Microservice Archtecture 기반 JWT 인증 IoT 서버 연구

한국산업기술대학교 스마트팩토리 융합학과
김태형, 서준민, 송예리

KPU 한국산업기술대학교

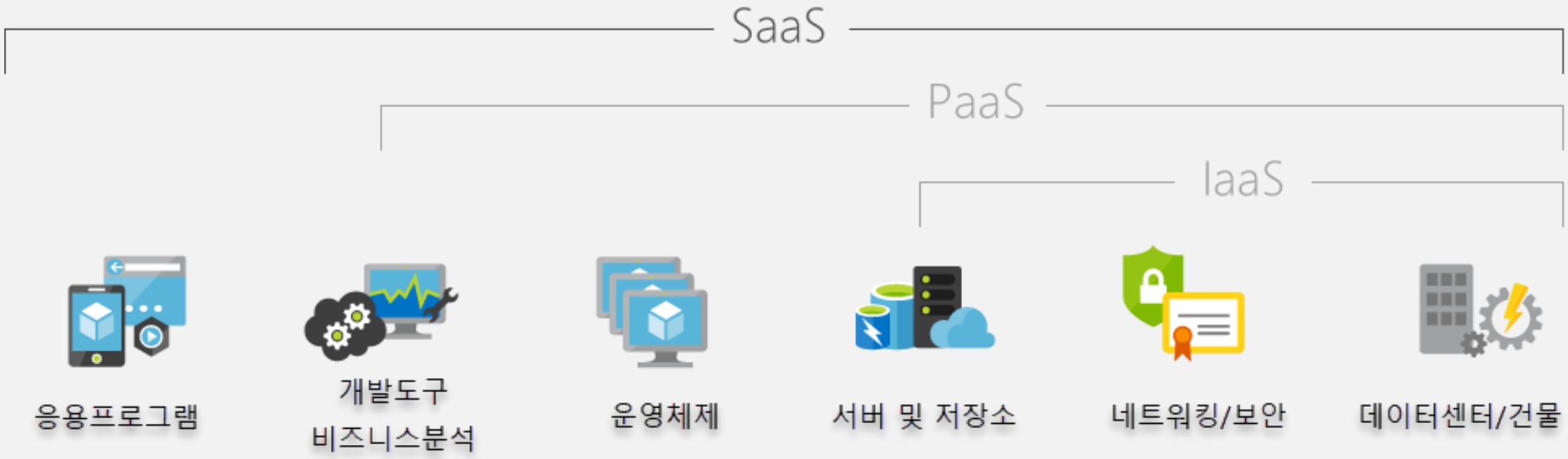
TABLE OF CONTENTS

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

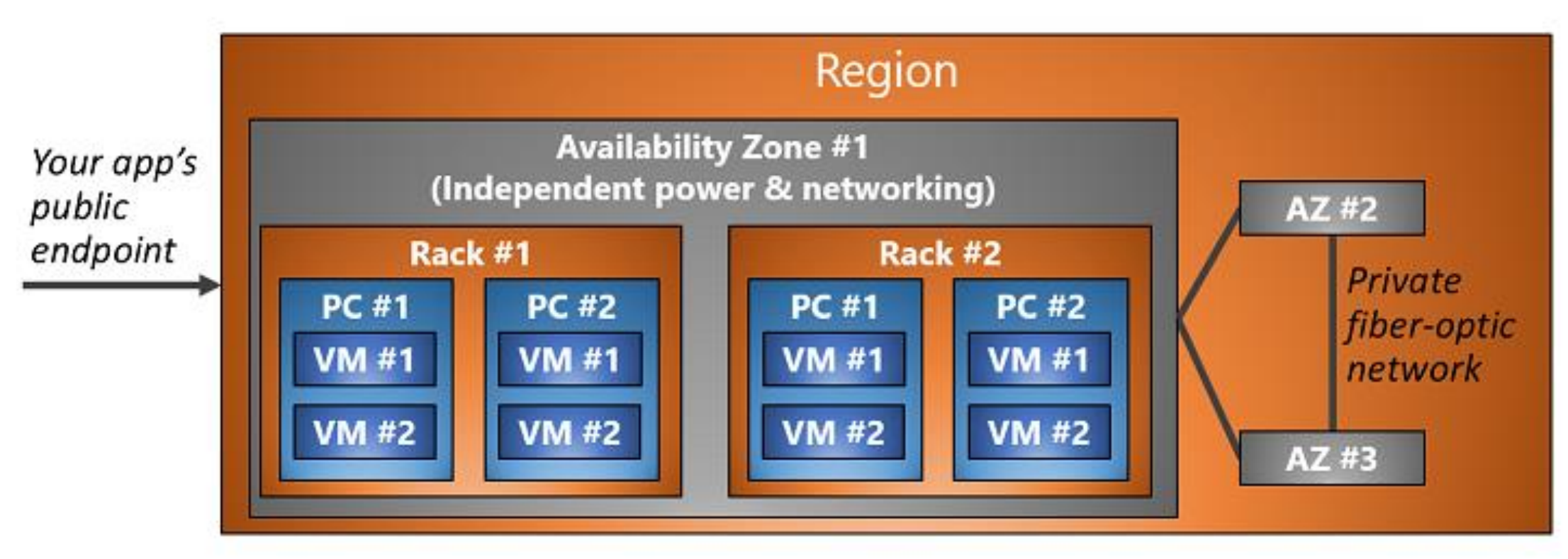
Ⅱ. Cloud

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

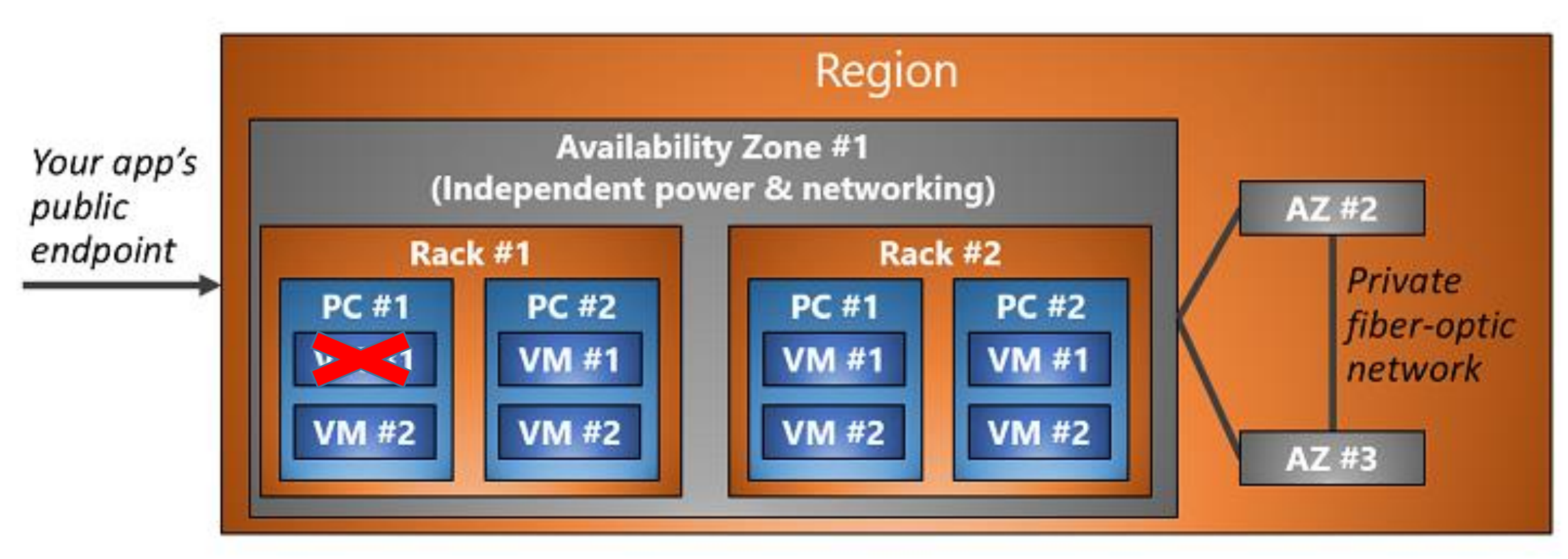
4) Cloud 데이터 센터 구조



II. Cloud

4) Cloud 데이터 센터 구조

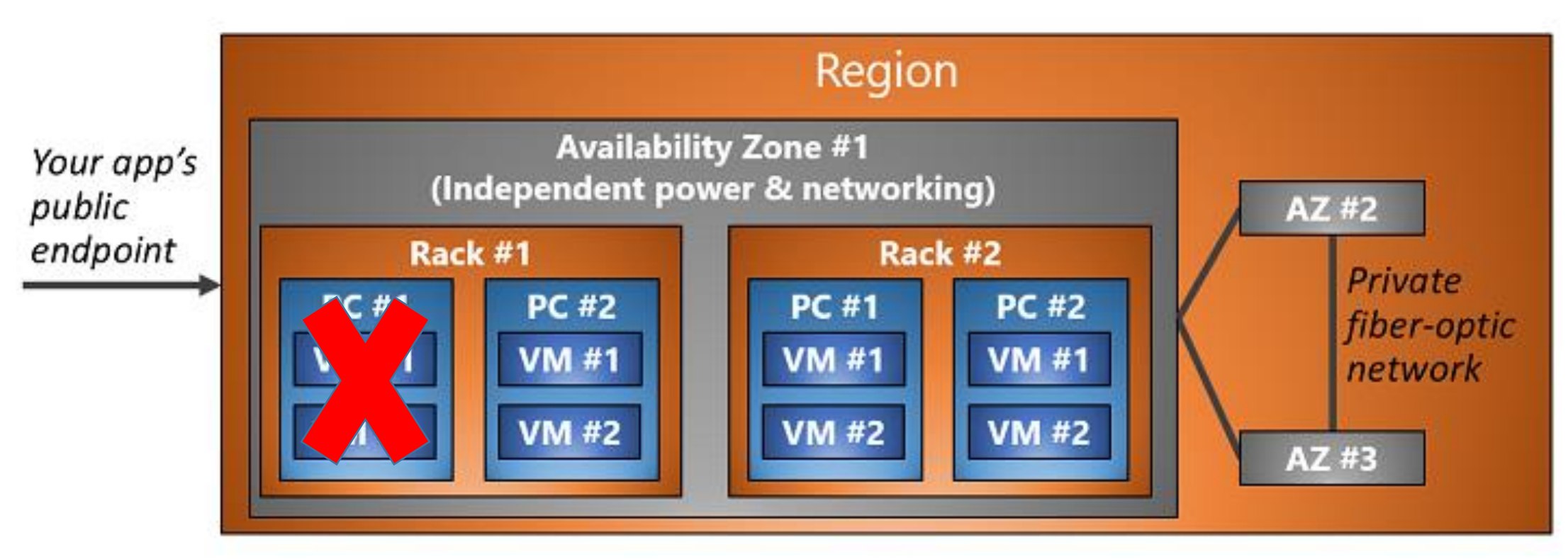
- 하나의 VM 정시 시, 서비스 장애 발생



II. Cloud

4) Cloud 데이터 센터 구조

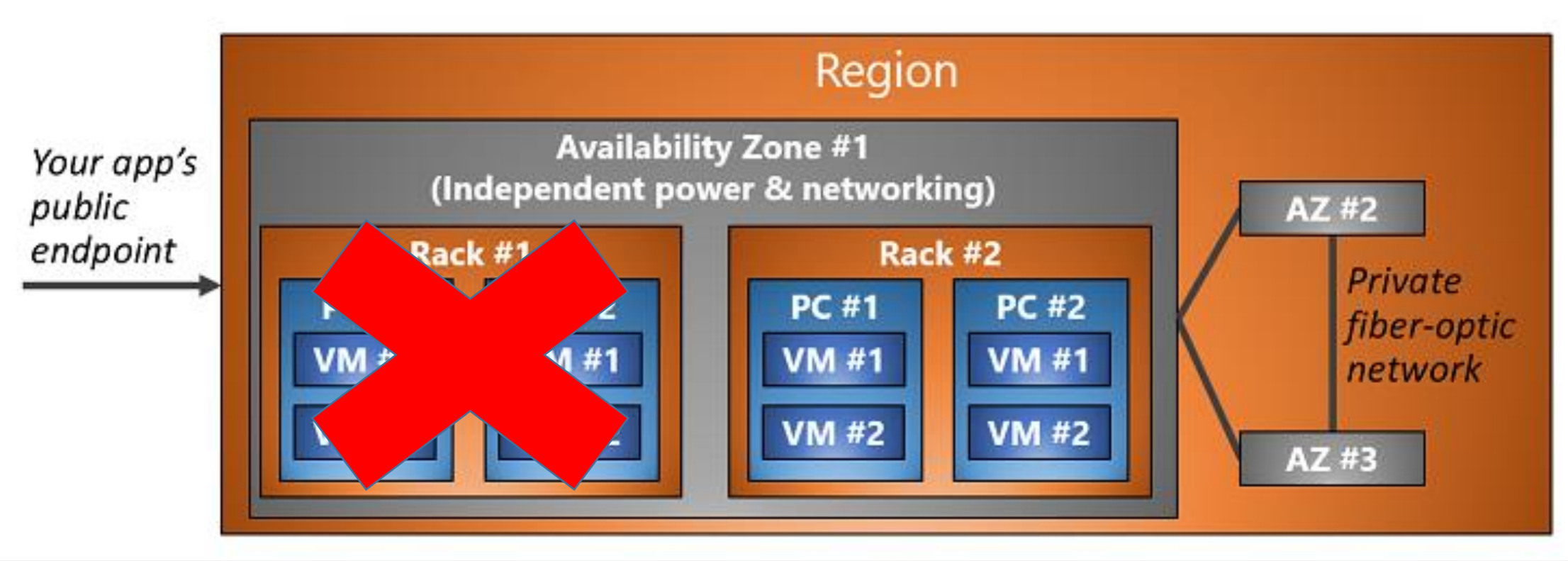
- 하나의 PC 정지 시, 속해 있는 모든 VM 장애 발생



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

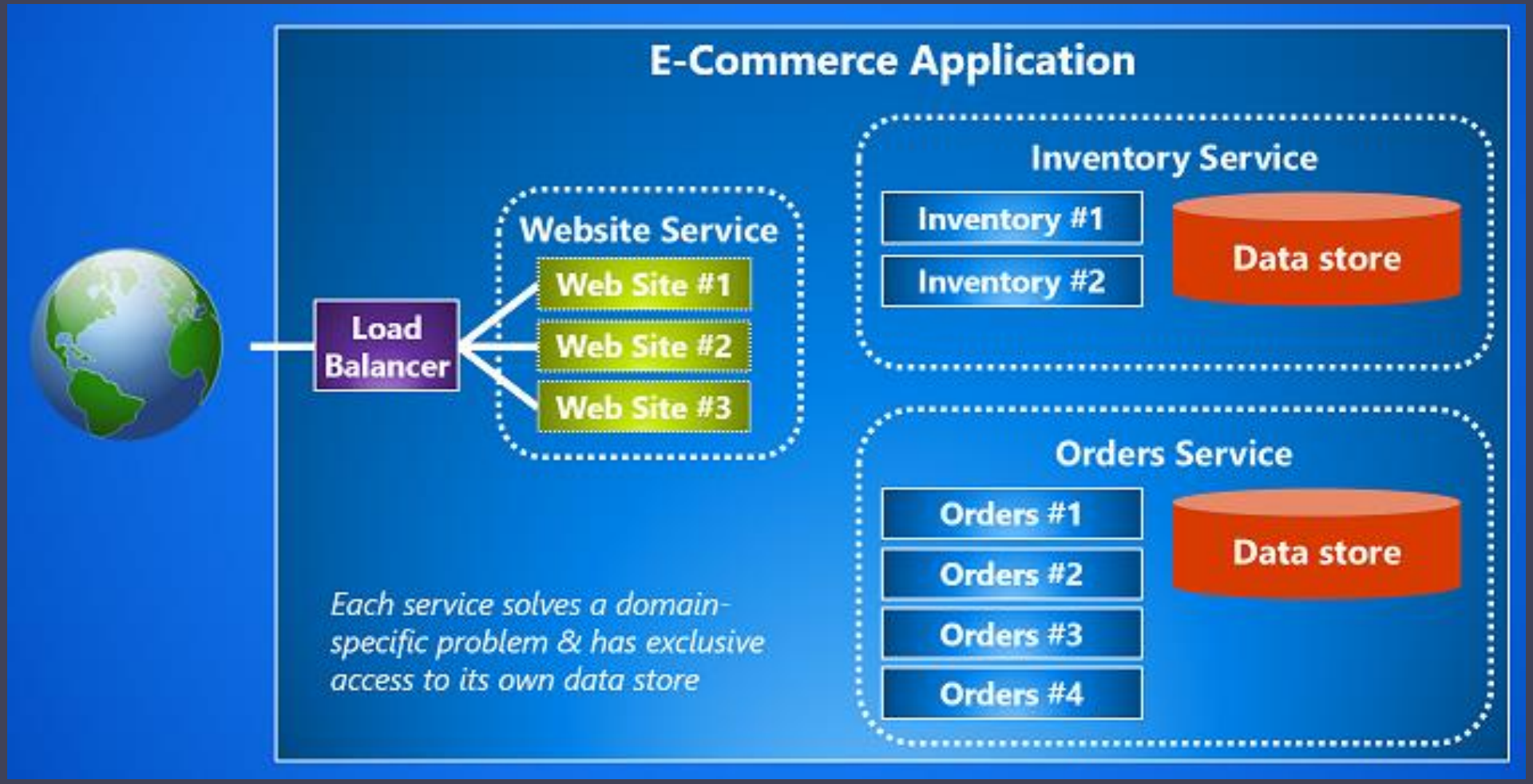
II. Cloud

4) Cloud 데이터 센터 구조

- 하나의 리전 정지 시, 속해 있는 모든 AZ 장애 발생

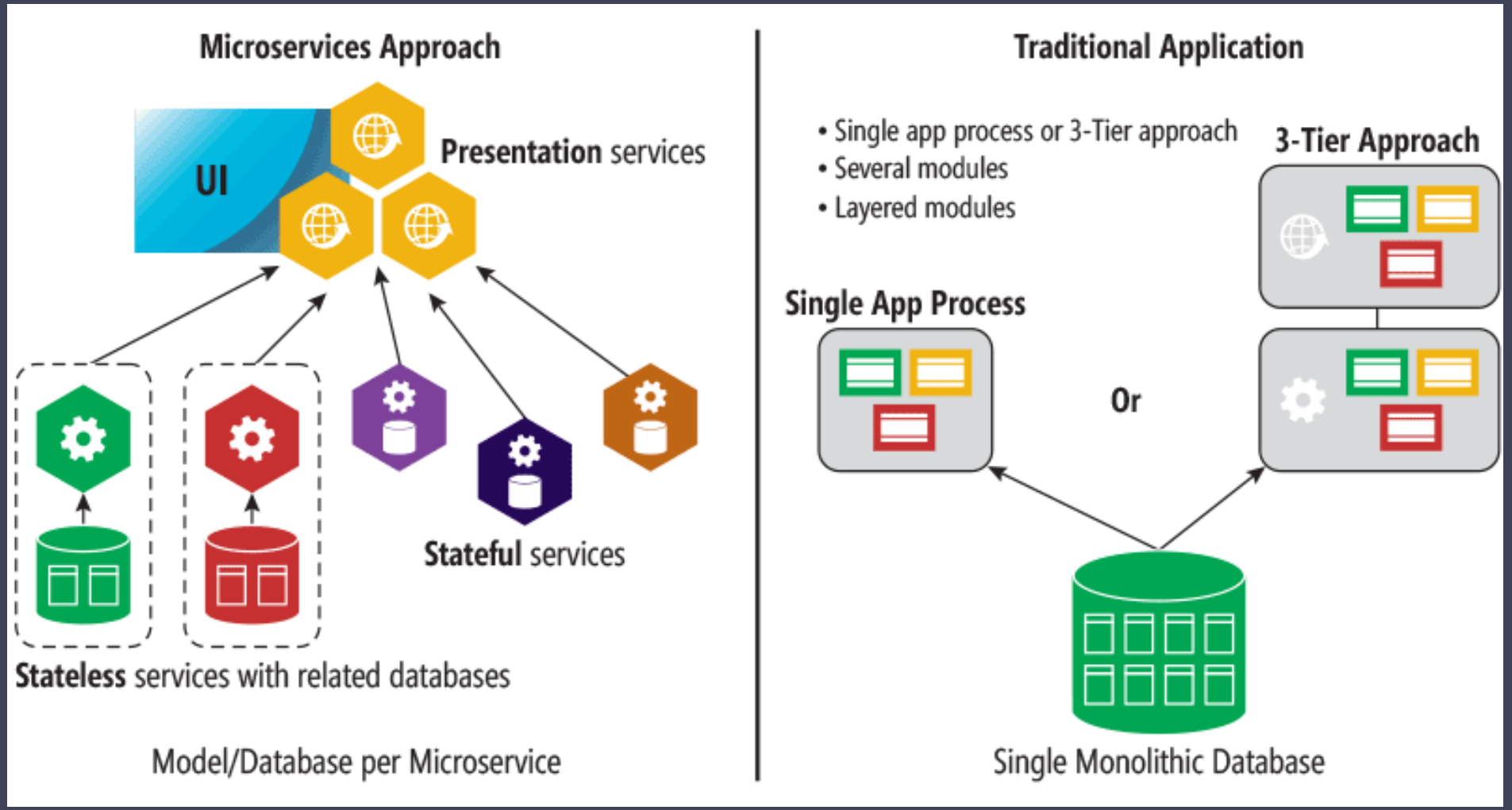
III. Microservice Architecture (MSA)

Microservice Architecture overview



III. Microservice Architecture (MSA)

