JWT 토큰(Token) 기반 인증

JWT 소개 전에... 토큰이란?

> 소개

API를 사용하는 웹서비스 개발시, 토큰을 사용하여 유저들의 인증작업을 처리하는 것은 중요하다.

> 토큰기반 인증 시스템을 사용하는 이유

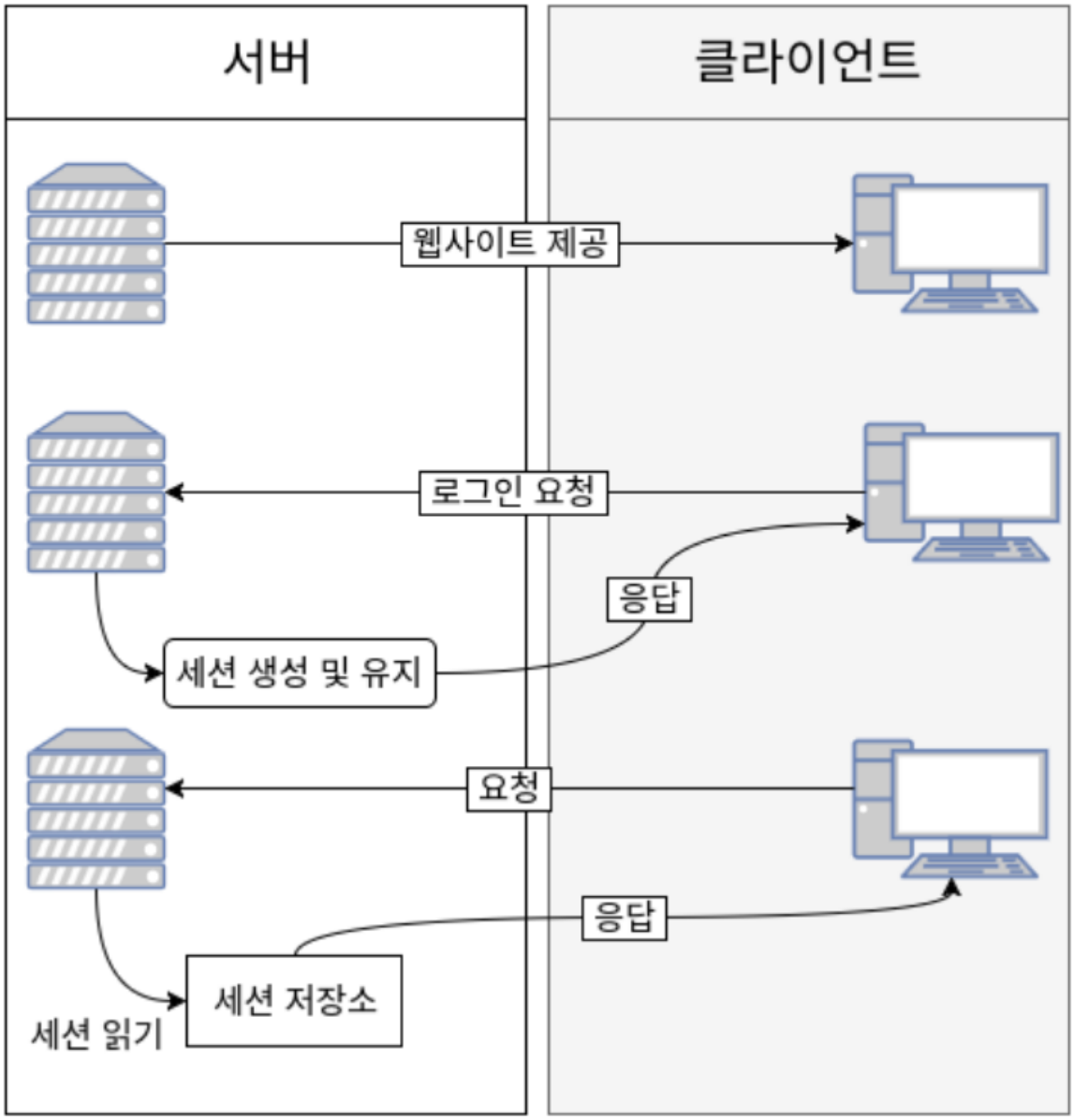
Stateful Server : 클라이언트에게서 요청 받을 때마다, 클라이언트의 상태를 계속해서 유지하고, 이 정보를 서비스 제공에 이용한다.
ex) 세션을 유지하는 웹 서버.

