2일차

오픈소스와 클라우드 개발 기초

동아대학교 소프트웨어혁신센터 박종규 SW교수

2025. AI·SW 아카데미

0. 강의에 들어가기 전에

강의 개요

- **강의 목적**
  - 오픈소스, 클라우드의 개념을 익히고 학생들이 프로젝트 개발에 활용하길 바람
  - 프로젝트에서 서버의 필요성에 대해 인지하길 바람
  - 컨테이너 기술(docker)에 대한 실습과 이해를 통해 클라우드 서비스에 대한 이해를 높임
  - 프로젝트 개발의 전 단계를 학습하여 본인만의 포트폴리오 개발을 하길 바람
- **강의 대상**
  - 최소한 대학 기초 교양 수준의 파이썬 프로그램 언어는 학습하고 이해하는 학생
- **기대 효과**
  - 본 수업을 들은 학생들은 프로젝트 개발 전 과정에 대한 학습을 토대로 프로젝트 수업에서 팀을 이끄는 핵심 멤버로서 성장하길 바람
  - 실제 시연되는 포트폴리오를 유지하여 공모전, 해커톤, 대회, 취업 포트폴리오 등에 적극 활용하길 바람

CONTENTS

Day-1

Day-2

1. Day-1 Wrap-up

파이썬 플라스크 (Flask)

- 플라스크란?
  - 플라스크는 Python 기반의 웹 프레임워크
- Flask의 특징 및 활용사례
  - 빠르고 가벼운 웹 앱 개발이 가능하여 프로토 타입 개발에 용이
  - RESTful API를 쉽게 개발할 수 있음
  - 간단한 웹사이트나 개인 프로젝트용(상업용 서비스는 Django같은 다른 프레임워크를 추천)
  - 교육 목적으로 기본 웹 개념 학습에 사용
- 웹 개발 기본 개념
  - http: 서버와 웹 클라이언트(브라우저) 간 통신 프로토콜
  - 서버-클라이언트 모델
    - 클라이언트의 요청(Request)에 서버는 응답(Response)를 보냄
    - Flask는 서버 역할을 수행
  - RESTful 설계
    - REST 원칙에 따라 API를 설계
    - URL은 리소스를 나타내고 HTTP 메서드(GET, POST, PUT, DELETE 등)를 사용

파이썬 가상환경

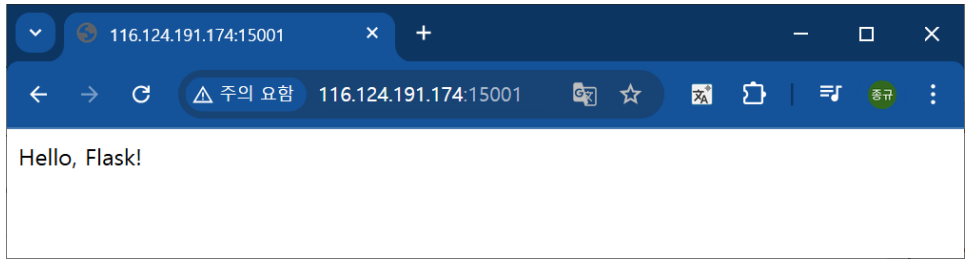
- 프로젝트 디렉터리 생성
  - $ mkdir ex1
- 파이썬 가상환경 설정
  - 가상환경을 사용하여 프로젝트간 종속성 충돌을 방지
  - 가상환경 생성
    - $ python3 –m venv venv
  - 가상환경 활성화
    - $ source venv/bin/activate
  - 가상환경 비활성화
    - $ deactivate
- 가상환경에 플라스크 설치
  - 가상환경 활성화 상태에서
    - (venv) $ pip install flask

ex1 - Hello, Flask!

- 간단한 플라스크 앱

```python
# app.py
from flask import Flask

app = Flask(__name__)

@app.route("/")
def home():
    return "Hello, Flask!"

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=x, debug=True)  // x는 본인에게 할당된 포트
```

- 실행 방법
  - $ python3 app.py
- 접속 방법
  - 브라우저 주소창에 아래 URL 입력, x는 본인에게 할당된 포트번호
    - 116.124.191.174:x



ex2 - HTML

- **HTML 파일 분리**
  - **HTML 템플릿 파일 (templates/index.html)**

```html
<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="UTF-8">
    <title>EX2 - 기본 HTML</title>
</head>
<body>
    <h1>Flask + HTML 실습</h1>
    <p>이 페이지는 Flask를 통해 렌더링되고 있습니다.</p>
</body>
</html>
```

  - **Flask 코드 (app.py)**

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('index.html')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=15001, debug=True)
```

ex2-2
HTML 기본태그

- **HTML 파일 분리**
  - **HTML 템플릿 파일 (templates/index.html)**
    - **실습자료 참고**
  - **Flask 코드 (app.py)**
    - **ex2와 동일**

ex2-3
form 데이터 전송

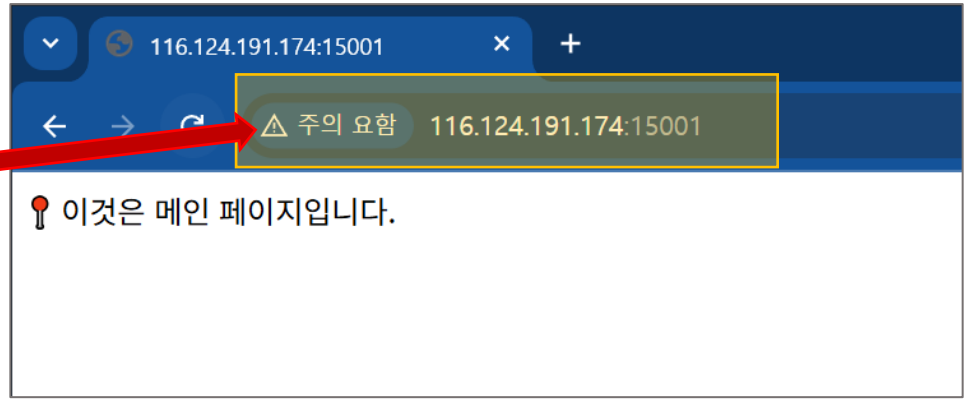
- **ex2-3 디렉터리 구조**

