

스프링AI 와 Ollama 를 활용한 나만의 AI 크래시 원인 분석기 구성하기

지난 수십년 동안 컴퓨터에서 음성과 작성된 언어를 보고 이해 하고 번역하고 분석하는 등의 기능을 개발하기 위해서 다양한 접근과 시도가 있었다. 최근 딥 러닝(Deep learning) 알고리즘 중 대규모 언어 모델(LLM) 인 Chat-GPT가 사람들의 관심을 받고 있습니다. 사람들의 관심 속에 다양한 LLM 모델들이 공개되고 있습니다. 과거에도 구글이 GFS(Google File System) 와 맵리듀스(MapReduce) 를 논문으로 발표하고 나서 오픈 소스 소프트웨어로 Hadoop 이 개발된 것처럼 이번에도 오픈소스 LLM 들이 공개되고 있습니다.

그 중에서 메타가 공개한 Llama 가 가장 빠르고 광범위하게 사용될 수 있도록 제공되고 있습니다. 하지만, 모델 공개되고 누구나 사용가능하다고 하지만 저 같은 자바 개발자에게는 그림의 떡 입니다. 사용해보고 싶지만 어떻게 해야 할지 막막합니다. 그렇다고 지금에 와서 pytorch 를 배우고 모델 개발 및 사용법에 대해 공부한다는 것은 더 큰 부담입니다. 이런 고민을 저만한 것이 아니었습니다. 제가 즐겨 사용하는 스프링 프레임워크 내부에서도 기존 스프링 기반 어플리케이션이 좀더 쉽게 AI 기술을 사용할 수 있도록 도와줄 프레임워크를 개발하고 있었습니다. Spring AI 는 기존 모델과 통신할 수 있고 데이터를 주고 받기 쉽도록 추상화 단계를 제공하고 있습니다.

제가 공개 모델을 접하게 되었을 때도 그 모델과 Rest API 로 통신할 수 있는 가상의 계층을 구성할 수 있다면 기존의 자바 어플리케이션들도 필요한 부분에 대해서 LLM 의 활용한 서비스를 개발 할 수 있을 것이라 생각했습니다. 모델과 통신할 수 있는 계층을 만들고 Docker 이미지로 구성하는 것은 개발자 개인 해결하기에는 어려운 문제였습니다. 일반 개발자들은 대부분 GPU 를 활용할 수 있는 서버에 접근이 어렵고 범용적인 VM 내에 구성하는 것도 쉽지 않은 일입니다. 하지만, 이 어려움에 헤어나지 못할 때 스프링AI 를 통해서 Ollama 를 알게 되었습니다.

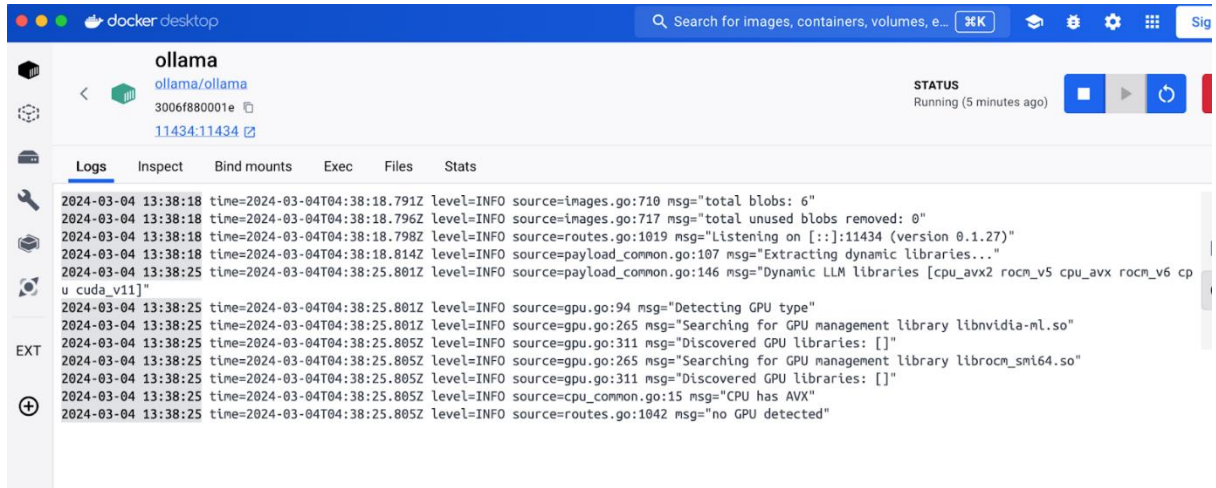
Ollama 는 여러 오픈소스 LLM 을 사용할 수 있게 도와주는 추상화 통신 계층 서비스 제공하는 오픈소스 프로젝트입니다. Ollama 에서 제공하는 프로그램을 사용하면 개발자 PC 에서 LLM 을 사용해 볼 수 있습니다. 또한 Docker 이미지로도 제공하기 때문에 쿠버네티스 환경에서도 쉽게 실행해 볼 수 있습니다.