MSA vs Monolithic

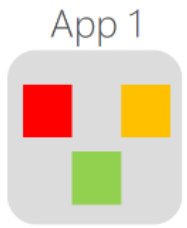


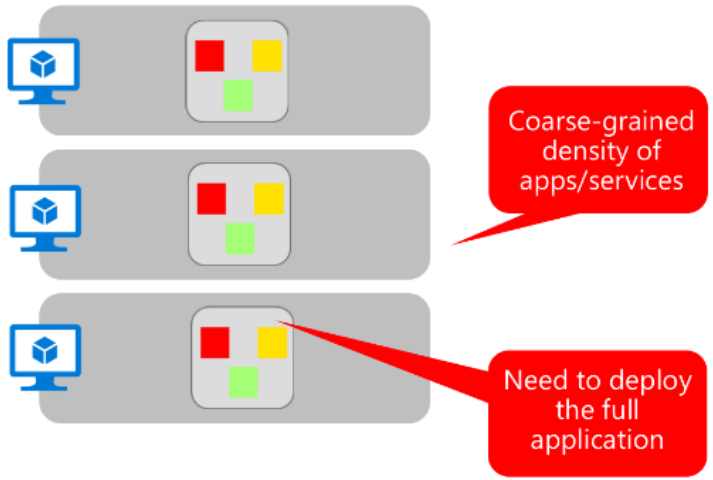
III. Microservice Architecture (MSA)

MSA vs Monolithic

Monolithic deployment approach

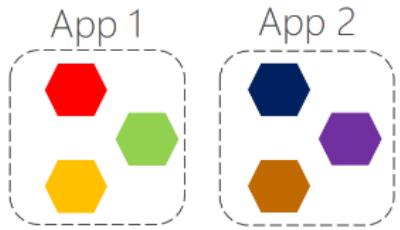
- A traditional application has most of its functionality within a few processes that are componentized with layers and libraries.
- Scales by cloning the app on multiple servers/VMs

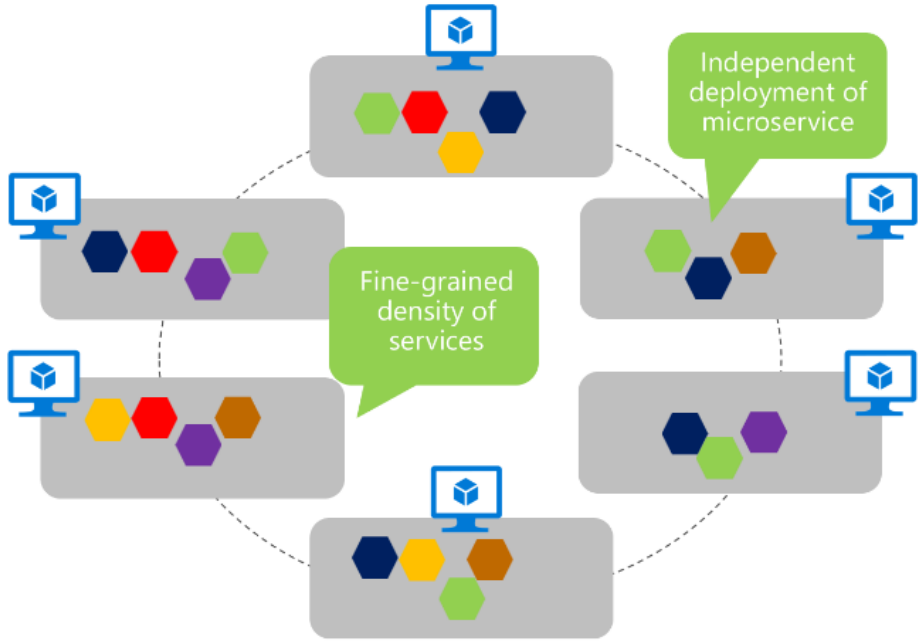




Microservices application approach

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

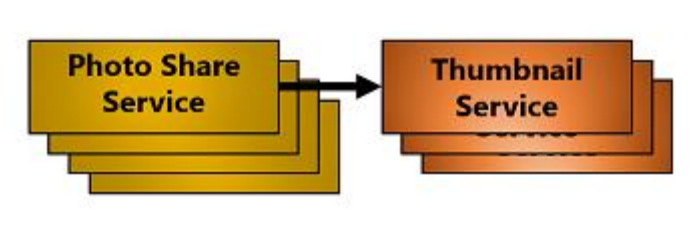




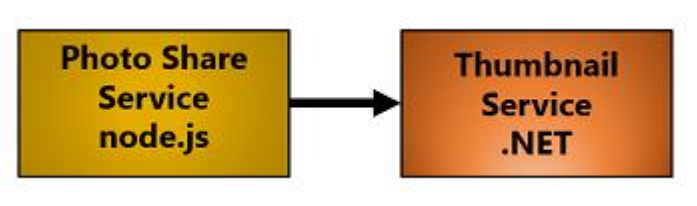
III. Microservice Architecture (MSA)

MSA 장점

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



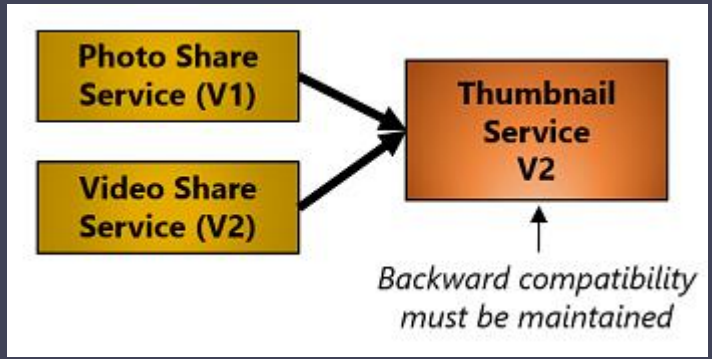
- 기반 프레임워크 다양화 가능
  - 각 인스턴스마다 다른 프레임워크를 사용한 개발 가능



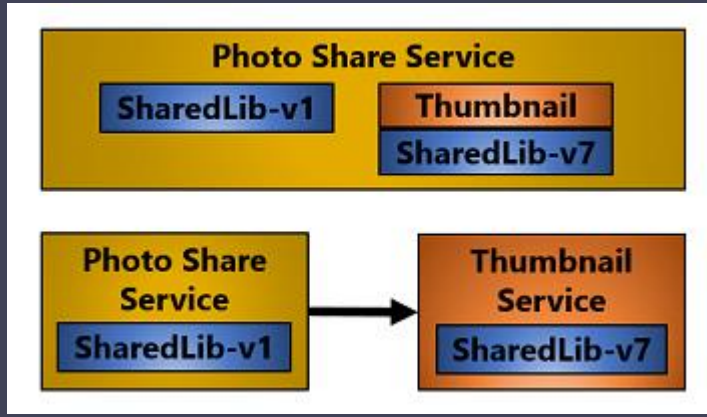
III. Microservice Architecture (MSA)

