클라우드 보안

최 민

- **학습목표**
  - 클라우드 환경에서 IAM 과 JWT를 통한 인증
- **학습활동**
  - Identity and Access Management (IAM) 개념과 원리를 학습합니다.
  - IAM 역할, 정책(Policy)을 수립하고 관리하는 방법을 살펴봅니다.
  - 클라우드 환경에서 JWT의 장점을 이해합니다.
  - 클라우드 환경에서 널리 쓰이는 JWT를 통한 인증 방법을 알아봅니다.

사전학습

- 클라우드 계정보안 서비스(**IAM**) 사례

https://youtu.be/SXSqhTn2DuE

1. 계정에 대한 공유 엑세스 : 암호나 액세스 키를 공유하지 않고도 AWS 계정의 리소스를 관리하고 사용할 수 있는 권한을 다른 사람에게 부여 가능
2. 세분화된 권한 : 리소스에 따라 여러 사람 또는 EC2 인스턴스에서 실행되는 애플리케이션에 안전하게 제공
3. 멀티 팩터 인증(MFA) : 계정 작업을 위해 암호나 액세스 키뿐 아니라 특별히 구성된 디바이스 코드도 제공
4. 자격 증명 연동 : 기업 네트워크나 인터넷 자격 증명 공급자와 같은 다른 곳에 이미 암호가 있는 사용자에게 AWS 계정에 대한 임시 액세스 권한 부여 가능
5. IAM 액세스 방식 : AWS Management Console, AWS 명령줄 도구(CLI, Window용 PowerShell), AWS SDK, IAM HTTPS API

Identity and Access Management (IAM)

IAM

- **IAM(Identity and Access Management)**
  - IAM : AWS 리소스에 대한 액세스를 시큐어하게 제어할 수 있도록 하는 웹 서비스.

IAM is a web service that helps you to securely control access to AWS resources. Use IAM to control who is *authenticated* (signed in) and *authorized* (has permissions) to use resources.

  - 인증(Authentication) : 액세스를 요청하는 사람에 관한 것.
  - 인가(Authorization)는 액세스할 수 있는 항목을 결정함.

- **궁극적으로 IAM은 사용자, 그룹, 역할 및 정책을 사용하여 AWS 리소스에 대한 인증 및 인가(권한 부여)를 제공함.**

IAM

- 인증 **(Authentication)**
  - AWS 계정과 그 안에 있는 리소스에 대한 액세스를 누가 요청하는지 여부
  - 자격 증명을 통해 요청자의 신원을 확인
  - 요청자는 사람 또는 애플리케이션일 수 있음 <- IAM은 이들을 보안 주체(principals)라 함

- 인가**(Authroization, 권한의 부여)**
  - 요청자가 인증 절차를 거친 후, 무엇을 할 수 있도록 허용할 것인지
  - IAM은 요청을 허용할지 또는 거부할지 결정하기 위해 요청과 관련된 정책을 확인함.

IAM

- **IAM(Identity and Access Management)**
  - AWS 리소스에 대한 개인 및 그룹 액세스를 안전하게 공유 및 제어
  - 다른 많은 AWS 서비스와 통합
  - 연합 ID 관리 지원
  - 세분화된 권한 지원
  - 다단계 인증(MFA) 지원
  - 보증을 위한 신원 정보 제공:

AWS Identity and Access Management (IAM)

IAM

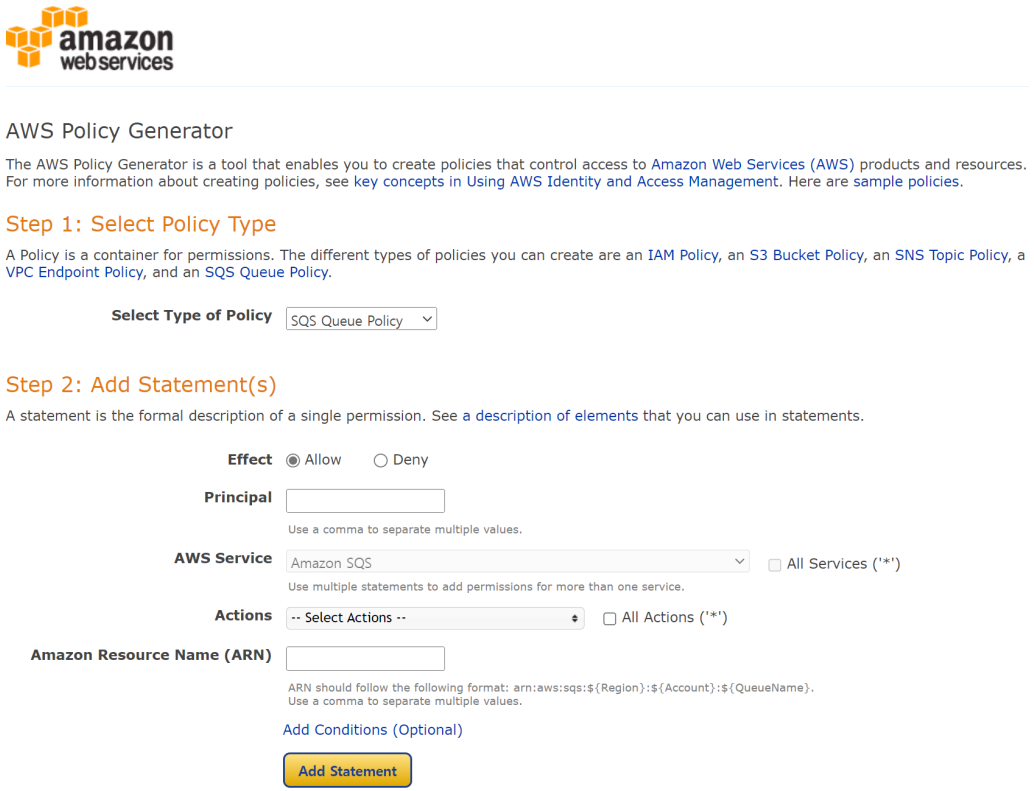
- 사용자(User)
  - AWS 계정으로 인증할 수 있는 사람 또는 애플리케이션
- 그룹(Group)
  - 동일한 권한이 부여된 IAM 사용자 모음
- 역할(Role)
  - AWS 서비스 요청을 할 수 있는 임시 권한 집합을 부여하는 데 사용되는 자격 증명 메커니즘
- IAM 정책(IAM Policy)
  - 액세스 할 수 있는 자원(resource)과 각 자원(resource)에 대한 접근 수준을 정의한 문서

IAM

- **IAM 엔티티(entity)**
  - Used by AWS for authentication (users and roles)
- **IAM 아이덴티티(identity)**
  - Used to identify and group
  - You can attach a policy to an IAM identity (user, group, or role).
- **IAM 리소스(resource):**
  - The user, group, role, policy, and identity provider objects that are stored in IAM
  - You can add, edit, and remove resources from IAM.
- **Principal**
  - A person or application that uses the AWS account root user, IAM user, or IAM role to sign in and make requests to AWS

IAM

```
{
    "Version": "2012-10-17",
    "Id": "Policy1687505316376",
    "Statement": [
        {
            "Sid": "Stmt1687505298027",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::ph
        }
    ]
}
```