제가 맡고 있는 크래시리포트는 게임 서비스 중에 비정상 종료되었을 때는 데이터를 수집하고 분석을 위해서 그 데이터를 제공하는 것이 목적입니다. 하지만 시스템 내에서 원인 분석을 위한 서비스가 없습니다. 그렇다 보니, AI를 통한 개별 크래시 데이터의 원인 분석을 시도해 본다면 어떤 결과가 나올지 궁금 했었습니다. 이제는 실행해 볼 수 있는 환경이 되었습니다. 스프링AI 와 Ollama 를 통해서 개발 환경을 구성하고 검증할 수 있는 환경을 만들었습니다.

Ollama는 도커이미지 파일을 제공하고 있습니다. CPU only 명령어로 로컬에서 Ollama 도커 이미지를 실행해보겠습니다. Nvidia GPU 를 사용하여 실행 할 수 있으니, 이 부분도 참고해주세요.

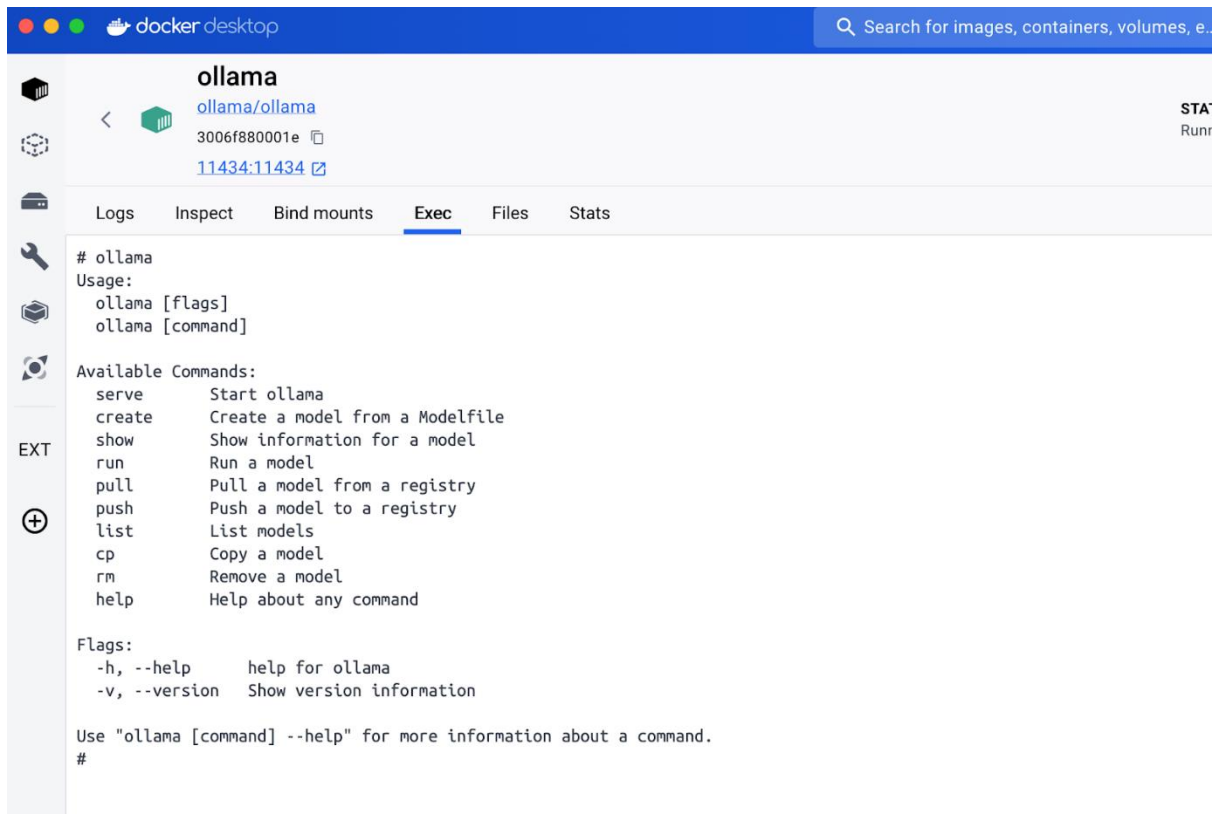
```
docker run -d -v ollama:/root/.ollama -p 11434:11434 --name ollama ollama/ollama
```

