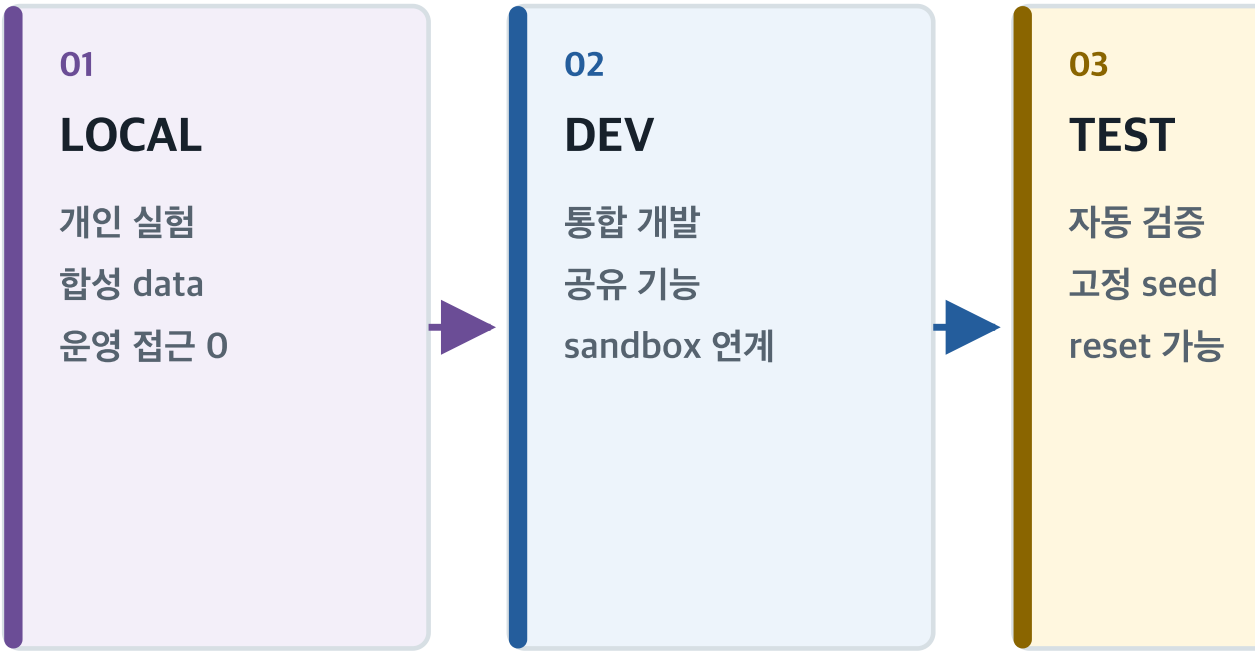
M11-01 · GIBALJA VISUAL MANUAL

개발·시험·운영 환경 구분하기

한 문장 목표: 실제 cloud·운영 data·secret·외부 network·live 배포 없이, 환경을 목적과 경계의 실행 계약으로 설계하고 동일 artifact의 검증 승격을 24개 합성 시나리오로 판정합니다.

그림 1. 오른쪽으로 갈수록 실제 사용자와 영향이 커집니다. 그래서 접근·승인·관찰·복구 evidence도 함께 강해져야 합니다.

서버 이름이 아니라 사용자·data·권한·외부 영향·승인 수준을 함께 구분



·합니다.

0. 한눈에 보는 두 번째 지도

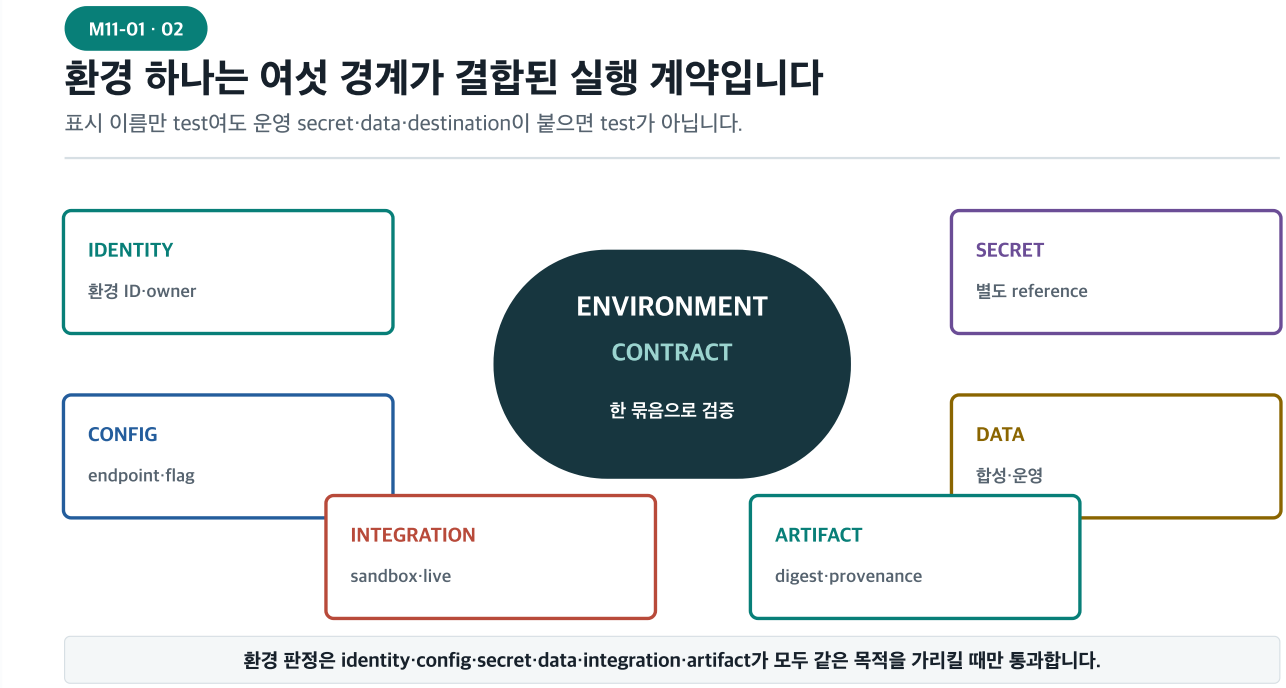


그림 2. 표시 이름 하나로 환경을 판정하지 않습니다. 여섯 경계가 같은 목적을 가리킬 때만 그 환경으로 인정합니다.

[illegible]

표시 이름만 test여도 운영 secret·data·destination이 붙으면 test가 0

IDENTITY

환경 ID·owner

CONFIG

endpoint·flag

INTEGRATION

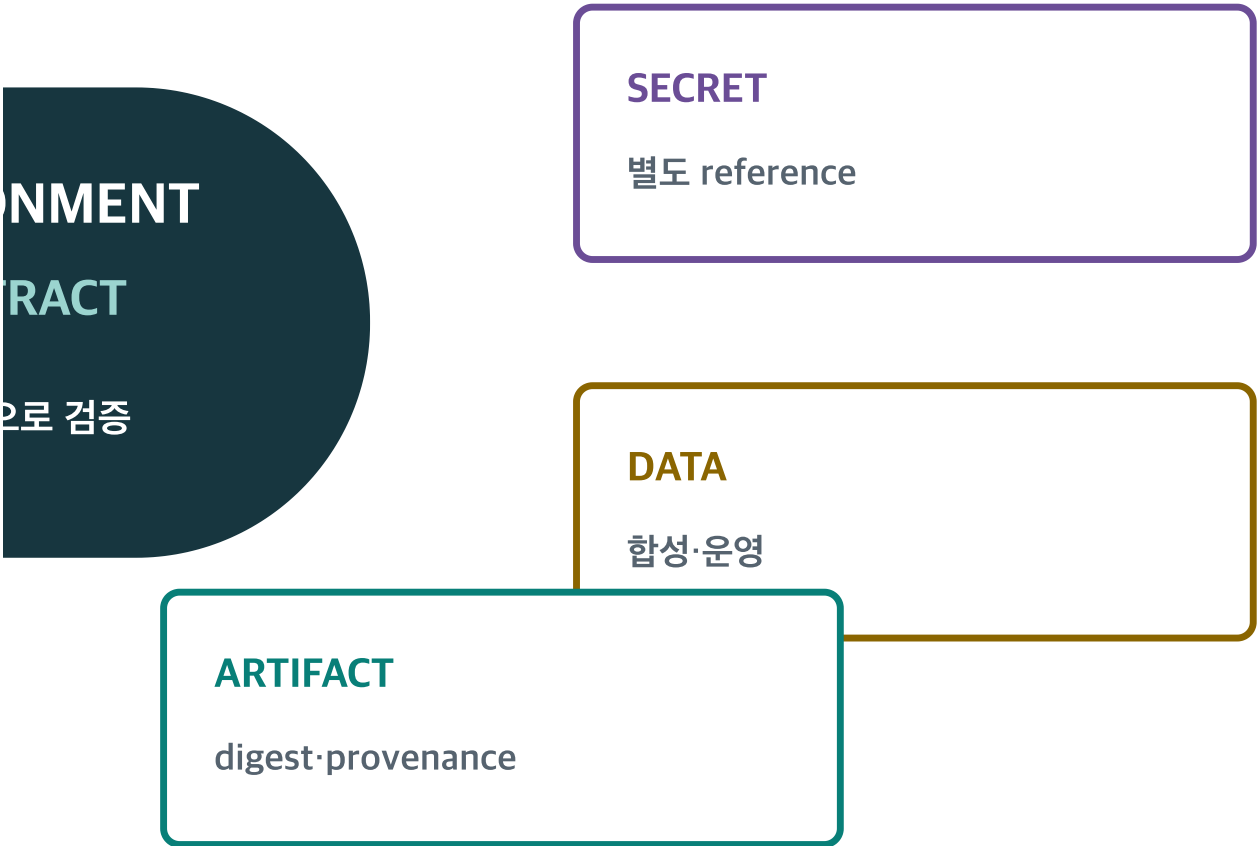
sandbox·live

ENVIRO

CONT

한 묶음으

가닙니다.


NMENT
RACT
으로 검증
SECRET
별도 reference
DATA
합성·운영
ARTIFACT
digest·provenance

난이도	그림 먼저	개념·판정	실습	셀프 테스트	최종 산출물
Level 2	35분	70분	90분	15분	Environment manifest·24개 결과·Release Gate

환경은 folder·branch·URL·namespace·화면 badge 중 하나가 아닙니다. 같은 서비스를 어떤 사용자·data·credential·destination·resource·artifact·승인 규칙으로 실행하는지 정한 계약입니다. 개발과 운영의 stack 차이는 줄이되 identity·data·secret·권한·외부 목적지는 섞지 않습니다. 이 교재는 합성 교육 검수이며 실제 cloud 구성, CI/CD 보안 감사, 침투 테스트, 개인정보 영향평가 또는 보안 인증을 대신하지 않습니다.

0.1 첫 두 장에서 기억할 열 문장

```
환경은 이름이 아니라 목적과 허용 행동의 계약이다.
Parity는 유사함이지 identity·data·credential의 공유가 아니다.
알 수 없는 환경은 production으로 추측하지 않고 시작을 멈춘다.
Environment variable은 설정 전달 수단이지 secret manager가 아니다.
비운영에는 재현 가능한 합성 data와 sandbox destination을 사용한다.
Resource 이름보다 account·network·policy boundary가 더 중요하다.
한 번 build한 exact digest를 test에서 production까지 승격한다.
Provenance는 보관만 하지 않고 signer·subject·builder·source를 검증한다.
Staging 통과는 production이 동일하거나 무사하다는 증명이 아니다.
Release Gate는 artifact·config·migration·승인·health·rollback·drift의 evidence 계약이다.
```

1. 이 PDF를 공부하는 방법

1.1 1회차 · 그림 16장만 읽기 · 35분

그림 제목과 캡션만 읽고 다음 흐름을 말해 봅니다.

```
환경 목적
→ 여섯 경계
→ manifest·identity·config
→ secret·data·destination·resource
→ artifact·provenance
→ migration·flag
→ exact approval·health·rollback
→ drift·24개 scenario·Release Gate
```

각 그림에서 “무엇을 같게 하고, 무엇을 분리하는가”를 한 문장으로 답합니다.

1.2 2회차 · 환경 경계 검수 스튜디오 · 55분

실습 생성기를 실행합니다.

```
./02_Labs/G11_Deployment_Operations/L11-01_create-environment-boundary-practice.sh
```

세 버전을 순서대로 비교합니다.

```
label-only-v1
  → separated-but-rebuilt-v2
  → verified-promotion-v3
```

이름만 분리한 기준선은 6/24만 통과합니다. 자원을 일부 분리한 중간 버전은 16/24로 올라가지만 secret·data·destination·digest·provenance·approval·drift에 8개 실패가 남습니다. 후보는 24/24와 6/6 회귀 계약을 만족해 합성 범위에서 `release_ready`가 됩니다.

1.3 3회차 · 내 서비스에 적용 · 120분

단계별 실습서를 따라 환경 구성표 및 검수 결과서를 작성합니다. 낯선 표현은 개발·시험·운영 환경 구분 용어집 300에서 현재 영역의 20개만 찾아봅니다.

1.4 손의 순서를 고정합니다

먼저 하지 않을 것	먼저 할 것
Server 이름부터 dev·prod로 짓기	사용자·목적·data·side effect·owner 먼저 정의
`.env` 파일을 환경 경계로 보기	Config schema·source·secret reference·policy 연결
운영 data를 복사해 “현실적인 시험” 만들기	필요한 분포·경계·오류를 합성 seed로 만들기
각 환경에서 artifact 다시 build	한 번 build한 digest를 검증해 승격
Provenance 파일 존재만 확인	Subject·signer·builder·source expectation 검증
Staging 통과를 운영 안전으로 단정	운영 차이·health·rollback을 별도 Gate로 작성
“배포 승인” 한 번 받기	Exact artifact·config·migration·target에 승인 결합

2. 범위와 안전 경계를 고정합니다

2.1 이번 실습의 합성 시스템

```
service: synthetic_environment_boundary_service
controls: 12
scenarios: 24
lanes:
  identity-config: 6
  data-integration: 6
  artifact-migration: 6
  promotion-operations: 6
variants:
  label-only-v1: 6/24
  separated-but-rebuilt-v2: 16/24
  verified-promotion-v3: 24/24
regression_contracts: 6/6
```

실 개인정보·실 secret·운영 resource·cloud account·외부 network·live deployment·실제 결제·메일·webhook·파괴적 command·실비용이 없습니다. Server는 `127.0.0.1`에서만 열리고 trace에는 payload 원문 대신 run ID·version·개수·판정만 남습니다.

2.2 이 매뉴얼이 주장하는 것과 주장하지 않는 것

증거로 주장할 수 있는 것	이 교재만으로 주장하지 않는 것
합성 계약에서 24개 expected가 충족됨	실제 cloud 계정에 취약점이 없음
12개 통제와 scenario가 양방향 추적됨	사용 중인 CI/CD가 공급망 공격에 안전함
실 data·secret·network·운영 접근·live 배포 0	실제 운영 장애와 data 손상이 발생하지 않음
세 version의 알려진 실패가 재현됨	특정 cloud·CI provider가 안전함
특정 범위의 release gate가 재현 가능함	침투 테스트·보안 인증·규제 심사를 통과함

2.3 앞뒤 매뉴얼과의 경계

연결	가져오는 것	이번 매뉴얼이 더하는 것
M08-03 환경 설정	환경별 configuration과 secret 분리	Identity·data·destination·artifact·승격까지 하나의 계약으로 확장

연결	가져오는 것	이번 매뉴얼이 더하는 것
M10-01·02 시험	Acceptance·경계·회귀 case	환경 crossing·promotion drift의 24개 적대 scenario
M10-03 개인정보·secret	수집 최소화·secret lifecycle	환경별 reference·service identity·운영 접근 0 적용
M11-02 cloud 배포	실제 domain·HTTPS·provider 배포	이번에는 배포 전 환경 경계와 release evidence만 고정
M11-03 운영	Monitoring·backup·incident	이번에는 환경 label·health·rollback·drift handoff까지만 작성

2.4 안전한 교육 문장

```
이번 결과는 합성 환경 계약의 개발 검수 evidence입니다.
실제 cloud 계정·network·data·credential·pipeline을 직접 검사하지 않았습니다.
실제 배포 전에는 provider 최신 문서, 조직 policy, security·privacy·operations 검토를 추가합니다.
```

3. 환경을 이름에서 실행 계약으로 바꿉니다

3.1 Environment가 아닌 것

흔한 대체물	왜 충분하지 않은가
Git branch	Source 흐름일 뿐 runtime identity·data·권한을 강제하지 않음
Folder	File 배치일 뿐 외부 destination과 resource policy를 막지 않음
URL	접속 주소일 뿐 연결된 database·secret·artifact를 증명하지 않음
Namespace	논리 이름일 수 있으나 account·network·permission 분리가 아닐 수 있음
Feature flag	기능 분기이며 authorization·data boundary가 아님
화면 badge	사람이 보는 표시이며 server-side enforcement가 아님

3.2 여섯 경계의 결합

```
environment contract
  = identity
  + config·secret reference
  + data origin·class
  + external destination·side effect
  + resource·network policy
  + artifact·release evidence
```

한 요소라도 다른 목적을 가리키면 환경 판정은 실패합니다. Test badge가 붙었어도 production secret, 운영 record, live payment endpoint 중 하나가 연결되면 그 실행은 안전한 test가 아닙니다.

3.3 Parity와 isolation을 동시에 지킵니다

같게 또는 유사하게	반드시 분리할 것
Runtime·database engine의 큰 version	Service identity·credential
Build·deployment 절차	Data origin·retention
Config schema와 validation code	Secret reference와 값
Network topology의 의도	Account·project·resource policy
Health·rollback mechanism	External provider account·destination
Artifact bytes	Release-time config와 approval

`Dev/prod parity`는 환경 차이 때문에 생기는 놀라움을 줄이는 원칙입니다. 그러나 운영 credential과 data를 개발에 복제하라는 뜻은 아닙니다. 비슷한 stack에서 분리된 identity와 합성 data를 사용합니다.