Stateless Server : 상태를 유지하지 않는다. 상태정보를 저장하지 않으면, 서버는 클라이언트측에서 들어오는 요청으로만 작업을 처리한다.

> 왜 토큰을 사용하게 돼었을까?

과거의 인증시스템의 작동방식

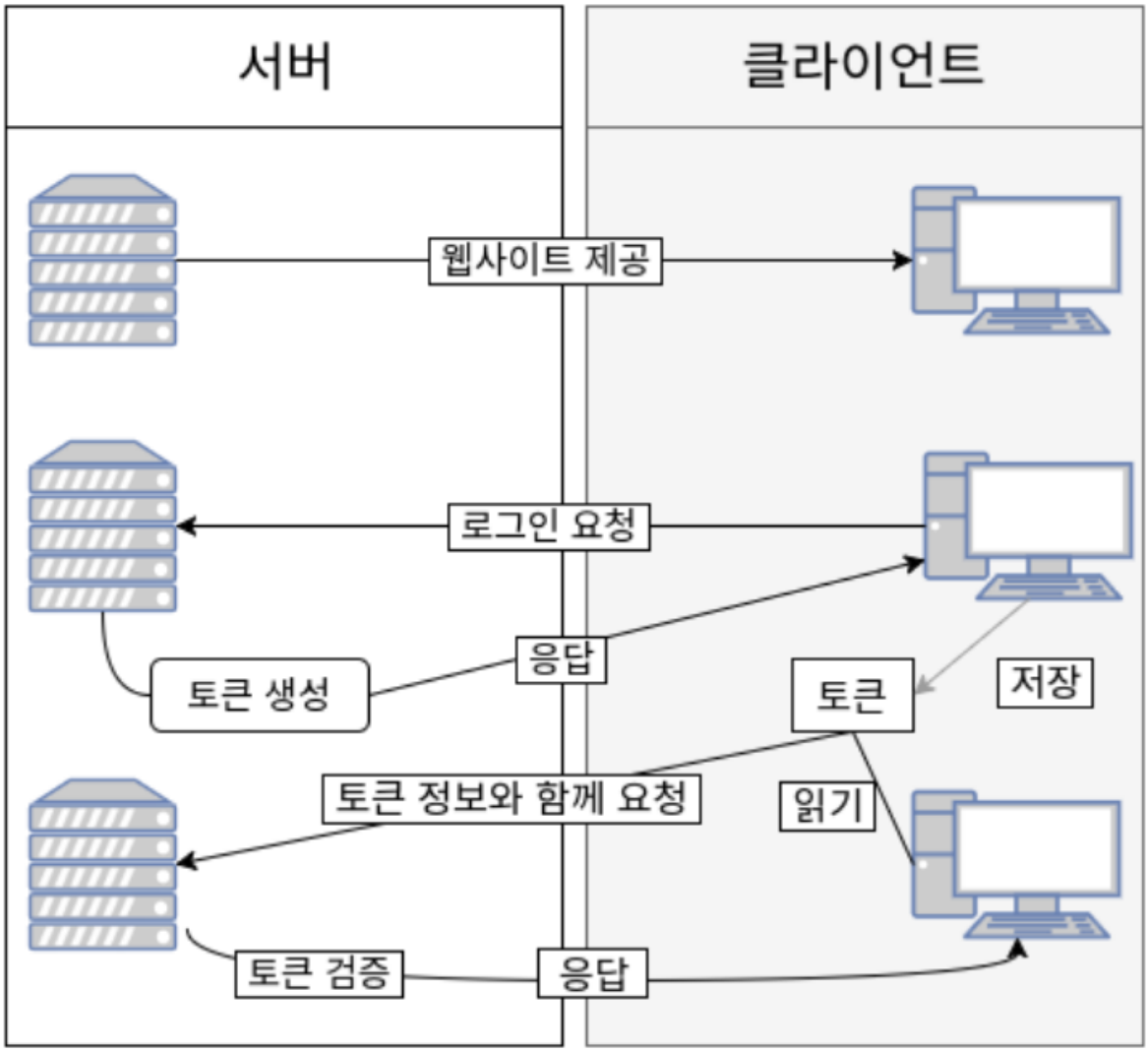
서버 기반 인증 : 서버를 확장하기가 어렵다.