IAM

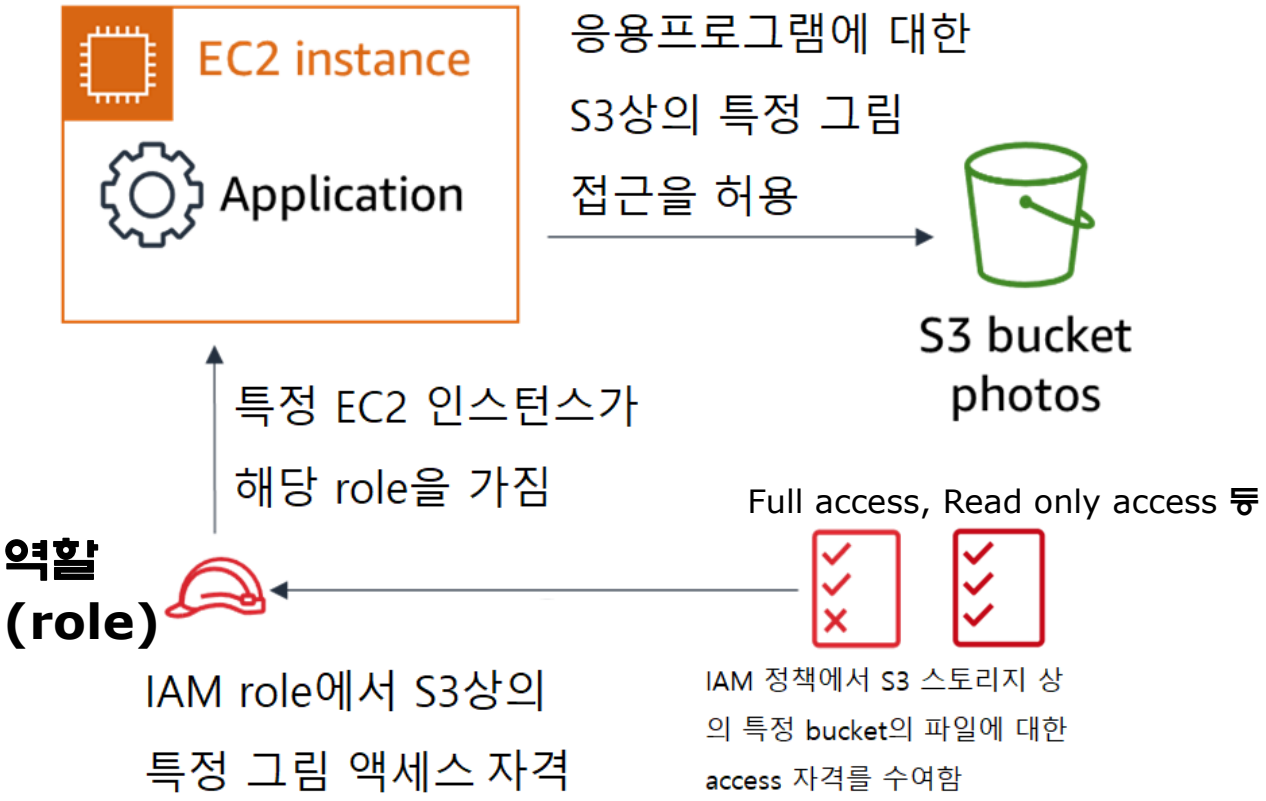
- **IAM 요청(request)**
  - 보안 주체가 AWS Management Console, 애플리케이션 프로그래밍 인터페이스(API) 또는 AWS 명령줄 인터페이스(AWS CLI)를 사용하려고 시도할 때마다 요청이 이루어집니다.
- **IAM 요청(request)에 포함되는 정보**
  - 행동 또는 운영(actions/operations): 주체가 수행하고자 하는 것
  - 리소스: 작업 또는 작업이 수행되는 객체
  - 주체(principal): 사용자 또는 역할을 사용하여 요청을 보내는 사람 또는 애플리케이션
  - 환경 데이터: IP 주소, 사용자 에이전트, SSL(Secure Sockets Layer) 활성화 상태 또는 시간
  - 자원 데이터: 요청 중인 자원과 관련된 데이터

IAM

- **IAM 역할(Roles) 특성**
  - 임시 보안 자격 증명을 제공.
  - 역할(role)은 한 사람과 고유하게 연결되지 않음.
  - 사람, 애플리케이션 또는 AWS 서비스가 해당 역할을 맡을 수 있음.
  - 역할은 종종 액세스 권한을 위임하는 데 사용됨
  - AWS Security Token Service(AWS STS)는 임시 보안 자격 증명의 발급 가능
- **일반적인 사용 사례**
  - Amazon Elastic Compute Cloud(Amazon EC2)에서 실행되는 애플리케이션
  - IAM 사용자의 교차 계정 액세스
  - 모바일 애플리케이션

IAM

- **IAM 역할(role)**의 사용 사례



시나리오:

EC2 인스턴스에서 실행되는 애플리케이션에 S3 버킷에 대한 액세스 권한이 필요

방법:

S3 버킷에 대한 액세스 권한을 부여하는 IAM 정책 정의
정책을 역할에 연결
EC2 인스턴스가 이 역할을 수임하는 것을 허용

IAM

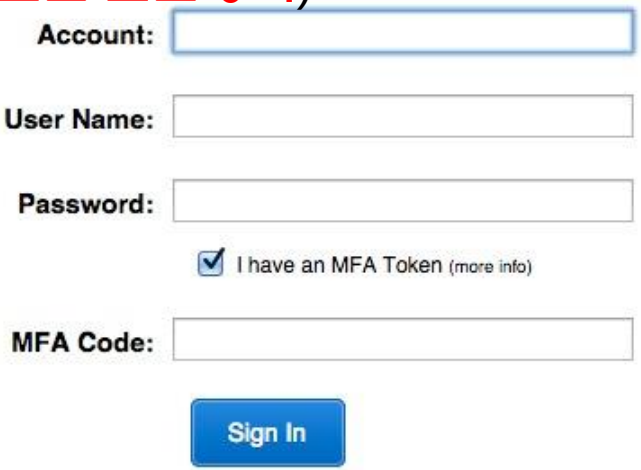
- 인증을 위한 **IAM** 크레덴셜**(credentials)**
  - A credential can be defined as any document, object, or data structure that vouches for the identity of a person through some method of trust and authentication.

```
3
4   //AWS access details
5   AWS.config.update({
6       accessKeyId: '
7       secretAccessKey: '                                        ',
8       region: 'ap-southeast-2'
9    });
```

ACCESS KEY ID
AKIAIOSFODNN7EXAMPLE
SECRET KEY
wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

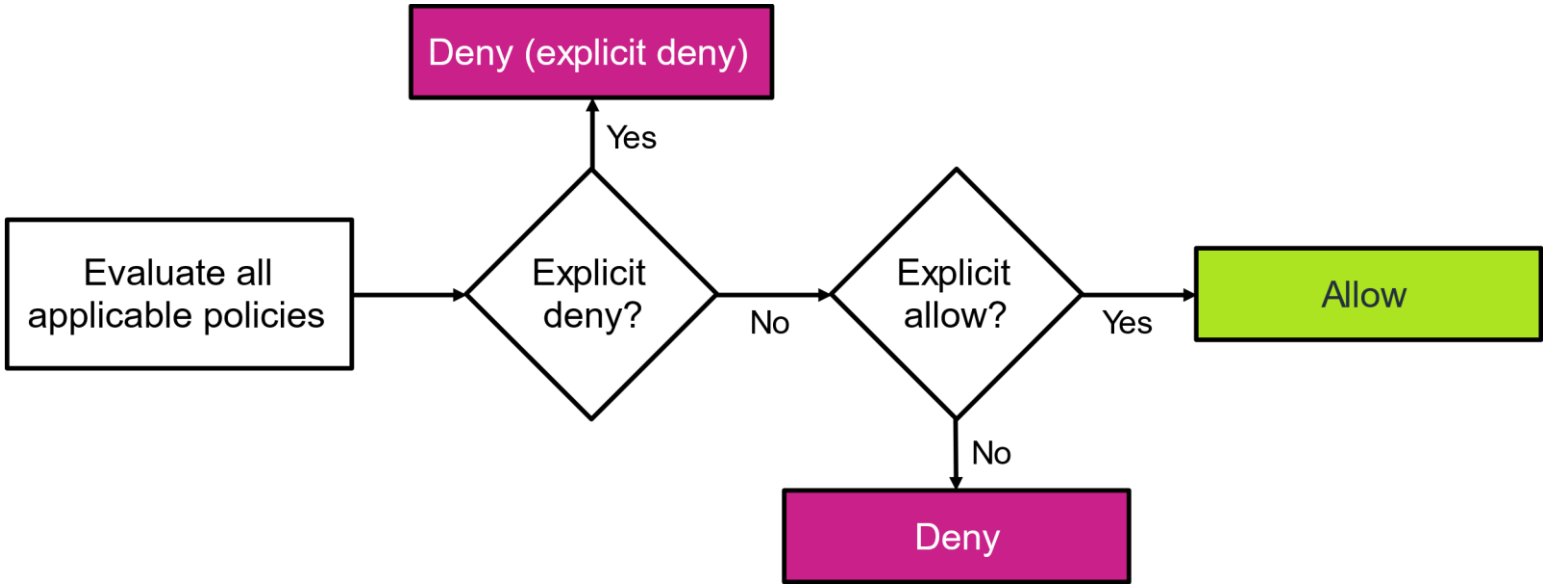
액세스키 와 비밀 액세스 키 (프로그램에 의한 접근방식)