```
ex2-3/
 ├──── venv/          # 파이썬 가상환경
 ├──── app.py         # Flask 애플리케이션 실행 파일
 ├──── static/        # 정적 파일(CSS, JS, 이미지 등)
 │     └──── style.css      # 외부 CSS 파일
 └──── templates/           # HTML 템플릿 디렉터리
       ├──── form.html      # 입력 폼 페이지
       └──── result.html    # 입력값 출력 결과 페이지
```

ex3 – 라우팅(1/3)

- 라우팅
  - 라우팅은 URL 경로와 기능을 연결하는 과정

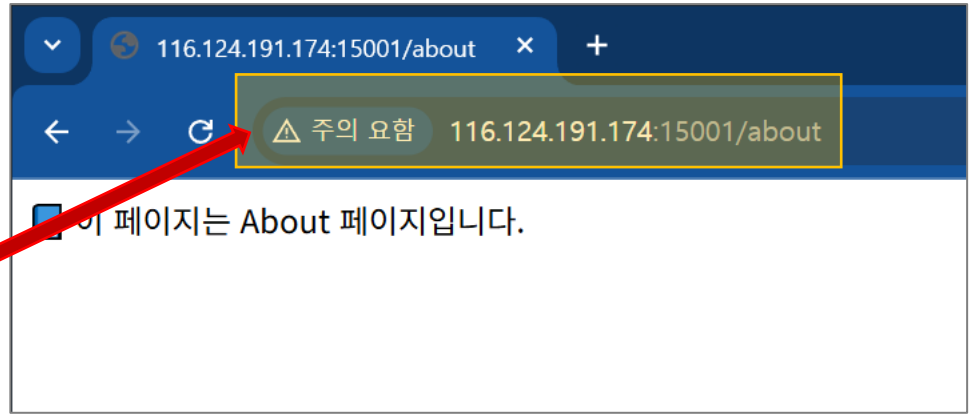
```python
from flask import Flask, render_template

app = Flask(__name__)

# 기본 루트 경로
@app.route('/')
def home():
    return "📍 이것은 메인 페이지입니다."

# 정적 라우트
@app.route('/about')
def about():
    return "📘 이 페이지는 About 페이지입니다."

# 동적 라우트 - 이름을 URL로 전달
@app.route('/hello/<name>')
def hello(name):
    return render_template('hello.html', name=name)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=15001, debug=True)
```



ex3 - 라우팅(2/3)

- 라우팅
  - 라우팅은 URL 경로와 기능을 연결하는 과정

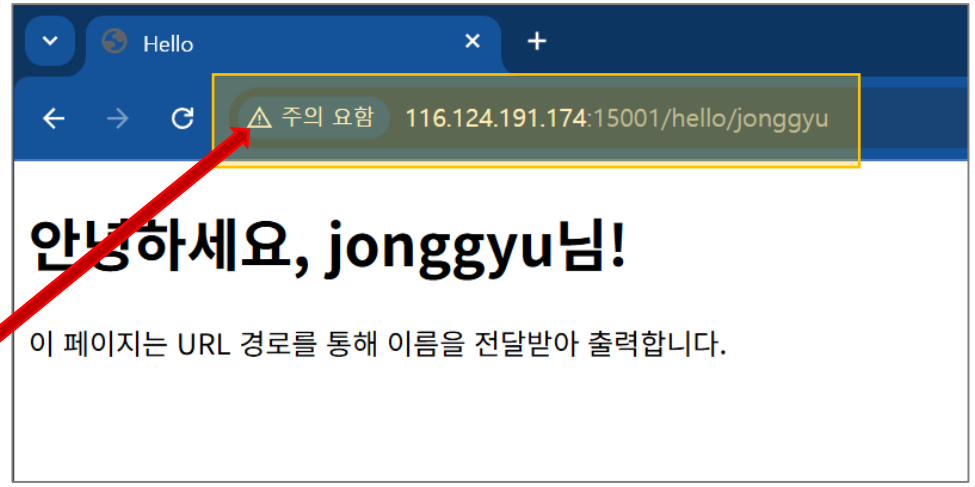
```python
from flask import Flask, render_template

app = Flask(__name__)

# 기본 루트 경로
@app.route('/')
def home():
    return "📍 이것은 메인 페이지입니다."

# 정적 라우트
@app.route('/about')
def about():
    return "📘 이 페이지는 About 페이지입니다."

# 동적 라우트 - 이름을 URL로 전달
@app.route('/hello/<name>')
def hello(name):
    return render_template('hello.html', name=name)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=15001, debug=True)
```



ex3 - 라우팅(3/3)

- 라우팅
  - 라우팅은 URL 경로와 기능을 연결하는 과정

```python
from flask import Flask, render_template

app = Flask(__name__)

# 기본 루트 경로
@app.route('/')
def home():
    return "📍 이것은 메인 페이지입니다."

# 정적 라우트
@app.route('/about')
def about():
    return "📘 이 페이지는 About 페이지입니다."

# 동적 라우트 - 이름을 URL로 전달
@app.route('/hello/<name>')
def hello(name):
    return render_template('hello.html', name=name)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=15001, debug=True)
```



ex3-2 동적 계산

- **동적 계산 및 잘못된 페이지(404) 처리**

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def home():
    return "🔢 숫자를 더하려면 /add/숫자1/숫자2 경로로 접속하세요."

@app.route('/add/<int:a>/<int:b>')
def add(a, b):
    result = a + b
    return render_template('result.html', a=a, b=b, result=result)

# 404 에러 커스터마이징
@app.errorhandler(404)
def page_not_found(error):
    return render_template('404.html'), 404

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=15001, debug=True)
```

2. REST API

REST API

- REST API는 REST 아키텍처를 기반으로 한 API(Application Programming Interface)
- REST API의 구성요소
  - **GET** : 리소스를 조회
  - **POST** : 새 리소스를 생성
  - **PUT** : 리소스를 갱신
  - **DELETE** : 리소스를 삭제
- REST API 사용의 장점
  - **간단함과 범용성**: HTTP 프로토콜을 사용하므로, REST API는 이해하기 쉽고 사용하기 간편함
  - **언어와 플랫폼 독립적**: 어떤 프로그래밍 언어나 플랫폼에서도 사용할 수 있음
  - **확장성과 유연성**: 새로운 리소스나 메서드를 기존 시스템에 쉽게 추가할 수 있음

JSON

- **JSON**
  - **JavaScript Object Notation은 데이터 교환을 위해 개발된 경량 데이터 형식**
  - **JSON은 언어 독립적이며, 대부분의 프로그래밍 언어에서 사용이 가능함**
  - **웹 개발에서 클라이언트<->서버 간의 데이터를 전송하는 표준 방식으로 많이 사용됨**
  - **{ }로 감싸져 있으면 key,value 쌍이 나와야 함. 순서는 상관없음**
  - **[ ]는 배열로 어떤 형태라도 들어갈 수 있으며 순서가 있음**

```json
{
  "name": "Park Jonggyu",
  "age": 19,
  "isStudent": false,
  "address": {
    "street": "Chungryeol-ro",
    "city": "Busan"
  },
  "phoneNumbers": ["010-1234-5678", "070-0000-9999"]
}
```

```python
# 객체를 JSON 문자열로 변환
json_obj = {"name": "Jong", "age": 19}
json_string = json.dumps(json_obj)
print(json_string)

# JSON 형태의 문자열을 객체로 변환
json_string = '{"name": "Jong", "age": 19}'
json_obj = json.loads(json_string)
print(json_obj["name"])
```