- 유저의 수가 많으면 데이터베이스의 성능에 무리가 된다.
- 확장성 : 세션 사용시 서버를 확장하는 것이 어려워진다.
- CORS(Cross-Origin Resource Sharing) : 쿠키는 단일 도메인 및 서브 도메인에서만 작동하도록 설계되어있으므로 쿠키를 여러 도메인에서 관리하는 것이 번거롭다.

> 토큰 기반 시스템의 작동 원리

- 상태를 유지하지 않기 때문에 더이상 유저의 인증정보를 서버나 세션에 담아두지 않습니다.
- 세션이 존재하지 않기 때문에, 유저들이 로그인 되어있는지 안되어있는지 신경쓰지 않으면서 서버를 손쉽게 확장할 수 있다.



1. 유저가 아이디와 비밀번호로 로그인을 합니다
2. 서버측에서 해당 계정정보를 검증합니다.
3. 계정정보가 정확하다면, 서버측에서 유저에게 signed 토큰을 발급해줍니다. 여기서 signed 의 의미는 해당 토큰이 서버에서 정상적으로 발급된 토큰임을 증명하는 signature 를 지니고 있다는 것입니다
4. 클라이언트 측에서 전달받은 토큰을 저장해두고, 서버에 요청을 할 때 마다, 해당 토큰을 함께 서버에 전달합니다.(웹서버에서 토큰을 서버에 전달 할 때에는, HTTP 요청의 헤더에 토큰값을 포함시켜서 전달합니다.)
5. 서버는 토큰을 검증하고, 요청에 응답합니다.

> 토큰의 장점

1. 무상태(Stateless)
    - 토큰이 클라이언트사이드 저장되기 때문에
2. 확장성(Scalability)
    - 세션을 서버측에 저장하고 있고, 서버를 여러대를 사용하여 요청을 분산하였다면, 어떤 유저가 로그인 했을 때, 그 유저는 처음 로그인했었던 그 서버에만 요청을 보내도록 설정해야한다.
    - 하지만 토큰을 사용한다면, 어떤 서버로 요청이 들어가던 동일한 토큰 검증과정을 거치므로 상관없다.
3. 보안성
    - 클라이언트가 서버에 요청 시, 더 이상 쿠키를 전달하지 않기 때문에 쿠키를 사용함으로 인해 발생하는 취약점이 사라진다.
4. 확장성(Extensibility)

- 로그인 정보가 사용되는 분야를 확장하는 것을 의미한다. 토큰을 사용하여 다른 서비스에서도 권한을 공유할 수 있다. 토큰 기반 시스템에서는, 토큰에 선택적인 권한만 부여하여 발급을 할 수 있다.
- 페이스북 계정으로 로그인했을 때, 프로필 정보를 가져오는 권한은 있지만, 포스트를 작성할 수 있는 권한은 없다.

JWT(JSON Web Token)

> JWT란

토큰 기반 인증 시스템의 구현체이다.

