Cloudflare Cache 심화 실습 가이드

Cache Configuration 전기능 테스트 시나리오

엔지니어 교육용 한국어 자료

2026

시작하기

실습 환경

Lab Origin 서버

본 실습은 Cache 심화 전용 Origin 서버를 사용합니다. 각 시나리오에 맞는 응답 헤더를 동적으로 반환합니다.

항목	값
Cache Lab Origin	`cache.cflarekr.dev` (GCP VM, europe-west3, 34.185.205.86)
Origin 접근 방식	**DNS-only** A 레코드 → 학생 Zone에서 Proxied CNAME
서버 소프트웨어	Python Flask, 포트 80
서버 소스	`/opt/cf-cache-lab/server.py`
서버 인덱스	`http://test-cache.cflarekr.dev/` (전체 엔드포인트 목록)
대시보드	`https://dash.cloudflare.com`
주 측정 도구	`curl -sI`, `curl -sv`, `cf-cache-status` 헤더

학생 Zone 설정 (1회)

각 학생은 본인 Cloudflare Zone에서 아래 DNS 레코드를 추가합니다:

```
Type:    CNAME
Name:    cache   (또는 원하는 이름)
Target:  cache.cflarekr.dev
Proxy:   ON (Proxied, Orange Cloud)
```

이후 모든 실습은 `https://cache.<student-zone>/<scenario-path>` 형태로 진행합니다.

> **왜 DNS-only → Proxied CNAME 구조인가?** `cache.cflarekr.dev`가 DNS-only이므로 GCP IP(34.185.205.86)가 노출되지 않고, 각 학생 Zone의 CF Edge가 이 origin으로 요청을 proxy합니다. 학생 Zone의 Cache Rules·설정이 실제로 적용되는 구조입니다.

기존 Lab vs Cache 심화 비교

항목	App Services Lab	Cache 심화 Lab
Origin	`20.88.188.200` (nginx, 정적파일)	`cache.cflarekr.dev` (Python Flask)
Cache-Control	없음 (기본값)	시나리오별 지정 가능
ETag	Strong ETag	Strong/Weak 선택 가능 + 304 지원
Cache-Tag	없음	`/s19/*` 엔드포인트
에러 시뮬레이션	불가	`/s26/down?always=1`
커버리지	~55% 시나리오	**100% 시나리오**

서버 구성 변경 없이 모든 시나리오 진행 가능

> **중요**: 실습 중 Origin 서버(`cache.cflarekr.dev`)를 수정하거나 재설정할 필요가 없습니다. 서버는 이미 모든 시나리오에 맞는 응답 헤더를 반환하도록 사전 구성되어 있습니다.

모든 시나리오는 **Cloudflare 대시보드 설정만** 변경합니다:

작업	어디서
Cache Rules 생성/수정	Dashboard → Caching → Cache Rules
Edge/Browser TTL	Cache Rule 설정 항목
Cache Key 커스텀	Cache Rule → Cache Key 섹션
Tiered Cache	Caching → Tiered Cache
Always Online / Dev Mode	Caching → Configuration
Purge	Caching → Configuration → Purge

Origin 서버가 반환하는 값(Cache-Control, ETag, Cache-Tag, Content-Type 등)은 **URL 경로로 제어**합니다: - `/s05/cc?val=no-cache` → `Cache-Control: no-cache` 반환 - `/s06/nonexist` → `404` 반환 - `/s19/product/123` → `Cache-Tag: product-123, ...` 반환 - (각 시나리오의 "서버 엔드포인트" 표 참조)

필수 사전 지식

- DNS 레코드 생성 및 Proxy 토글
- `curl` 기본 사용 (`-I`, `-v`, `-H`, `--compressed`)
- HTTP 헤더 읽기 (`Cache-Control`, `ETag`, `Age` 등)

검증 공통 패턴

```
# 캐시 상태 한 줄 확인
curl -sI "https://<zone>/path" | grep -i "cf-cache-status\|age\|cache-control\|etag"

# 본문 포함 전체 응답
curl -sv --compressed "https://<zone>/path" 2>&1 | grep -E "< (cf-|cache|age|etag|content)"

# 반복 확인 (HIT 전환 대기)
for i in 1 2 3; do
  echo "=== Request $i ===" && \
  curl -sI "https://<zone>/path" | grep -i "cf-cache-status\|age"
  sleep 1
done
```

cf-cache-status 값 전체 참조

값	의미
HIT	캐시에서 서빙
MISS	캐시 없어 origin fetch, 이후 저장
EXPIRED	캐시 있으나 TTL 만료, origin 재검증
REVALIDATED	조건부 요청으로 origin이 304 응답, 캐시 갱신
UPDATING	Stale 서빙 중, 백그라운드에서 origin fetch
STALE	TTL 만료 + origin 불가, stale 서빙
BYPASS	캐시 우회 (Cookie, Cache-Control: no-cache 등)
DYNAMIC	캐시 비대상 (HTML 등 기본 미캐시)
NONE/UNKNOWN	캐시 불가 응답

Docs: Cache Responses

Part I — Cache 기초 이해

Scenario 1: Default Cache Behavior 확인

목적

Cloudflare가 기본적으로 캐시하는 파일 타입과 캐시하지 않는 타입을 직접 확인한다.

서버 엔드포인트 (`cache.<student-zone>` 기준)

경로	Content-Type	Cache-Control	예상 결과
`/static/style.css`	text/css	없음	MISS→HIT (기본 캐시)
`/static/app.js`	application/javascript	없음	MISS→HIT
`/static/logo.png`	image/png	없음	MISS→HIT
`/html/page.html`	text/html	없음	DYNAMIC (HTML 기본 미캐시)
`/api/data.json`	application/json	없음	DYNAMIC
`/cache/set-cookie`	text/html	없음	BYPASS (Set-Cookie 포함)

배경 지식

- Cloudflare는 **파일 확장자 기반**으로 기본 캐시 여부를 결정 (MIME type 아님)
- HTML, JSON은 **기본적으로 캐시하지 않음**
- CSS, JS, 이미지, 폰트 등은 기본 캐시 대상
- `Set-Cookie` 헤더가 있는 응답은 캐시하지 않음

실습 단계

Step 1: 기본 캐시 대상 파일 확인

```
ZONE="cache.<student-zone>"

# CSS (기본 캐시 대상) → MISS → HIT
curl -sI "https://$ZONE/static/style.css" | grep cf-cache-status
curl -sI "https://$ZONE/static/style.css" | grep cf-cache-status
# → HIT

# JS
curl -sI "https://$ZONE/static/app.js" | grep cf-cache-status
# → HIT (2번째 요청)

# 이미지 (PNG)
curl -sI "https://$ZONE/static/logo.png" | grep cf-cache-status
```

Step 2: 기본 캐시 비대상 확인

```
# HTML → DYNAMIC (기본 미캐시)
curl -sI "https://$ZONE/html/page.html" | grep cf-cache-status

# JSON API → DYNAMIC
curl -sI "https://$ZONE/api/data.json" | grep cf-cache-status
```

Step 3: Set-Cookie 헤더 영향 테스트

```
# Set-Cookie 반환 엔드포인트 → BYPASS
curl -sI "https://$ZONE/s13/set-cookie?group=A" | grep -i "cf-cache-status\|set-cookie"
```

예상 결과

경로	cf-cache-status	이유
`/static/style.css` (2회째)	HIT	기본 캐시 대상 (.css)
`/html/page.html`	DYNAMIC	기본 미캐시 (HTML)
`/api/data.json`	DYNAMIC	기본 미캐시 (JSON)
`/s13/set-cookie?group=A`	BYPASS	Set-Cookie = 캐시 차단

학습 포인트

> 기본 캐시 대상 확장자: CSS, JS, JPG, PNG, GIF, SVG, ICO, WOFF, WOFF2, MP4, PDF, ZIP 등 HTML/JSON을 캐시하려면 Cache Rule에서 **Cache Everything**을 명시해야 함

Docs

Scenario 2: Default Edge TTL 확인 (status code별)

목적

Cache-Control 헤더가 없을 때 Cloudflare가 status code별로 적용하는 기본 TTL을 확인한다.

