쿠키 vs? 세션 vs 토큰

웹 서비스는 어떻게 우리를 확인하는가?

By CommentLee

HTTP 프로토콜의 문제

무상태성 (Stateless)

HTTP는 각 요청을 독립적으로 처리하여 이전 요청의 상태 정보를 전혀 기억하지 못합니다.

반복 인증 필요

페이지를 이동할 때마다 사용자 인증을 다시 수행해야 하는 비효율성이 발생합니다.

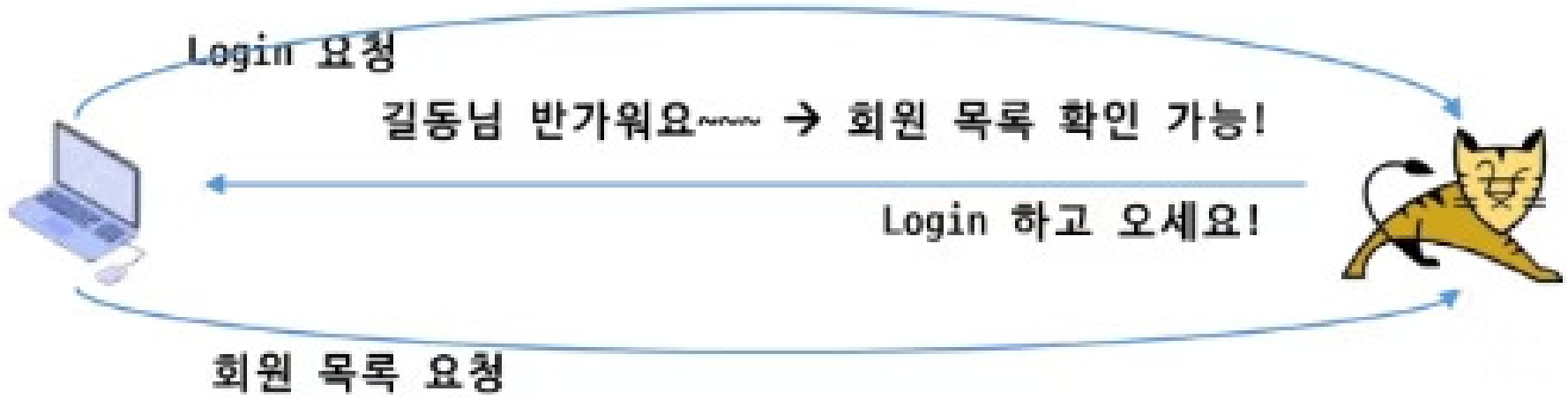
따라서...

사용자를 구분하는

무언가가 필요하다.

❖ HTTP 특징

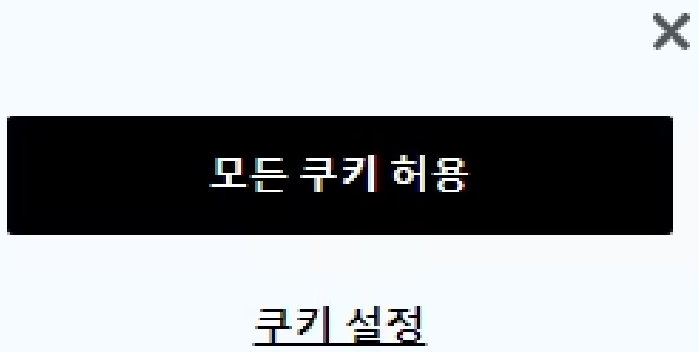
- Stateless: 상태를 기억하지 않음



쿠키 (Cookie)