MSA 장점

- 다양한 API 버전의 서비스 동시 구동 가능

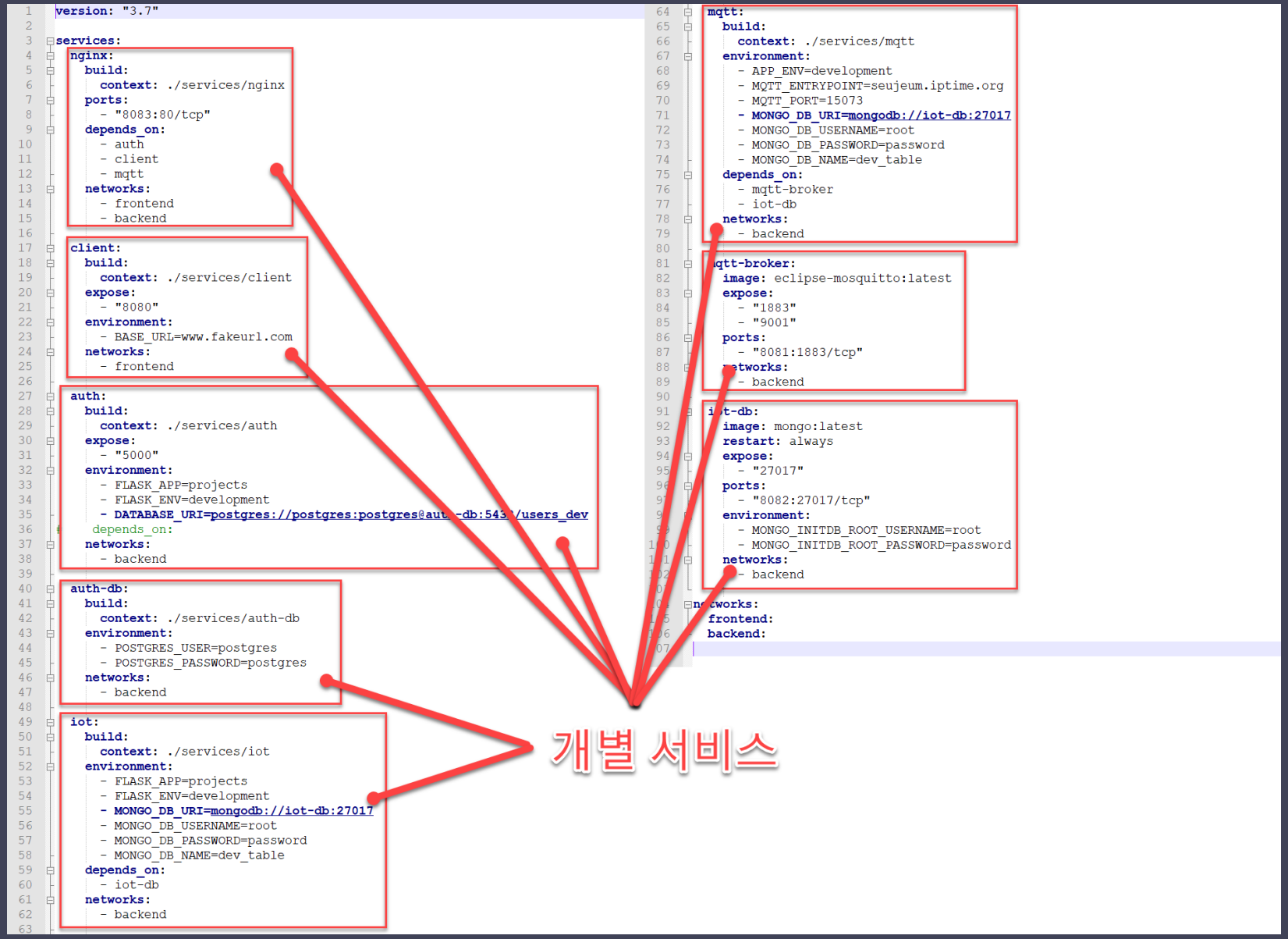


- 상이한 버전의 공통 라이브러리 사용 가능



III. Microservice Architecture (MSA)

MSA 단점 – 복잡도 증가



III. Microservice Architecture (MSA)

MSA 단점 – 복잡도 증가

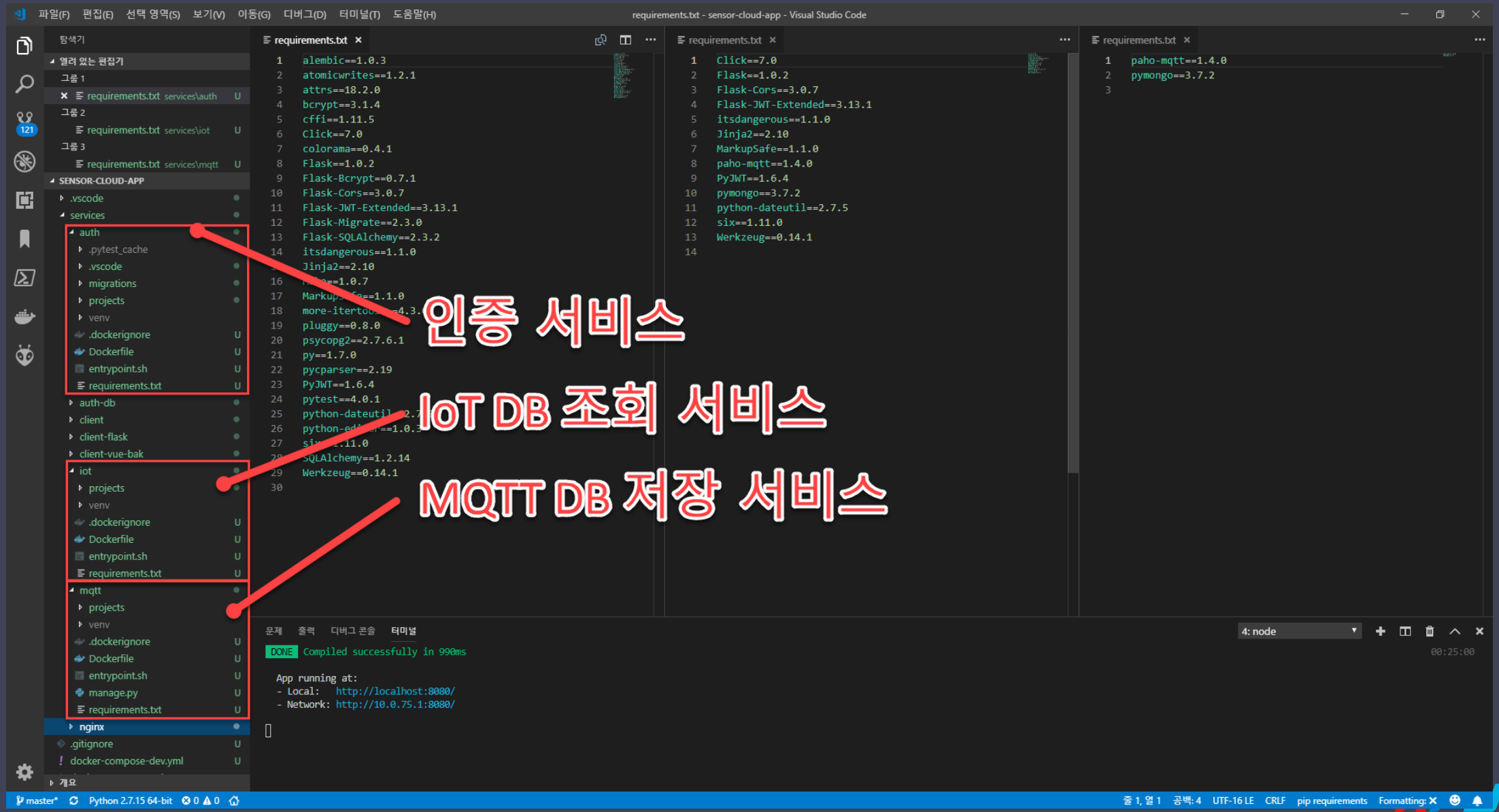
```yaml
version: "3.7"

services:
  nginx:
    build:
      context: ./services/nginx
    ports:
      - "8083:80/tcp"
    depends_on:
      - auth
      - client
      - mqtt
    networks:
      - frontend
      - backend

  client:
    build:
      context: ./services/client
    expose:
      - "8080"
    environment:
      - BASE_URL=www.fakeurl.com
    networks:
      - frontend

  auth:
    build:
      context: ./services/auth
    expose:
      - "5000"
    environment:
      - FLASK_APP=projects
      - FLASK_ENV=development
      - DATABASE_URI=postgres://postgres:postgres@auth-db:5432/users_dev
#    depends_on:
    networks:
      - backend

  auth-db:
    build:
      context: ./services/auth-db
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    networks:
      - backend

  iot:
    build:
      context: ./services/iot
    environment:
      - FLASK_APP=projects
      - FLASK_ENV=development
      - MONGO_DB_URI=mongodb://iot-db:27017
      - MONGO_DB_USERNAME=root
      - MONGO_DB_PASSWORD=password
      - MONGO_DB_NAME=dev_table
    depends_on:
      - iot-db
    networks:
      - backend

  mqtt:
    build:
      context: ./services/mqtt
    environment:
      - APP_ENV=development
      - MQTT_ENTRYPOINT=seujeum.iptime.org
      - MQTT_PORT=15073
      - MONGO_DB_URI=mongodb://iot-db:27017
      - MONGO_DB_USERNAME=root
      - MONGO_DB_PASSWORD=password
      - MONGO_DB_NAME=dev_table
    depends_on:
      - mqtt-broker
      - iot-db
    networks:
      - backend

  mqtt-broker:
    image: eclipse-mosquitto:latest
    expose:
      - "1883"
      - "9001"
    ports:
      - "8081:1883/tcp"
    networks:
      - backend

  iot-db:
    image: mongo:latest
    restart: always
    expose:
      - "27017"
    ports:
      - "8082:27017/tcp"
    environment:
      - MONGO_INITDB_ROOT_USERNAME=root
      - MONGO_INITDB_ROOT_PASSWORD=password
    networks:
      - backend

networks:
  frontend:
  backend:
```

각 서비스 별로 배포할 때 필요한

Env나 의존성이 상이

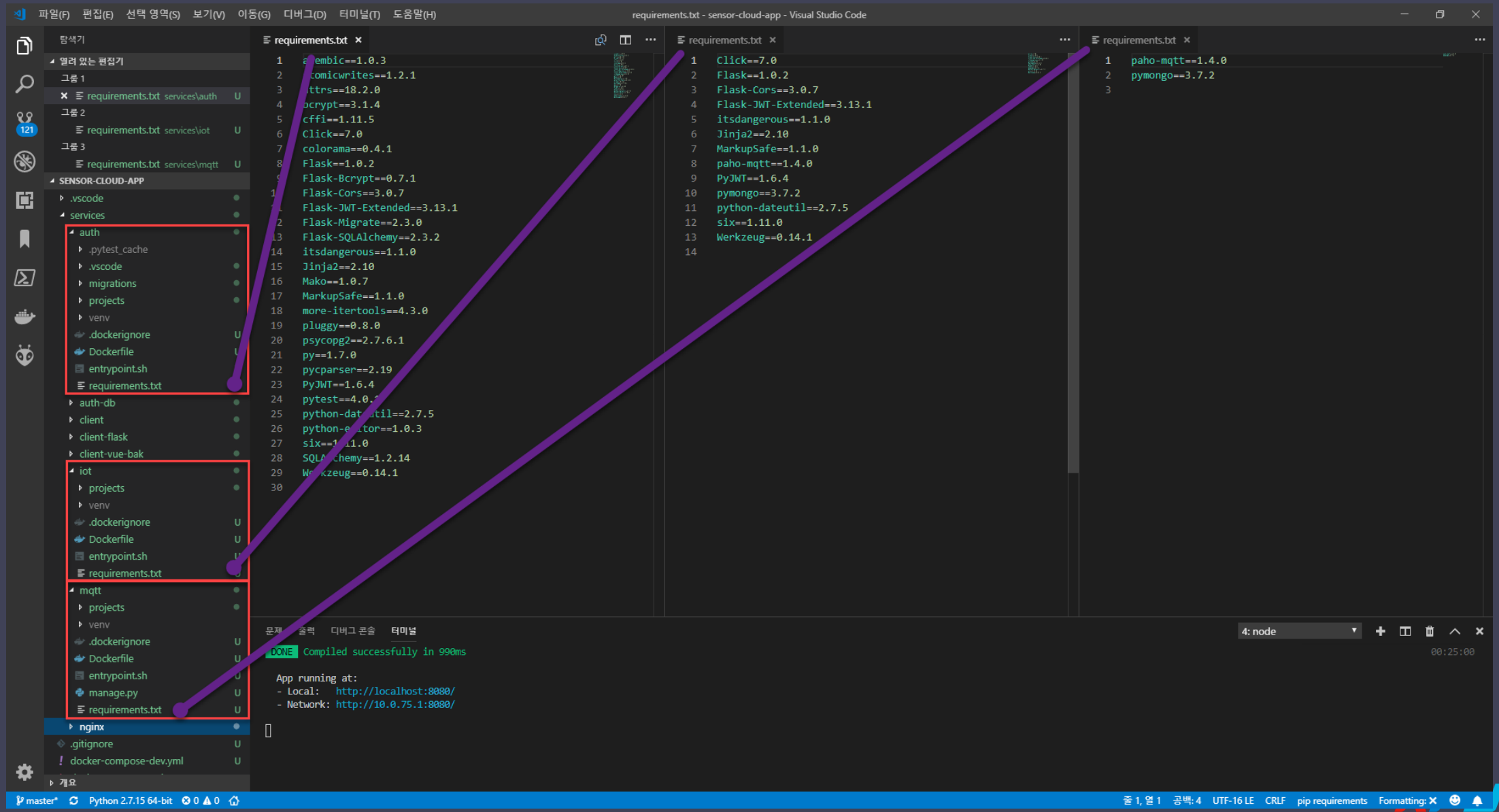
III. Microservice Architecture (MSA)