배경 지식

HTTP Status	기본 Edge TTL
200, 206, 301	120분
302, 303	20분
404, 410	3분
기타	캐시 안 함

실습 단계

Step 1: 200 응답의 기본 TTL 확인

```
# 첫 요청 (MISS)
curl -sI "https://<zone>/s02/asset.js"

# 60초 후 (HIT, age ~60)
sleep 60 && curl -sI "https://<zone>/s02/asset.js" | grep -i "age:"
```

Step 2: 404 응답의 3분 TTL 확인

```
# 존재하지 않는 경로 → 404
curl -sI "https://<zone>/s06/nonexist"
# cf-cache-status: MISS (첫 요청)

# 1분 후 재요청
sleep 60 && curl -sI "https://<zone>/s06/nonexist"
# cf-cache-status: HIT, age ~60

# 4분 후 (TTL 만료)
sleep 180 && curl -sI "https://<zone>/s06/nonexist"
# cf-cache-status: EXPIRED or MISS (TTL 3분 초과)
```

예상 결과

- 200 응답: 최대 120분 캐시, age 헤더로 경과 시간 확인 가능
- 404 응답: 3분 캐시 후 만료

Docs

Part II — Cache Rules 핵심 설정

Scenario 3: Cache Eligibility — HTML/JSON 캐싱 (Cache Everything)

목적

기본적으로 캐시되지 않는 HTML을 Cache Rule로 캐시 대상으로 만든다.

실습 단계

Step 1: 현재 상태 확인

```
# HTML 현재 상태 → DYNAMIC 예상
curl -sI "https://<zone>/s03/page" | grep cf-cache-status
```

Step 2: Cache Rule 생성

대시보드: `Caching > Cache Rules > Create rule`

```
Rule name: Cache HTML - S03
When: URI Path contains /s03/
Then: Cache eligibility → Eligible for cache
      Edge TTL: Override origin → 30 seconds
```

Step 3: 규칙 적용 후 확인

```
# 첫 요청 → MISS
curl -sI "https://<zone>/s03/page"

# 2번째 요청 → HIT
curl -sI "https://<zone>/s03/page"
```

Step 4: TTL 만료 확인 (30초 후)

```
sleep 35
curl -sI "https://<zone>/s03/page" | grep -i "cf-cache-status\|age"
# EXPIRED or MISS
```

검증 포인트

- DYNAMIC → HIT 전환 확인
- age: 헤더로 캐시된 시간 확인

Docs

Scenario 4: Bypass Cache

목적

특정 경로의 캐시를 완전히 우회(bypass)하는 규칙을 만든다.

배경 지식

- Bypass: 요청이 항상 origin으로 전달됨
- DYNAMIC vs BYPASS: DYNAMIC은 캐시 비대상이지만 CF가 처리, BYPASS는 규칙에 의해 명시적 우회

실습 단계

Step 1: 캐시되어 있는 리소스 확인

```
curl -sI "https://<zone>/s03/page" | grep cf-cache-status
# HIT (Scenario 3에서 설정한 캐시)
```

Step 2: Bypass Cache Rule 생성

```
Rule name: Bypass Cache - Admin
When: URI Path contains /admin/
Then: Cache eligibility → Bypass cache
```

Step 3: Bypass 경로 확인

```
curl -sI "https://<zone>/s10/no-cache" | grep cf-cache-status
# DYNAMIC (bypass 상태에서는 DYNAMIC으로 표시)
```

주의사항

> Bypass rule 적용 시 cf-cache-status가 `BYPASS`가 아닌 `DYNAMIC`으로 표시될 수 있음. 이는 캐시 비적격 처리이기 때문.

Docs

Scenario 5: Edge TTL — 3가지 모드 비교

목적

Edge TTL의 세 가지 모드 동작을 각각 확인한다.

서버 엔드포인트

경로	반환 헤더	용도
`/s05/cc?val=public,max-age=3600`	`Cache-Control: public, max-age=3600`	respect_origin 테스트
`/s05/cc?val=no-cache`	`Cache-Control: no-cache`	no-cache origin 동작
`/s05/cc?val=private`	`Cache-Control: private`	private origin 동작
`/s05/no-cc`	Cache-Control 없음	bypass_by_default 테스트

배경 지식

모드	동작
`respect_origin` (Use CC if present, default if not)	origin Cache-Control 따름. 없으면 기본 CF TTL
`override_origin` (Ignore CC, use this TTL)	origin 헤더 무시, 지정 TTL 강제 적용
`bypass_by_default` (Use CC if present, bypass if not)	origin CC 따름. 없으면 캐시 안 함

실습 단계

Step 1: origin이 `public, max-age=3600` 반환하는 경로 확인

```
ZONE="cache.<student-zone>"
curl -sI "https://$ZONE/s05/cc?val=public,max-age=3600" | grep -i cache-control
# Cache-Control: public, max-age=3600
```

Step 2: override_origin 테스트 (10초 TTL 강제)

```
Cache Rule: URI Path = /s05/cc
Edge TTL: Ignore cache-control header → 10 seconds
```

```
curl -sI "https://$ZONE/s05/cc?val=public,max-age=3600"
# MISS (첫 요청, origin은 3600초 반환)

sleep 12
curl -sI "https://$ZONE/s05/cc?val=public,max-age=3600" | grep cf-cache-status
# EXPIRED or MISS (CF는 10초만 캐시, origin 헤더 무시됨)
```

Step 3: bypass_by_default 테스트

Cache-Control 없는 경로에 적용:

```
Cache Rule: URI Path = /s05/no-cc
Edge TTL: Use cache-control if present, bypass if not
```

```
curl -sI "https://$ZONE/s05/no-cc" | grep cf-cache-status
# DYNAMIC (CC 없음 → bypass_by_default → 캐시 안 함)
```

Step 4: respect_origin으로 no-cache 동작 확인

```
# origin이 no-cache → 매 요청 revalidate (REVALIDATED)
curl -sI "https://$ZONE/s05/cc?val=no-cache" | grep -i "cf-cache-status\|cache-control"
```

Docs

Scenario 6: Status Code TTL

목적

HTTP status code별로 다른 캐시 TTL을 설정한다.

실습 단계

Step 1: Cache Rule — Status Code TTL 설정

```
Rule name: Status Code TTL Test
When: URI Path starts with /s03/
Cache eligibility: Eligible
Edge TTL: Ignore CC
  - 200: 300 seconds
  - 404: 10 seconds
  - 5xx (500-599): 0 seconds (no-cache)
```

Step 2: 200 응답 캐시 확인

```
curl -sI "https://<zone>/s03/page" | grep -i "cf-cache-status\|age"
# HIT, age 증가
```

Step 3: 404 캐시 확인 (10초)

```
curl -sI "https://<zone>/s06/nonexist"
# MISS → 첫 요청

sleep 5 && curl -sI "https://<zone>/s06/nonexist"
# HIT (404 캐시됨)

sleep 8 && curl -sI "https://<zone>/s06/nonexist"
# EXPIRED (10초 초과)
```

학습 포인트

> 404 캐시는 흔한 운영 이슈. 없는 페이지를 잘못 캐시하면 복구 어려움. Status Code TTL로 4xx를 짧게, 또는 0으로 설정 권장.

Docs

Scenario 7: Browser TTL

목적

브라우저 캐시 TTL을 Cloudflare에서 강제로 지정한다.

배경 지식

- `Browser TTL`은 브라우저에 내려가는 `Cache-Control: max-age=` 값을 제어
- Edge TTL (CF → 저장)과 Browser TTL (브라우저 → 저장)은 독립적

실습 단계

Step 1: 현재 브라우저 TTL 확인

```
curl -sI "https://<zone>/static/style.css" | grep -i "cache-control"
```

Step 2: Browser TTL Override 설정

```
Rule: URI Path ends with .css
Browser TTL: Override origin → 86400 seconds (1일)
```

Step 3: 적용 확인

```
curl -sI "https://<zone>/static/style.css" | grep -i "cache-control"
# Cache-Control: max-age=86400
```

Step 4: Browser TTL Bypass

```
Rule: URI Path contains /s05/
Browser TTL: Bypass cache (max-age=0)
```

```
curl -sI "https://<zone>/s05/cc?val=no-cache" | grep -i cache-control
# Cache-Control: max-age=0
```

주의사항

> Browser TTL을 너무 길게 설정하면 콘텐츠 업데이트 후에도 구 버전이 브라우저에 캐시되어 Purge 효과 없음.

