출입 통제 시스템: 상세 기능 명세서

본 문서는 출입 통제 시스템이 제공하는 모든 핵심 기능을 상세하게 정의합니다.

1. 핵심 출입 통제 기능 (Core Access Control) (상세)

시스템의 심장부로, 실시간 보안 및 판정을 담당합니다.

1.1. 하이브리드 판정 엔진 및 다중 인증 방식 (버전별 상세)

본 아키텍처는 단말기 종류에 따라 판정 주체와 작동 방식이 완전히 달라지는 **하이브리드 모델**을 지원합니다.

A. SDAC 버전 (옵션 1: ACU 미적용 / Pull 방식) - 유연성 및 저비용

- **핵심 원칙:** "Dumb Terminal" - 리더기는 ID만 읽고, **중앙 서버(`Access Control Service`)가 모든 판정**을 수행합니다.
- **통신 흐름:** `리더기` → `API Gateway` (Device Token 인증) → `Access Control Service` (실시간 판정) → `API Gateway` → `리더기` (문 열림)
- **판정 로직:** `Access Control Service`는 요청받은 `ID`를 `Redis` 캐시에서 조회하여 0.1초 내로 판정합니다.

인증 방식	작동 흐름 (Pull 방식)
1. 카드 방식	1. 사용자가 'IP 카드 리더기'에 '출입 카드'를 태그합니다. 2. 리더기는 `Card_ID`와 `Device Token`을 `API Gateway`로 전송합니다. 3. `Access Control Service`가 `Redis`에서 `Card_ID`를 조회하여 "GRANT"를 응답합니다. 4. 리더기가 응답을 받고 문을 엽니다.
2. 스마트폰 태그	1. 사용자가 'NFC/BLE IP 리더기'에 스마트폰을 태그합니다. 2. 리더기는 `NFC_ID`와 `Device Token`을 `API Gateway`로 전송합니다. 3. `Access Control Service`가 `Redis`에서 `NFC_ID`를 조회하여 "GRANT"를 응답합니다.
3. 지문 방식	1. 사용자가 'IP 지문 리더기'에 지문을 스캔합니다. 2. 리더기 하드웨어가 로컬 템플릿과 비교하여 `User_ID`를 식별합니다.

인증 방식	작동 흐름 (Pull 방식)
	3. 리더기는 `User_ID`와 `Device Token`을 `API Gateway`로 전송합니다. 4. `Access Control Service`가 `Redis`에서 `User_ID`의 **2차 정책(시간/구역)**을 판정하여 "GRANT"를 응답합니다.
4. QR 코드 방식	1. 방문객이 'IP QR 리더기'에 QR 코드를 스캔합니다. 2. 리더기는 `QR_ID`와 `Device Token`을 `API Gateway`로 전송합니다. 3. `Access Control Service`가 `Redis`에서 `QR_ID`의 유효성(시간 만료)을 판정하여 "GRANT"를 응답합니다. (Redis EXPIRE 기능 활용)

B. ACU 버전 (옵션 2: 하이브리드 / Push 방식) - 안정성 및 고비용

- **핵심 원칙:** "Smart Controller" - `Access Control Service`가 **정책을 ACU에 미리 Push**해 두면, **ACU가 로컬에서 판정**합니다.
- **통신 흐름:** `리더기` → `ACU` (로컬 판정 및 문 열림) → `API Gateway` (사후 로그 전송)
- **판정 로직:** `ACU`가 로컬 메모리에서 판정하므로 중앙 서버 장애 시에도 작동합니다.

인증 방식	작동 흐름 (Push 방식)
1. 카드 방식	1. **(사전 작업)** `Access Control Service`가 "카드 ID 1A-2B-3C는 문 A 통과 가능" 정책을 `ACU`에 미리 **Push**해 둡니다. 2. 사용자가 '카드 리더기'(ACU에 연결됨)에 태그합니다. 3. `ACU`가 `Card_ID`를 받고, **로컬 메모리**의 정책을 조회하여 "GRANT"를 **즉시 판정**하고 문을 엽니다. 4. **(사후 전송)** `ACU`가 `API Gateway`로 로그를 전송합니다.
2. 스마트폰 태그	1. **(사전 작업)** `Access Control Service`가 "NFC ID 9F-8E-7D" 정책을 `ACU`에 **Push**합니다. 2. 사용자가 'NFC 리더기'(ACU에 연결됨)에 태그합니다. 3. `ACU`가 `NFC_ID`를 받고 **로컬 메모리**에서 즉시 판정 및 문을 엽니다. 4. **(사후 전송)** `ACU`가 로그를 전송합니다.
3. 지문 방식	1. **(사전 작업)** `Access Control Service`가 "User ID Kim123" 정책을 `ACU`에 **Push**합니다.

