

# Microservice Architecture 기반 JWT 인증 IoT 서버 연구

한국산업기술대학교 스마트팩토리 융합학과  
김태형, 서준민, 송예리

# TABLE OF CONTENTS

I. 프로젝트 목표

II. Cloud

III. Microservice Architecture (MSA)

IV. JSON Web Token (JWT) & Authentication

V. 프로젝트 결과

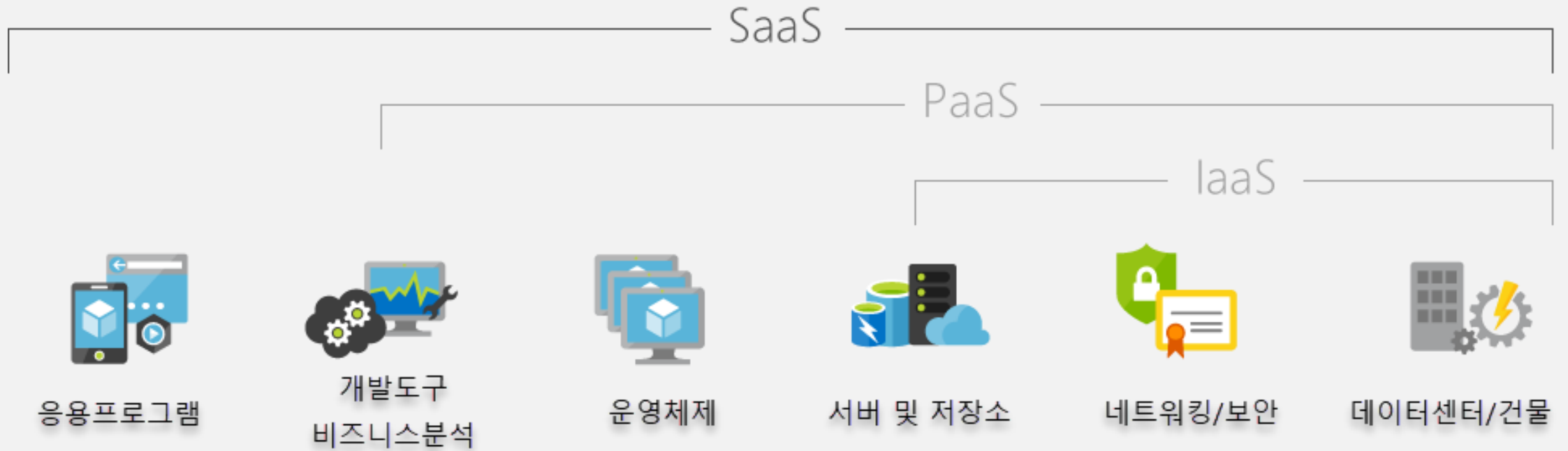
VI. 결론

# I. 프로젝트 목표

- JWT를 이용해 인증을 수행하는 Microservice Architecture  
클라우드 어플리케이션 개발

## II. Cloud

### 1) SAAS, PAAS, IAAS



## ***II. Cloud***

### 2) Cloud 어플리케이션 장점

- Cost
- Speed
- Global scale
- Productivity
- Performance
- Reliability

## II. Cloud

### 2) Cloud 어플리케이션 디자인 고려사항

- Demand
- Datacenters
- Operations
- Scale
- Failure
- Machine loss

## II. Cloud

### 2) Cloud 어플리케이션의 특징

- 과거/현재 어플리케이션의 특징은 다음과 같음

Feature	Past	Present
Client	Enterprise/Intranet	Public/Internet
Demand	Stable (small)	Dynamic (small > massive)
Datacenter	Single tenant	Multi-tenant
Operations	People (expensive)	Automation (cheap)
Scale	Up via few reliable (expensive ) PCs	Out via lots of (cheap) commodity PCs
Failure	Unlikely but possible	Very likely
Machine loss	Catastrophic	Normal (no big deal)

## II. Cloud

### 2) Cloud 어플리케이션의 특징

- 어플리케이션에 따라 에러에 대처하는 방법 상이

Example	Past	Present
Exceptions	Catch, swallow & keep running	Crash & restart
Communication	In order Exactly once	Out of order Client must retry & servers must be idempotent



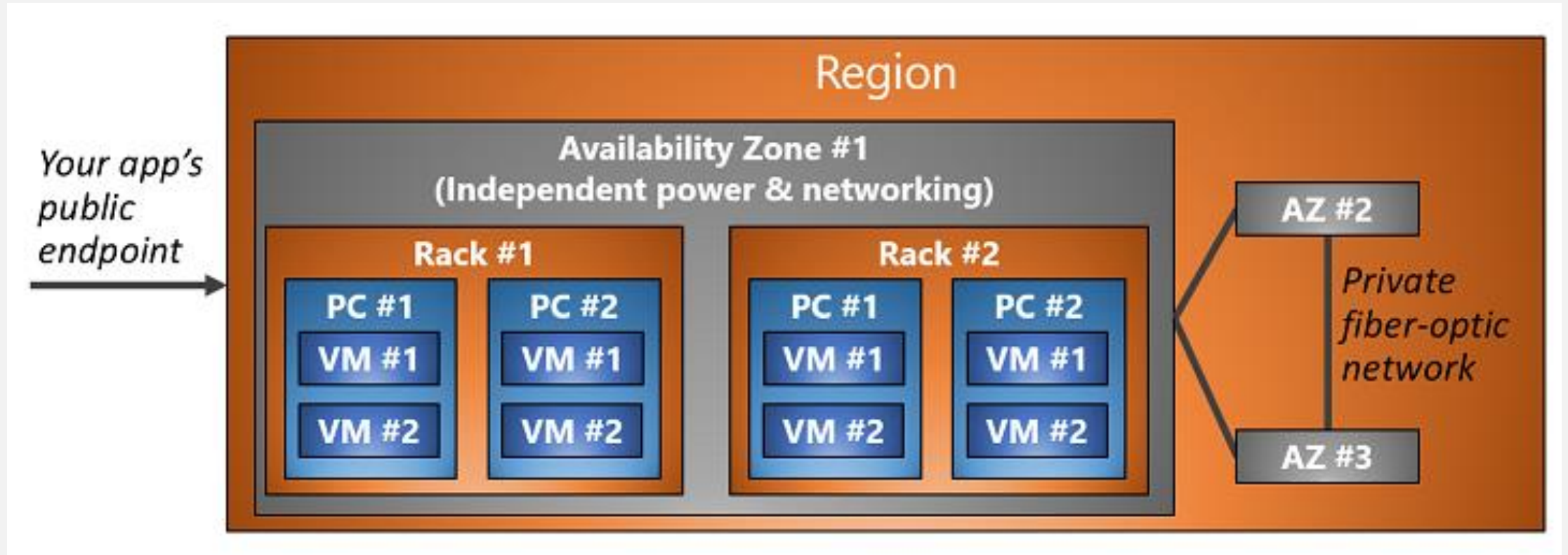
## II. Cloud

### 2) Cloud어플리케이션의 특징

- 클라우드 어플리케이션은 '**장애 대처**'가 키 포인트
  - 서비스 인스턴스 장애 발생 이유
    - 1) 개발자: 처리되지 않는 예외
    - 2) 데브옵스: 서비스 인스턴스 스케일 다운
    - 3) 데브옵스: 서비스 코드 업데이트
    - 4) 오케스트레이터: 서비스 인스턴스 이동
    - 5) 하드웨어 장애: 파워 서플라이, 과열, 하드 디스크 등등...