Ollama 실행 결과



The screenshot shows the Docker Desktop interface with the 'ollama' container selected. The 'Logs' tab is active, displaying a series of log messages from the container. The messages indicate the container is listening on port 11434, extracting dynamic libraries, and detecting GPU type. The status bar shows the container is 'Running (5 minutes ago)'.

```
2024-03-04 13:38:18 time=2024-03-04T04:38:18.791Z level=INFO source=images.go:710 msg="total blobs: 6"
2024-03-04 13:38:18 time=2024-03-04T04:38:18.796Z level=INFO source=images.go:717 msg="total unused blobs removed: 0"
2024-03-04 13:38:18 time=2024-03-04T04:38:18.798Z level=INFO source=routes.go:1019 msg="Listening on [::]:11434 (version 0.1.27)"
2024-03-04 13:38:18 time=2024-03-04T04:38:18.814Z level=INFO source=payload_common.go:107 msg="Extracting dynamic libraries..."
2024-03-04 13:38:25 time=2024-03-04T04:38:25.801Z level=INFO source=payload_common.go:146 msg="Dynamic LLM libraries [cpu_avx2 rocm_v5 cpu_avx rocm_v6 cp
u cuda_v11]"
2024-03-04 13:38:25 time=2024-03-04T04:38:25.801Z level=INFO source=gpu.go:94 msg="Detecting GPU type"
2024-03-04 13:38:25 time=2024-03-04T04:38:25.801Z level=INFO source=gpu.go:265 msg="Searching for GPU management library libnvidia-ml.so"
2024-03-04 13:38:25 time=2024-03-04T04:38:25.805Z level=INFO source=gpu.go:311 msg="Discovered GPU libraries: []"
2024-03-04 13:38:25 time=2024-03-04T04:38:25.805Z level=INFO source=gpu.go:265 msg="Searching for GPU management library librocm-smi64.so"
2024-03-04 13:38:25 time=2024-03-04T04:38:25.805Z level=INFO source=gpu.go:311 msg="Discovered GPU libraries: []"
2024-03-04 13:38:25 time=2024-03-04T04:38:25.805Z level=INFO source=cpu_common.go:15 msg="CPU has AVX"
2024-03-04 13:38:25 time=2024-03-04T04:38:25.805Z level=INFO source=routes.go:1042 msg="no GPU detected"
```



The screenshot shows the Docker Desktop interface with the 'ollama' container selected. The 'Exec' tab is active, displaying the command prompt for the container. The prompt shows the usage of the 'ollama' command and lists available commands and flags.