인증 방식	작동 흐름 (Push 방식)
	2. 사용자가 '지문 리더기'(ACU에 연결됨)에 스캔합니다. 3. 리더기가 `User_ID`를 식별하여 `ACU`로 전달합니다. 4. `ACU`가 **로컬 정책**과 비교하여 "GRANT"를 **즉시 판정**하고 문을 엽니다. 5. **(사후 전송)** `ACU`가 로그를 전송합니다.
4. QR 코드 방식	1. **(사전 작업)** `Access Control Service`가 "임시 QR ID Temp_ABC" 정책을 `ACU`에 **Push**합니다. 2. 방문객이 'QR 리더기'(ACU에 연결됨)에 스캔합니다. 3. `ACU`가 `QR_ID`와 **현재 시간**을 **로컬 정책**과 비교하여 "GRANT"를 **즉시 판정**하고 문을 엽니다. 4. **(사후 전송)** `ACU`가 로그를 전송합니다.

1.2. 고급 보안 및 비상 정책 (상세)

기능 분류	상세 기능	아키텍처 구현 (연동 서비스)
1.2.1. 고급 보안 정책	**안티-패스백 (Anti-Passback):** • 'In' 태그 후 'Out' 태그 없이 다시 'In' 태그를 시도하는 카드 돌려쓰기를 방지합니다.	• `Access Control Service`가 판정 시 `Redis`에 저장된 사용자의 **현재 상태(In/Out)**를 실시간으로 조회하여 판정합니다. • "GRANT" 판정 후, `Access Control Service`가 해당 사용자의 `Redis` 상태를 "In"에서 "Out"으로 (또는 그 반대로) 즉시 업데이트합니다.
	테일게이팅 (꼬리물기) 감지: • 1명 인가 후 2명 이상이 물리적으로 통과하는 것을 감지하고 경보를 울립니다.	• 문에 설치된 `3D/빔 센서`가 `네트워크 I/O 컨트롤러`에 물리 신호를 보냅니다. • `I/O 컨트롤러`는 이 신호를 `API Gateway`의 비상 엔드포인트(예: `/api/alert/tailgating`)로 전송합니다. • `Access Control Service`가 이벤트를 받아 즉시 현장 경보를 울리고 `Kafka`로 "테일게이팅 위반" 로그를 발행합니다.
1.2.2. 비상 상황 제어	**화재 연동 (Fail-Safe):** • 화재 수신반(FACP) 신호 수신 시, 지정된 모	• `네트워크 I/O 컨트롤러`가 화재 신호를 `API Gateway`의 최우선 비상 엔드포인트(예: `/api/emergency/fire`)로 전송합니다. • `Access Control Service`가 이 요청을 **모든 정책**