12자리 계정ID 또는 별칭
사용자 아이디(user name) 와
비밀번호(password)
(콘솔 액세스에 의한 접근방식)



IAM

- **IAM 정책의 평가로직**



  - IAM 정책을 IAM 사용자, IAM 그룹 또는 IAM 역할에 연결할 경우, 즉 계정 액세스 권한을 부여할 때 최소 권한 원칙을 따르기를 권고하며, IAM이 권한을 결정할 때 명시적 거부(deny)는 항상 모든 허용(allow) 문을 재정의함

IAM 정책

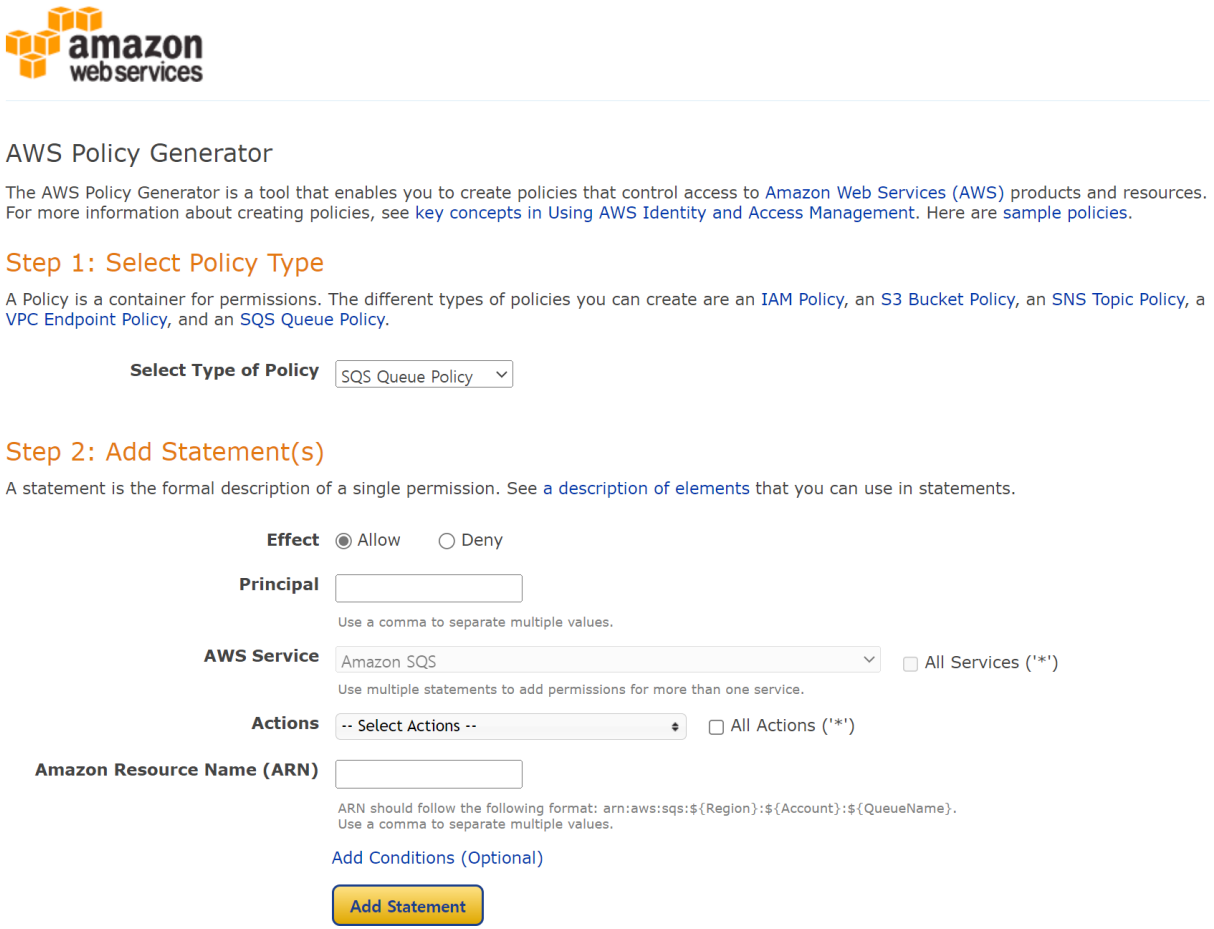
```
{
  "Version": "2012-10-17",
  "Statement":[{
    "Effect":"Allow",
    "Action":["DynamoDB:*","s3:*"],
    "Resource":[
            "arn:aws:dynamodb:region:account-number-without-hyphens:table/table-name",
      "arn:aws:s3:::bucket-name",
      "arn:aws:s3:::bucket-name/*"]
    },
    {
    "Effect":"Deny",
    "Action":["dynamodb:*","s3:*"],
    "NotResource":["arn:aws:dynamodb:region:account-number-without-hyphens:table/table-
name",
      "arn:aws:s3:::bucket-name",
      "arn:aws:s3:::bucket-name/*"]
    }
  ]
}
```

명시적으로 허용하는 행위는 사용자에게 특정 DynamoDB 테이블, 특정 S3 버킷에 대한 액세스 권한을 부여함.

명시적으로 거부하는 행위는 허용(allow)문보다 우선 적용됨.

IAM 정책

- 정책(Policy) 생성기



IAM 정책

- 정책 종류(**Identity based, Resource based**)

Identity 기반 정책

홍길동

Resource	Read	Write	List
Resource X	Allow	Allow	Allow

이순신

Resource	Read	Write	List
Resource Y	Allow	N/A	N/A
Resource Z	Allow	N/A	N/A

관리자

Resource	Read	Write	List
Resource X	N/A	N/A	Allow
Resource Y	N/A	N/A	Allow
Resource Z	N/A	N/A	Allow