## II. Cloud

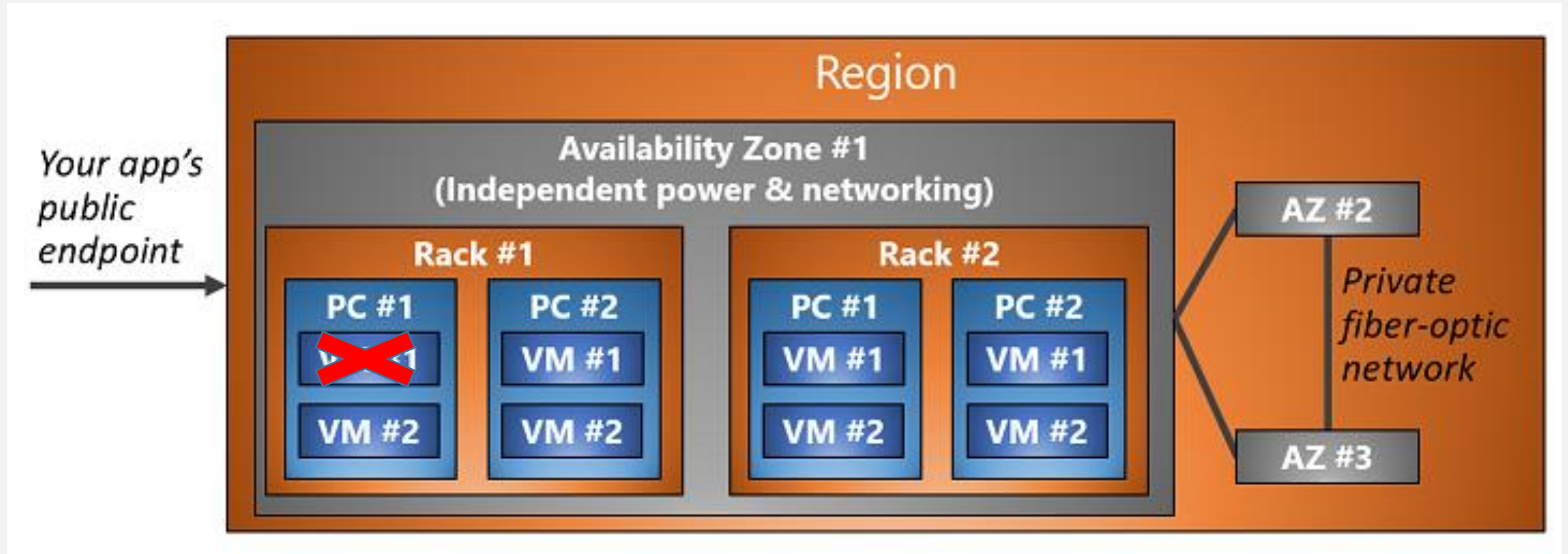
### 4) Cloud 데이터 센터 구조



## II. Cloud

### 4) Cloud 데이터 센터 구조

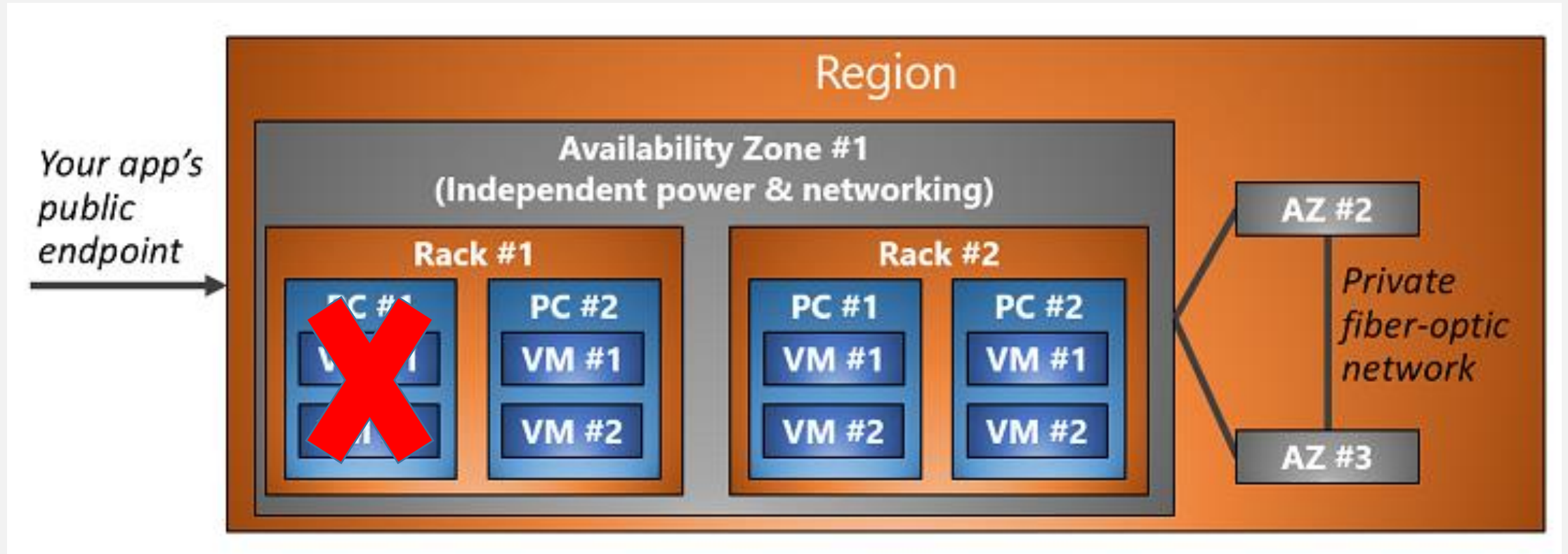
- 하나의 VM 정시 시, 서비스 장애 발생



## II. Cloud

### 4) Cloud 데이터 센터 구조

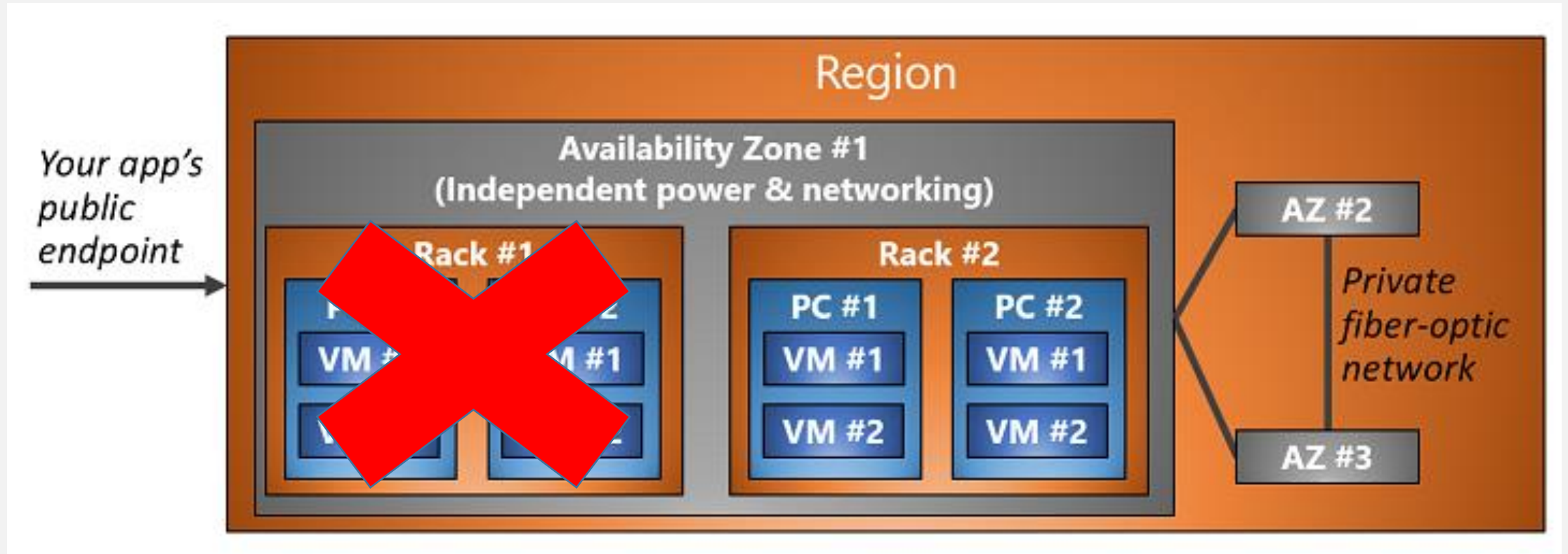
- 하나의 PC 정지 시, 속해 있는 모든 VM 장애 발생



## II. Cloud

### 4) Cloud 데이터 센터 구조

- 하나의 Rack 정지 시, 속해 있는 모든 PC 장애 발생





## II. Cloud

### 4) Cloud 데이터 센터 구조

- 하나의 AZ 정지 시, 속해 있는 모든 Rack 장애 발생



## II. Cloud

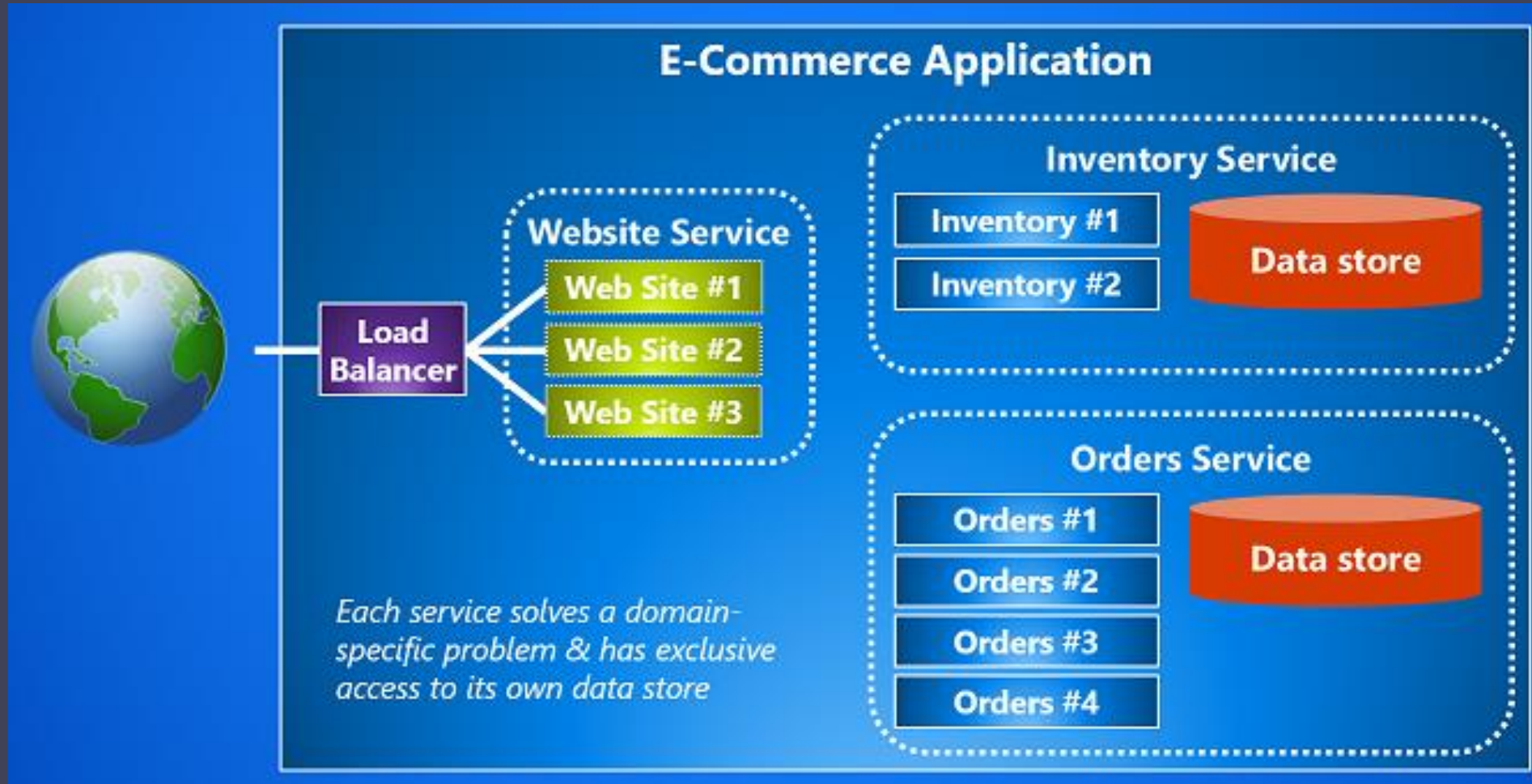
### 4) Cloud 데이터 센터 구조

- 하나의 리전 정지 시, 속해 있는 모든 AZ 장애 발생



# III. Microservice Architecture (MSA)

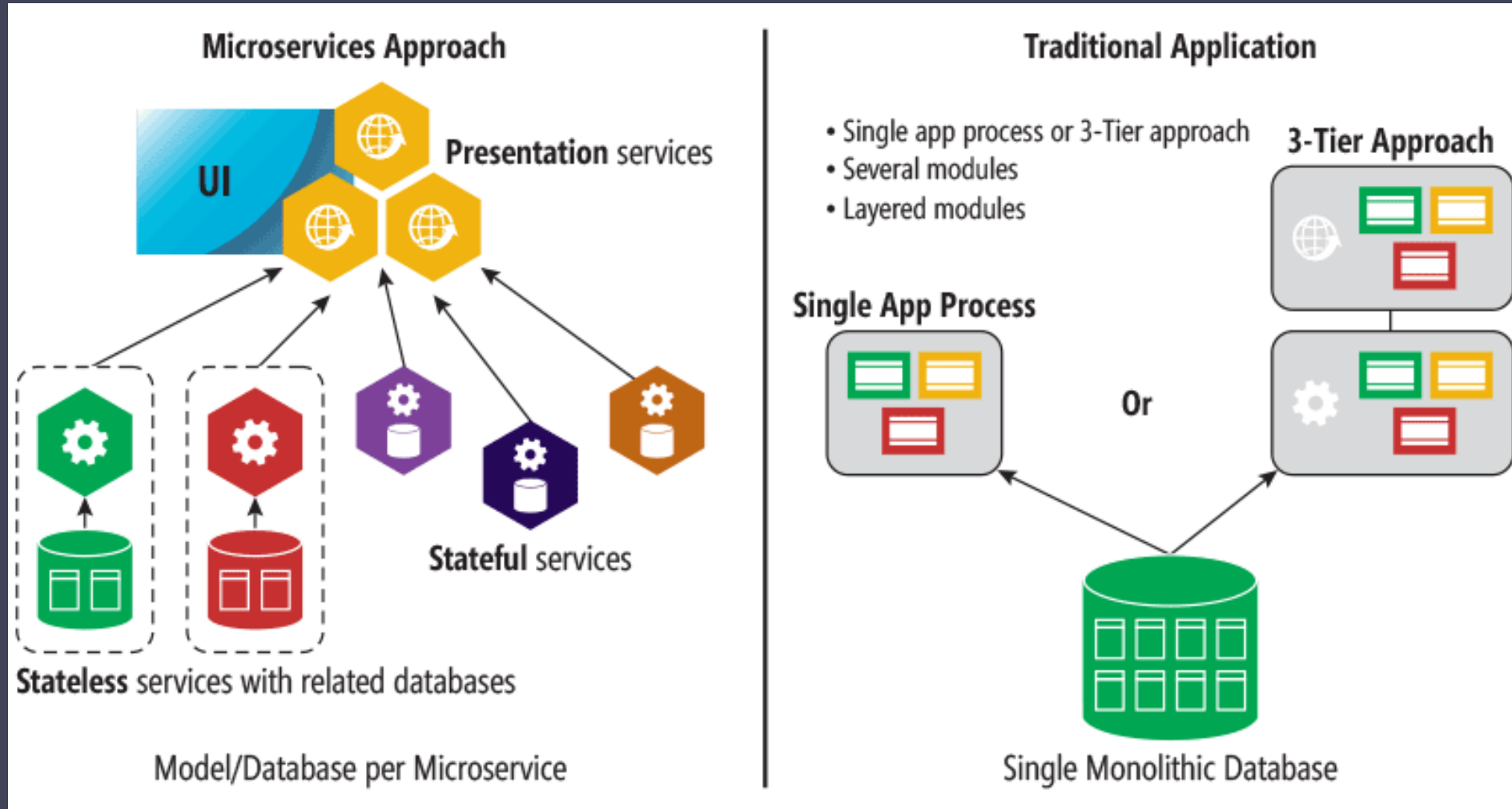
## Microservice Architecture overview





# III. Microservice Architecture (MSA)

## MSA vs Monolithic

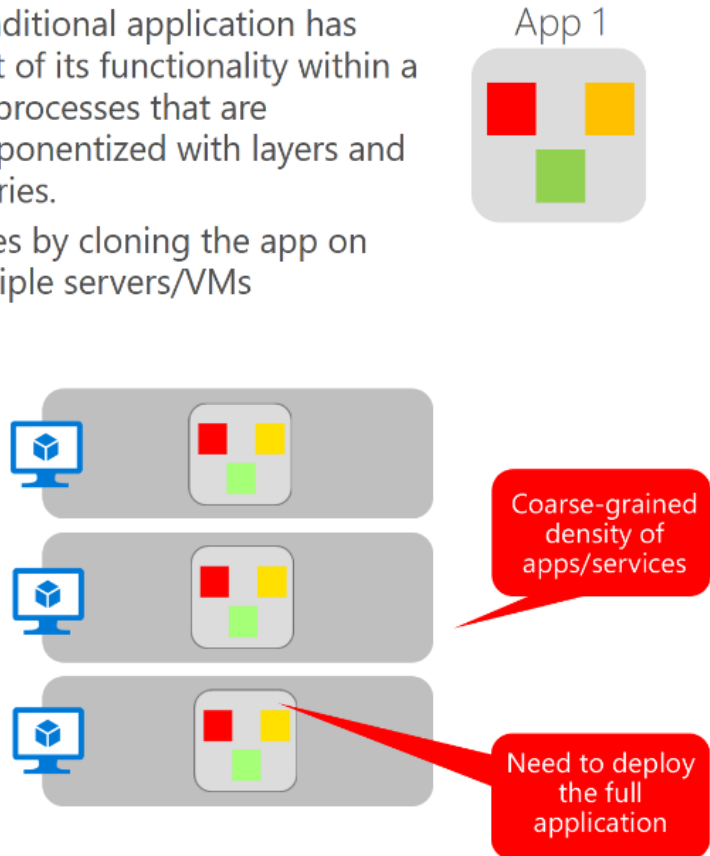


# III. Microservice Architecture (MSA)

## MSA vs Monolithic

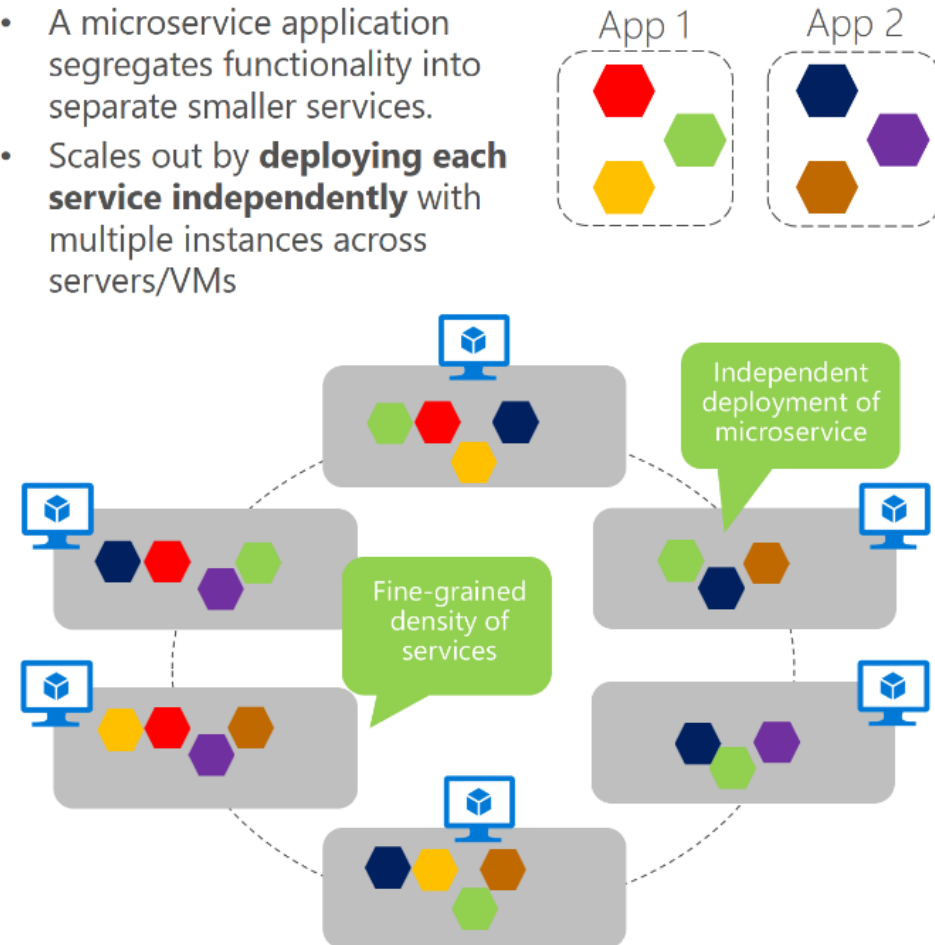