기능 분류	상세 기능	아키텍처 구현 (연동 서비스)
	든 문을 자동으로 개방합니다.	**(Policy)을 무시**하고 최우선으로 처리하여, `Kafka`를 통해 모든 관련 `IP 리더기` / `ACU`에 "즉시 개방" 명령을 브로드캐스트합니다.
	중앙 통제 (Lockdown): • `Admin UI`에서 관리자가 클릭 한 번으로 특정 구역 또는 전체 시스템을 즉시 **봉쇄 (Lockdown)**하거나 **개방(All-Open)**합니다.	• `Access Control Service`가 이 명령을 받아, `Policy Service`의 모든 일반 규칙을 무시하는 **전역 상태 (Global State)**를 `Redis`에 설정합니다. (예: `SET global:lockdown:zone_A true`) • 모든 출입 판정 시 이 전역 상태를 먼저 확인하여 시스템을 즉시 통제합니다.
	관리자 원격 개방 (Remote Unlock): • 관리자가 `Admin UI`에서 CCTV 등을 보며 특정 문(예: 'Door101')을 선택하여 **수동으로 즉시 개방**합니다.	1. `Admin UI`가 `API Gateway` API(예: `/admin/commands/open-door`)를 호출합니다. 2. `API Gateway`가 관리자 **JWT 토큰의 권한 (Role)**을 확인합니다. 3. `Access Control Service`가 이 명령을 받아 해당 `IP 리더기` 또는 `ACU`에 직접 개방 명령을 전송합니다. 4. `Access Control Service`는 이 행위를 **Kafka**의 감사 로그 토픽(topic)으로 즉시 발행(Produce)하여, `Log Service`가 **영구 기록(Audit)**하도록 합니다.
1.2.3. 고급 알람 관리	**문 강제 개방 (Door Forced Open):** • 인가된 태그 없이 문이 물리적으로 강제로 열렸을 때 즉시 경보가 울리고 기록됩니다.	• 문틀에 설치된 `도어 센서(Door Sensor)`가 `IP 리더기` 또는 `ACU`의 I/O 포트에 연결됩니다. • `Access Control Service`가 "GRANT" 신호를 보내지 않았는데 도어 센서 신호가 감지되면, `IP 리더기/ACU`가 즉시 `API Gateway`로 "강제 개방" 알람 이벤트를 전송합니다.
	문 열림 시간 초과 (Door Held Open): • 문이 열린 후 설정된 시간(예: 30초) 이내에 닫히지 않으면 경보가 울립니다.	• `Access Control Service`가 "GRANT" 신호를 보낼 때 `Redis`에 타이머를 설정합니다(예: `EXPIRE door:101:timer 30s`). • `도어 센서`가 닫힘 신호를 보내면 타이머가 삭제됩니다. • 30초가 지나도록 닫힘 신호가 없어 타이머가 만료

기능 분류	상세 기능	아키텍처 구현 (연동 서비스)
		되면, 시스템이 "시간 초과" 알람을 `Admin UI`와 `Log Service`로 전송합니다.

1.3. 엘리베이터 제어 기능 (Elevator Control) (상세)

기능 분류	상세 기능	아키텍처 구현 (연동 서비스)
1.3.1. 층별 접근 제어	**엘리베이터 목적층 제어:** • 사용자가 엘리베이터 내부 리더기에 카드/지문/NFC 태그 시, 허가된 층의 버튼만 활성화 시킵니다.	1. 사용자가 엘리베이터 내부의 `IP 리더기`에 태그합니다. 2. `Access Control Service`가 `Redis`에서 `User_ID`와 `Policy_ID`를 조회하여 "허가된 층 목록" (예: `[1, 3, 7]`)을 확인합니다. 3. `Access Control Service`가 **엘리베이터 제어 전용 `네트워크 I/O 컨트롤러`**의 API를 호출하여, [1, 3, 7]층 버튼에 해당하는 **릴레이(Relay)만** 몇 초간 활성화(Close)시키라는 명령을 전송합니다.

1.4. 물리적 장치 제어 (출입문 개폐기) (상세)

모든 출입 판정(GRANT)의 최종 목표는 물리적인 '출입문 개폐기'(도어락, EM Lock 등)를 작동시키는 것입니다. 중앙의 `Access Control Service`(소프트웨어)는 개폐기에 직접 연결되지 않으며, 제어 주체는 아키텍처 버전에 따라 나뉩니다.

버전	물리적 제어기	물리적 연결	제어 흐름 (GRANT 시)
A. SDAC 버전 (옵션 1)	**`IP 리더기`**	`개폐기` → `IP 리더기` (릴레이 포트)	1. `Access Control Service`가 `API Gateway`를 통해 `IP 리더기`에 **IP 명령** 전송. 2. `IP 리더기`가 명령을 수신하여, 자신에게 연결된 `개폐기`에 **12V 전기 신호**를 보내 문을 엽니다.
B. ACU 버전 (옵션 2)	**`ACU`**	`개폐기` → `ACU` (릴레이 포트)	1. `ACU`가 로컬 메모리에서 판정. 2. `ACU`가 `Access Control Service`의 개입 없이, **직접** `개폐기`에 **12V 전기 신호**를 보내 문을 엽니다.