Resource-기반 정책

자원 1

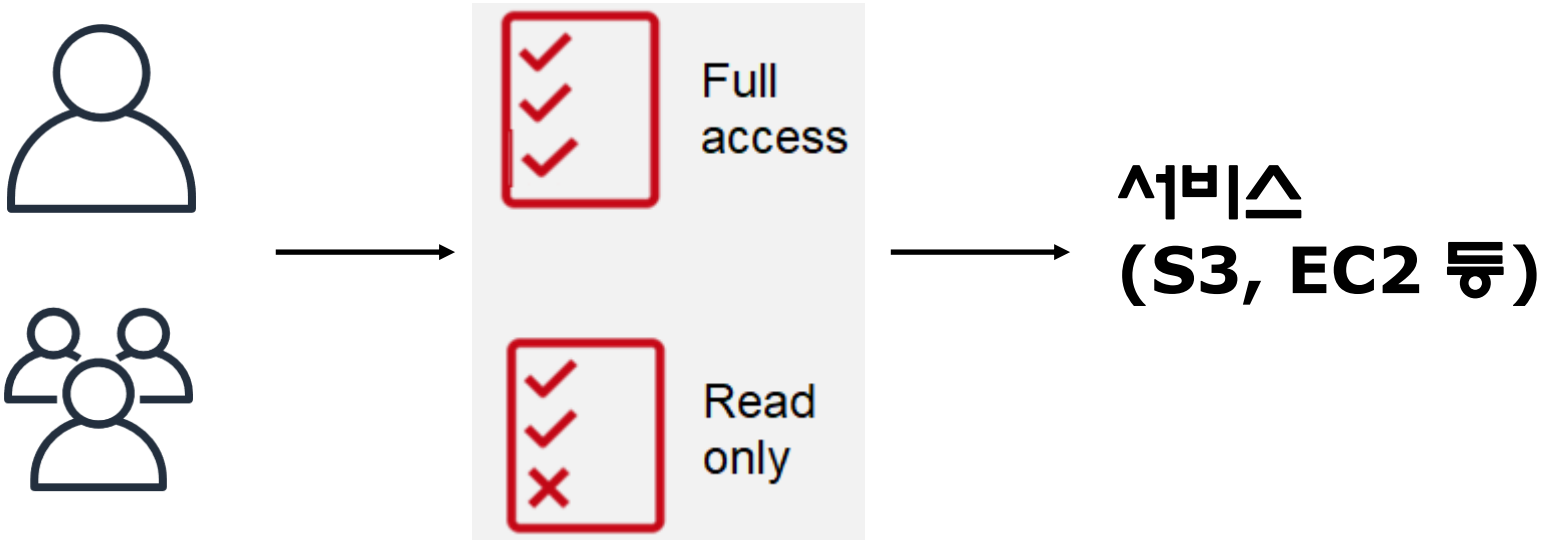
User	Read	Write	List
Ana	Allow	Allow	Allow
Akua	Allow	Allow	Allow
Mary	Allow	N/A	Allow
Mateo	N/A	N/A	Allow

자원 2

User	Read	Write	List
Paulo	Allow	Allow	Allow
Nikki	Allow	N/A	N/A
Mateo	N/A	Allow	Allow

IAM 정책

- 사용자, 그룹, 역할 및 리소스에 권한 부여시, 인증의 표준 보안 조언과 최소 권한 원칙**(principal of least privilege)**을 따름
- 이 방법은 작업을 수행하는 데 필요한 최소한의 권한만 부여한다는 의미임



JSON Web Token (JWT)

JWT

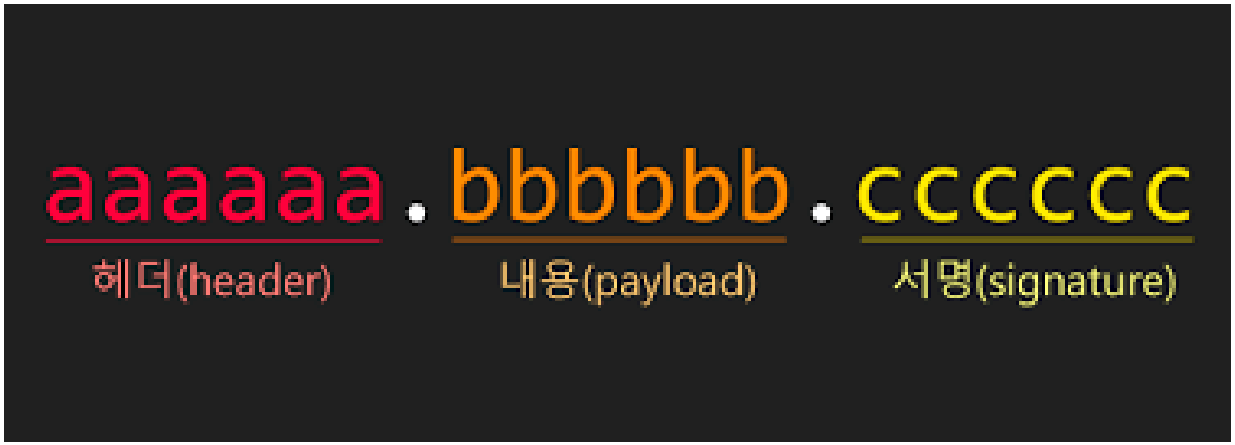
- **JWT**
  - IETF RFC7519 Web Standard
  - Signature 토큰 유효성 검증을 위한 문자열
- **장점**
  - 중앙의 인증서버, 데이터 스토어에 대한 의존성 없음(시스템 수평 확장에 유리)
  - Base64 URL Safe Encoding을 사용하기 때문에 URL, Cookie, Header 모두 사용 가능
- **단점**
  - Payload 정보가 많아지면 네트워크 사용량 증가
  - 토큰이 클라이언트에 저장되기 때문에 서버에서 클라이언트의 토큰 내용을 조작할 수 없음.

JWT

- **JWT(Json Web Token)**
  - JWT(Json Web Token)은 웹표준(RFC7519)로서 일반적으로 클라이언트와 서버, 서비스와 서비스 사이 통신 시 권한 인가(Authorization)을 위해 사용하는 토큰
- **HEADER.PAYLOAD.SIGNATURE**
  - 일반적으로 헤더, 페이로드, 서명 세 부분을 점으로 구분하는 구조
  - 헤더 : 토큰 타입과 해싱 알고리즘을 저장
  - 페이로드 : 실제로 전달하는 정보,
  - 서명 : 위변조를 방지하기위한 값

JWT

- **헤더(header), 페이로드(payload), verify signature**
  - header : Header, Payload, Verify Signature 를 암호화할 방식(alg), 타입(Type) 등을 포함
  - Payload :서버에서 보낼 데이터 - 일반적으로 user의 id, 유효기간 포함
  - Verify Signature : Base64 방식으로 인코딩한 Header, Payload, Secret key 를 더한 값



JWT

Debugger

Warning: JWTs are credentials, which can grant access to resources. Be careful where you paste them! We do not record tokens, all validation and debugging is done on the client side.

Algorithm HS256

Encoded PASTE A TOKEN HERE


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```


Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Signature Verified

SHARE JWT

JWT

- 알고리즘 **1) HS256 (HMAC with SHA-256)**
  - HMAC (Hash based Message Authentication Code = Hash(Message, Key) + Message
  - 클라이언트와 서버간 또는 서버 대 서버간 메시지를 주고 받았을 때 변조여부를 확인해야 한다. 원본 메시지와 공유된 메시지를 비교하여 변조 여부를 확인하는 것이 MAC(Message Authentication Code)
  - Hash 함수는 SHA1, SHA2등의 알고리즘 사용
  - HMAC는 원본 메시지가 변하면 그 해시값도 변하는 해싱(Hashing)의 특징을 활용하며 메시지의 변조 여부를 확인하여 무결성을 제공하는 기술

JWT

- **알고리즘 2) RS256 (RSA Signature with SHA-256)**
  - An asymmetric algorithm, 비대칭형 알고리즘, 공개키(Public Key)와 개인키(Private Key) 2개의 키를 활용
  - 큰 수의 소인수분해의 어려움을 활용하는 암호화 알로리즘
  - 공개키로 암호화 한 암호문은 개인키로 풀 수 있음
  - 개인키는 오픈해서는 안되며 공개키로 개인키를 알아낼 수 없음.
- **절차**
  - A Service에서 공개키(Samsung)와 개인키(Apple)을 생성함
  - B Service에서 공개키를 요청함
  - A Service에서 B Service로 공개키를 전달함
  - B Service는 Message(Korea)를 A Service에서 전달 받은 공개키(Samsung)으로 암호화함
  - A Service로 암호화된 값을 전달함
  - A Service는 암호화된 값을 개인키(Apple)로 복호화함