MSA 단점 – 의존성 파편화



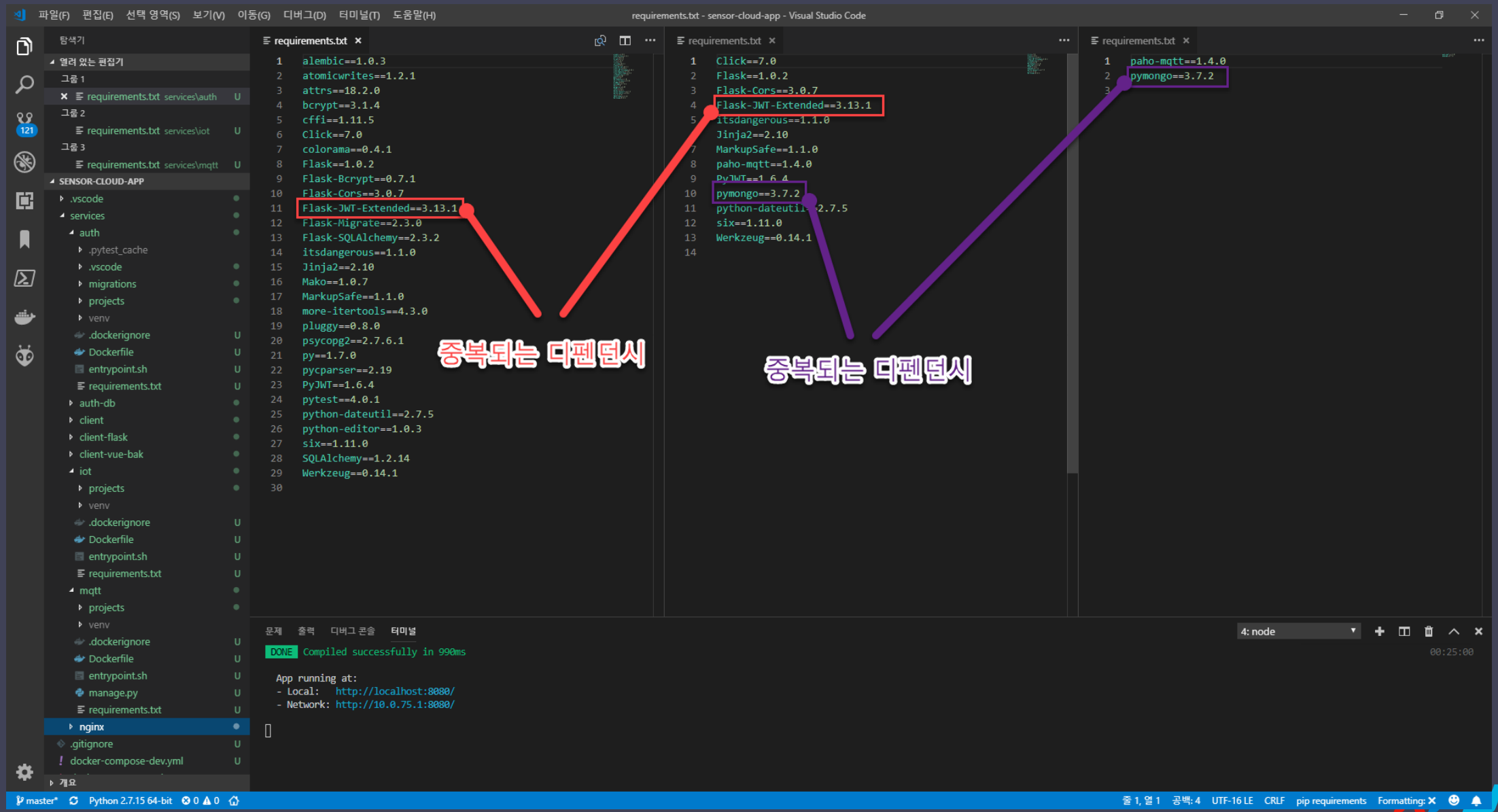
III. Microservice Architecture (MSA)

MSA 단점 – 의존성 파편화



III. Microservice Architecture (MSA)

MSA 단점 – 의존성 파편화



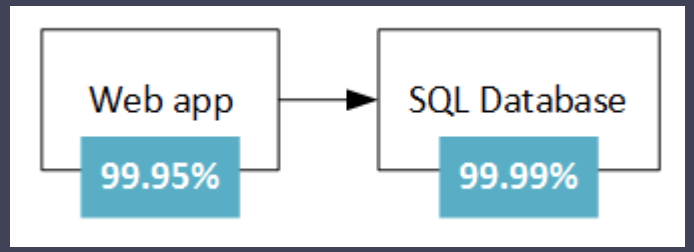
III. Microservice Architecture (MSA)

MSA 단점 - 신뢰성

- 시스템 신뢰성
  - $R_s = R_1 \times R_2 \times \cdots \times R_n = \prod_{i=1}^{n} R_i$
- 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 x 260s)s/mo
99.999%	99.999%, 26s/mo	99.998%, 52s/mo	99.997%, 78s/mo	99.999^n, (n x 26s)s/mo

- Web 어플리케이션과 SQL DB가 있는 시스템의 신뢰성

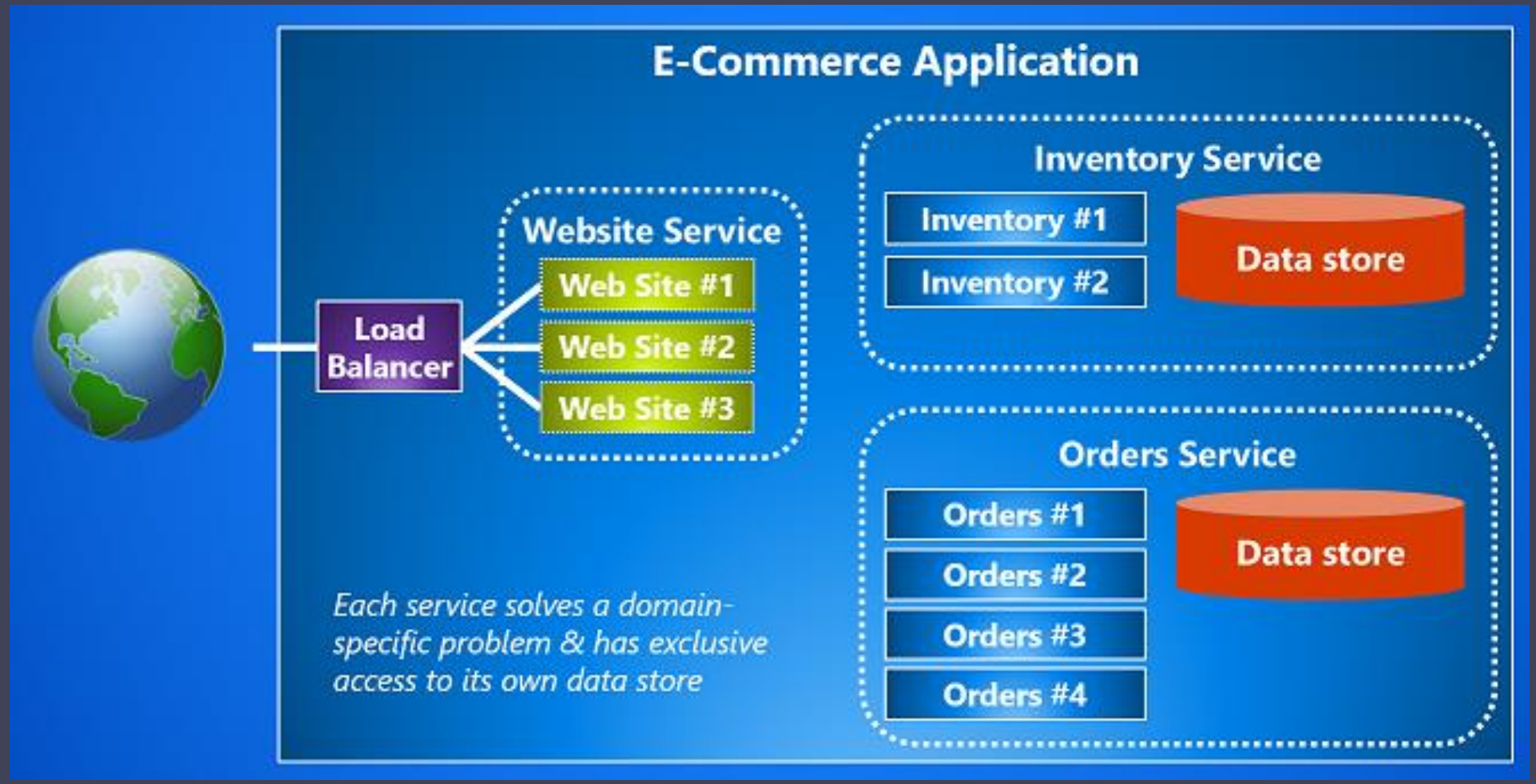


  - SLA = 99.95% x 99.99% = 99.94%

III. Microservice Architecture (MSA)

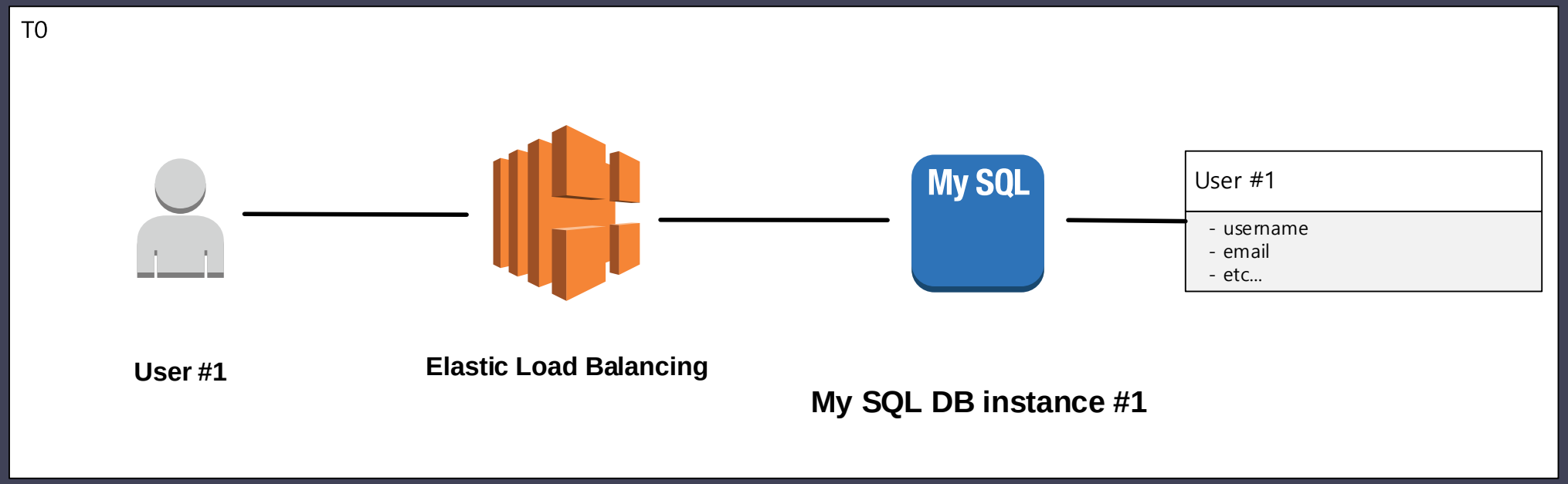
MSA 단점 - 신뢰성

- 신뢰성을 향상시키기 위해서 같은 인스턴스를 복제할 수 있다.



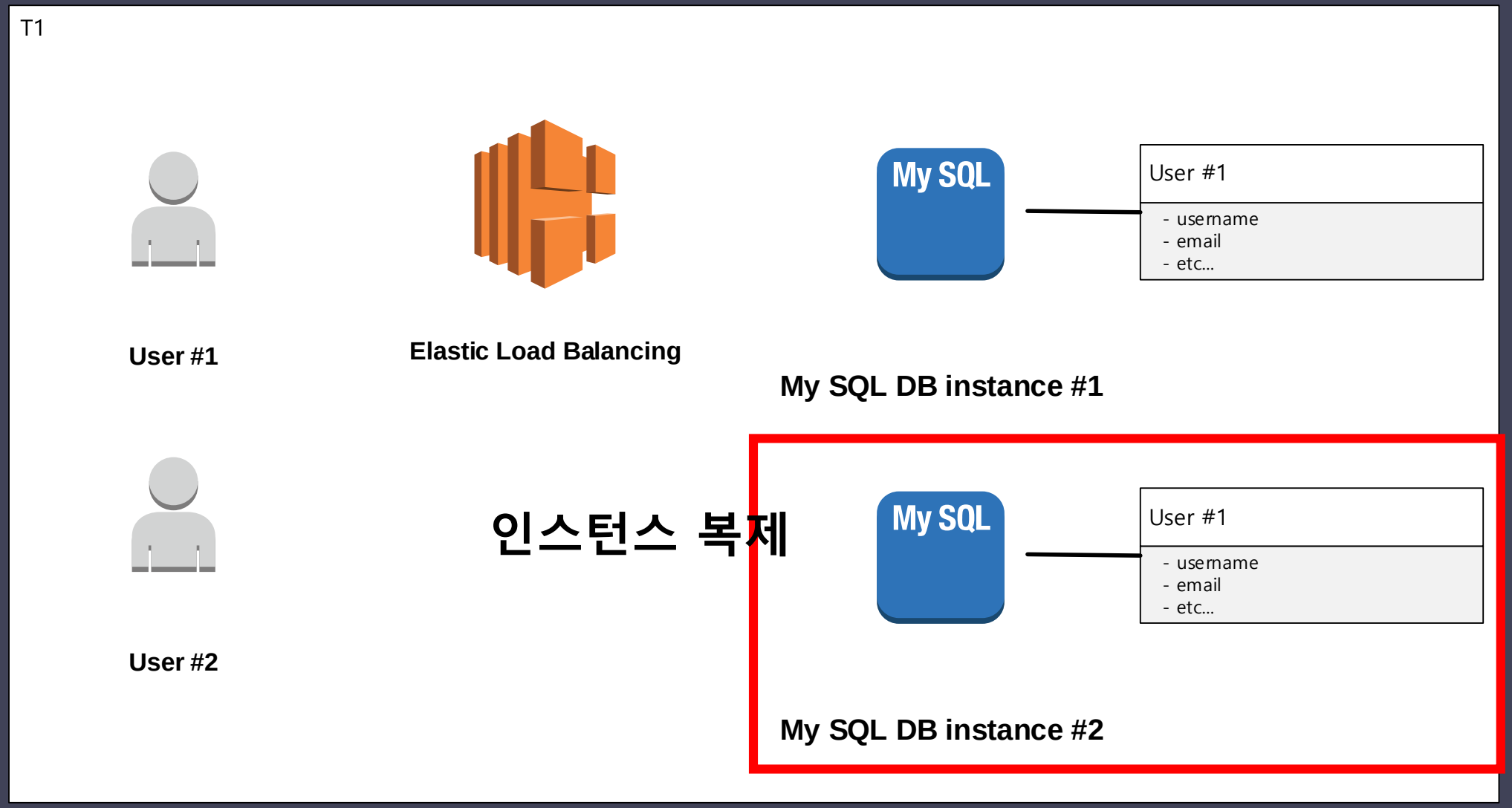
III. Microservice Architecture (MSA)

