1일차

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. 오픈소스와 클라우드 개념

오픈소스

- **오픈소스란 무엇인가?**
  - 소스코드가 공개되어 누구나 자유롭게 사용, 수정, 배포할 수 있는 소프트웨어
  - 라이선스 조건에 따라 자유도의 범위가 다름
- **역사와 발전 배경**
  - 1980-90년대: 리처드 스톨먼의 GNU 프로젝트 + 리누스 토발즈의 리눅스 커널의 결합
    -> 자유 소프트웨어(Free Software) 운동
  - 1998년: Open Source Initiative(OSI) 설립 -> “자유(Free)” 대신 “오픈소스”라는 표현을 사용
- **오픈소스의 특징과 장점**
  - 누구나 참여 가능
  - 빠른 업데이트 및 버그 수정(그러나 많은 버그 또한 존재)

오픈소스

- **오픈소스는 무료?**
  - "오픈소스는 소스코드가 공개 돼있으니까 누구나 쓸 수 있고, 무료이며, 아무렇게나 써도 되는 것인가?"
    - 누구에게나 개방 되어있는 도서관(공공시설)에서 떠들고 더럽히며 무료로 빌리는 책을 훼손해도 될까?
    - 단순한 '무료(free)'와는 개념이 다름
- **무료 소프트웨어(Freeware)와 자유 소프트웨어(Free Software)**
  - 가격이 무료인 소프트웨어는 프리웨어라고 부르며, 자유 소프트웨어와는 차이를 둠