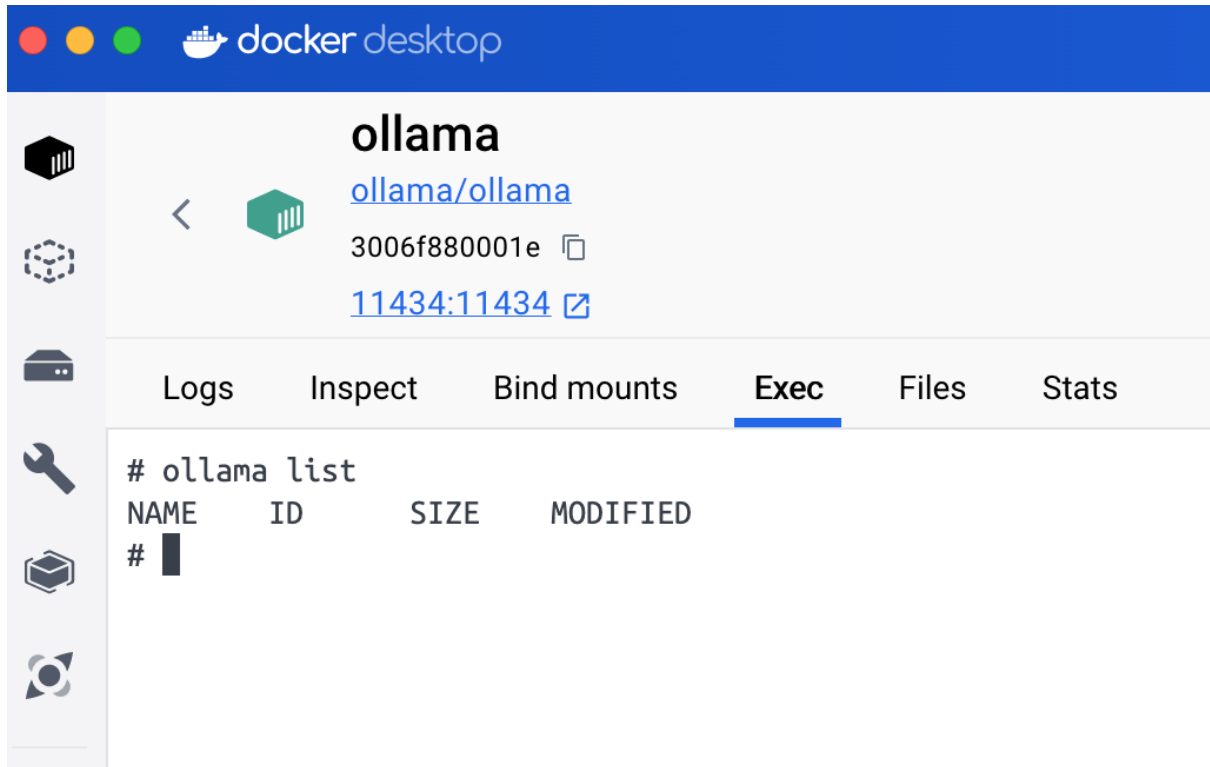
```
# ollama
Usage:
  ollama [flags]
  ollama [command]

Available Commands:
  serve      Start ollama
  create     Create a model from a Modelfile
  show       Show information for a model
  run        Run a model
  pull       Pull a model from a registry
  push       Push a model to a registry
  list       List models
  cp         Copy a model
  rm         Remove a model
  help       Help about any command

Flags:
  -h, --help      help for ollama
  -v, --version    Show version information

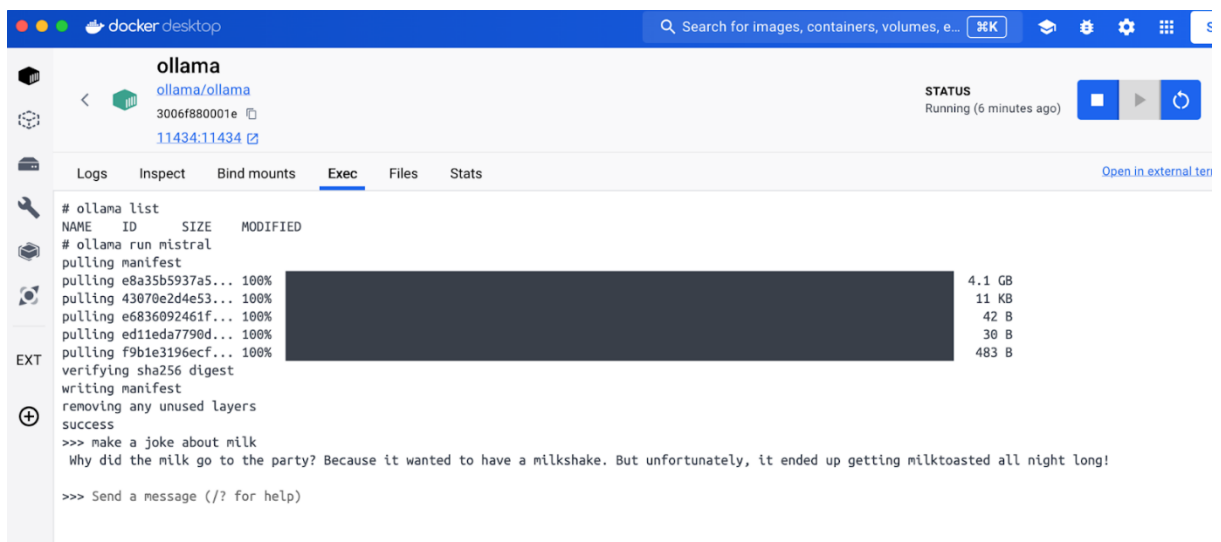
Use "ollama [command] --help" for more information about a command.
#
```