2. 관리자 기능 (Admin UI - Management) 상세

`Admin UI` 웹 애플리케이션을 통해 제공되는 모든 관리 및 설정 기능의 상세 명세입니다. 모든 요청은 `API Gateway`를 통해 인가된 관리자만 접근할 수 있습니다.

2.1. 출입자 관리 (User Service 연동)

- **출입자 통합 ID 관리 (CRUD):**
  - **등록(Create):** 출입자 기본 정보(이름, 부서, 직급, 연락처)를 등록합니다.
  - **조회(Read):** 모든 출입자를 검색, 필터링, 페이징하여 조회합니다.
  - **수정(Update):** 등록된 출입자의 정보를 수정합니다.
  - **인증 매체 매핑:** 특정 출입자(`User_ID`)에게 다양한 인증 매체를 매핑합니다.
    - `물리 카드 ID` (1:N 매핑 가능, 예: 신용카드, 사원증)
    - `스마트폰 NFC ID` (모바일 사원증)
    - `지문 ID 템플릿` (지문 리더기를 통해 등록된 ID)
    - `일회용 QR ID` (방문객용, `2.5`에서 생성됨)
- **출입자 그룹 관리:**
  - **그룹 생성/삭제:** '임원', '엔지니어', '방문객', '미화팀' 등 정책 할당의 기준이 되는 그룹을 생성하고 관리합니다.
  - **그룹 멤버 관리:** 출입자를 특정 그룹에 소속시키거나 제외시킵니다. (1인 다수 그룹 소속 가능)
- **출입자 상태 관리 (보안):**
  - **즉시 비활성화(정지):** 카드 분실, 도난, 퇴사자 발생 시, 해당 `User_ID` 또는 `Card_ID`의 상태를 '정지'로 변경합니다.
  - `User Service`는 이 변경 사항을 즉시 `PostgreSQL`에 저장하고, **`Redis Cache`에도 동기화**하여 0.1초 내에 출입 판정(`Access Control Service`)에 반영되도록 합니다.

2.2. 정책/구역 관리 (Policy Service 연동)

- **구역(Zone) 정의:**
  - '1층 로비', '서버실', 'R&D 구역' 등 물리적 공간을 논리적 구역으로 정의합니다.
  - `Device Service`(가상)와 연동하여 특정 '문(Door)'을 특정 '구역'에 할당합니다.
- **시간표(Time Schedule) 관리:**

    - '주간 근무 시간 (월-금, 09-18)', '심야 시간 (00-06)', '공휴일' 등 재사용 가능한 시간표 템플릿을 생성합니다.
- **접근 규칙(Access Rule) 매핑 (핵심):**
    - **[누가]** '엔지니어 그룹'이
    - **[어디를]** '서버실' 구역을
    - **[언제]** '주간 근무 시간'에만
    - **[어떻게]** '카드+지문 (2-Factor)' 방식(선택적)으로만
    - 접근할 수 있는지 조합하여 **최종 접근 정책**을 생성하고 매핑합니다.
    - 생성된 정책은 `PostgreSQL`에 영구 저장되며, 판정에 필요한 데이터는 `Redis`에 캐시됩니다.
- **ACU 동기화 (ACU 버전 전용):**
    - 정책 저장 시, `Access Control Service`를 트리거하여 해당 정책을 사용하는 `ACU` 하드웨어로 정책을 **Push(동기화)**하도록 명령합니다.

