

Logcat APP/Tool

2025.02.04

모바일시스템공학과 조민혁

소프트웨어학과 이승민, 정성원, 정연수

INDEX

01

Logcat App/Tool

02

APP 향후 계획

03

Spring Boot/Docker 기반 백엔드 구성

04

NAS 서버 설정 및 배포

05

서버 향후 계획



01

Logcat App/Tool

❖무엇을 해야하는가?

① Anti-Forensic 행위 인지 및 로그 생성 - OK

- Timestamp 조작, logcat -c, 전원 끄기 및 재부팅, File 메타데이터 조작

② APP 관련 로그 생성 - OK

- 전화 앱, SMS 앱, 블루투스

③ 서버로 보내기 - OK

- NAS? ICT관 3층 서버? 클라우드?

④ 자동차 관련 앱 및 통신 적용 - 진행중

- 스마트폰과 자동차와의 통신을 중심으로 .. + 자동차에서 발생하는 로그
- 블루링크, 내비게이션 APP (네이버 지도, 카카오맵, TMap), Android Auto App

⑤ 커널 level에서 작동시키기

- eBPF .. APP을 명시적으로 실행하는 것이 아닌 디폴트로 시작되도록 작동시키는 것이 최종 목표

❖ 자동차 관련하여 해야할 것 - Navigation App

1. 어플 실행 여부 및 어떤 내비게이션 앱인지? (Android Auto 정보 포함)
2. 스마트폰과 연결된 자동차 정보
3. 버튼 클릭 정보 (경로 취소 포함)
4. 출발지-목적지 정보 -- Timestamp 포함
5. 가능하다면, 설정한 경로(목적지) 운행 중에, 음성 인식이나 음성 안내 정보, 또는 이동 경로 정보
6. 가능하다면, 다음 이벤트 정보 -- USB/Bluetooth 연결, GPS Time Syncs, 속도 데이터

❖ 어플 실행 여부 및 어떤 내비게이션 앱인지?

현재 포어그라운드 앱: `com.nhn.android.nmap`
네이버 지도 앱이 포어그라운드에 있습니다.



앱이 백그라운드로 전환되었습니다.

< 네이버 지도 >

현재 포어그라운드 앱: `net.daum.android.map`
카카오맵 앱이 포어그라운드에 있습니다.



앱이 백그라운드로 전환되었습니다.

< 카카오맵 >

현재 포어그라운드 앱: `com.skt.tmap.ku`
TMap 앱이 포어그라운드에 있습니다.



앱이 백그라운드로 전환되었습니다.

< Tmap >

- 포어그라운드/백그라운드 여부에 따른 앱 실행/종료 여부 추적 가능
- 패키지명을 통한 실행되는 앱 식별 가능

❖ 버튼 클릭 정보

➤ 3가지의 CASE로 나뉨

CASE 1) 클릭한 요소에 대한 정보가 모두 선명하게 나오는 경우

CASE 2) 클릭한 요소에 대한 텍스트 정보만 나오지 않는 경우 (좌표로 식별 가능)

CASE 3) 클릭한 요소에 대한 식별 가능한 정보가 나오지 않는 경우

❖ 버튼 클릭 정보 - CASE 1) 클릭한 요소에 대한 정보가 모두 선명하게 나오는 경우

```
Text: 내비게이션 탭
Content Description: 내비게이션 탭
Class Name: android.view.ViewGroup
Clickable: true, Enabled: true, Focusable: true
Bounds in Screen: [648,2049][864,2256]
뷰 위치: (648, 2049)
뷰 크기: 216 x 207
Margin: left=648, top=2049, right=216, bottom=144
Bounds in Parent: [0,0][216,207]
클릭 이벤트 절대 좌표 (중앙 값): X=756, Y=2152
Width=216, Height=207
```

```
클릭한 UI 요소 정보:
Text: 대중교통 탭
Content Description: 대중교통 탭
Class Name: android.view.ViewGroup
Clickable: true, Enabled: true, Focusable: true
Bounds in Screen: [432,2049][648,2256]
뷰 위치: (432, 2049)
뷰 크기: 216 x 207
Margin: left=432, top=2049, right=432, bottom=144
Bounds in Parent: [0,0][216,207]
클릭 이벤트 절대 좌표 (중앙 값): X=540, Y=2152
Width=216, Height=207
```

```
클릭한 UI 요소 정보:
Text: 주변 탭
Content Description: 주변 탭
Class Name: android.view.ViewGroup
Clickable: true, Enabled: true, Focusable: true
Bounds in Screen: [0,2049][216,2256]
뷰 위치: (0, 2049)
뷰 크기: 216 x 207
Margin: left=0, top=2049, right=864, bottom=144
Bounds in Parent: [0,0][216,207]
클릭 이벤트 절대 좌표 (중앙 값): X=108, Y=2152
Width=216, Height=207
```

- 각 버튼 클릭할 때마다 식별 잘됨
- 이 경우 문제 x

❖ 버튼 클릭 정보 - CASE 2) 클릭한 요소에 대한 텍스트 정보만 나오지 않는 경우 (좌표로 식별 가능)

클릭한 UI 요소 정보:

```
Class Name: android.widget.ImageView
Clickable: true, Enabled: true, Focusable: true
Bounds in Screen: [900,2094][1008,2202]
뷰 위치: (900, 2094)
뷰 크기: 108 x 108
Margin: left=900, top=2094, right=72, bottom=198
Bounds in Parent: [0,0][108,108]
클릭 이벤트 절대 좌표 (중앙 값): X=954, Y=2148
Width=108, Height=108
```

클릭한 UI 요소 정보:

```
Class Name: android.widget.ImageView
Clickable: true, Enabled: true, Focusable: true
Bounds in Screen: [948,2093][1080,2225]
뷰 위치: (948, 2093)
뷰 크기: 132 x 132
Margin: left=948, top=2093, right=0, bottom=175
Bounds in Parent: [0,0][132,132]
클릭 이벤트 절대 좌표 (중앙 값): X=1014, Y=2159
Width=132, Height=132
```

- 텍스트 정보는 따로 출력 안되고 좌표만 서로 다르게 나옴
- 이 경우 좌표를 통해 식별 가능

❖ 버튼 클릭 정보 - CASE 3) 클릭한 요소에 대한 식별 가능한 정보가 나오지 않는 경우

클릭한 UI 요소 정보:

```
Class Name: android.widget.FrameLayout
Clickable: false, Enabled: true, Focusable: false
Bounds in Screen: [0,0][1080,2400]
뷰 위치: (0, 0)
뷰 크기: 1080 x 2400
Margin: left=0, top=0, right=0, bottom=0
Bounds in Parent: [0,0][1080,2400]
클릭 이벤트 절대 좌표 (중앙 값): X=540, Y=1200
Width=1080, Height=2400
```

클릭한 UI 요소 정보:

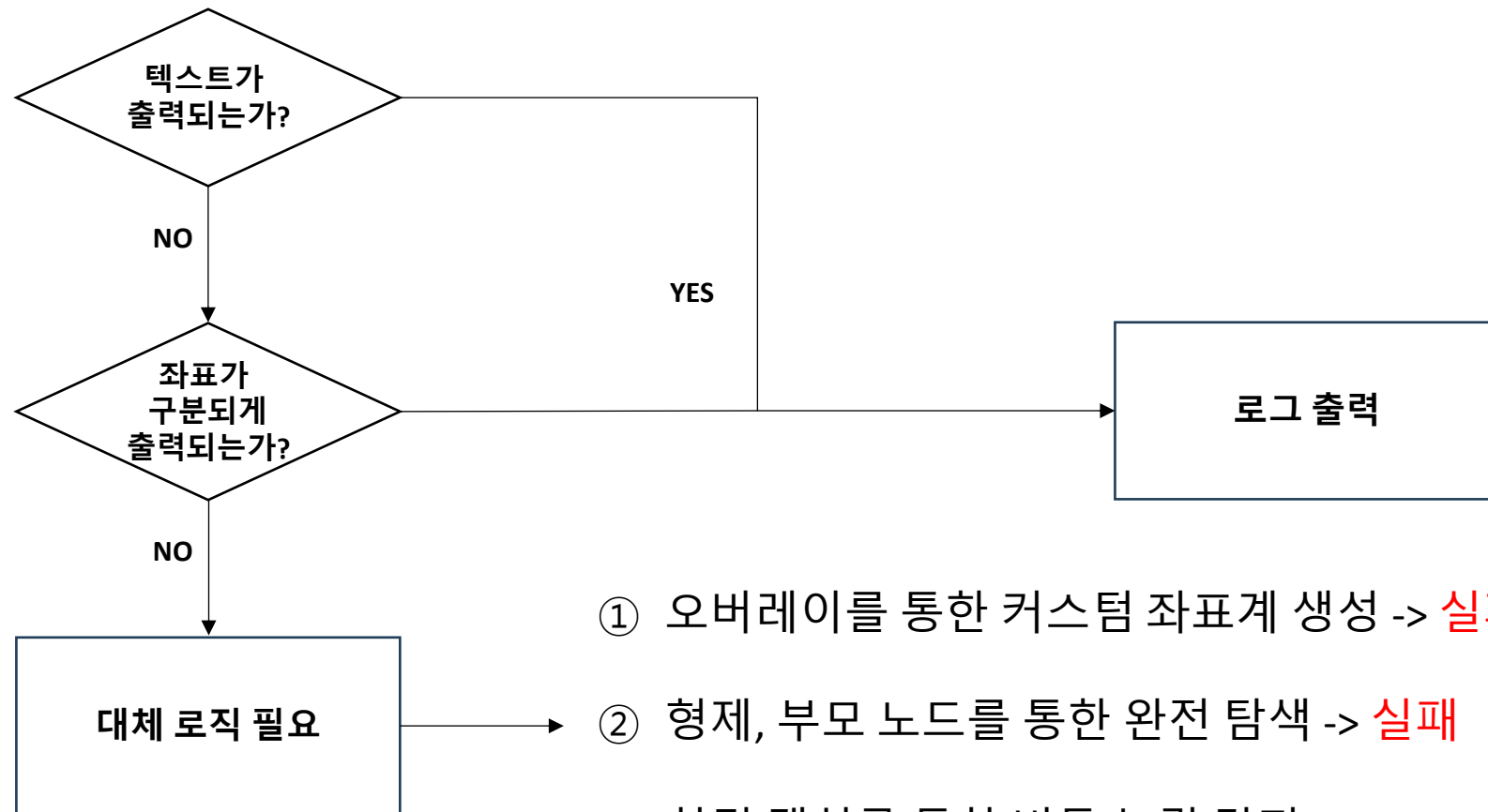
```
Class Name: android.widget.FrameLayout
Clickable: false, Enabled: true, Focusable: false
Bounds in Screen: [0,0][1080,2400]
뷰 위치: (0, 0)
뷰 크기: 1080 x 2400
Margin: left=0, top=0, right=0, bottom=0
Bounds in Parent: [0,0][1080,2400]
클릭 이벤트 절대 좌표 (중앙 값): X=540, Y=1200
Width=1080, Height=2400
```

클릭한 UI 요소 정보:

```
Class Name: android.widget.FrameLayout
Clickable: false, Enabled: true, Focusable: false
Bounds in Screen: [0,0][1080,2400]
뷰 위치: (0, 0)
뷰 크기: 1080 x 2400
Margin: left=0, top=0, right=0, bottom=0
Bounds in Parent: [0,0][1080,2400]
클릭 이벤트 절대 좌표 (중앙 값): X=540, Y=1200
Width=1080, Height=2400
```

- 서로 다른 3개의 버튼을 눌렀음에도 같은 결과 출력
- 이 경우 어떤 버튼 눌렀는 지 식별 전혀 불가능

❖ 버튼 클릭 정보에 대한 처리 프로세스



- ① 오버레이를 통한 커스텀 좌표계 생성 -> 실패
- ② 형제, 부모 노드를 통한 완전 탐색 -> 실패
- ③ 화면 캡처를 통한 버튼 눌림 감지
- ④ 어플 자체에서 발생하는 인텐트 감지