Docs

Scenario 8: Serve Stale Content While Revalidating

목적

TTL 만료 시 origin 요청 동안 stale 콘텐츠를 계속 서빙하는 동작을 확인한다.

배경 지식

- `Serve stale` = 기본값 ON: TTL 만료 후 origin fetch 중에도 이전 응답 서빙 (`UPDATING` 상태)
- 비활성화하면 TTL 만료 시 사용자가 origin fetch 완료를 기다려야 함

실습 단계

Step 1: 기본 동작 확인 (Serve stale ON)

```
Rule: URI Path = /s03/page
Edge TTL: 10 seconds
Serve stale: 활성화 (기본값)
```

```
# 첫 요청
curl -sI "https://<zone>/s03/page" | grep cf-cache-status
# MISS

# 12초 후 (TTL 만료)
sleep 12 && curl -sI "https://<zone>/s03/page" | grep cf-cache-status
# UPDATING (stale 서빙 중, 백그라운드 fetch)

# 바로 다시
curl -sI "https://<zone>/s03/page" | grep cf-cache-status
# HIT (백그라운드 fetch 완료)
```

Step 2: Serve stale OFF 비교

```
Rule 동일, Serve stale: Disable stale while updating = ON
```

```
sleep 12 && curl -sI "https://<zone>/s03/page" | grep cf-cache-status
# EXPIRED or MISS (사용자가 origin fetch 기다림)
```

Docs

Scenario 9: Respect Strong ETags

목적

Strong ETag를 통한 조건부 요청(Conditional Request) 동작을 확인한다.

배경 지식

- **Strong ETag**: byte-for-byte 동일성 보장 (`ETag: "abc123"`)
- **Weak ETag**: 의미적 동일성 (`ETag: W/"abc123"`)
- 기본값: CF가 Strong → Weak으로 변환
- `Respect Strong ETags` ON → CF가 Strong ETag 유지, If-None-Match 조건부 요청 지원

실습 단계

Step 1: ETag 기본 동작 확인 (Weak 변환)

```
curl -sI "https://<zone>/static/app.js" | grep -i etag
# ETag: W/"..."  (Weak ETag)
```

Step 2: Respect Strong ETags 활성화

```
Rule: URI Path ends with .js
Respect Strong ETags: ON
```

Step 3: 조건부 요청 테스트

```
# ETag 값 저장
ETAG=$(curl -sI "https://<zone>/static/app.js" | grep -i etag | awk '{print $2}' | tr -d
        '\r')

# If-None-Match 조건부 요청 → 파일 미변경 시 304 예상
curl -sI -H "If-None-Match: $ETAG" "https://<zone>/static/app.js"
# HTTP/2 304 (Not Modified)
# cf-cache-status: REVALIDATED
```

Step 4: 파일 변경 후 (origin content 변경 시)

```
# ETag가 다르면 200 + 새 콘텐츠 반환
# cf-cache-status: MISS or EXPIRED
```

Docs

Scenario 10: Origin Cache Control (Enterprise)

목적

Origin의 Cache-Control 헤더를 CF가 얼마나 엄격하게 따르는지 확인한다.

서버 엔드포인트

경로	반환 Cache-Control	의미
`/s10/no-cache`	`no-cache`	매 요청 revalidate 강제
`/s10/no-store`	`no-store`	캐시 저장 자체 금지
`/s10/private`	`private`	개인 캐시만, CF 캐시 안 함
`/s10/public`	`public, max-age=3600`	공개 캐시 1시간
`/s10/immutable`	`public, max-age=31536000, immutable`	영구 캐시

배경 지식

- **Origin Cache Control ON** (Enterprise 기본): RFC 7234 엄격 준수
  - `no-cache` → 매번 revalidate
  - `no-store` → 캐시 불가
  - `private` → 캐시 불가
- **Origin Cache Control OFF**: CF가 일부 지시어 무시 가능

실습 단계

Step 1: no-cache 동작 확인

```
ZONE="cache.<student-zone>"
curl -sI "https://$ZONE/s10/no-cache" | grep -i "cache-control\|cf-cache-status"
# Cache-Control: no-cache
# cf-cache-status: DYNAMIC (revalidate 대상)
```

Step 2: private 동작 확인

```
curl -sI "https://$ZONE/s10/private" | grep -i "cache-control\|cf-cache-status"
# Cache-Control: private
# cf-cache-status: DYNAMIC (CF 캐시 거부)
```

Step 3: Cache Rule로 no-cache 무시 (override_origin)

```
Cache Rule: URI Path = /s10/no-cache
Edge TTL: Ignore cache-control header → 60 seconds
```

```
curl -sI "https://$ZONE/s10/no-cache" | grep cf-cache-status
# MISS (첫 요청)
curl -sI "https://$ZONE/s10/no-cache" | grep cf-cache-status
# HIT (no-cache 무시, 강제 캐시됨)
```

학습 포인트

> `override_origin` Edge TTL은 `no-cache`, `private` 등을 무시하고 강제 캐시. API 응답 잘못 캐시 위험 → 신중히 사용.

Docs

Scenario 11: Origin Error Page Pass-through

목적

Origin의 5xx 에러 페이지를 그대로 방문자에게 전달할지 또는 CF 기본 에러 페이지를 보여줄지 설정한다.

서버 엔드포인트

경로	응답	내용
`/s11/down`	503	커스텀 HTML 에러 페이지 (항상 503)
`/s11/partial-down`	503 또는 200	50% 확률 503 (간헐적 장애 시뮬레이션)

실습 단계

Step 1: 기본 동작 확인 — CF 에러 페이지 표시됨

```
ZONE="cache.<student-zone>"
curl -sv "https://$ZONE/s11/down" 2>&1 | grep -E "< HTTP|< content"
# HTTP 503
# CF 기본 에러 페이지 표시 (origin 페이지 아님)
```

Step 2: Origin 에러 페이지 내용 확인 (원본)

```
# origin이 반환하는 실제 503 HTML 직접 확인
curl -s "https://$ZONE/s11/down" | grep -i "서비스\|origin\|503"
```

Step 3: Origin Error Page Pass-through 활성화

```
Cache Rule: URI Path = /s11/down
Origin error page pass-through: ON
```

```
curl -s "https://$ZONE/s11/down" | grep -i "서비스 점검\|X-Error-From"
# origin의 실제 에러 페이지 내용 표시
```

Part III — Cache Key 고급 설정

Scenario 12: Query String 캐시 동작

목적

Query String 포함 여부에 따른 캐시 분기 동작을 실험한다.

배경 지식

- 기본: URL 전체 (쿼리 포함)가 Cache Key → ?v=1과 ?v=2는 별도 캐시
- Ignore Query String: 쿼리 무관 하나의 캐시 → 캐시 히트율 향상
- Query String Sort: 쿼리 파라미터 순서 정렬 → ?b=2&a=1과 ?a=1&b=2를 동일 캐시

실습 단계

Step 1: 기본 Query String 캐시 분기 확인

```
# 두 URL이 서로 다른 캐시 키
curl -sI "https://<zone>/s03/page?color=red&size=L" | grep cf-cache-status
curl -sI "https://<zone>/s03/page?color=blue&size=M" | grep cf-cache-status
# 각각 별도 MISS → HIT
```

Step 2: Cache Everything + Ignore Query String 설정

```
Rule name: Cache Ignore QS
When: URI Path = /s03/page
Cache eligibility: Eligible for cache
Edge TTL: 60 seconds
Cache Key > Query String: Ignore (exclude: *)
```

```
# 쿼리와 무관하게 같은 캐시 사용
curl -sI "https://<zone>/s03/page?color=red" | grep cf-cache-status
# MISS (첫 요청)

curl -sI "https://<zone>/s03/page?color=blue" | grep cf-cache-status
# HIT (쿼리 무시, 동일 캐시)
```

Step 3: Query String Sort 확인

```
Rule: Query String Sort → ON
```

```
curl -sI "https://<zone>/s12/items?b=2&a=1" | grep cf-cache-status
# MISS (첫 요청)

curl -sI "https://<zone>/s12/items?a=1&b=2" | grep cf-cache-status
# HIT (파라미터 정렬 후 동일 캐시 키)
```

Step 4: 마케팅 파라미터 제외 (Enterprise)

```
Cache Key > Query String:
  All parameters except: utm_source, utm_medium, utm_campaign, utm_content, fbclid, gclid
```

```
# utm 파라미터 있어도 같은 캐시
curl -sI "https://<zone>/blog/post?utm_source=newsletter" | grep cf-cache-status
curl -sI "https://<zone>/blog/post?utm_source=twitter" | grep cf-cache-status
# 두 번째 → HIT (utm 무시, 동일 콘텐츠)
```

캐시 히트율 개선 포인트

> Analytics tracking parameters (utm_*, fbclid, gclid)는 콘텐츠와 무관 → Cache Key에서 제외하면 히트율 대폭 향상

Docs

Scenario 13: Custom Cache Key — Cookie 기반 분기 (Enterprise)

목적

Cookie 값에 따라 다른 콘텐츠를 캐시하도록 Cache Key를 커스터마이징한다.