### Monolithic deployment approach

- A traditional application has most of its functionality within a few processes that are componentized with layers and libraries.
- Scales by cloning the app on multiple servers/VMs



### Microservices application approach

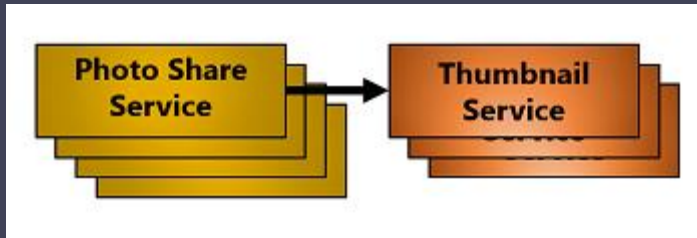
- A microservice application segregates functionality into separate smaller services.
- Scales out by **deploying each service independently** with multiple instances across servers/VMs



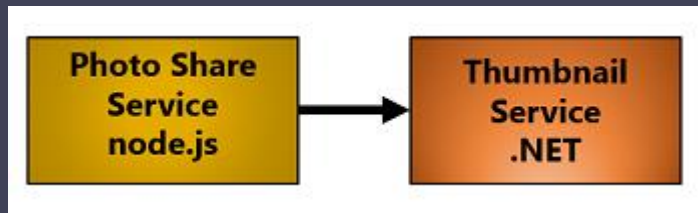
### III. Microservice Architecture (MSA)

#### MSA 장점

- 인스턴스 별 유연성 있는 스케일링 가능
  - 부하가 걸리는 인스턴스만 독자적으로 스케일 업/다운 가능



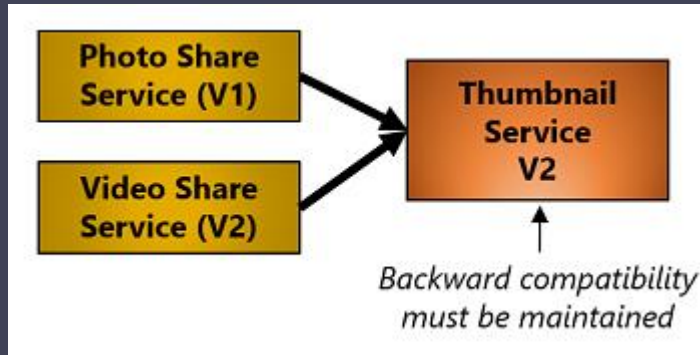
- 기반 프레임워크 다양화 가능
  - 각 인스턴스마다 다른 프레임워크를 사용한 개발 가능



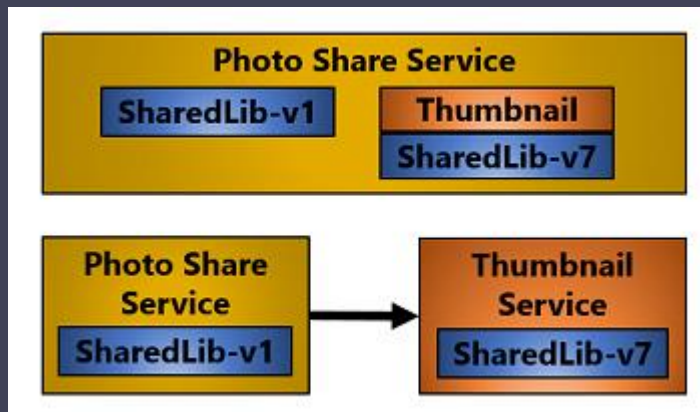
### III. Microservice Architecture (MSA)