02

APP 향후 계획

1. 만약 버튼 감지 로직이 모두 실패한다면 시스템 어플 제작 고려

➔ INJECT_EVENTS permission으로 버튼 감지 가능할 것이라 생각

2. 리빙랩 차량을 통한 차량과의 추출 가능 로그 데이터 식별 후 구현

3. 정연수 연구원 합류를 통한 인력 보강 완료

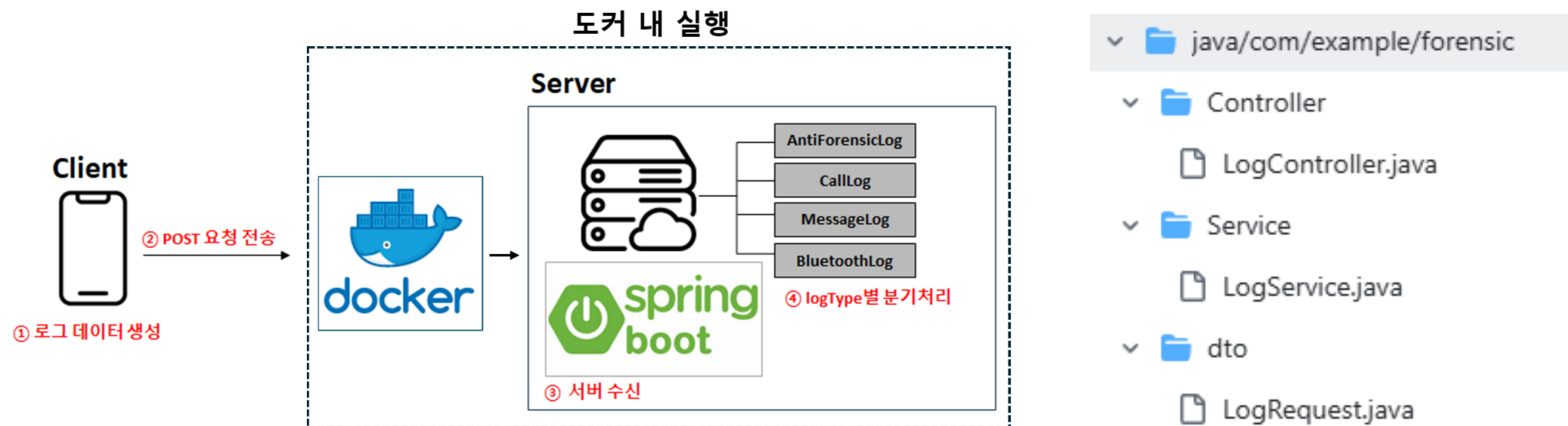
➔ 블루링크 부분 담당 + 무결성 검사 로직 작성 예정

1. 어플 실행 여부 및 어떤 네비게이션 앱인지? (Android Auto 정보 포함)
2. 스마트폰과 연결된 자동차 정보
3. 버튼 클릭 정보 (경로 취소 포함)
4. 출발지-목적지 정보 -- Timestamp 포함
5. 가능하다면, 설정한 경로(목적지) 운행 중에, 음성 인식이나 음성 안내 정보, 또는 이동 경로 정보
6. 가능하다면, 다음 이벤트 정보 -- USB/Bluetooth 연결, GPS Time Syncs, 속도 데이터

03

Spring Boot/Docker 기반 백엔드 구성

스프링 부트를 이용한 백엔드 구성

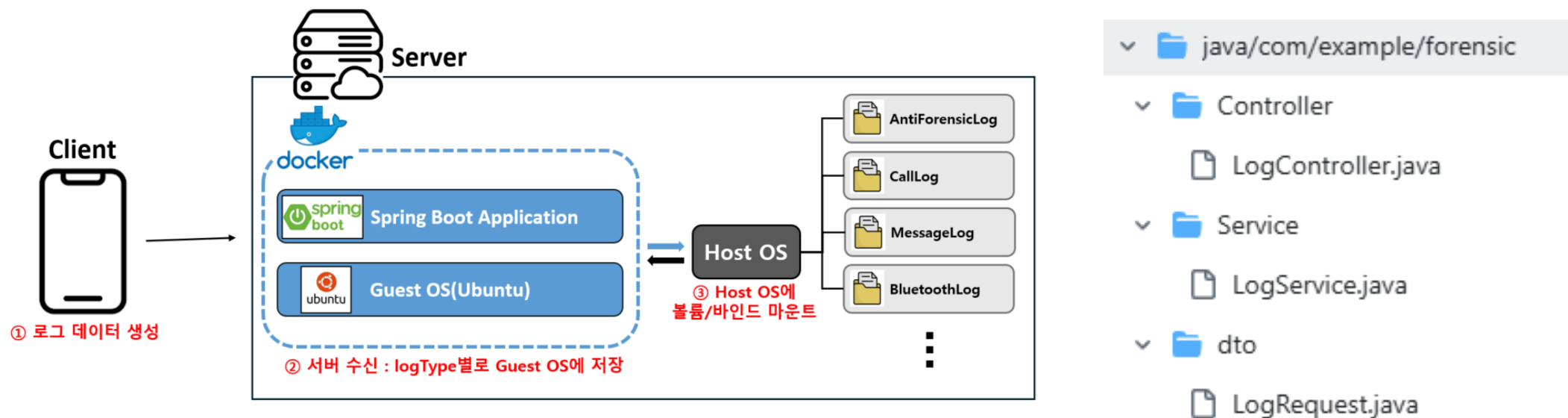


❖ HTTP/HTTPS(REST API)를 이용하여 클라이언트 디바이스에서 서버로 JSON 형식의 데이터를 POST 방식으로 전송.

- 서버는 스프링 부트가 제공하는 REST API 엔드포인트에서 데이터를 수신

❖ 스프링 컨트롤러에서는 클라이언트가 보낸 JSON 파일에서 로그 데이터의 라벨(logType)을 읽어, 각 라벨에 해당하는 파일에 저장

스프링 부트를 이용한 백엔드 구성



- ❖ HTTP/HTTPS(REST API)를 이용하여 클라이언트 디바이스에서 서버로 JSON 형식의 데이터를 POST 방식 전송
 - 서버는 스프링 부트가 제공하는 REST API 엔드포인트에서 데이터를 수신
- ❖ 스프링 컨트롤러에서는 클라이언트가 보낸 JSON 파일에서 로그 데이터의 라벨(logType)을 읽어, 각 라벨에 해당하는 파일을 GuestOS에 저장

Controller

❖ POST - `http://{SeverIP}/logs`

❖ Sever에 로그 데이터를 logType별로 전송

❖ 예시 JSON 데이터와 DTO 정의

```
{  
  "sequenceNumber": 1,  
  "timestamp": "2025-01-13T22:44:07Z",  
  "message": "Bluetooth Connected to:  
Air Pro 2[41:42:54:B4:4F:05]",  
  "logType": "BluetoothLog",  
}
```

LogDTO.java

```
public class LogRequest {  
    private int sequenceNumber;  
    private String timestamp;  
    private String message;  
    private String logType;  
}
```

LogController.java

```
1  package com.example.forensic.Controller;  
2  
3  import com.example.forensic.Service.LogService;  
4  import com.example.forensic.dto.LogRequest;  
5  import org.springframework.http.ResponseEntity;  
6  import org.springframework.web.bind.annotation.*;  
7  
8  @RestController  
9  @RequestMapping("/logs")  
10 public class LogController {  
11  
12     private final LogService logService;  
13  
14     public LogController(LogService logService) {  
15         this.logService = logService;  
16     }  
17  
18     @PostMapping  
19     public ResponseEntity<String> handleLog(@RequestBody LogRequest logRequest) {  
20         // Append the log and get the written content  
21         String appendedContent = logService.appendLog(logRequest);  
22  
23         // Return the appended content in the response  
24         return ResponseEntity.ok("Appended to file: " + appendedContent);  
25     }  
26  
27     @GetMapping("/{logType}")  
28     public ResponseEntity<String> getLogContents(@PathVariable String logType) {  
29         String fileContents = logService.readLog(logType);  
30         return ResponseEntity.ok(fileContents);  
31     }  
32  
33 }
```

❖ GET - `http://{SeverIP}/logs/{logType}`

❖ Sever에 저장된 로그를 Logtype별로 조회 (테스트용)

❖ 로그 추가 기능 (appendLog)