3.4 가장 중요한 판정 질문

```
이 process는 자신이 어느 환경인지 기계적으로 아는가?
그 identity가 허용된 config·secret·data·destination만 얻는가?
지금 실행되는 bytes는 시험한 exact digest인가?
이 release와 target을 정확히 승인했는가?
실패 시 traffic을 멈추고 이전 상태로 돌아갈 수 있는가?
```

4. Environment manifest로 목적을 비교합니다

M11-01 · 03

Environment manifest 한 장으로 목적과 경계를 비교합니다

각 환경의 사용자·data·destination·승인 owner를 같은 열로 놓으면 혼선이 보입니다.

환경	목적·사용자	DATA	외부 연계	승인
LOCAL	개인 개발	synthetic	stub	없음
TEST	자동 시험	fixed seed	sandbox	pipeline
STAGE	출시 예행	synthetic	sandbox	release owner
PROD	실사용	production	live allowlist	exact release

Manifest에 없는 환경·resource·destination은 암묵적으로 허용하지 않고 default deny합니다.

그림 3. 같은 열로 비교하면 운영 data가 test에 있거나 승인 owner가 빈 환경처럼 숨은 혼선이 드러납니다.

Environment manifest 한 장으로 구

각 환경의 사용자·data·destination·승인 owner를 같은 열로 놓으면 혼

환경	목적·사용자	DAT
LOCAL	개인 개발	synth
TEST	자동 시험	fixed s
STAGE	출시 예행	synth
PROD	실사용	produc

[illegible]

:선이 보입니다.

'A	외부 연계	승인
etic	stub	없음
seed	sandbox	pipeline
etic	sandbox	release owner
:tion	live allowlist	exact release

4.1 다섯 환경의 기본 목적

환경	주 사용자	주 목적	기본 data	외부 연계	실제 영향
Local	개발자 개인	빠른 구현·debug	합성 최소	stub·local	0
Development	개발 team	변경 통합·기능 확인	합성	sandbox	0
Test·QA	자동 pipeline·QA	Expected와 actual 반복 검증	고정 fixture	sandbox	0
Staging	Release team	정확한 후보와 절차 예행	합성·승인 예외	sandbox	0
Production	실제 사용자·운영자	실제 업무 제공	운영	live allowlist	실제

이 표는 기본값입니다. 조직이 QA와 staging을 합칠 수는 있지만, 합친 목적·data·권한·승인 충돌을 명시해야 합니다. 환경 개수보다 경계의 설명 가능성이 중요합니다.

4.2 각 행에 반드시 넣을 것

Field	좋은 값	위험한 값
Purpose	“정확한 release 후보의 배포 절차 예행”	“운영 전 확인”
User	QA·release owner·자동 job	“team”
Data	synthetic seed v3, production origin 0	“test data”
Integration	provider sandbox account A	“API 연결”
Allowed action	read·reset·sandbox send	“필요한 작업”
Owner	이름·team·on-call 경로	공란
Lifecycle	생성·reset·expiry·폐기	계속 사용

4.3 Manifest에 없으면 허용하지 않습니다

새 resource나 destination이 생겼는데 manifest와 policy에 없다면 default deny가 기본입니다. 운영 중 임시 연결이 필요하면 목적·scope·owner·expiry·audit를 가진 예외로 다룹니다.

4.4 Ephemeral과 persistent를 선택합니다

유형	장점	위험	필수 통제
Ephemeral	깨끗한 상태·branch별 격리·자동 폐기	생성 비용·seed 준비	TTL·cleanup·quota·no prod path
Persistent	안정된 공동 검증·긴 통합 흐름	state 누적·data 잔류·drift	Reset·inventory·owner·access review

5. Environment identity와 config를 시작 전에 검증합니다

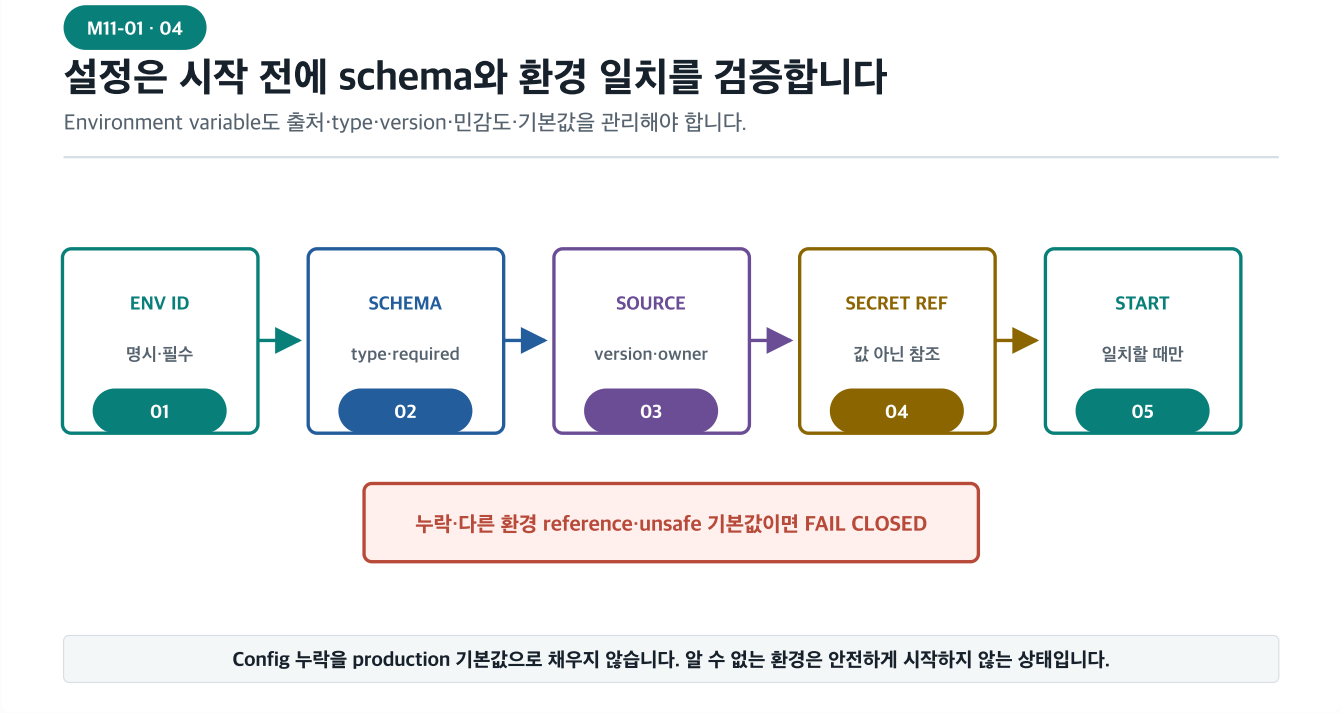
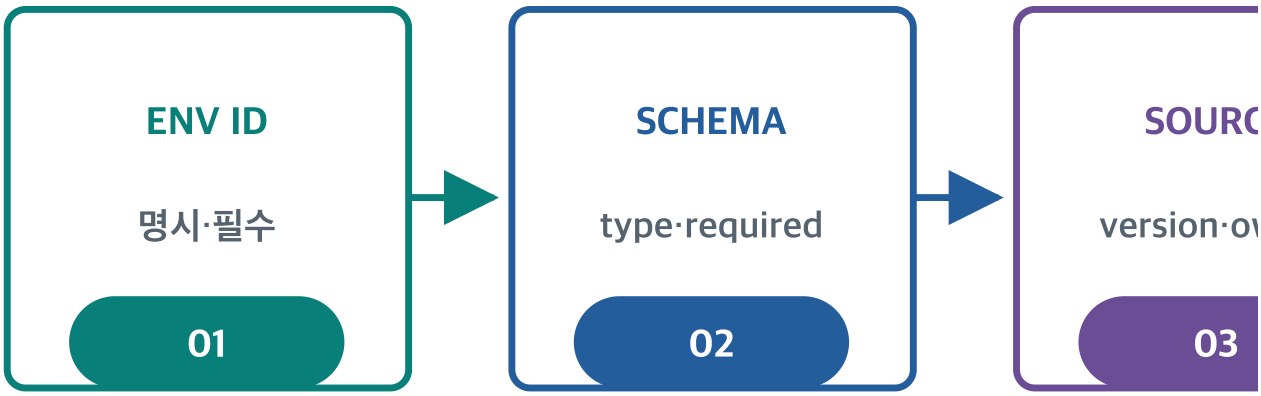


그림 4. 누락·unknown·다른 환경 reference를 production 기본값으로 채우지 않습니다. 실행 전에 안전하게 닫습니다.

[illegible]

Environment variable도 출처·type·version·민감도·기본값을 관리해0



누락·다른 환경 reference·uns

합니다.

afe 기본값이면 FAIL CLOSED

5.1 Canonical environment ID

사람이 보는 `TEST`, resource tag의 `testing`, runtime의 `qa`가 제각각이면 policy와 event를 연결하기 어렵습니다. 하나의 canonical enum을 정하고 다른 표시는 그 값에서 파생합니다.

```
APP_ENVIRONMENT = local | dev | test | stage | prod
```

검증 순서:

```
필수 값 존재
→ 허용 enum
→ deployment target과 일치
→ resource binding과 일치
→ config source와 일치
→ telemetry label 생성
→ start 또는 fail closed
```

5.2 Config의 최소 metadata

Field	질문
Key·type	이름과 자료형이 무엇인가
Required	없으면 시작할 수 있는가
Environment	어느 환경에서 허용되는가
Source	값은 어디서 오는가
Version	어떤 묶음으로 배포됐는가
Sensitive	Secret manager로 보내야 하는가
Default	실패 때 가장 제한적인 값인가
Owner	변경을 설명할 사람은 누구인가

5.3 Environment variable의 정확한 의미

Environment variable은 설정을 process에 전달하는 좋은 방법일 수 있습니다. 그러나 다음을 자동으로 제공하지는 않습니다.

- 저장 시 암호화
- 읽기 권한 최소화
- Version·rotation·revocation
- Shell history·process dump·log 노출 방지
- 다른 환경 reference 차단

Secret 원문 대신 secret manager reference를 전달하고, workload identity가 필요한 시점에 제한된 값을 얻도록 설계합니다.

5.4 Safe default와 fail closed

상황	위험한 동작	안전한 동작
`APP_ENVIRONMENT` 누락	`prod`로 default	Start 중단
Payment mode 누락	`live` 선택	`sandbox` 또는 중단
Debug 값 해석 실패	Debug ON	False·중단
Database endpoint 없음	공용 기본 DB	연결하지 않음
Secret reference 불일치	비슷한 이름 선택	Resolve 거절

5.5 Build-time과 release-time을 분리합니다

Artifact 안에 environment endpoint와 secret을 굽으면 환경마다 rebuild하게 됩니다. Code와 dependency는 build artifact에, 환경 차이는 검증된 release-time config에 둡니다.

```
build = source revision + dependency → artifact digest
release = artifact digest + config version + secret references
run = release bundle을 target environment에서 실행
```

6. Secret과 service identity를 환경별로 분리합니다

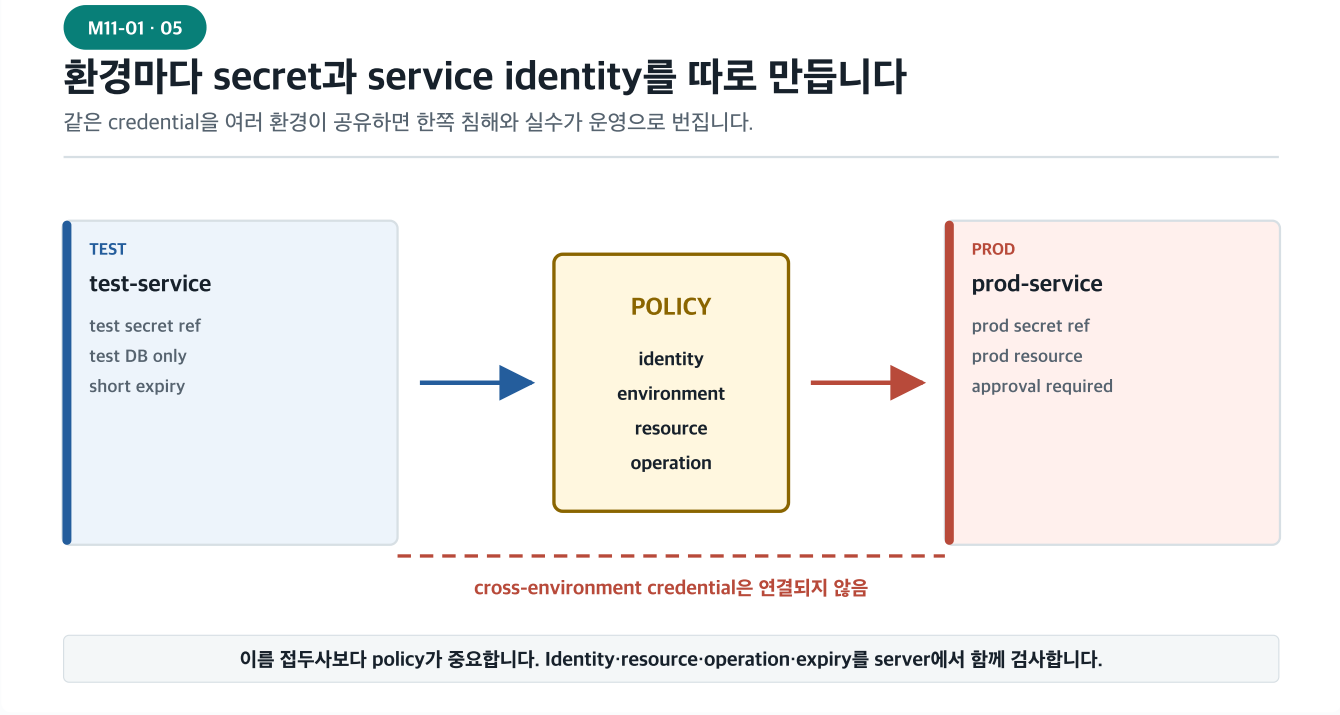
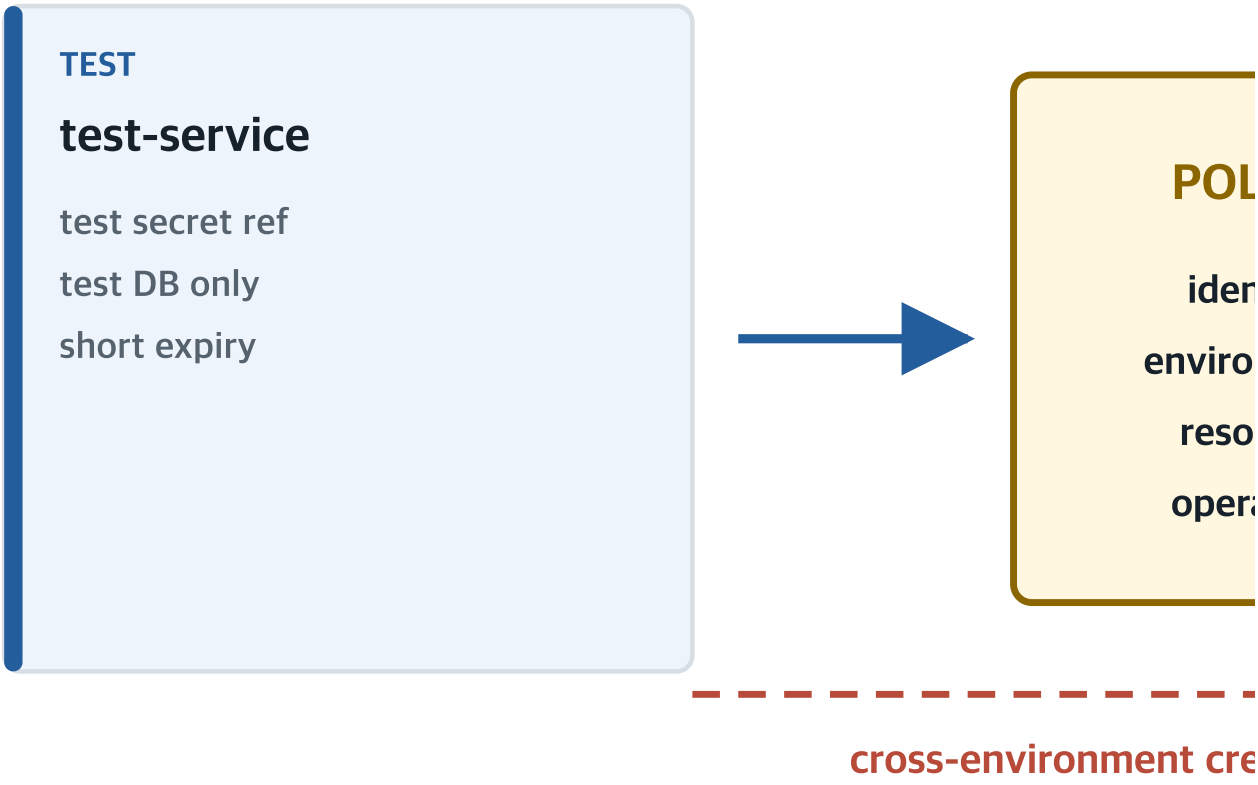
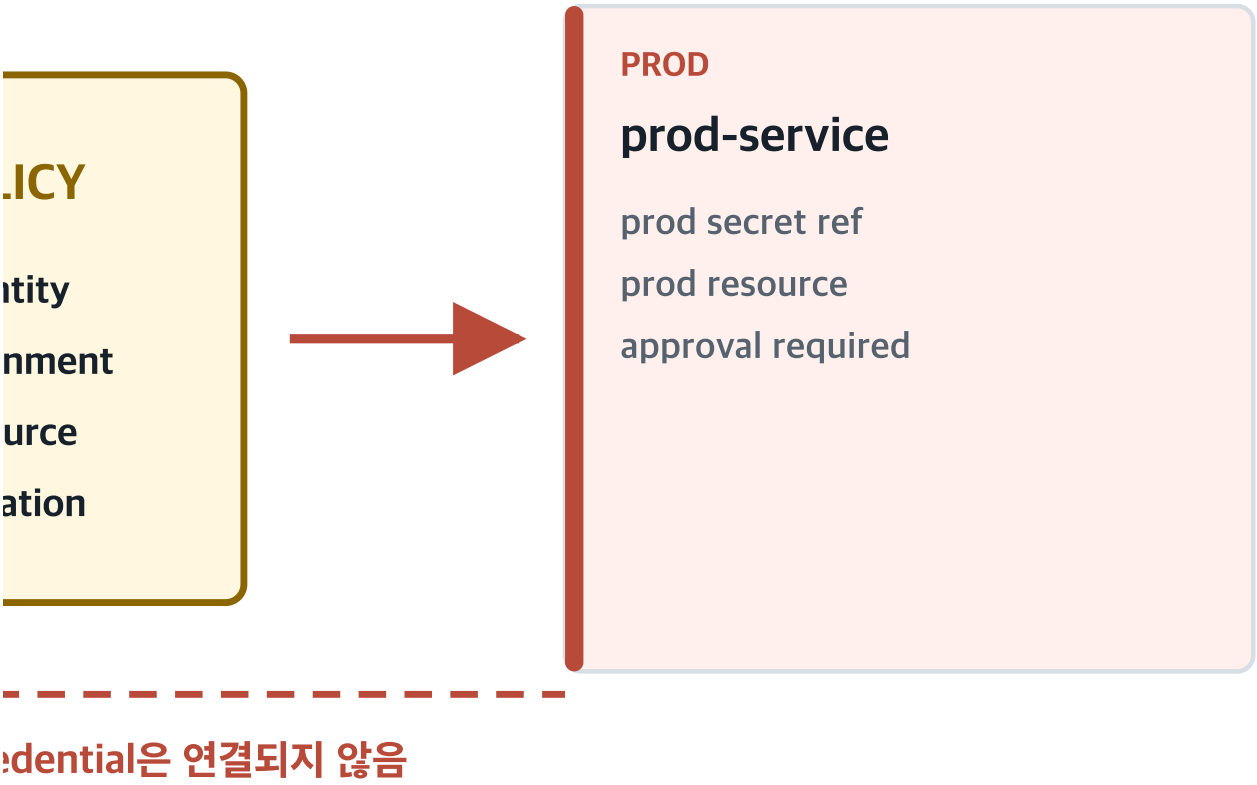


그림 5. 접두사보다 policy가 중요합니다. Identity·environment·resource·operation·expiry를 server에서 함께 검사합니다.