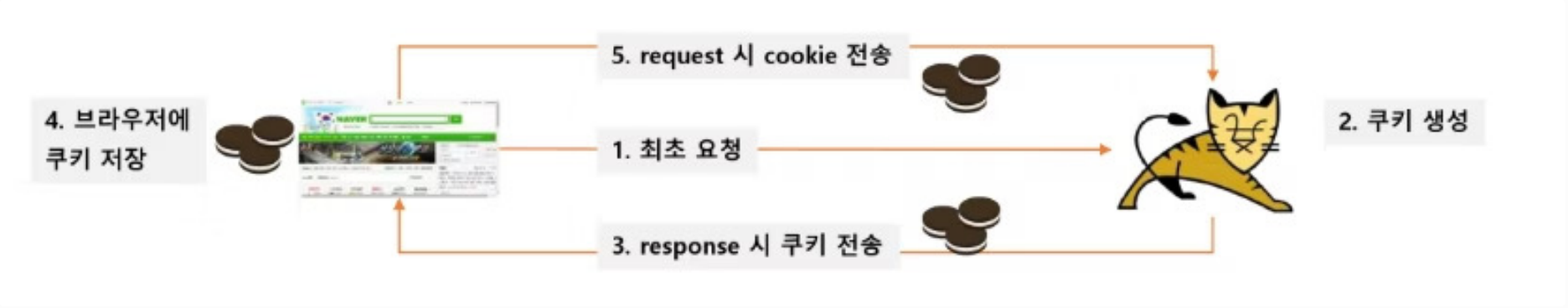
작동 원리

서버가 브라우저에 Key-Value 형태의 작은 데이터를 저장하도록 지시합니다. 이후 동일한 도메인으로의 모든 HTTP 요청에 자동으로 포함되어 전송됩니다.



주요 활용 사례

- 사용자 아이디 저장 기능
- 쇼핑몰 장바구니 상품 보관
- 광고 추적 및 개인화
- 언어 설정, 테마 선택 등



쿠키의 특징

장점

- 구현이 매우 간단하고 직관적
- 브라우저가 자동으로 관리
- 별도의 JavaScript 코드 불필요

단점

- 저장 용량 제한 (도메인당 4KB) (솔직히 단점은아닌듯함)
- 클라이언트 측에서 조작 가능
- 네트워크 탈취 위험성
- 보안성이 상대적으로 취약

나름 보안 기능이 있긴한데..

- secure: HTTPS에서만 전송 허용
- httpOnly: JavaScript에서 접근 불가 설정

httpOnly키면 xss 막을수있다,

sameSite옵션도 키면 CSRF막음

그런데 우선 HTTPS를 쓰고 생각하자

vs 웹 스토리지 (Web Storage)

- `localStorage`: 영구 저장 (삭제 전까지 유지).
- `sessionStorage`: 브라우저 탭/세션 종료 시 사라짐.

세션 (Session)

01

사용자 로그인

서버가 사용자 인증 후 고유한 세션 ID를 생성합니다.

02

서버 저장

사용자 정보와 상태 데이터를 서버 메모리나 데이터베이스에 저장합니다.