- ❖ 로그 저장 디렉토리(logs/) 생성
- ❖ LogRequest 객체에서 전달받은 LogType에 따라 특정 파일 로그를 저장
- ❖ StandardOpenOption.CREATE, APPEND 옵션

❖ 로그 조회 기능 (readLog)

- ❖ 특정 로그 타입에 해당하는 파일을 읽어 반환

LogService.java

```
@Service
public class LogService {

    private static final String LOG_DIRECTORY = "logs/";

    public String appendLog(LogRequest logRequest) {
        try {
            // Ensure the log directory exists
            Files.createDirectories(Paths.get(LOG_DIRECTORY));

            // Get the log file name
            String logFileName = LOG_DIRECTORY + logRequest.getLogType() + ".txt";

            // Format the log entry
            String logEntry = logRequest.getTimestamp() + " - " + logRequest.getMessage() + System.lineSeparator();

            // Write to the log file
            Files.write(Paths.get(logFileName), logEntry.getBytes(), StandardOpenOption.CREATE, StandardOpenOption.APPEND);

            // Return the written content
            return logEntry;

        } catch (IOException e) {
            throw new RuntimeException("Failed to write log file", e);
        }
    }

    public String readLog(String logType) {
        try {
            String logFileName = LOG_DIRECTORY + logType + ".txt";
            Path logFilePath = Paths.get(logFileName);

            if (!Files.exists(logFilePath)) {
                throw new RuntimeException("Log file not found: " + logFileName);
            }

            // Read the entire file content
            return Files.readString(logFilePath);

        } catch (IOException e) {
            throw new RuntimeException("Failed to read log file", e);
        }
    }
}
```

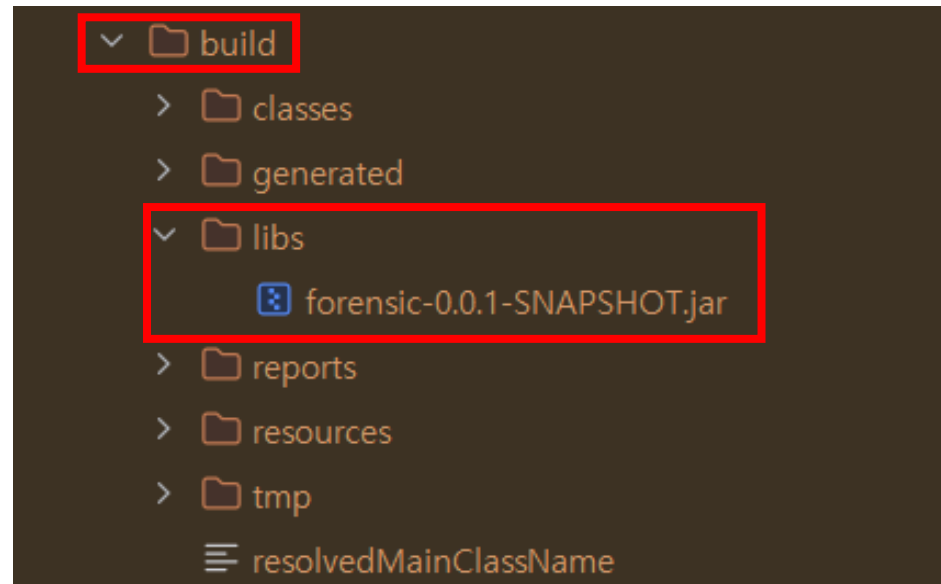
JAR(Java Archive) 파일

❖JAR(Java Archive, 자바 아카이브)는 여러 개의 자바 클래스 파일과, 관련 리소스(텍스트, 그림 등) 및 메타데이터를 하나의 파일로 모아서 배포하기 위한 패키지 파일

- 따라서 서버의 응용 프로그램 배포에서는 '실행 가능한' jar 파일이 필요

❖따라서 프로젝트 내 ./gradlew bootJar 명령어로 실행가능한 jar파일 생성

- build/libs에 위치



2. DockerFile 작성

❖ DockerFile 작성

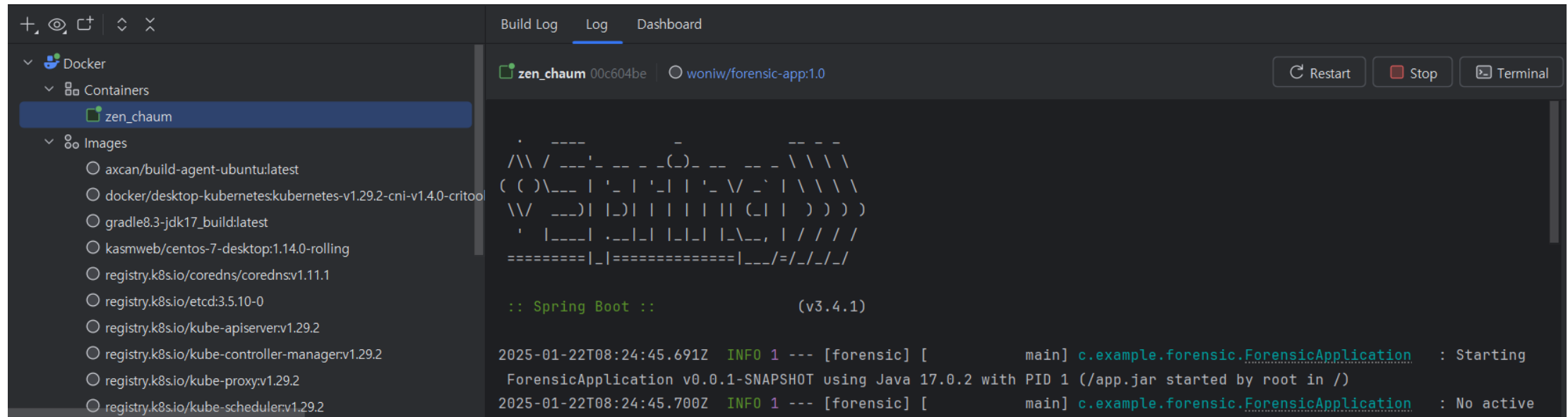
```
1 # 1. 베이스 이미지를 설정합니다.
2 >> FROM openjdk:17-jdk-slim
3
4 # 2. 애플리케이션 JAR 파일을 컨테이너 내부에 복사합니다.
5 ARG JAR_FILE=build/libs/forensic-0.0.1-SNAPSHOT.jar
6 COPY ${JAR_FILE} app.jar
7
8 # 3. 컨테이너가 실행될 때 JAR 파일을 실행하도록 설정합니다.
9 ENTRYPOINT ["java", "-jar", "/app.jar"]
10
```

❖ Build 수행

```
PS C:\Users\woniw\IdeaProjects\forensic> docker build -t woniw/forensic-app:1.0 .
[+] Building 0.0s (0/0) docker:default
[+] Building 1.6s (7/7) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 397B
=> [internal] load metadata for docker.io/library/openjdk:17-jdk-slim
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/2] FROM docker.io/library/openjdk:17-jdk-slim@sha256:aaa3b3cb27e3e520b8f116863d0580c438ed55ecfa0bc126b41f68c3f62f9774
=> [internal] load build context
=> => transferring context: 117B
=> CACHED [2/2] COPY build/libs/forensic-0.0.1-SNAPSHOT.jar app.jar
=> exporting to image
```

❖ Build 완료 후 테스트 예시

- Port는 8080으로 임시 설정







The screenshot shows the Docker Desktop interface. On the left, the 'Containers' section is expanded, showing a container named 'zen_chaum'. The 'Images' section is also visible, listing various Docker images. The main panel displays the 'Log' for the 'zen_chaum' container, which is running the 'woniw/forensic-app:1.0' image. The log output shows the application starting and running successfully, with a message indicating that the application is starting and then becoming active.

