aws FOR GAMES

# GAMES ON AWS 2026

GS Neotek | MEGAZONE CLOUD

GAMES ON AWS 2026

aws FOR GAMES

# 레거시를 넘어 클라우드 네이티브로

## AWS Transform 기반 게임 플랫폼 DB 현대화 스마일게이트 STOVE 사례


**강병천**

Solutions Architect

AWS

**이병민**

DB플랫폼팀 DBA

스마일게이트


GS Neotek | MEGAZONE CLOUD

# 목차

**PART 1**
현대화 개념부터 AWS Transform까지

01 현대화의 비즈니스 가치

02 AI 자동 변환의 한계

03 AWS Transform 소개

**PART 2**
스마일게이트 STOVE 사례

04 AWS Transform 활용 흐름

05 검증 기준과 보정

06 사람의 역할과 교훈

# 개요 : 레거시 애플리케이션 현대화

# 게임 플랫폼, 인터넷 시대의 기술 집약 서비스

**이커머스**
대규모 거래와 결제

**SNS**
관계와 실시간 소통

**게임 플랫폼**
계정부터 결제, 커뮤니티까지

- 계정과 인증
- 결제
- 커뮤니티
- 데이터와 분석

그동안은 게임이라는 주 콘텐츠의 **조력자 역할**

# 게임 플랫폼이 마주한 비즈니스 고려 사항

## 01

### 라이선스가 만드는 비용 구조

필요한 기능이 곧
비용 구간을 바꾼다

→ 비용 최적화

## 02

### 클라우드 네이티브하지 않은 구성

탄력성과 글로벌 확장이
막힌다

→ 글로벌 확장

## 03

### 레거시가 만드는 변경의 어려움

바꾸기 어려워
속도를 못 맞춘다

→ 빠른 기능 출시

현대화는 기술 과제가 아니라 **‘비즈니스 결정’**이다

GAMES ON AWS 2026

aws FOR GAMES

# AI 자동 변환 : 어디까지 맡길 수 있나

GS Neotek | MEGAZONE CLOUD

# 어려웠던 문제, 이제는 다르게 접근할 수 있다

지금까지

재작성은 몇 년짜리
차세대 프로젝트라
계속 미뤄졌다

›

이제는

AI가 재작성의
상당 부분을
대신한다

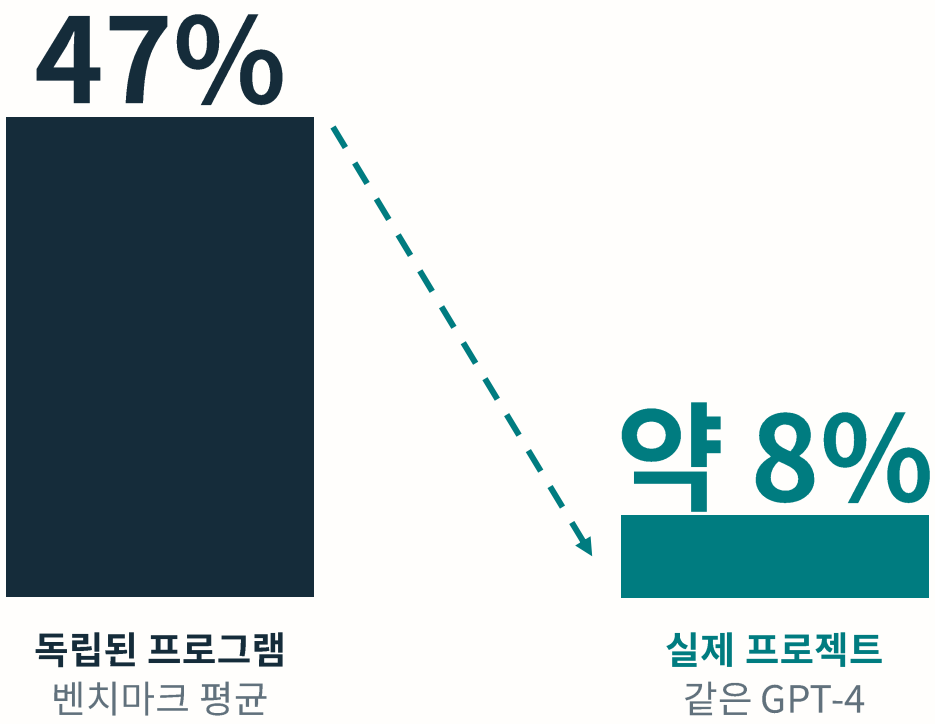
›

남은 질문

그렇다면 AI를
어떻게 써야 할까?

**AI에게 통째로 맡기면 될까?**

# 통째로 맡기면, 실제 프로젝트에서는 열에 하나도 안 된다




특히 어려웠던 지점

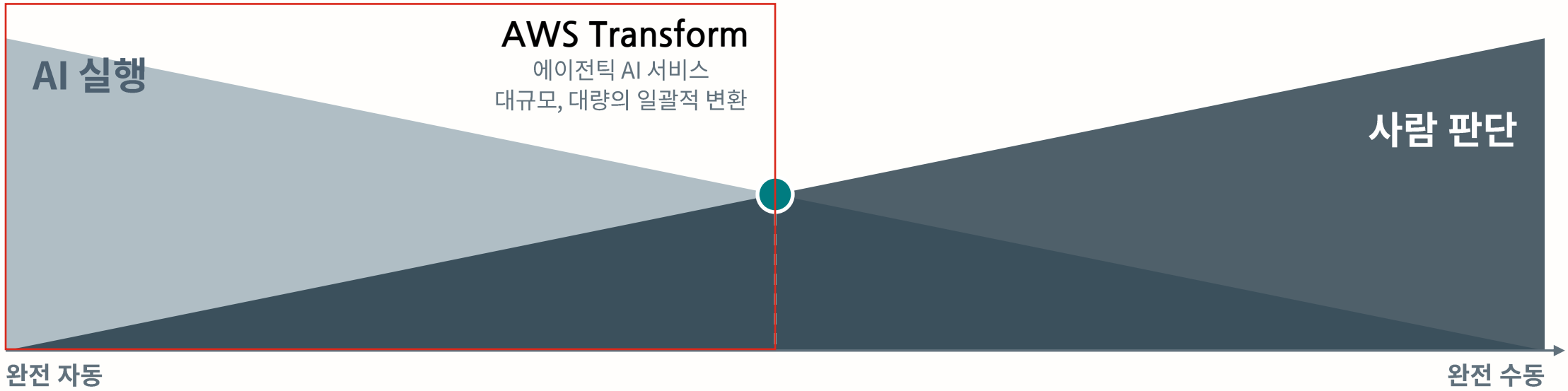
- 상속
- 오버로딩
- 어노테이션
- 메서드 간 의존

## 사람이 판단하고, AI가 실행한다

Pan et al., Lost in Translation, ICSE 2024 (arxiv.org/abs/2308.03109) · 성공 = 수정 없이 테스트 통과

