운영 설정 자동 반영 설계

DB 설정을 재기동 없이 자동으로 반영하는 방법

LHG 2026-08-29

CONTENTS

이 글에서 쓰는 용어

처음 보는 사람도 이해되게 먼저 풀어둔다

- `ag_sys_config` : 운영자가 바꾸는 설정값을 저장하는 테이블
- **확인 간격(ttl-seconds)**: DB 변경 여부를 확인하는 최소 간격. 이름은 TTL이지만 값의 만료 시간이 아니다
- **HIL(Human-in-the-loop)**: 그래프가 사용자 승인·추가 입력을 기다리며 멈춰 있는 지점
- `db-over-yml` : DB에 있는 설정값이 YAML 파일보다 우선하도록 만드는 플래그
- `lru_cache` : 계산 결과를 메모리에 저장해 뒀다가 재사용하는 파이썬 캐싱 기능

운영자가 `ag_sys_config`를 수정하면
다음 사용자 요청이 들어올 때 서버가 스스로 설정을 다시 읽는다.

1. 전체 흐름과 10초의 의미

재기동도, 배치도, 인스턴스 목록도 필요 없다

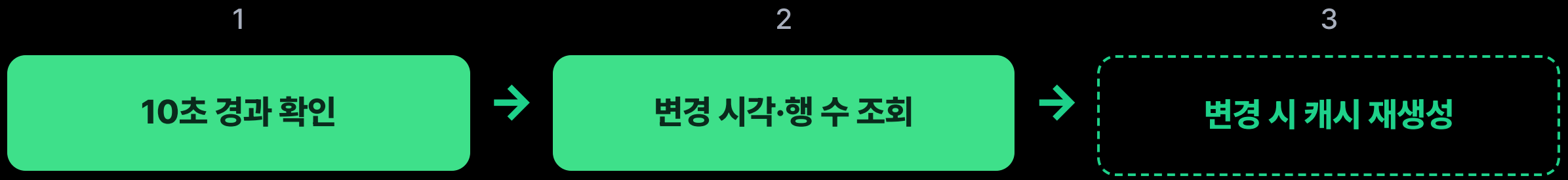
무엇이 좋아지나

확인 간격 10초

- 서버를 재기동하지 않아도 일반 설정이 반영된다
- 별도 배치나 인스턴스 목록 없이, 서버가 여러 대여도 각자 스스로 최신 설정에 맞춰진다

요청이 들어올 때 큰 흐름

LHG



- 변경이 없으면 기존 설정으로, 있으면 캐시를 비우고 다시 만든 뒤 처리한다

10초는 기다리는 시간이 아니다

LHG

```
사용자 요청 도착
→ 마지막 DB 확인 후 10초가 지났나?
  → 아니오: DB를 조회하지 않고 기존 설정으로 바로 처리
  → 예: 지금 바로 DB 조회
```

요청마다 간단한 시간 비교는 하지만, 요청마다 DB를 조회하지는 않는다.

ttl-seconds라는 이름의 함정

LHG

> 이름은 TTL이지만 실제 의미는 "DB 변경 여부를 확인하는 최소 간격"

- 설정값이 10초 뒤에 만료된다는 뜻이 아니다
- 10초가 지난 뒤 들어오는 첫 요청만 DB를 조회한다

변경 시간은 이렇게 비교한다


```
서버가 이전에 기억한 max(updated_at): 2026-08-20 10:00:00
운영자가 설정 수정 → 현재 max(updated_at): 2026-08-29 15:30:00

다음 DB 확인 → 시간이 다름 → 변경으로 판단
→ ag_sys_config 전체 다시 읽기 → 캐시 갱신
```

서버가 DB를 3초 전에 확인했다면 바로 다음 요청은 기존 설정을 그대로 쓴다.