❖ Docker Hub에 Image push

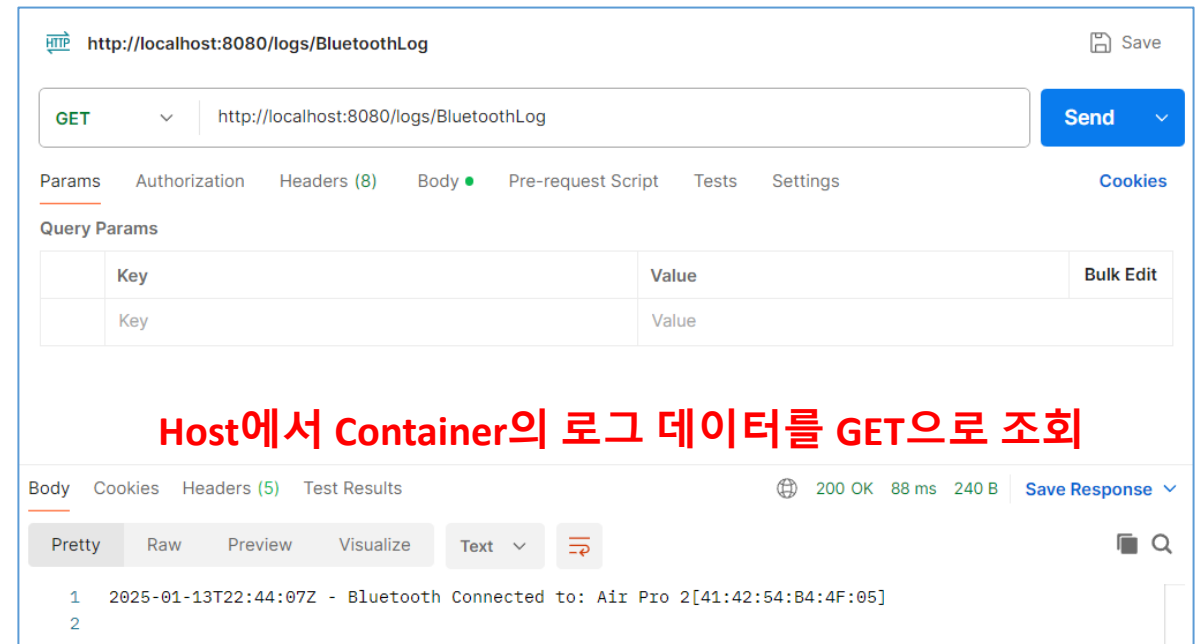
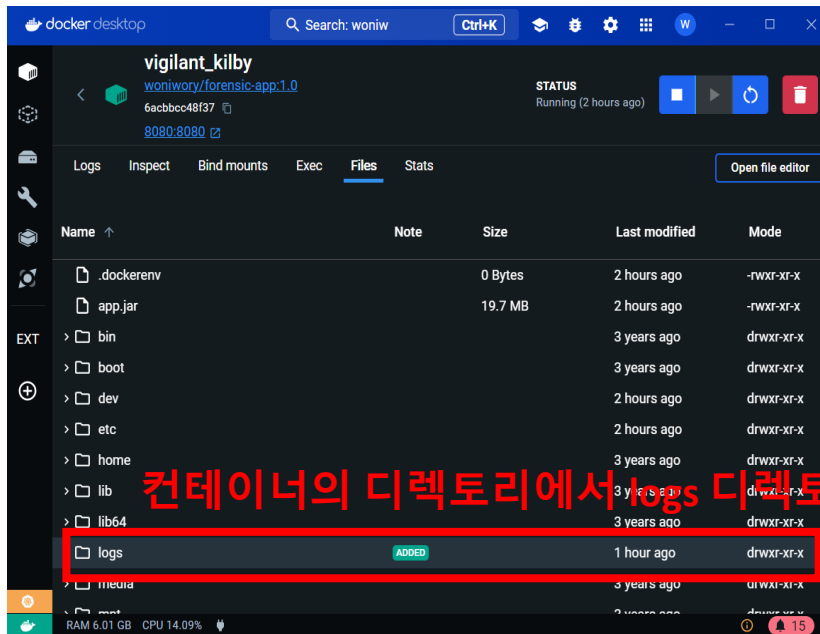
```
PS C:\Users\woniw\IdeaProjects\forensic> docker push woniwory/forensic-app:1.0
The push refers to repository [docker.io/woniwory/forensic-app]
Get "https://registry-1.docker.io/v2/": context deadline exceeded (Client.Timeout exceeded while awaiting headers)
```

❖ Push된 이미지로 컨테이너를 실행

- `Docker run -d -p 8080:8080 woniwory/forensic-app:1.0`

| <input type="checkbox"/> | Name | Image | Status | Port(s) | CPU (%) | Last started | Actions |
|--------------------------|---|------------------------------|---------|---------|---------|---------------|---|
| <input type="checkbox"/> |  zen_chaum 00c604bef795 | 820b4b728d95 | Running | | 0.18% | 2 minutes ago |    |