# 자동화의 스펙트럼

따라갈 규칙이 코드에 있느냐가 경계를 정한다




## 코드가 곧 규칙

표준 SQL
표준 데이터 접근 패턴

→ 자동 변환

## 코드 밖에 있는 경우가 많음

자체 데이터 접근 래퍼
DI·설정·로깅 방식 · 엔진 고유 기능

→ 사람 판단 + AI 실행

## 대응하는 규칙이 없음

레거시 COM · 네이티브
Windows 전용 의존성

→ 재작성

GAMES ON AWS 2026

aws FOR GAMES

Windows 현대화 측면 (.NET/MS SQL)

# AWS Transform : 무엇이고, 무엇이 다른가

GS Neotek | MEGAZONE CLOUD

# *AWS Transform:* 모든 도구. 모든 에이전트. 하나의 플랫폼

**KIRO AGENT BUILDER**
새로운 에이전트를 구축하고 *AWS Transform*에 등록

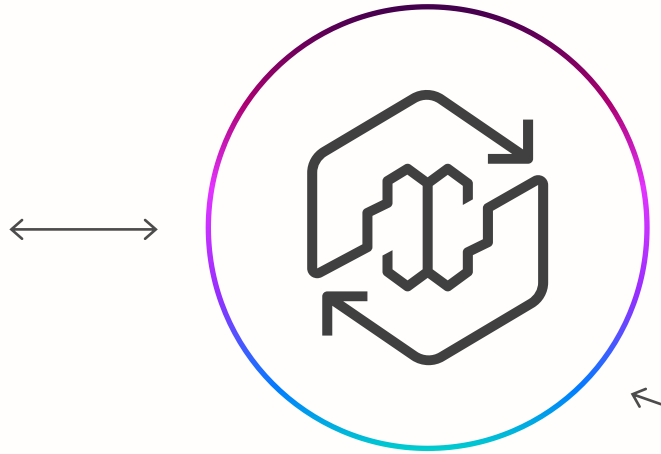
**CUSTOM OOB**
플랫폼 내에서 즉시 사용 가능한 솔루션을 생성하고 게시

**AWS Marketplace**
서드 파티 에이전트를 가져와서 *AWS Transform* 내부에서 실행

**확 장**

**CLI**
명령줄에서 변환을 시작하고 웹에서 추적하세요

**IDE**
편집기를 벗어나지 않고 작업을 시작하고, 진행 상황을 모니터링하고, 결과를 가져올 수 있습니다



**KIRO**
*Kiro*를 사용하여 결과를 직접 로드하고 다음 단계를 실행합니다

**통 합**

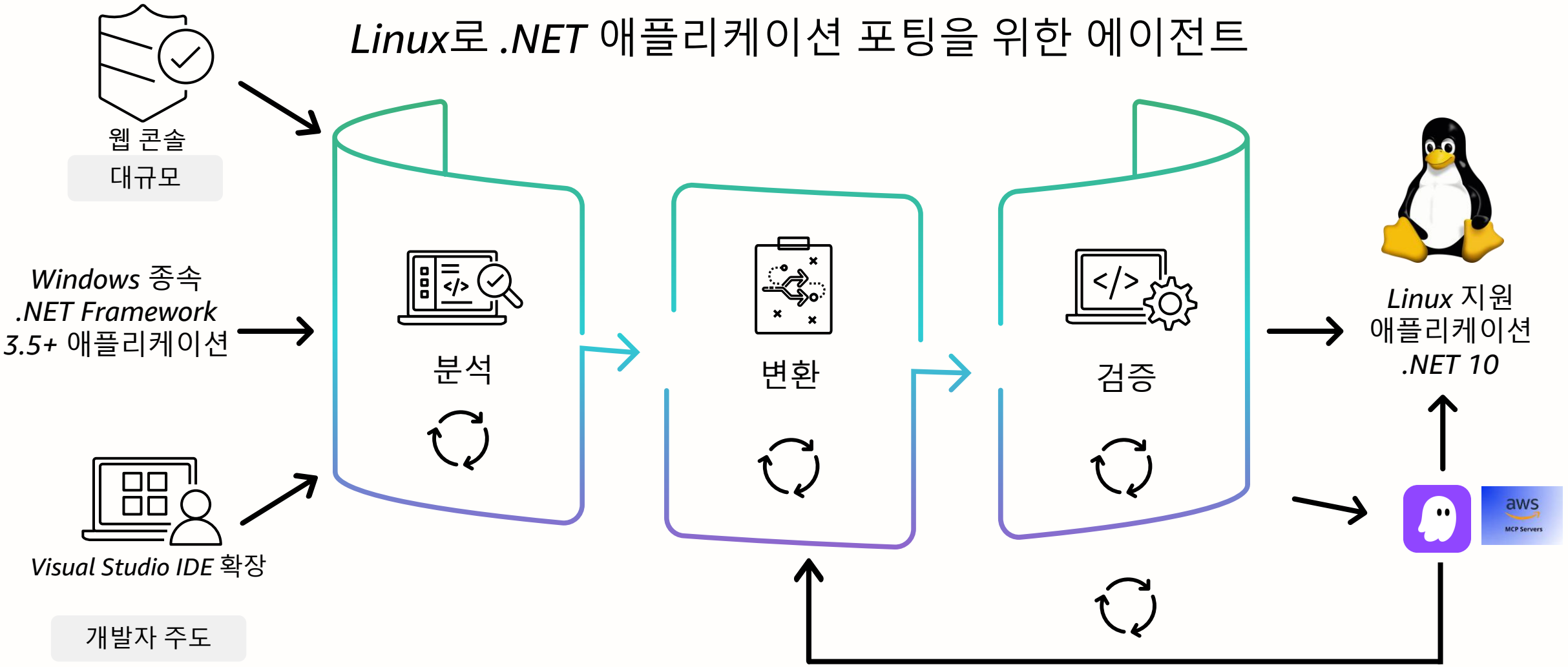
**MCP/A2A Gateway**
등록된 *Transform* 에이전트에 액세스