최초 실행 후 ollama list 명령어를 실행하면 저장된 모델이 없음을 확인 할 수 있습니다.

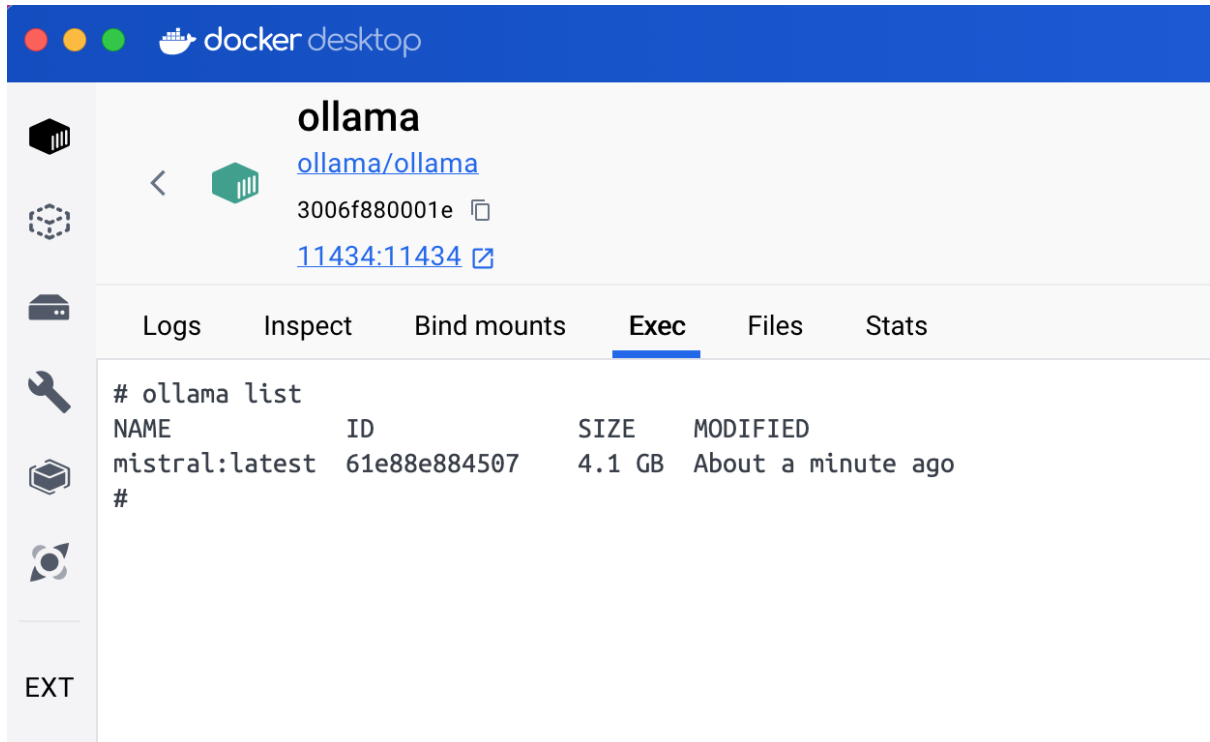


다음 명령어를 통해서 mistral 모델을 설치하고 실행 해보겠습니다.

```