서버 엔드포인트

경로	동작
`/s13/landing`	`ab_group` 쿠키 값에 따라 **다른 HTML** 반환 (A=오렌지, B=파란 배경)
`/s13/set-cookie?group=A`	`ab_group=A` 쿠키 set 후 `/s13/landing`으로 redirect
`/s13/set-cookie?group=B`	`ab_group=B` 쿠키 set 후 `/s13/landing`으로 redirect

사용 케이스

- A/B 테스트: `ab_group=A` vs `ab_group=B`
- 언어 설정 쿠키: `lang=ko` vs `lang=en`
- 로그인 여부: `session` 쿠키 존재 여부

실습 단계

Step 1: Cookie 없는 기본 동작 — 쿠키 무관하게 같은 캐시

```
ZONE="cache.<student-zone>"

# Cache Rule 먼저 적용: URI Path = /s13/landing, Eligible for cache, TTL=60s
curl -sI "https://$ZONE/s13/landing" | grep cf-cache-status
# MISS (첫 요청)

curl -sI "https://$ZONE/s13/landing" -H "Cookie: ab_group=B" | grep cf-cache-status
# HIT (쿠키 무관, 같은 캐시) ← 문제! A 그룹 콘텐츠가 B 그룹에 서빙
```

Step 2: Cookie 기반 Cache Key 분기 설정

```
Cache Rule: URI Path = /s13/landing
Cache eligibility: Eligible for cache
Cache Key > Cookie:
  Include cookie names and values: ab_group
```

```
# A 그룹 → 별도 캐시
curl -sI "https://$ZONE/s13/landing" -H "Cookie: ab_group=A" | grep cf-cache-status
# MISS (A 그룹 캐시 없음)

# B 그룹 → 또 다른 캐시
curl -sI "https://$ZONE/s13/landing" -H "Cookie: ab_group=B" | grep cf-cache-status
# MISS (B 그룹 별도 캐시 키)

# A 그룹 재요청 → HIT
curl -sI "https://$ZONE/s13/landing" -H "Cookie: ab_group=A" | grep cf-cache-status
# HIT ✅
```

Step 3: 실제 콘텐츠 다른지 확인

```
# A 그룹 콘텐츠 확인 (오렌지 배경 HTML)
curl -s "https://$ZONE/s13/landing" -H "Cookie: ab_group=A" | grep -i "color\|group"

# B 그룹 콘텐츠 확인 (파란 배경 HTML)
curl -s "https://$ZONE/s13/landing" -H "Cookie: ab_group=B" | grep -i "color\|group"
```

Step 3: Cookie 존재 여부만 체크

```
Cache Key > Cookie:
  Check presence of: session
```

```
# 쿠키 있는 요청 (로그인 사용자)
curl -sI "https://<zone>/dashboard" -H "Cookie: session=abc123" | grep cf-cache-status

# 쿠키 없는 요청 (비로그인)
curl -sI "https://<zone>/dashboard" | grep cf-cache-status
# 두 경우 별도 캐시 키 (session 쿠키 존재 여부로 분기)
```

주의사항

> 쿠키를 Cache Key에 포함하면 캐시가 분산(sharding)되어 히트율이 떨어질 수 있음. 세션 쿠키 전체를 포함하면 사용자별 개별 캐시 → 사실상 캐시 효과 없음. `Check presence of`(존재 여부만)를 활용하여 최소 분기 권장.

Docs

Scenario 14: Custom Cache Key — User (Device Type, Country, Language)

목적

기기 타입, 국가, 언어에 따라 다른 콘텐츠를 캐시한다.

실습 단계

Step 1: Cache by Device Type

```
Rule: URI Path contains /s14/
Cache Key > User: Device Type ON
```

```
# 모바일 User-Agent
curl -sI "https://<zone>/s14/content" \
  -H "User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 14_0 like Mac OS X)" | grep cf-cache-
      status

# 데스크톱 User-Agent
curl -sI "https://<zone>/s14/content" \
  -H "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)" | grep cf-cache-status
# 각각 별도 캐시
```

Step 2: Cache by Country

```
Rule: URI Path contains /s14/
Cache Key > User: Country ON
```

```
# 실제 국가별 요청은 IP 위치로 결정 → CF-IPCountry 헤더 확인
curl -sI "https://<zone>/s14/content" | grep -i "cf-ipcountry\|cf-cache-status"
```

Step 3: Cache by Language

```
Rule: URI Path contains /content/
Cache Key > User: Language ON
```

```
# 한국어
curl -sI "https://<zone>/s14/content" -H "Accept-Language: ko-KR,ko;q=0.9" | grep cf-cache-
        status

# 영어
curl -sI "https://<zone>/s14/content" -H "Accept-Language: en-US,en;q=0.9" | grep cf-cache-
        status
# 별도 캐시
```

Docs

Scenario 15: Cache Deception Armor

목적

Cache Deception 공격을 방지하는 설정을 활성화하고 동작을 확인한다.

배경 지식

Cache Deception 공격 원리: 1. 공격자가 `https://bank.com/account/secret.css` URL 피해자에게 공유 2. CF가 `.css` 확장자 → 캐시 대상으로 인식 3. 피해자가 접속 → 개인정보 포함 응답이 CF에 캐시 4. 공격자가 같은 URL 요청 → 피해자 응답 획득

Cache Deception Armor 동작: - URL의 확장자 타입과 `Content-Type` 헤더를 비교 - 불일치 시 (`Content-Type: text/html` + `.css` URL) → 캐시 거부

서버 엔드포인트

경로	Content-Type 반환	URL 확장자	취약 여부
`/user/profile.css`	`text/html`	.css	⚠️ 불일치 (취약)
`/account/data.js`	`text/html`	.js	⚠️ 불일치 (취약)
`/user/avatar.png`	`text/html`	.png	⚠️ 불일치 (취약)
`/static/style.css`	`text/css`	.css	✅ 일치 (정상)

실습 단계

Step 1: 취약 상태 확인

```
ZONE="cache.<student-zone>"
# /user/profile.css → origin이 HTML 반환 (Content-Type: text/html)
curl -sI "https://$ZONE/user/profile.css" | grep -i "cf-cache-status\|content-type"
# cf-cache-status: MISS→HIT (위험! HTML이 CSS URL로 캐시됨)
# Content-Type: text/html
```

Step 2: Cache Deception Armor 활성화

```
Rule: URI Path ends with .css (or: all cache eligible requests)
Cache Key > Cache Deception Armor: ON
```

```
# 재확인
curl -sI "https://<zone>/user/profile.css" | grep -i "cf-cache-status\|content-type"
# cf-cache-status: MISS or BYPASS (Content-Type 불일치 → 캐시 거부)
```

학습 포인트

> Cache Deception Armor를 활성화하면 URL 확장자와 실제 Content-Type이 일치할 때만 캐시됨. 개인정보가 포함된 응답이 잘못 캐시되는 것을 방지하는 필수 보안 설정.

Docs

Scenario 16: Bypass Cache on Cookie

목적

특정 쿠키가 있는 요청은 캐시를 우회하도록 설정한다.