JWT

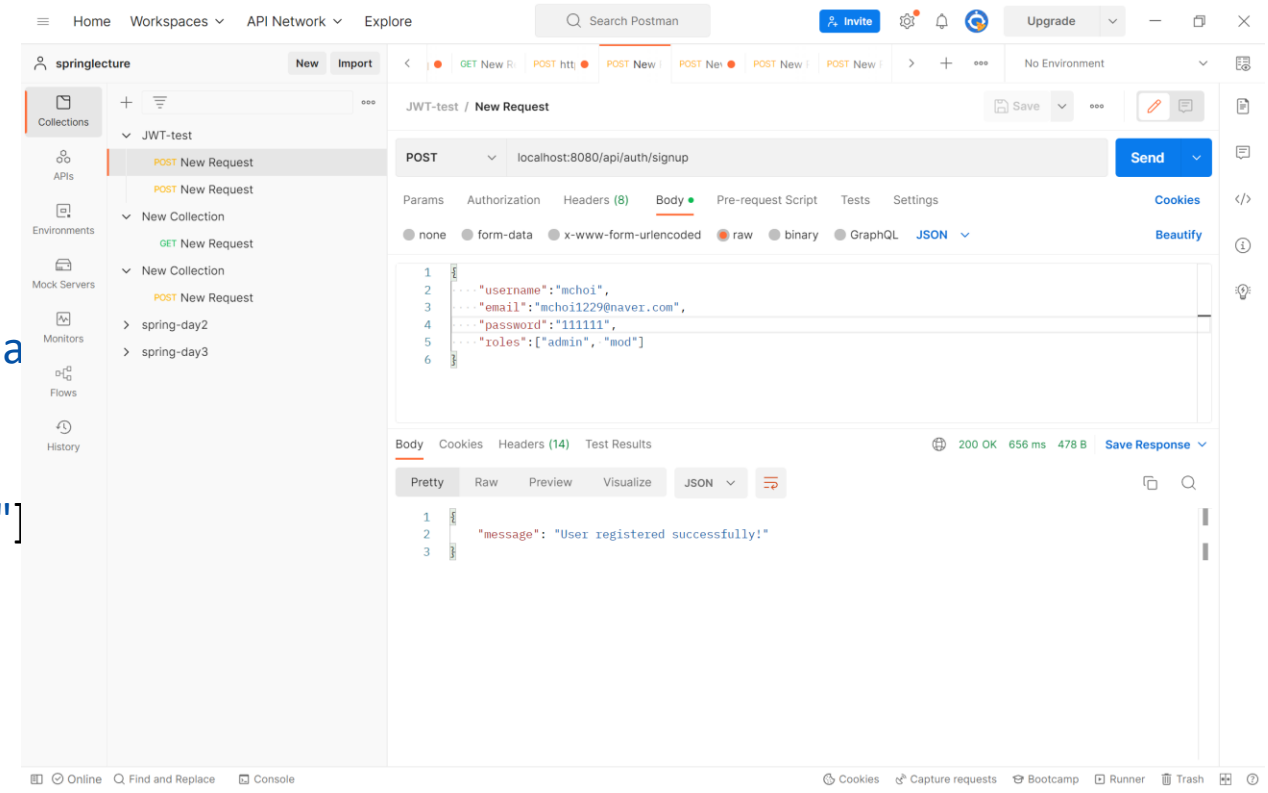
- **JWT를 통한 인증절차**
  - 사용자가 로그인함
  - 서버에서는 계정 정보를 읽어 사용자를 확인 후, 사용자의 고유 ID 값을 부여한 후 기타 정보와 함께 Payload 에 넣음
  - JWT 토큰의 유효기간을 설정함
  - 암호화할 Secret key 를 이용해 Access Token 을 발급함
  - 사용자는 Access Token 을 받아 저장 후, 인증이 필요한 요청마다 토큰을 헤더에 실어 보냄
  - 서버에서는 해당 토큰의 Verify Signature 를 Secret key 로 복호화 한 후, 조작 여부, 유효기간을 확인함
  - 검증이 완료되었을 경우, Payload 를 디코딩 하여 사용자의 ID 에 맞는 데이터를 가져옴

Access Token의 유효기간은 매우 짧음.
Refresh Token을 별도로 발급하여, Access Token이 만료되면
새로운 JWT를 발급할 수 있도록 함.

JWT

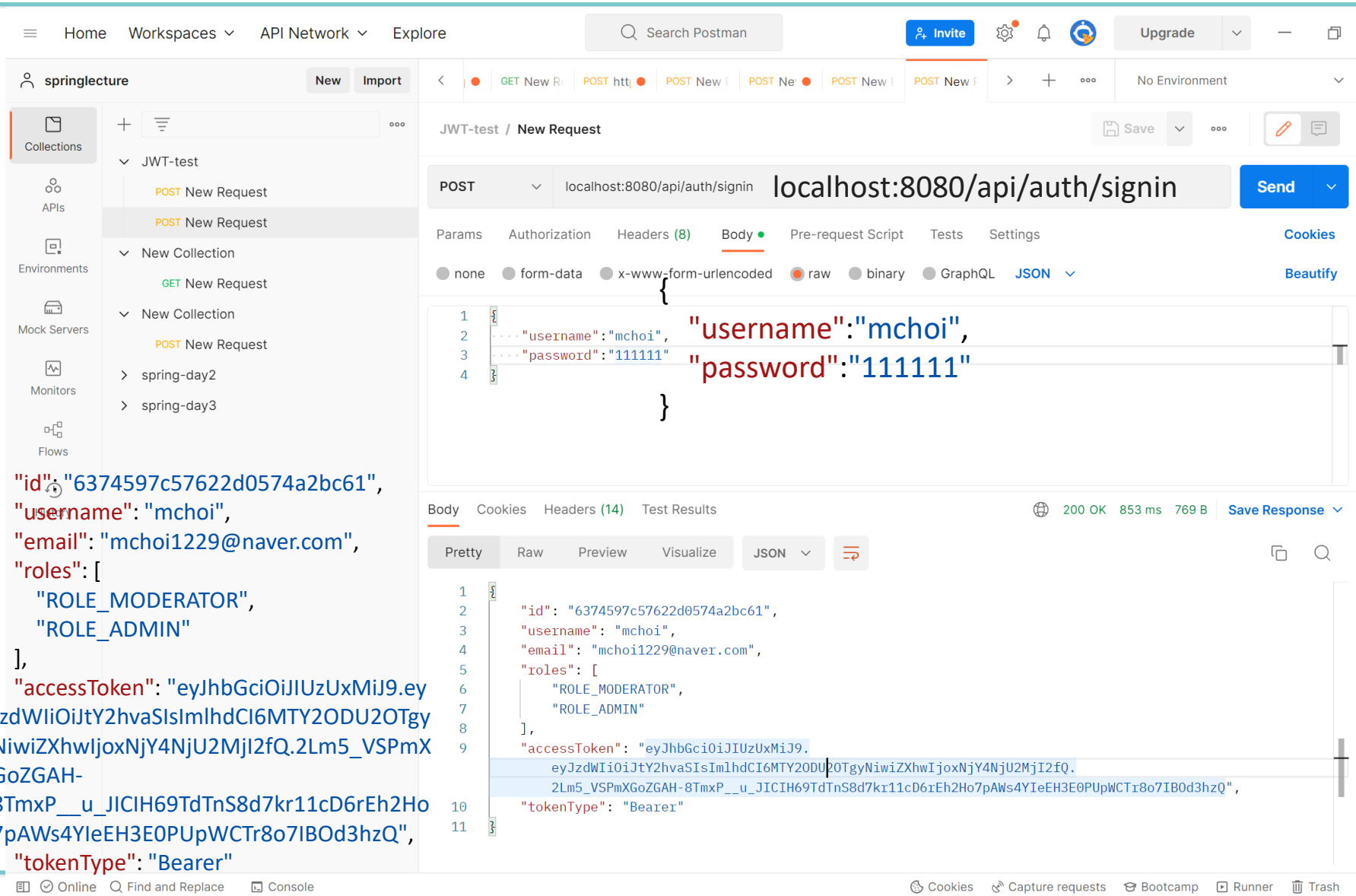
- **JWT 인증 사례** localhost:8080/api/auth/signup

```
{
   "username":"mchoi",
   "email":"mchoi1229@na
om",
   "password":"111111",
   "roles":["admin", "mod"]
}
{
   "code": 200,
   "msg": "Success"
}
```