#### MSA 장점

- 다양한 API 버전의 서비스 동시 구동 가능



- 상이한 버전의 공통 라이브러리 사용 가능



# III. Microservice Architecture (MSA)

## MSA 단점 - 복잡도 증가



# III. Microservice Architecture (MSA)

## MSA 단점 - 복잡도 증가

```
1 version: "3.7"
2
3 services:
4   nginx:
5     build:
6       context: ./services/nginx
7     ports:
8       - "8083:80/tcp"
9     depends_on:
10      - auth
11      - client
12      - mqtt
13     networks:
14      - frontend
15      - backend
16
17   client:
18     build:
19       context: ./services/client
20     expose:
21       - "8080"
22     environment:
23       - BASE_URL=www.fakeurl.com
24     networks:
25       - frontend
26
27   auth:
28     build:
29       context: ./services/auth
30     expose:
31       - "5000"
32     environment:
33       - FLASK_APP=projects
34       - FLASK_ENV=development
35       - DATABASE_URI=postgres://postgres:postgres@auth-db:5432/users_dev
36     depends_on:
37       - postgres
38     networks:
39       - backend
40
41   auth-db:
42     build:
43       context: ./services/auth-db
44     environment:
45       - POSTGRES_USER=postgres
46       - POSTGRES_PASSWORD=postgres
47     networks:
48       - backend
49
50   iot:
51     build:
52       context: ./services/iot
53     environment:
54       - FLASK_APP=projects
55       - FLASK_ENV=development
56       - MONGO_DB_URI=mongodb://iot-db:27017
57       - MONGO_DB_USERNAME=root
58       - MONGO_DB_PASSWORD=password
59       - MONGO_DB_NAME=dev_table
60     depends_on:
61       - iot-db
62     networks:
63       - backend
64
65   mqtt:
66     build:
67       context: ./services/mqtt
68     environment:
69       - APP_ENV=development
70       - MQTT_ENTRYPOINT=seujeum.iptime.org
71       - MQTT_PORT=15073
72       - MONGO_DB_URI=mongodb://iot-db:27017
73       - MONGO_DB_USERNAME=root
74       - MONGO_DB_PASSWORD=password
75       - MONGO_DB_NAME=dev_table
76     depends_on:
77       - mqtt-broker
78       - iot-db
79     networks:
80       - backend
81
82   mqtt-broker:
83     image: eclipse-mosquitto:latest
84     expose:
85       - "1883"
86       - "9001"
87     ports:
88       - "8081:1883/tcp"
89     networks:
90       - backend
91
92   iot-db:
93     image: mongo:latest
94     restart: always
95     expose:
96       - "27017"
97     ports:
98       - "8082:27017/tcp"
99     environment:
100      - MONGO_INITDB_ROOT_USERNAME=root
101      - MONGO_INITDB_ROOT_PASSWORD=password
102     networks:
103       - backend
104
105   networks:
106     frontend:
107     backend:
```

각 서비스 별로 배포할 때 필요한  
Env나 의존성이 상이

# III. Microservice Architecture (MSA)

## MSA 단점 - 의존성 파편화

The screenshot displays a Visual Studio Code interface with a project structure on the left, a code editor in the center, and a terminal at the bottom. The project structure shows a 'services' directory with subdirectories 'auth', 'iot', and 'mqtt'. Each subdirectory contains a 'requirements.txt' file. The 'auth' directory is highlighted with a red box, and the 'iot' and 'mqtt' directories are also highlighted with red boxes. The 'requirements.txt' files are listed in the center editor, showing various dependencies like 'alembic', 'atomicwrites', 'attrs', 'bcrypt', 'click', 'colorama', 'Flask', 'Flask-Bcrypt', 'Flask-Cors', 'Flask-JWT-Extended', 'Flask-Migrate', 'Flask-SQLAlchemy', 'itsdangerous', 'Jinja2', 'MarkupSafe', 'paho-mqtt', 'PyJWT', 'pymongo', 'python-dateutil', 'six', and 'Werkzeug'. The terminal at the bottom shows the command 'pip requirements' and the output 'DONE Compiled successfully in 990ms'. The status bar at the bottom indicates 'Python 2.7.15 64-bit' and 'pip requirements'.

인증 서비스

IoT DB 조회 서비스

MQTT DB 저장 서비스

# III. Microservice Architecture (MSA)

## MSA 단점 - 의존성 파편화

The screenshot displays the Visual Studio Code interface with three open files, each containing a `requirements.txt` for a different microservice. This illustrates the problem of dependency fragmentation in a Microservice Architecture (MSA).

- Left Panel (File Explorer):** Shows the project structure. The `requirements.txt` files are located in the `services/auth`, `services/iot`, and `services/mqtt` directories, each highlighted with a red box.
- Middle Panel (requirements.txt - sensor-cloud-app):** Lists dependencies for the `sensor-cloud-app` service, including `Flask-Bcrypt`, `Flask-Cors`, `Flask-JWT-Extended`, `Flask-Migrate`, `Flask-SQLAlchemy`, `itsdangerous`, `Jinja2`, `Mako`, `MarkupSafe`, `more-itertools`, `pluggy`, `psycopy2`, `py`, `pycparser`, `PyJWT`, `pytest`, `python-dateutil`, `python-editor`, `six`, `SQLAlchemy`, and `Werkzeug`.
- Right Panel (requirements.txt - sensor-cloud-app):** Lists dependencies for the `sensor-cloud-app` service, including `Click`, `Flask`, `Flask-Cors`, `Flask-JWT-Extended`, `itsdangerous`, `Jinja2`, `MarkupSafe`, `paho-mqtt`, `PyJWT`, `pymongo`, `python-dateutil`, `six`, and `Werkzeug`.
- Bottom Panel (Terminal):** Shows the command `4: node` and the output `DONE Compiled successfully in 990ms`. Below this, it indicates the application is running at `http://localhost:8888/` and `http://10.0.75.1:8888/`.

The status bar at the bottom indicates the current environment is `Python 2.7.15 64-bit` and the file encoding is `UTF-16 LE`.



# III. Microservice Architecture (MSA)

## MSA 단점 - 의존성 파편화

The screenshot displays three separate 'requirements.txt' files in Visual Studio Code, illustrating the problem of dependency fragmentation in a Microservice Architecture (MSA). Red and purple arrows point to identical dependency entries across different files, highlighting redundancy.

**Left File (requirements.txt):**

```
1 alembic==1.0.3
2 atomicwrites==1.2.1
3 attrs==18.2.0
4 bcrypt==3.1.4
5 cffi==1.11.5
6 Click==7.0
7 colorama==0.4.1
8 Flask==1.0.2
9 Flask-Bcrypt==0.7.1
10 Flask-Cors==3.0.7
11 Flask-JWT-Extended==3.13.1
12 Flask-Migrate==2.3.0
13 Flask-SQLAlchemy==2.3.2
14 itsdangerous==1.1.0
15 Jinja2==2.10
16 Mako==1.0.7
17 MarkupSafe==1.1.0
18 more-itertools==4.3.0
19 pluggy==0.8.0
20 pycogp2==2.7.6.1
21 py==1.7.0
22 pycparser==2.19
23 PyJWT==1.6.4
24 pytest==4.0.1
25 python-dateutil==2.7.5
26 python-editor==1.0.3
27 six==1.11.0
28 SQLAlchemy==1.2.14
29 Werkzeug==0.14.1
30
```

**Middle File (requirements.txt):**

```
1 Click==7.0
2 Flask==1.0.2
3 Flask-Cors==3.0.7
4 Flask-JWT-Extended==3.13.1
5 itsdangerous==1.1.0
6 Jinja2==2.10
7 MarkupSafe==1.1.0
8 paho-mqtt==1.4.0
9 PyJWT==1.6.4
10 pymongo==3.7.2
11 python-dateutil==2.7.5
12 six==1.11.0
13 Werkzeug==0.14.1
14
```

**Right File (requirements.txt):**

```
1 paho-mqtt==1.4.0
2 pymongo==3.7.2
3
```

**Terminal Output:**

```
문제 출력 디버그 콘솔 터미널
DONE Compiled successfully in 990ms

App running at:
- Local: http://localhost:8888/
- Network: http://10.0.75.1:8888/
```

**Bottom Status Bar:**

master Python 2.7.15 64-bit 0 0 0 0 줄 1, 열 1 공백: 4 UTF-16 LE CRLF pip requirements Formatting: X

### III. Microservice Architecture (MSA)

MSA 단점 - 신뢰성

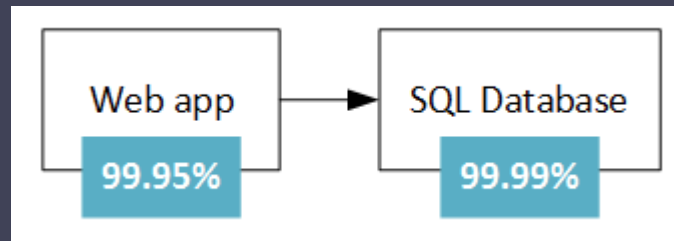
- 시스템 신뢰성

- $R_s = R_1 \times R_2 \times \dots \times R_n = \prod_{i=1}^n R_i$

- 서비스 수준 협약서(SLA)에 따른 서비스별 장애 시간

Each service's SLA	1 service	2 services	3 services	n services
99.99%	99.99%, 260s/mo	99.98%, 520s/mo	99.97%, 780s/mo	$99.99^n\%$ , $(n \times 260s)/mo$
99.999%	99.999%, 26s/mo	99.998%, 52s/mo	99.997%, 78s/mo	$99.999^n\%$ , $(n \times 26s)/mo$