사용 케이스

- 로그인한 사용자 (session 쿠키 존재) → 개인화 콘텐츠 → 캐시 우회
- WordPress 관리자 (`wordpress_logged_in_*` 쿠키) → 캐시 우회

실습 단계

Step 1: Cache Rule 설정

```
Rule name: Bypass Cache for Logged-in Users
When: Cookie contains "session" or "wordpress_logged_in_"
Then: Cache eligibility → Bypass cache
```

또는 Expression:

```
http.cookie contains "session" or http.cookie contains "wordpress_logged_in_"
```

Step 2: 쿠키 없는 요청 (캐시됨)

```
curl -sI "https://$ZONE/s16/page" | grep cf-cache-status
# HIT
```

Step 3: 쿠키 있는 요청 (캐시 우회)

```
curl -sI "https://$ZONE/s16/page" -H "Cookie: session=user123" | grep cf-cache-status
# DYNAMIC (bypass)
```

Docs

Part IV — Purge (캐시 제거)

Scenario 17: Purge — 단일 파일 (Single File)

목적

특정 URL의 캐시를 즉시 제거한다.

실습 단계

Step 1: 캐시 확인

```
curl -sI "https://<zone>/static/style.css" | grep cf-cache-status
# HIT
```

Step 2: 대시보드에서 Purge

```
Caching > Configuration > Purge Cache > Purge by URL
```

```
URL: https://<zone>/static/style.css
```

Step 3: Purge 후 확인

```
curl -sI "https://<zone>/static/style.css" | grep cf-cache-status
# MISS (캐시 제거됨)
```

Step 4: API로 Purge

```
curl -X POST "https://api.cloudflare.com/client/v4/zones/<ZONE_ID>/purge_cache" \
  -H "Authorization: Bearer <TOKEN>" \
  -H "Content-Type: application/json" \
  --data '{"files": ["https://<zone>/static/style.css"]}'
```

Docs

Scenario 18: Purge — Prefix (URL 경로 접두사)

목적

특정 경로 아래의 모든 캐시를 한 번에 제거한다.

요구 사항

- Business 이상 Plan 또는 Enterprise

실습 단계

```
# /s03/ 하위 모든 캐시 Purge
curl -X POST "https://api.cloudflare.com/client/v4/zones/<ZONE_ID>/purge_cache" \
  -H "Authorization: Bearer <TOKEN>" \
  -H "Content-Type: application/json" \
  --data '{"prefixes": ["<zone>/s03/"]}'
```

확인:

```
curl -sI "https://<zone>/s03/page" | grep cf-cache-status
# MISS
curl -sI "https://<zone>/s03/page" | grep cf-cache-status
# MISS
```

Docs

Scenario 19: Purge — Cache Tags

목적

Cache-Tag 헤더를 통해 그룹핑된 콘텐츠를 태그 기반으로 일괄 Purge한다.

요구 사항

- Enterprise Plan
- Origin이 Cache-Tag 응답 헤더를 반환해야 함 ← **서버가 이미 지원**

서버 엔드포인트

경로	Cache-Tag 헤더
/s19/product/123	product-123, category-merchandise, all-products
/s19/product/456	product-456, category-merchandise, all-products
/s19/product/789	product-789, category-books, all-products
/s19/category/merchandise	category-merchandise, all-products
/s19/category/books	category-books, all-products

실습 단계

Step 1: Cache-Tag 헤더 확인

```
ZONE="cache.<student-zone>"
curl -sI "https://$ZONE/s19/product/123" | grep -i "cache-tag\|cache-control"
# Cache-Tag: product-123, category-merchandise, all-products
# Cache-Control: public, max-age=3600
```

Step 2: 여러 상품 캐시

```
# 상품 3개 모두 캐시
curl -sI "https://$ZONE/s19/product/123" | grep cf-cache-status  # MISS→HIT
curl -sI "https://$ZONE/s19/product/456" | grep cf-cache-status  # MISS→HIT
curl -sI "https://$ZONE/s19/category/merchandise" | grep cf-cache-status  # MISS→HIT
```

Step 3: 태그 기반 Purge (product-123만)

```
curl -X POST "https://api.cloudflare.com/client/v4/zones/<ZONE_ID>/purge_cache" \
  -H "Authorization: Bearer <TOKEN>" \
  -H "Content-Type: application/json" \
  --data '{"tags": ["product-123"]}'
```

Step 4: 선택적 Purge 확인

```
curl -sI "https://$ZONE/s19/product/123" | grep cf-cache-status
# MISS (product-123 태그 → Purge됨)

curl -sI "https://$ZONE/s19/product/456" | grep cf-cache-status
# HIT (product-456은 영향 없음 ← 다른 태그)

curl -sI "https://$ZONE/s19/category/merchandise" | grep cf-cache-status
# MISS (category-merchandise 태그도 123과 공유 → Purge됨)
```

활용 시나리오

> 상품 정보 업데이트 → 해당 상품 ID 태그 Purge → 상품 관련 모든 페이지 자동 갱신

Docs

Scenario 20: Purge — Custom Cache Key로 Purge

목적

Custom Cache Key를 사용하는 리소스를 정확히 Purge한다.

배경 지식

Custom Cache Key 사용 시 단순 URL Purge로는 해당 캐시를 찾을 수 없음. Cache Key에 포함된 헤더/쿠키 값을 명시해야 함.

실습 단계

Step 1: Custom Cache Key 설정 (헤더 포함)

```
Rule: URI Path = /s14/content
Cache Key > Header: X-Language (include value)
```

```
# 캐시 저장
curl -sI "https://<zone>/s14/content" -H "X-Language: ko" | grep cf-cache-status
# MISS → HIT
```

Step 2: Custom Cache Key로 Purge

```
curl -X POST "https://api.cloudflare.com/client/v4/zones/<ZONE_ID>/purge_cache" \
  -H "Authorization: Bearer <TOKEN>" \
  -H "Content-Type: application/json" \
  --data '{
    "files": [{
      "url": "https://<zone>/s14/content",
      "headers": {"X-Language": "ko"}
    }]
  }'
```

Step 3: 다른 Cache Key 변형은 영향받지 않는지 확인

```
curl -sI "https://<zone>/s14/content" -H "X-Language: en" | grep cf-cache-status
# HIT (en 캐시는 Purge 안 됨)

curl -sI "https://<zone>/s14/content" -H "X-Language: ko" | grep cf-cache-status
# MISS (ko 캐시만 Purge됨)
```

Docs

Scenario 21: Purge Everything

목적

Zone 전체 캐시를 한 번에 제거한다.

주의

- **매우 강력한 작업**: 모든 캐시가 제거되어 origin에 급격한 트래픽 유입 가능
- 운영 환경에서는 신중하게 사용

실습 단계

Step 1: 현재 캐시 상태 확인

```
curl -sI "https://<zone>/static/style.css" | grep cf-cache-status
# HIT
curl -sI "https://<zone>/static/app.js" | grep cf-cache-status
# HIT
```

Step 2: Purge Everything

대시보드: `Caching > Configuration > Purge Cache > Purge Everything`

Step 3: 전체 MISS 확인

```
curl -sI "https://<zone>/static/style.css" | grep cf-cache-status
# MISS
curl -sI "https://<zone>/static/app.js" | grep cf-cache-status
# MISS
```

Docs

Part V — 고급 캐시 기능

Scenario 22: Tiered Cache

목적

Tiered Cache를 활성화하여 origin 요청을 최소화하는 효과를 실험한다.

배경 지식

```
[사용자] → [CF 로컬 PoP] → (미스) → [CF 상위 PoP (Tier)] → (미스) → [Origin]
```

- 전 세계 330+ PoP 각각이 origin에 직접 요청하는 대신
- 상위 계층 PoP에서 캐시를 가져옴
- Origin 요청 수를 크게 감소

실습 단계

Step 1: Cache Analytics 기준값 기록

`Caching > Cache Analytics`에서: - Cache Hit Rate 기록 - Origin requests/min 기록

Step 2: Tiered Cache 활성화

`Caching > Tiered Cache > Smart Tiered Cache Topology → Enable`

Step 3: 다양한 지역에서 요청 시뮬레이션

```
# 한국, 미국, 유럽 등 다양한 CF-RAY 데이터센터 확인
for i in $(seq 1 10); do
  curl -sI "https://<zone>/large/1mb" | grep -i "cf-ray\|cf-cache-status"
done
```

Step 4: Analytics 비교

- Tiered Cache 적용 후 origin requests 감소 확인
- Cache Hit Rate 개선 확인

학습 포인트

> Smart Tiered Cache: CF가 자동으로 최적의 상위 PoP 선택 Custom Tiered Cache: 직접 상위/하위 PoP 지정 (Enterprise)

Docs

Scenario 23: Cache Reserve

목적

Cache Reserve로 대용량 파일을 R2에 영구 저장하여 캐시 eviction을 방지한다.