두 가지 시간, 두 가지 역할

LHG

마지막 DB 확인 시각

- 10초가 지났는지 판단하는 데 쓴다

max(updated_at)

- 실제로 값이 바뀌었는지 판단하는 데 쓴다

2. 서버를 처음 켤 때

DB 접속 정보만큼은 DB에서 읽을 수 없다

접속 정보는 YAML·환경변수에서 나머지는 DB에서

DB에 접속하려면 먼저 DB 주소를 알아야 한다

- 최초 DB 접속 정보는 YAML과 환경변수로 확보한다
- 이후 `application.yaml` → 프로파일별 YAML → `ag_sys_config` → 환경변수 순서로 값을 합쳐 최종 설정을 만든다

우선순위: 환경변수 > DB > YAML

기본 운영 설정

- db-over-yml: true
- 환경변수가 가장 먼저, 그다음 DB가 YAML보다 우선

로컬 프로파일

- 테스트 편의를 위해 YAML을 우선하도록 예외 설정

get_settings()는 한 번만 계산한다

- 매번 DB를 읽지 않도록 최종 설정을 메모리에 한 번 저장한다
- 이 설정으로 그래프, 워커 목록, 실행 계획, 프롬프트 설정 등을 만든다
- 서버 기동이 끝나면 그 시점의 `ag_sys_config` 상태를 비교 기준값으로 저장한다

3. 사용자 요청이 들어올 때

타이머가 아니라 요청이 확인을 시작한다

확인 대상 요청 6가지


```
/agent/execute
/agent/stream
/agent/replay
/agent/replay/stream
/agent/hil/respond
/agent/hil/stream-respond
```

이 요청들이 들어올 때만 설정 변경 여부를 확인한다.

요청마다 DB를 조회하지는 않는다

LHG

- 마지막 확인 후 10초가 지났을 때만 다시 확인한다
- 동시에 여러 요청이 들어와도 그중 하나만 확인을 수행한다
- 요청이 없는 서버는 DB를 계속 조회하지 않고 첫 요청이 올 때 비로소 확인한다

상태 1: 사용자 요청이 들어온 상태


```
사용자 요청 도착
→ 마지막 DB 확인 후 10초가 지났는지 확인
→ 지나지 않음: 기존 설정으로 바로 처리
→ 지남: DB 변경 여부 확인
→ 변경 없음: 기존 설정으로 처리
→ 변경 있음: 캐시 갱신을 최대 3초 기다린 뒤 처리
```

설정 확인은 그래프 실행 전에 이뤄진다.

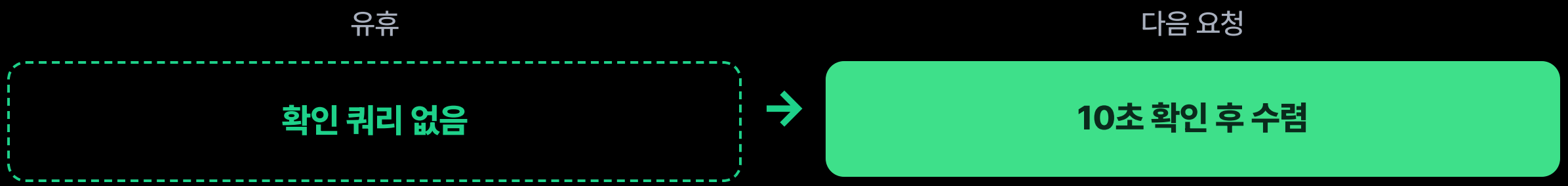
상태 2: HIL 응답을 기다리는 상태


```
그래프가 사용자 승인·추가 입력을 기다림
→ 별도 설정 확인 없음
→ HIL 응답 요청 도착
→ 10초 확인 간격이 지났으면 DB 변경 여부 확인
→ 변경이 있으면 캐시 갱신을 기다린 뒤 그래프 재개
```

설정 캐시를 갱신해도 대기 중인 LangGraph `MemorySaver` 상태는 그대로 유지한다.

상태 3: 요청이 없는 유휴 상태

LHG



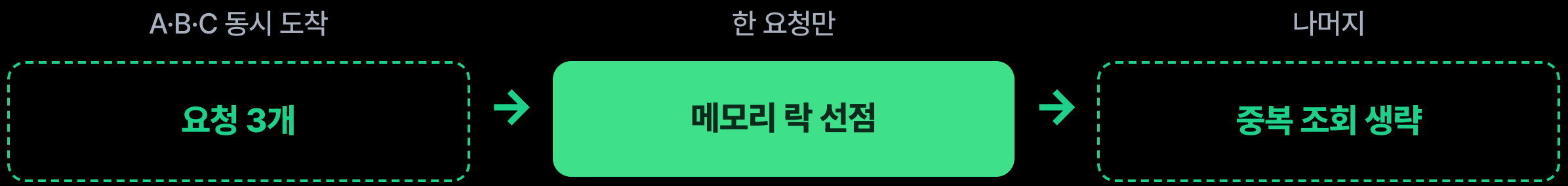
- 별도 폴링이나 배치가 돌지 않아 불필요한 DB 조회가 없다
- 각 인스턴스는 자신에게 다음 요청이 들어오는 시점에 맞춰 최신 설정으로 수렴한다