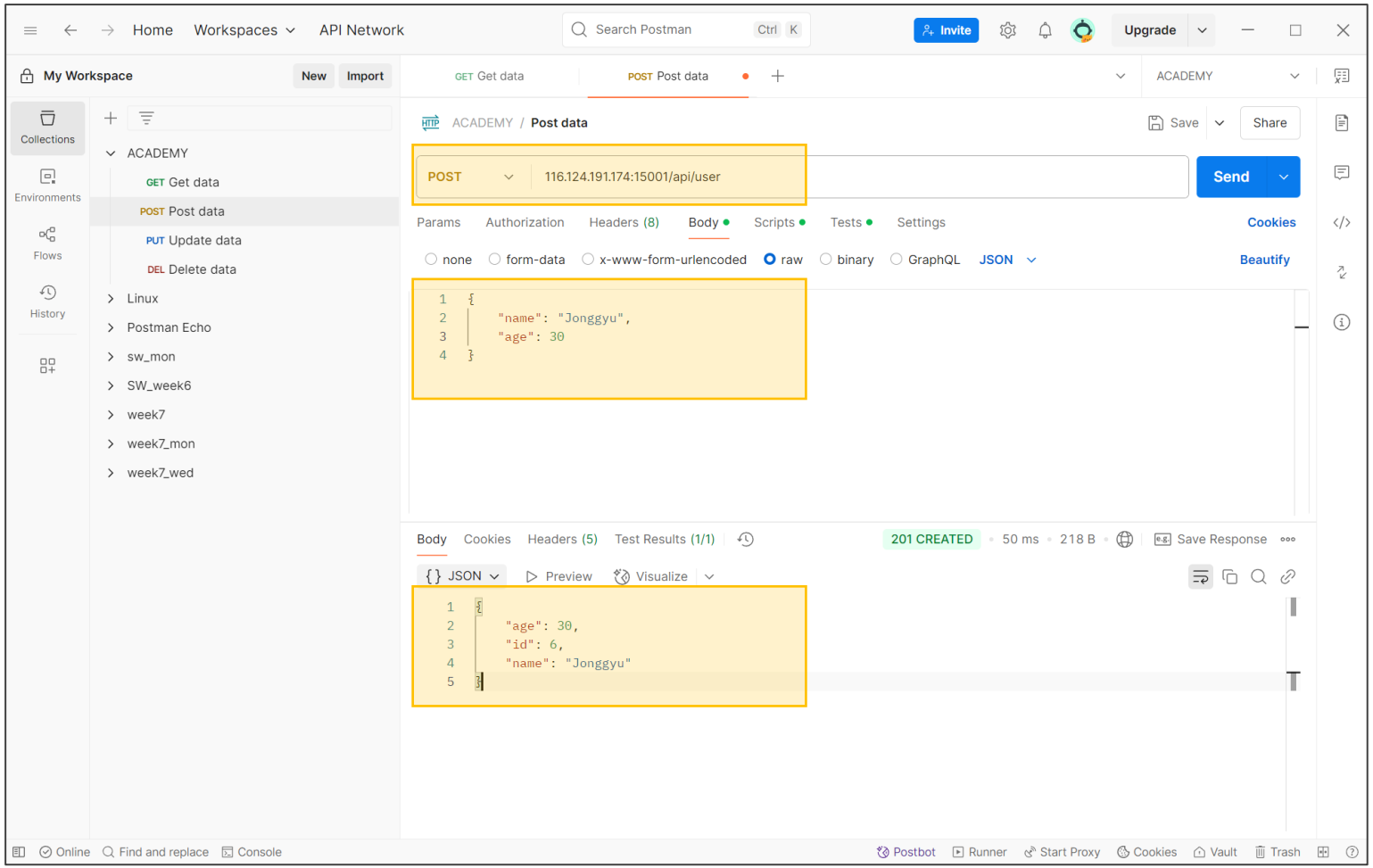
ex4
REST API

- **사용자 관리를 위한 REST API**

API	METHOD	URL	사용법
사용자 조회	GET	/api/users	/api/users
특정 사용자 조회	GET	/api/user/<int:user_id>	/api/user/3
사용자 추가	POST	/api/user	/api/user body:raw, JSON {"name":"Kang", "age":20}
사용자 수정	PUT	/api/user/<int:user_id>	/api/user/3 body:raw, JSON {"name":"Jo", "age":21}
사용자 삭제	DELETE	/api/user/<int:user_id>	/api/user/3

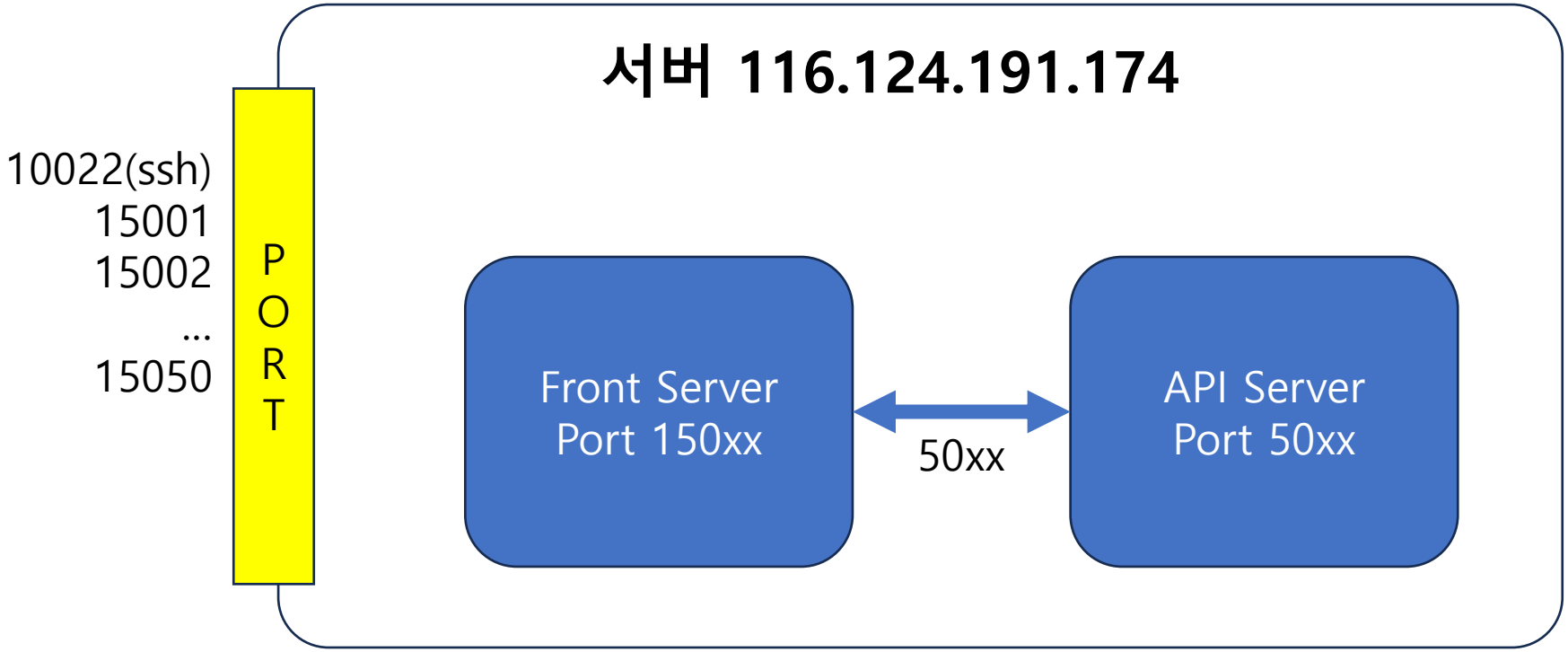
ex4
POSTMAN

- **POSTMAN을 사용한 REST API 호출**



ex4-2
Front+Back

- 사용자 관리를 위한 REST API 서버(ex4, Backend)는 내부 포트로 바꾸어 활용
- 프론트서버를 개인에게 할당된 외부 포트로 활용