Data consistency issues



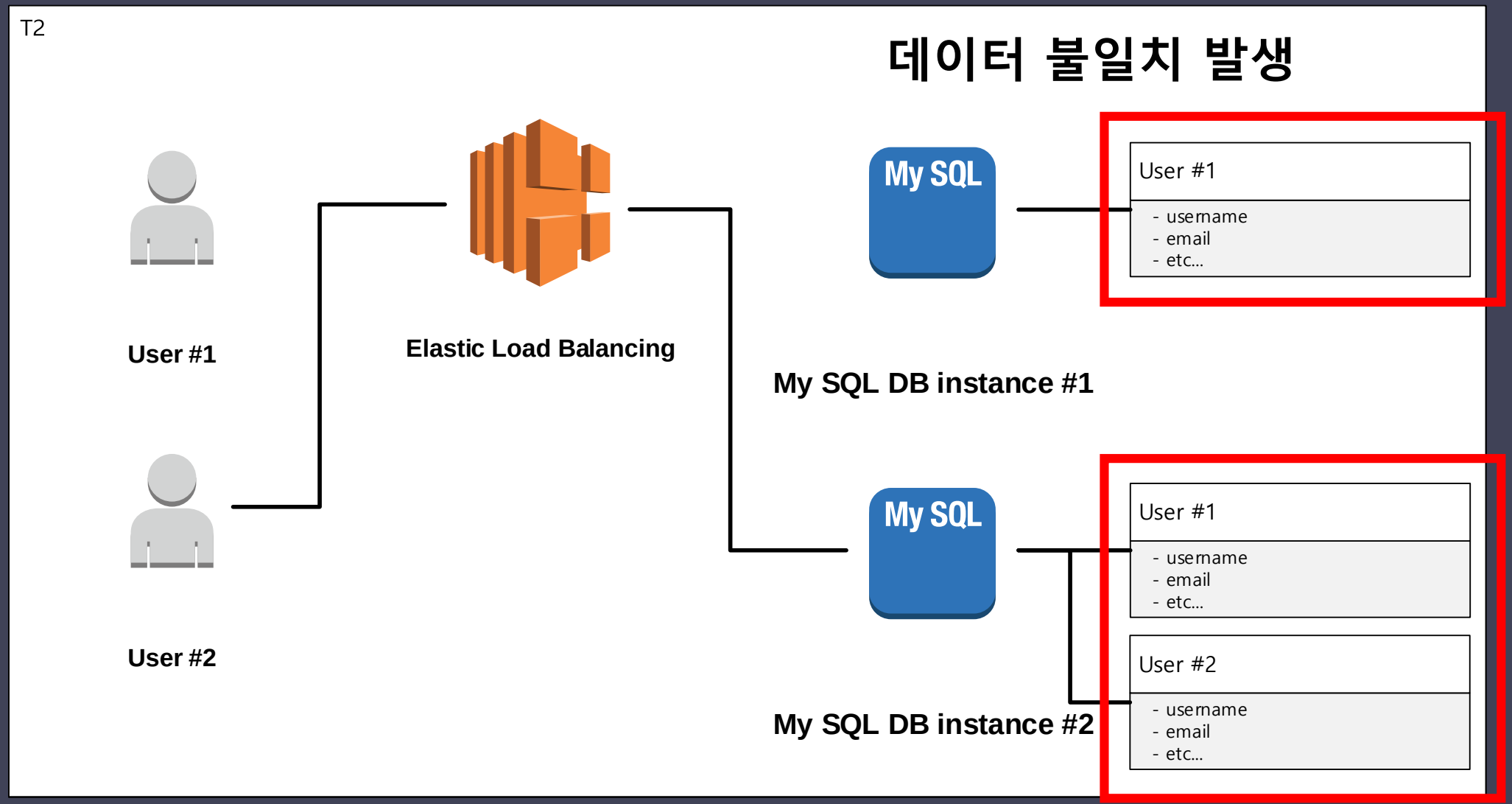
III. Microservice Architecture (MSA)

Data consistency issues



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

```
eyJhbGciOiJIUzI1NiIsInR5c
CI6IkpXVCJ9.eyJzdWIiOiIxM
jM0NTY3ODkwIiwibmFtZSI6Ik
pvaG4gRG9lIiwiYWRtaW4iOnR
ydWV9.TJVA95OrM7E2cBab30R
MHrHDcEfxjoYZgeFONFh7HgQ
```

Decoded

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

Header

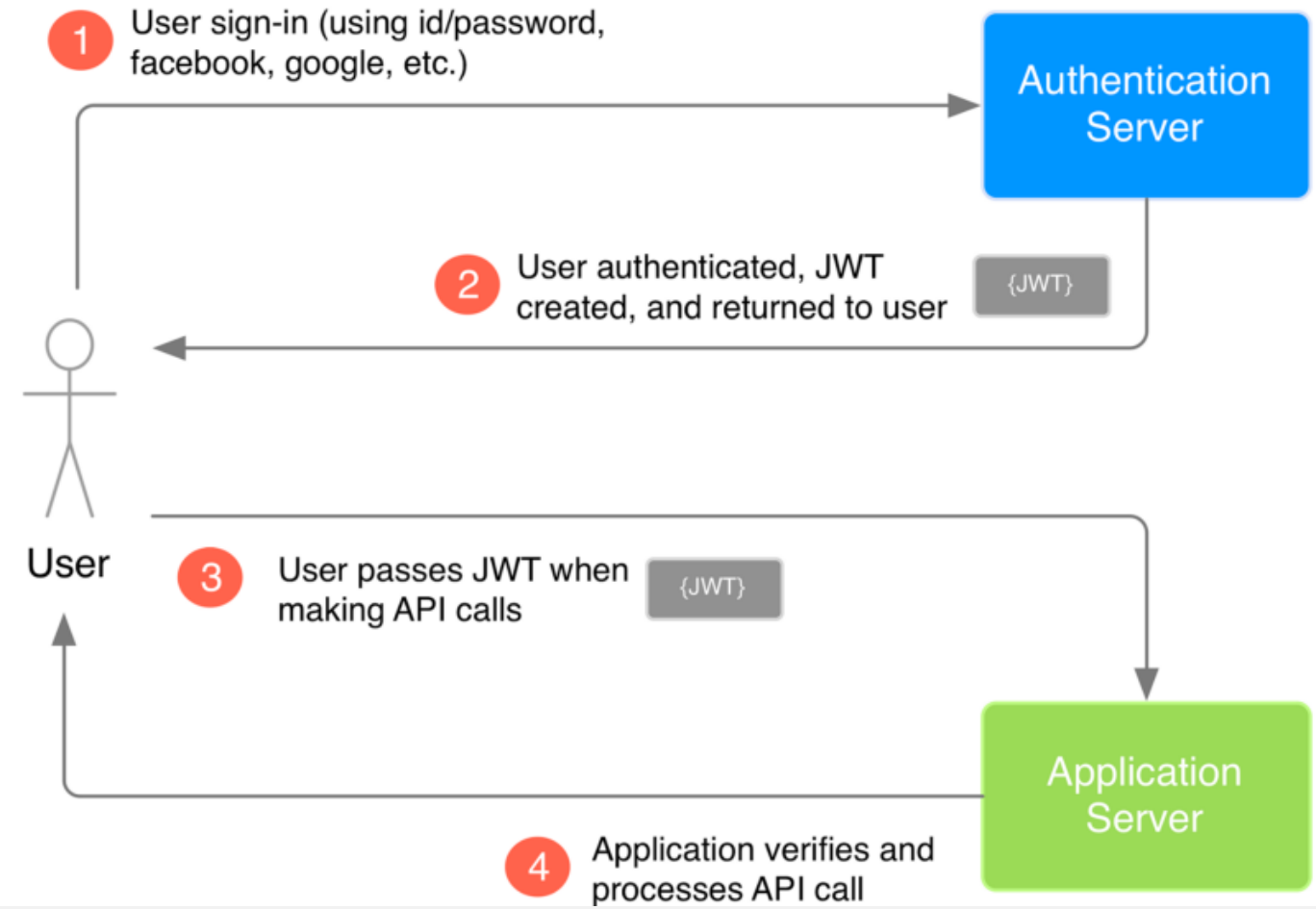
Payload

Signature

IV. JSON Web Token (JWT) & Authentication

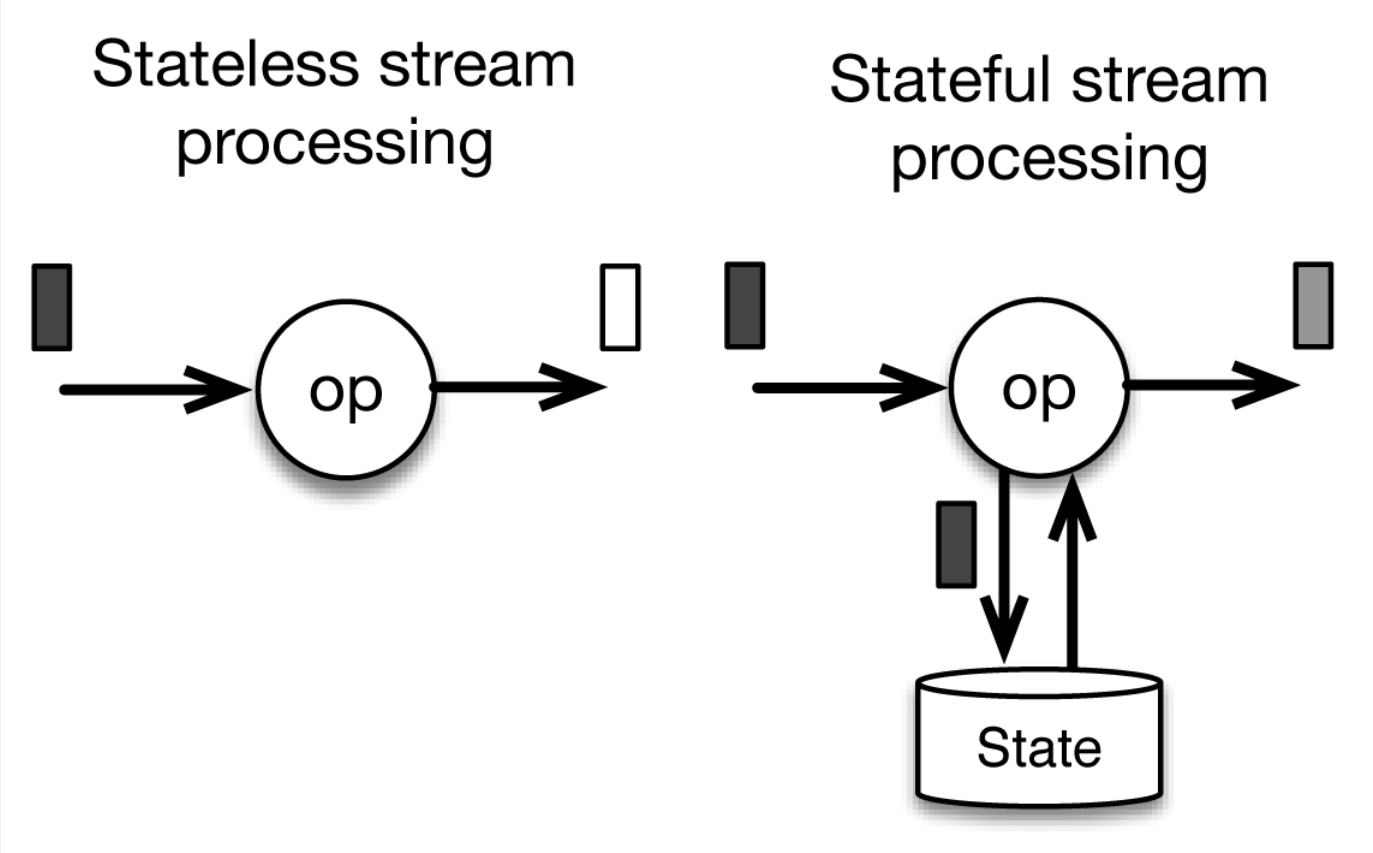
2) JWT란?