4. 다중 요청이 들어올 때

요청이 한 번에 몰리거나 서버가 여러 대라면 달라진다

같은 worker 프로세스에 몰리면 하나만 확인한다

LHG



- 먼저 도착한 요청이 아니라 락을 먼저 획득한 요청이 확인을 담당한다
- 변경이 있으면 그 요청이 설정 갱신을 시작하고 최대 3초까지 기다린다
- 확인 간격을 0으로 설정하면 모두 조회하지만 갱신 작업은 하나로 합친다

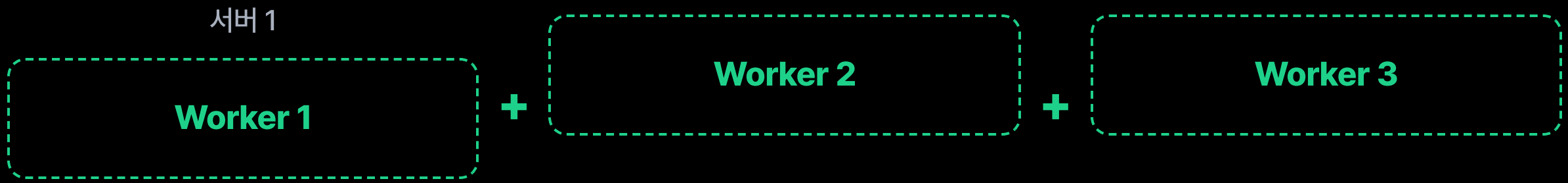
짧은 불일치 구간이 생길 수 있다

설정 전환 순간, 같은 프로세스 안에서도 확인 담당 요청은 새 설정 / 나머지는 이전 설정

- DB 중복 조회와 요청 전체 대기를 줄이기 위한 선택이다
- 모든 요청이 반드시 동시에 새 설정을 써야 한다면 갱신 중인 요청도 `_refresh_task`를 함께 기다리도록 보완해야 한다

서버 한 대에도 worker 프로세스는 여러 개다

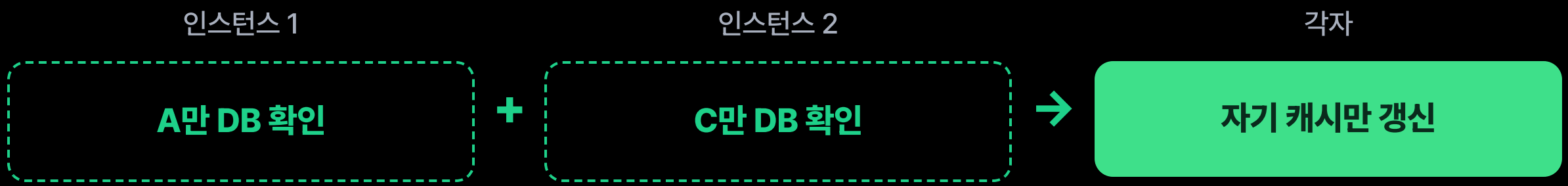
LHG



- 각 worker 프로세스는 자기만의 메모리 락과 마지막 DB 확인 시각을 가진다
- 확인 중복 방지 범위는 서버 전체가 아니라 worker 프로세스 하나다

여러 인스턴스에 몰리면 각자 확인한다

LHG



- 마지막 확인 시각·락·캐시는 인스턴스와 worker 프로세스끼리 공유하지 않는다
- 다른 인스턴스에 갱신을 전파하지 않고 각자 스스로 다시 읽는다

인스턴스 3대, 요청 300개라도 쿼리는 약 3건

LHG

300건이 아니라 활성 worker당 1건

- 각 인스턴스가 worker 1개일 때 기준이며, worker가 여러 개면 그만큼 쿼리도 늘어난다
- 같은 worker 프로세스 안에서는 한 요청만 확인하므로 요청 수만큼 쿼리가 늘어나지 않는다

DB 변경 직후 요청이 들어오면

LHG

```
반영 시점은 "DB 변경 후 정확히 10초"가 아니라
"확인 간격이 지난 뒤 들어온 첫 요청"이다
```

10초가 지나지 않았으면 그 요청은 기존 설정을 쓰고 지난 뒤 첫 요청이 변경을 감지한다.