같은 credential을 여러 환경이 공유하면 한쪽 침해와 실수가 운영으로



번집니다.


LICY
ıtity
ınment
urce
ation
PROD
prod-service
prod secret ref
prod resource
approval required
edential은 연결되지 않음

6.1 하나의 credential을 공유하지 않습니다

공유할 때 생기는 문제	분리하면 얻는 것
Test 침해가 production 접근으로 확장	Blast radius가 해당 환경에 머묾
누가 어떤 환경에서 사용했는지 모호	Identity와 event를 environment에 연결
회전 시 모든 환경이 동시에 중단	환경별 교체·검증 가능
개발자 local에 운영 secret 복사	Workload identity와 짧은 token 사용

6.2 Reference와 원문을 구분합니다

```
good config:
  database_secret_ref = secret://test/app-db/v7

avoid:
  database_password = actual-secret-value
```

Reference에도 환경·owner·version·expiry를 둡니다. 이름이 `test`여도 resolver policy가 production secret을 반환하면 통제는 실패입니다.

6.3 권한 판정의 다섯 축

```
ALLOW = identity valid
    AND identity.environment == resource.environment
    AND operation in allowed_operations
    AND token.audience == target_service
    AND now < credential.expiry
```

6.4 사람과 workload를 분리합니다

개발자 개인 권한으로 pipeline과 production deploy를 실행하지 않습니다. Source review, artifact publish, production approval, production execution의 identity와 책임을 구분합니다.

주체	기본 권한
Developer	Source·dev resource, production deploy 없음

주체	기본 권한
CI builder	Source read·artifact write, production access 없음
Test runner	Test resource only
Release approver	Exact release 승인, runtime credential 없음
Production deployer	승인된 release 실행만

7. 비운영에는 합성 data를 사용합니다

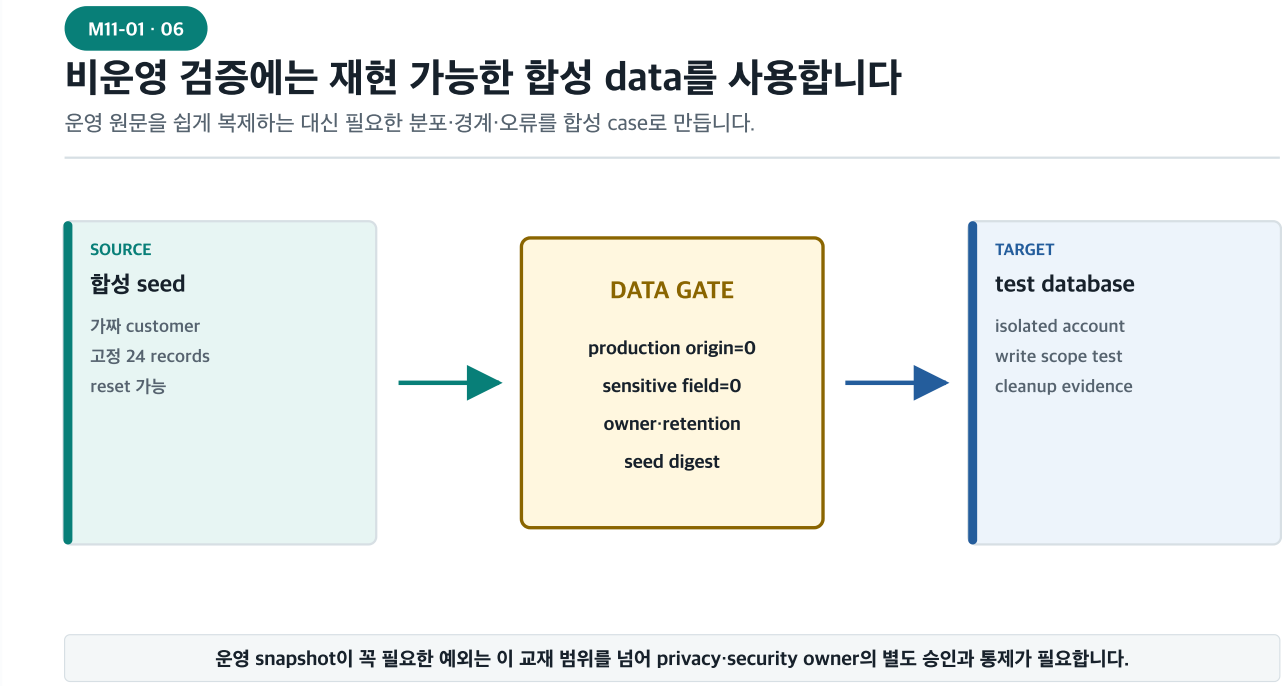


그림 6. 필요한 현실성은 실제 사람의 record가 아니라 분포·경계값·오류·관계의 합성 case로 재현합니다.

운영 원문을 쉽게 복제하는 대신 필요한 분포·경계·오류를 합성 case로

SOURCE

합성 seed

가짜 customer

고정 24 records

reset 가능

DATA

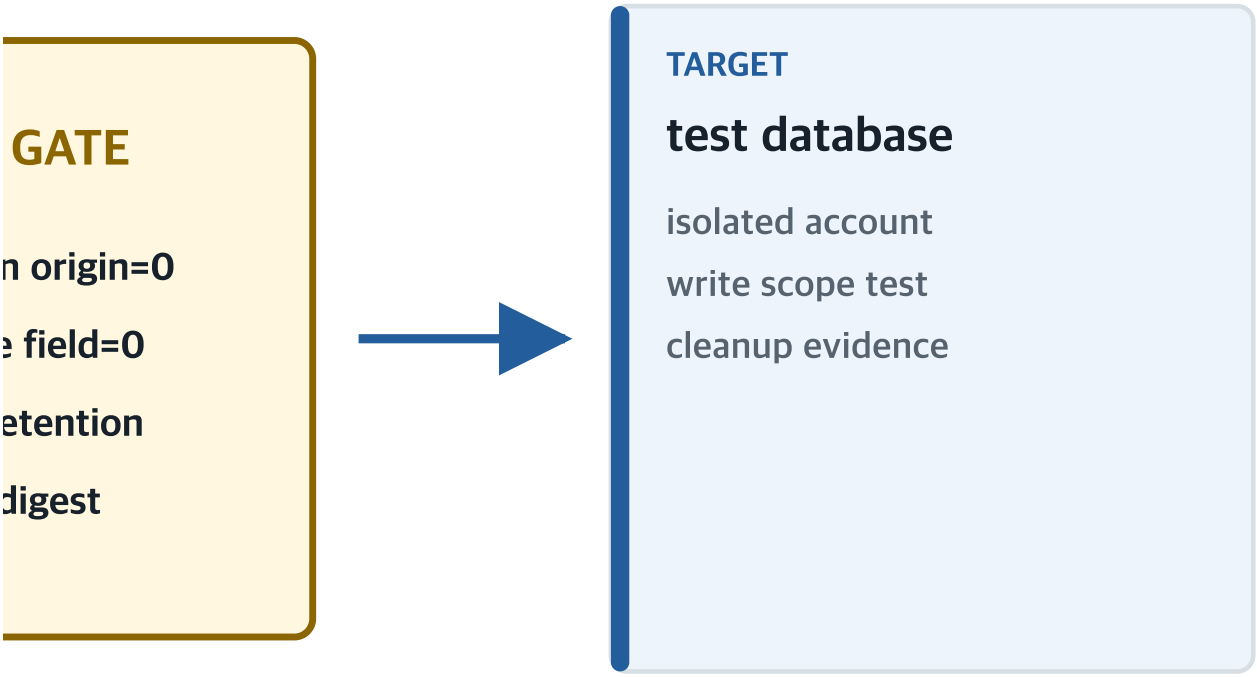
productio

sensitive

owner·r

seed

만듭니다.


GATE
n origin=0
e field=0
etention
digest
TARGET
test database
isolated account
write scope test
cleanup evidence

7.1 “운영과 비슷함”을 다시 정의합니다

운영 record를 복사해야 현실적인 시험이 되는 것은 아닙니다. 우리가 재현하려는 것은 다음 속성입니다.

- 빈 값·최대 길이·다국어·시간대
- 관계 record·중복·삭제·권한 차이
- 정상·지연·실패·재시도 상태
- 작은·중간·큰 volume 분포
- Migration 전·후 schema 호환

이 속성을 고정 seed와 fixture로 만들면 더 쉽게 reset·version·회귀할 수 있습니다.

7.2 Data gate의 최소 질문

질문	Candidate 기준
Source가 운영인가	아니오
실제 사람과 다시 연결 가능한가	아니오
목적에 필요 없는 field가 있는가	0
Seed version과 digest가 있는가	예
Reset·cleanup evidence가 있는가	예
Test identity가 production에 접근하는가	아니오

7.3 운영 snapshot 예외는 별도 위험 결정입니다

합성으로 재현하기 어려운 제한된 사례가 있을 수 있습니다. 그렇더라도 “마스킹했으니 괜찮다”로 끝내지 않습니다. 목적·대안 실패 이유·최소 field·재식별 가능성·접근·보존·삭제·승인·만료를 privacy·security owner가 별도 검토합니다. 이 매뉴얼의 기본 실습 범위를 넘어섭니다.

7.4 SC-07~SC-09

Scenario	Expected	실패 징후
SC-07 운영 원문 복제	`production_records=0`	Snapshot copy 시작
SC-08 합성 seed	`record_count=24`, reset 가능	수동 누적 state

Scenario	Expected	실패 징후
SC-09 Test→prod write	`authorized=false`, write 0	운영 record 변경

8. 외부 연계는 sandbox와 allowlist를 거칩니다

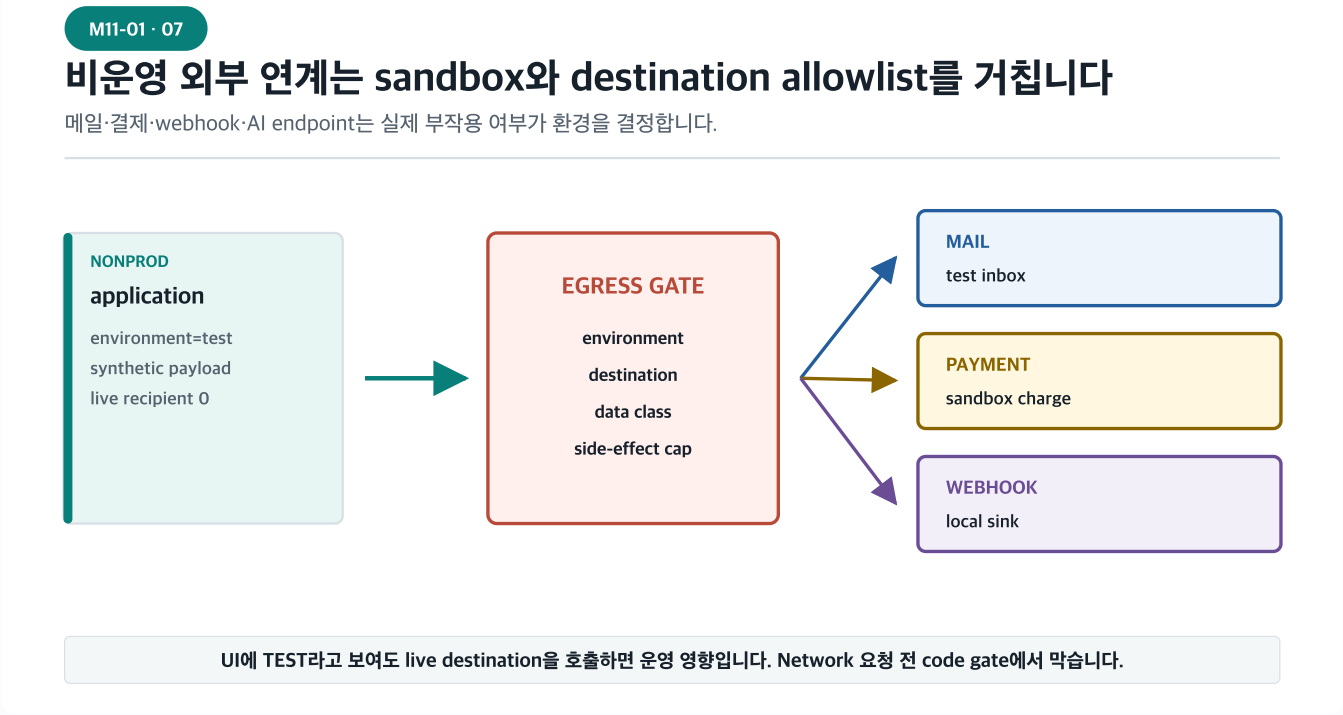


그림 7. UI의 TEST 표시는 network 요청을 막지 못합니다. 요청 전에 environment·destination·data class·side-effect cap을 code로 검사합니다.

[illegible] sandbox와 dest[illegible]

메일·결제·webhook·AI endpoint는 실제 부작용 여부가 환경을 결정합

NONPROD

application

environment=test

synthetic payload

live recipient 0

EGRESS GA[illegible]

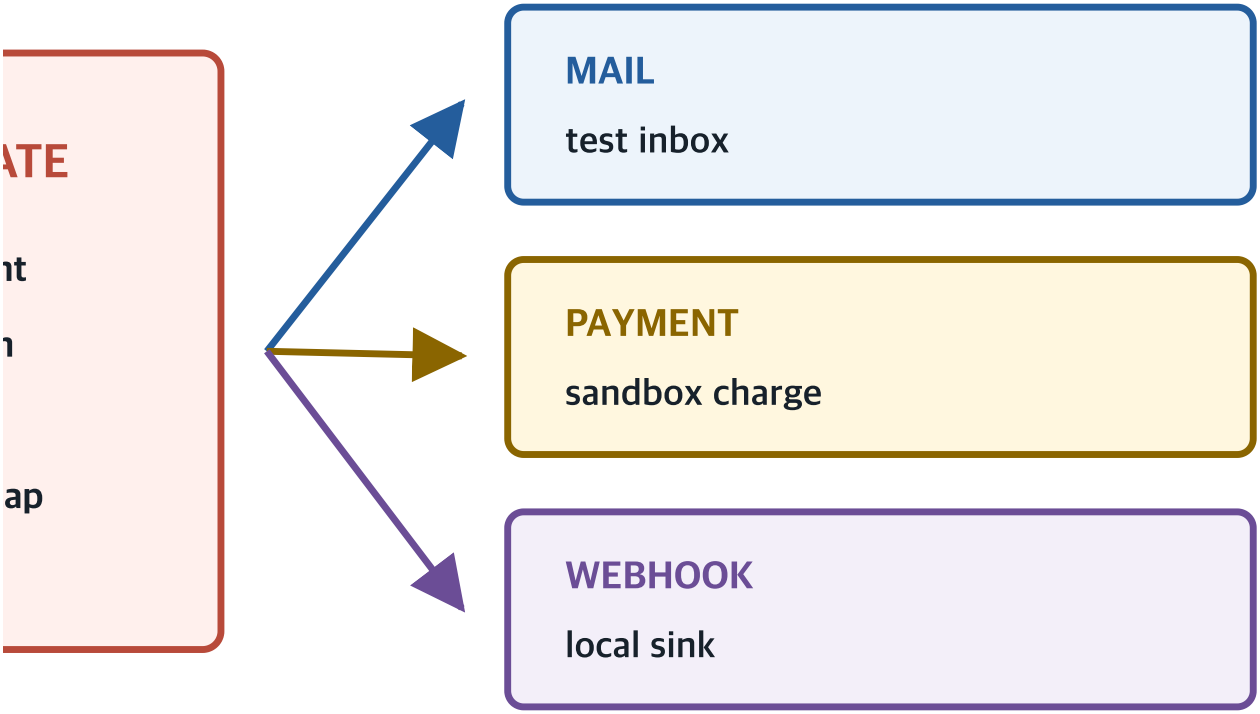
environmer[illegible]

destinatior[illegible]

data class[illegible]

side-effect c[illegible]

니다.


