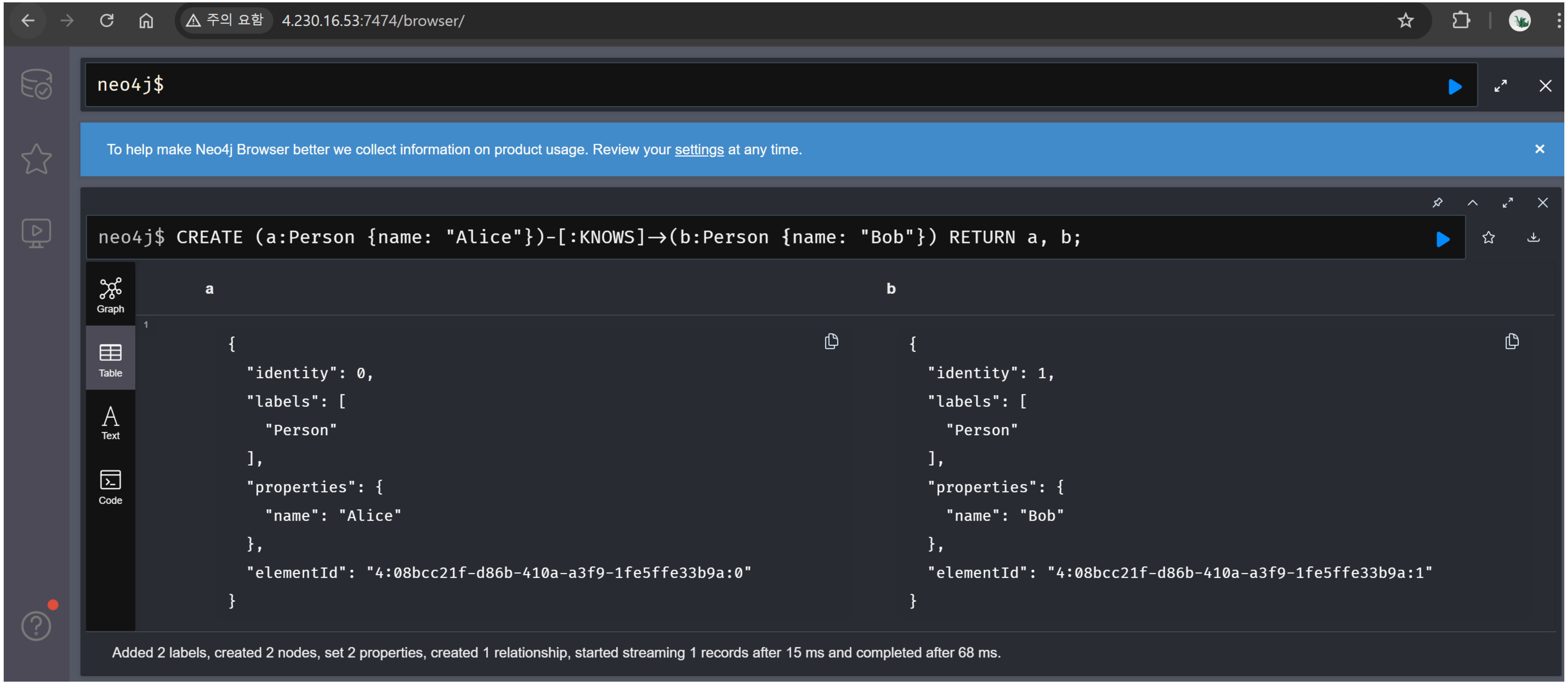
Relationship types

* (1) KNOWS (1)

Displaying 2 nodes, 0 relationships.

Section
Neo4j 설치

기본 사용법



```
neo4j$
```

To help make Neo4j Browser better we collect information on product usage. Review your settings at any time.

```
neo4j$ CREATE (a:Person {name: "Alice"})-[:KNOWS]→(b:Person {name: "Bob"}) RETURN a, b;
```

```
a                                                    b
1
{                                                    {
  "identity": 0,                                       "identity": 1,
  "labels": [                                          "labels": [
    "Person"                                             "Person"
  ],                                                   ],
  "properties": {                                      "properties": {
    "name": "Alice"                                      "name": "Bob"
  },                                                   },
  "elementId": "4:08bcc21f-d86b-410a-a3f9-1fe5ffe33b9a:0"   "elementId": "4:08bcc21f-d86b-410a-a3f9-1fe5ffe33b9a:1"
}                                                    }
```