**코딩 에이전트**
*Kiro* 또는 다른 코딩 에이전트를 통해 *Transform* 에이전트와 상호 작용

**진 입 점**

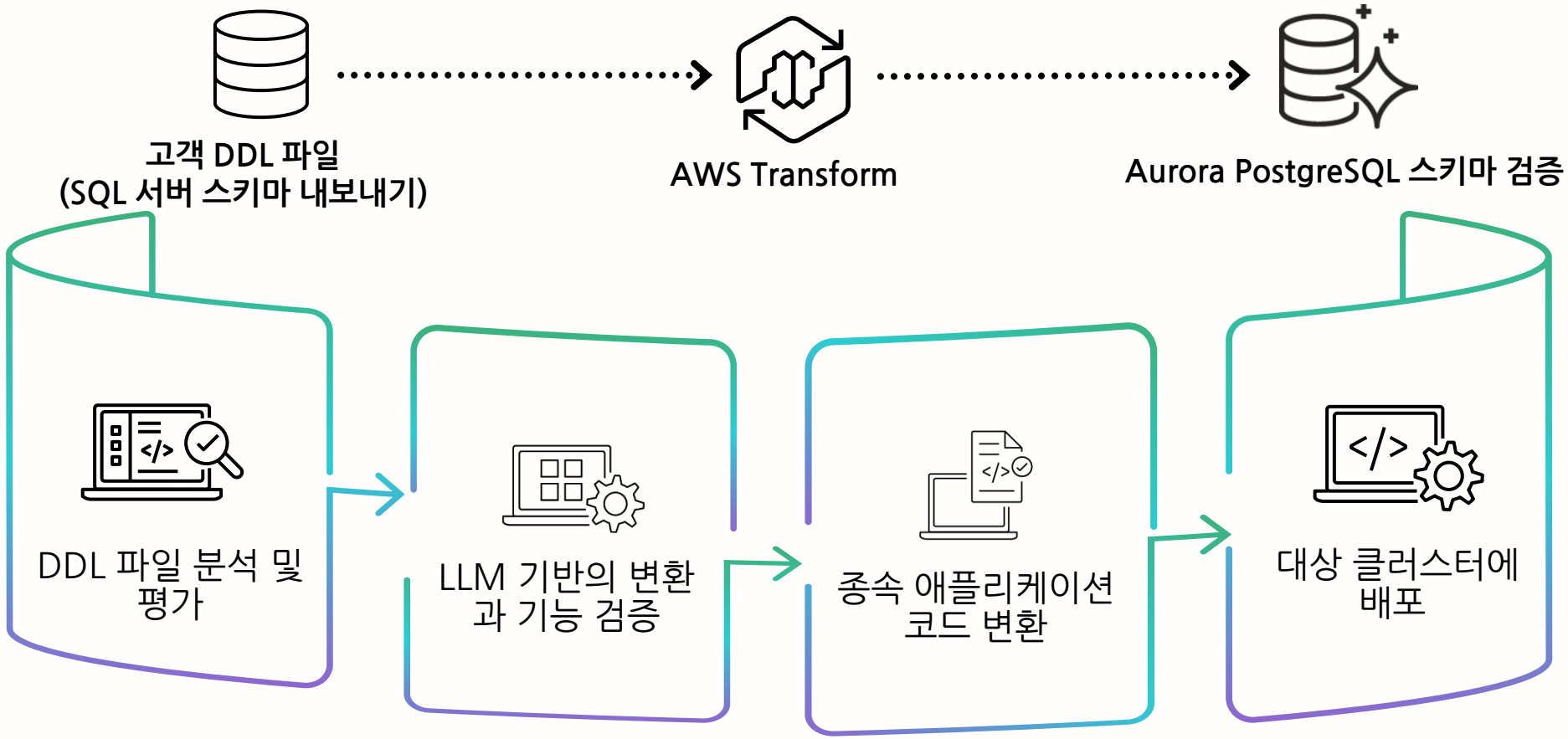
# .NET 변환 에이전트

## Linux로 .NET 애플리케이션 포팅을 위한 에이전트



# SQL 서버 현대화를 위한 AWS Transform

## 오프라인 소스 기능



# 로컬 *IDE*에서 계속적인 스키마 변환

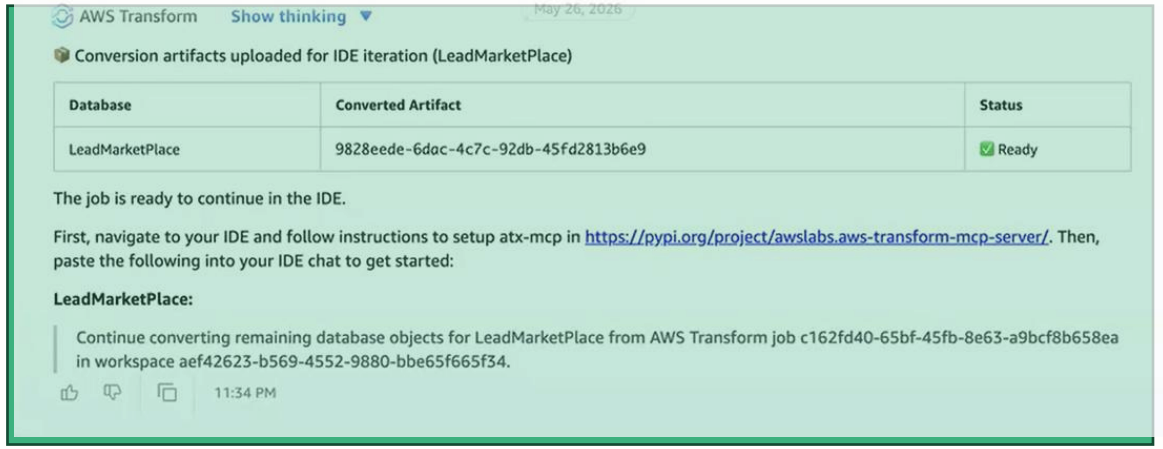
## 선호하는 코드 도우미를 사용

*AWS Transform*은 자세한 정보 *(*전체, 부분, 혹은 변환 할 수 없는 객체*)*를 제공하는 스키마 변환 리포트를 생성하고 이 후 수동적 작업을 위한 정보 제공