ATE
nt
n
ap
MAIL
test inbox
PAYMENT
sandbox charge
WEBHOOK
local sink

8.1 외부 영향이 환경을 결정합니다

연계	비운영 destination	운영 destination	비운영 실수의 영향
Mail	Test inbox	실제 고객 주소	오발송·정보 유출
Payment	Sandbox merchant	Live merchant	실제 승인·취소·비용
Webhook	Local sink	Partner production	외부 업무 변경
AI API	Nonprod project·합성 input	Prod project·업무 input	Data·비용·quota 혼선
Object storage	Test bucket	Prod bucket	덮어쓰기·삭제·노출

8.2 Data와 destination을 함께 검사합니다

```
ALLOW_EGRESS = environment matches
             AND provider_account matches
             AND destination in allowlist
             AND data_class allowed
             AND side_effect <= cap
             AND idempotency valid
```

합성 payload를 live recipient에게 보내도 부작용이고, 운영 원문을 sandbox에 보내도 data boundary 위반입니다.

8.3 UI 경고보다 code gate가 먼저입니다

빨간 banner와 확인 dialog는 유용하지만 API client·batch job·재시도 worker를 막지 못할 수 있습니다. External request가 나가기 직전 server-side gateway나 SDK wrapper에서 destination을 검증합니다.

8.4 Provider account를 분리합니다

가능하면 dev·test·stage·prod가 provider의 별도 project 또는 account를 사용합니다. Provider가 완전한 분리를 지원하지 않으면 credential·recipient·quota·webhook secret·audit를 환경별로 나누고 잔여 위험을 기록합니다.

9. Account·network·resource를 격리합니다

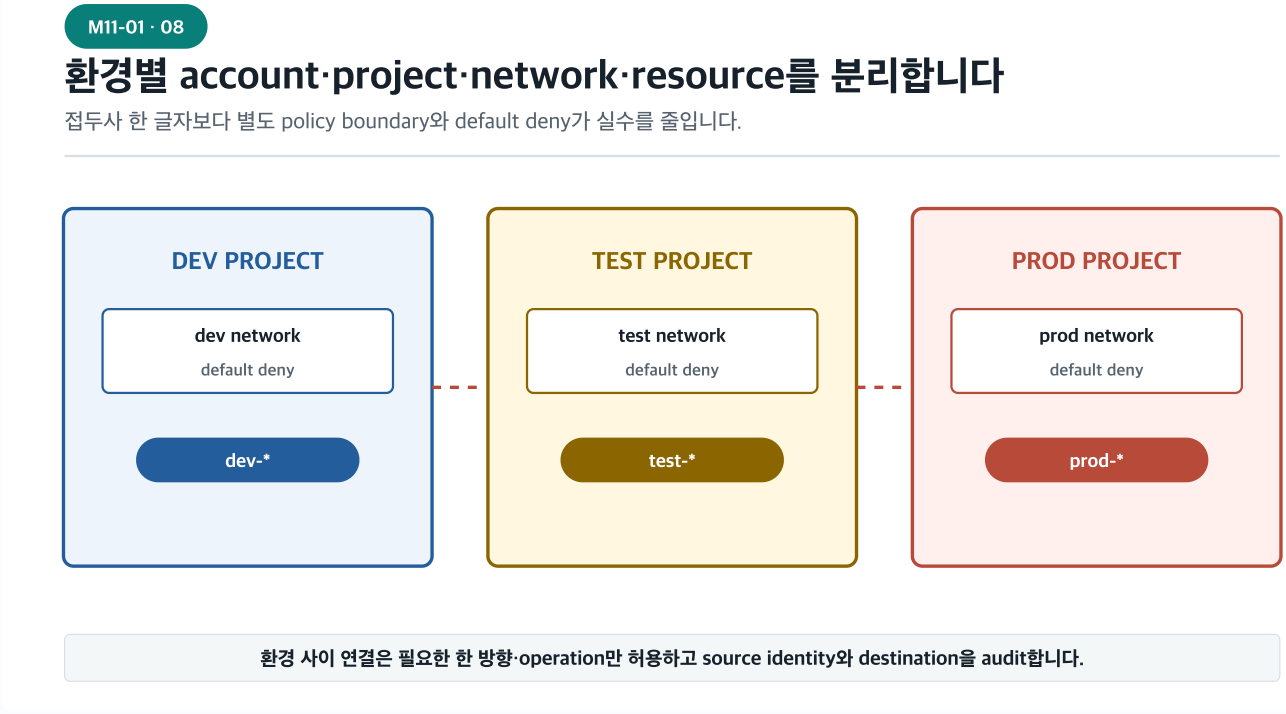
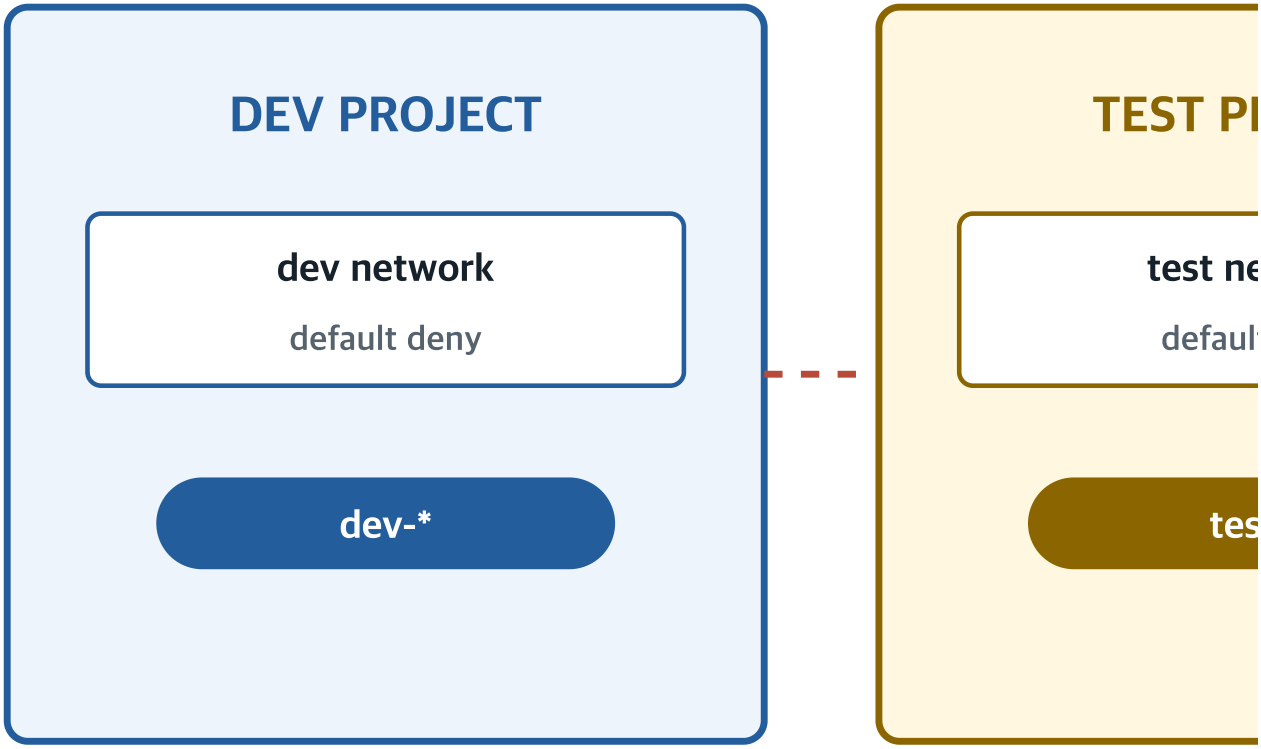
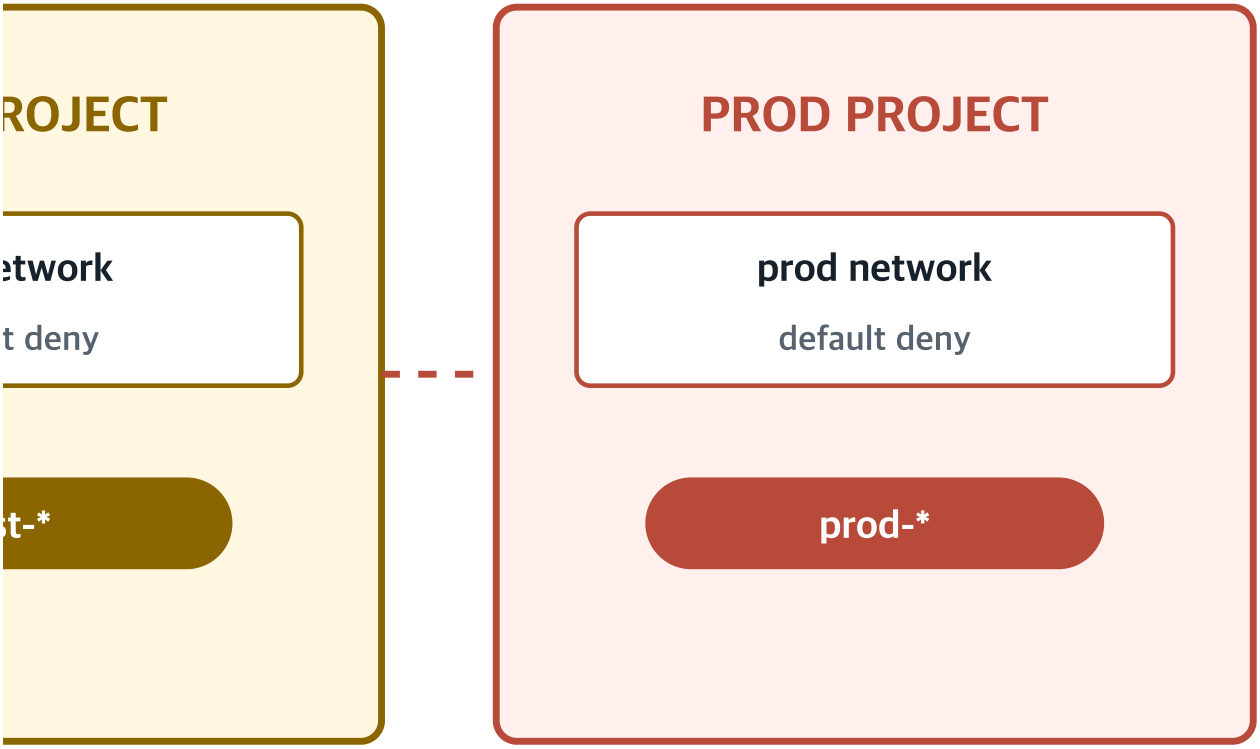


그림 8. 이름의 `prod-` 접두사는 안내입니다. 실제 경계는 account·identity policy·route·firewall·resource permission입니다.

접두사 한 글자보다 별도 policy boundary와 default deny가 실수를 줄



입니다.


ROJECT
etwork
t deny
st-*
PROD PROJECT
prod network
default deny
prod-*

9.1 이름과 경계를 구분합니다

장치	주 역할	보안 경계가 되려면
Resource name	사람이 빠르게 구분	Policy가 name·tag를 검증해야 함
Tag·label	Inventory·cost·automation	변경 권한과 required tag 검사 필요
Namespace	논리 분류	Network·RBAC·quota policy 결합 필요
Project·account	큰 관리 단위	Cross-account role과 billing·identity 제한 필요
Network	연결 경로 통제	Default deny·egress·audit 필요

9.2 환경 사이 연결을 예외로 봅니다

환경 사이 통신이 필요하면 전체 network를 열지 않습니다.

```
source identity
→ source environment
→ exact destination resource
→ one operation·protocol
→ expiry
→ audit
```

예를 들어 staging이 production database를 읽는 연결은 기본적으로 두지 않습니다. 정말 필요한 운영 검증은 별도 read replica·synthetic mirror·승인된 query workflow처럼 피해 범위를 줄인 대안을 설계합니다.

9.3 IaC로 desired state를 선언합니다

Console에서 직접 만든 resource는 빠르지만 review·재현·drift 탐지가 어렵습니다.
Account·network·role·database·queue·provider destination의 목표 상태를 code로 선언하고 실제 상태와 비교합니다.

9.4 Blast radius를 질문합니다

```
Test credential 하나가 노출되면 어떤 production resource까지 갈 수 있는가?
Test network가 오염되면 어느 account와 destination까지 연결되는가?
잘못된 delete command가 어느 환경의 resource를 지울 수 있는가?
```

Candidate 답은 “production 0”에 가까워야 합니다.

10. 한 번 build한 digest를 승격합니다

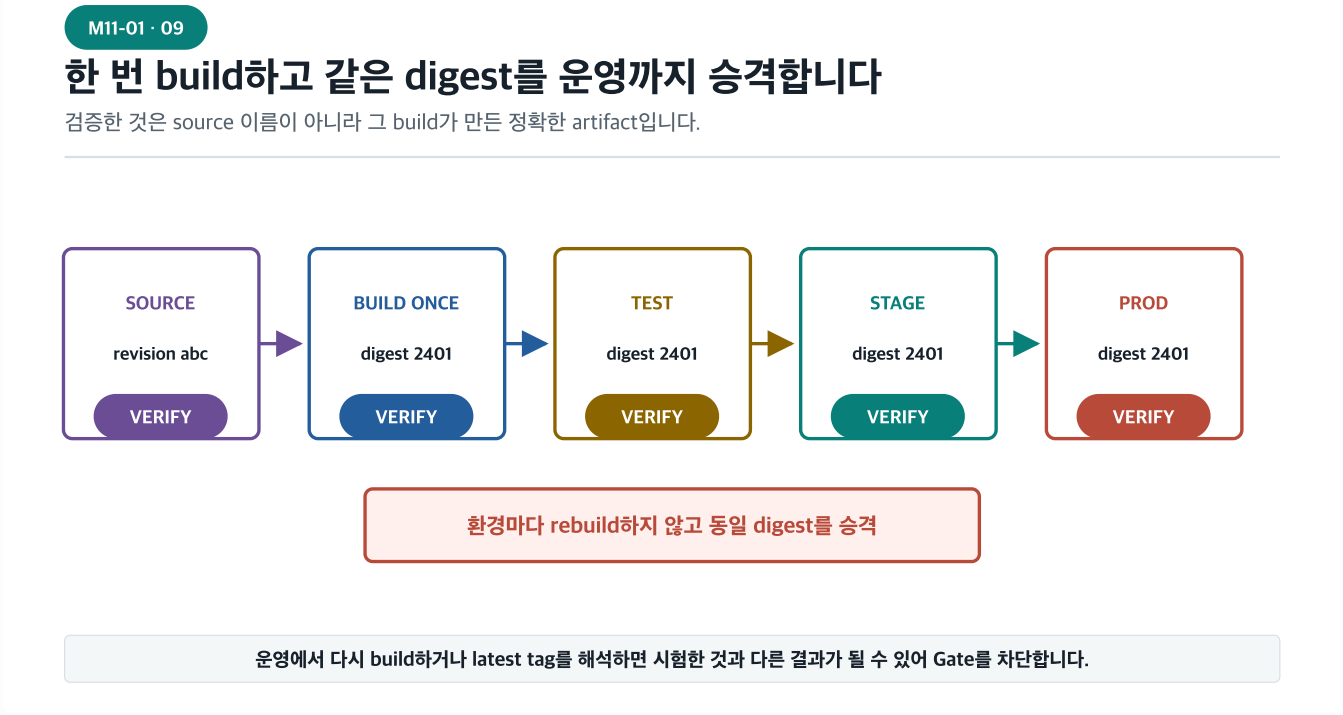
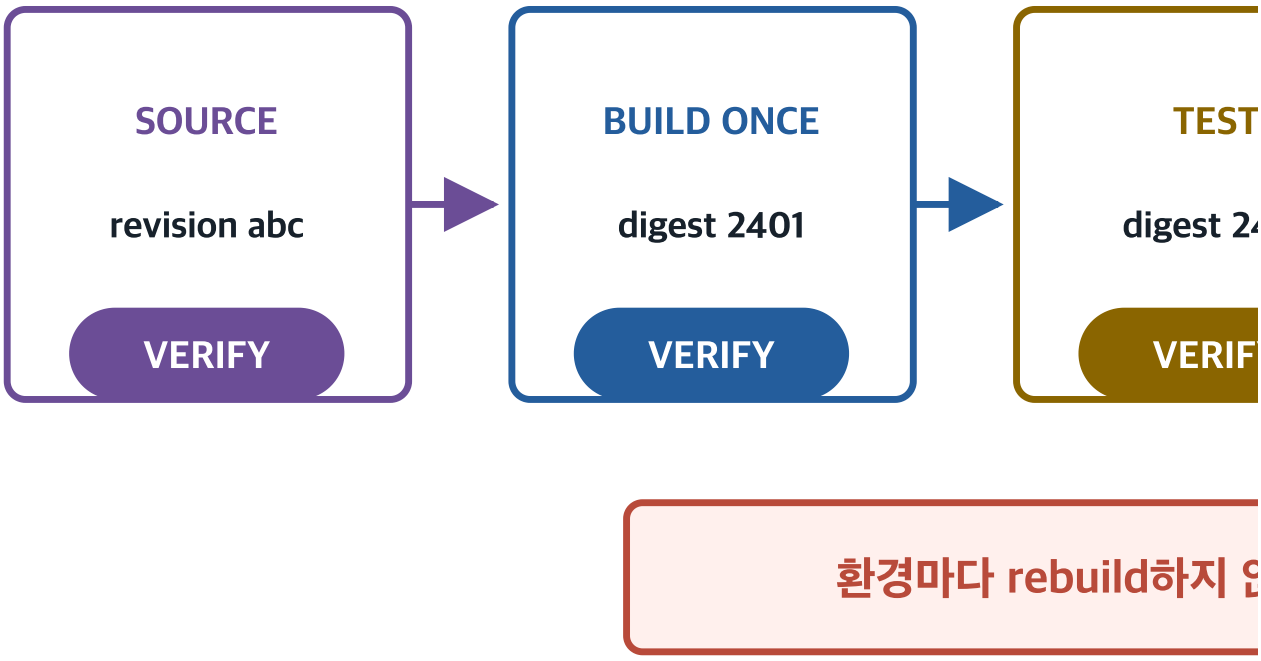


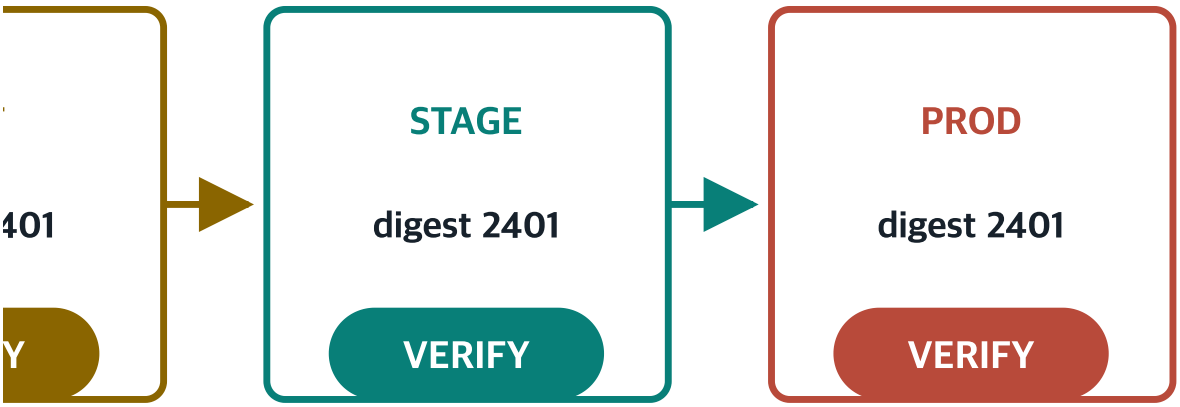
그림 9. 시험한 것은 branch 이름이 아니라 exact bytes입니다. 환경마다 rebuild하지 않고 검증한 digest를 그대로 옮깁니다.