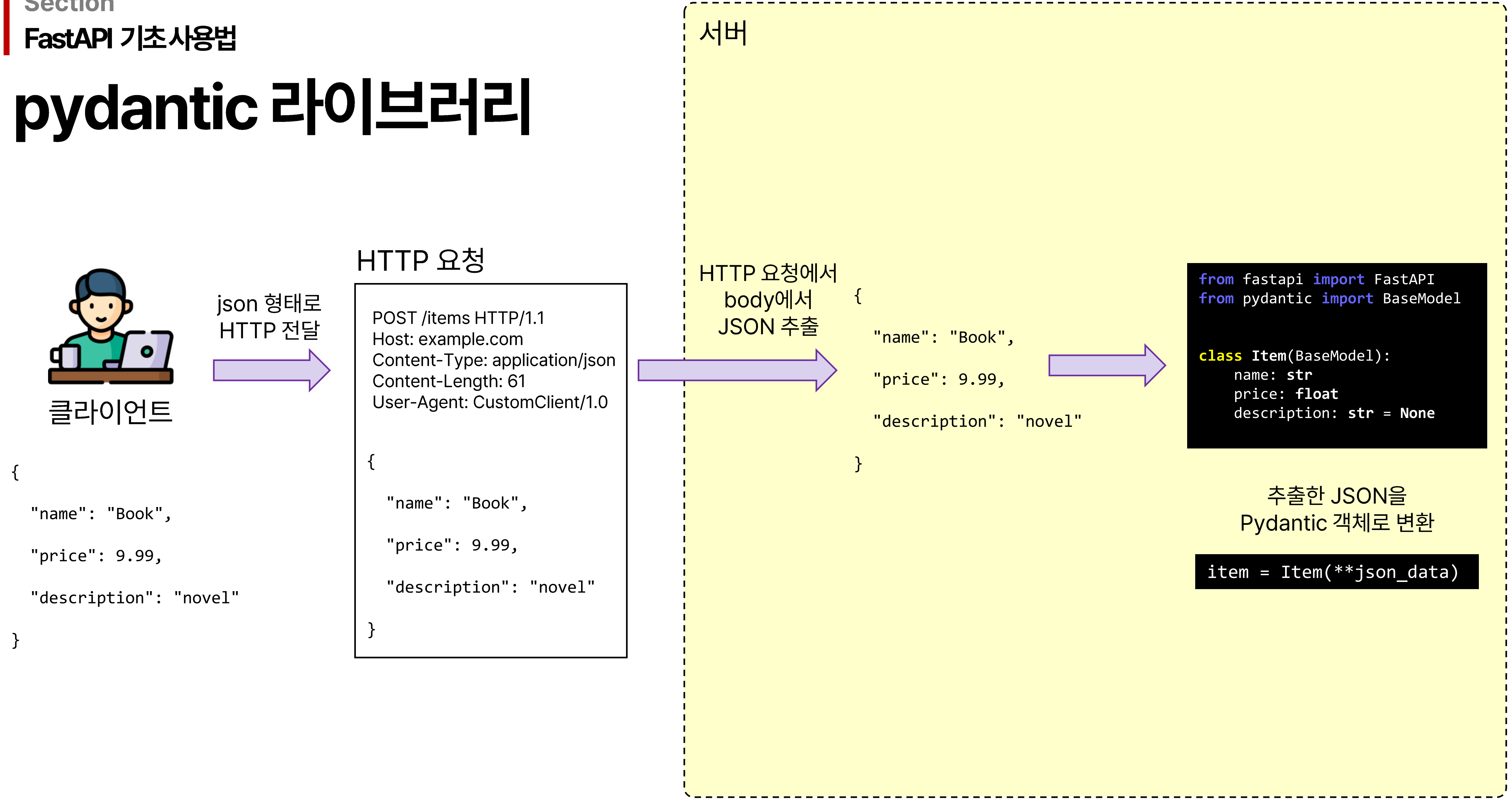
Added 2 labels, created 2 nodes, set 2 properties, created 1 relationship, started streaming 1 records after 15 ms and completed after 68 ms.