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



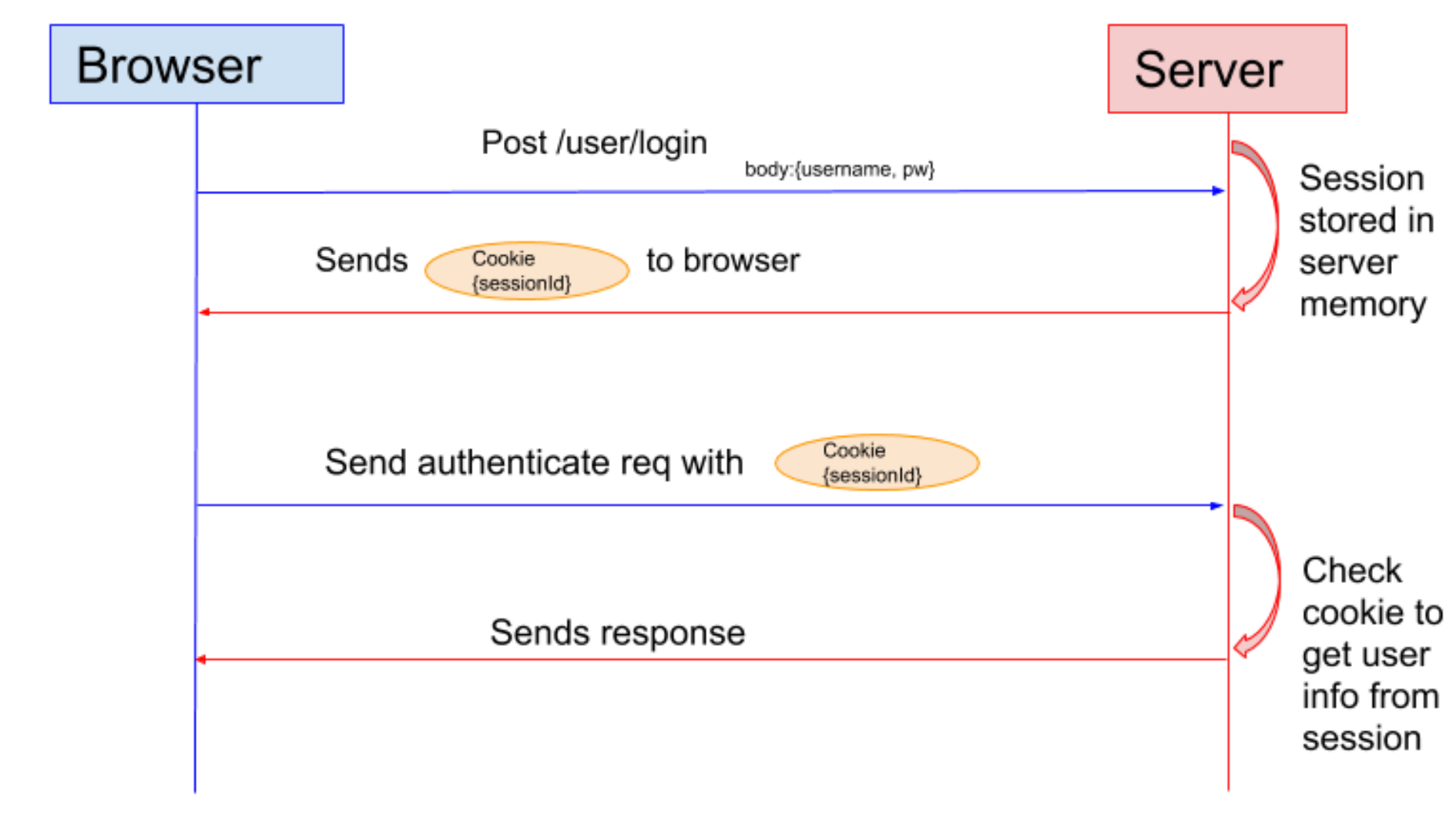
IV. JSON Web Token (JWT) & Authentication

Stateful? Stateless?



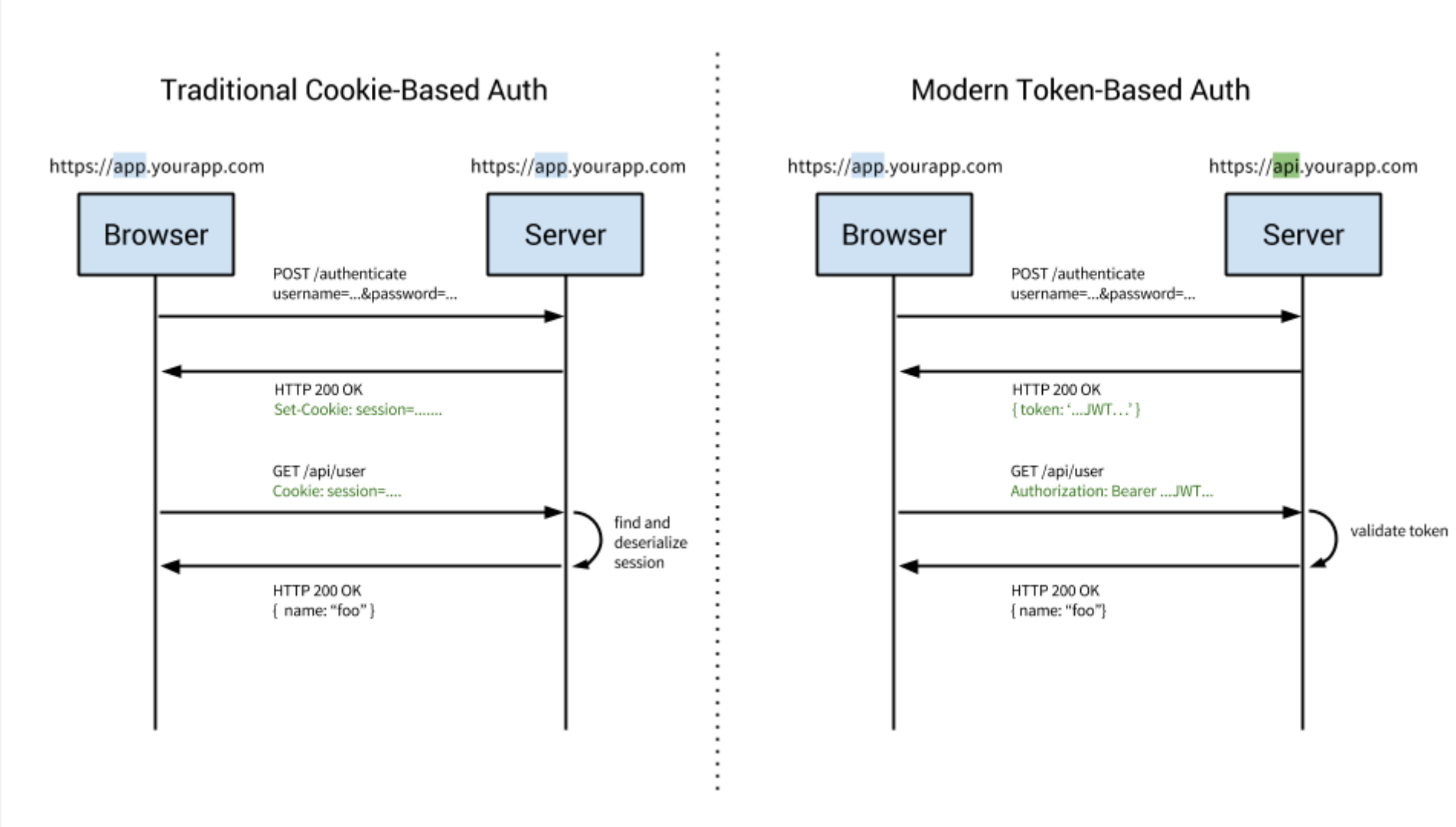
IV. JSON Web Token (JWT) & Authentication

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)

```python
@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
    except Exception as e:
        logger.warn(f'Unexpected error occured!')
        logger.warn(f'err_log: {str(e)}')
        response_object['message'] = f'Unexpected error occured!'
        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

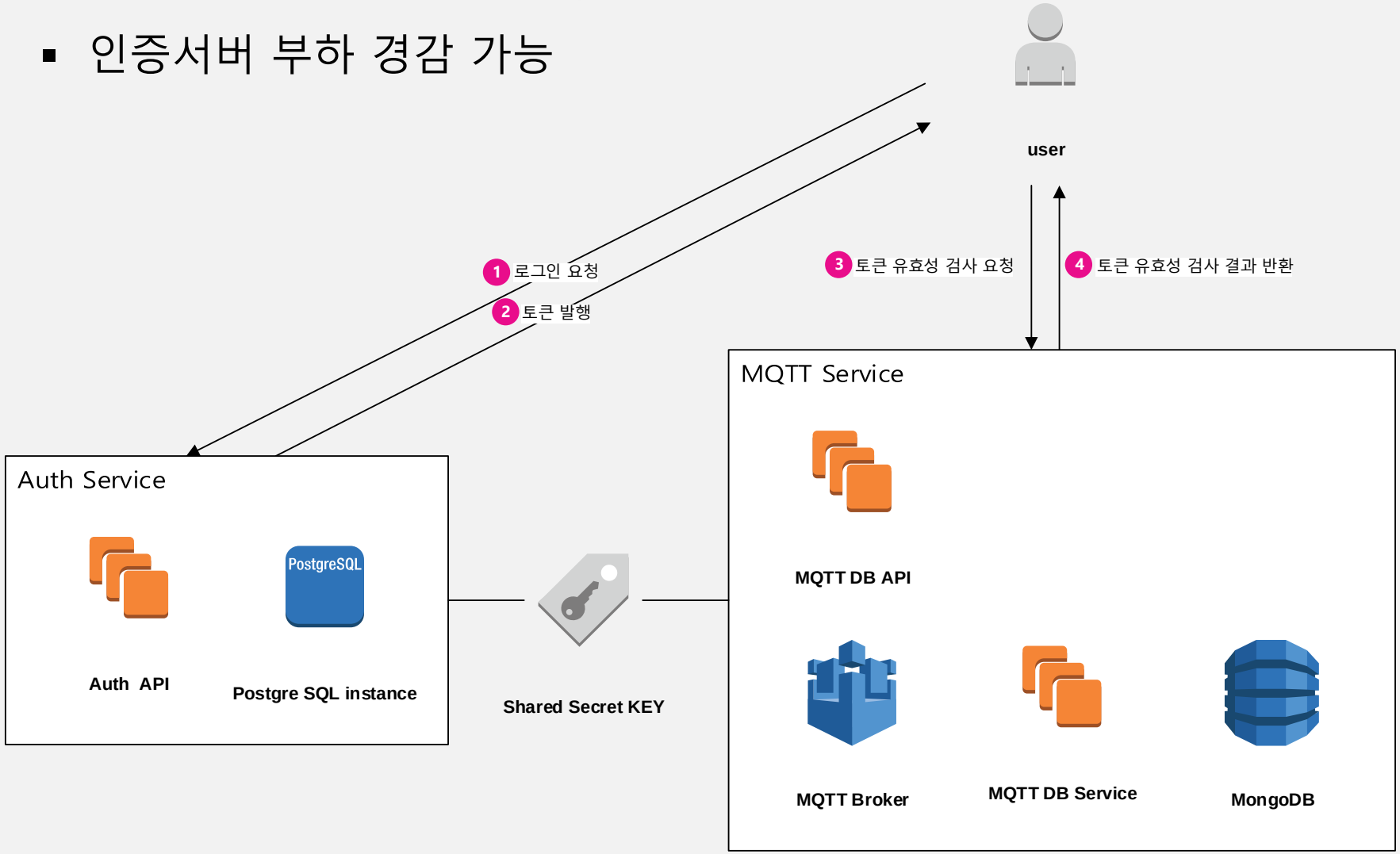
```python
@iot_api.route('/iot/<string:topic>', methods=['GET'])
@iot_api.route('/iot/<string:topic>/latest', methods=['GET'])
@jwt_required
def get_data_single(topic):
    collection = mongo.db[topic]
    response_object = {}
    logger.info(f'get latest data {topic} in {collection}')
    try:
    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