ollama run mistral
```

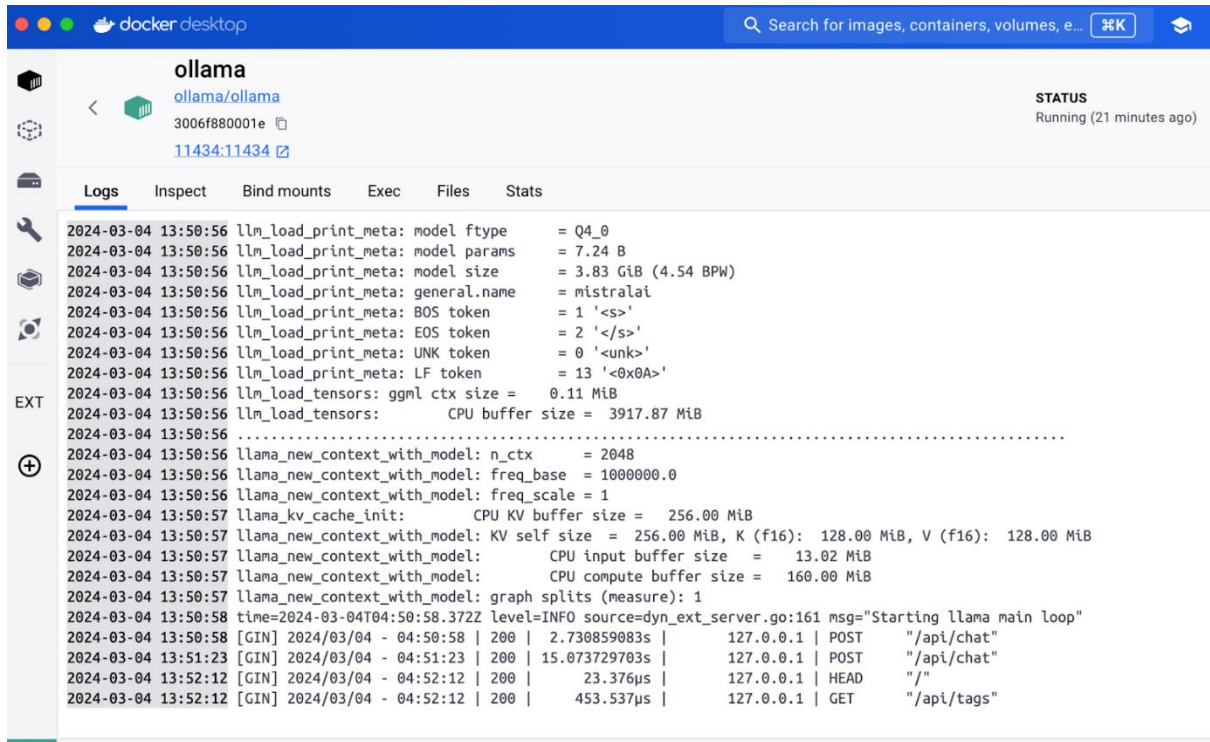


위와 같이 실행된 후 간단한 질의를 한 결과가 표시되었습니다. 또한 Ollama list 를 통해서 저장된 모델을 확인 할 수 있습니다.