GEN AI 인텐시브 과정

Section 1. 실무 적용

Section 1-3. 실제 서비스

pydantic 라이브러리



```
{
  "name": "Book",
  "price": 9.99,
  "description": "novel"
}
```

```
POST /items HTTP/1.1
Host: example.com
Content-Type: application/json
Content-Length: 61
User-Agent: CustomClient/1.0

{
  "name": "Book",
  "price": 9.99,
  "description": "novel"
}
```

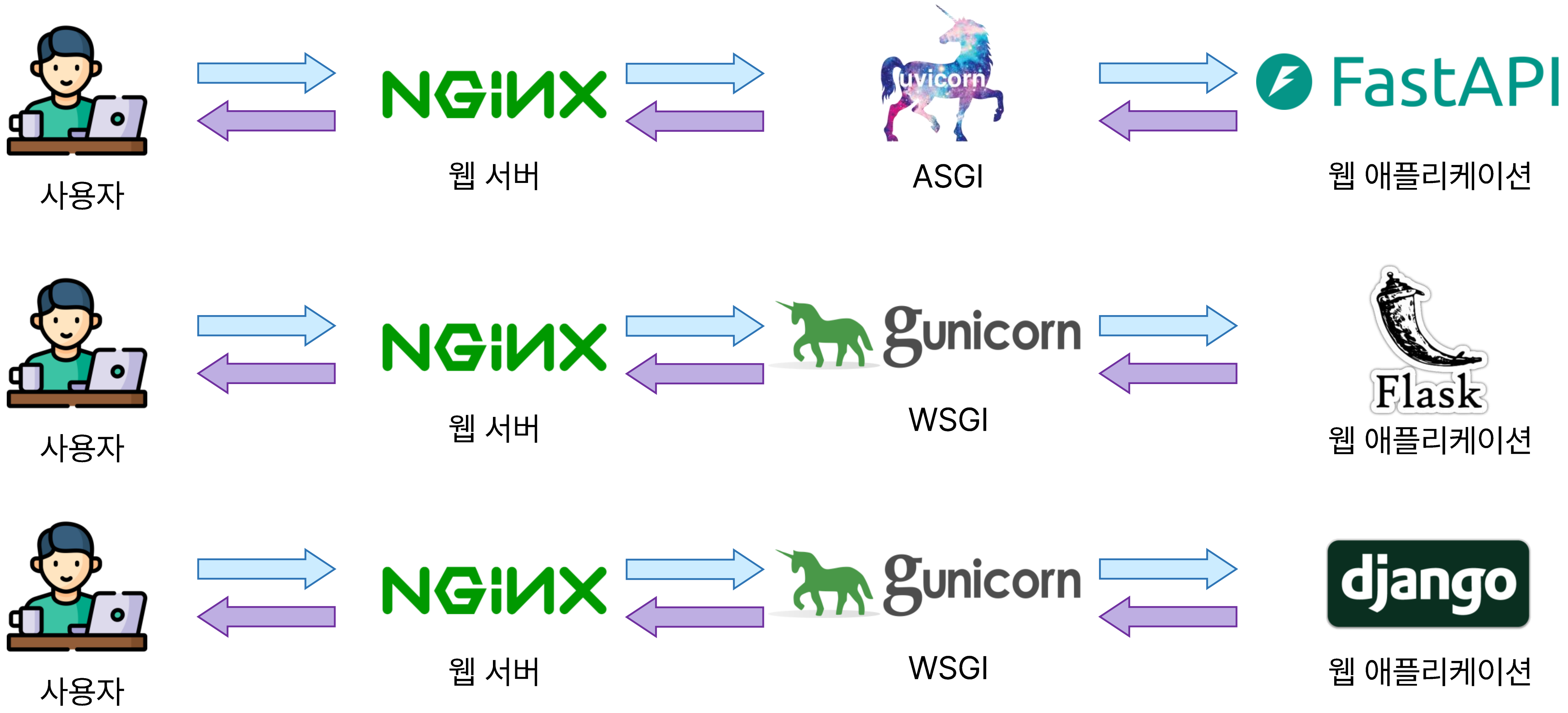
```
{
  "name": "Book",
  "price": 9.99,
  "description": "novel"
}
```

```python
from fastapi import FastAPI
from pydantic import BaseModel


class Item(BaseModel):
    name: str
    price: float
    description: str = None
```