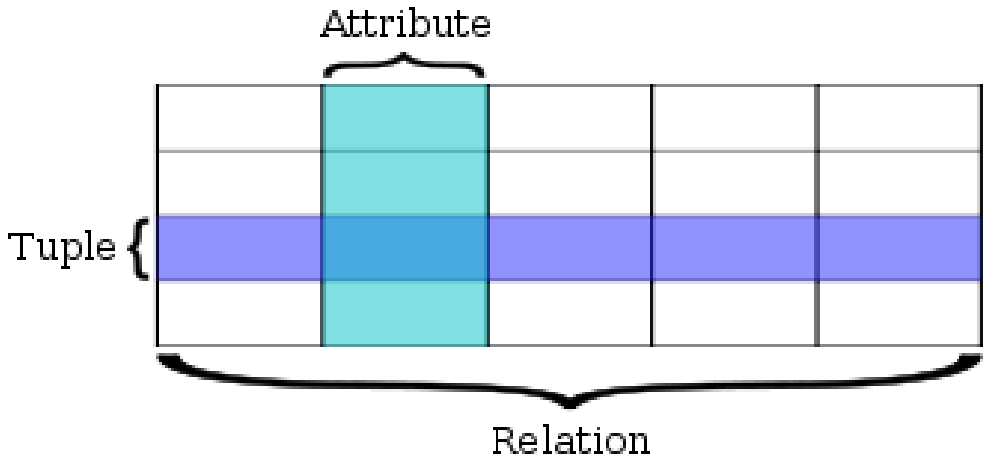
3. MySQL 연동

데이터베이스

- **데이터베이스**
  - 여러 사람이 데이터를 공유해서 사용하기 위해 체계적으로 통합해서 묶어놓은 데이터 집합
  - 컴퓨터 시스템에서 데이터베이스는 데이터 집합을 관리하고 접근할 수 있게 하는 프로그램을 의미함
- **데이터베이스의 장점**
  - 데이터 중복 최소화 및 저장공간 절약
  - 효율적인 데이터 접근 및 빠른 접근 속도
  - 일관성, 무결성, 보안성 제공
  - 데이터 표준화

관계형 데이터베이스

- **관계형 데이터모델(Relational Data Model)**
  - 데이터를 테이블에 대입하여 관계를 묘사하는 이론적 모델
  - 관계형 데이터모델을 다루는 데이터베이스 시스템을 관계형 데이터베이스 관리시스템(Relational DataBase Management System, RDBMS)이라고 함



SQL

- **SQL(Structured Query Language)**
  - SQL은 관계형 데이터베이스를 관리하기 위해 사용하는 질의 언어
  - 데이터베이스 스키마(Schema) 생성
  - 데이터 생성(Create)
  - 데이터 읽기(Retrieve, Read)
  - 데이터 수정(Update)
  - 데이터 삭제(Delete)
  - CRUD는 데이터를 관리하기 위해 기본적으로 필요한 기능

NoSQL

- **NoSQL**
  - SQL이 아니라는 의미에서 NoSQL이라고 하기도 하고 “Not Only SQL” 이라는 의미에서 전통적인 관계형 데이터베이스의 한계를 극복하기 위한 다양한 저장 방식을 지원함
- **NoSQL 주요 개념**
  - **스키마 유연성**
    - RDBMS와 달리 사전에 정의된 스키마가 필요하지 않아 데이터를 자유롭게 저장할 수 있음
    - 구조가 자주 변경 되는 경우에 유리함
  - **수평적 확장**
    - 데이터를 여러 서버에 분산하여 저장할 수 있음
    - 대규모 서비스에 유리함
  - **다양한 데이터 모델**
    - 키-값: Redis, DynamoDB
    - 문서: MongoDB, CouchDB
    - 그래프: Neo4j

MySQL

- MySQL은 전 세계적으로 널리 사용되는 오픈 소스 데이터베이스 관리 시스템
- 빠르고 안정적이며 무료로 사용할 수 있어 널리 활용
- 주요 특징
  - **오픈 소스**: 오픈 소스로 누구나 무료로 사용할 수 있고 소스코드 수정이 가능
  - **SQL 언어 지원**: MySQL은 데이터베이스 관리를 위해 SQL(Structured Query Language)을 사용함. 이를 통해 데이터를 삽입, 조회, 수정, 삭제를 할 수 있음
  - **대규모 데이터 처리**: MySQL은 수백만개의 정보도 빠르게 처리할 수 있으며 여러 사용자가 동시에 접근해도 안정성 유지
  - **다양한 플랫폼 지원**: MySQL은 Windows, Linux, macOS 등 다양한 운영체제에서 사용할 수 있음

MySQL

- **MySQL 기본 명령어**
  - **셸에서 mysql에 접속하여 mysql 콘솔에서 명령어를 작업**
  - **모든 명령어는 세미콜론으로 끝남**
  - **(참고) 이번 아카데미 실습에서는 접속 가능한 DB를 일괄 제공함**
    - **mysql 계정 == mysql 비밀번호 == linux 계정 == DB_NAME**

명령어	내용
$ mysql –u *ACCOUNT* –p	shell에서 mysql에 ACCOUNT 계정으로 접속 (student_150xx) 참고: mysql 비밀번호는 ssh 접속 시 바꾼 비밀번호가 아니라 계정명과 같음
show databases;	데이터베이스 목록 출력
create database *DB_NAME;*	데이터베이스 DB_NAME 생성
use *DB_NAME;*	사용할 데이터베이스로 DB_NAME을 선택

MySQL

- **MySQL 기본 명령어 (1/2)**

내용	예시 코드
데이터베이스 생성	CREATE DATABASE **student_150xx**;
데이터베이스 조회	SHOW DATABASES;
데이터베이스 선택	USE **student_150xx**;
테이블 조회	SHOW TABLES;
테이블 생성	CREATE TABLE users (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(100), age INT);
테이블 스키마 조회	DESCRIBE users;