배경 지식

- 일반 CF 캐시: LRU 기반 eviction, 자주 요청되지 않는 파일은 제거
- Cache Reserve: R2(영구 스토리지)에 저장 → eviction 없음
- 활용: 대용량 동영상, PDF, 소프트웨어 배포 파일 등

요구 사항

- Cache Reserve 별도 구독 (R2 기반 과금)

실습 단계

Step 1: Cache Reserve 활성화

`Caching > Cache Reserve → Enable`

Step 2: Cache Rule로 Cache Reserve 대상 지정

```
Rule name: Cache Reserve – Large Files
When: File extension is in [mp4, zip, dmg, iso, gz]
      OR Response size > 50MB (불가능 → 대신 경로 기반)
Cache eligibility: Eligible
Cache Reserve Eligibility: ON
  Minimum file size: 10 MB (10,000,000 bytes)
```

Step 3: 대용량 파일 캐시 확인

```
curl -sI "https://<zone>/s23/installer.zip" | grep -i "cf-cache-status\|cf-edge"
# cf-cache-status: HIT
# (Cache Reserve에 저장된 경우 cf-cache-reserve 헤더 확인)
```

Step 4: Cache Analytics에서 Cache Reserve 히트 확인

`Caching > Cache Analytics > Cache Reserve hits` 항목 확인

Docs

Scenario 24: Query String Sort

목적

쿼리 파라미터 순서에 무관하게 같은 캐시를 사용하여 캐시 히트율을 높인다.

실습 단계

Step 1: 파라미터 순서에 따른 기본 분기 확인

```
curl -sI "https://<zone>/s12/items?b=2&a=1" | grep cf-cache-status
# MISS

curl -sI "https://<zone>/s12/items?a=1&b=2" | grep cf-cache-status
# MISS (파라미터 순서가 달라 다른 캐시 키)
```

Step 2: Query String Sort 활성화

방법 A: Cache Rule에서 설정

```
Cache Key > Sort query string: ON
```

방법 B: Zone 전체에 적용 `Caching > Configuration > Query String Sort → ON`

Step 3: 적용 후 확인

```
curl -sI "https://<zone>/s12/items?b=2&a=1" | grep cf-cache-status
# MISS (첫 요청)

curl -sI "https://<zone>/s12/items?a=1&b=2" | grep cf-cache-status
# HIT (내부적으로 ?a=1&b=2로 정렬되어 동일 캐시 키)
```

Docs

Scenario 25: CDN-Cache-Control 헤더

목적

`CDN-Cache-Control` 헤더로 CF Edge TTL만 독립적으로 제어한다.

서버 엔드포인트

경로	반환 헤더
`/s25/api/catalog`	`Cache-Control: public, max-age=60` + `CDN-Cache-Control: max-age=86400`
`/s25/api/catalog?edge=3600&browser=30`	쿼리 파라미터로 TTL 커스터마이징

배경 지식

- 표준 `Cache-Control` 헤더: 브라우저와 CDN 모두에 적용

- `CDN-Cache-Control`: **CDN만**을 위한 별도 TTL 지시어 (`Cloudflare-CDN-Cache-Control`도 지원)
- 브라우저에는 `Cache-Control`이 그대로 전달됨

```
Cache-Control: public, max-age=60      ← 브라우저 60초 캐시
CDN-Cache-Control: max-age=86400       ← CF Edge는 86400초 캐시
```

실습 단계

Step 1: origin이 두 헤더 모두 반환하는지 확인

```
ZONE="cache.<student-zone>"
curl -sI "https://$ZONE/s25/api/catalog" | grep -iE "cache-control|cdn-cache"
# Cache-Control: public, max-age=60
# CDN-Cache-Control: max-age=86400
```

Step 2: CF에서 각각 다른 TTL 적용 확인

```
curl -sI "https://$ZONE/s25/api/catalog" | grep -i "cache-control\|age\|cf-cache-status"
# Cache-Control: public, max-age=60      (브라우저용, 변경 없이 전달)
# Age: <증가하는 값>                     (CF는 86400초 기준으로 캐시)
# cf-cache-status: HIT
```

Step 3: 쿼리 파라미터로 TTL 변경 테스트

```
# Edge 3600초, Browser 30초
curl -sI "https://$ZONE/s25/api/catalog?edge=3600&browser=30" | grep -iE "cache-control|cdn-
        cache"
```

Step 4: 86400초 초과 전 HIT 유지, 60초 후 브라우저 재요청 유도

활용 케이스

> API 응답: 브라우저는 항상 최신 확인 (짧은 CC), CF는 길게 캐시 (트래픽 절감)

Docs

Scenario 26: Always Online

목적

Origin 서버가 다운됐을 때 CF가 캐시된 버전을 서빙하는 동작을 확인한다.

서버 엔드포인트

경로	동작
`/s26/down?always=1`	**항상 503** 반환 (완전 다운 시뮬레이션)
`/s26/down`	50% 확률 503 (간헐적 장애)
`/s26/recovery`	정상 응답 (서버 복구 확인용)

배경 지식

- Always Online ON: origin 5xx 응답 시 캐시된 페이지 제공
- 표시: `cf-cache-status: STALE` + `stale-while-revalidate` 동작
- Internet Archive 연동으로 CF에 없는 페이지도 일부 제공

실습 단계

Step 1: Always Online 활성화 + 사전 캐시

`Caching > Configuration > Always Online → ON`

```
ZONE="cache.<student-zone>"
# Cache Rule 적용: URI Path = /s26/recovery, Eligible, TTL=300s
curl -sI "https://$ZONE/s26/recovery" | grep cf-cache-status
# MISS → HIT (캐시됨)
```

Step 2: Origin 강제 다운 시뮬레이션

```
# 이 경로는 항상 503 반환 (실제 origin 다운 시뮬레이션)
curl -sI "https://$ZONE/s26/down?always=1" | grep -iE "^HTTP|cf-cache-status"
```

> **이전에 `/s26/recovery`를 캐시했다면**: Cache Rule에서 두 경로를 같은 경로 prefix로 설정하면 Always Online 테스트 가능. 실제로는 `/s26/recovery`를 사전 캐시한 뒤 origin 자체를 내리는 방식이 가장 정확한 테스트.

Step 3: Always Online 서빙 확인

```
# recovery 경로가 캐시되어 있고 origin이 503인 상황
curl -sI "https://$ZONE/s26/recovery" | grep -i "cf-cache-status"
# cf-cache-status: STALE (Always Online이 캐시된 버전 서빙)
```

Docs

Scenario 27: Development Mode

목적

개발/테스트 중 캐시를 일시적으로 전체 우회한다.

배경 지식

- Development Mode: 3시간 동안 모든 캐시 무력화
- 새 콘텐츠를 배포하고 즉시 효과 확인 시 사용
- 기간 후 자동 비활성화

실습 단계

Step 1: Development Mode 활성화

`Caching > Configuration > Development Mode → ON`

Step 2: 모든 요청이 origin으로 전달되는지 확인

```
curl -sI "https://<zone>/static/style.css" | grep cf-cache-status
# BYPASS 또는 MISS (개발 모드 중 캐시 우회)
```

Step 3: origin 응답 실시간 확인

```
# 캐시 없이 항상 최신 origin 응답 확인
curl -sv "https://<zone>/static/style.css" 2>&1 | grep "< "
```

Step 4: 개발 모드 비활성화 후 정상 캐시 복귀

```
curl -sI "https://<zone>/static/style.css" | grep cf-cache-status
# MISS → HIT (캐시 재저장)
```

Docs

Scenario 28: Early Hints (103)

목적

103 Early Hints를 통해 브라우저가 Critical Resource를 미리 로드하도록 한다.

배경 지식

- HTTP 103 Early Hints: CF가 origin 응답 도착 전 `Link: preload` 헤더 미리 전송
- CSS, JS, Font 등 Critical Resource를 브라우저가 선제적으로 요청
- 체감 성능(LCP, FCP) 개선

실습 단계

Step 1: Early Hints 활성화

`Speed > Optimization > Early Hints → ON`

Step 2: origin에서 Link 헤더 반환

```
Link: </static/style.css>; rel=preload; as=style,
      </static/app.js>; rel=preload; as=script
```

Step 3: 103 응답 확인 (Chrome DevTools)

브라우저 DevTools → Network → Response Headers에서 `103` 응답 확인

Step 4: curl로 확인 (HTTP/2 필요)

```
curl -v --http2 "https://<zone>/" 2>&1 | grep -A5 "103"
```

Docs

Scenario 29: Crawler Hints (IndexNow)

목적

콘텐츠 업데이트 시 검색 엔진 크롤러에게 IndexNow 프로토콜로 즉시 알린다.