Ollama 에서 제공되는 모델은 Llama2, Mistral, Dolphin Phi 등 15개 이상의 모델들을 사용해 볼 수 있습니다. 각 모델들의 특성을 고려하여 본인의 서비스 목적에 맞도록 선택할 수 있습니다. 더 나아가 이렇게 제공되는 무료 모델들은 수정도 가능하기에 생성형 AI 를 활용한 서비스 개발이 점점 쉬워지고 있습니다.

테스트 질의 후 Ollama 의 실행 로그를 보면 REST api 로그가 출력 된 것을 확인할 수 있습니다. Ollama 의 REST API 규격을 통해서 외부 어플리케이션도 쉽게 LLM 모델을 활용한 서비스를 만들어 볼 수 있습니다. 이때 Spring AI 에서 이 규격을 맞춘 기능을 제공하고 있습니다.



스프링AI 를 사용하면 Ollama 의 REST API 규격을 이해하는 OllamaChatClient 클래스를 제공합니다. 이외에도 다양한 AI 모델 서비스와 통신이 가능한 기능을 제공하고 있으니, 공식 문서를 참고해주세요. 제공된 클래스를 통해서 개발자는 쉽게 데이터를 주고 받을 수 있습니다.

```
@RestController
public class ChatController {

    private final OllamaChatClient chatClient;

    @Autowired
    public ChatController(OllamaChatClient chatClient) {
        this.chatClient = chatClient;
    }

    @GetMapping("/ai/generate")
    public Map generate(@RequestParam(value = "message", defaultValue = "Tell me a joke") String message) {
        return Map.of("generation", chatClient.call(message));
    }

    @GetMapping("/ai/generateStream")
    public Flux<ChatResponse> generateStream(@RequestParam(value = "message", defaultValue = "Tell me a joke") String message) {
        Prompt prompt = new Prompt(new UserMessage(message));
        return chatClient.stream(prompt);
    }
}
```

크래시리포트에서는 스프링AI 와 Ollama 를 활용하여 AI 진단 기능 서비스를 구성해 보았습니다.

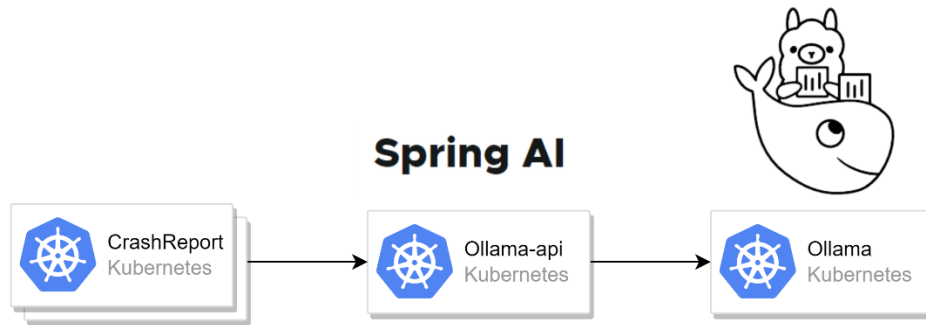


그림 과 같이 스프링AI 를 이용해서 Ollama 서비스와 연결되는 API 서버를 개발하였습니다. 이로써 사용자 화면에서 크래시 분석을 요청하면 LLM 모델까지 전달 될 수 있는 경로가 구성된 것입니다.

참고자료

1. <https://llama.meta.com/llama2>
2. <https://ollama.com/>
3. <https://ollama.com/library>
4. <https://github.com/ollama/ollama>
5. <https://hub.docker.com/r/ollama/ollama>
6. <https://docs.spring.io/spring-ai/reference/index.html>
7. <https://docs.spring.io/spring-ai/reference/api/clients/ollama-chat.html>
8. <https://docs.mistral.ai/guides/model-selection/>