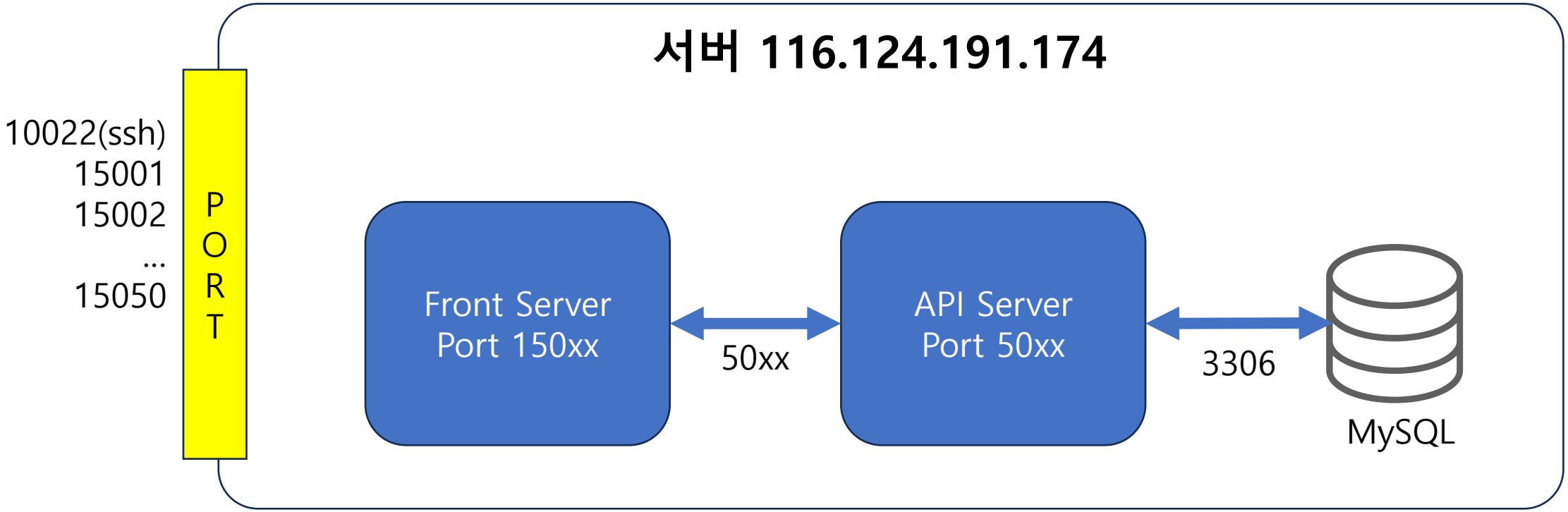
MySQL

- **MySQL 기본 명령어 (2/2)**

내용	예시 코드
데이터 삽입	INSERT INTO users (name, age) VALUES ('Jonggyu', 17);
데이터 조회	SELECT * FROM users;
데이터 조회(조건)	SELECT * FROM users WHERE name = 'Jonggyu';
데이터 업데이트	UPDATE users SET age = 15 WHERE name = 'jonggyu';
데이터 삭제	DELETE FROM users WHERE name = 'jonggyu';
테이블 수정	ALTER TABLE users ADD COLUMN email VARCHAR(100);
테이블 삭제	DROP TABLE users;
데이터베이스 삭제	DROP DATABASE **student_150xx**;

ex5
Front + Back + DB

- 사용자 관리를 위한 REST API 서버(ex5, Backend)는 내부 포트로 바꾸어 활용
- API 서버는 MySQL DB와 연동
- 프론트서버를 개인에게 할당된 외부 포트로 활용, 프론트 서버(ex4-2)는 변동 없음

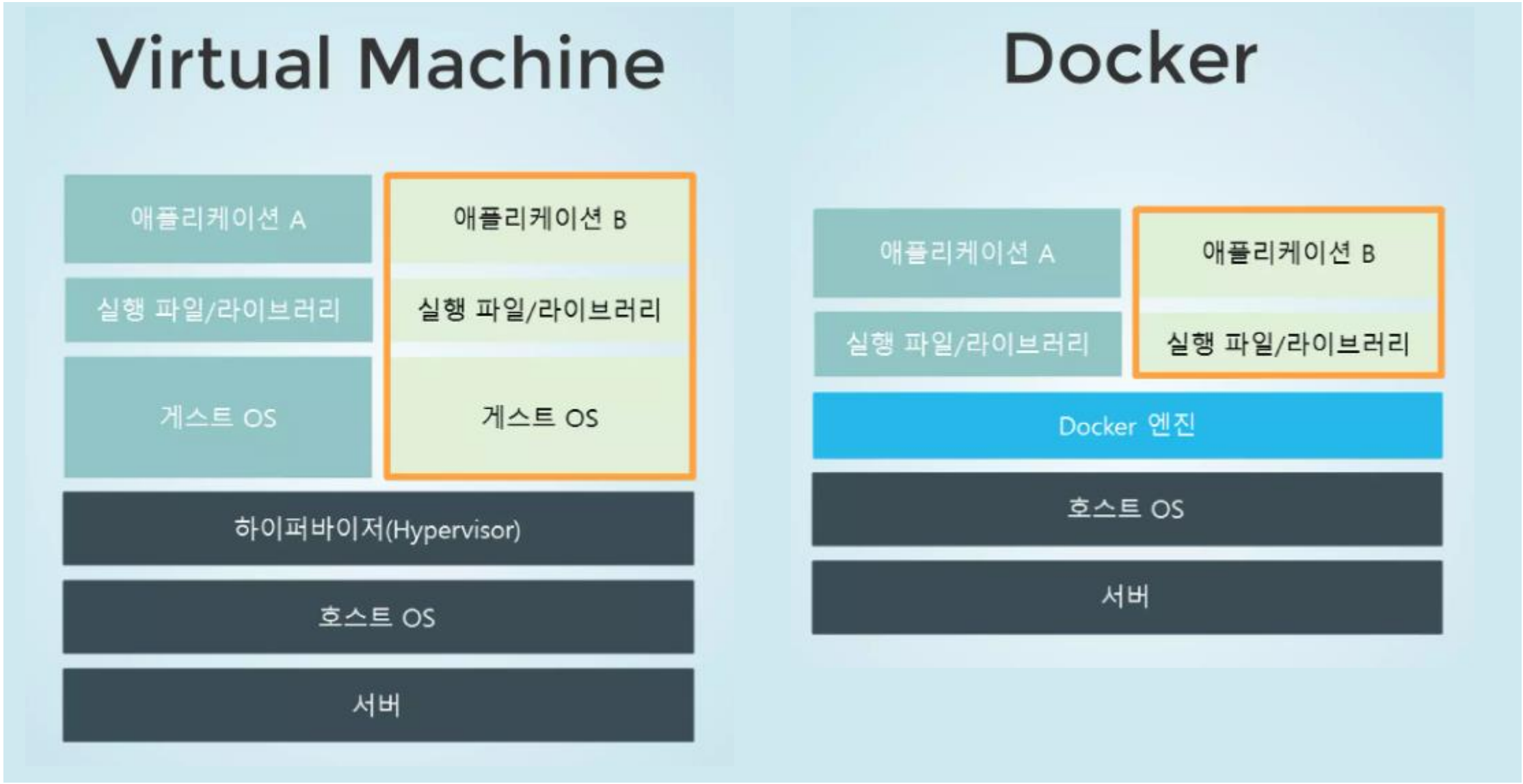


4. Docker

도커

- **도커(Docker)**
  - 도커는 애플리케이션을 컨테이너에 담아 실행할 수 있게 하는 컨테이너 기반 가상화 기술
  - 개발자가 만든 애플리케이션을 컨테이너에 넣으면 어디서나 동일한 환경에서 실행
- **컨테이너(Container)**
  - 애플리케이션과 그 실행에 필요한 모든 것을 포함한 독립적인 실습환경
  - 컨테이너는 OS 커널을 공유하며 앱마다 격리된 환경을 제공함
  - 가상머신(VM)보다 가볍고 빠름