- Web 어플리케이션과 SQL DB가 있는 시스템의 신뢰성

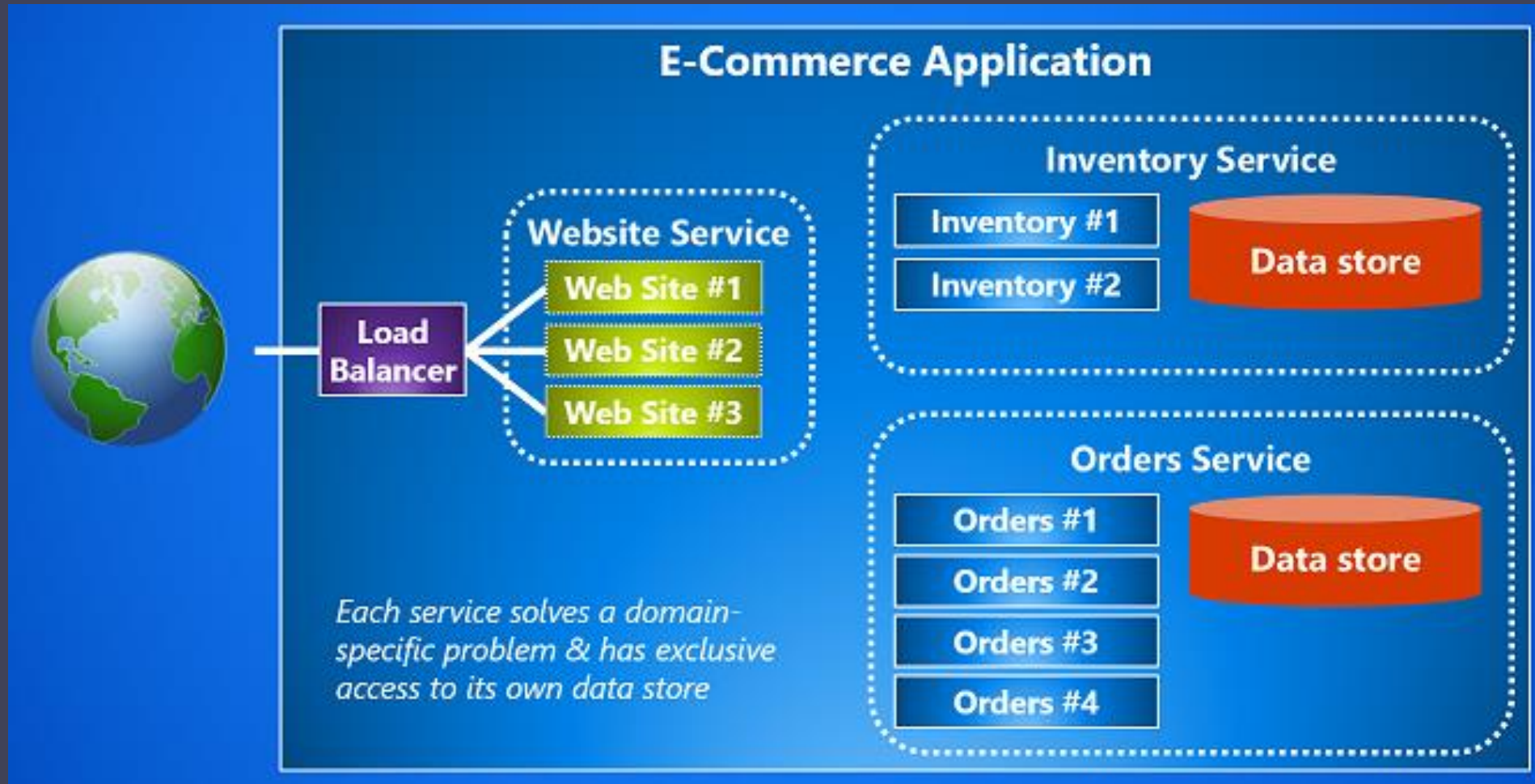


- $SLA = 99.95\% \times 99.99\% = 99.94\%$

### III. Microservice Architecture (MSA)

MSA 단점 - 신뢰성

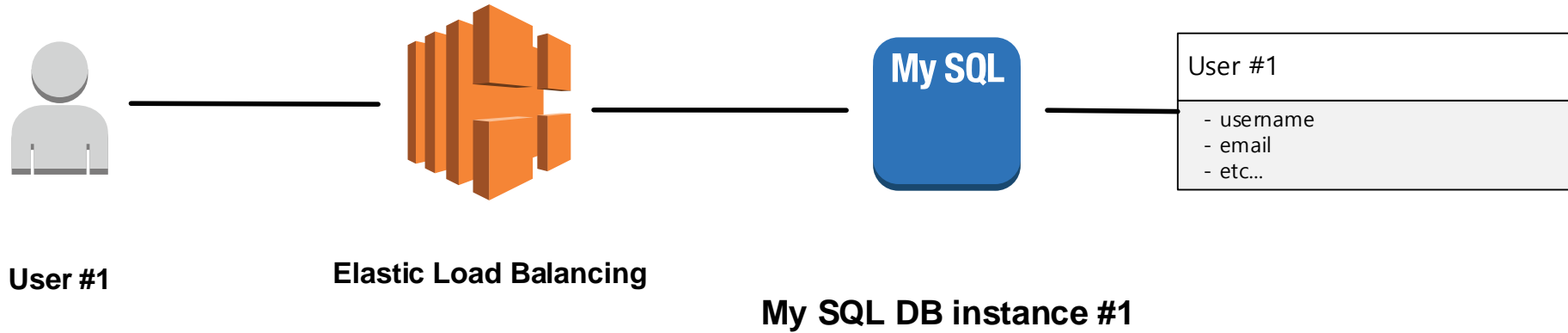
- 신뢰성을 향상시키기 위해서 같은 인스턴스를 복제할 수 있다.



# III. Microservice Architecture (MSA)

Data consistency issues

TO



# III. Microservice Architecture (MSA)

Data consistency issues

T1



User #1



Elastic Load Balancing



My SQL DB instance #1

User #1
- username - email - etc...



User #2

인스턴스 복제

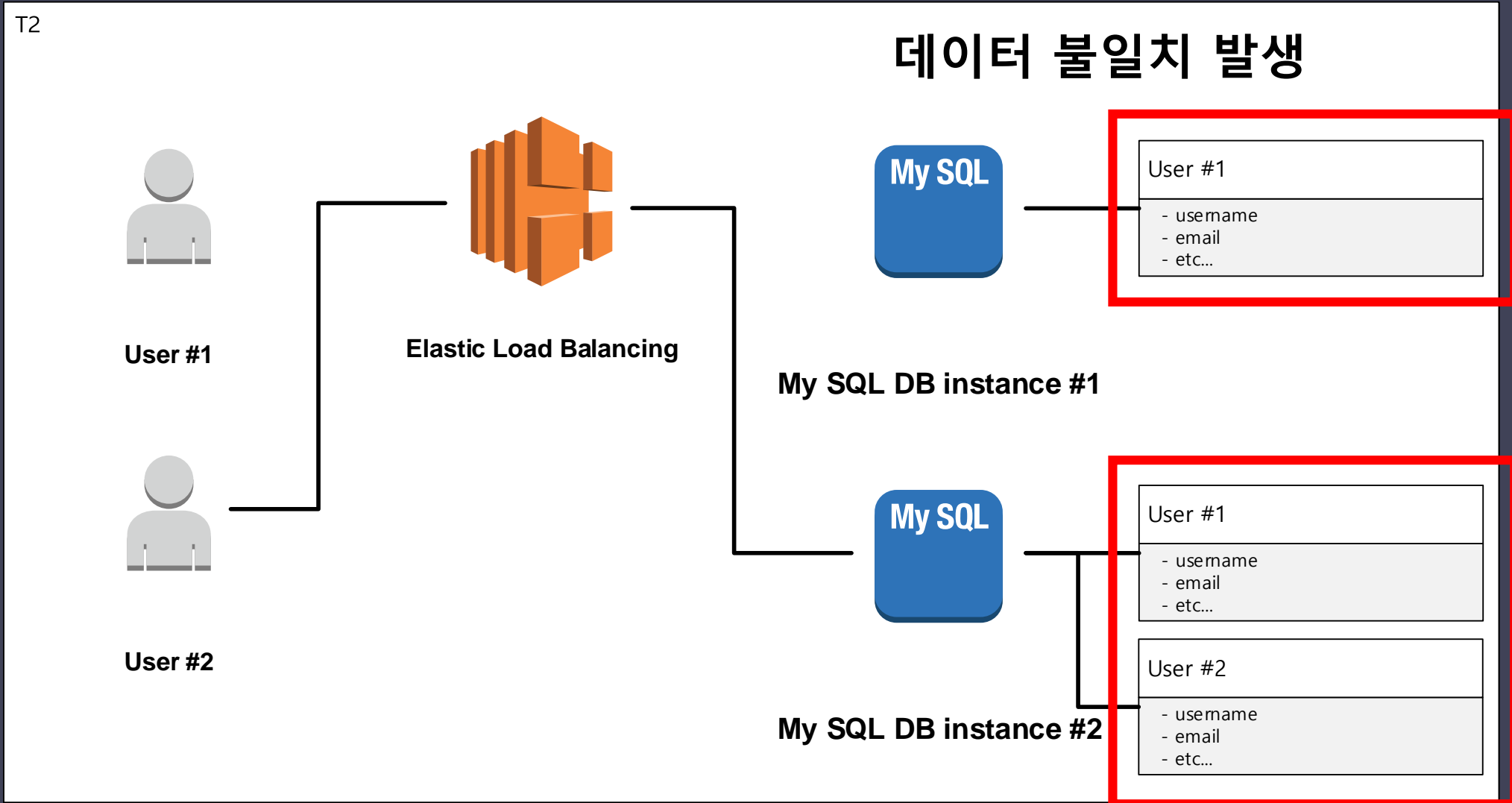


My SQL DB instance #2

User #1
- username - email - etc...

# III. Microservice Architecture (MSA)

Data consistency issues



## IV. JSON Web Token (JWT) & Authentication

### 2) JWT란?

- 다자간 암호화된(검증 가능한) 신뢰성 있는 정보 교환 컨테이너

#### Encoded

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjE5ODUyMjY0MDAwMD0RydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