복잡한 변환 이슈에 대해서 *DBA/*개발자 주도로 선택한 *IDE*환경에서 변환 완료 가능

*AWS Transform MCP* 서버를 설정하여 아티팩트를 가져오고 보고된 이슈를 해결

변경 사항을 대상 클러스터에 배포하여 검증하고 이를 반복



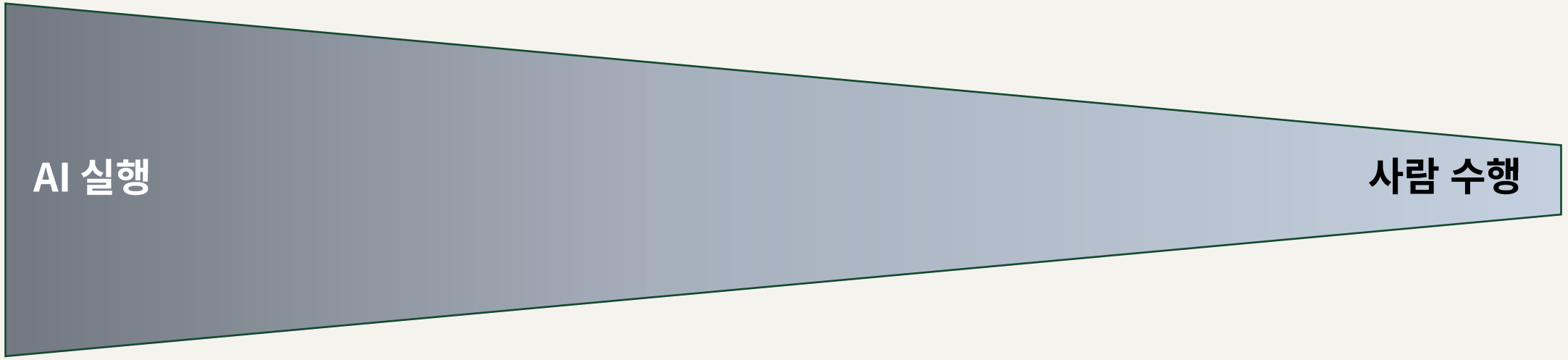

**Retry conversion**

**Address schema issues**

KIRO

# 변환 실행 체계

동일한 파이프라인, 동일한 맥락에서 가속화하여 변환을 마무리




## AWS Transform

대규모
일괄적 변환

→ 자동 변환

평가 문서
변환된 DDL, 변환된 종속 코드
변환 보고서 (경계확인)

## Kiro/Claude Code + DBA/개발자

AWS Transform 산출물을 컨텍스트로 활용
Remediation Fix, 검증 및 테스트

→ 사람 판단 + AI 실행

GAMES ON AWS 2026

aws FOR GAMES

# 스마일게이트 현대화 사례

이병민 · 스마일게이트 DB플랫폼팀 DBA

PoC 범위 — DB 와 앱 데이터 접근 코드의 변환 · 검증 (라이브 전환 · 성능 · API 트래픽 전환 제외)

GS Neotek | MEGAZONE CLOUD

# 스마일게이트 · STOVE — 게임 런칭 · 배포 플랫폼의 상점 DB

## STOVE · STOVEPAY

게임 런칭 · 배포 · 결제 플랫폼
매출이 지나가는 상점 DB

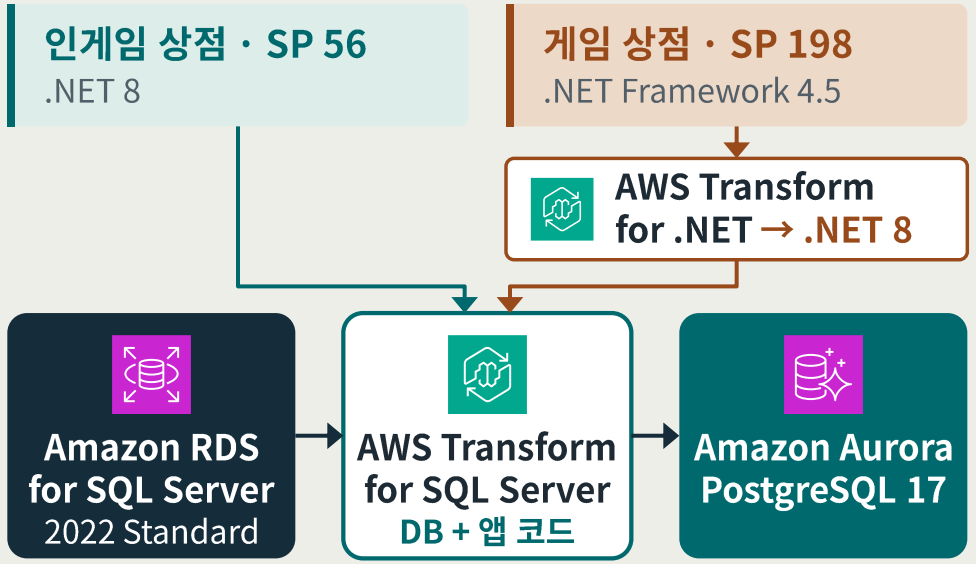
목표

## 효율적인 비용 절감

라이선스 제거 · 수작업 대신 Transform 으로 빠르게

**범위 밖** 라이브 전환 · 성능 · API 트래픽 전환

### 이번 PoC 범위




**호출 SP 252개** (공통 2개 중복 제외)

# 왜 상점 DB 인가 — 적당한 복잡도, 큰 비중

**파일럿 후보 — SQL Server 서비스 중 3곳**

| | | |
|---|---|---|
| **아이템** | 단순 구조 · 사용처 적음<br>전환 효용 · 학습 효과 낮음 | 제외 |
| **상점 (인게임 + 게임 상점)** | .NET 앱 · SP 중심<br>적당히 복잡 · 영향 큼 | ✓ 선정 |
| **결제 DB** | 결제 전 처리 · 주문 처리<br>멈추면 전 서비스 영향 | 후순위 |