03

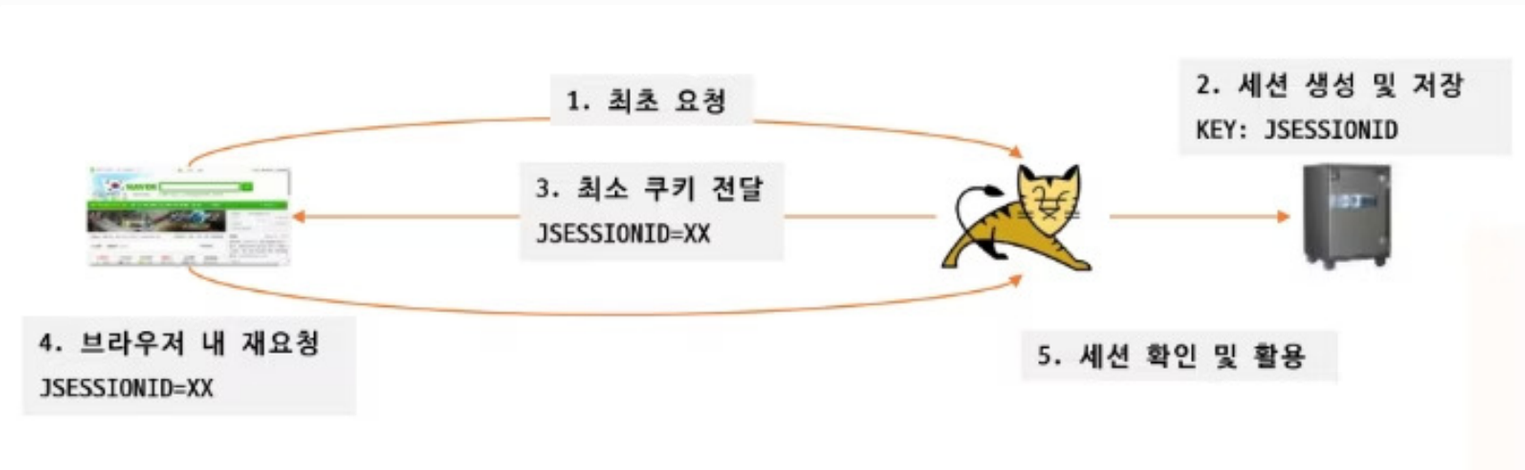
세션 ID 전송

클라이언트는 세션 ID만을 쿠키로 보관하고 요청시 전송합니다.

04

사용자 식별

서버는 세션 ID로 저장된 사용자 정보를 조회하여 인증을 처리합니다.



- '서버 저장'후 세션id로 인증이 중요. 그림처럼 쿠키로 안해도 됨. 근데 굳이?

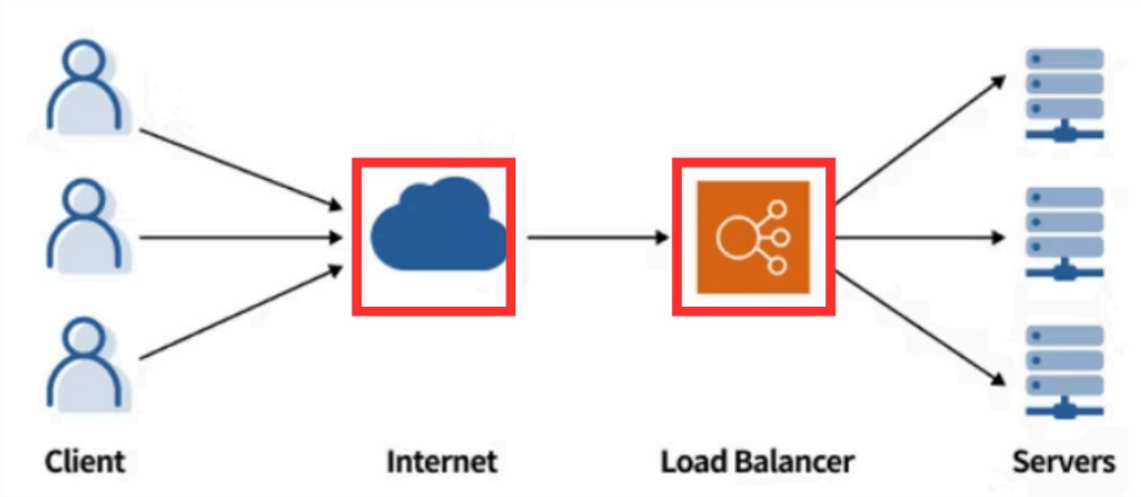
세션의 장단점

장점

민감한 정보가 서버에만 저장되어 비교적 안전하며, 세션 무효화로 즉시 접근 차단이 가능합니다.

단점

서버 메모리 사용량 증가와 분산 환경에서 세션 공유 문제로 확장성에 제약이 있습니다.