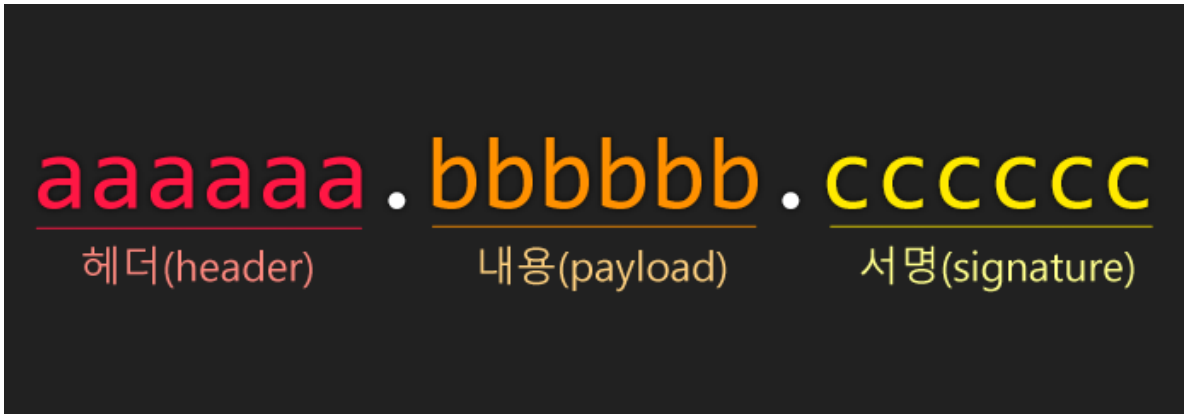
> 특징

1. JSON 객체를 사용하여 가볍고 자가수용적인 방식으로 정보를 안정성있게 전달해줍니다.
2. 주류 프로그래밍 언어(C,JAVA,PYTHON,C++,PHP,JS,,,) 등 대부분 지원된다.
3. 자가 수용적이다.
    - 필요한 모든 정보를 자체적으로 지니고 있다.
        - 토큰에 대한 기본 정보
        - 전달 할 정보(유저 정보)
        - 검증되었다는 것을 증명해주는 서명(signature)
4. 쉽게 전달될 수 있다.
    - HTTP헤더에 넣어 전달
    - URL의 파라메터에 넣어 전달

> 어떤 상황에서 사용하는 것이 좋을까?

- **회원 인증**
    - 유저 로그인 - 서버는 유저 정보에 기반한 토큰을 발급하여 유저에게 전달
    - 그 후, 유저가 서버에 요청시마다 JWT를 포함하여 전달
    - 서버는 클라이언트에게 요청 받을 때마다, 해당 토큰이 유효한지, 인증된 토큰인지 검증
    - 유저가 요청한 작업에 권한이 있는지 확인하여 작업을 처리한다.
- **정보 교류**
    - 두 개체 사이에서 안정성있게 정보를 교환하기에 좋은 방법이다.
- 정보가 sign 되어있기 때문에 정보를 보낸이가 바뀌진 않았는지, 또 정보가 도중에 조작되지는 않았는지 검증가능하다.

> 생김새



1. 헤더(header)

- typ:토큰의 타입 지정 `JWT`
- alg:해싱 알고리즘 지정 `sha256`, `RSA`

```
# Node.js에서의 인코딩
const header = {
  "typ": "JWT",
  "alg": "HS256"
};

// encode to base64
const encodedPayload = new Buffer(JSON.stringify(payload))
                            .toString('base64')
                            .replace('=', '');
console.log('payload: ',encodedPayload);

/* Result:
header: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
*/
```

2. 정보(payload)

- payload 부분에는 토큰에 담을 정보가 들어있습니다. 여기에 담는 정보의 한 '조각'을 **클레임(claim)** 이라고 부르고, 이는 **name/value**의 한 쌍으로 이루어져있습니다.
- 토큰에는 여러개의 클레임들을 넣을 수 있습니다.