POSTMAN 프로그램으로 실행한 화면

JWT




```
{
  "id": "6374597c57622d0574a2bc61",
  "username": "mchoi",
  "email": "mchoi1229@naver.com",
  "roles": [
    "ROLE_MODERATOR",
    "ROLE_ADMIN"
  ],
  "accessToken": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJtY2hvaSIsImlhdCI6MTY2ODU2OTgyNiwiZXhwIjoxNjY4NjU2MjI2fQ.2Lm5_VSPmXGoZGAH-8TmxP__u_JICIH69TdTnS8d7kr11cD6rEh2Ho7pAWs4YIeEH3E0PUpWCTr8o7IBOd3hzQ",
  "tokenType": "Bearer"
}
```

JWT

- **JWT Test code in Springboot**

```
@Log4j2
public class JWTUtil {
    private String secretKey = "zerock12345678";
    //1month
    private long expire = 60 * 24* 30;
    public String generateToken(String content) throws Exception{
        return Jwts.builder()
                .setIssuedAt(new Date())
                .setExpiration(Date.from(ZonedDateTime.now().plusMinutes(expire).toInstant()))
                //.setExpiration(Date.from(ZonedDateTime.now().plusSeconds(1).toInstant()))
                .claim("sub", content)
                .signWith(SignatureAlgorithm.HS256, secretKey.getBytes("UTF-8"))
                .compact();
    }
```

```
    public String validateAndExtract(String tokenStr)throws Exception {
        String contentValue = null;
        try {
            DefaultJws defaultJws = (DefaultJws) Jwts.parser()
                    .setSigningKey(secretKey.getBytes("UTF-
8")).parseClaimsJws(tokenStr);
            log.info(defaultJws);
            log.info(defaultJws.getBody().getClass());
            DefaultClaims claims = (DefaultClaims) defaultJws.getBody();
            log.info("------------------------");
            contentValue = claims.getSubject();
        }catch(Exception e){
            e.printStackTrace();
            log.error(e.getMessage());
            contentValue = null;
        }
        return contentValue;
    }
}
```

JWT

- **JWT Test code in Springboot**

```
@SpringBootTest
class JwtUserWebApplicationTests {

   private JWTUtil jwtUtil;

   @DisplayName("Token 생성 및 검증")
   @Test
   void test1() {

      Algorithm AL = Algorithm.HMAC256("test");
      String token = JWT.create()
            .withSubject("mchoi")
            .withClaim("exp", Instant.now().getEpochSecond()+3)
            .withArrayClaim("role", new String[]{"ROLE_ADMIN",
"ROLE_USER"})
            //.sign(Algorithm.none());
            .sign(AL);

      System.out.println(token);

      DecodedJWT decode = JWT.decode(token);
      System.out.println("typ = " + decode.getHeaderClaim("typ"));
      System.out.println("alg = " + decode.getHeaderClaim("alg"));

      try {
         Thread.sleep(1000);
      } catch (InterruptedException e) {
         throw new RuntimeException(e);
      }
      DecodedJWT decode2 =
JWT.require(AL).build().verify(token);
      System.out.println("decoding result = " + decode2);

      System.out.println("waiting for 3 seconds.....");
      try {
         Thread.sleep(3000);
      } catch (InterruptedException e) {
         throw new RuntimeException(e);
      }
      System.out.println("After 3 seconds..... ");

      DecodedJWT decode3 =
JWT.require(AL).build().verify(token);
      System.out.println("decoding result = " + decode3);

   }
```

JWT

- **Roles 컬렉션 설정**
  - Mongodb Tools 를 통하여 roles 컬렉션에 다음 3개 ROLE 정보를 저장(Role 정보가 없을 경우, 실행시 Roles not found 예외 발생)

```
db.roles.insertMany([
                { name: "ROLE_USER" },
                { name: "ROLE_MODERATOR" },
                { name: "ROLE_ADMIN" },
])
```

Mongodb tools **설치를 통해 문장 실행 가능**

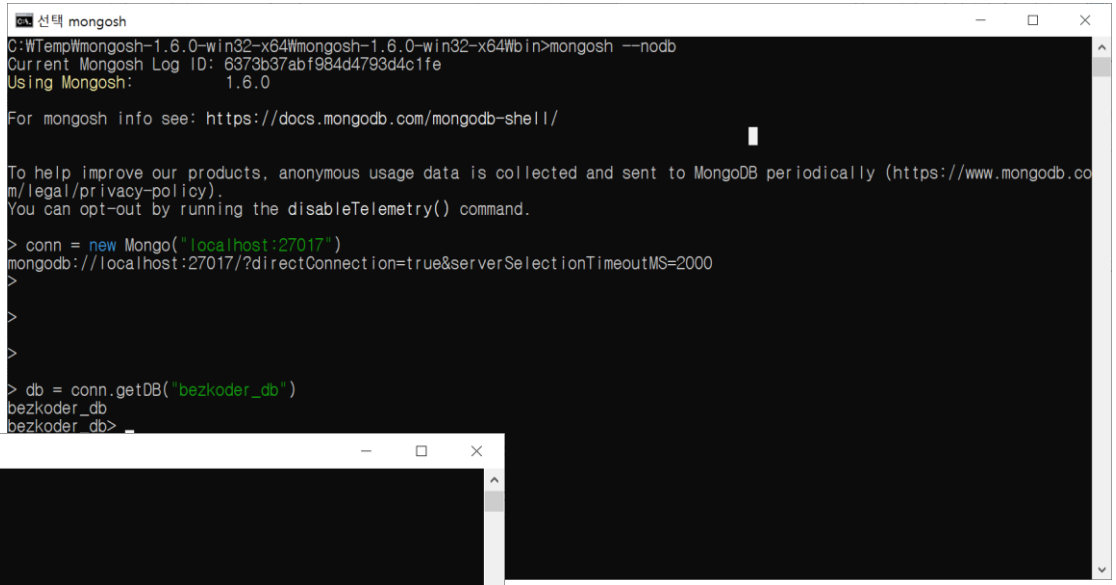
JWT

- **/api/test/all**
  - for public access
- **/api/test/user**
  - for users has ROLE_USER or ROLE_MODERATOR or ROLE_ADMIN
- **/api/test/mod**
  - for users has ROLE_MODERATOR
- **/api/test/admin**
  - for users has ROLE_ADMIN