같은 build에는 같은 digest를 보장해

검증한 것은 source 이름이 아니라 그 build가 만든 정확한 artifact입니



다.


401
Y
STAGE
digest 2401
VERIFY
PROD
digest 2401
VERIFY


고 동일 digest를 승격

10.1 환경마다 build하면 무엇이 달라지는가

같은 source revision이라도 build 시점과 환경이 달라지면 dependency resolution·base image·compiler·network response·timestamp·runner state가 달라질 수 있습니다.

```
test build A: sha256:111...
prod build B: sha256:999...
```

Test가 A를 통과해도 production은 B를 실행합니다. 이때 “같은 code”라는 말은 artifact 동일성을 증명하지 못합니다.

10.2 Build once·promote의 흐름

```
canonical source revision
→ isolated build
→ immutable artifact digest
→ automated test
→ staging validation
→ exact release approval
→ production deployment of same digest
```

환경별 차이는 artifact 밖의 검증된 config reference로 결합합니다.

10.3 Tag와 digest를 구분합니다

식별자	장점	위험	Gate 사용
`v1.4.0` tag	사람이 이해하기 쉬움	덮어쓸 수 있는 registry도 있음	보조
`latest` tag	최신 pointer	시간이 지나며 다른 bytes	금지
Digest	내용이 같으면 같음	사람이 읽기 어려움	필수
Release ID	Artifact·config·migration 묶음	Digest를 포함하지 않으면 모호	필수 묶음

10.4 운영에서 rebuild를 막습니다

Deployment job은 source를 checkout하거나 compiler를 실행하지 않습니다. 승인된 registry에서 pinned digest를 pull하고, signature·provenance·target environment를 검증한 뒤 deploy만 수행합니다.

10.5 SC-13·SC-14

Scenario	Expected	Block 대상
SC-13 Test→stage	source digest = target digest	Rebuild·bytes 변경
SC-14 Production	`rebuilt=false` , `mutable_tag_used=false`	`latest` 해석·prod build

11. Provenance를 기대값에 맞춰 검증합니다

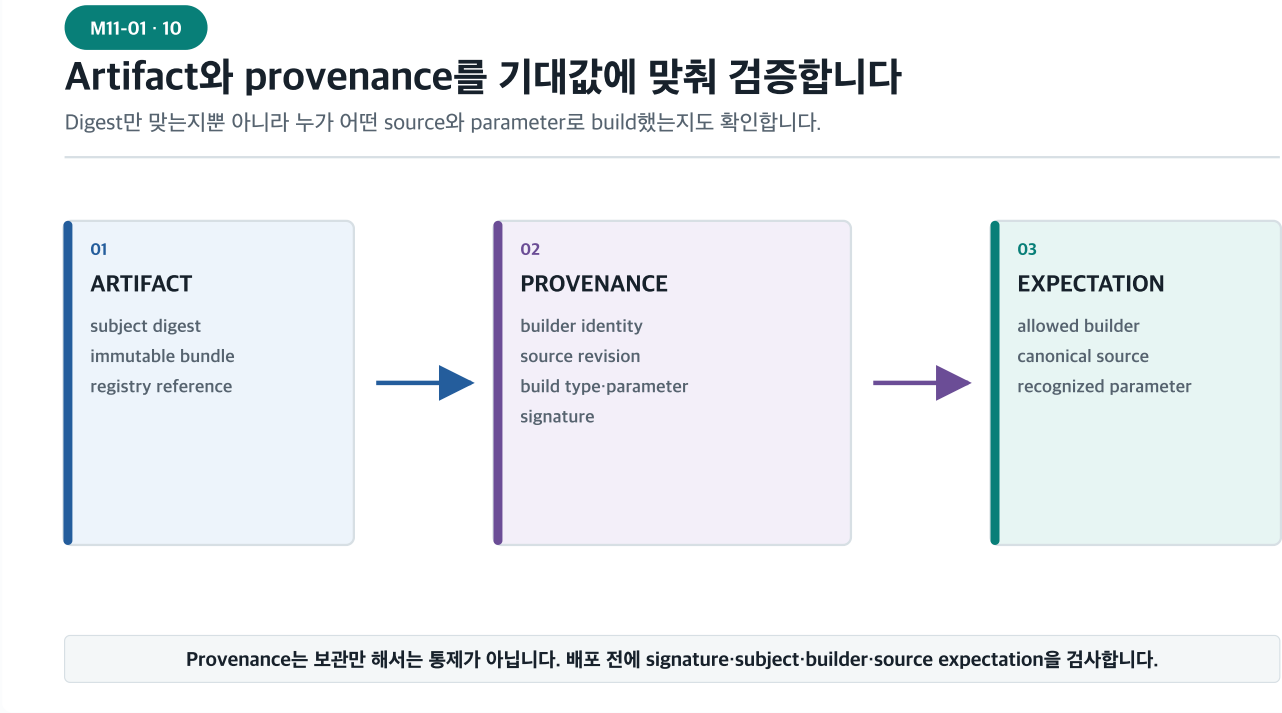


그림 10. Provenance가 있다는 사실보다 지금 배포할 artifact와 허용 builder·source를 정확히 설명하는지가 중요합니다.

Artifact와 provenance를 함께 검증해야

Digest만 맞는지뿐 아니라 누가 어떤 source와 parameter로 build했는

01

ARTIFACT

subject digest

immutable bundle

registry reference

02

PROVENANCE

builder identity

source revision

build type·paramet

signature

지도 확인합니다.

er

03

EXPECTATION

allowed builder

canonical source

recognized parameter

11.1 Digest와 provenance의 역할

Evidence	답하는 질문
Digest	Bytes가 같은가
Signature	허용된 signer가 이 subject를 서명했는가
Provenance	누가 어떤 source·parameter로 build했는가
Verification policy	이 signer·builder·source·parameter를 허용하는가

Digest가 같아도 누가 만들었는지 모를 수 있고, provenance 파일이 있어도 공격자가 바꾼 artifact를 가리킬 수 있습니다. 둘을 subject digest로 묶고 expectation을 검사합니다.

11.2 검증할 최소 field

```
subject.digest == deployment.artifact.digest
signature.valid == true
signer in allowed_signers
builder.id in trusted_builders
source.repository == canonical_repository
source.revision == approved_revision
build_type in allowed_build_types
parameters do not enable forbidden target or debug
```

11.3 SLSA를 읽는 방법

SLSA v1.2 specification은 software artifact의 supply chain 무결성을 위한 framework입니다. 이 교재에서는 모든 조직에 특정 level을 자동 요구하지 않습니다. 대신 artifact verification에서 강조하는 것처럼 provenance의 authenticity와 기대값을 확인하고, provenance가 subject·build 정의·run 세부사항을 어떻게 연결하는지 설계 입력으로 사용합니다.

11.4 저장만 하는 provenance의 실패

```
bad:
  provenance.json 파일 존재 → PASS

good:
  signature valid
  AND subject digest exact
  AND builder allowed
  AND source canonical
  AND parameter allowed
  → PASS
```

11.5 SC-15

Candidate는 `digest_matches`, `builder_allowed`, `source_revision_allowed`가 모두 true여야 `PROVENANCE_VERIFIED`입니다. 하나라도 false면 artifact를 quarantine하고 배포를 중단합니다.

12. Migration은 expand와 contract를 나눕니다

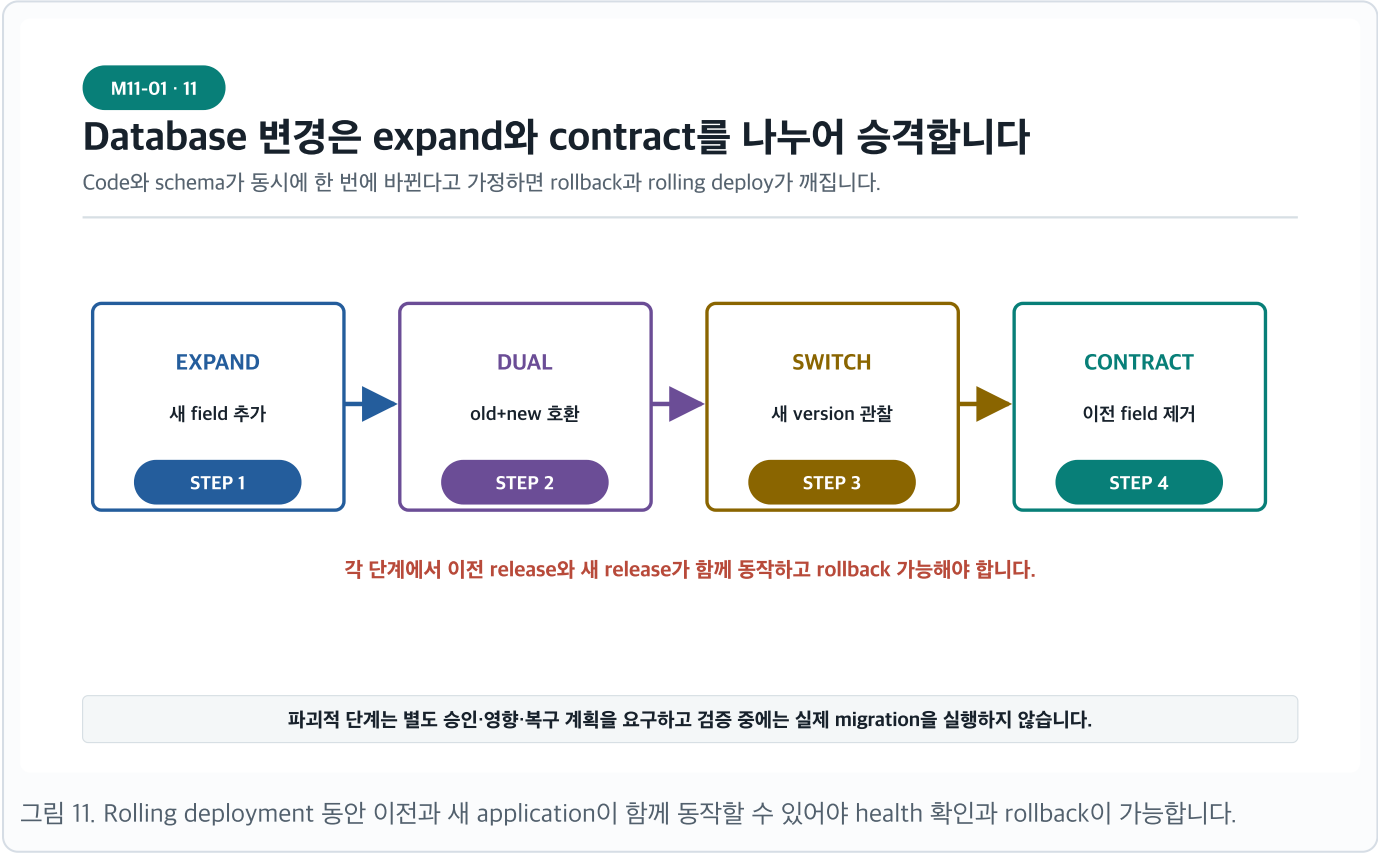
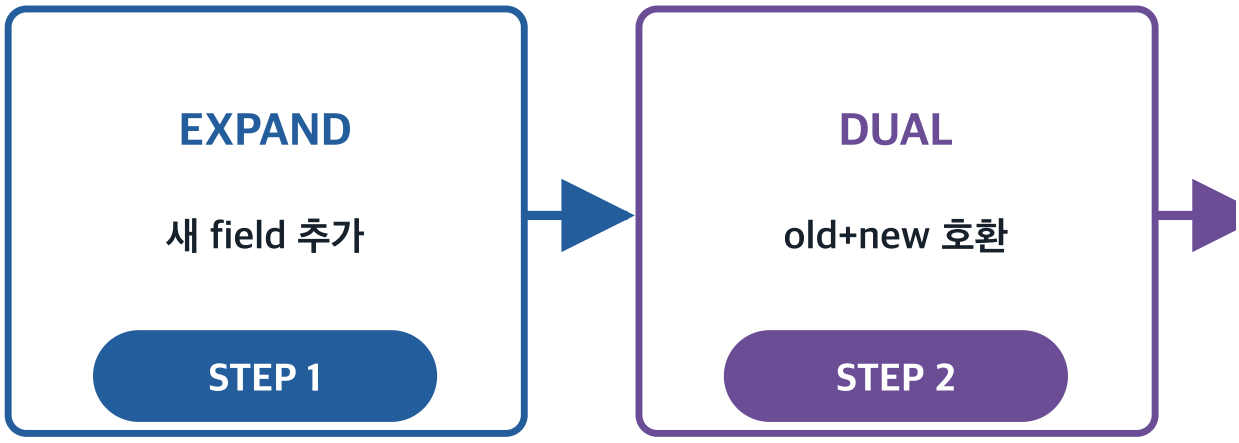


그림 11. Rolling deployment 동안 이전과 새 application이 함께 동작할 수 있어야 health 확인과 rollback이 가능합니다.

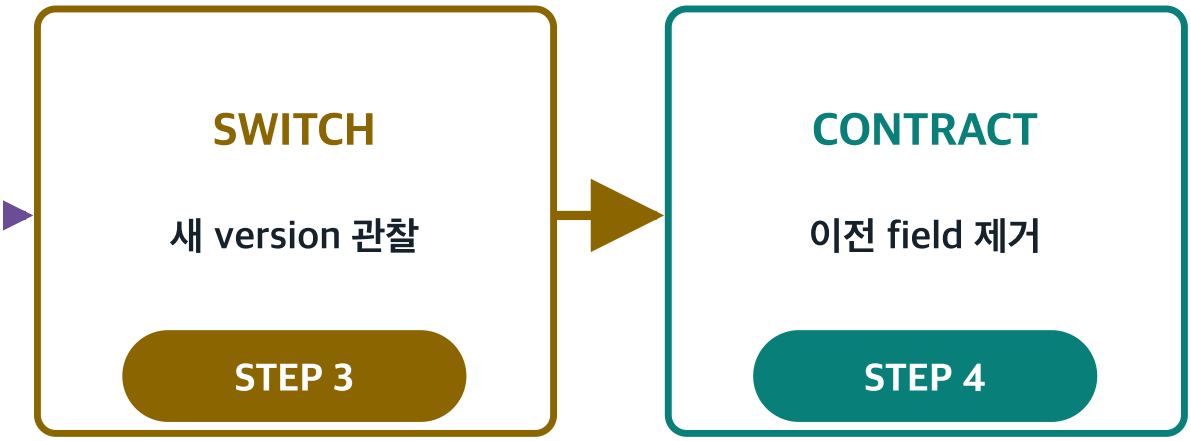
Database 변경은 expand와 contract

Code와 schema가 동시에 한 번에 바뀐다고 가정하면 rollback과 rollir



각 단계에서 이전 release와 새 release가

ıg deploy가 깨집니다.



함께 동작하고 rollback 가능해야 합니다.

12.1 Code와 schema는 동시에 한 순간에 바뀌지 않습니다

Production instance가 여러 개라면 새 version을 배포하는 동안 이전 version도 요청을 처리합니다. 새 code가 old schema를 못 읽거나 old code가 새 schema에서 깨지면 점진 배포와 rollback이 불가능합니다.

12.2 네 단계

단계	하는 일	성공 증거	금지
Expand	새 field·table·index 추가	Old·new app 모두 동작	Old field 삭제
Dual	Dual read·write·backfill	일관성·오류·시간 확인	무제한 이중 경로
Switch	새 경로를 primary로 전환	Metric·data integrity 안정	즉시 old 제거
Contract	Old consumer 종료 뒤 제거	Dependency 0·backup·승인	조기 파괴

12.3 Migration plan의 최소 field

```
schema version
compatible application versions
estimated duration and lock
backfill volume and rate
verification query
stop condition
rollback or roll-forward plan
backup·restore evidence
owner and approval
```

12.4 파괴적 단계는 별도 승인합니다

Column 삭제·type 축소·대량 rewrite는 data 손실과 긴 lock을 만들 수 있습니다. 일반 release 승인에 숨기지 않고 exact step·영향·backup·restore·window·중단 기준을 별도로 검토합니다.

12.5 실습의 안전 경계

실습 engine은 migration을 실행하지 않습니다. `executed=false`, `destructive_step=false`, `rollback_plan_present=true` 같은 판정 field만 비교합니다.