❖ Local 환경에서 API 테스트 완료

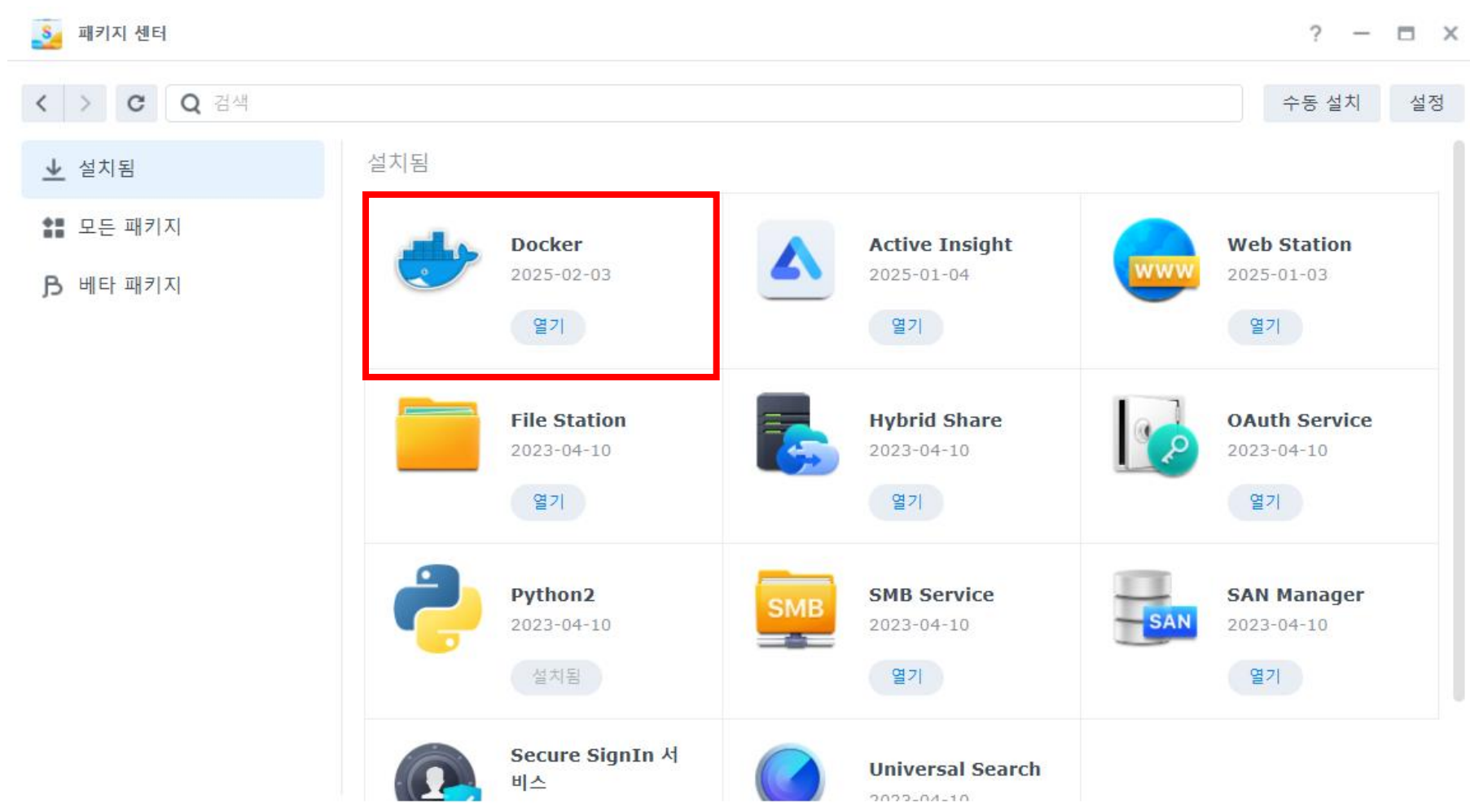


04

NAS 서버 설정 및 배포

NAS 서버에서 Docker 환경 구성

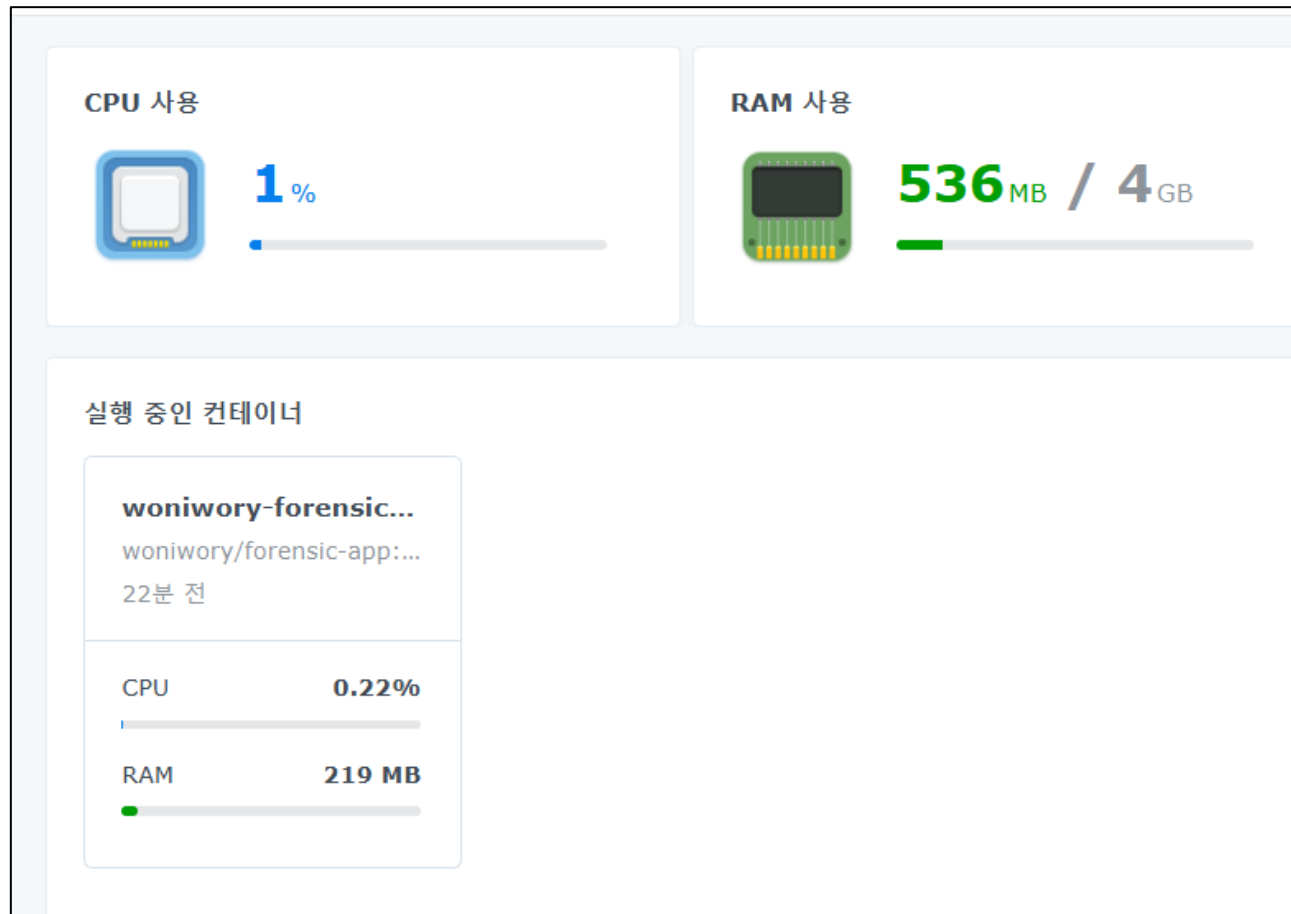
❖ NAS 서버의 패키지 센터에서 Docker 설치



NAS 서버에서의 컨테이너 실행

❖ Push된 이미지를 NAS에서 Pull 하여 컨테이너를 실행

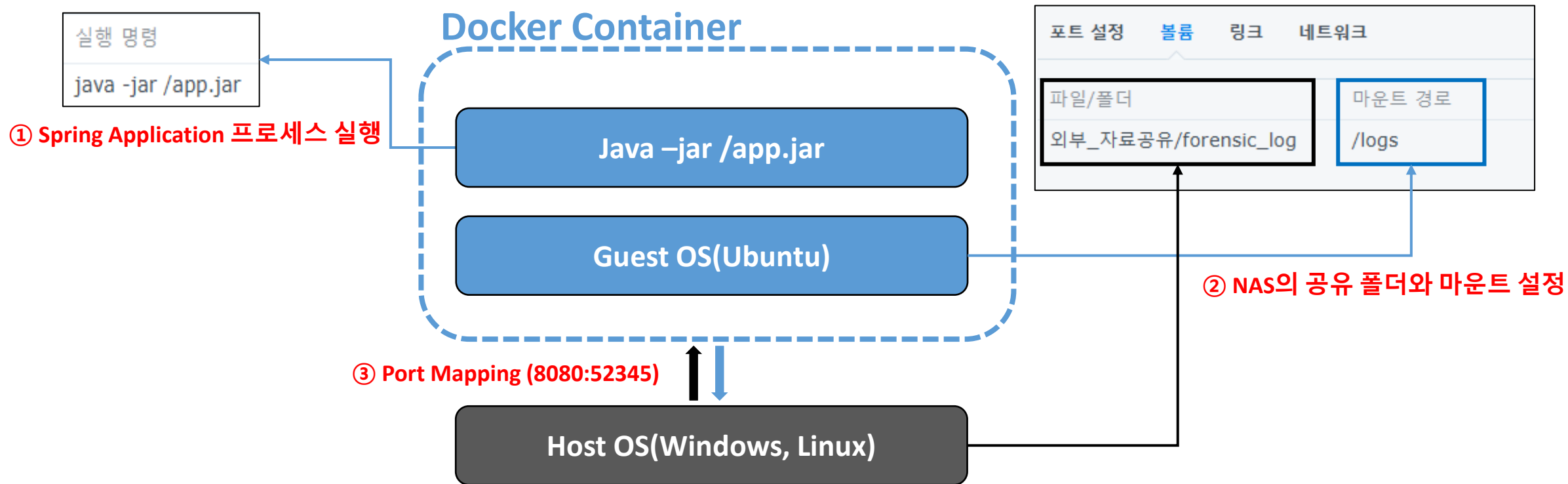
- `Docker run -d -p 52345:8080 woniwory/forensic-app:1.0`



| 포트 설정 | | | 볼륨 | 링크 | 네트워크 |
|-------|---------|-----|----|----|------|
| 로컬 포트 | 컨테이너 포트 | 유형 | | | |
| 52345 | 8080 | tcp | | | |

| 포트 설정 | | | 볼륨 | 링크 | 네트워크 |
|----------------------|--------|----|----|----|------|
| 파일/폴더 | 마운트 경로 | 유형 | | | |
| 외부_자료공유/forensic_log | /logs | rw | | | |