**353**

SP 와 함수 — SP 342 · 함수 11

**98**

테이블 — 약 407만 행

**2**

호출 앱 — 인게임 상점 · 게임 상점

**상품 · 할인 · 구매 · 정산 — 매출이 지나가는 루틴(SP · 함수) 353개 · 그중 앱이 부르는 SP 252개가 검증 범위**

# 결론 먼저 — 요금과 작업 시간

공시 요금 — 도쿄 · 온디맨드 · 월

**SQL Server · db.r7i.4xlarge**

$2,721

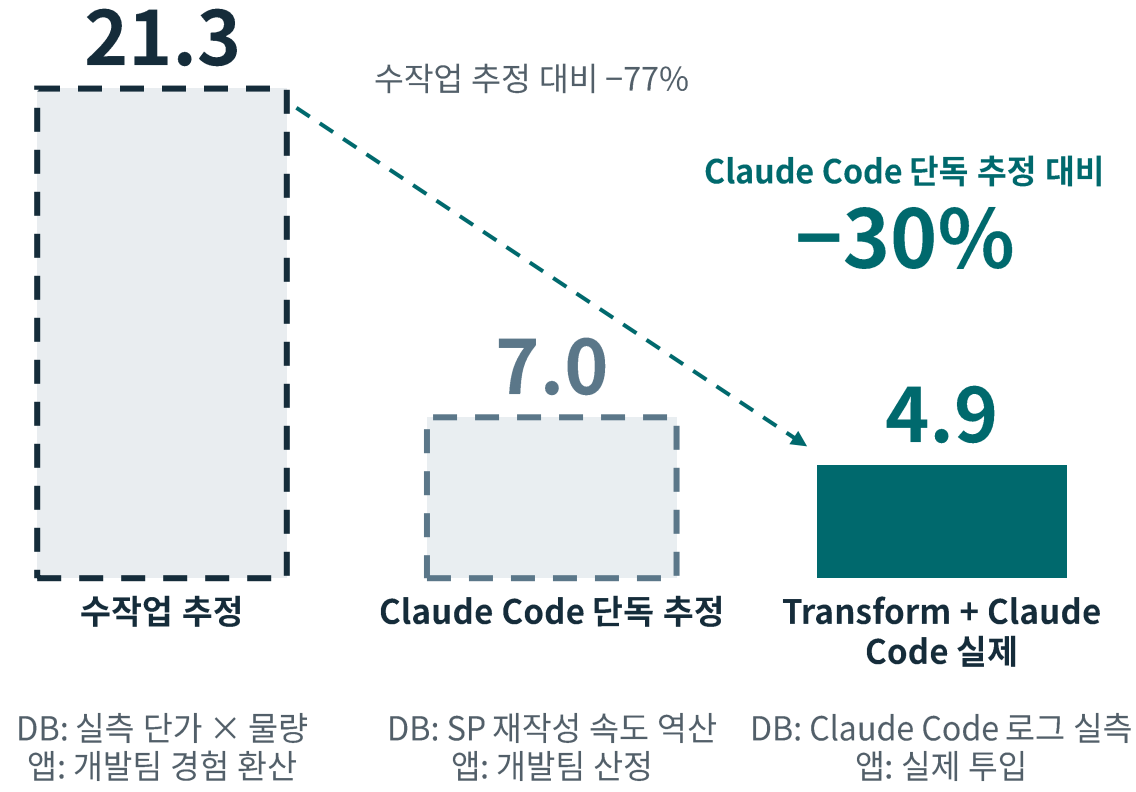
**Aurora PostgreSQL · db.r7g.4xlarge**

$1,943

**−29%**

SQL Server Standard · 라이선스 포함
스토리지 · I/O · 이중화 제외
공시 요금 비교 · 실측 절감액 아님

작업 시간 (MM) · DB + 앱 합산 · 점선 = 추정



GAMES ON AWS 2026

aws FOR GAMES

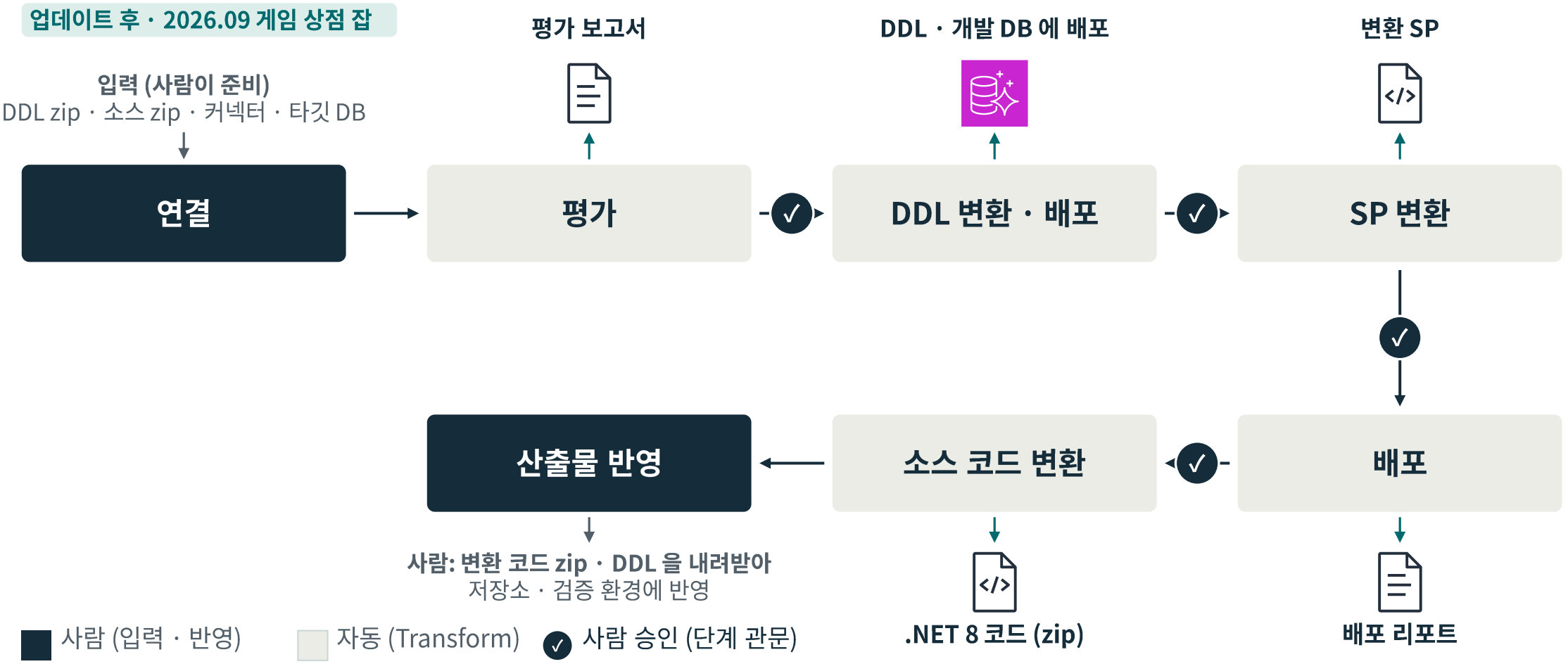
# 도구 선택과 활용 흐름 : Transform 을 어떻게 썼나

GS Neotek | MEGAZONE CLOUD

# 도구 선택 — 세 가지 후보 중 Transform 을 고른 이유

| | AWS SCT + AWS DMS | Babelfish | AWS Transform |
|---|---|---|---|
| **방식** | 규칙 기반 스키마 변환 + 데이터 이관 | T-SQL 호환 계층<br>변환 없이 Aurora 가 T-SQL 을 받아 줌 | 규칙 + LLM 에이전트<br>앱 코드까지 한 흐름 |
| **결과물** | PL/pgSQL 스키마 · 못 바꾼 것은 수작업 | 지원 범위의 T-SQL 을<br>Aurora 위에서 호환 실행 | 네이티브 PL/pgSQL + Npgsql 앱 코드 |