2.3. 장치/하드웨어 관리 (Device Service 연동)

- **단말기 등록 및 관리:**
    - `ACU` 또는 `IP 리더기` 설치 시, `Device ID` 및 `Device Token`(비밀 키)을 시스템에 등록합니다.
    - 네트워크 정보(IP, Port), 설치 위치, 연결된 '문' 정보를 매핑합니다.
- **문(Door) 설정:**
    - 물리적인 '출입문 개폐기(도어락)'에 대한 설정을 정의합니다.
    - **문 이름:** 예: '서버실 정문'
    - **재잠금 시간(Time to Relock):** 'GRANT' 신호 후 문이 다시 잠길 때까지의 시간 (예: 5초)
    - **개방 시간 초과(Held Open) 알람:** 문이 설정된 시간(예: 30초) 이상 닫히지 않으면 `1.2.3`의 알람을 울리도록 설정합니다.
- **네트워크 I/O 컨트롤러 관리:**
    - 화재 수신반, 엘리베이터 릴레이, 3D 센서 등과 연동된 `네트워크 I/O 컨트롤러`를 등록하고 설정합니다.

2.4. 시스템 관리자 및 권한 관리 (IAM) 상세 (수정됨)

아키텍처 설명 (Auth/User 분리): 시스템 관리자 기능은 2개의 서비스가 협력합니다.

- `Auth Service`: '인증(Authentication)'을 전담합니다. (로그인, 토큰 발급/검증/갱신)
- `User Service`: '인가(Authorization)' 및 '계정 데이터(Identity)'를 전담합니다. (관리자 계정 정보 CRUD, 역할(Role), 권한(Permission) 관리, 암호 재설정)
- `Auth Service`는 로그인 시 `User Service`를 `OpenFeign`으로 호출하여 '비밀번호'와 '역할' 정보를 조회합니다.

기능 분류	상세 기능	아키텍처 구현 (연동 서비스)
2.4.1. 관리자 인증 (Auth)	**관리자 로그인 (Access/Refresh 토큰):** • `Admin UI`에서 ID/PW로 로그인하면, `Auth Service`가 이를 검증하고 **Access Token** (단기, API 호출용)과 **Refresh Token** (장기, HttpOnly, 갱신용)을 발급합니다.	`Auth Service` → `User Service` (PW 해시 및 역할 조회)
	관리자 로그아웃: • `Auth Service`가 `Refresh Token`을 무효화(Revoke)하여 세션을 종료시킵니다.	`Auth Service` (Token Revocation)
	토큰 갱신: • `Access Token` 만료 시, 프론트엔드가 `Refresh Token`을 사용하여 `Auth Service`에 조용히 새 `Access Token`을 재발급 받습니다.	`Auth Service`
	비밀번호 변경 (로그인 상태): • `Admin UI`에서 현재 비밀번호와 새 비밀번호를 입력하여 변경합니다.	`User Service` (PW 해시 업데이트)
	비밀번호 재설정 (분실 시): • **(폐쇄망 방식)** 관리자가 암호 분실 시, **'최고 관리자(Super Admin)'**가 `Admin UI`에서 해당 관리자의 비밀번호를 **직접 초기화(재설정)**합니다. (이메일 인증 불필요)	`User Service` (PW 해시 업데이트 by Super Admin)
2.4.2. 관리자 계정 (Identity)	**관리자 계정 관리 (CRUD):** • `Admin UI`에 접속할 관리자 계정(ID, 이름,	`User Service` (Admin User Table CRUD)

기능 분류	상세 기능	아키텍처 구현 (연동 서비스)
	부서, 연락처)을 생성, 수정, 삭제합니다. (출입자 계정과 별도 관리)	
2.4.3. 권한 관리 (Access)	**역할(Role) 관리 (CRUD):** • '최고 관리자', '정책 관리자', '보안팀(로그 조회 전용)' 등 역할(Role)을 생성하고 관리합니다.	`User Service` (Roles Table CRUD)
	권한(Permission) 관리 (CRUD): • '사용자_조회', '정책_수정', '로그_조회' 등 시스템의 모든 개별 API/메뉴 접근 권한을 정의합니다.	`User Service` (Permissions Table CRUD)
	역할-권한 매핑: • '정책 관리자' 역할(Role)은 '정책_수정'과 '사용자_조회' 권한(Permission)을 갖도록 매핑합니다.	`User Service` (Role_Permission_Map Table)
	관리자-역할 할당: • 'admin_kim' 관리자 계정에 '정책 관리자' 역할(Role)을 할당합니다.	`User Service` (User_Role_Map Table)
	API 인가 (Authorization): • `API Gateway`가 모든 `/admin/**` 요청 수신 시, `Access Token`의 역할(Role)/권한(Permission)을 검사하여, 해당 API를 호출할 권한이 있는지 확인합니다.	`API Gateway` (JWT Role/Permission 검증 필터)