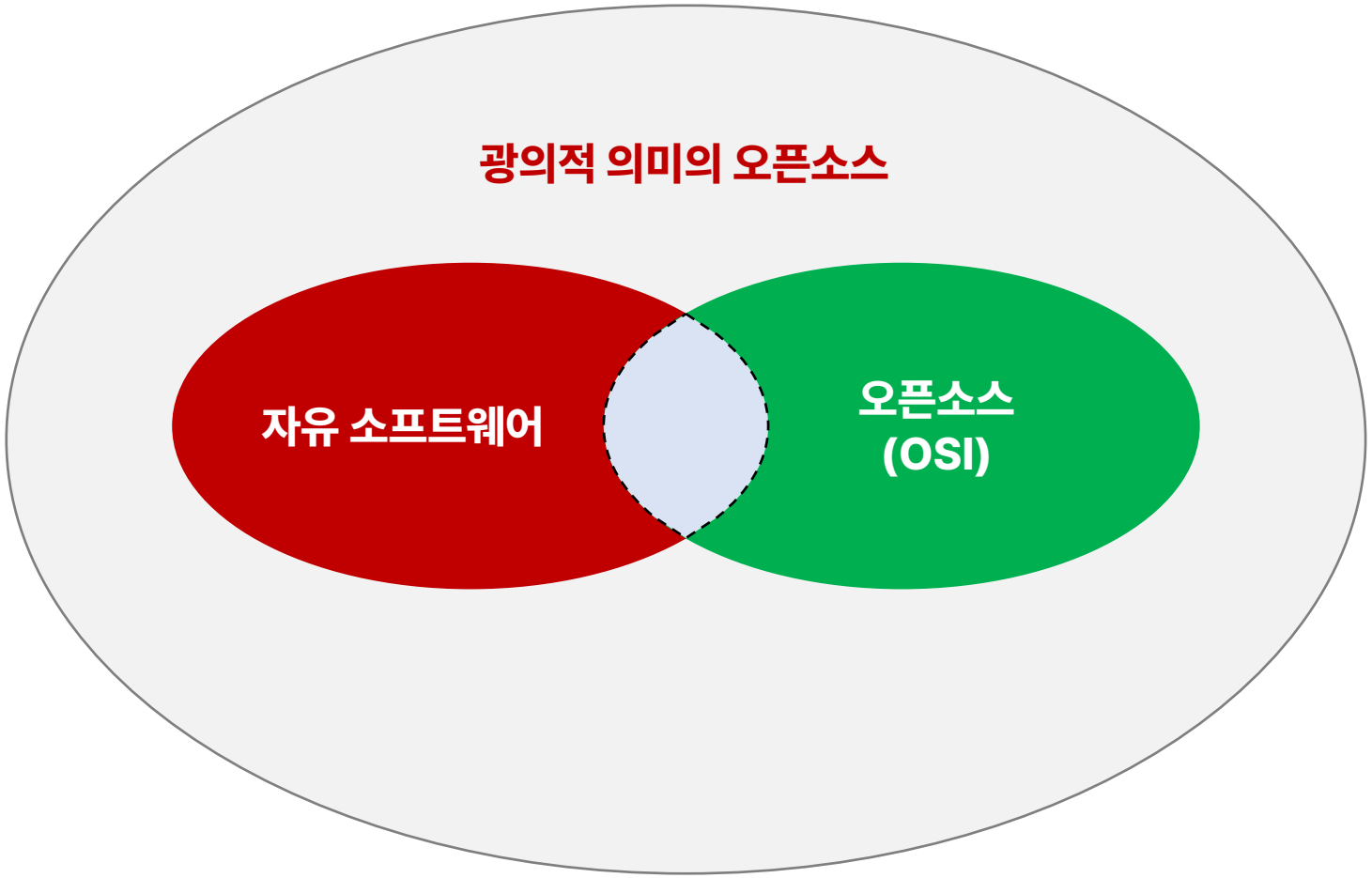
오픈소스 비교

- **오픈소스, 자유 소프트웨어, 프리웨어 비교**

항목	오픈소스	자유 소프트웨어	프리웨어
소스코드 공개	✅ 공개	✅ 공개	❌ 보통 비공개
자유도	✅ 사용/수정/배포 자유 (라이선스에 따라 다름)	✅ 4가지 자유 보장 (사용/수정/배포/공유)	✅ 사용만 가능 ❌ 수정/배포 불가
강조점	기술적 품질, 협업	윤리적 자유, 권리	무료 제공
상업적 사용	✅ 대부분 가능(조건 있음)	✅ 대부분 가능(조건 있음)	❌ 금지되거나 제한
예시	Python, Linux, git	Linux, git, GCC, LibreOffice	ALZIP, ZOOM,

오픈소스의 범위

- **오픈소스의 범위**



오픈소스와 자유 소프트웨어

• 오픈소스 V.S. 자유 소프트웨어

항목	오픈소스	자유 소프트웨어
출발점	**효율적 개발과 협업**을 위한 실용주의	**사용자의 자유와 권리**를 지키기 위한 운동
철학	"좋은 소프트웨어는 함께 만들면 더 낫다"	"소프트웨어는 자유롭게 쓸 수 있어야 한다."
우선순위	**개방성과 품질** 중심 -> 협업 도구로서의 가치 강조	**사용자 자유(4대 자유)** 강조 -> 독점 소프트웨어에 반대
대표 단체	OSI(Open Source Initiative)	FSF(Free Software Foundation)
라이선스 범위	**다양한 라이선스 수용** (MIT, Apache, BSD, GPL 등)	**대부분 GPL만 엄격히 수용**
정책적 태도	비공개 소프트웨어와 공존 가능, 기업 협업 환영	비공개 소프트웨어 반대, 상업용이라도 소스 공개 필수 주장
관용성	현실적이고 유연함 -> 기업도 쉽게 참여	원칙에 엄격 -> 기업이 꺼릴 수 있음

기업과 오픈소스

- **무료인 오픈소스로 기업은 어떻게 돈을 버는가?**
  - **오픈소스는 무료로 소스를 공개하지만, 그 주변에서 다양한 방식으로 수익을 창출**

수익 모델	설명	예시
SaaS 모델	오픈소스를 클라우드 서비스 형태로 유료 제공	MongoDB Atlas, GitLab, Red Hat OpenShift
Support 모델	기업에 기술지원, 유지보수, 컨설팅	Red Hat Linux, Ubuntu, Docker
Dual License	비상업적 사용은 무료, 기업용은 유료	MySQL, Qt
교육/출판/컨퍼런스	오픈소스 관련 책, 강의, 세미나 등을 통한 수익	O'Reilly, PyCon
비영리재단	기업 및 개인으로부터의 후원	Python(Python Software Foundation)

클라우드

- **클라우드란?**
  - 클라우드 컴퓨팅: 인터넷을 통해 컴퓨팅 자원(서버, 저장소, 데이터베이스 등)을 제공하는 기술
  - 예전에는 물리 서버를 직접 설치 -> 이제는 필요할 때 빌려서 사용
- **발전 배경**
  - 2000년대 중반: Amazon이 AWS(아마존 웹서비스) 출시 -> 클라우드 시대 개막
  - 이후 Google Cloud, Microsoft Azure 등 대형 클라우드 서비스 등장
  - 서버 유지보수 비용 절감, 확장성 용이, 접근성 증가 등으로 빠르게 보급
- **클라우드 서비스 모델**
  - IaaS(Infrastructure as a Service): 가상 서버, 스토리지 (예: AWS EC2)
  - PaaS(Platform as a Service): 개발 플랫폼 제공 (구글 앱 엔진)
  - SaaS(Software as a Service): 완제품 형태의 서비스 제공(Google Docs, Zoom, Discord)

클라우드 비유

- **IaaS, PaaS, SaaS 그리고 피자**

항목	내가 할 일	서비스가 제공	비유
On-Premise	전부 직접	없음	재료 사서 집에서 요리
IaaS	OS 위 설치는 직접	하드웨어, OS까지 제공	냉동 도우는 있지만 토핑은 직접 요리
PaaS	앱만 개발	런타임, 서버설정까지 제공	도우+기본토핑 제공, 추가토핑과 소스만 선택하여 직접 뿌림
SaaS	로그인	모든 것을 제공	배달