| **앱 수정** | 전부 수동 | 호환 범위에 따라 수정 | 데이터 접근 코드는 자동<br>호출 규약은 사람 |
| **이번 판단** | 참고 비교만<br>같은 DB 자동 · 최소 수정 29% | 평가 안 함<br>PL/pgSQL 네이티브 전환이 목표 | **채택 — 자동 61%**<br>**권장 액션 SCT 8,017 → 2,750** |

# Transform 활용 흐름 — 입력과 반영은 사람, 변환은 Transform



# 검증 자산 재사용 — 첫 앱 3.5주, 두 번째 앱 1주

**.NET 6 이상인가?**
AWS Transform for SQL Server 변환 대상

## 인게임 상점 .NET 8 · 호출 SP 56

| | | |
|---|---|---|
| 1 | SQL Server 변환 DB + 코드 | Transform 8~10시간 |
| 2 | DB 보정 | 사람 1.5주 |
| 3 | 원본과 값 비교 도구 제작 | 사람 2주 |
| 4 | 호출부 77곳 | 개발팀 3시간 |

**55 / 56** 호출 SP 검증 완료

검증 방법 · 도구 재사용

**3.5주 → 1주**
DB 보정 + 값 비교

## 게임 상점 .NET Framework 4.5 · 호출 SP 198

| | | |
|---|---|---|
| 1 | .NET 8 업그레이드 | Transform 7시간 / 사람 1주 |
| 2 | SQL Server 변환 DB + 코드 | Transform 약 2일 |
| 3 | DB 보정 + 값 비교 도구 재사용 | 사람 1주 |
| 4 | 호출부 144곳 | DB플랫폼팀 1일 + 개발팀 |

**195 / 198** 호출 SP 검증 완료

**한 번에 하나씩 바꾸고, 검증 방법은 재사용한다**

잡 시점 — 인게임 상점 7월(업데이트 전) · 게임 상점 9월(업데이트 후)