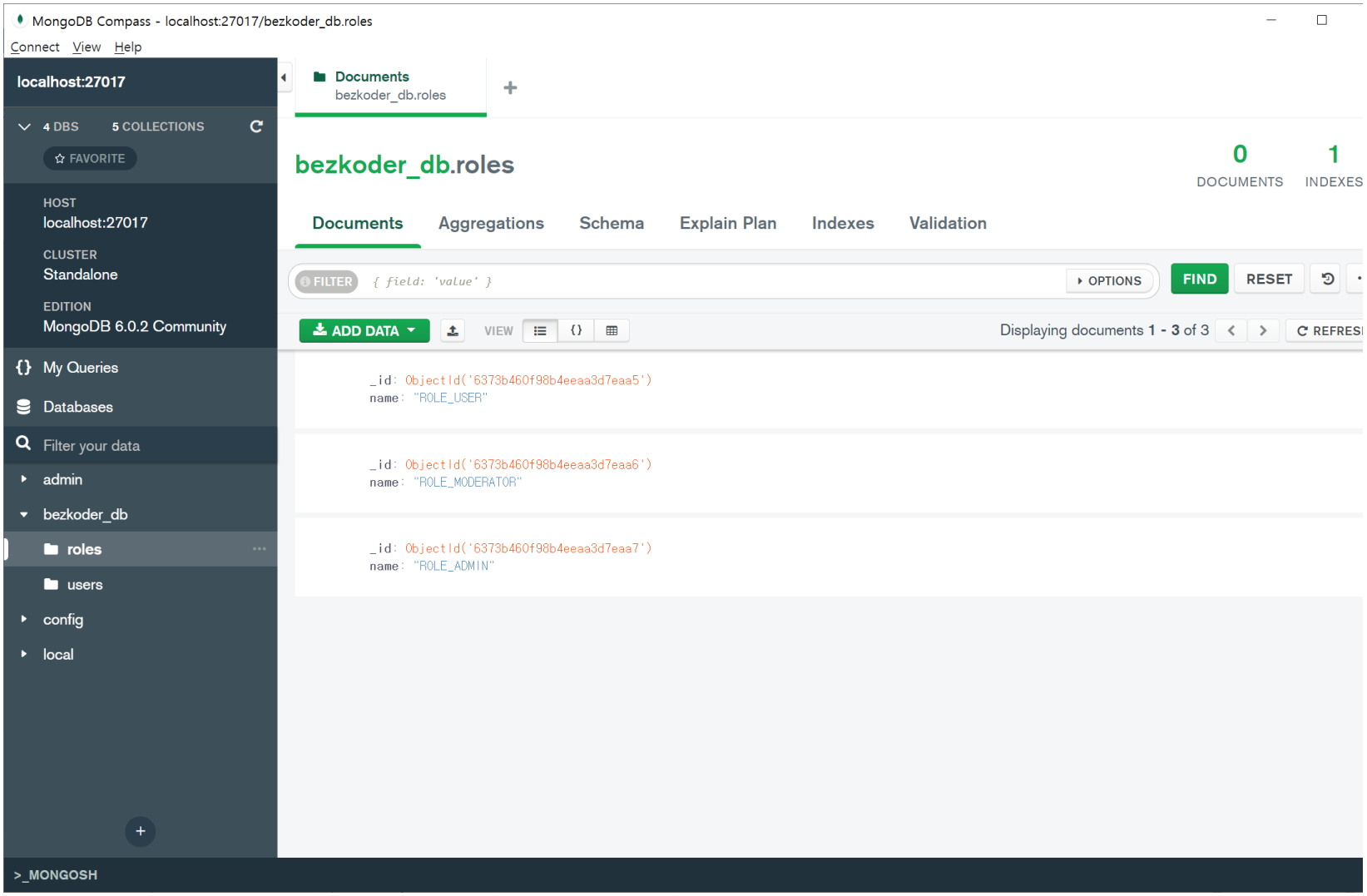
```
dependencies {
...
implementation "org.springframework.boot:spring-boot-starter-security“
testImplementation 'org.springframework.security:spring-security-test’
implementation 'io.jsonwebtoken:jjwt:0.9.1’
...
}
```

JWT

- **Roles 컬렉션 설정**



JWT


MongoDB Compass - localhost:27017/bezkoder_db.roles
Connect View Help
localhost:27017
4 DBS 5 COLLECTIONS
FAVORITE
HOST
localhost:27017
CLUSTER
Standalone
EDITION
MongoDB 6.0.2 Community
My Queries
Databases
Filter your data
admin
bezkoder_db
roles
users
config
local
Documents
bezkoder_db.roles
bezkoder_db.roles
0
DOCUMENTS
1
INDEXES
Documents
Aggregations
Schema
Explain Plan
Indexes
Validation
FILTER
{ field: 'value' }
OPTIONS
FIND
RESET
ADD DATA
VIEW
Displaying documents 1 - 3 of 3
_id: ObjectId('6373b460f98b4eeaa3d7eaa5')
name: "ROLE_USER"
_id: ObjectId('6373b460f98b4eeaa3d7eaa6')
name: "ROLE_MODERATOR"
_id: ObjectId('6373b460f98b4eeaa3d7eaa7')
name: "ROLE_ADMIN"
>_MONGOSH

JWT

- WebSecurityConfig: WebSecurityConfigurerAdapter를 상속받은 Configuration
- UserDeatailServiceImpl: UserDetailService의 구현체
- JwtUtils: JWT를 파싱, 생성, 검증을 하는 방법을 제공하는 클래스
- Controller: 가입 및 로그인 등의 승인된 요청을 처리하기 위한 레이어
- UserController: signIn, signup 등
- TestController: 유저 권한 인증을 테스트 하기 위한 컨트롤러
- Service: 요청한 작업에 대해 데이터를 처리하거나 응답해주는 로직이 담긴 레이어
- Repository: Jpa를 사용하여 실제로 Entity를 통한 동적 쿼리를 만드는 레이어
- Model: 테이블에 해당하는 Entity를 모아 놓는 레이어

JWT

SpringSecurity 설정(configure)

```
@Override
    protected void configure(HttpSecurity http) throws Exception {
        final String[] GET_WHITELIST = new String[]{
                "/login",
                "/user/login-id/**",
                "/user/email/**",
                "/affiliate"
        };
        final String[] POST_WHITELIST = new String[]{
                "/client-user"
        };
        http.csrf().disable()
                .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and().exceptionHandling()
                .authenticationEntryPoint(authenticationEntryPoint) // 인증 실패
                .accessDeniedHandler(accessDeniedHandler) // 인가 실패
                .and().authorizeRequests()
                .antMatchers(HttpMethod.GET, GET_WHITELIST).permitAll() // 해당 GET URL은 모두 허용
                .antMatchers(HttpMethod.POST, POST_WHITELIST).permitAll() // 해당 POST URL은 모두 허용
                .antMatchers("/client-user/**").hasAnyRole(UserType.CL.getRoll()) // 권한 적용
                .anyRequest().authenticated() // 나머지 요청에 대해서는 인증을 요구
                .and() // 로그인하는 경우에 대해 설정함
                .formLogin().disable() // 로그인 페이지 사용 안함
                .addFilterBefore(authenticationFilter(), UsernamePasswordAuthenticationFilter.class)
                .addFilterBefore(jwtFilter(), UsernamePasswordAuthenticationFilter.class);
    }
```