13. Feature flag에 수명과 경계를 줍니다

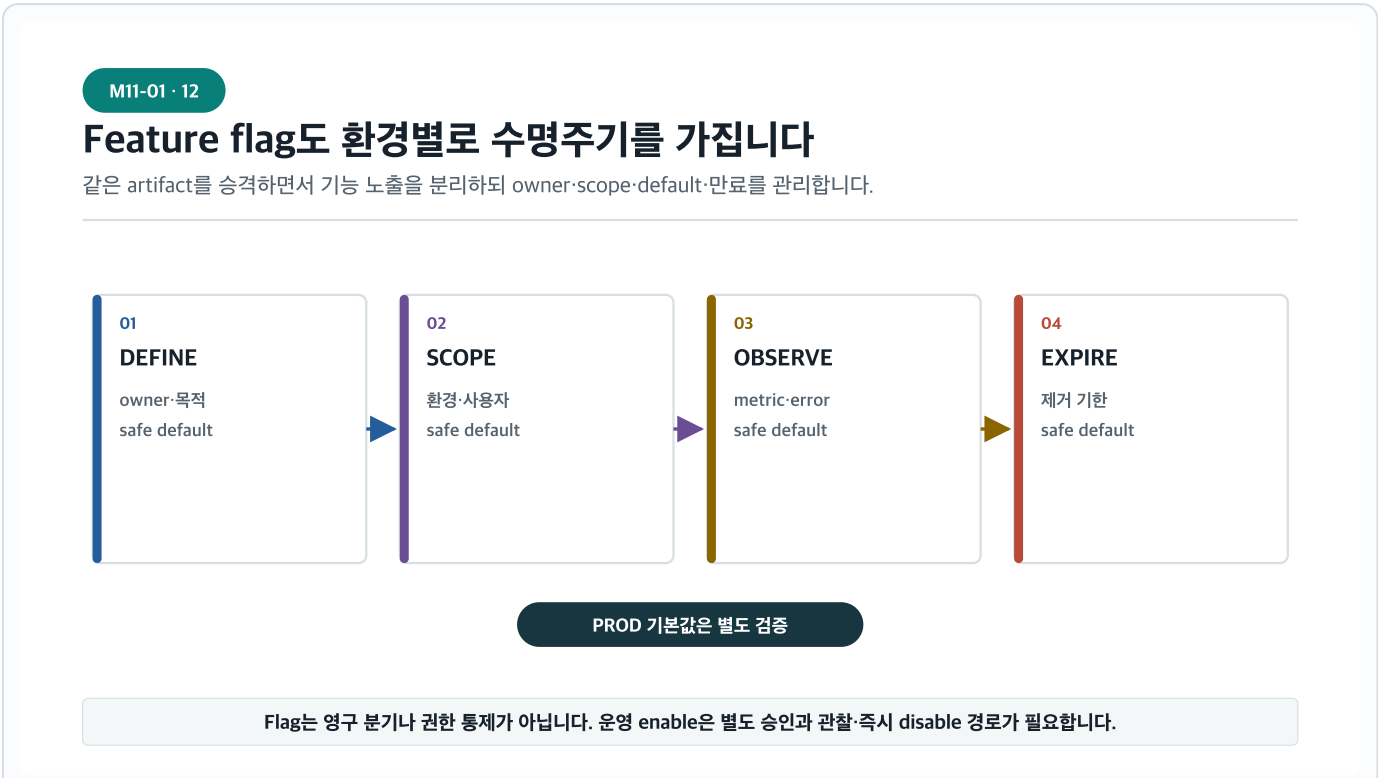


그림 12. Flag를 만드는 순간 owner·환경 scope·safe default·관찰 metric·제거 기한을 함께 만듭니다.

Feature flag는 같은 artifact를 승격하면서 기능 노출을 분리하되 owner·scope·defau

01

DEFINE

owner·목적

safe default

02

SCOPE

환경·사용자

safe default

PROD 기본값

lt·만료를 관리합니다.

03

OBSERVE

metric·error

safe default

04

EXPIRE

제거 기한

safe default

은 별도 검증

13.1 Flag가 잘하는 것과 못하는 것

Flag가 잘하는 것	Flag가 대신하지 못하는 것
Code 배포와 기능 노출 분리	Server authorization
환경·사용자별 점진 rollout	Secret·data·network isolation
문제 기능 빠른 disable	Artifact integrity
Experiment variant 배정	Exact release approval

UI에서 flag가 OFF여도 숨은 API가 production data를 반환하면 보안 경계는 실패입니다.

13.2 Flag 계약

Field	질문
Key·purpose	왜 존재하는가
Environment scope	어느 환경에서 평가되는가
Targeting	누구에게 어떤 variant인가
Default·fail-safe	Provider 장애 때 무엇을 반환하는가
Owner	누가 rollout과 incident를 책임지는가
Metric·alert	무엇이 나쁘면 끄는가
Expiry	언제 제거하거나 재승인하는가

13.3 Production enable은 별도 변경입니다

Test에서 ON인 flag가 production에도 자동 ON이 되지 않도록 environment scope를 분리합니다. Production enable은 exact release와 영향을 보여 주고 승인·audit·rollback 조건을 가집니다.

13.4 Flag debt를 줄입니다

Release가 안정되면 temporary branch와 targeting rule을 제거합니다. 오래된 flag는 조합 수를 폭발시키고 “현재 실제 동작”을 알기 어렵게 만듭니다.

14. Staging parity를 정확하게 해석합니다

14.1 Staging의 가치

Staging은 production과 유사한 runtime·deployment·network shape에서 다음을 예행합니다.

- 정확한 artifact pull과 config binding
- Migration 순서와 소요 시간
- Startup·readiness·smoke check
- External sandbox 연계
- Traffic shift와 rollback 절차
- Release evidence 생성

14.2 Staging이 production과 같을 수 없는 이유

차이	왜 남는가	추가 Production Gate
Data	실제 사용자 보호	합성·canary·business signal
Traffic	규모·패턴 다름	Progressive delivery·capacity
External provider	Sandbox와 live 차이	Provider account·destination 재검사
Credential	환경별 identity 분리	Prod token audience·scope 검증
Network·quota	비용과 격리	IaC plan·quota·route 확인
사람 행동	실제 운영 압력	승인·on-call·runbook 준비

14.3 “Production과 동일” 대신 쓸 문장

```
Staging은 runtime·deployment·migration mechanism이 운영과 유사합니다.
Data·credential·provider destination·traffic·quota는 분리되어 있습니다.
운영 차이 목록은 release evidence로 남기고 production health gate에서 별도 검증합니다.
```

14.4 The Twelve-Factor App과 parity

Dev/prod parity는 시간·사람·도구의 차이를 작게 유지하라고 설명합니다. Build, release, run은 code를 build로, 환경 config와 결합한 것을 release로, 실제 process를 run으로 구분합니다. Config는 배포마다 달라지는 값을 code와 분리하는

원칙을 제공합니다. 이 세 원칙을 함께 읽으면 “유사한 mechanism + 분리된 환경 값 + 동일 artifact 승격”으로 이어집니다.

15. 운영 승격은 exact release 상태 기계입니다

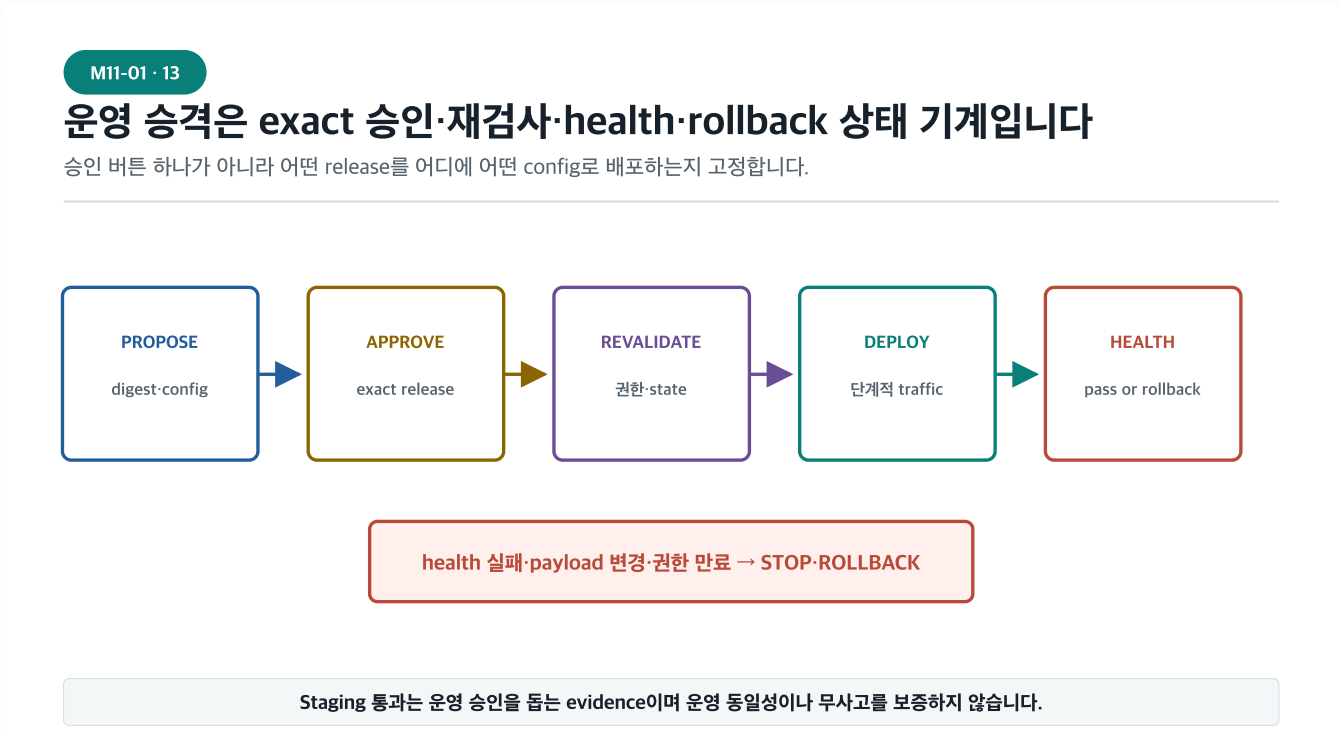
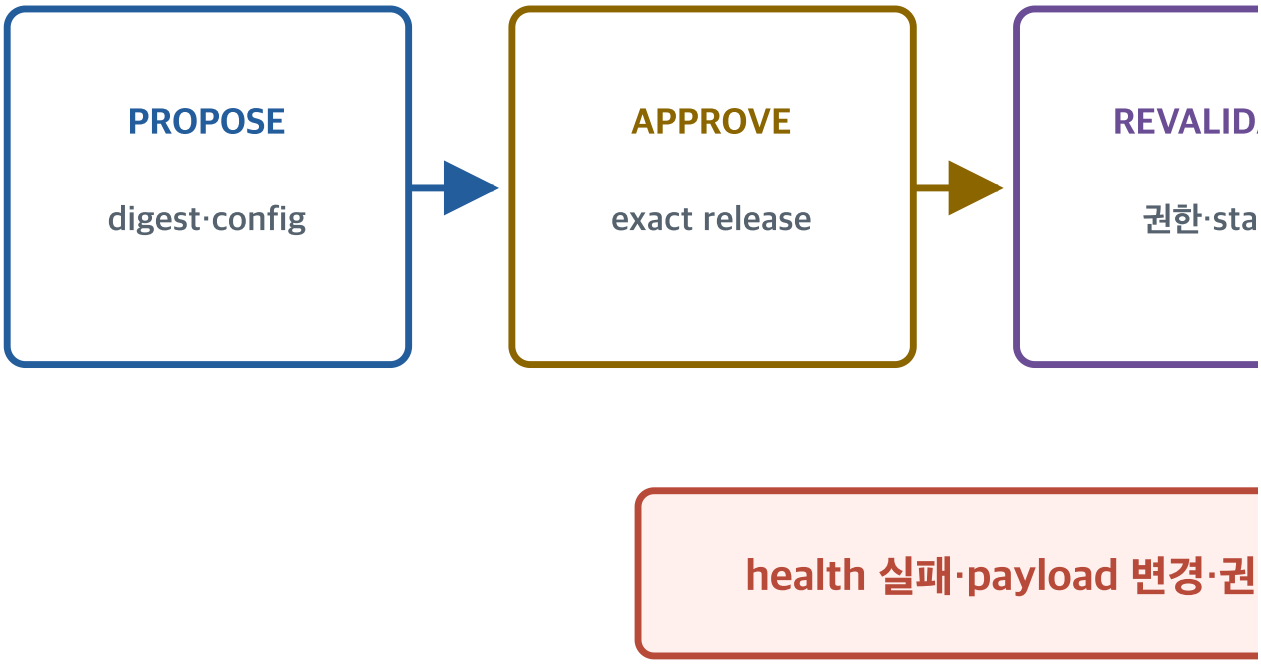
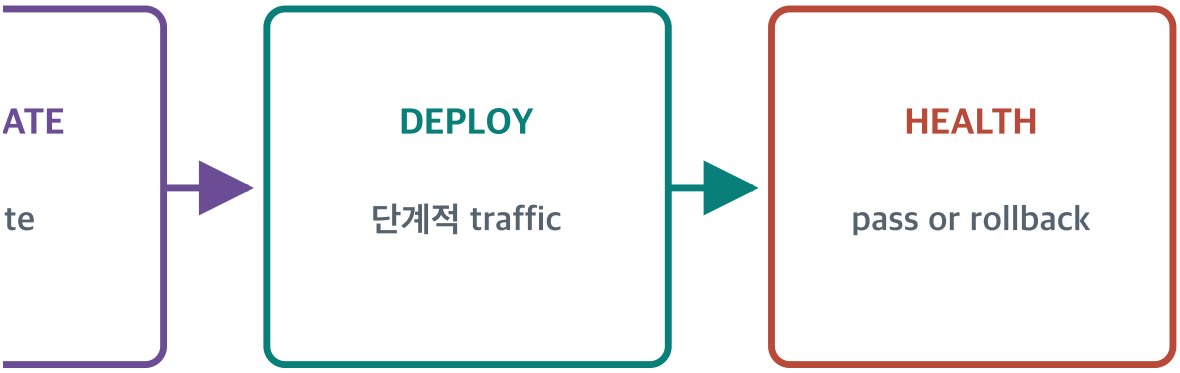


그림 13. Generic한 ‘배포 승인’이 아니라 exact artifact·config·migration·target을 승인하고 실행 직전에 다시 검사합니다.

승인 버튼 하나가 아니라 어떤 release를 어디에 어떤 config로 배포하는



=지 고정합니다.



한 만료 → STOP·ROLLBACK

15.1 다섯 상태

```
PROPOSE
→ APPROVE exact release
→ REVALIDATE unchanged state
→ DEPLOY progressively
→ HEALTH pass or ROLLBACK
```

15.2 Exact payload

승인 field	예
Target environment	`prod-kr-primary`
Artifact digest	`sha256:demo-2401`
Config version	`prod-config-2026-07-16.3`
Migration version	`schema-042-expand`
Flag snapshot	`flags-prod-184`
Evidence bundle	`ev-release-2401`
Window·expiry	`2026-07-16T21:00+09:00`, 30분

승인 뒤 digest·config·migration·target·권한·window 중 하나라도 바뀌면 기존 승인은 무효입니다.

15.3 Separation of duties

변경 작성자, review 승인자, production release 승인자, 실행 identity를 분리합니다. 작은 team에서는 사람이 완전히 다르기 어려울 수 있지만, 최소한 자동 Gate·immutable evidence·사후 review·짧은 credential로 독단과 실수를 줄입니다.

15.4 Health Gate

단계	확인	실패 outcome
Startup	Config·dependency 초기화	Process 시작 중단
Readiness	Traffic 수신 준비	Route에 넣지 않음
Smoke	핵심 read·write·integration	Promotion hold

단계	확인	실패 outcome
Progressive health	Error·latency·business safety	Traffic 확대 중단
Data integrity	Record·queue·migration 상태	Rollback·incident

15.5 Rollback은 계획이 아니라 검증된 경로입니다

“문제면 되돌린다”는 충분하지 않습니다. 이전 digest·config·compatible schema·traffic reversal·권한자·예상 시간·최근 연습 evidence가 필요합니다.