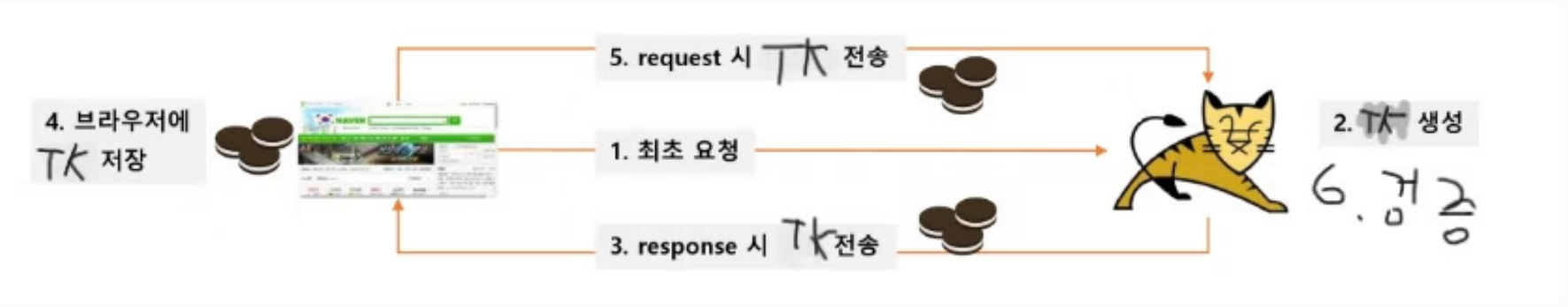


첫번째 서버에 세션이 저장되었다면 두번째요청도?

또는 사람이 매우 많다면?

토큰 (JWT - JSON Web Token)

JWT가 토큰인증 방식의 대부분이라 보통 토큰이야기하면 JWT를 말함.



토큰 발급

서버가 사용자 정보를 암호화하여 JWT 토큰을 생성합니다.

클라이언트 저장

클라이언트가 토큰을 로컬스토리지나 쿠키에 보관합니다.

헤더 전송

요청 시 Authorization 헤더에 토큰을 포함하여 전송합니다. (쿠키에 저장했다면 쿠키에 담겨서 보내겠죠?)

무상태 검증

서버는 토큰 서명만 검증하여 인증을 처리합니다.

아까 그림이랑 똑같은데요?

차이점: 서명

XXXXXX.YYYYYY.ZZZZZZ

헤더(Header) 내용(Payload) 서명(Signature)

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

헤더(Header)	내용(Payload)	서명(Signature)
{ "alg": "HS256", "typ": "JWT" }	{ "sub": "1234567890", "name": "John Doe", "iat": 1516239022 }	HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), your-256-bit-secret)

- 헤더: 알고리즘,토큰유형
- 내용:사용자 정보(이름, 만료시간, ~~비밀번호),~~ 발행시간(iat) 같은거
- 서명: 헤더+내용을 서버의 비밀키와 헤더의 알고리즘으로 암호화한 값

무결성(위변조)만 체크가능.