배경 지식

- IndexNow: 검색 엔진(Bing, Yandex 등)에 콘텐츠 변경 즉시 통보
- Cloudflare Crawler Hints: Cache Purge 발생 시 자동으로 IndexNow 신호 전송

- 검색 색인 업데이트 속도 개선

실습 단계

Step 1: Crawler Hints 활성화

`Caching > Configuration → Crawler Hints: ON`

Step 2: 콘텐츠 Purge 실행

```
# 단일 파일 Purge (Crawler Hints가 검색엔진에 알림)
curl -X POST "https://api.cloudflare.com/client/v4/zones/<ZONE_ID>/purge_cache" \
  -H "Authorization: Bearer <TOKEN>" \
  --data '{"files": ["https://<zone>/s29/blog/post"]}'
```

Step 3: Bing Webmaster Tools에서 IndexNow 수신 확인

- `https://www.bing.com/webmasters/indexnow` 에서 제출 기록 확인

Docs

Scenario 30: Vary for Images

목적

브라우저가 지원하는 이미지 포맷에 따라 최적의 포맷(WebP, AVIF)을 서빙하고 각각 별도로 캐시한다.

서버 엔드포인트

경로	동작
`/s30/hero.jpg`	`Accept` 헤더에 따라 다른 `Content-Type` 반환. `Vary: Accept` 헤더 포함.
`/s30/info`	현재 요청의 Accept 헤더 디버깅 페이지

```
# 서버가 반환하는 Content-Type 확인 (CF 없이 직접 origin 확인)
curl -sI https://cache.cflarekr.dev/s30/hero.jpg \
  -H "Accept: image/webp" | grep -i "content-type\|vary\|x-image"
# Content-Type: image/webp
# Vary: Accept
# X-Image-Format: webp
```

배경 지식

- `Vary: Accept` 헤더: 브라우저 Accept 헤더에 따라 다른 버전 캐시
- CF는 `Accept: image/webp` 지원 브라우저에게 WebP 서빙
- Vary for Images: 자동으로 포맷별 캐시 버전 관리

요구 사항

- Polish (이미지 최적화) 활성화 필요 (없으면 서버의 Vary 헤더만 확인)

실습 단계

Step 1: Vary for Images 활성화

```
Speed > Optimization > Polish → Lossy or Lossless
Speed > Optimization → Vary for Images: ON
```

Step 2: origin 응답 포맷 확인

```
ZONE="cache.<student-zone>"

# WebP 지원 요청
curl -sI "https://$ZONE/s30/hero.jpg" \
  -H "Accept: image/webp,image/*,*/*;q=0.8" | grep -iE "content-type|x-image-format|vary"
# Content-Type: image/webp
# X-Image-Format: webp
# Vary: Accept
```

Step 3: JPEG only 요청

```
curl -sI "https://$ZONE/s30/hero.jpg" \
  -H "Accept: image/jpeg,image/*;q=0.8" | grep -iE "content-type|x-image-format|cf-cache-
      status"
# Content-Type: image/jpeg
# X-Image-Format: jpeg
# cf-cache-status: MISS (Accept가 달라 별도 캐시 키)
```

Step 4: HIT 확인

```
# 두 번째 요청 (같은 Accept 헤더) → HIT
curl -sI "https://$ZONE/s30/hero.jpg" \
  -H "Accept: image/webp,image/*,*/*;q=0.8" | grep cf-cache-status
# HIT
```

Step 5: Purge Varied Images

```
# WebP 버전만 Purge
curl -X POST "https://api.cloudflare.com/client/v4/zones/<ZONE_ID>/purge_cache" \
  -H "Authorization: Bearer <TOKEN>" \
  -H "Content-Type: application/json" \
  --data '{"files": [{"url": "https://<student-zone>/s30/hero.jpg",
                      "headers": {"Accept": "image/webp"}}]}'
# JPEG 버전은 영향 없음 (별도 캐시 키)
```

Docs

Part VI — Revalidation & ETag

Scenario 31: Cache Revalidation 흐름 이해

목적

Conditional Request(조건부 요청)를 통한 캐시 재검증 과정 전체를 추적한다.

서버 엔드포인트

경로	동작
`/s31/resource.js`	Strong ETag + `Cache-Control: public, max-age=10` (짧은 TTL)
	If-None-Match 요청 시 → **304 Not Modified** 자동 응답
`/s09/resource.js`	동일 동작 (60초 TTL 버전)

배경 지식

```
브라우저 → CF 캐시 → origin

TTL 만료 전: CF가 캐시에서 바로 응답 (HIT)
TTL 만료 후: CF가 origin에 조건부 요청
  → origin 미변경: 304 Not Modified → CF 캐시 갱신 (REVALIDATED)
  → origin 변경됨: 200 + 새 콘텐츠 → 새 캐시 저장 (MISS/EXPIRED)
```

실습 단계

Step 1: 짧은 TTL 리소스 캐시 설정

```
Cache Rule: URI Path = /s31/resource.js
Edge TTL: respect_origin (origin이 max-age=10 반환)
Respect Strong ETags: ON
```

Step 2: 초기 캐시 확인

```
ZONE="cache.<student-zone>"
curl -sI "https://$ZONE/s31/resource.js" | grep -i "cf-cache-status\|etag\|age"
# cf-cache-status: MISS
# ETag: "f1418e1e8c06cc233cd2ce40fd9340ca" (Strong ETag)
```

Step 3: HIT 상태 확인

```
curl -sI "https://$ZONE/s31/resource.js" | grep -i "cf-cache-status\|age"
# cf-cache-status: HIT
# age: 3
```

Step 4: TTL 만료 후 재검증 관찰 (10초 후)

```
sleep 12
curl -sI "https://$ZONE/s31/resource.js" | grep cf-cache-status
# EXPIRED → CF가 origin에 If-None-Match 요청 전송
# origin이 304 → REVALIDATED (origin 내용 미변경 확인)
# age: 0 (새로 시작)
```

Step 5: Wireshark / CF Trace로 실제 요청 확인

```
# CF Trace로 조건부 요청 확인
curl "https://cloudflare.com/cdn-cgi/trace" \
  -H "Host: <zone>" | grep -i "colo\|visit_scheme"
```

Docs

Part VII — 보안 관련 캐시 설정

Scenario 32: Web Cache Poisoning 방지

목적

캐시 포이즈닝 공격 벡터를 이해하고 CF 설정으로 방어한다.

서버 엔드포인트

경로	동작
`/s32/reflect`	`X-Forwarded-Host` 헤더 값을 응답 본문/헤더에 반영 (취약한 origin 시뮬레이션)

배경 지식

공격 원리: 1. 공격자가 `X-Forwarded-Host: evil.com` 헤더로 요청 2. origin(`/s32/reflect`)이 이 헤더를 응답에 반영 3. CF가 이 응답을 캐시 4. 일반 사용자가 악성 콘텐츠 수신

실습 흐름:

```
ZONE="cache.<student-zone>"
# 공격자 요청: X-Forwarded-Host 인젝션
curl -sI "https://$ZONE/s32/reflect" -H "X-Forwarded-Host: evil.com" | grep -i x-reflected
# X-Reflected-Host: evil.com (origin이 반영함)
```

방어 단계

Step 1: Unkeyed Headers 제거

```
Cache Rule > Cache Key:
  Headers: X-Forwarded-Host → Bypass (캐시 키에서 제외하되, origin에 전달 안 함)
```

또는 Transform Rule에서 해당 헤더 제거:

```
Rule: Remove request header X-Forwarded-Host before origin
```

Step 2: URL 정규화 활성화

`Rules > Normalization → Normalize incoming URLs: ON`

Step 3: Cache Deception Armor 확인

Scenario 15 참조. Content-Type과 URL 확장자 불일치 방지.