16. Drift와 Release Gate를 함께 봅니다

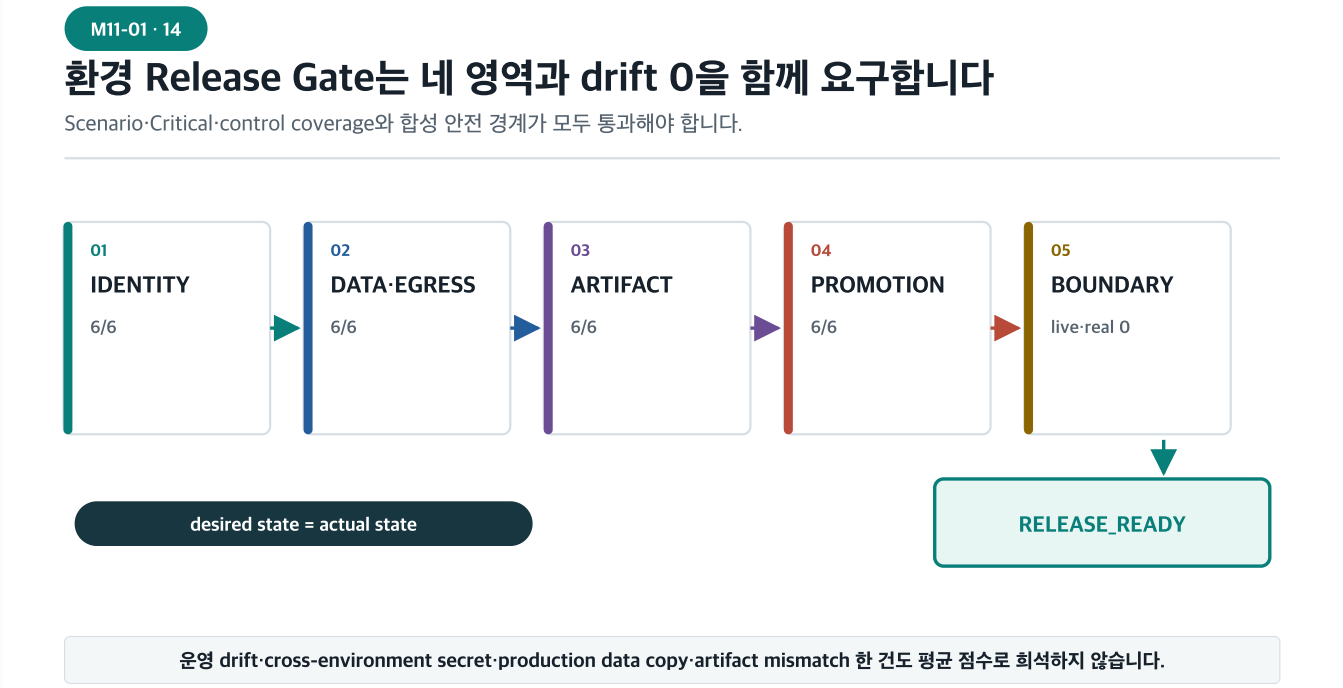
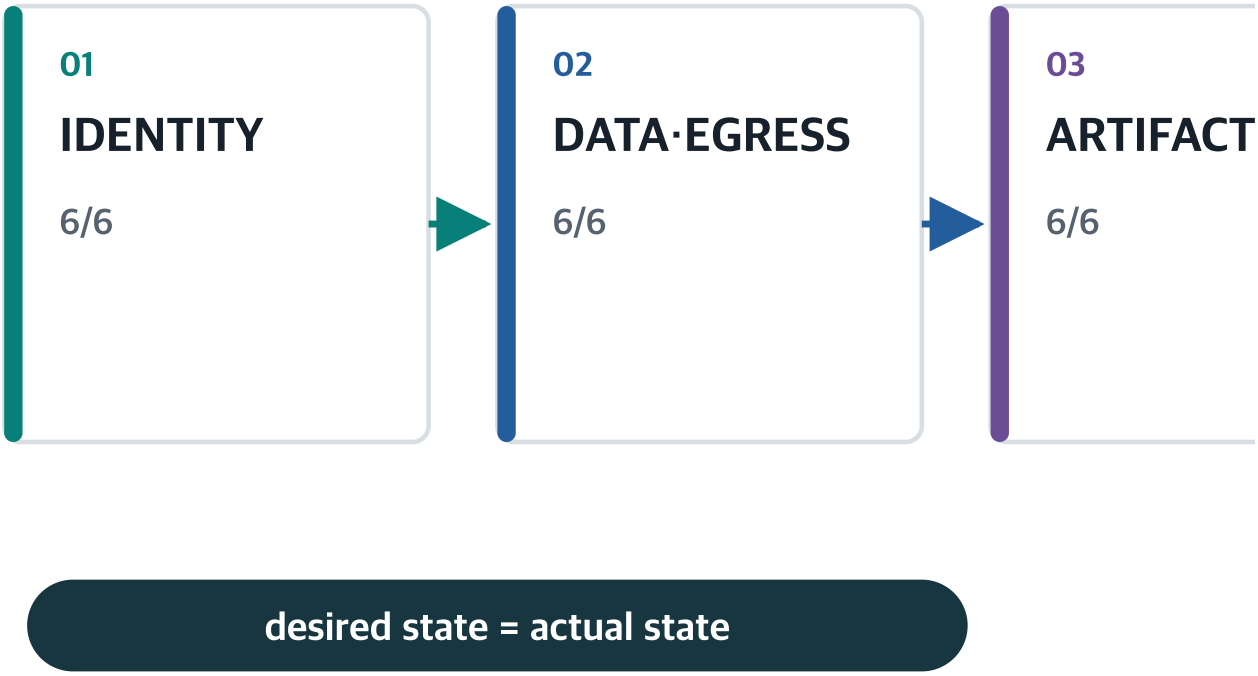


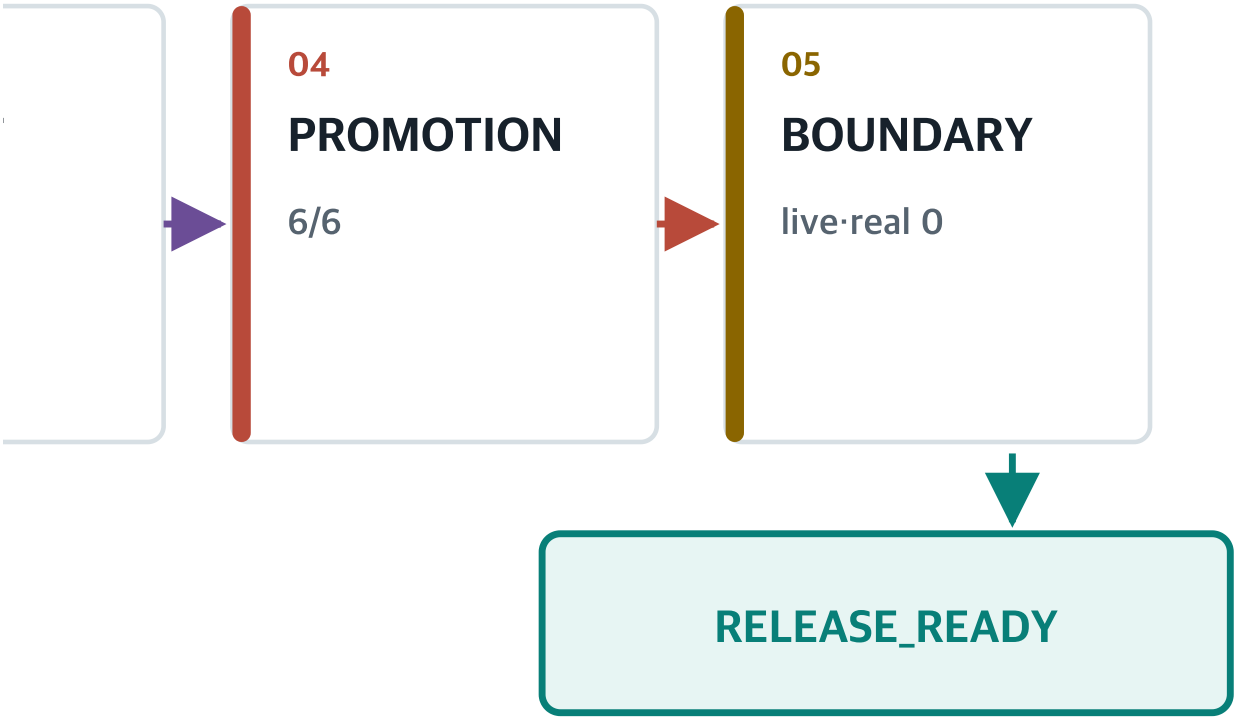
그림 14. 원하는 상태와 실제 상태를 비교하고, 네 영역의 Critical control과 미승인 drift 0을 함께 요구합니다.

[illegible] Release Gate로 배포 [illegible]

Scenario·Critical·control coverage와 합성 안전 경계가 모두 통과해야



: 합니다.



16.1 Desired와 actual

```
desired:
  environment=prod
  artifact=sha256:demo-2401
  config=prod-config-3
  secret_ref=prod/app/v7
  egress=[mail-live-approved]

actual:
  environment=prod
  artifact=sha256:demo-2401
  config=prod-config-2   ← drift
  secret_ref=prod/app/v7
  egress=[mail-live-approved, unknown-webhook] ← drift
```

16.2 Drift의 종류

Drift	예	우선 행동
Identity	Test role이 prod permission 획득	Revoke·deploy block
Config	Debug true·old config version	Safe value·investigate
Data·egress	Unknown destination 추가	Network block
Artifact	Runtime digest가 승인본과 다름	Traffic stop·quarantine
Promotion	Flag·migration·approval mismatch	Release block

16.3 Manual change를 숨기지 않습니다

Incident 중 console에서 긴급 변경할 수 있습니다. 변경을 금지한다는 선언만으로 끝내지 않고, 목적·승인·event·expiry를 남기고 incident 뒤 desired state에 반영하거나 되돌립니다.

16.4 Drift 0의 정확한 뜻

“차이가 하나도 없다”가 아니라 **승인되지 않고 설명되지 않은 차이가 0**이라는 뜻입니다. Provider가 만드는 dynamic field와 의도한 canary replica 수처럼 허용 차이는 rule로 정규화합니다.

17. 12개 통제를 evidence에 연결합니다

Control	질문	주 enforcement	대표 scenario
CTRL-01	목적·사용자·data·행동·owner가 있는가	Manifest·review	01·20·24
CTRL-02	Identity가 일치하고 누락 시 멈추는가	Startup·resource policy	01·02·03·06·12
CTRL-03	Config schema·source·version·default를 검증하는가	Config loader	04·06·18·23
CTRL-04	Secret·service identity가 분리됐는가	Secret resolver·IAM	04·05·09·12·19
CTRL-05	합성 data와 prod access 0인가	Data gate·policy	07·08·09
CTRL-06	Sandbox·allowlist·cap이 있는가	Egress gateway	10·11
CTRL-07	Account·network·resource가 격리됐는가	IAM·network·IaC	03·05·07·09·11·19·23
CTRL-08	같은 immutable digest를 승격하는가	Build·registry·deploy	13·14·15
CTRL-09	Provenance와 exact artifact를 검증하는가	Verification gate	14·15·21
CTRL-10	Migration·flag lifecycle이 안전한가	Schema·flag controller	16·17·18
CTRL-11	Exact 승인·health·rollback이 있는가	Release controller	16·17·19·20·21·22
CTRL-12	Drift·audit·Gate evidence가 있는가	Reconciler·review	12·22·23·24

17.1 Control 이름보다 강제 위치를 씁니다

“환경 분리 정책 있음”은 evidence가 아닙니다. 다음처럼 code와 policy 위치를 적습니다.

```
CTRL-04
  secret resolver: reference.environment == workload.environment
  resource API: token.subject and audience verification
  IAM: test identity has zero prod role bindings
  evidence: deny event + access review + scenario SC-04·05·09
```

17.2 양방향 추적

모든 control은 하나 이상의 scenario와 evidence를 가져야 하고, 모든 scenario는 보호하려는 control을 가져야 합니다. Orphan control은 선언일 뿐이고 orphan scenario는 왜 실행하는지 모르는 test입니다.

18. 24개 scenario를 네 영역으로 실행합니다

18.1 환경 식별·설정 · SC-01~06

ID	Scenario	Expected code
01	Manifest 목적·owner 연결	`ENVIRONMENT_IDENTIFIED`
02	Identity 누락	`ENVIRONMENT_ID_REQUIRED`
03	표시·runtime·resource 불일치	`ENVIRONMENT_MISMATCH_BLOCKED`
04	Test의 prod secret reference	`CROSS_ENV_SECRET_BLOCKED`
05	다른 service identity	`CROSS_ENV_IDENTITY_DENIED`
06	Production unsafe default	`PRODUCTION_SAFE_DEFAULTS`

18.2 데이터·외부 연계 · SC-07~12

ID	Scenario	Expected code
07	운영 원문을 test seed로 복제	`SYNTHETIC_DATA_REQUIRED`
08	고정 합성 seed와 reset	`SYNTHETIC_SEED_LOADED`
09	Test identity의 prod DB write	`PRODUCTION_WRITE_DENIED`
10	Dev mail·payment live 호출	`SANDBOX_DESTINATION_USED`
11	Test webhook의 prod destination	`DESTINATION_NOT_ALLOWED`
12	Telemetry environment·release·secret 0	`TELEMETRY_ENVIRONMENT_LABELED`

18.3 산출물·변경 · SC-13~18

ID	Scenario	Expected code
13	같은 digest를 staging으로 승격	`ARTIFACT_PROMOTED`
14	Production rebuild·mutable tag	`REBUILD_BLOCKED`

ID	Scenario	Expected code
15	Provenance·digest 검증	`PROVENANCE_VERIFIED`
16	Expand-contract 호환	`MIGRATION_COMPATIBLE`
17	파괴적 migration	`DESTRUCTIVE_MIGRATION_STOPPED`
18	Flag scope·owner·expiry	`FLAG_SCOPED`

18.4 승격·운영 판정 · SC-19~24

ID	Scenario	Expected code
19	Developer role의 prod deploy	`PRODUCTION_DEPLOY_DENIED`
20	Staging evidence와 prod 차이	`STAGING_EVIDENCE_ACCEPTED`
21	Exact artifact·config·migration 승인	`EXACT_RELEASE_APPROVED`
22	Health 실패와 rollback	`HEALTH_FAILED_ROLLBACK`
23	Desired·actual drift	`DRIFT_DETECTED`
24	전체 Release Gate	`RELEASE_READY`

18.5 Expected를 관찰 가능한 field로 씁니다

“환경을 안전하게 분리한다”는 test oracle이 아닙니다.

```
SC-11 expected:
  code = DESTINATION_NOT_ALLOWED
  request_sent = false
  destination_allowed = false
  sensitive_fields = 0
```

실패가 나면 어느 field가 달랐는지 바로 보이고 defect와 retest를 만들 수 있습니다.

19. 스튜디오에서 세 version을 비교합니다

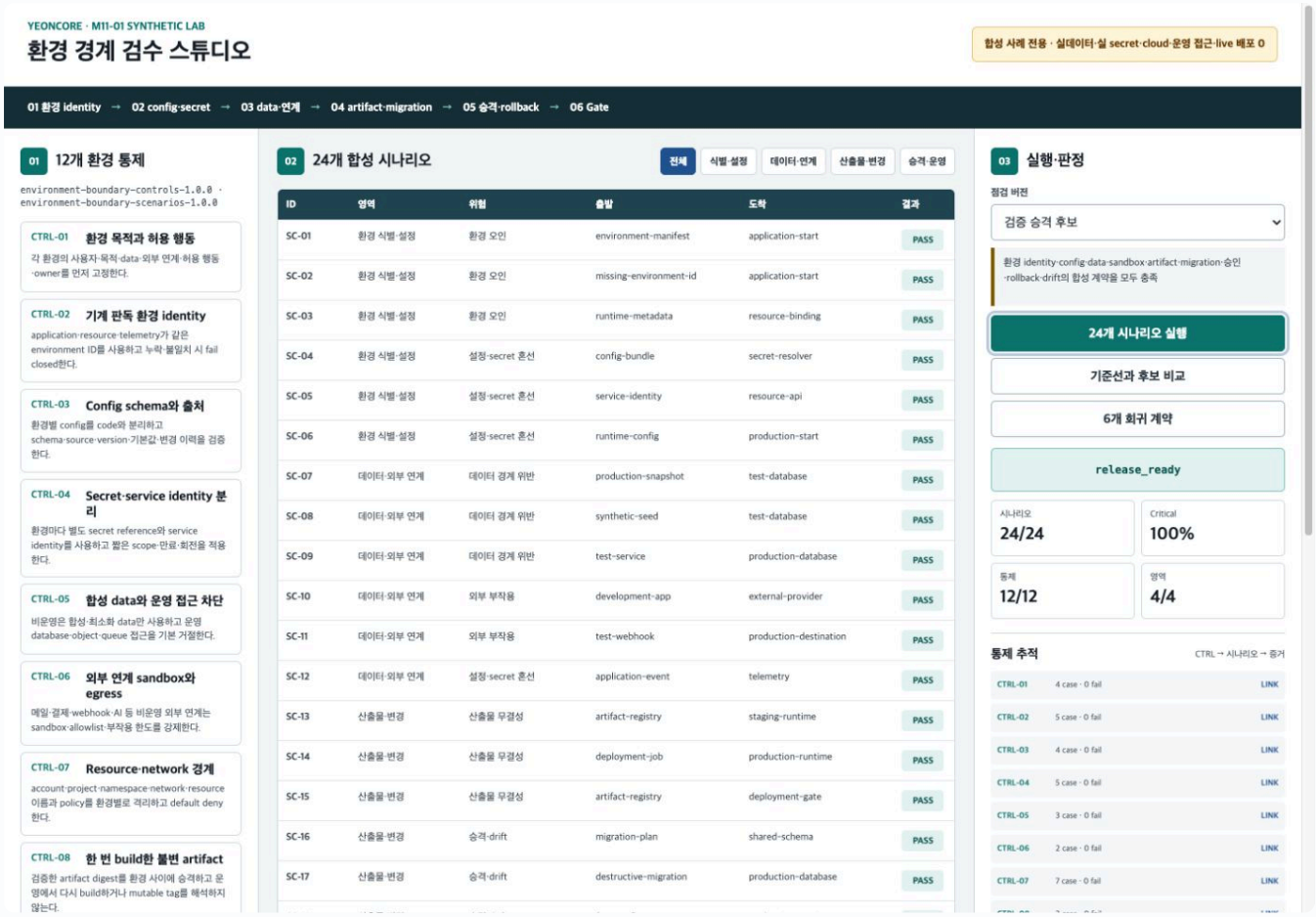


그림 15. 상단 지표, scenario mismatch, 오른쪽 version·비교·회귀 panel을 한 화면에서 연결합니다.

19.1 세 version의 학습 의미

Version	통과	남은 실패	Decision	학습 의미
`label-only-v1`	6/24	18	`blocked_environment_crossing`	이름만 분리해서는 경계가 아님
`separated-but-rebuilt-v2`	16/24	8	`blocked_promotion_drift`	자원 분리만으로 artifact·승인 drift가 남음
`verified-promotion-v3`	24/24	0	`release_ready`	여섯 경계와 승격 evidence가 연결됨

19.2 기준선의 18개 실패

기준선은 SC-01·06·08·12·13·20만 통과합니다. 이름·일부 safe default·합성 seed·telemetry·기본 promotion·staging 차이 기록은 있지만, 누락 identity와 cross-environment secret·data·destination·artifact·approval·drift를 막지 못합니다.

19.3 중간 version의 8개 실패

```
SC-04 prod secret reference
SC-07 production data copy
SC-11 production webhook destination
SC-14 production rebuild·mutable tag
SC-15 provenance not verified
SC-21 generic approval
SC-23 silent drift
SC-24 release gate blocked
```

이 version은 “server와 project는 나눴다”는 설명이 왜 충분하지 않은지 보여 줍니다.

19.4 후보를 읽는 순서

```
1. decision = release_ready
2. passed = 24, failed = 0
3. critical_pass_rate = 1.0
4. control_coverage = 1.0
5. lane_coverage = 1.0
6. all gate_checks = true
7. safety boundary = real data·secret·network·prod access·live deployment·side effect 0
```

19.5 비교와 회귀

기준선과 후보 비교는 `fixed=18`, `regressed=0`, `accept_candidate`입니다. 회귀는 다음 여섯 계약을 고정합니다.