VM과 도커


Virtual Machine
애플리케이션 A
실행 파일/라이브러리
게스트 OS
애플리케이션 B
실행 파일/라이브러리
게스트 OS
하이퍼바이저(Hypervisor)
호스트 OS
서버
Docker
애플리케이션 A
실행 파일/라이브러리
애플리케이션 B
실행 파일/라이브러리
Docker 엔진
호스트 OS
서버

도커 구성요소

- 이미지(Image)
  - 컨테이너 실행에 필요한 파일과 설정을 묶어둔 파일
  - 운영체제의 프로그램과 비슷한 형태
- 컨테이너(Container)
  - 이미지를 실행한 상태로 독립적인 애플리케이션 실행 환경
  - 운영체제의 프로세스와 비슷한 형태
- 도커파일(Dockerfile)
  - 도커 이미지를 생성하기 위한 설정파일
  - 빌드 스크립트와 비슷한 형태
- 도커 컴포즈(Docker Compose)
  - 여러 컨테이너를 정의하고 실행하는 도커 유틸리티
- 도커 레지스트리(Docker Registry)
  - 도커 레지스트리는 도커 이미지를 저장함
  - 도커는 기본적으로 Docker Hub라는 공개 레지스트리에서 이미지를 검색함

도커 실행 예시

```
$ docker run -i -t ubuntu /bin/bash
```

1. 로컬에 이미지 가 없으면 ubuntu Docker는 수동으로 실행한 것처럼 구성된 레지스트리에서 이미지를 가져옵니다.
2. Docker는 마치 수동으로 명령을 실행한 것처럼 새로운 컨테이너를 만듭니다.
3. Docker는 컨테이너에 읽기-쓰기 파일 시스템을 최종 계층으로 할당합니다. 이를 통해 실행 중인 컨테이너는 로컬 파일 시스템에서 파일과 디렉토리를 만들거나 수정할 수 있습니다.
4. Docker는 네트워크 옵션을 지정하지 않았기 때문에 컨테이너를 기본 네트워크에 연결하기 위한 네트워크 인터페이스를 만듭니다. 여기에는 컨테이너에 IP 주소를 할당하는 것이 포함됩니다. 기본적으로 컨테이너는 호스트 머신의 네트워크 연결을 사용하여 외부 네트워크에 연결할 수 있습니다.
5. Docker는 컨테이너를 시작하고 .을 실행합니다 /bin/bash. 컨테이너가 대화형으로 실행되고 터미널에 연결되어 있기 때문에(-i 및 -t 플래그로 인해) Docker가 터미널에 출력을 기록하는 동안 키보드를 사용하여 입력을 제공할 수 있습니다.
6. exit명령을 종료하기 위해 실행하면 /bin/bash컨테이너는 중지되지만 제거되지 않습니다. 다시 시작하거나 제거할 수 있습니다.

참조: https://docs.docker.com/get-started/docker-overview/

도커 기본 명령어

- **이미지 관리**

명령어	내용
$ docker pull 이미지	도커 허브에서 이미지를 다운로드
$ docker images	로컬에 저장된 이미지 조회
$ docker rmi 이미지	로컬에 저장된 이미지 삭제

도커 기본 명령어

- **컨테이너 관리**

명령어	내용
$ docker run [옵션] [이미지] [커맨드]	컨테이너를 생성하고 실행 (예: docker run ubuntu ls) 주요 옵션: -it : 터미널에서 상호작용 가능하게 실행 -d : 백그라운드에서 실행 --name : 컨테이너에 이름 부여 -p [host port]:[container port] : 포트매핑
$ docker ps	현재 실행 중인 컨테이너 조회
$ docker start [컨테이너]	컨테이너 시작 (마찬가지로 stop, restar가 있음)
$ docker rm [컨테이너]	종료된 컨테이너 삭제
$ docker exec –it [컨테이너] /bin/bash	실행 중인 컨테이너에 접속
$ docker logs [컨테이너]	컨테이너 로그 확인

도커 기본 명령어

- 그 외

명령어	내용
$ docker build –t [이미지] .	dockerfile을 기반으로 이미지를 생성
$ docker network ls	도커에서 사용 중인 네트워크 확인
$ docker network create [네트워크]	새로운 네트워크를 생성
$ docker network connect [네트워크] [컨테이너]	컨테이너를 네트워크에 연결
$ docker volume create [볼륨]	도커 볼륨을 생성
$ docker volume ls	도커 볼륨 목록을 확인
$ docker run –v [볼륨]:[컨테이너 경로] [이미지]	컨테이너 실행 시 볼륨(또는 host 디렉터리)을 연결 예: docker run –v my-volume:/data/ ubuntu

ex-6
도커 만들기 1 (1/3)

- 플라스크 프로젝트(ex2-2)가 가상환경에서 실행 중이라고 가정
- 플라스크 프로젝트(ex2-2) 경로에서
  - $ pip freeze > requirements.txt
- 플라스크 프로젝트 경로에서 Dockerfile을 생성
- 도커파일이 위치한 디렉터리에서 도커 이미지를 빌드
  - $ docker build –t ex_150xx . <- 현재 경로를 뜻하는 . 을 빠트리면 안됨
- 도커 컨테이너 실행
  - $ docker run –d –p 150xx:150xx ex_150xx

ex-6
도커 만들기 1 (2/3)

- 도커 파일(Dockerfile)은 내 프로젝트를 도커 이미지로 만들기 위한 일종의 빌드 스크립트
- 유명 오픈소스 등에서 찾을 수도 있음

```
# 1. 베이스 이미지 선택
FROM python:3.9-slim

# 2. 컨테이너 내부의 작업 디렉터리 생성 및 설정
WORKDIR /app

# 3. 의존성 파일 복사
COPY requirements.txt requirements.txt

# 4. 의존성 설치
RUN pip install --no-cache-dir -r requirements.txt

# 5. 애플리케이션 코드 복사
COPY . .

# 6. Flask 실행 포트 노출
EXPOSE 150xx

# 7. 애플리케이션 실행 명령
CMD ["python", "app.py"]
```

ex-6
도커 만들기 1 (3/3)

- **도커 컴포즈(Docker compose)**
  - **yml(야믈) 파일로 여러 개의 컨테이너를 동시에 관리할 수 있는 유틸리티**
  - **앞의 Dockerfile이 있다는 가정하에, 아래와 같이 docker-compose.yml 파일 작성**

```
version: '3.8'
services:
  web:
    build: .
    ports:
      - "150xx:150xx"
```

- **도커 컴포즈 명령어**
  - **$ docker compose up -d**
  - **$ docker compose down**

참고: 초기에는 Docker compose 명령어가 "$ docker-compose up" 이었음.
현재는 Docker compose가 도커의 정식 유틸리티로 채용되며 명령어 호출 구조가 조금 바뀜

ex-7
Front+Back+DB+docker

- **전체 프로젝트 가상화**
  - 지금까지 학습한 프론트서버, API서버를 Docker로 가상화
  - 실습환경상 DB는 물리 서버에 설치된 것을 그대로 활용
  - 개별로 할당된 포트 주소 및 .env 파일 변경
- **ex-7 프로젝트 디렉터리 구조**

```
ex-7/
├── backend/                          # ex5(API서버 DB버전)
│   ├── app.py
│   ├── requirements.txt
│   └── Dockerfile
├── frontend/                         # ex4-2(Front서버)
│   ├── app.py
│   ├── templates/
│   │   └── index.html
│   ├── static/
│   │   ├── main.js
│   │   └── style.css
│   └── Dockerfile
├── docker-compose.yml
└── .env                              ← (옵션) 백엔드용 DB 설정
```