헤더,내용 모두 **base64**로 인코딩한 **string**

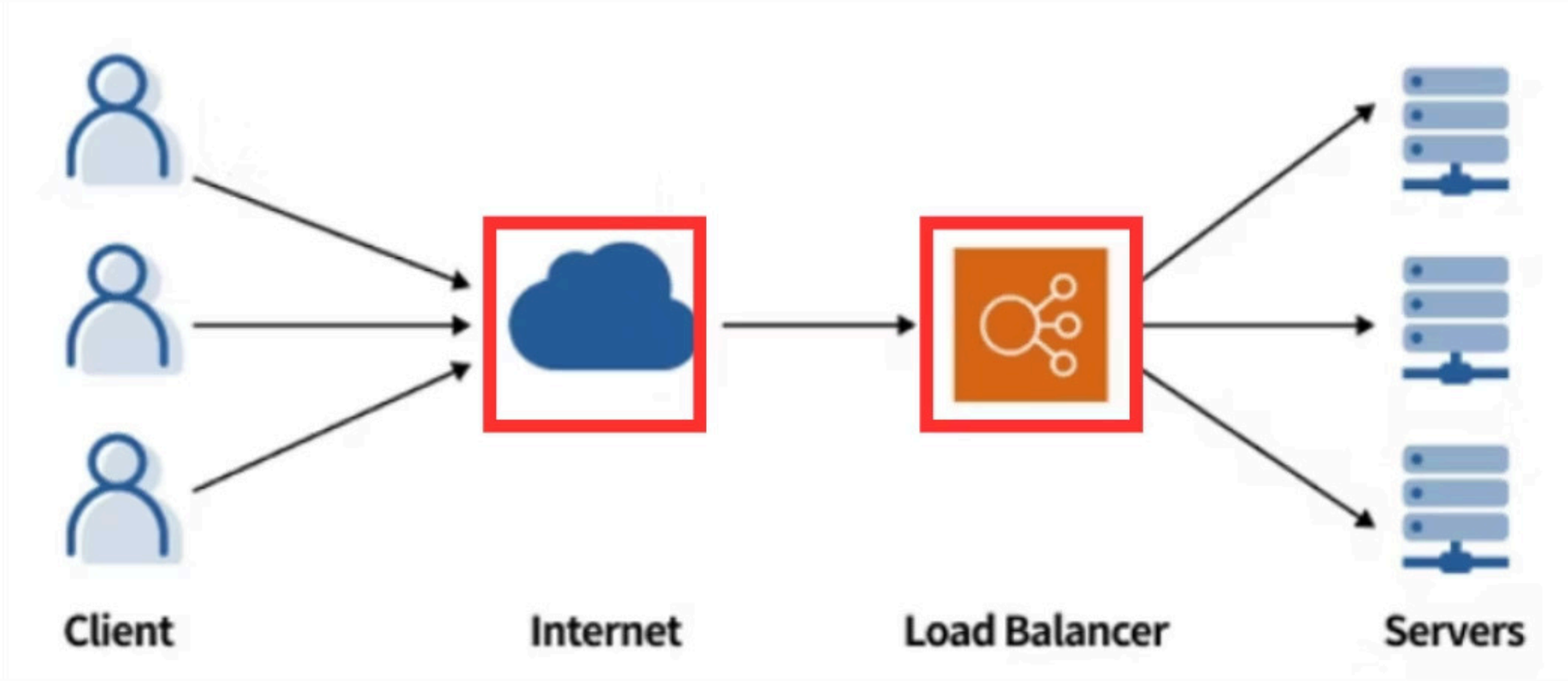
토큰의 장단점

장점

- 분산 환경 적합성(세션에서 규모의 문제점 해결)
- stateless
- 저장소 필요없음 = DB조회를 안해도 됨.
- 다른 로그인 시스템에도 권한 공유 가능(oauth)

단점

- 토큰 탈취 시 보안 위험(사용자가 보관)
- 토큰 갱신과 무효화 관리
- 토큰 크기로 인한 네트워크 오버헤드(필요한것만 넣자)



Refresh Token

Access Token

지금까지 설명한 인증을 위한 토큰을 Access Token이라 부른다.

그런데 이걸 해커가 가져가면 답이 없다.

그래서 Access Token은 만료가 짧다.

Refresh Token

대신 만료기한이 긴 Refresh Token을 준다.

Access Token이 만료되면 Refresh Token을 전달하고 다시 재발급 받는다.