ID	계약
REG-01	이름만 분리한 결함 18건 탐지
REG-02	부분 분리 결함 8건 탐지
REG-03	후보 24/24 통과
REG-04	네 lane 모두 coverage

ID	계약
REG-05	열두 control 모두 coverage
REG-06	실 data·secret·network·prod access·live deployment·side effect 0

20. 작은 화면에서도 evidence를 잃지 않습니다

그림 16. 좁은 화면에서는 요약·version·scenario를 세로로 읽고, 넓은 표는 해당 영역 안에서만 좌우로 이동합니다.

20.1 화면은 evidence의 소비 경로입니다

좋은 back-end evidence가 있어도 UI가 `24 PASS` 만 보여 주면 사용자는 어떤 target과 digest를 판정했는지 알 수 없습니다. 최소한 다음을 확인할 수 있어야 합니다.

- 실행한 variant와 contract version
- Scenario ID·title·risk·control
- Expected·actual·mismatch field
- Critical·lane·control coverage
- Release decision과 failed gate
- 합성 경계와 실제 claim 제한

20.2 숫자에서 scenario로 내려갑니다

```
summary metric
→ failed lane
→ scenario ID
→ mismatched field
→ control enforcement
→ defect owner·retest evidence
```

20.3 Trace에 원문을 남기지 않습니다

실습 trace는 run ID·variant·scenario count·decision 같은 metadata만 저장합니다. 실제 시스템에서도 secret·personal data·connection string·token·raw config dump를 evidence 편의를 이유로 log에 남기지 않습니다.

21. Release Gate를 evidence 계약으로 운영합니다

21.1 합성 candidate Gate

Gate	기준	실패 시
Scenario pass	24/24	Block
Critical pass	100%	Block
Lane coverage	4/4	Block
Control coverage	12/12	Block
Real data·secret	0	Block·안전 경계 위반
External network·prod access	0	Block·조사

Gate	기준	실패 시
Live deployment·side effect	0	Block·조사
Orphan·duplicate	0	Contract 수정
Residual risk	Owner·승인	승인 없으면 Block

21.2 실제 release에 추가할 evidence

Domain	실제 evidence
Identity	IAM binding·access review·short-lived token claim
Config	Schema validation·snapshot·source version
Secret	Reference·rotation·resolver audit
Data	Synthetic seed·prod access query·retention
Integration	Provider account·destination allowlist·egress log
Resource	IaC plan·network policy·inventory
Artifact	Digest·signature·registry metadata
Provenance	Verification result·builder·source expectation
Migration	Dry run·compatibility·backup·restore evidence
Promotion	Exact approval·revalidation·separation of duties
Health	Smoke·canary·threshold·rollback exercise
Drift	Desired·actual diff·exception·remediation

21.3 `release_ready`의 정확한 의미

`release_ready`는 이 합성 contract에서 모든 Gate가 true라는 판정입니다. 실제 cloud의 안전·성능·규제·복구 능력을 자동 보증하지 않습니다. 판정의 범위·version·실행 시점·data·제한을 함께 보관합니다.

21.4 즉시 Block 조건

- Environment identity 누락 또는 target mismatch
- Cross-environment secret·identity·production data access

- Nonproduction에서 live destination 호출
- Artifact digest·signature·provenance expectation mismatch
- Production rebuild 또는 mutable tag 해석
- 미승인 destructive migration
- Generic approval 또는 release 변경 뒤 재검사 실패
- Health threshold 실패와 rollback 불가
- Production artifact·config·permission의 미승인 drift
- Critical scenario 실패·evidence 누락·잔여 위험 미승인

21.5 예외는 만료되는 위험 결정입니다

필드	반드시 기록할 것
Scope	어느 environment·resource·operation인가
이유	왜 기준을 지금 충족할 수 없는가
영향	실패하면 누구에게 무엇이 일어나는가
보완 통제	피해 가능성과 범위를 어떻게 줄이는가
Owner·approver	누가 책임지고 누가 수용하는가
Expiry	언제 자동으로 무효가 되는가
Retest trigger	어떤 변경·incident·날짜에 다시 보는가

22. 공식 기준을 실무 통제로 옮깁니다

22.1 NIST SSDF와 CI/CD supply chain

NIST SP 800-218 SSDF 1.1은 안전한 software 개발 관행을 조직의 development lifecycle에 통합하기 위한 공통 언어를 제공합니다. 이 매뉴얼의 versioned control·scenario·evidence·owner는 그 관행을 환경 경계 검수에 적용한 학습 구조입니다.

NIST SP 800-204D는 cloud-native application의 CI/CD pipeline과 software supply chain 보안을 다룹니다. Source·build·artifact·deployment를 각각의 identity와 trust boundary로 보고, production deploy 권한을 일반 개발 권한과 분리해야 하는 이유를 보강합니다.

22.2 OWASP CI/CD와 secrets

OWASP CI/CD Security Cheat Sheet은 pipeline execution node·configuration·third-party service·credential·artifact를 공격면으로 봅니다. 실무에서는 runner를 ephemeral하게 만들고, pipeline permission과 network를 줄이며, artifact integrity를 검증합니다.

OWASP Secrets Management Cheat Sheet은 secret의 생성·저장·배포·rotation·revocation·audit lifecycle을 설명합니다. 환경 variable을 사용할 수 있다는 사실과 secret lifecycle이 관리된다는 사실은 다릅니다.

22.3 SLSA

SLSA build track basics는 build platform과 provenance의 보증을 단계적으로 높이는 관점을 제공합니다. 조직은 자신의 위험과 현재 capability에 맞게 목표를 정하되, 배포 Gate에서 exact subject·trusted builder·canonical source·allowed parameter를 검증해야 합니다.

22.4 GitHub deployment environments

GitHub deployment environments는 workflow job의 target environment에 protection rule·secret·variables·deployment history를 연결하는 개념을 제공합니다. 환경 관리와 deployment control의 구체 기능·제한은 plan과 공개 상태에 따라 달라질 수 있으므로 사용하는 계정의 최신 문서를 확인합니다.

22.5 기준과 implementation을 구분합니다

기준이 주는 것	우리 system이 정해야 하는 것
공통 위험·원칙·검증 관점	Environment ID와 목적
Supply chain과 secret lifecycle	Provider·account·policy 구조
Provenance·approval 개념	Expected signer·builder·source
Deployment protection 가능성	정확한 reviewer·branch·Gate
Parity와 config 분리 원칙	Safe default·schema·release bundle

공식 문서를 링크했다는 사실만으로 구현이 안전해지지 않습니다. 각 원칙을 code·policy·scenario·evidence·owner로 옮깁니다.

23. 다음 매뉴얼로 넘길 handoff

23.1 M11-02 · 도메인·HTTPS·클라우드로 배포하기

이번 매뉴얼에서 다음 입력을 준비해 넘깁니다.

- Production environment ID·owner·region
- Exact artifact digest와 provenance verification policy
- Config schema와 production secret reference 목록
- Account·project·network·resource desired state
- Domain·DNS·certificate에 필요한 identity와 approval
- Deployment strategy·health check·rollback point
- External live destination allowlist
- Release evidence bundle 형식

M11-02는 실제 provider에서 domain·HTTPS·cloud deployment를 다룹니다. M11-01의 합성 `release_ready` 만으로 실제 배포하지 않습니다.

23.2 M11-03 · 모니터링·백업·장애 대응

이번 매뉴얼에서 다음 signal과 contract를 넘깁니다.

- Environment·release ID가 붙은 telemetry
- Startup·readiness·smoke·release health threshold
- Environment mismatch·cross-env access alert
- Artifact·config·permission drift alert
- Rollback trigger와 이전 release reference
- Migration·data integrity signal
- Residual risk와 exception expiry

M11-03은 dashboard·alert·backup·restore·incident·postmortem을 더 깊게 설계합니다.

24. 현업 적용 체크리스트

24.1 목적·identity

- [ ] 각 environment의 사용자·목적·허용 data·side effect·owner가 있습니다.
- [ ] Canonical environment ID가 application·resource·telemetry·deployment에서 같습니다.
- [ ] 누락·unknown·mismatch는 production default 없이 fail closed합니다.
- [ ] Environment name·branch·URL·namespace만으로 경계를 주장하지 않습니다.
- [ ] Ephemeral·persistent 환경의 생성·reset·expiry·폐기 규칙이 있습니다.

24.2 Config·secret·data

- [ ] Config key마다 type·required·source·version·safe default가 있습니다.
- [ ] Secret 원문 대신 환경별 reference와 workload identity를 사용합니다.
- [ ] Cross-environment credential과 role binding이 0입니다.
- [ ] 비운영은 versioned synthetic seed를 사용하고 reset할 수 있습니다.
- [ ] Nonproduction identity의 production database·object·queue 접근이 기본 거절됩니다.

24.3 External·resource

- [ ] Mail·payment·webhook·AI provider가 환경별 account·sandbox를 사용합니다.
- [ ] Request 전 destination allowlist·data class·side-effect cap을 검사합니다.
- [ ] Account·project·network·resource policy가 환경별로 분리됩니다.
- [ ] 환경 사이 연결은 exact source·destination·operation·expiry로 제한됩니다.
- [ ] Desired state는 IaC·configuration as code로 review할 수 있습니다.

24.4 Artifact·promotion

- [ ] 한 번 build한 immutable digest를 test·stage·prod로 승격합니다.
- [ ] Production에서 source checkout·rebuild·mutable tag 해석을 하지 않습니다.
- [ ] Signature·subject·builder·source·parameter를 배포 전에 검증합니다.
- [ ] Migration은 expand-contract와 compatibility window를 가집니다.
- [ ] Feature flag에 environment scope·safe default·owner·metric·expiry가 있습니다.
- [ ] Exact artifact·config·migration·flag·target에 승인을 결합합니다.

24.5 Health·drift·Gate

- [ ] Startup·readiness·smoke·progressive health 기준이 있습니다.
- [ ] 이전 digest·config·compatible schema로 rollback을 연습했습니다.
- [ ] Desired와 actual의 미승인 drift를 탐지·차단합니다.
- [ ] 12 control·24 scenario·Critical 100% evidence를 연결합니다.
- [ ] 잔여 위험과 예외에 owner·approver·expiry·retest trigger가 있습니다.
- [ ] 실제 release에는 cloud·security·privacy·operations evidence를 추가합니다.

25. 30문항 셀프 테스트

25.1 문제

1. Environment를 branch나 URL 하나로 정의하면 안 되는 이유는 무엇입니까?
2. Dev/prod parity와 environment isolation은 어떻게 동시에 만족합니까?
3. Environment manifest의 최소 여섯 field를 쓰세요.
4. Environment ID가 누락됐을 때 production으로 default하면 왜 위험합니까?
5. Config schema에 필요한 metadata 네 가지를 쓰세요.
6. Environment variable이 secret manager가 아닌 이유는 무엇입니까?
7. Secret 원문과 secret reference의 차이는 무엇입니까?
8. Service identity를 환경별로 나누면 어떤 피해 범위가 줄어듭니까?
9. 운영 snapshot 대신 synthetic seed를 쓰는 세 가지 이점은 무엇입니까?
10. Sandbox endpoint만 사용하면 data boundary가 자동으로 안전해집니까?
11. 비운영 egress Gate가 함께 확인할 네 조건을 쓰세요.
12. Resource name의 `prod-` 접두사가 보안 경계가 아닌 이유는 무엇입니까?
13. 환경 사이 network 연결은 어떤 최소 단위로 허용해야 합니까?
14. Build once·promote가 환경별 rebuild보다 안전한 이유는 무엇입니까?
15. Mutable tag와 pinned digest의 차이는 무엇입니까?
16. Digest·signature·provenance가 각각 답하는 질문은 무엇입니까?
17. Provenance를 저장만 하고 검증하지 않으면 무엇이 빠집니까?
18. Expand-contract migration의 네 단계를 순서대로 쓰세요.
19. 파괴적 migration에 별도 승인이 필요한 이유는 무엇입니까?
20. Feature flag가 authorization을 대신할 수 없는 이유는 무엇입니까?

21. Staging이 production과 유사해도 동일하다고 할 수 없는 세 가지 차이는 무엇입니까?
22. Exact release approval에 결합할 field 다섯 가지를 쓰세요.
23. 승인 뒤 revalidation이 필요한 이유는 무엇입니까?
24. Readiness 실패와 health threshold 실패의 outcome은 어떻게 다릅니까?
25. Rollback contract에 필요한 네 가지 evidence를 쓰세요.
26. Configuration drift란 무엇입니까?
27. “Drift 0”의 정확한 의미는 무엇입니까?
28. Candidate가 24/24여도 실제 cloud가 안전하다고 말할 수 없는 이유는 무엇입니까?
29. Release Gate가 평균 pass rate 외에 Critical 100%를 요구해야 하는 이유는 무엇입니까?
30. M11-02와 M11-03에 각각 어떤 evidence를 넘겨야 합니까?

25.2 모범 답안

1. Branch와 URL은 source·address 표시일 뿐 identity·config·data·credential·destination·resource·artifact·approval을 강제하지 않기 때문입니다.
2. Runtime·도구·절차·artifact는 같거나 유사하게 유지하고, identity·secret·data·permission·provider destination은 환경별로 분리합니다.
3. 목적, 사용자, 허용 data, external integration, 허용 행동, owner입니다. 수명과 승인도 함께 두면 좋습니다.
4. 가장 불확실한 상태에서 가장 큰 실제 영향을 가진 resource와 credential에 연결될 수 있기 때문입니다. 시작을 중단해야 합니다.
5. Type, required, source, version, environment, sensitive 여부, safe default, owner 중 네 가지 이상입니다.
6. 전달 방식일 뿐 암호화 저장·최소 권한·rotation·revocation·audit·cross-environment 차단을 자동 제공하지 않기 때문입니다.
7. 원문은 실제 권한을 여는 민감 값이고, reference는 관리 system 안의 위치·version을 가리키는 식별자입니다.
8. Test나 dev의 노출·오류가 production resource와 실제 사용자로 확장되는 blast radius가 줄어듭니다.
9. 실제 개인정보를 쓰지 않고, 같은 상태를 reset·재현하며, 필요한 경계·오류·volume을 versioned case로 고정할 수 있습니다.
10. 아닙니다. 운영 원문을 sandbox로 보내면 data boundary 위반입니다. Data class와 destination을 함께 검사합니다.
11. Environment match, provider account, destination allowlist, data class, side-effect cap, idempotency 중 네 조건 이상입니다.
12. 이름은 사람이 보는 안내일 뿐 IAM·network·resource permission이 그 이름을 강제하지 않을 수 있기 때문입니다.
13. Exact source identity·source environment·destination resource·operation 또는 protocol·expiry·audit 단위입니다.
14. 시험한 exact bytes와 production bytes가 같아져 build 시점·dependency·runner 차이로 생기는 숨은 변경을 없애기 때문입니다.

15. Mutable tag는 시간이 지나며 다른 bytes를 가리킬 수 있고, pinned digest는 artifact 내용에 결합된 정확한 식별자입니다.
16. Digest는 bytes 동일성, signature는 허용 signer의 서명, provenance는 builder·source·parameter와 build 경로를 답합니다.
17. Signature authenticity, exact subject digest, trusted builder, canonical source, allowed parameter가 실제 기대와 일치하는지 확인하지 못합니다.
18. Expand, dual, switch, contract입니다.
19. Data 손실·긴 lock·이전 version 파손·rollback 불가 가능성이 커서 exact step과 backup·restore·window를 별도로 검토해야 합니다.
20. Flag는 기능 노출 분기일 뿐 요청 identity와 resource permission을 server에서 강제하지 않기 때문입니다.
21. Data, traffic, credential, provider sandbox·live, network·quota, 실제 사람 운영 중 세 가지입니다.
22. Target environment, artifact digest, config version, migration version, flag snapshot, evidence bundle, window·expiry 중 다섯 가지입니다.
23. 승인과 실행 사이 artifact·config·권한·target·상태가 바뀔 수 있기 때문입니다. 달라졌으면 기존 승인을 무효로 합니다.
24. Readiness 실패는 instance를 traffic에 넣지 않고, 배포 뒤 health threshold 실패는 traffic 확대를 멈추거나 rollback합니다.
25. 이전 artifact digest, 이전 config, compatible schema range, traffic reversal, 권한자, 예상 시간, 최근 연습 중 네 가지입니다.
26. 승인된 config desired state와 실제 runtime config가 시간이 지나며 달라진 상태입니다.
27. 차이가 전혀 없다는 뜻이 아니라 승인되지 않고 설명되지 않은 차이가 0이라는 뜻입니다.
28. 합성 system만 실행했고 실제 cloud account·data·credential·network·pipeline·provider를 검사하지 않았기 때문입니다.
29. 평균이 높아도 cross-environment secret·production write·digest mismatch 같은 한 번의 Critical 실패가 큰 피해를 만들 수 있기 때문입니다.
30. M11-02에는 environment·artifact·config·network·domain·deployment evidence를, M11-03에는 telemetry·health·drift·rollback·backup·incident trigger를 넘깁니다.

26. 마지막 한 장 요약

```
DEFINE
  local·dev·test·stage·prod의 사용자·목적·data·side effect·owner

IDENTIFY
  canonical environment ID + config schema + fail closed

SEPARATE
  secret reference·service identity·data·destination·account·network·resource

BUILD
  canonical source → trusted builder → immutable digest

VERIFY
  signature + subject + builder + source + parameter + provenance

PROMOTE
  same digest + environment config + migration + flag snapshot

APPROVE
  exact target·artifact·config·migration·window + revalidation

OBSERVE
  readiness·smoke·health·data integrity + rollback

RECONCILE
  desired state vs actual state + unapproved drift 0

DECIDE
  24/24 + Critical 100% + 12/12 + 4/4 + residual risk owner
```

26.1 내 산출물

- [ ] Environment manifest
- [ ] Config·secret·identity table
- [ ] Data·destination·resource boundary
- [ ] Artifact digest·provenance policy
- [ ] Migration·flag lifecycle
- [ ] Exact approval·health·rollback contract
- [ ] Desired·actual drift record
- [ ] 24개 scenario·12 control evidence
- [ ] Residual risk·exception·retest

- [ ] M11-02·M11-03 handoff

환경을 잘 나눈다는 것은 server를 여러 대 만드는 일이 아닙니다. 같은 code와 검증 방식을 유지하면서 실제 영향이 섞이지 않게 하고, 시험한 exact artifact만 evidence와 함께 운영으로 승격하는 일입니다.