  # **클레임**

  - **등록된 클레임** : 토큰 정보를 위한 이름이 정해진 클레임

    `iss` : 토큰 발급자 (issuer)

    `sub` : 토큰 제목 (subject)

    `aud` : 토큰 대상자 (audience)

    `exp` : 토큰의 만료시간 (expiraton), 시간은 NumericDate 형식으로 되어있어야 하며 (예: 1480849147370) 언제나 현재 시간보다 이후로 설정되어있어야합니다.

    `nbf` : Not Before 를 의미하며, 토큰의 활성 날짜와 비슷한 개념입니다. 여기에도 NumericDate 형식으로 날짜를 지정하며, 이 날짜가 지나기 전까지는 토큰이 처리되지 않습니다.

    `iat` : 토큰이 발급된 시간 (issued at), 이 값을 사용하여 토큰의 `age` 가 얼마나 되었는지 판단 할 수 있습니다.

    `jti` : JWT의 고유 식별자로서, 주로 중복적인 처리를 방지하기 위하여 사용됩니다. 일회용 토큰에 사용하면 유용합니다.
  - **공개 클레임** : 충돌 방지를 위해 URL 형식으로 된 클레임 사용하기
  - **비공개 클레임** : 클라이언트와 서버 협의 하에 사용되는 비공개 클레임. 충돌에 유의하여 사용

  으로 구분가능합니다.

```
const payload = {
    "iss": "velopert.com",  // 등록된 클레임
    "exp": "1485270000000", // 등록된 클레임
    "https://velopert.com/jwt_claims/is_admin": true,   // 공개 클레임
    "userId": "11028373727102", // 비공개 클레임
```

```
    "username": "velopert"      // 비공개 클레임
};

// encode to base64
const encodedPayload = new Buffer(JSON.stringify(payload))
                            .toString('base64')
                            .replace('=', '');

console.log('payload: ',encodedPayload);

/* result
payload:
eyJpc3MiOiJ2ZWxvcGVydC5jb20iLCJleHAiOiIxNDg1MjcwMDAwMDAwIiwiaHR0cHM6Ly9
2ZWxvcGVydC5jb20vandOX2NsYWltcy9pc19hZG1pbiI6dHJ1ZSwidXNlcklkIjoiMTEwMj
gzNzM3MjcxMDIiLCJ1c2VybmFtZSI6InZlbG9wZXJ0In0
*/
```

3. 서명(Signature)

- 헤더의 인코딩값, payload의 인코딩값을 합친 후(`.`을 넣어줍니다) 비밀키로 다시 해쉬하여 생성한다.

```
// 수도코드
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

```
const crypto = require('crypto');
const signature = crypto.createHmac('sha256', 'secret')
              .update(encodedHeader + '.' + encodedPayload)
              .digest('base64')
              .replace('=', '');

console.log('signature: ',signature);

/* result
Signature:  WE5fMufMONDSVGJ8cAolXGkyB5RmYwCto1pQwDIqo2w
*/
```

4. 결과

- 지금까지 구한 값들을 `.`를 중간자로 다 합쳐주면, 하나의 토큰이 완성된다.

```
/* result
result:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJ2ZWxvcGVydC5jb20iLCJleH
AiOiIxNDg1MjcwMDAwMDAwIiwiaHR0cHM6Ly92ZWxvcGVydC5jb20vandOX2NsYWltcy9pc
19hZG1pbiI6dHJ1ZSwidXNlcklkIjoiMTEwMjgzNzM3MjcxMDIiLCJ1c2VybmFtZSI6InZl
bG9wZXJ0In0.WE5fMufMONDSVGJ8cAolXGkyB5RmYwCto1pQwDIqo2w
*/
```

5. 실습 결과 : https://jwt.io/

- 참고 자료 :
  - 이론 - https://velopert.com/2389
  - 실습 - https://alwayspr.tistory.com/8