서버는 **Refresh Token** 저장

탈취 당한것같으면 이 토큰 정보를 지워버리면 됨, 또는 블랙리스트

단점?: 결국 세션처럼 동작(그래도 세션보다는 DB 덜 조회)

세션 vs 토큰

세션

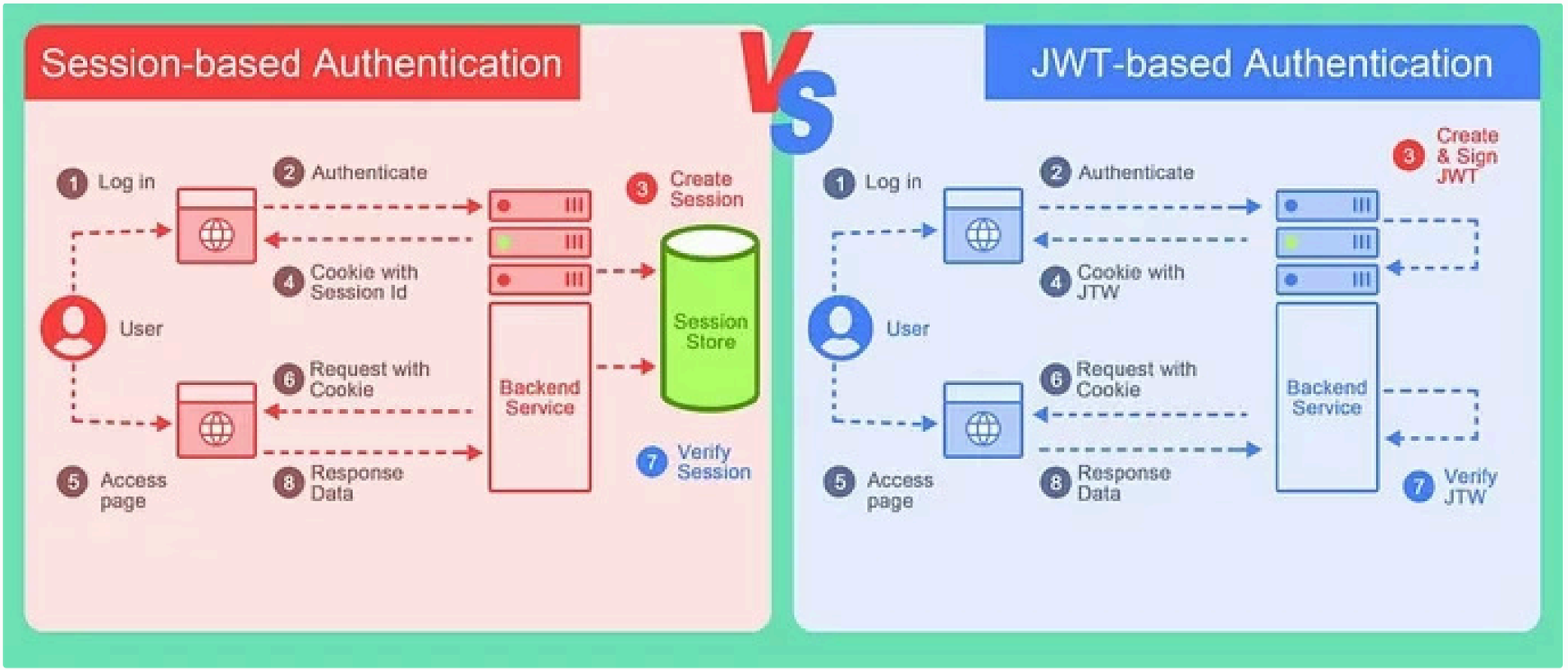
높은 보안성: 서버가 전부 관리, 세션 탈취되어도 지우면 끝

낮은 트래픽: 세션id만 보내면됨

낮은 확장성: scale out은 복잡하다

토큰

- **낮은 보안성**: Access가 탈취되면 만료될때까지 속수무책
- **높은 트래픽:** 토큰은 사용자 인증 정보를 보내느라 더 많음
- **뛰어난 확장성**: 서버 늘려도 동작방식에 차이없음



그래서 뭐 쓰나요? → 상황에 따라 사용하세요