JWT

- **AuthenticationProvider 구현**
  - AuthenticationManager 하위에 실제로 인증을 처리할 AuthenticationProvider를 구현

```
@Component
@RequiredArgsConstructor
public class AuthenticationProviderImpl implements AuthenticationProvider {

    private final UserDetailsService userDetailsService;
    private final PasswordEncoder passwordEncoder;

    @Override
    public Authentication authenticate(Authentication authentication) throws AuthenticationException {
        UsernamePasswordAuthenticationToken token = (UsernamePasswordAuthenticationToken) authentication;
        String username = token.getName();
        String password = (String) token.getCredentials();

        UserDetailsImpl userDetail = (UserDetailsImpl) userDetailsService.loadUserByUsername(username);
        if (!passwordEncoder.matches(password, userDetail.getPassword()))
            throw new BadCredentialsException(userDetail.getUsername() + "Invalid password");
        // 인증 성공 시 UsernamePasswordAuthenticationToken 반환
        return new UsernamePasswordAuthenticationToken(userDetail.getUsername(), "", userDetail.getAuthoritie
    }
    @Override
    public boolean supports(Class<?> authentication) {
        return authentication.equals(UsernamePasswordAuthenticationToken.class);
    }
}
```

토큰발급과 토큰검증

```
@Component
@RequiredArgsConstructor
public final class JwtProvider {

    private final UserDetailsService userDetailsService;
    @Value("${jwt.secret-key}")
    private String secretKey;
    private final long accessTokenValidTime = 2 * 60 * 60 * 1000L;
    private final long refreshTokenValidTime = 2 * 7 * 24 * 60 * 60 * 1000L;

    @PostConstruct
    private void init() {
        secretKey = Base64.getEncoder().encodeToString(secretKey.getBytes());
    }

    private Claims getClaimsFormToken(String token) {
        return
Jwts.parser().setSigningKey(DatatypeConverter.parseBase64Binary(secretKey)).parseCl
aimsJws(token).getBody();
    }

    private String getSubject(String token) {
        return getClaimsFormToken(token).getSubject();
    }
```

토큰발급과 토큰검증

```
public Authentication getAuthentication(String token) {
        UserDetails userDetails = userDetailsService.loadUserByUsername(this.getSubject(token));
        return new UsernamePasswordAuthenticationToken(userDetails, "", userDetails.getAuthorities());
    }

public String generateJwtToken(Authentication authentication) {
        Claims claims = Jwts.claims().setSubject(String.valueOf(authentication.getPrincipal()));
        claims.put("roles", authentication.getAuthorities());
        Date now = new Date();
        return Jwts.builder()
                .setClaims(claims)
                .setIssuedAt(now)
                .setExpiration(new Date(now.getTime() + accessTokenValidTime))
                .signWith(SignatureAlgorithm.HS256, secretKey)
                .compact();
    }

public boolean isValidToken(String token) {
        try {
            Claims claims = getClaimsFormToken(token);
            return !claims.getExpiration().before(new Date());
        } catch (JwtException | NullPointerException exception) {
            return false;
        }
    }
}
```

문제

번호	문제	정답	난이도	해설	관련학습보기
2	**웹에서 사용되는 JSON 형식의 토큰에 대한 표준 규격으로서, 주로 사용자의 인증(authentication) 또는 인가(authorization) 정보를 서버와 클라이언트 간에 안전하게 주고 받기 위해서 사용되는것은?** ① IAM ② JWT ③ Socket ④ SSL	2	하	JWT는 웹에서 사용되는 JSON 형식의 토큰에 대한 표준규격이라는 의미로서 JSON Web Token의 약어입니다.	
3	**IAM에서 사용해 AWS 서비스와 상호 작용 하기 위한 IAM 사용자가 제공해야 하는 것은 무엇인가?** ① Access Key ② ID ③ Password ④ Role	2.	중	IAM 서비스와 상호작용하기 위해서 IAM 사용자는 Access Key를 제공해야 합니다.	
4	**IAM에서 인증된 계정 사용자에 대하여 특정한 권한이 있음이 생성할 수 있는 IAM 자격 증명** ① 신분 ② 역할 ③ 직급 ④ 부서	2	중	인가(Authorization)에 대한 설명으로서, IAM으로 인증된 사용자는 세부적으로 어떠한 권한을 가지는지 “역할”을 통해서 설정할 수 있습니다.	
5	**계정보안 서비스는 새 IAM 사용자가 생성되면 원칙적으로 대부분의 액세스 권한을 허용 또는 권한 중 어떠한 것으로 설정하는가?** ① 허용 ② **거부**	2	하	“비명시적 거부“라 합니다. 원칙적으로 대부분의 액세스 권한은 거부로 설정되어 있으며, 허용하기를 원할경우 명시적으로 허용하도록 설정하여야 합니다.	

문제

번호	문제	정답	난이도	해설	관련학습보기
7	JWT는 구분자(separator)인 "."을 이용하여 4가지 부분으로 구성되어 있다. ① O ② X	2	하	JWT는 구분자를 기준으로 3가지 부분으로 분할됩니다.	
8	JWT에서 사용자의 정보나 데이터에 해당하는 issuer, expires, username, role 등의 정보가 담기는 부분은 어디인가? ① header ② payload ③ body ④ signature	2	상	JWT에서 사용자정보가 담기는 부분은 payloa입니다. Body는 JWT에서 존재하지 않습니다.	
9	JWT에서 사용할 수 있는 알고리즘 방식중 비대칭키 암호화방식에 해당하는것은? ① HS256 ② RS256	2	상	RS256 은 RSA 알고리즘에 기반한 것이므로 비대칭키 암호화 방식에 해당합니다.	
10	스프링시큐리티에서 JWT를 사용하여 로그인 인증 처리를 할 때, 사용자의 ID / PW 처리를 위해서 사용하는 클래스 명칭은? ① UserInfoService ② UserDataService ③ UserDetailService ④ UserAuthenticationService	2	상	SpringSecurity에서 사용자 로그인 인증과정에서 ID/PW를 처리하기 위한 것은 UserDetail 클래스입니다	

학습정리

- **IAM**
  - 계정에 대한 공유 엑세스 : 암호나 액세스 키를 공유하지 않고도 AWS 계정의 리소스를 관리하고 사용할 수 있는 권한을 다른 사람에게 부여 가능
  - 세분화된 권한 : 리소스에 따라 여러 사람 또는 EC2 인스턴스에서 실행되는 애플리케이션에 안전하게 제공
  - 멀티 팩터 인증(MFA) : 계정 작업을 위해 암호나 액세스 키뿐 아니라 특별히 구성된 디바이스 코드도 제공
  - 자격 증명 연동 : 기업 네트워크나 인터넷 자격 증명 공급자와 같은 다른 곳에 이미 암호가 있는 사용자에게 AWS 계정에 대한 임시 액세스 권한 부여 가능
  - IAM 액세스 방식 : AWS Management Console, AWS 명령줄 도구 (CLI, Window용 PowerShell), AWS SDK, IAM HTTPS API

학습정리

- **JWT**
  - JWT는 사용자 인증을 위해 사용되는 open standard(RFC 7519)
  - JSON 포맷을 이용하여 self-contained 방식으로 사용자에 대한 정보를 저장하는 claim 기반 web token
  - IDP(identity provider)가 사용자의 정보를 담은 내용에 서명함으로써 토큰을 생성하고, 유저가 서버에 요청할 때 사용하도록 하고, 서버는 토큰의 integrity와 authenticity를 보장함.(유저가 전송하는 데이터를 숨기는 것보다는, 유저를 인증하는데 집중한다)
  - JWT는 token based stateless authentication 방식으로서, 서버에서 해당 정보를 저장할 필요가 없는 특징이 있어, 기존 session 기반 토큰의 단점을 보완함.

참고문헌

- http://www.wordpress.com
- http://aws.mazon.com
- JSON Web Tokens - jwt.io
- 임철수, "클라우드 컴퓨팅 보안기술", 정보보호학회지 정보보호 특집
- 앤서니 T, "미래코드 클라우드 컴퓨팅", 전자신문사
- 편집부, "클라우드 컴퓨팅 차세대 컴퓨팅 기술", 데이코 산업연구소
- https://spring.io/projects/spring-security/
- IAM이란 무엇입니까? - AWS Identity and Access Management (amazon.com)