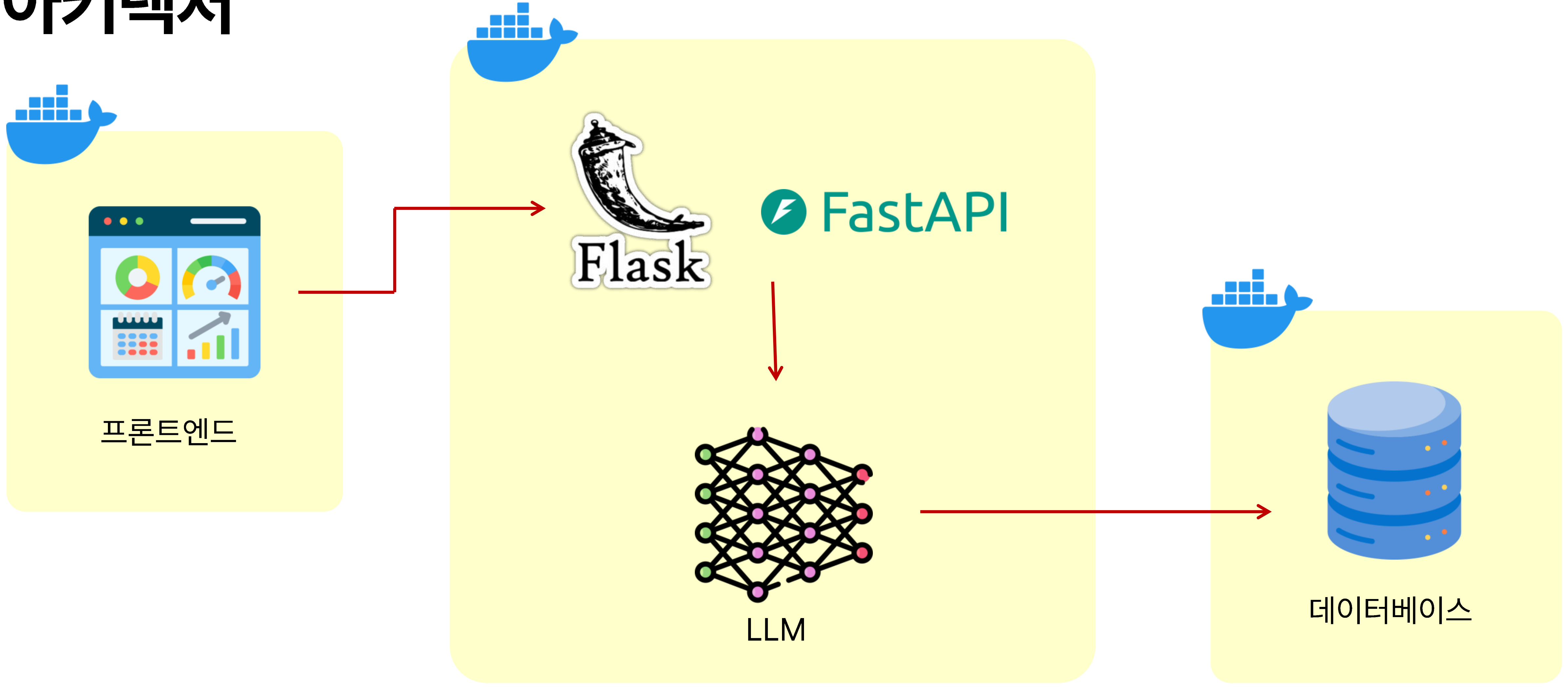
```python
item = Item(**json_data)
```

ASGI vs WSGI

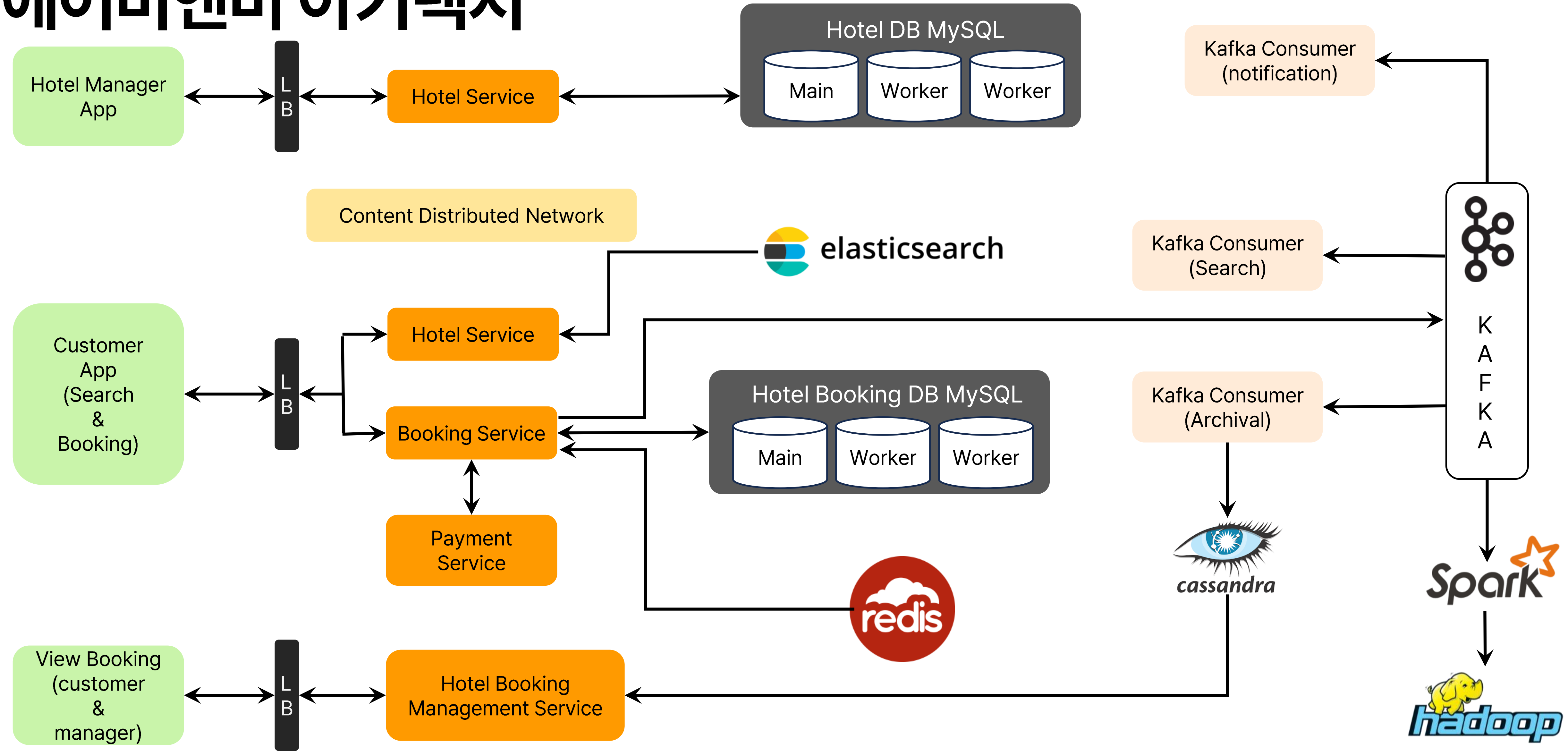
ASGI는 비동기 지원



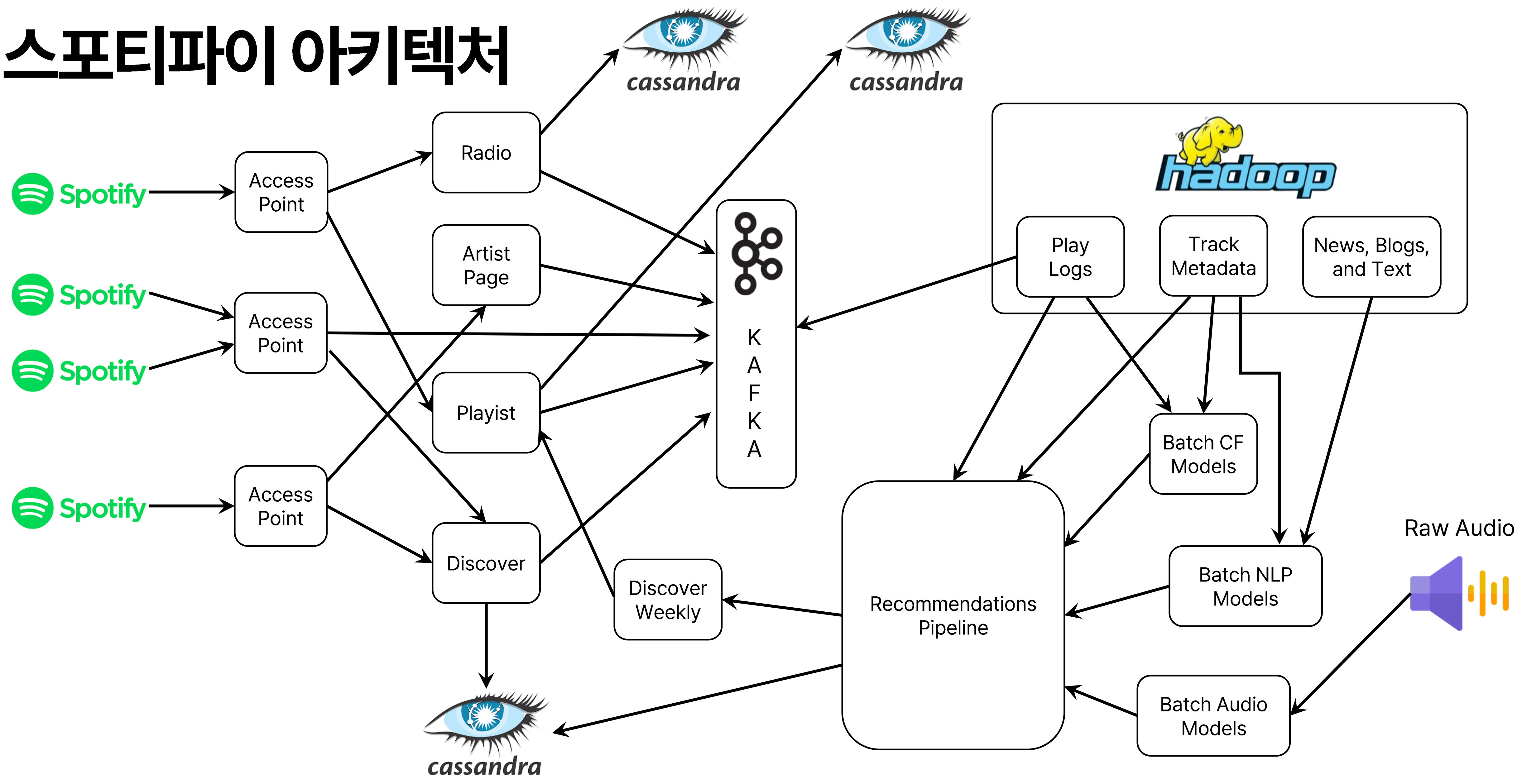
아키텍처



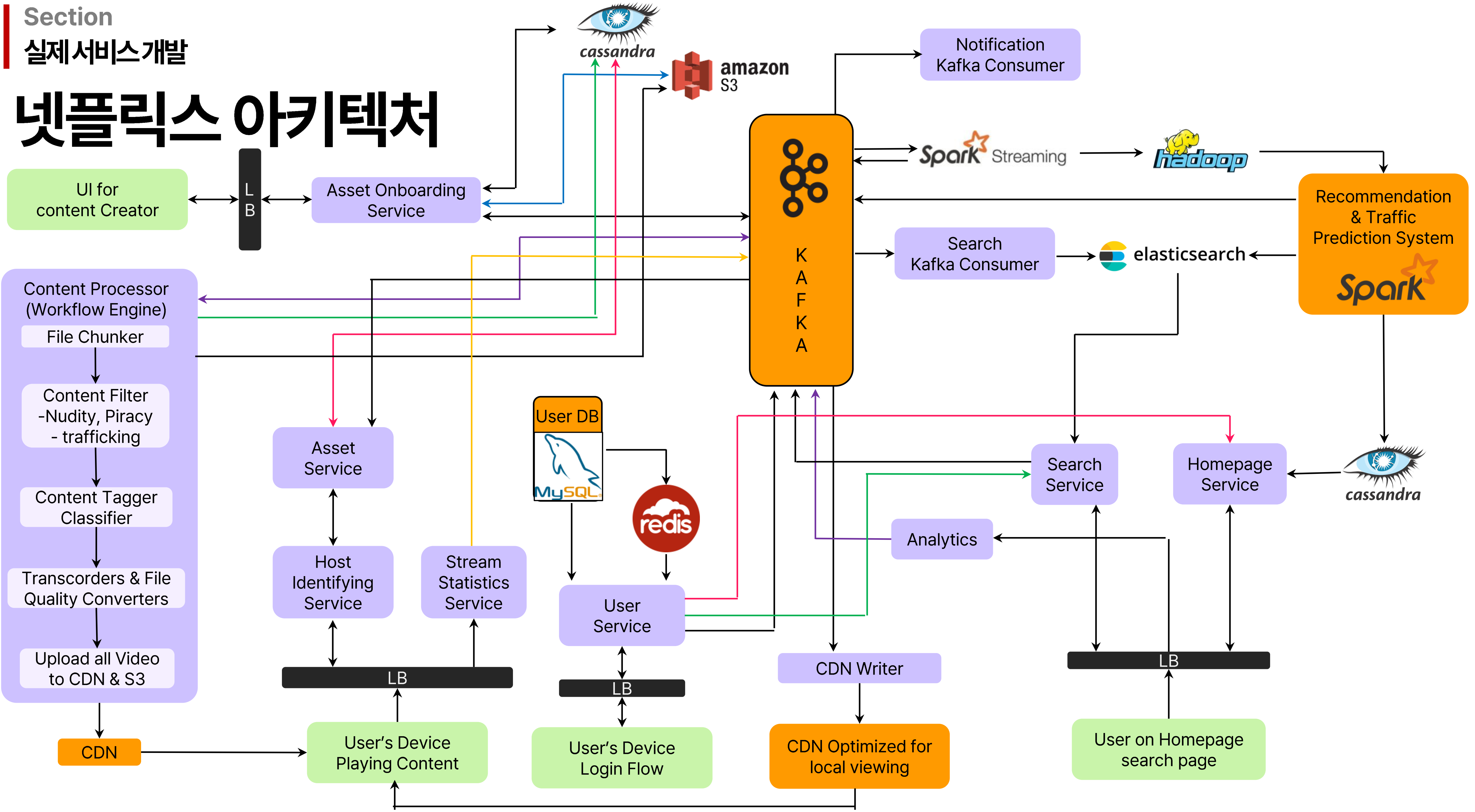
에어비앤비 아키텍처


Hotel Manager App
LB
Hotel Service
Hotel DB MySQL
Main
Worker
Worker
Kafka Consumer (notification)
Content Distributed Network
elasticsearch
Kafka Consumer (Search)