IV. JSON Web Token (JWT) & Authentication

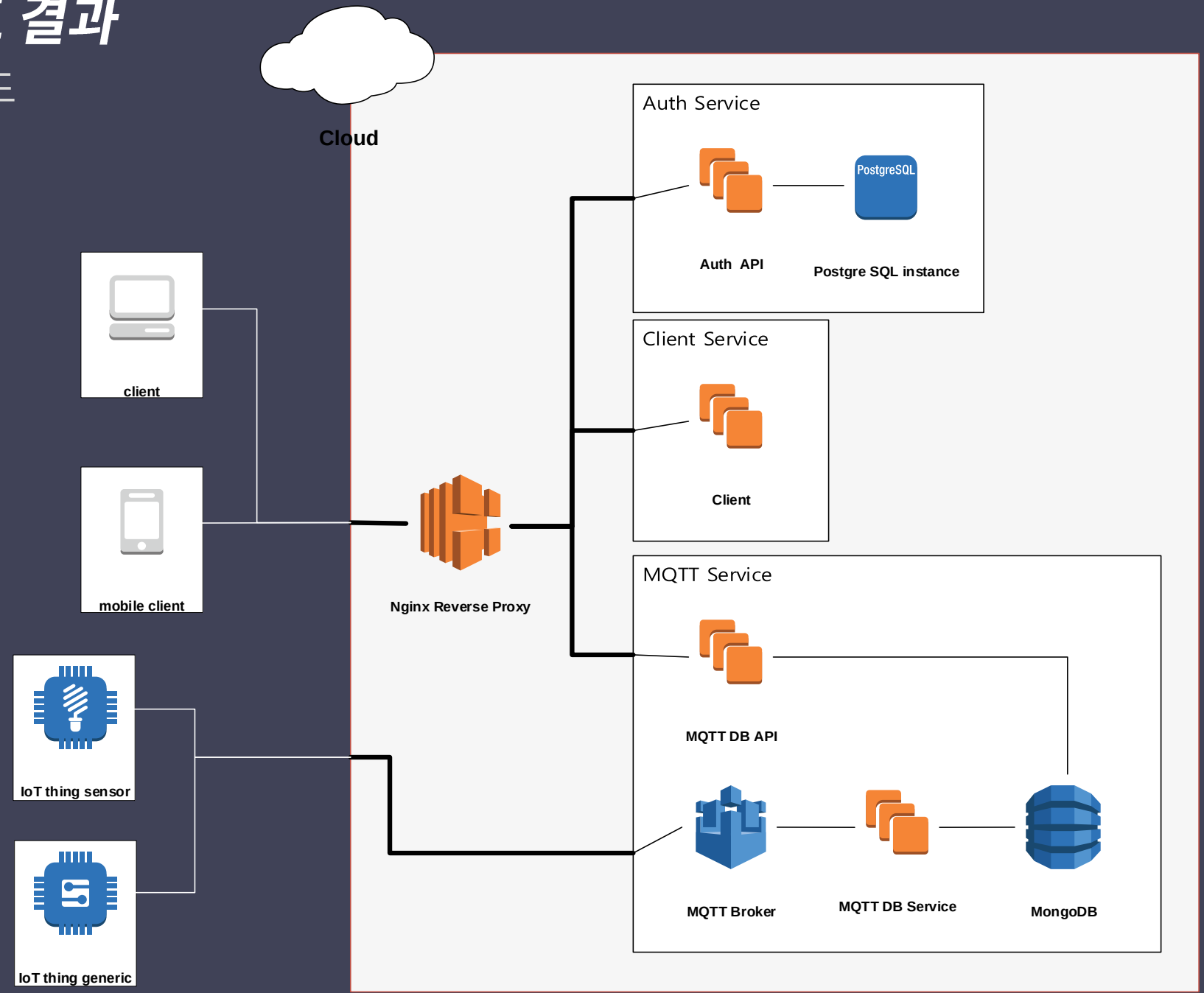
Authentication using JWT

- 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

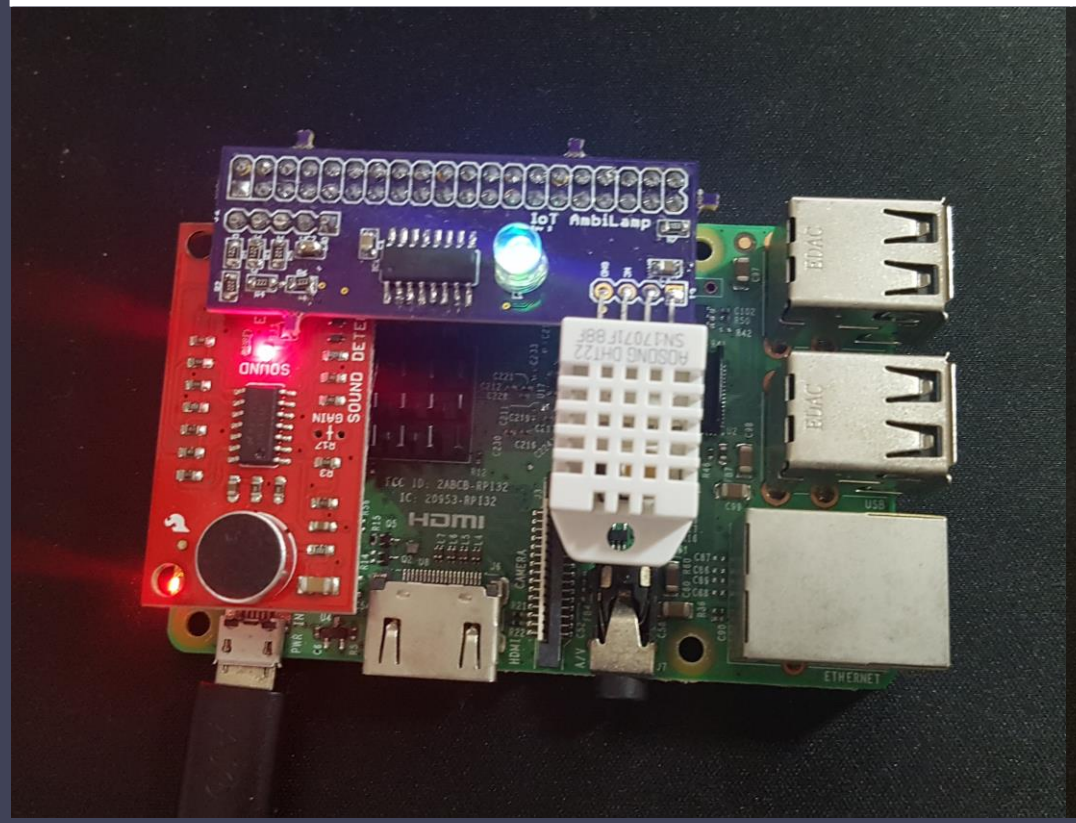
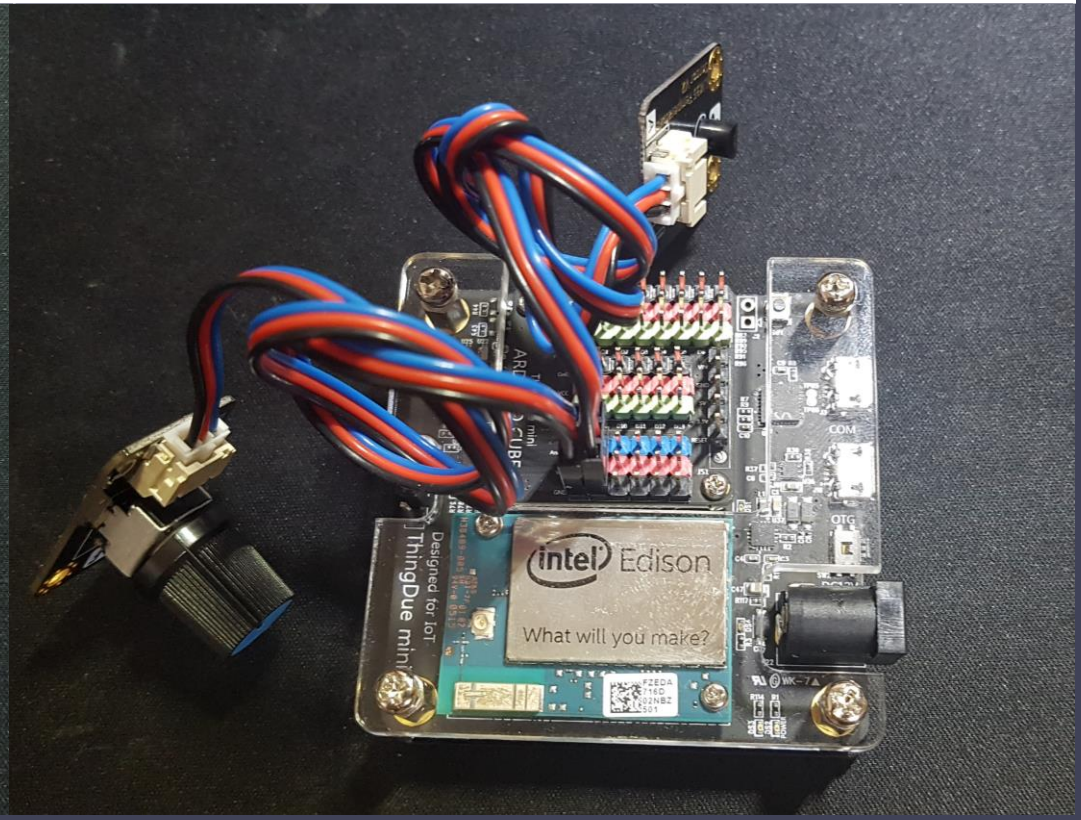
POST	/auth/login 로그인 라우트
POST	/auth/register 사용자 계정 생성 라우트
GET	/users 전체 사용자 목록 조회 라우트
GET	/users/{user_id} 특정 사용자 정보 조회 라우트

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)
	

V. 프로젝트 결과

센서 데이터 요청

최신 센서 데이터 요청 결과

localhost:8083/iot/raspberrypi.temp


```
{
  "data": [
    {
      "date": "2018-12-07T07:54:56+00:00",
      "value": "28.4"
    }
  ],
  "topic": "raspberrypi.temp"
}
```


기간별 센서 데이터 요청 결과

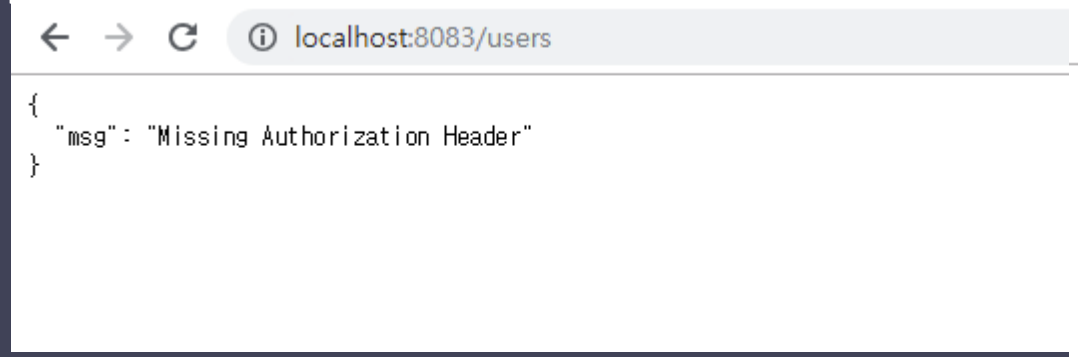
localhost:8083/iot/raspberrypi.temp/2018-12-05/2018-12-08


```
{
  "data": [
    {
      "date": "2018-12-07T07:55:23+00:00",
      "value": "28.3"
    },
    {
      "date": "2018-12-07T07:55:21+00:00",
      "value": "28.3"
    },
    {
      "date": "2018-12-07T07:55:18+00:00",
      "value": "28.3"
    },
    {
      "date": "2018-12-07T07:55:16+00:00",
      "value": "28.4"
    },
    {
      "date": "2018-12-07T07:55:13+00:00",
      "value": "28.4"
    },
    {
      "date": "2018-12-07T07:55:11+00:00",
      "value": "28.4"
    },
    {
      "date": "2018-12-07T07:55:08+00:00",
      "value": "28.4"
    },
    {
      "date": "2018-12-07T07:55:06+00:00",
      "value": "28.4"
    },
```

V. 프로젝트 결과

전체 사용자 목록 조회 결과

전체 사용자 목록 요청 (미인증)

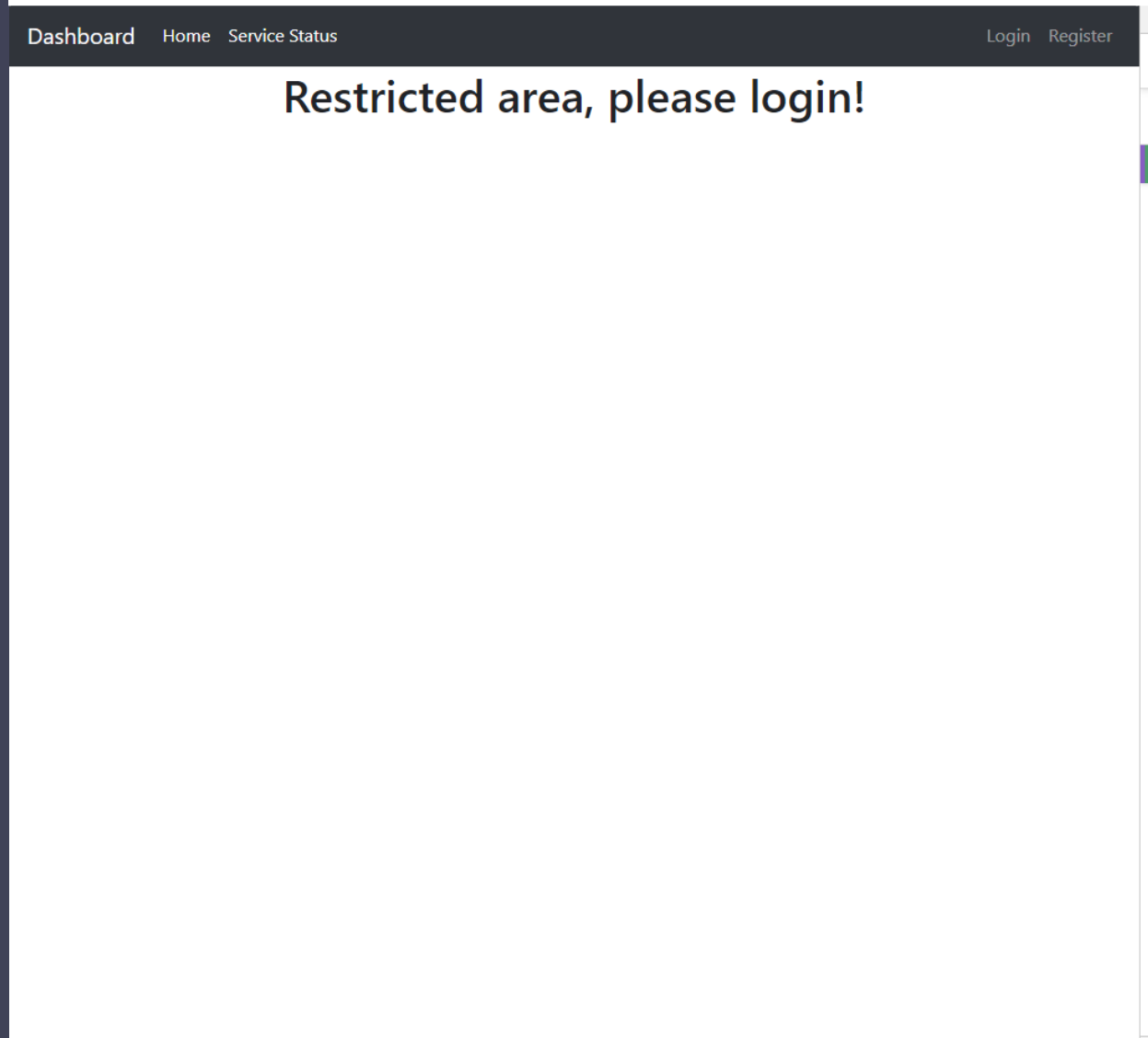


전체 사용자 목록 요청 (인증)