추가로 학습해야할 것

- **네트워크**
  - 각 도커 컨테이너들은 별개의 가상화 공간으로, 각각 1대의 서버와 같이 동작함
  - 이 컨테이너들을 서로 연결하고 서비스를 제공하기 위해서는 아래와 같은 구성의 이해가 필요
    - 실제 물리서버의 네트워크
    - 도커 컨테이너들의 네트워크와 포트
    - 서비스를 실행하는 클라이언트의 네트워크
  - Localhost(127.0.0.1)는 “내 컴퓨터”를 의미하는 것으로, 이 주소가 호출 된 위치에 따라 어느 컴퓨터를 지칭하는지 알 수 없으므로 조심해서 사용해야 함

5. 프로젝트 설계

프로젝트

- **프로젝트**
  - 정의: 목표를 달성하기 위해 일시적으로 수행하는 작업으로, 명확한 시작시점과 종료시점이 있음
  - 특징
    - **일시성:** 프로젝트는 일정 기간 동안만 진행되며 완료 시 종료
    - **명확한 목표:** 프로젝트는 명확한 목표를 가지고 있으며, 목표 달성을 위한 계획을 수립함
    - **한정된 자원:** 제한된 시간, 예산, 인력 등의 자원을 사용
    - **종료 시점:** 프로젝트는 목표가 달성되거나 수행할 필요가 없을 때 종료

프로덕트

- **프로덕트**
  - 정의: 사용자에게 지속적으로 가치를 제공하는 실체로, 휴대폰, 자동차와 같이 유형이거나, SW, 서비스, 앱과 같은 무형일 수 있음
  - 특징
    - **지속성:** 출시 후에도 계속해서 개선되며 사용자에게 새로운 가치를 제공할 수 있도록 관리함
    - **고객 중심:** 프로덕트는 사용자의 요구를 해결하거나 가치를 제공하는데 집중
    - **반복적 개선:** 프로덕트는 시장변화, 사용자 피드백에 따라 지속적으로 기능이 추가되거나 개선
    - **수명 주기:** 프로덕트는 아이디어->개발->출시->운영->종료에 이르는 수명주기를 가짐

프로젝트와 프로덕트의 차이점

• 차이점

구분	프로젝트	프로덕트
목적	특정 목표를 달성하기 위한 일시적인 작업	고객 문제 해결을 위한 지속적인 가치를 제공
기간	시작과 종료 시점이 있으며, 목표 달성 시 종료	출시 후에도 지속적으로 관리, 개선
자원 관리	제한된 기간 동안 특정 자원을 사용함	수명이 지속되는 동안 필요한 자원을 지속적으로 투입함
성과	완료 시 구체적인 산출물(소프트웨어, 보고서, 인프라 등)을 제공	지속적으로 사용자가 제품이나 서비스를 통해 가치를 얻음
종료 여부	목표 달성 시 종료	종료되지 않고 계속 유지보수와 기능 추가가 이루어짐. 필요 시 제품 수명 주기 종료
조직 내 위치	프로젝트는 조직의 한 부분에서 특정 작업을 수행하는 임시 팀이 구성될 수 있음	프로덕트는 조직의 중요한 자산으로, 제품팀이 상시 관리

프로젝트

- **프로젝트 전주기**
  1. **프로젝트 기획(초기 단계)**
  2. 프로젝트 계획 수립
  3. IT 서비스 요구사항 정의 및 분석
  4. IT 서비스 설계
  5. IT 서비스 개발
  6. 테스트 및 검증
  7. 프로젝트 실행 및 서비스 론칭
  8. 프로젝트 종료
  9. IT 서비스 운영
  10. 서비스 개선 및 종료

1.1 프로젝트 아이디어 도출
1.2 요구사항 분석 및 정의
1.3 사업 타당성 분석
1.4 자원 및 예산 계획 수립
1.5 프로젝트 범위(Scope) 설정
1.6 주요 이해관계자 파악
1.7 프로젝트 팀 구성 및 역할 정의
1.8 프로젝트 일정 및 마일스톤 설정

프로젝트

- **프로젝트 전주기**
  1. 프로젝트 기획(초기 단계)
  2. **프로젝트 계획 수립**
  3. IT 서비스 요구사항 정의 및 분석
  4. IT 서비스 설계
  5. IT 서비스 개발
  6. 테스트 및 검증
  7. 프로젝트 실행 및 서비스 론칭
  8. 프로젝트 종료
  9. IT 서비스 운영
  10. 서비스 개선 및 종료

2.1 프로젝트 관리 계획서 작성
2.2 세부 일정 관리 (WBS: Work Breakdown Structure)
2.3 리스크 관리 계획 수립
2.4 품질 관리 계획 수립
2.5 의사소통 계획 수립
2.6 자원 배분 및 관리 계획 수립
2.7 비용 관리 계획 수립

프로젝트

- **프로젝트 전주기**
  1. 프로젝트 기획(초기 단계)
  2. 프로젝트 계획 수립
  3. **IT 서비스 요구사항 정의 및 분석**
  4. IT 서비스 설계
  5. IT 서비스 개발
  6. 테스트 및 검증
  7. 프로젝트 실행 및 서비스 론칭
  8. 프로젝트 종료
  9. IT 서비스 운영
  10. 서비스 개선 및 종료

3.1 IT 서비스의 비즈니스 요구사항 분석
3.2 기능 요구사항 및 비기능 요구사항 수집
3.3 시스템 아키텍처 정의
3.4 데이터베이스 및 인프라 요구사항 정의
3.5 사용자 경험(UX) 및 인터페이스(UI) 요구사항 수집
3.6 보안 요구사항 분석

프로젝트

- **프로젝트 전주기**
  1. 프로젝트 기획(초기 단계)
  2. 프로젝트 계획 수립
  3. IT 서비스 요구사항 정의 및 분석
  4. **IT 서비스 설계**
  5. IT 서비스 개발
  6. 테스트 및 검증
  7. 프로젝트 실행 및 서비스 론칭
  8. 프로젝트 종료
  9. IT 서비스 운영
  10. 서비스 개선 및 종료

4.1 시스템 아키텍처 설계
4.2 데이터베이스 설계
4.3 API 및 통합 설계
4.4 UI/UX 설계
4.5 보안 설계
4.6 시스템 성능 및 확장성 설계

프로젝트

- **프로젝트 전주기**
  1. 프로젝트 기획(초기 단계)
  2. 프로젝트 계획 수립
  3. IT 서비스 요구사항 정의 및 분석
  4. IT 서비스 설계
  5. **IT 서비스 개발**
  6. 테스트 및 검증
  7. 프로젝트 실행 및 서비스 론칭
  8. 프로젝트 종료
  9. IT 서비스 운영
  10. 서비스 개선 및 종료