GAMES ON AWS 2026

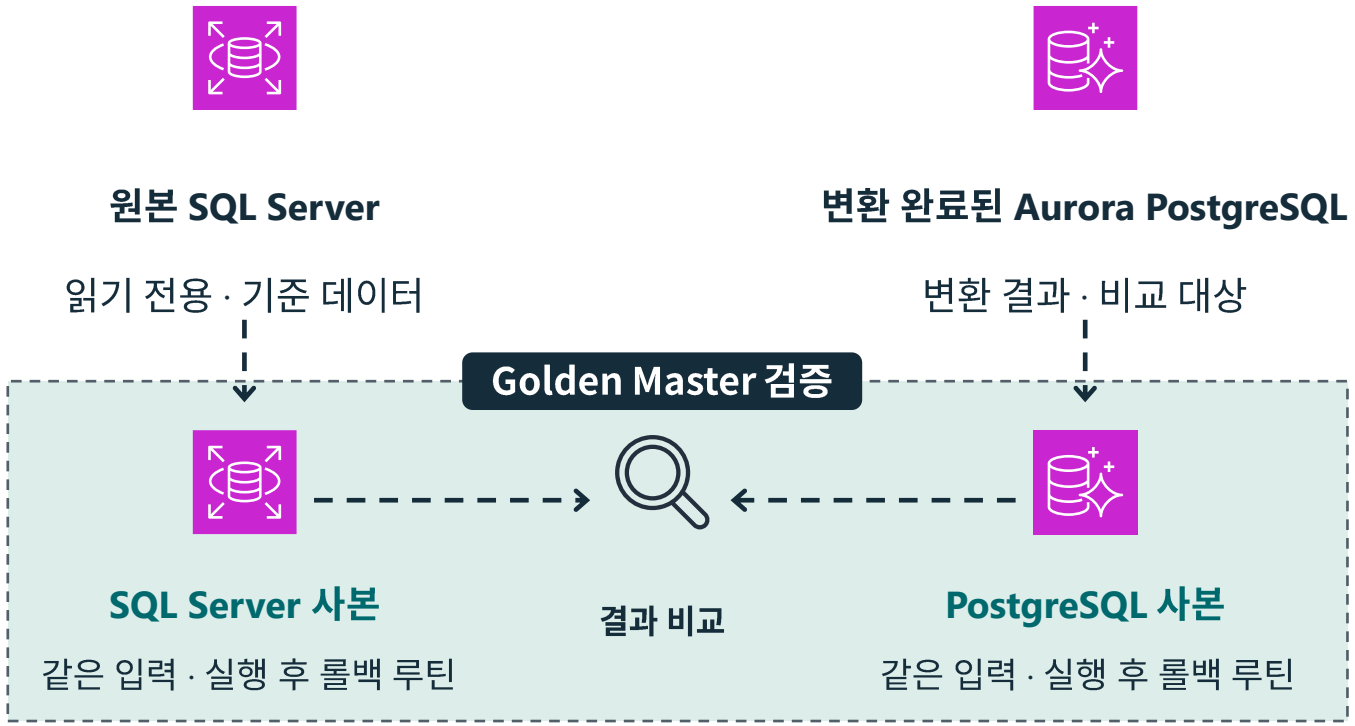
aws FOR GAMES

# 검증과 보정 :
# 무엇으로 판정하고 무엇을 고쳤나

GS Neotek | MEGAZONE CLOUD

# 검증 환경과 PASS 기준 — 같은 입력, 두 DB의 결과 비교

인게임 상점을 진행하며 만든 환경 · 게임 상점에 그대로 재사용




**롤백 루틴 — 생성 스크립트(전체 DDL) + 데이터(407만 행) 재적재로 원상 복구**
원본 SP 155개가 호출자 트랜잭션을 되돌림 → 원본엔 쓰기 금지, 쓰기 검증은 사본에서만

## 무엇이 같아야 PASS 인가

| | |
|---|---|
| 읽기 SP · 함수 | 조회 결과 + 반환값 |
| 쓰기 SP | 바뀐 건수 · 값 + 반환값 |
| API | HTTP 응답 + 오류 건수 |

**양쪽 빈 결과만으로 PASS 아님**

# 오류로 드러난 7가지

빌드 오류 | 실행 오류 | API 오류 → 원인별로 묶어 Transform 초안 위에서 일괄 수정

| Transform | 작업 유형 | Transform 결과 | 사람 + Claude Code 가 수정 |
|---|---|---|---|
| ✓ 완료 | **DB 연결 · 드라이버** | Npgsql 로 교체 | → 추가 수정 없이 그대로 사용 |
| △ 초안 Claude Code 수정 | **대소문자** | 스키마만 소문자로 | → 엔티티-테이블 매핑 보정 · 접두사 통일 |
| | **임베디드 SQL** | 호출문 변환 · 접두사 삭제 | → search_path 설정 — 설정 이슈 |
| | **데이터 유형 · 정렬** | 대소문자 무시 타입 적용 | → 형 변환 · ICU 콜레이션 후보 |
| | **SP 실행 · 타입 매핑** | CALL 호출 · 타입 불일치 | → 입력 jsonb 통일 · 호출 규약 정의 |
| | **결과 집합 · TVP** | refcursor 반환 · TVP 불일치 | → refcursor 수신 · TVP 는 복합 타입으로 |
| ✗ 재작성 | **트랜잭션 · 예외 처리** | 예외 처리 미변환 — 실제 예외가 가려짐 | → 예외 처리 로직 재작성 · 162 SP · 261곳 |

**Transform 먼저 — 1 완료 · 5 초안 위 수정 · 1 재작성, 초안이 있어 수정할 곳이 바로 보였다**

# 오류 없이 값만 달라진 4가지

**SQL 오류 0** | **빌드 · 배포 성공** | **API 정상 응답** → Golden Master 검증으로만 발견

## 1 OUTER APPLY → CROSS JOIN — 129곳

증상 할인 정보가 없는 상품 행이 결과에서 빠짐

영향 **상점 목록에서 상품이 사라져도 응답은 정상**

수정 LEFT JOIN LATERAL … ON TRUE 로 복원

## 2 SELECT INTO 0행 → NULL — 약 150곳

증상 조회 결과가 없으면 이전 값 대신 NULL

영향 **거절해야 할 요청(없는 상점)이 정상 처리됨**

수정 NOT FOUND 면 이전 값 복원 (COALESCE 는 오답)

## 3 GETDATE → clock_timestamp — 626곳

증상 한 문장으로 만든 행들의 생성 시각이 제각각

영향 **같은 시각이어야 할 기록이 서로 어긋남**

수정 (SELECT clock_timestamp()) — 문장당 한 번 계산

## 4 XML 노드 처리 → 첫 노드만 — 314곳

증상 여러 노드 중 첫 노드 값만 추출

영향 **여러 항목을 넘겨도 일부만 반영됨**

수정 XMLTABLE 헬퍼로 모든 노드 읽기

# Transform 이 놓친 빈틈을 다음 전환의 자산으로

같은 입력으로 비교 → 원인 확인 → 같은 패턴 전체 수정 → 재검사

| 발견한 빈틈 | 남긴 자산 | 다음 전환에서 할 일 |
|---|---|---|
| **값만 조용히 달라짐** | 값 비교 도구 · 결함 카탈로그 | 같은 패턴부터 찾아 비교 |
| **앱 · DB 호출이 어긋남** | 호출 규약 표 · 공통 코드 6파일 | 호출부를 한 번에 맞춤 |
| **주석 처리된 SP 32개** | 전체 DDL · 빈 DB 재적용 절차 | 본문 대조 후 빈 DB 적용 |
| **실행 오류가 숨음** | PASS 기준 · 수정 59묶음<br>회귀 검사 스크립트 | 이전 통과분도 재검사 |