2.5. 방문객 관리 시스템 (VMS - 폐쇄망 방식)

- **방문객 신청/승인 (폐쇄망 방식):**
  - 내부 담당자가 `Admin UI`에서 방문객 정보(이름, 회사, 방문 사유)를 입력하고 "승인"합니다.
  - 백엔드(`User/Policy Service`)는 이 정보를 바탕으로 **임시** `QR_ID`(Unique ID)와 **임시 접근 정책**(예: "10월 10일 14시~15시, 1층 로비 구역만")을 생성하여 `PostgreSQL` 및 `Redis`에 저장합니다.
  - `Redis`에 저장된 `QR_ID`는 `Redis EXPIRE` 기능을 사용하여 **설정된 만료 시간(예: 15시)**이 지나면 **자동으로 삭제**됩니다.

- **QR 코드 발급 (물리적):**
  - 승인 성공 시, `Admin UI`는 `QR_ID` 값으로 **QR 코드 이미지**를 즉시 생성하여 **모니터에 표시**합니다.
  - 관리자는 이 QR 코드를 방문객이 **스마트폰으로 사진을 찍게** 하거나, **라벨 프린터로 인쇄**하여 방문증(스티커)으로 발급합니다.
- **방문객 강제 만료:**
  - 방문객이 용무를 일찍 마쳤을 경우, 관리자가 `Admin UI`에서 "방문 종료" 버튼을 누릅니다.
  - `User Service`는 `Redis`에서 해당 `QR_ID` 키를 **즉시 `DEL`(삭제)**하여 QR 코드를 무효화합니다.
- **방문객 체크인/아웃:**
  - 방문객이 로비에서 발급받은 QR 코드(사진/인쇄물)로 스캔하여 체크인/체크아웃합니다. (`1.1.A`의 QR 코드 방식)

3. 모니터링 및 감사 (Monitoring & Auditing) (상세)

시스템의 투명성과 장애 추적을 위한 기능입니다.

기능 분류	상세 기능	아키텍처 구현 (연동 서비스)
3.1. 실시간 대시보드	**실시간 출입 현황:** • 모든 출입문에서 발생하는 GRANT/DENY 이벤트를 실시간으로 `Admin UI` 대시보드에 표시합니다.	1. `Access Control Service`가 판정 후 `Kafka` 토픽(예: `access_events`)에 로그를 발행합니다. 2. 별도의 `WebSocket Service`가 이 `Kafka` 토픽을 구독(Consume)합니다. 3. `WebSocket Service`는 `Socket.IO`를 사용하여 `Admin UI`에 연결된 모든 관리자에게 이벤트를 브로드캐스트합니다.
	실시간 장치 상태: • 모든 `ACU` 및 `IP 리더기`의 네트워크 연결 상태 (Online/Offline)를 실시간으로 모니터링합니다.	1. 가상의 `Device Service` (또는 `Access Control Service`)가 모든 등록된 단말기에 주기적으로 **Heartbeat(Ping)**를 보냅니다. 2. 3회 이상 응답이 없는 장치는 'Offline'으로 간주합니다. 3. 상태 변경(Online↔Offline)이 감지되