#### Decoded

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}  
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "admin": true  
}  
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  secret  
)
```

Header

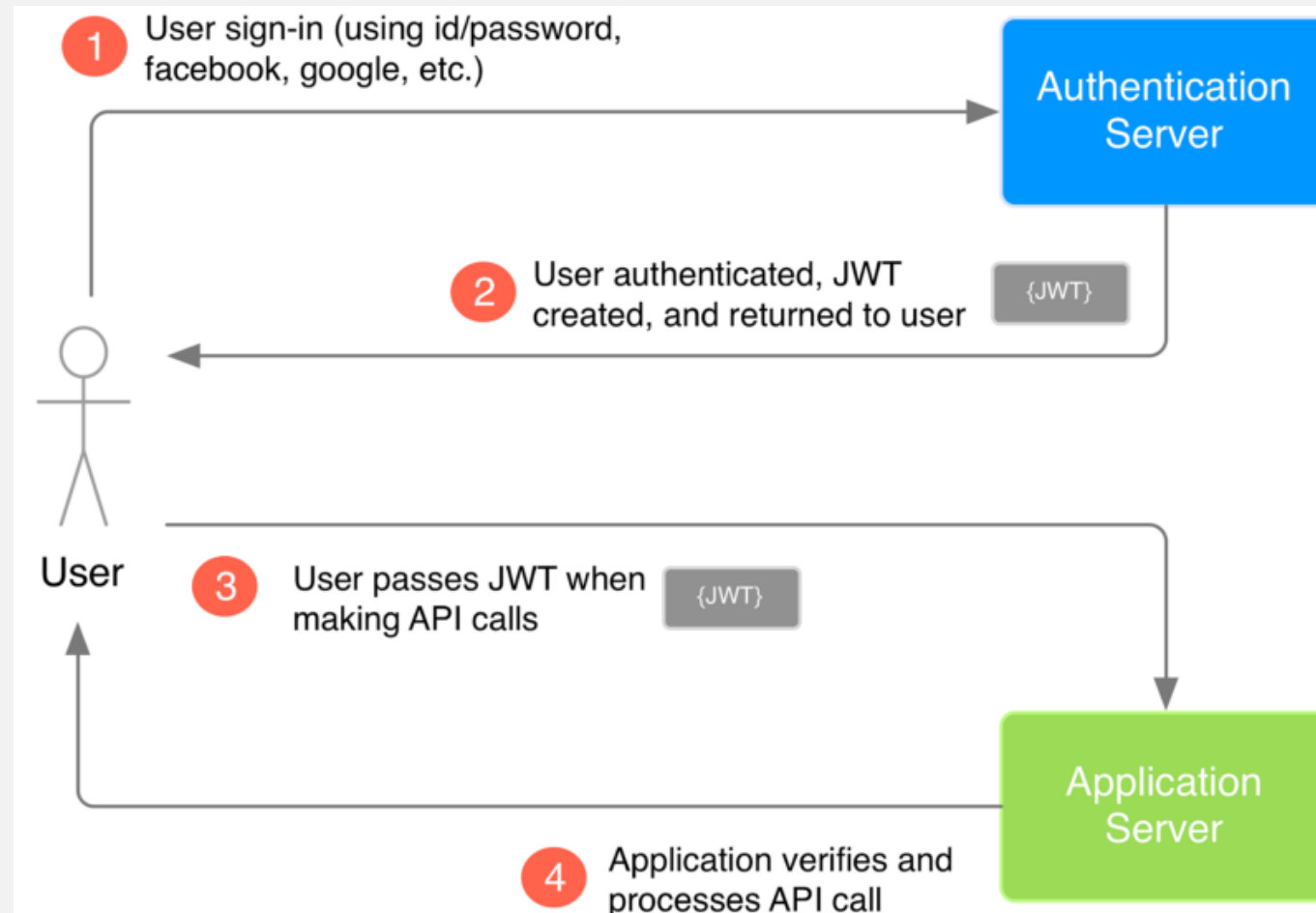
Payload

Signature

# IV. JSON Web Token (JWT) & Authentication

## 2) JWT란?

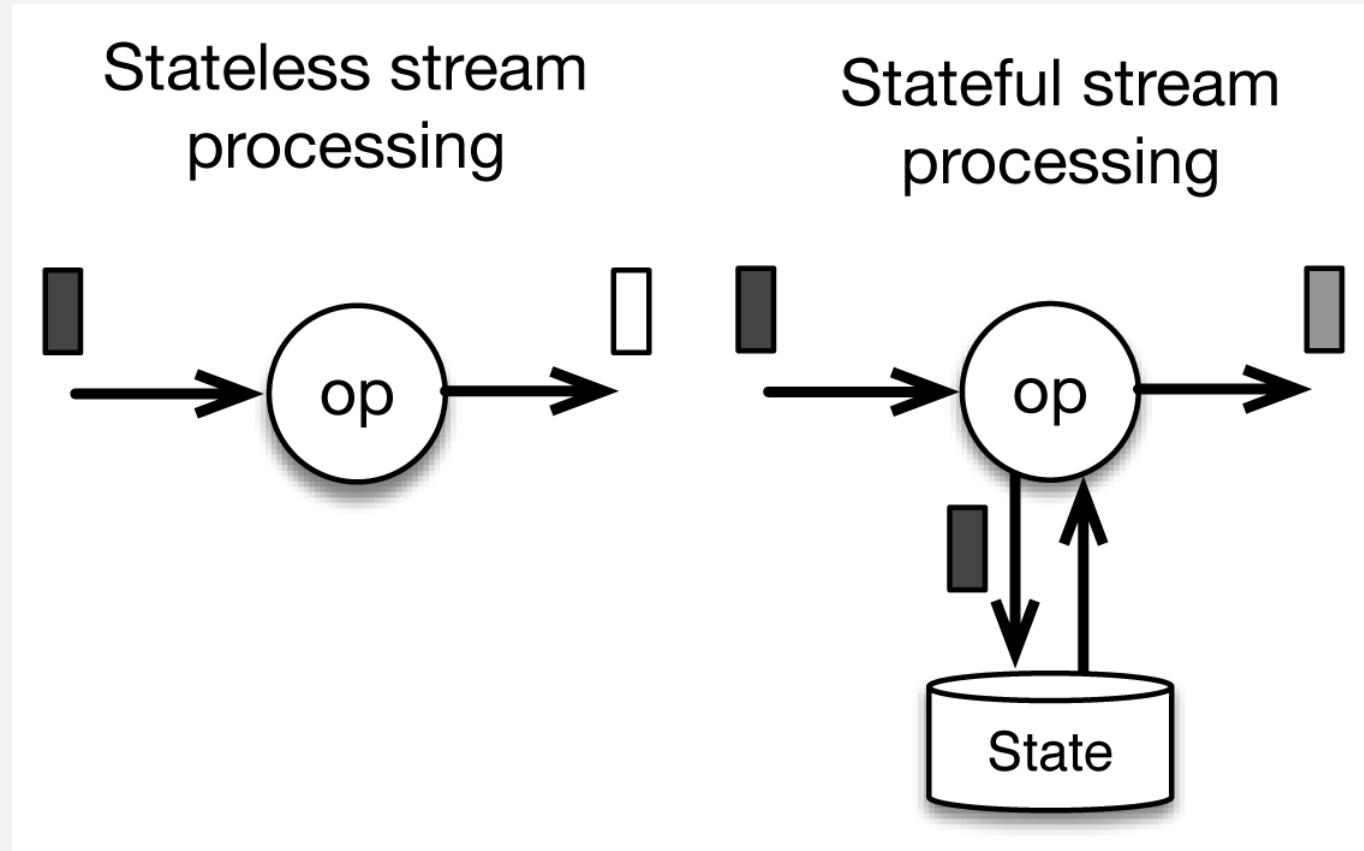
- 자가 수용적(Self-contained) 특성 때문에 Stateless "인증" 서비스에 유용하게 사용 가능





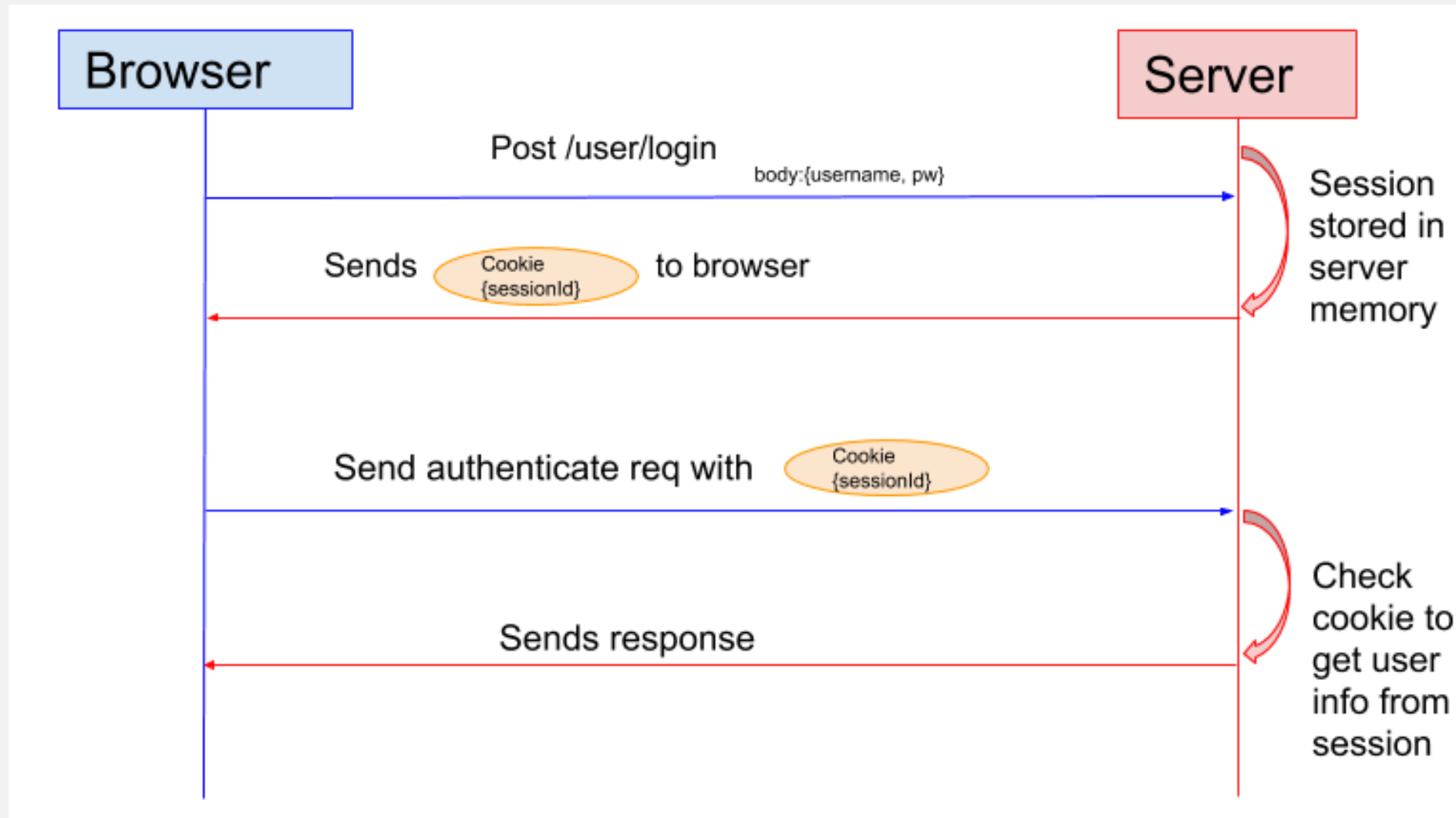
## IV. JSON Web Token (JWT) & Authentication

Stateful? Stateless?



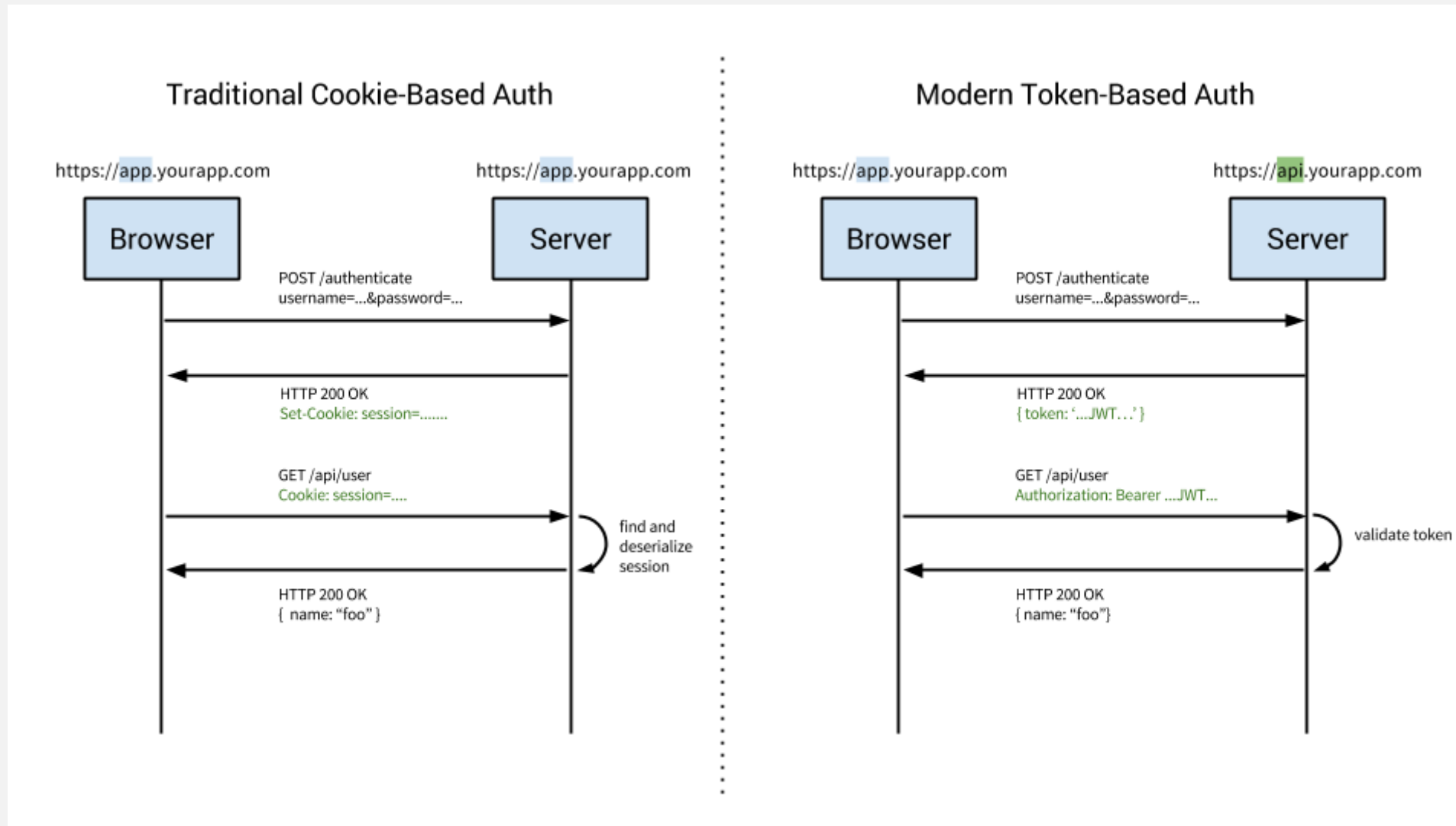
# IV. JSON Web Token (JWT) & Authentication

## Authentication & Session



# IV. JSON Web Token (JWT) & Authentication

## Authentication & Session



# IV. JSON Web Token (JWT) & Authentication

Authentication using JWT

## 로그인 라우트 (유저가 로그인을 시도하면 토큰 발행)