**도구 · 절차 · 기준을 다음 전환의 출발점으로**

# AWS 실행체계와 우리 단계

**AWS 권고 — 앞 AWS 파트의 '변환 실행 체계'**

**AWS Transform**
대규모 · 일괄 변환
**→ 자동 변환**

산출물
평가 문서 · 변환된 DDL
종속 코드 · 변환 보고서

Kiro/**Claude Code + DBA/개발자**
Remediation Fix · 검증 · 테스트
**→ 사람 판단 + AI 실행**

**스마일게이트 실제 — 같은 틀, 같은 구간**

**AWS Transform ✓**
평가 → DDL → SP → 배포 → 코드 변환
**→ 자동 변환 ✓**

**받은 산출물 ✓**
평가 보고서 · DDL · 변환 SP
배포 리포트 · .NET 8 코드

**Claude Code + DBA · 개발팀 ✓**
**Kiro 대신 Claude Code 로 진행**
계열 일괄 수정 · Golden Master 검증 · 회귀 검사
**→ 사람 판단 + AI 실행 ✓**

**더한 것 — 판정 기준 = 원본과 같은 값**

GAMES ON AWS 2026

aws FOR GAMES

# 사람의 역할과 교훈 :
# 무엇을 배웠고, 무엇이 남았나

GS Neotek | MEGAZONE CLOUD

# 사람의 역할 — 작성자에서 판정자로

**평가 보고서**
자동 변환으로 분류

61%

**보정 후 최종본**
SP · 함수 353개

초안 그대로 19% | 초안 위에서 수정 81% — SP 통째 재작성 아님 · 같은 원인 59묶음

같은 353개 기준 — 자동 변환 분류 217개(61%) · 초안 그대로 67개(19%) · 초안 위 수정 286개(81%)

작성
**Transform**
DDL · SP · .NET 코드 초안

→

보정
**사람 + Claude Code**
초안을 남긴 채 같은 원인 일괄 수정

⇄

판정
**Golden Master 검증**
원본과 같은 값인지 SP 단위 비교

FAIL 이면 다시 보정 · PASS 까지 반복

# 판정 기준 — 무엇을 같다고 인정할 것인가

사람이 정해야 했던 세 가지

판정 1

## 실행 성공이면 충분한가

기준

**아니다 — 출발선일 뿐**

사본 2벌 · 롤백 루틴에서 353개 실행
오류 없이 돌아가도 값은 다를 수 있다

잡은 것

**오류로 드러난 7가지**

호출 규약 · 타입 · 예외 처리
→ 원인별 일괄 수정

→

판정 2

## 원본과 같은 값인가

기준

**결과셋 · 반환값 · 변경 결과**

같은 입력을 두 사본에 넣고 맞댄다
오류 건수까지 · 양쪽 0행은 PASS 아님

잡은 것

**오류 없이 값만 달라진 4가지**

OUTER APPLY · 0행 · GETDATE · XML
→ 계열 전체 일괄 수정

→

판정 3

## 어떤 차이를 받아들이나

기준

**받아들일 차이 vs 판단할 차이**

받아들임 — 생성 ID · 시각 · 원본 비결정
판단 — 원인 확인 후 수정 · 수용 결정

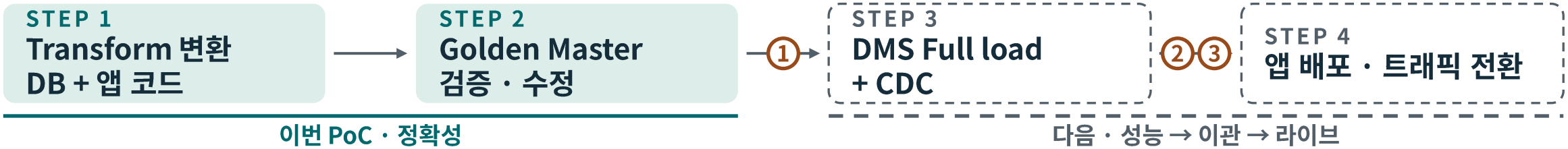
결과

**195 / 198 검증 완료 · 차이 20**

받아들임 17 · 개발팀 판단 3
검증 불가 3 은 원본 결함 — 별개

**사람이 정해야 했던 것은 변환 방법보다, 통과시킬 기준이었다**

# 운영 전환 전, 세 가지 관문




① GATE 01

## 동작을 합의했나?

- 정렬 · NULL · 동적 SQL · 트랜잭션 결정 남음

통과 기준

☐ 핵심 API 의 동작 차이 처리 기준 확정

② GATE 02

## 실제 부하를 견디나?

- 다음 단계에서 측정

통과 기준

☐ 실데이터 적재 후 목표 부하 · 동시 접속 통과

③ GATE 03

## 짧은 점검 안에 옮길 수 있나?

- 다음 단계에서 리허설

통과 기준

☐ Full load + CDC 리허설 점검 시간 · 시퀀스 · 건수 검증 되돌림 경로 확인

# 같은 전환을 준비한다면 — 세 가지 원칙

PRINCIPLE 1

## 기준을 먼저 정한다

01 PASS 기준부터

02 실제 호출 SP 목록

03 적당히 복잡하고 비중 큰 하나

PRINCIPLE 2

## 변경을 나누고 사본에서 검증한다

04 변환 가능한 앱부터

05 버전 · DB · 호출부 분리

06 쓰기는 사본에서

PRINCIPLE 3

## 통과 근거를 자산으로 남긴다

07 본문 대조 · 빈 DB 재적용

08 도구 한계 → AWS 에 재현 사례

**Transform 이 초안을 쓰고, 사람은 원본과 같은 값으로 판정한다 — 변환률보다 판정 기준을 먼저**

aws FOR GAMES

# 감사합니다

변환된 코드를 받는 것에서,

**원본과 같은 값을 확인하는 것까지.**

이병민 | 스마일게이트 DB플랫폼팀

GS Neotek | MEGAZONE CLOUD