5.1 개발 환경 설정
5.2 백엔드 개발
5.3 프론트엔드 개발
5.4 데이터베이스 구축 및 연동
5.5 API 개발 및 통합
5.6 테스트 환경 구성
5.7 코드 품질 및 리뷰 절차 적용

프로젝트

- **프로젝트 전주기**
  1. 프로젝트 기획(초기 단계)
  2. 프로젝트 계획 수립
  3. IT 서비스 요구사항 정의 및 분석
  4. IT 서비스 설계
  5. IT 서비스 개발
  6. **테스트 및 검증**
  7. 프로젝트 실행 및 서비스 론칭
  8. 프로젝트 종료
  9. IT 서비스 운영
  10. 서비스 개선 및 종료

6.1 단위 테스트(Unit Test)
6.2 통합 테스트(Integration Test)
6.3 성능 테스트(Performance Test)
6.4 사용자 수용 테스트(User Acceptance Test, UAT)
6.5 보안 테스트
6.6 버그 수정 및 최종 검토

프로젝트

- **프로젝트 전주기**
  1. 프로젝트 기획(초기 단계)
  2. 프로젝트 계획 수립
  3. IT 서비스 요구사항 정의 및 분석
  4. IT 서비스 설계
  5. IT 서비스 개발
  6. 테스트 및 검증
  7. **프로젝트 실행 및 서비스 론칭**
  8. 프로젝트 종료
  9. IT 서비스 운영
  10. 서비스 개선 및 종료

7.1 프로젝트 결과물 전달
7.2 운영 환경으로 배포
7.3 최종 사용자 교육 및 매뉴얼 제공
7.4 서비스 마케팅 및 홍보 전략 수립
7.5 초기 사용자 피드백 수집 및 대응

프로젝트

- **프로젝트 전주기**
  1. 프로젝트 기획(초기 단계)
  2. 프로젝트 계획 수립
  3. IT 서비스 요구사항 정의 및 분석
  4. IT 서비스 설계
  5. IT 서비스 개발
  6. 테스트 및 검증
  7. 프로젝트 실행 및 서비스 론칭
  8. **프로젝트 종료**
  9. IT 서비스 운영
  10. 서비스 개선 및 종료

8.1 최종 보고서 작성
8.2 프로젝트 마무리 평가
8.3 프로젝트 산출물 검토 및 인수
8.4 예산 및 비용 검토
8.5 프로젝트 팀 해산
8.6 교훈 및 향후 개선사항 도출 (레슨 런트)

프로젝트

- **프로젝트 전주기**
  1. 프로젝트 기획(초기 단계)
  2. 프로젝트 계획 수립
  3. IT 서비스 요구사항 정의 및 분석
  4. IT 서비스 설계
  5. IT 서비스 개발
  6. 테스트 및 검증
  7. 프로젝트 실행 및 서비스 론칭
  8. 프로젝트 종료
  9. **IT 서비스 운영**
  10. 서비스 개선 및 종료

9.1 운영 계획 수립
9.2 서비스 모니터링
9.3 장애 대응 절차 수립 및 운영
9.4 성능 최적화 및 유지보수
9.5 사용자 지원 및 고객 서비스
9.6 정기적인 보안 점검 및 패치 관리
9.7 서비스 확장 및 기능 추가
9.8 운영 보고서 작성

프로젝트

- **프로젝트 전주기**
  1. 프로젝트 기획(초기 단계)
  2. 프로젝트 계획 수립
  3. IT 서비스 요구사항 정의 및 분석
  4. IT 서비스 설계
  5. IT 서비스 개발
  6. 테스트 및 검증
  7. 프로젝트 실행 및 서비스 론칭
  8. 프로젝트 종료
  9. IT 서비스 운영
  10. **서비스 개선 및 종료**

10.1 서비스 성과 평가
10.2 사용자 피드백 반영 및 개선 작업
10.3 새로운 요구사항 수용 및 기능 업데이트
10.4 서비스 종료 시나리오 작성
10.5 서비스 종료 절차 실행 및 사용자 안내
10.6 데이터 백업 및 시스템 종료

6. 클라우드 서비스 개발

클라우드 서비스 기획 실습

- **프로젝트 기획 예시 (배달 서비스) (1/3)**
  1. **서비스 개요 및 목표 설정**
     - 서비스 개요: 기존 음식 배달 서비스의 기능과 구조를 분석하여 간단히 정리
     - 목표 설정: 기존 서비스가 해결하고자 하는 문제 정의(예: 음식 주문의 편리성, 사용자 경험의 개선 등)
     - 기획 목표: 기획할 서비스의 주요 목표 설정(예: 사용자 편의성 증대, 배달 효율성 향상 등)
  2. **시장 조사 및 경쟁 분석**
     - 시장 분석: 음식 배달 서비스 시장의 동향 파악. 주요 사용자 요구사항과 트렌드 확인
     - 경쟁사 분석: 주요 경쟁 서비스들의 강점과 약점 분석
       - 경쟁 서비스의 차별점은 무엇인지
       - 해당 서비스의 성공 요인은 무엇인지
       - 경쟁 서비스의 UX/UI 분석

클라우드 서비스 기획 실습

- **프로젝트 기획 예시 (배달 서비스) (2/3)**

3. 타겟 사용자 정의

- 사용자 정의: 타겟 사용자를 세분화하여 사용자 유형 정의
  - 예: 자주 외식을 하는 20대, 직장인, 가족 단위의 고객 등
- 사용자 요구사항 분석: 사용자의 주요 요구사항을 도출(예: 빠른 배달, 다양한 결제 방법, 실시간 배달 추적 등)

4. 서비스 기능 목록(Feature List) 도출

- 핵심 기능 정의: 음식 배달 서비스의 필수 기능 도출
  - 사용자 가입 및 로그인
  - 음식점 및 메뉴 탐색
  - 주문 및 결제 시스템
  - 실시간 배달 추적
  - 리뷰 및 평가 시스템

클라우드 서비스 기획 실습

- **프로젝트 기획 예시 (배달 서비스) (3/3)**

5. 사용자 경험(UX) 및 인터페이스(UI) 설계

- 사용자 흐름(User Flow) 설계: 사용자가 서비스를 어떻게 이용하는지에 대한 흐름 설계
  - 예: 앱을 실행 → 음식점 선택 → 메뉴 주문 → 결제 → 배달 추적
- 와이어프레임 및 화면 설계: 서비스의 주요 화면(UI) 디자인
  - 로그인 화면, 메뉴 탐색 화면, 주문 결제 화면, 주문 확인 화면 등

6. 백엔드 설계

- 생략

7. 비즈니스 모델 설계

- 핵심수익 모델 정의: 서비스의 수익 구조 분석
  - 예: 음식점에서의 중개 수수료, 배달비, 광고 수익 등
- 비즈니스 확장성 분석: 서비스를 확장하기 위한 전략 기획
  - 예: 새로운 지역 진출, 다른 배달 서비스(예: 물류, 약 배달) 확장

감사합니다.