기능 분류	상세 기능	아키텍처 구현 (연동 서비스)
		면, `Socket.IO`를 통해 `Admin UI`의 장치 상태 대시보드를 즉시 업데이트합니다.
3.2. 감사 로그 및 통계	**출입 기록 상세 검색:** • `Admin UI`에서 기간, 사용자, 구역, 결과(DENY 사유) 등 상세 조건으로 과거 출입 기록을 검색하고 엑셀로 다운로드합니다.	1. `Admin UI`가 `API Gateway`(`/admin/logs`)로 상세 검색 API를 호출합니다. 2. `API Gateway`가 요청을 `Log Service`로 라우팅합니다. 3. `Log Service`는 이 요청을 `PostgreSQL`의 `access_logs` 테이블에 대한 SQL 쿼리로 변환하여 실행하고, 결과를 페이징하여 반환합니다.
	관리자 활동 로그: • "어떤 관리자가 언제 정책을 수정했는지" 등 모든 관리자 활동을 추적/감사할 수 있도록 기록합니다.	1. `API Gateway`는 `/admin/**` 경로의 `POST/PUT/DELETE` 요청을 받을 때마다, 해당 요청 정보를 `Kafka`의 `audit_admin_events` 토픽으로 발행합니다. 2. `Log Service`가 이 토픽을 구독하여 `PostgreSQL`의 `admin_audit_logs` 테이블에 영구 저장합니다.
	통계 보고서: • 일별/구역별 출입량, 특정 시간대 병목 현상, 가장 많이 거부된 사용자 등을 시각화하여 분석합니다.	• `Log Service`가 `/admin/logs/summary` 같은 별도 API를 제공합니다. 이 API는 `PostgreSQL`에서 `COUNT(*), GROUP BY date, zone` 등 미리 정의된 집계 쿼리를 실행하여 결과를 반환합니다. • 또는, `Grafana`가 `PostgreSQL`을 데이터 소스로 직접 연결하여 더 복잡한 시각화 대시보드를 구축합니다.
3.3. CCTV 영상 관제 연동	**이벤트-영상 연동:** • `Admin UI`에서 "오후 3:05, 서버실, 강제 개방 알람" 로그를 클릭하면, **그 시간에 해당 문을 비추던 CCTV 영상**이 팝업으로 즉시 재생됩니다.	1. (사전 작업) `Admin UI`의 `2.3. 장치 관리`에서 'Door101'에 'Camera_ID_5'와 'NVR_Server_IP'를 매핑해 둡니다. 2. `Admin UI`가 로그 클릭 시, `Log Service`에 `Event_ID`와 `Camera_ID`를 보냅니다. 3. `Log Service`는 별도의 **영상관제(NVR/VMS) 서버 API**를 호출하여, `Camera_ID_5`의 `Timestamp` 부근의 영상 스트리밍 URL을 요청합니다.

기능 분류	상세 기능	아키텍처 구현 (연동 서비스)
		4. `Log Service`가 이 URL을 `Admin UI`로 반환하고, `Admin UI`가 비디오 플레이어로 재생합니다.

4. 시스템 운영 및 고가용성(HA) 기능 (상세)

`sdac-infra-layer-guide.md`에 기반한 시스템 자체의 안정성 기능입니다.

기능 분류	상세 기능	아키텍처 구현 (연동 서비스)
4.1. 자동 장애 복구	**애플리케이션 Failover:** • `Node 1` 서버 다운 시, `Node 1`의 모든 서비스(Gateway, Access Control 등)가 `Node 2, 3`에 **자동으로 재시작**됩니다.	1. `Docker Swarm` 매니저(Node 2, 3)가 `Node 1`의 Heartbeat 중단을 감지합니다. 2. **과반수(Quorum)**(2/3)가 유지됨을 확인하고, 리더를 재선출합니다. 3. Swarm 매니저는 `Node 1`에서 실행되던(`replicas=2` 목표) 컨테이너들이 사라진 것을 인지합니다. 4. 평상시 **60%만 사용**하던 `Node 2, 3`의 **40% 예비 공간(Headroom)**에 `Node 1`의 컨테이너들을 즉시 재배치(Reschedule)하여 `replicas=2` 목표를 복원합니다.
	데이터베이스 Failover: • `DB1`(Primary) 서버 다운 시, `DB2`(Standby)가 **자동으로 `Primary`로 승격**됩니다.	1. `Patroni`가 `DB1`의 Heartbeat 중단을 감지합니다. 2. `Patroni`는 `DB2`를 즉시 `Primary`로 승격시킵니다. 3. `Patroni`는 HA-DB의 가상 IP(VIP)를 `DB2`의 IP로 변경합니다. 4. `Spring Boot` 서비스들은 잠시 연결이 끊겼다가, **동일한 VIP**로 재연결하여 `DB2`로 자동 접속됩니다.
	캐시/메시지 Failover: • `Redis` 또는 `Kafka` 노드 1대 다	• **Redis Cluster:** `Node 1`의 `Redis`(Primary Shard)가 다운되면, 클러스터는 즉시 `Node 2, 3`에 복제되어 있던 `Replica Shard`를 `Primary`로 승격시킵니