```
@auth_api.route('/auth/login', methods=['POST'])
def login_user():
    post_data = request.get_json()

    email = post_data.get('email')
    password = post_data.get('password')

    logger.info(f'Attempt login {email} {password}')

    response_object = {}
    try:
        user = User.query.filter_by(email=email).first()
        if user and user.check_password(password):
            auth_token = create_access_token(identity=user.id)

            if auth_token:
                response_object['message'] = 'Successfully logged in.'
                response_object['auth_token'] = auth_token

                return jsonify(response_object), 200
            else:
                response_object['message'] = 'User does not exist.'
                return jsonify(response_object), 404
        else:
            response_object['message'] = 'User does not exist.'
            return jsonify(response_object), 404
    except Exception as e:
        logger.warn(f'Unexpected error occurred!')
        logger.warn(f'err_log: {str(e)}')
        response_object['message'] = f'Unexpected error occurred!'
        return jsonify(response_object), 500
```

로그인 시도 계정 조회 및 검증

JWT 토큰 발행

패킷에 토큰 정보 삽입하여 반환

# IV. JSON Web Token (JWT) & Authentication

Authentication using JWT

## IoT 센서 데이터 조회 라우트

```
@iot_api.route('/iot/<string:topic>', methods=['GET'])
@iot_api.route('/iot/<string:topic>/latest', methods=['GET'])
@jwt_required
def get_data_single(topic):
    collection = mongo.db[topic]
    response_object = {}
    logger.info(f'get latest data {topic} in {collection}')
    try:
        data = collection.find_one(sort=[('_id', pymongo.DESCENDING)])

    except Exception as e:
        logger.warn(f'{topic} cannot found. {collection}')
        logger.warn(f'err_log: {str(e)}')
        return abort(404)

    date = data['_id'].generation_time.isoformat()
    value = data['data']
    logger.info(f'query {topic}: {date} {value}')

    response_object = {
        'topic': topic,
        'data': [
            {
                'date': date,
                'value': value
            }
        ]
    }