Container Architecture

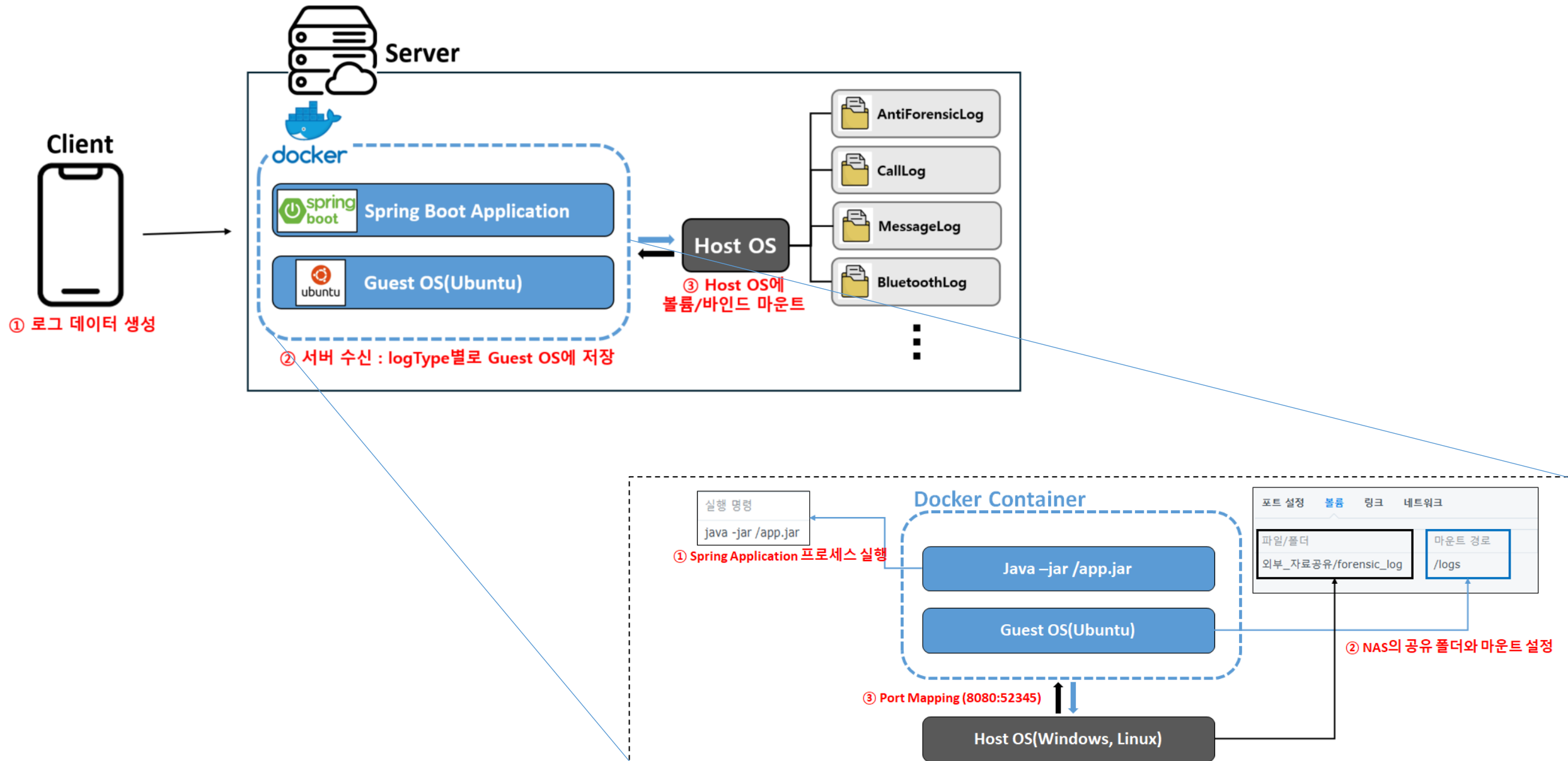


❖ 컨테이너 내의 `/logs` 디렉토리와 최종적으로 저장하려는 NAS 공유폴더를 바인드 마운트(Bind Mount)하여 저장

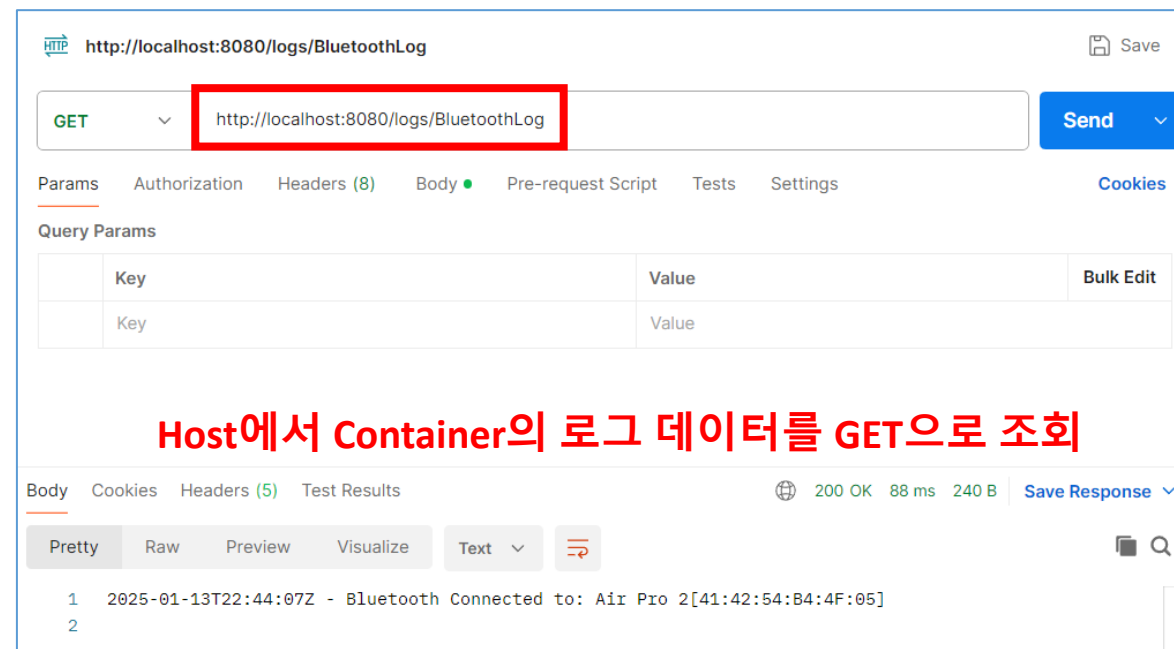
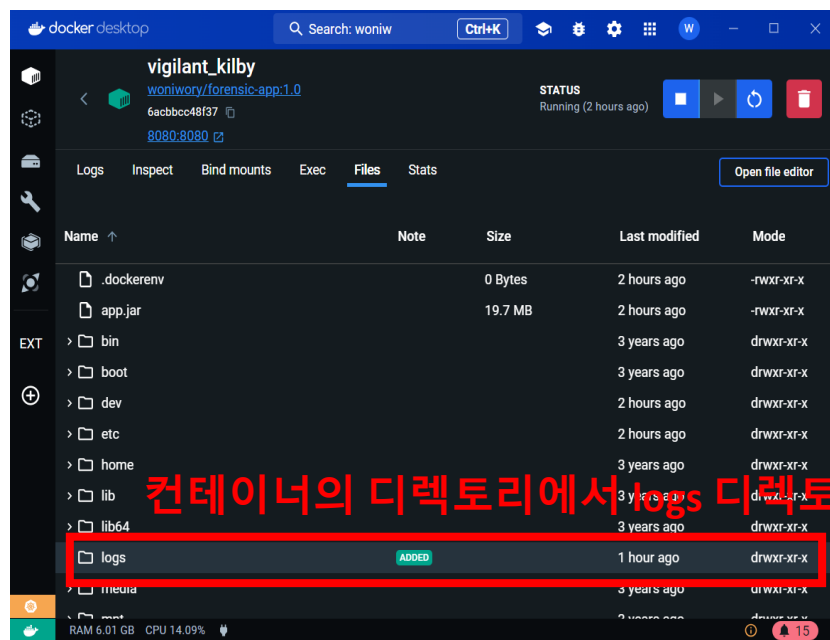
❖ Docker 컨테이너는 기본적으로 휘발성(Volatile)이므로 Volume/Bind Mount 사용이 필수적

- 컨테이너 삭제/재시작시 데이터 증발을 막기 위한 기법
 - Volume : Docker가 직접 관리하는 저장소를 생성
 - Bind Mount : 특정 호스트(서버)의 디렉토리를 컨테이너 내부 디렉터리에 직접 연결

스프링 부트를 이용한 백엔드 구성



❖ 기존 Local 환경에서 Client ↔ Sever 간 API 테스트



Host에서 Container의 로그 데이터를 GET으로 조회

기존 Local 환경에서 Client ↔ Sever 테스트와의 비교

❖ POST 요청으로 로그 데이터 NAS 서버의 전송 시, Bind Mount를 통해 컨테이너의 데이터가 NAS 서버의 공유 폴더에 저장되는 것을 확인 가능

The screenshot shows a REST client interface with a POST request to `http://NAS 공인IP/logs`. The response status is `200 OK` with a response time of `15 ms` and a size of `258 B`. The response body is displayed in text format, showing a log message: `Appended to file: 2025-01-13T22:44:07Z - Bluetooth Connected to: Air Pro 2[41:42:54:B4:4F:05]`.

A red box highlights a file explorer view showing the log file `BluetoothLog.txt` saved in the `외부_자료공유 > forensic_log` directory. The file size is `76 바이트` and the modification date is `2025-02-03 18:24:12`.

04

서버 향후 계획

❖ Docker를 이용한 NAS 서버 / 교내 서버에 JAR 방식의 배포 - 완료

❖ 현재 PC로 서버 역할을 수행 중 → Docker Hub에 Push한 이미지가 NAS 서버에서 동작 중

❖ 비식별화 구현

❖ JSON 필드의 형식에 따라 Service 클래스와 DTO(Data Transfer Object) 클래스에서
로직 구현 예정

❖ 순서번호, 메타데이터, HTTPS, 암호화 등 → 추가 사항

❖ Multi-User 방식을 위한 DB 구축 고려

감사합니다