Docs

Scenario 33: CORS 헤더와 캐시 상호작용

목적

CORS가 있는 리소스를 올바르게 캐시하여 cross-origin 요청이 정상 동작하도록 한다.

서버 엔드포인트

경로	반환 헤더
`/s33/api/data`	`Access-Control-Allow-Origin: *` + `Vary: Origin` + `Cache-Control: public, max-age=60`
`OPTIONS /s33/api/data`	CORS preflight 응답 (204)

배경 지식

- `Origin` 헤더가 다른 요청은 기본 Cache Key에 포함됨 (CORS 지원)
- Cache Key에 Origin을 포함해야 origin별 CORS 응답이 올바르게 캐시됨

실습 단계

Step 1: CORS 리소스 캐시 확인

```
ZONE="cache.<student-zone>"
# Origin A에서 요청
curl -sI "https://$ZONE/s33/api/data" \
  -H "Origin: https://app-a.example.com" | grep -i "access-control\|cf-cache-status\|vary"

# Origin B에서 요청
curl -sI "https://$ZONE/s33/api/data" \
  -H "Origin: https://app-b.example.com" | grep -i "access-control\|cf-cache-status"
```

Step 2: Origin 헤더 포함 검증

CF는 기본적으로 Origin 헤더를 Cache Key에 포함 → 각 Origin별 별도 캐시.

Step 3: Origin 헤더 제거 (단일 캐시로 통합할 경우)

```
Cache Key > Headers:
  Exclude Origin header
```

> 주의: CORS가 있는 경우 Origin 제거는 Cross-Origin 응답이 잘못 캐시될 수 있어 위험.

Docs

Part VIII — Cache Analytics & 성능 측정

Scenario 34: Cache Analytics 대시보드 활용

목적

Cache Analytics에서 의미 있는 데이터를 읽고 성능 개선 포인트를 찾는다.

실습 단계

Step 1: Cache Hit Rate 확인

`Caching > Cache Analytics`

- 전체 Cache Hit Rate 확인
- 목표: **90% 이상**이 이상적

Step 2: 가장 많이 MISS되는 경로 확인

> `Analytics 탭 → 요청 수 많은 MISS 리소스 상위 10개 확인`

분석 포인트: - MISS가 많은 HTML/JSON 경로 → Cache Everything Rule 추가 검토 - DYNAMIC이 많은 경로 → 의도적 미캐시인지 확인

Step 3: Bandwidth Saved 계산

> ```
> 절감 bandwidth = (HIT 요청 수 × 평균 응답 크기)
> 비용 절감 = 절감 bandwidth × origin egress 단가
> ```

Step 4: 시간대별 패턴 분석

- 특정 시간에 Cache Miss 급증 → 캐시 Purge 또는 TTL 만료 패턴
- TTL 조정으로 개선 가능한지 검토

Docs

Scenario 35: Cache Rule 우선순위 충돌 테스트

목적

여러 Cache Rule이 동시에 적용될 때 우선순위를 이해한다.

배경 지식

- Cache Rules는 위에서 아래로 순서대로 실행
- 첫 번째 매칭 Rule이 적용됨 (first match wins)
- `skip remaining rules` 옵션으로 이하 규칙 무시 가능

실습 단계

Step 1: 충돌하는 Rule 2개 생성

```
Rule 1 (상위): URI Path contains /products/
  → Eligible for cache, Edge TTL: 3600s

Rule 2 (하위): URI Path = /products/sale/
  → Bypass cache
```

Step 2: /products/sale/ 요청

```
curl -sI "https://$ZONE/s35/products/sale/" | grep cf-cache-status
# HIT or MISS (Rule 1이 먼저 매칭되어 캐시됨 → Rule 2는 실행 안 됨)
```

Step 3: Rule 순서 변경

Rule 2를 Rule 1 위로 이동

```
curl -sI "https://$ZONE/s35/products/sale/" | grep cf-cache-status
# DYNAMIC (Rule 2가 먼저 매칭 → Bypass)
```

학습 포인트

> 더 구체적인 경로를 가진 Rule을 더 위에 배치해야 의도한 대로 작동.

Docs

부록 A: 전기능 체크리스트

Cache Rules 설정 항목

기능	실습 완료	비고
Cache Eligibility: Bypass	☐	Scenario 4
Cache Eligibility: Eligible	☐	Scenario 3
Edge TTL: respect_origin	☐	Scenario 5
Edge TTL: override_origin	☐	Scenario 5
Edge TTL: bypass_by_default	☐	Scenario 5
Status Code TTL	☐	Scenario 6
Browser TTL	☐	Scenario 7
Serve Stale While Revalidating	☐	Scenario 8
Respect Strong ETags	☐	Scenario 9
Origin Cache Control	☐	Scenario 10
Origin Error Page Pass-through	☐	Scenario 11
Cache Key: Query String Sort	☐	Scenario 12
Cache Key: Ignore Query String	☐	Scenario 12
Cache Key: Include/Exclude QS	☐	Scenario 12
Cache Key: Cookie (Enterprise)	☐	Scenario 13
Cache Key: Header (Enterprise)	☐	Scenario 20
Cache Key: User — Device Type	☐	Scenario 14
Cache Key: User — Country	☐	Scenario 14
Cache Key: User — Language	☐	Scenario 14
Cache Deception Armor	☐	Scenario 15
Cache Reserve Eligibility	☐	Scenario 23

Purge 방법

기능	실습 완료
Purge by single file	☐

기능	실습 완료
Purge by prefix	☐
Purge by hostname	☐
Purge by cache-tags	☐
Purge by custom cache key	☐
Purge everything	☐
Purge varied images	☐

고급 기능

기능	실습 완료
Tiered Cache	☐
Cache Reserve	☐
Query String Sort (zone-level)	☐
Early Hints	☐
Crawler Hints	☐
Always Online	☐
Vary for Images	☐
Development Mode	☐
CDN-Cache-Control	☐
Cache Analytics	☐

부록 B: curl 치트시트

```bash
# 응답 헤더만 (간결)
curl -sI "https://<zone>/path"

# 전체 요청/응답 (verbose)
curl -sv "https://<zone>/path" 2>&1 | grep "^[<>]"

# 특정 헤더만 필터
curl -sI "https://<zone>/path" | grep -i "cf-cache-status\|age\|cache-control\|etag"

# Custom header 추가
curl -sI "https://<zone>/path" -H "Cookie: ab_group=A" -H "Accept-Language: ko"

# 반복 요청 (HIT 전환 대기)
for i in $(seq 1 5); do
  echo -n "[$i] " && curl -sI "https://<zone>/path" | grep -i cf-cache-status
  sleep 1
done

# 쿼리 파라미터 변형 테스트
for qs in "a=1&b=2" "b=2&a=1" "c=3&a=1&b=2"; do
  echo "QS: $qs" && curl -sI "https://<zone>/search?$qs" | grep cf-cache-status
done
```

부록 C: 주요 참고 문서

주제	URL
Cache 전체 개요	https://developers.cloudflare.com/cache/
Default Cache Behavior	https://developers.cloudflare.com/cache/concepts/default-cache-behavior/
Cache Responses (cf-cache-status)	https://developers.cloudflare.com/cache/concepts/cache-responses/
Cache Rules Settings	https://developers.cloudflare.com/cache/how-to/cache-rules/settings/
Cache Keys	https://developers.cloudflare.com/cache/how-to/cache-keys/
Purge	https://developers.cloudflare.com/cache/how-to/purge-cache/
Tiered Cache	https://developers.cloudflare.com/cache/how-to/tiered-cache/
Cache Reserve	https://developers.cloudflare.com/cache/advanced-configuration/cache-reserve/

주제	URL
Cache Analytics	https://developers.cloudflare.com/cache/performance-review/cache-analytics/
Cache Deception Armor	https://developers.cloudflare.com/cache/cache-security/cache-deception-armor/
Status Code TTL	https://developers.cloudflare.com/cache/how-to/configure-cache-status-code/
CDN-Cache-Control	https://developers.cloudflare.com/cache/concepts/cdn-cache-control/
Vary for Images	https://developers.cloudflare.com/cache/advanced-configuration/vary-for-images/
Early Hints	https://developers.cloudflare.com/cache/advanced-configuration/early-hints/