갱신 중 다시 바뀌거나, 실패하면

LHG

- DB가 갱신 도중 한 번 더 바뀌면 다음 확인에서 차이를 다시 발견해 한 번 더 갱신한다
- 변경 확인이나 설정 읽기가 실패하면 기존 설정으로 처리하고 실패한 인스턴스만 다음 확인 시점에 재시도한다
- 갱신이 3초를 넘으면 기다리던 요청도 기존 설정으로 진행한다

5. 요청 기반으로 만든 이유

타이머 대신 요청이 확인을 트리거하도록 설계했다

이렇게 만든 이유

- 유휴 인스턴스가 10초마다 불필요하게 DB를 조회하지 않는다
- 인스턴스 주소 목록이나 중앙 전파 서버가 필요 없다
- 인스턴스가 늘거나 줄어도 별도 등록 없이 각자 최신 설정으로 맞춰진다
- DB 장애가 사용자 요청 전체 장애로 번지지 않는다

이 방식이 안 맞는 경우도 있다

LHG

요청 기반 (지금 방식)

- 요청이 없으면 반영도 미뤄질 수 있다

백그라운드 폴링·알림

- 요청 유무와 상관없이 10초 안에 모든 인스턴스가 같은 설정이어야 할 때 더 적합하다

변경 여부 확인 쿼리


```
select max(updated_at), count(*)
from ag_sys_config
where config_value is not null
```

`max(updated_at)`은 최근 수정 시간, `count(*)`은 값이 있는 설정의 개수다.

행 수를 함께 보는 이유

LHG

설정이 삭제되면 updated_at은 남지 않지만 행 수는 바뀐다

- 전체 설정값을 매번 읽지 않고 변경 여부만 한 줄로 먼저 확인한다
- 실제 변경이 있을 때만 전체 설정을 다시 읽는다

변경이 확인되면 갱신되는 캐시들

- `get_settings()` 설정 정보, 기본 그래프와 HIL 재개용 그래프
- 실행 오케스트레이션 서비스, 워커·에이전트 모듈 목록
- 실행 계획·보고서 컬럼 규칙, JWT·임베딩 검색·출력 가드레일 설정

그래프 캐시는 새로 만들되, 실행 중 상태를 담은
LangGraph `MemorySaver` 는 그대로 유지한다.

반영에는 최대 3초까지만 기다린다

LHG

- 새 설정을 먼저 검증하고 성공했을 때만 기존 캐시를 비우고 적용한다
- 3초 안에 끝나면 최신 설정으로 처리한다
- 시간을 넘기면 기존 설정으로 처리하고 다음 확인 때 다시 시도한다

6. 설정값과 예외 상황

모든 설정이 재기동 없이 바뀌는 것은 아니다

YAML 기본값

LHG

```yaml
agent:
  config:
    db-over-yml: true
    auto-refresh:
      enabled: true
      ttl-seconds: 10
      apply-timeout-seconds: 3
```

`ttl-seconds: 0` 이면 요청마다 확인하고 `apply-timeout-seconds: 0` 이면 기다리지 않는다.

확인이 실패해도 요청은 실패하지 않는다

LHG

- 변경 여부 조회에 실패하면 기존 설정으로 정상 처리한다
- 새 설정을 읽는 데 실패하면 기존 설정을 유지하고 다음 확인 시점에 다시 시도한다
- DB가 잠시 불안정해도 사용자 요청 전체가 설정 확인 때문에 실패하지는 않는다

재기동이 필요한 값도 있다

LHG

DB 주소·포트·커넥션 풀·체크포인트 저장 방식 → 변경은 감지되지만 적용에는 재기동 필요

- 기동 시점에 이미 결정되는 값은 자동 반영 대상이 아니다

한 줄 답변

서버는 최초 접속 정보로 `ag_sys_config`를 읽어 설정과 그래프를 만든다.
이후 사용자 요청이 올 때마다 10초 간격으로 `max(updated_at)`과 행 수를 확인해,
변경이 있으면 `get_settings()`와 관련 캐시(`lru_cache`)를 비운 뒤
최신 설정으로 그래프와 서비스를 다시 만든다.