Customer App (Search & Booking)
LB
Hotel Service
Booking Service
Hotel Booking DB MySQL
Main
Worker
Worker
KAFKA
Kafka Consumer (Archival)
Payment Service
redis
cassandra
Spark
View Booking (customer & manager)
LB
Hotel Booking Management Service
hadoop

스포티파이 아키텍처


cassandra
cassandra
Spotify
Spotify
Spotify
Spotify
Access Point
Access Point
Access Point
Radio
Artist Page
Playist
Discover
KAFKA
hadoop
Play Logs
Track Metadata
News, Blogs, and Text
Batch CF Models
Batch NLP Models
Batch Audio Models
Raw Audio
Recommendations Pipeline
Discover Weekly
cassandra

넷플릭스 아키텍처


UI for content Creator
L B
Asset Onboarding Service
cassandra
amazon S3
Notification Kafka Consumer
KAFKA
Spark Streaming
hadoop
Recommendation & Traffic Prediction System
Spark
Search Kafka Consumer
elasticsearch
Content Processor (Workflow Engine)
File Chunker
Content Filter -Nudity, Piracy - trafficking
Content Tagger Classifier
Transcoders & File Quality Converters
Upload all Video to CDN & S3
CDN
Asset Service
Host Identifying Service
Stream Statistics Service
LB
User's Device Playing Content
User DB
MySQL
redis
User Service
LB
User's Device Login Flow
CDN Writer
CDN Optimized for local viewing
Search Service
Homepage Service
cassandra
Analytics
LB
User on Homepage search page

감사합니다.

Q & A