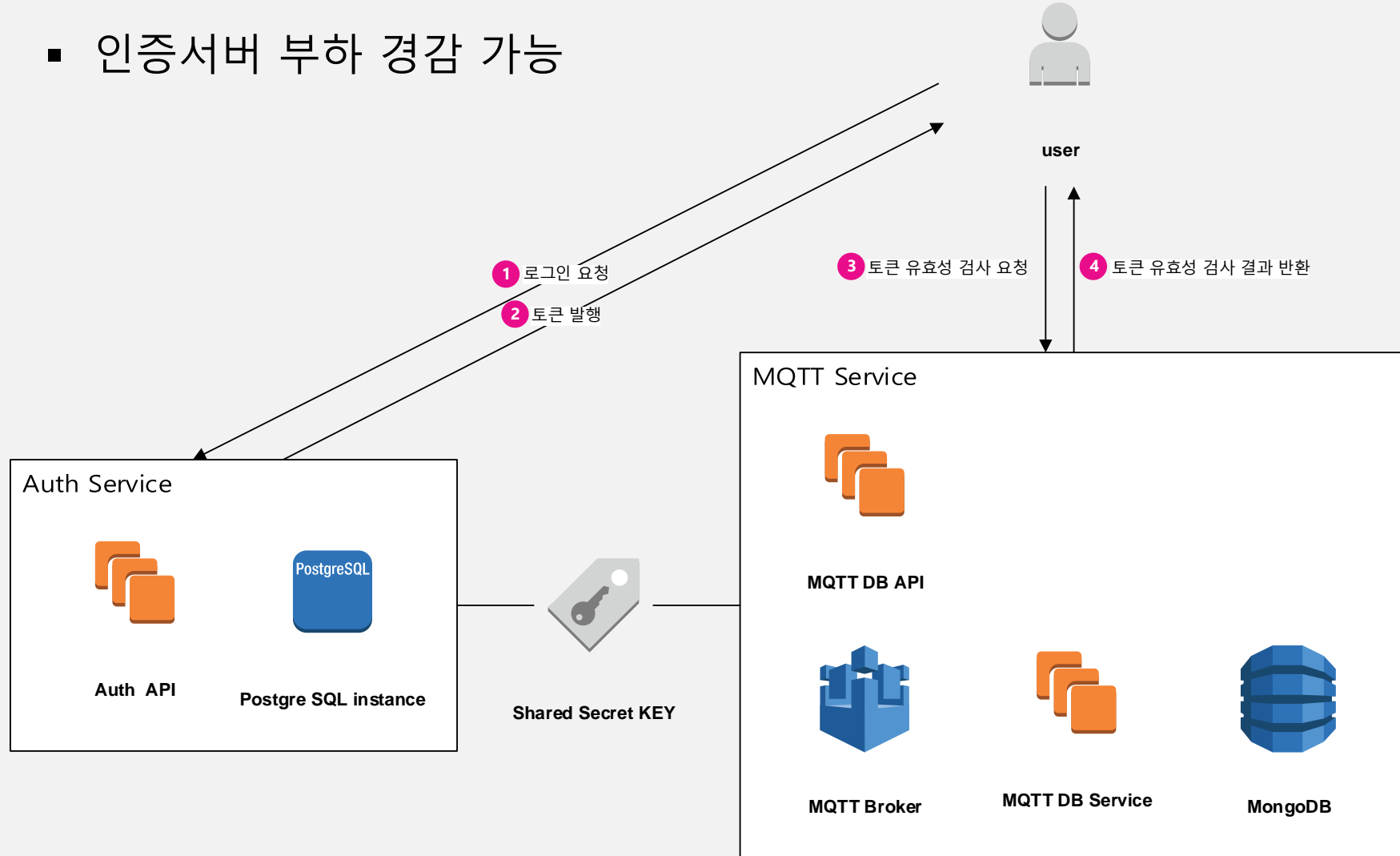
    return jsonify(response_object), 200
```

유효한 토큰이 없으면 액세스 불가

# IV. JSON Web Token (JWT) & Authentication

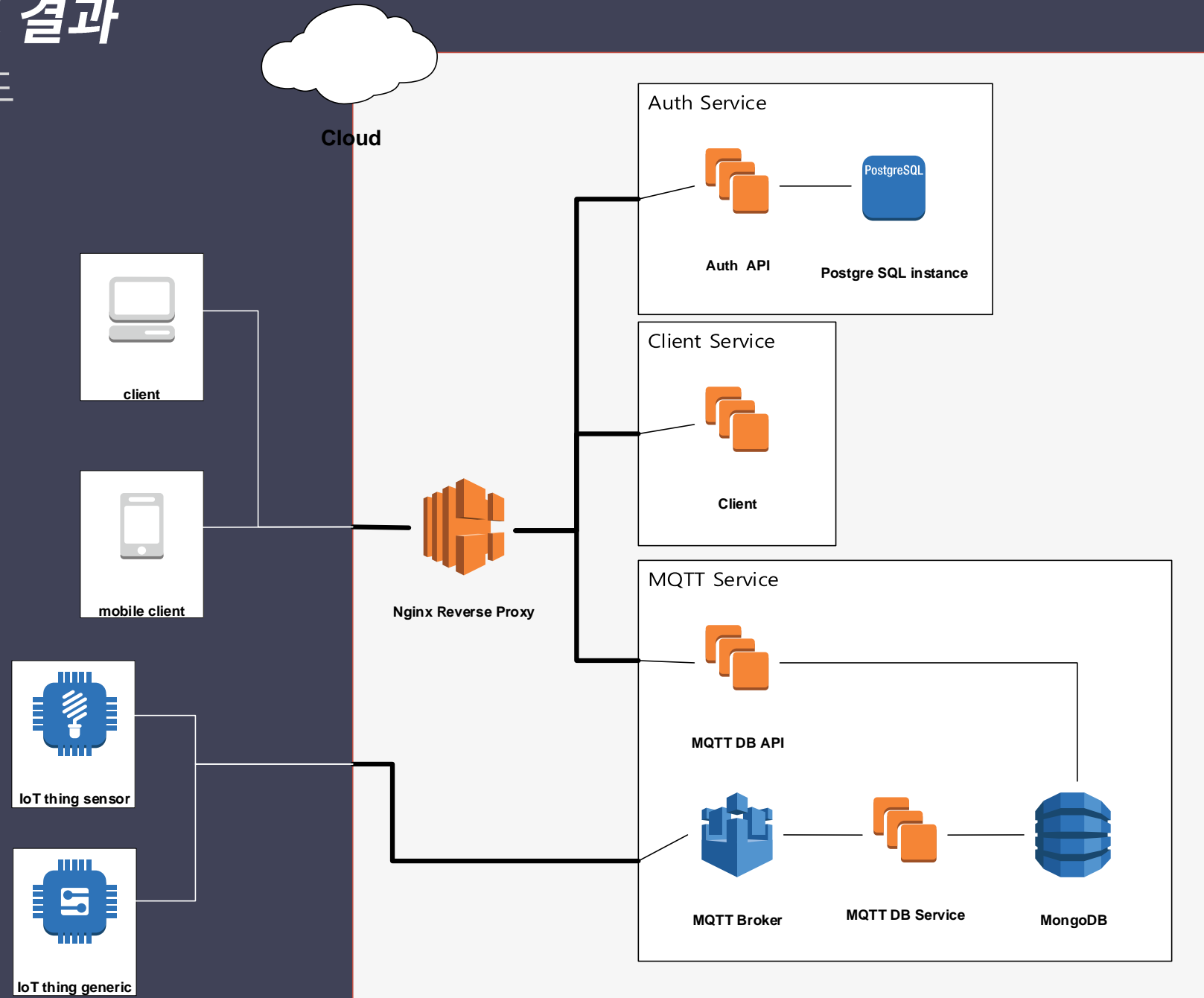
Authentication using JWT

- 모든 서비스가 암호 키를 공유, 인증 서버를 경유하지 않고 토큰 유효성 검증 가능
  - 인증서버 부하 경감 가능



# V. 프로젝트 결과

## 시스템 구성도



# V. 프로젝트 결과

## RESTful API

### Auth Service

POST

/auth/login 로그인 라우트

POST

/auth/register 사용자 계정 생성 라우트

GET

/users 전체 사용자 목록 조회 라우트

GET

/users/{user\_id} 특정 사용자 정보 조회 라우트

### IoT Service

GET

/iot/{topic}/latest 토픽에 대한 최신 데이터 조회

GET

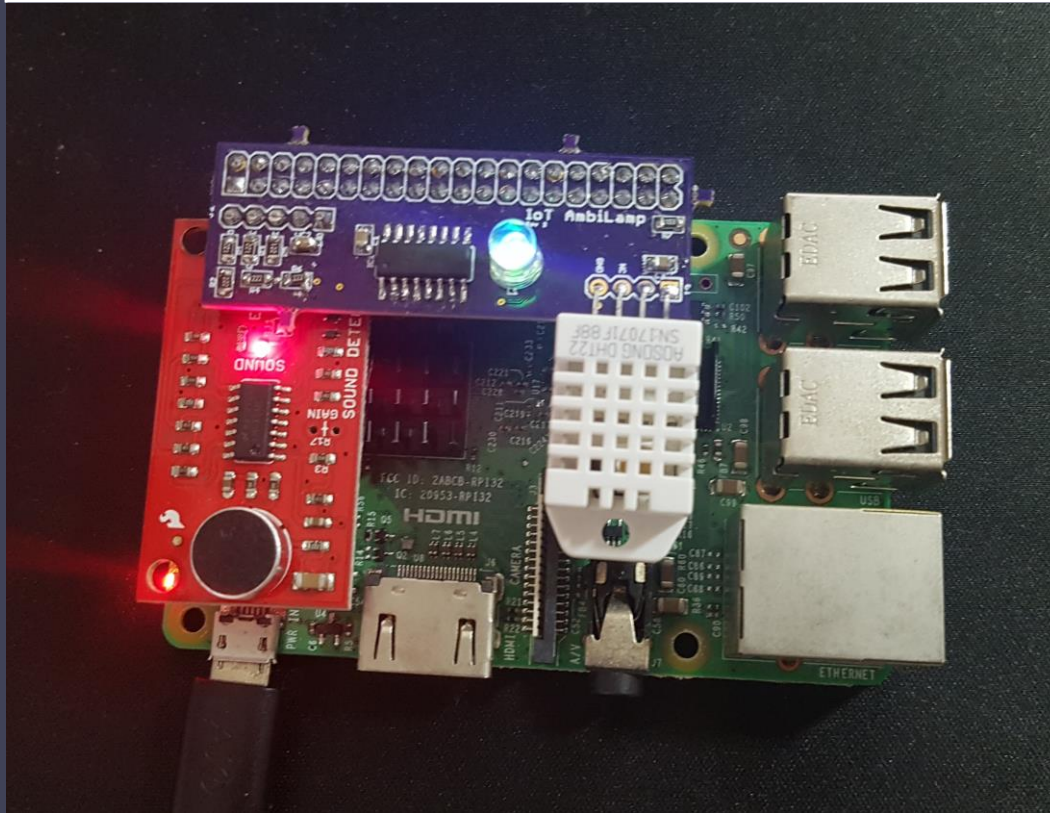
/iot/{topic}/{start\_date}/{end\_date} 토픽에 대한 기간별 데이터 조회



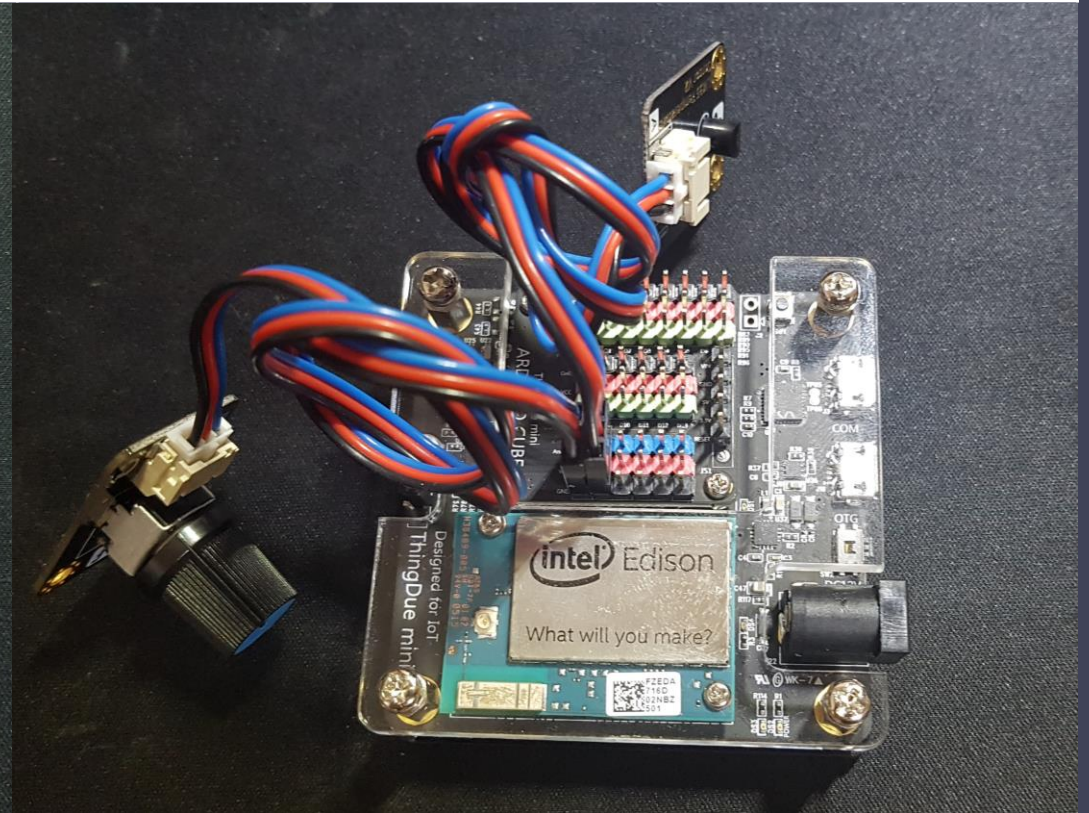
# V. 프로젝트 결과

센서

라즈베리파이 (온습도센서)



Intel Edison (온도 및 가변저항)



## 센서 데이터 요청



# V. 프로젝트 결과

전체 사용자 목록 조회 결과

## 전체 사용자 목록 요청 (미인증)

← → ↻ ⓘ localhost:8083/users

```
{
  "msg": "Missing Authorization Header"
}
```

## 전체 사용자 목록 요청 (인증)

← → ↻ ⓘ localhost:8083/users

```
{
  "data": {
    "users": [
      {
        "active": true,
        "email": "user@test.com",
        "id": 1,
        "username": "user"
      }
    ]
  }
}
```

# V. 프로젝트 결과

## 클라이언트

### 웹 페이지

Dashboard Home Service Status

Login Register

Restricted area, please login!

### 디버그 상태창

Elements Console Sources Network Performance Memory Application Security Audits Vue

Ready. Detected Vue 2.5.17.

Filter mutations

Base State

00:25:50

Filter inspected state

state

isAuthenticated: false  
token: ""

getters

getToken: ""

## V. 프로젝트 결과

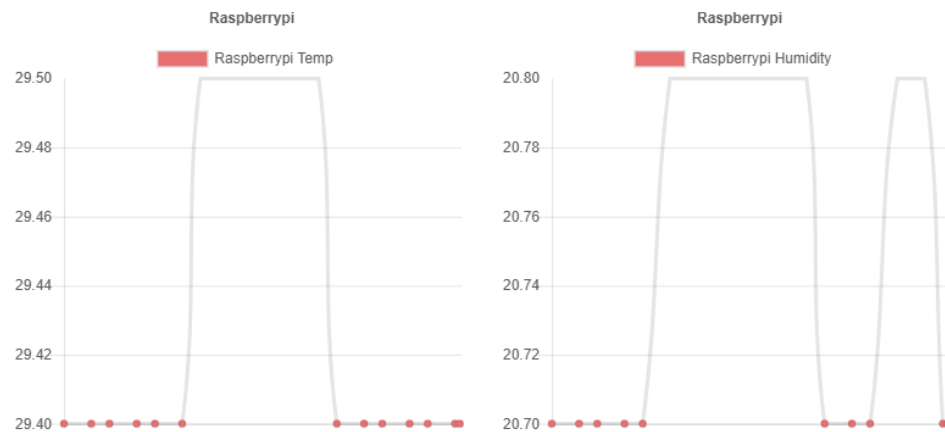
클라이언트

## 웹 페이지

## Dashboard

[Home](#)

## Service Status



디버그 상태창

Ready. Detected Vue 2.5.17.

🔍 Filter mutations

🔍 Filter inspected state

Base State

00:25:50

- ▼ state

```
isAuthenticated: true
```

```
token: "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlNDQyMDIwNjMsIm5iZiI6MTU0NDIwNjA2MywianRpIjoi"
```

- ▼ getters

```
getToken: "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE1NDQyMDIwNjMsIm5iZiI6MTU0NDIwMjA2MywiaW":
```

- ▼ mutation

```
payload: "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsInR5cCI6IkpYXQoImF1dG8iLCJpdWw6Im51ZiI6MTU0NDIiwia2MywianRpIj
```

```
type: "setToken"
```

# V. 프로젝트 결과

클라이언트

## 로그인 페이지

[Dashboard](#) [Home](#) [Service Status](#)

[Login](#) [Register](#)

로그인이 필요합니다.

로그인

[암호를 잊어버리셨습니까?](#)

## 회원가입 페이지

[Dashboard](#) [Home](#) [Service Status](#)

[Login](#) [Register](#)

### 회원 가입

필수 입력 항목

회원가입

## VI. 결론

- IoT 서비스에 Microservice Architecture를 적용해 Cloud에 적합한 어플리케이션 설계
- Token을 기반으로 Stateless Authentication을 적용하여 서비스의 수평 스케일링 용이
- 구조에 큰 변화를 주지 않고도 요청에 맞는 인스턴스 스케일링을 통해 대규모 서비스와 소규모 서비스에 모두 적합한 모델