기능 분류	상세 기능	아키텍처 구현 (연동 서비스)
	운 시에도 클러스터가 데이터 유실 없이 서비스를 지속합니다.	다. 데이터 유실이 없습니다. • **Kafka Cluster:** `Node 1`의 `Kafka`(Partition Leader)가 다운되면, 클러스터는 즉시 `Node 2, 3`에 복제되어 있던 `Follower`를 `Leader`로 승격시킵니다. 데이터 유실이 없습니다.
4.2. 무중단 운영	**무중단 배포 (Rolling Update):** • `User Service` v1.1 배포 시, `Node 1`에 v1.1을 먼저 띄우고, `Node 2`의 v1.0을 내리는 식으로 순차 배포하여 **서비스 중단 없이** 업데이트합니다.	• `docker service update --image user-service:1.1 user-service` 명령 실행. • Swarm은 `replicas=2`를 유지하기 위해, v1.1 컨테이너 1개를 먼저 `Node 1`에 띄워 `Running` 상태를 확인합니다. • v1.1이 정상이면, `Node 2`의 v1.0 컨테이너 1개를 내립니다. (순차적 롤링)
	동적 설정 변경: • DB 주소, JWT 비밀키 등 중요 설정 변경 시, **서비스 재배포 없이** 설정을 갱신하여 적용합니다.	1. 관리자가 `Spring Cloud Config Server`의 백엔드(예: Git)에 설정을 Push합니다. 2. 관리자가 `API Gateway`를 통해 `User Service`의 `/actuator/refresh` 엔드포인트를 호출합니다. 3. `User Service`가 `Config Server`로부터 **변경된 설정만** 다시 읽어와 즉시 적용합니다. (서비스 재시작 불필요)
4.3. 성능 및 장애 예측	**실시간 성능 감시:** • 모든 노드(CPU/RAM)와 서비스(응답 속도, JVM)의 성능 지표를 실시간으로 수집합니다.	• `Prometheus`가 모든 Spring Boot 서비스의 `/actuator/prometheus` 엔드포인트를 주기적으로 스크랩(Scrape)하여 지표를 수집합니다. • `Grafana`가 이 데이터를 시각화하여 "App Cluster 평균 CPU 60%" (HA 예비 공간 40% 남음) 등을 보여줍니다.
	장애 전파 차단: • `Policy Service`가 느려지면, `Access Control Service`가 `Policy Service` 호출을 **자동으로 차단(Circuit Breaking)**하여 장애가 전파되는 것을 막습니다.	1. `Spring Cloud Circuit Breaker (Resilience4j)`가 `Policy Service`의 응답 지연 횟수를 감지합니다. 2. 임계치 초과 시, `Access Control Service`는 `Policy Service` 호출을 1분간 차단(Open)하고 즉시 '기본 응답'을 반환

기능 분류	상세 기능	아키텍처 구현 (연동 서비스)
		합니다. 3. 1분 후 `Half-Open` 상태로 1건의 테스트 호출을 보내, 성공 시 `Closed` (정상)으로 복귀합니다. 4. `Prometheus/Grafana`가 이 상태(Open/Closed)를 실시간으로 대시보드에 표시합니다.
	중앙 로그 분석: • 모든 컨테이너의 시스템 로그(Error, Stack Trace)를 중앙에서 검색하여 장애 원인을 신속히 분석합니다.	1. `Fluentd`가 **Global Service**로 모든 노드(`Node 1, 2, 3, DB1, 2`)에 1개씩 배포됩니다. 2. `Fluentd`는 각 노드의 Docker 데몬 로그(`/var/run/docker.sock`)를 스트리밍하여 모든 컨테이너의 표준 출력(stdout/stderr) 로그를 수집합니다. 3. 수집된 로그를 `EFK/ELK Stack`(Elasticsearch)으로 전송하여 `Kibana`에서 검색할 수 있게 합니다.