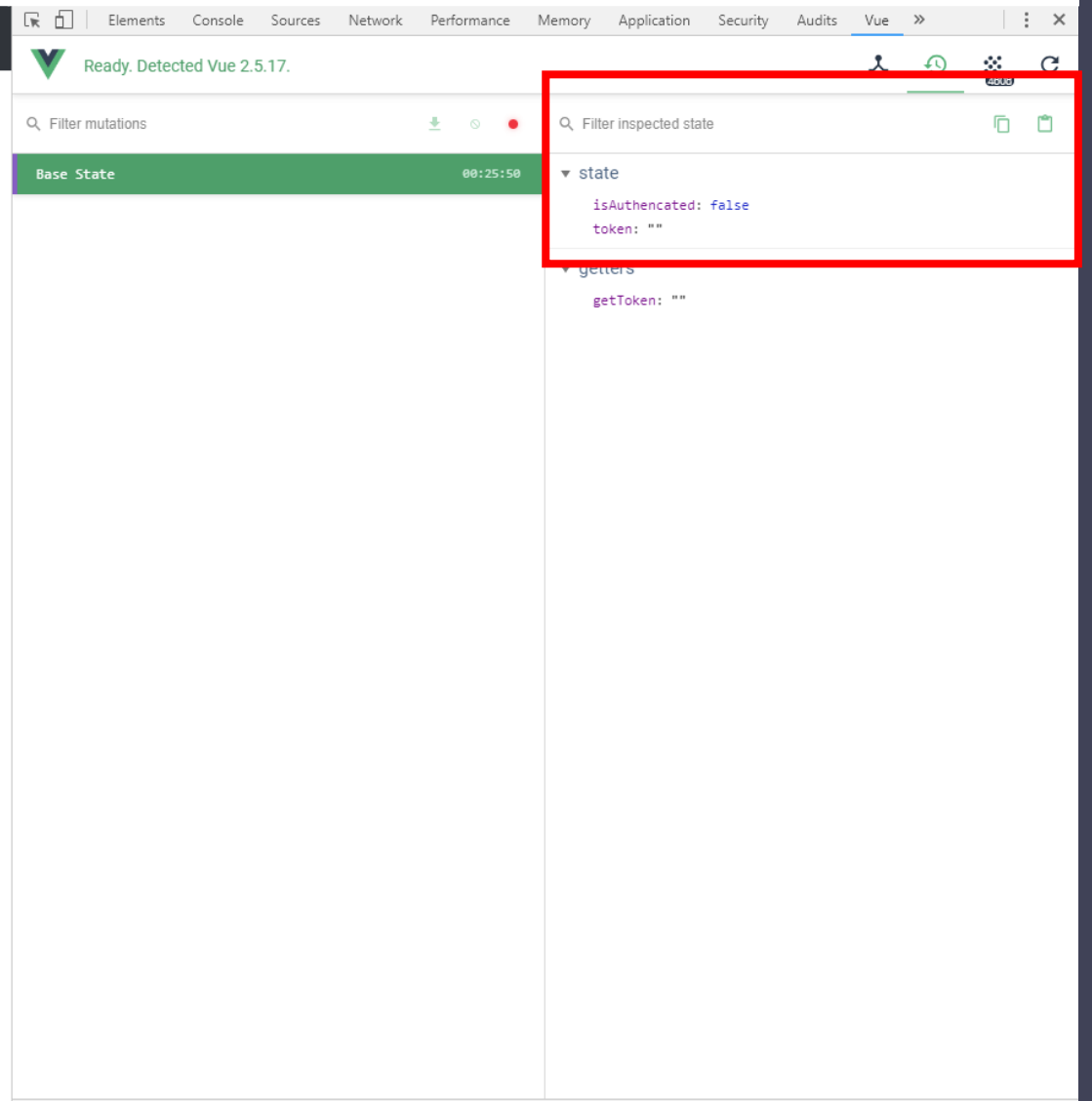
V. 프로젝트 결과

클라이언트

웹 페이지



디버그 상태창

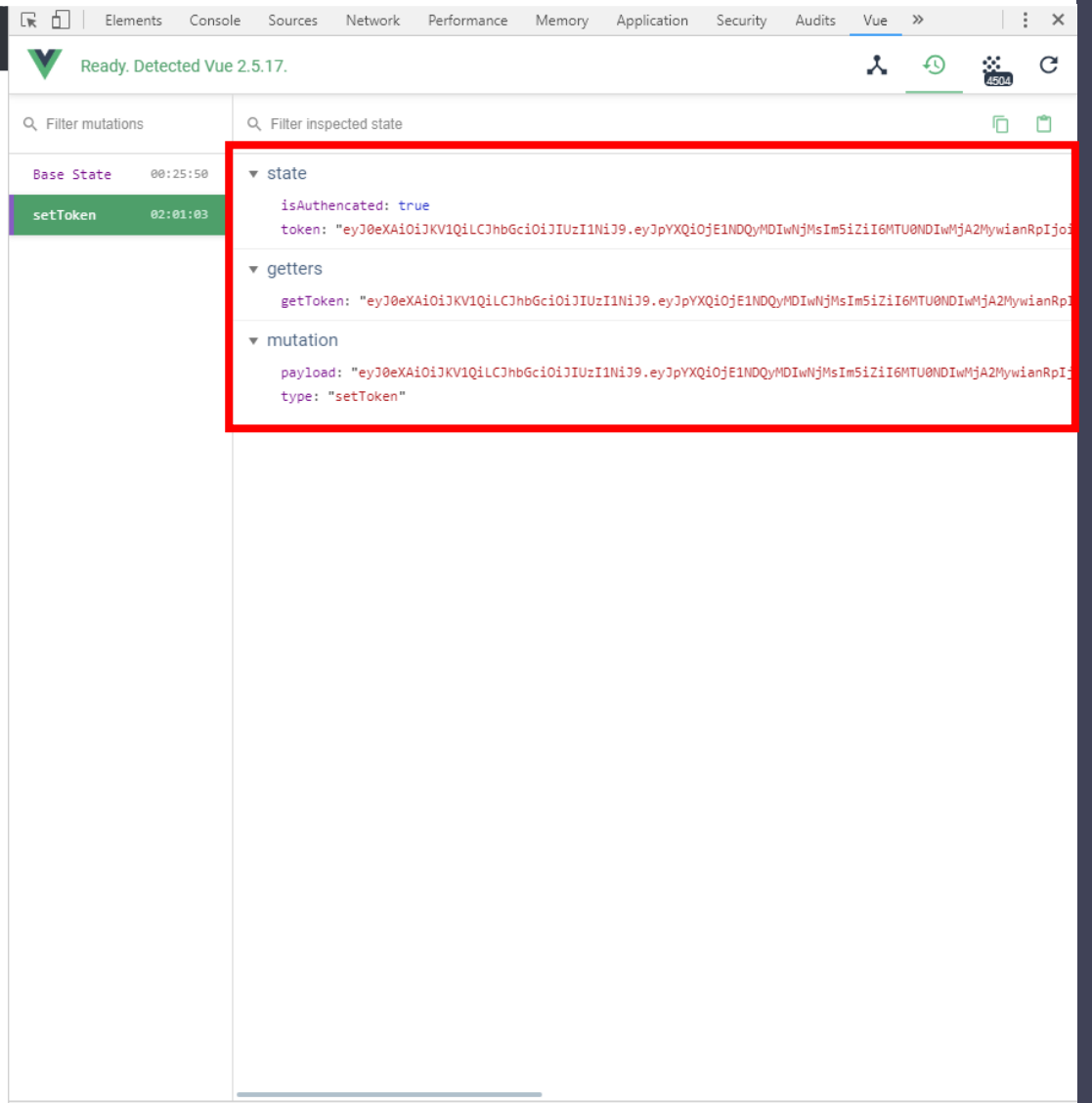


V. 프로젝트 결과

클라이언트

웹 페이지

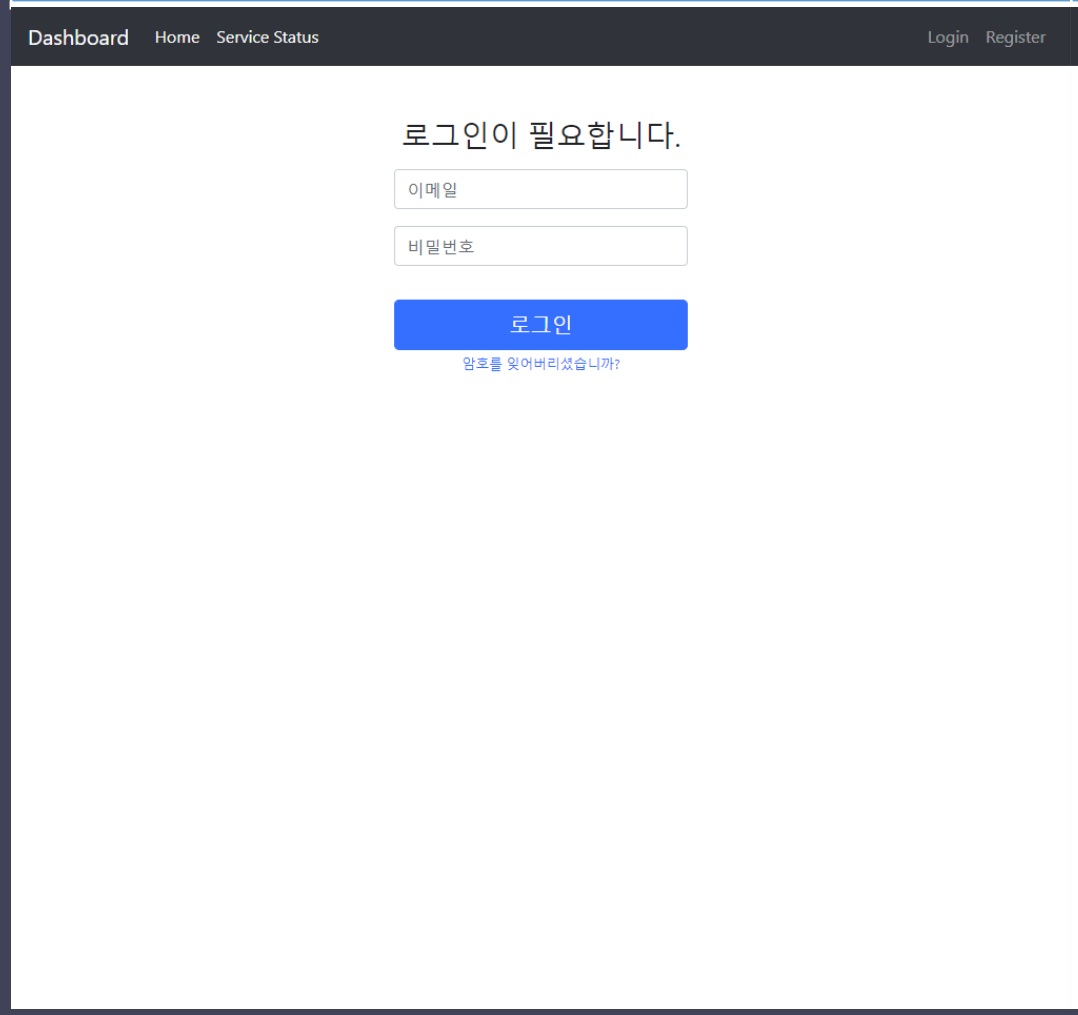
디버그 상태창



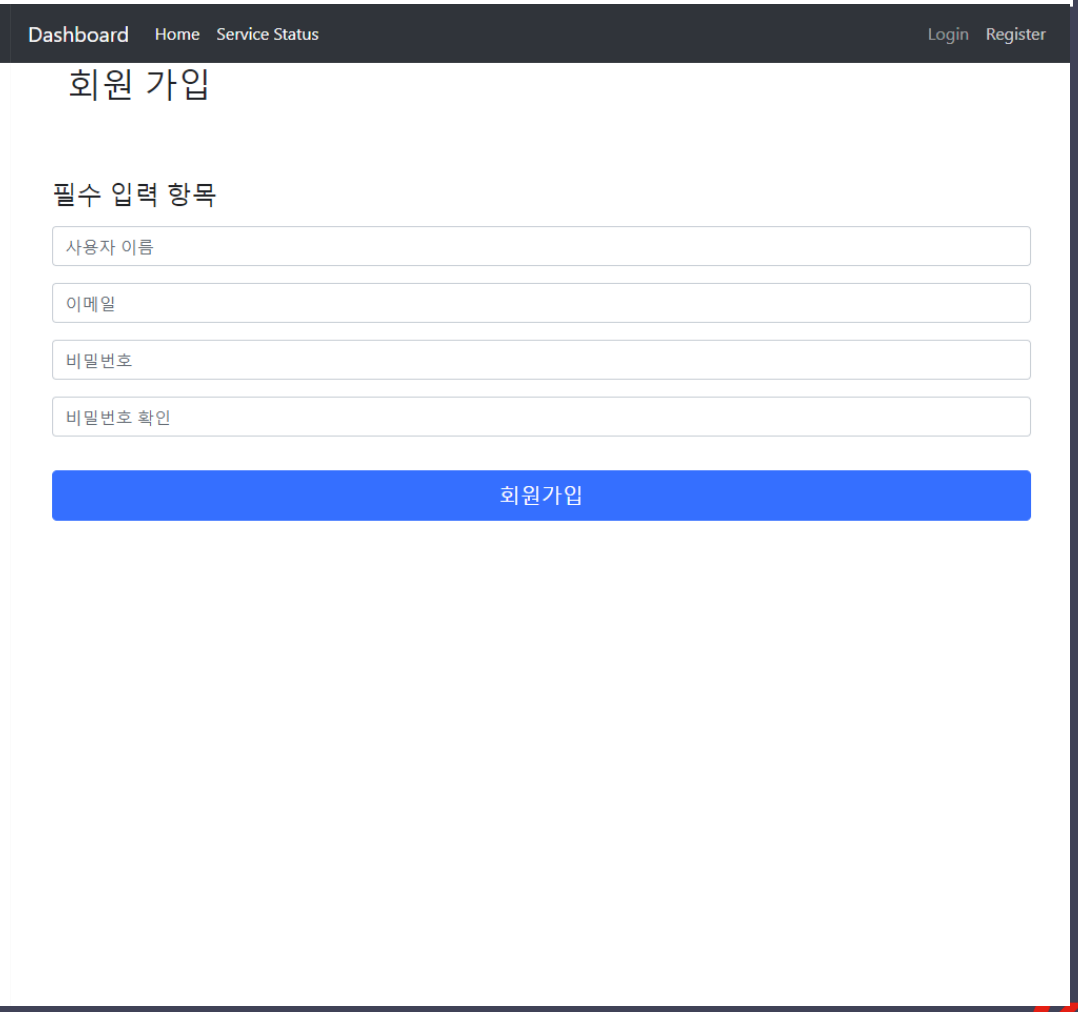
V. 프로젝트 결과

클라이언트

로그인 페이지



회원가입 페이지



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