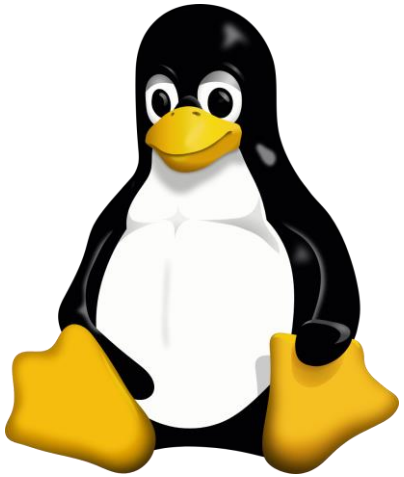
오픈소스와 클라우드

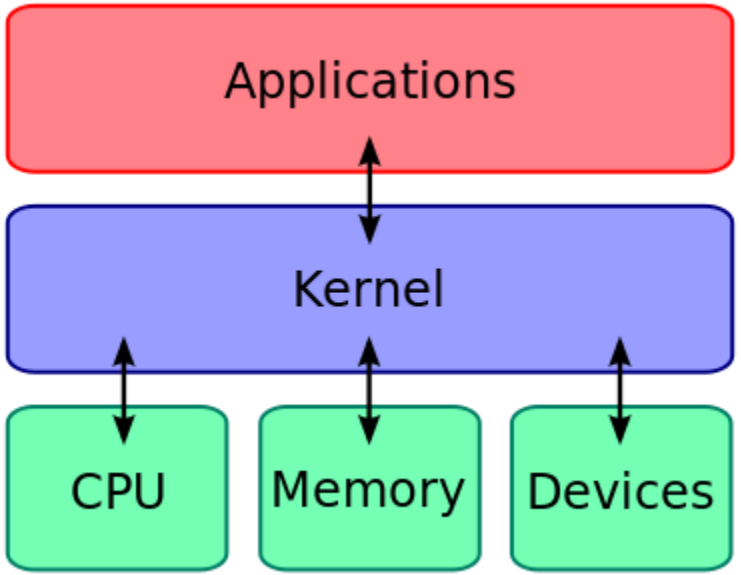
- **클라우드는 오픈소스의 힘으로 발전**
  - 리눅스는 클라우드 서버 OS의 표준
  - Kubernetes, Docker 등 대부분의 클라우드 기술은 오픈소스로 시작
- **클라우드는 오픈소스의 확산을 촉진**
  - 누구나 쉽게 서버를 띄우고 오픈소스 앱을 설치해 실험 가능
- **개방과 공유**
  - 오픈소스: 코드 공유
  - 클라우드: 자원 공유
  - 현대 개발환경은 오픈소스와 클라우드를 결합하여 서비스 개발

2. 기본 리눅스 명령어 실습

리눅스

- 다중 사용자, 다중 작업, 다중 스레드를 지원하는 네트워크 운영체제
- 본디 리눅스는 리누스 토발즈가 개발한 "커널"
  - 이후 운영체제로 확장
- 자유 소프트웨어와 오픈소스의 대표





리눅스의 역사

- 1983년 리처드 스톨만이 GNU 프로젝트를 시작
  - GNU(GNU is Not Unix) 프로젝트 : 자유 소프트웨어로만 구성된 유닉스 시스템을 만드는 것
- 1989년 GNU는 커널을 제외한 GNU 소프트웨어 완성
  - 시스템 라이브러리, 컴파일러, 텍스트 에디터, 쉘 등
- 리눅스 커널과 FSF의 GNU 소프트웨어가 결합
  - FSF는 운영체제 커널로 GNU Hurd를 개발하는 중이었으나 개발이 늦어짐
  - 이후 리누스 토발즈가 개발한 “리눅스 커널”이 GNU 프로젝트의 소프트웨어와 합쳐지며 운영체제로써의 “리눅스”가 성장함

리눅스 배포판

- **리눅스 배포판(Linux Distribution)**
  - 리눅스 커널과 GNU 소프트웨어 그리고 다양한 자유 소프트웨어들로 구성된 운영체제
  - 대표적인 리눅스 배포판으로는 RedHat, Fedora, Debian, Ubuntu, CentOS 등이 존재
- **초기 리눅스는 커널과 GNU 도구들이 모여있는 이미지로 배포**
  - 설치와 설정 등이 복잡하여 회사나 커뮤니티 단위로 이를 단순하게 하기 위해 여러가지 배포판들이 발생하게 됨
  - 배포판에 따라 기본적으로 설치 되어있는 프로그램이나 설치 경로가 조금씩 다름

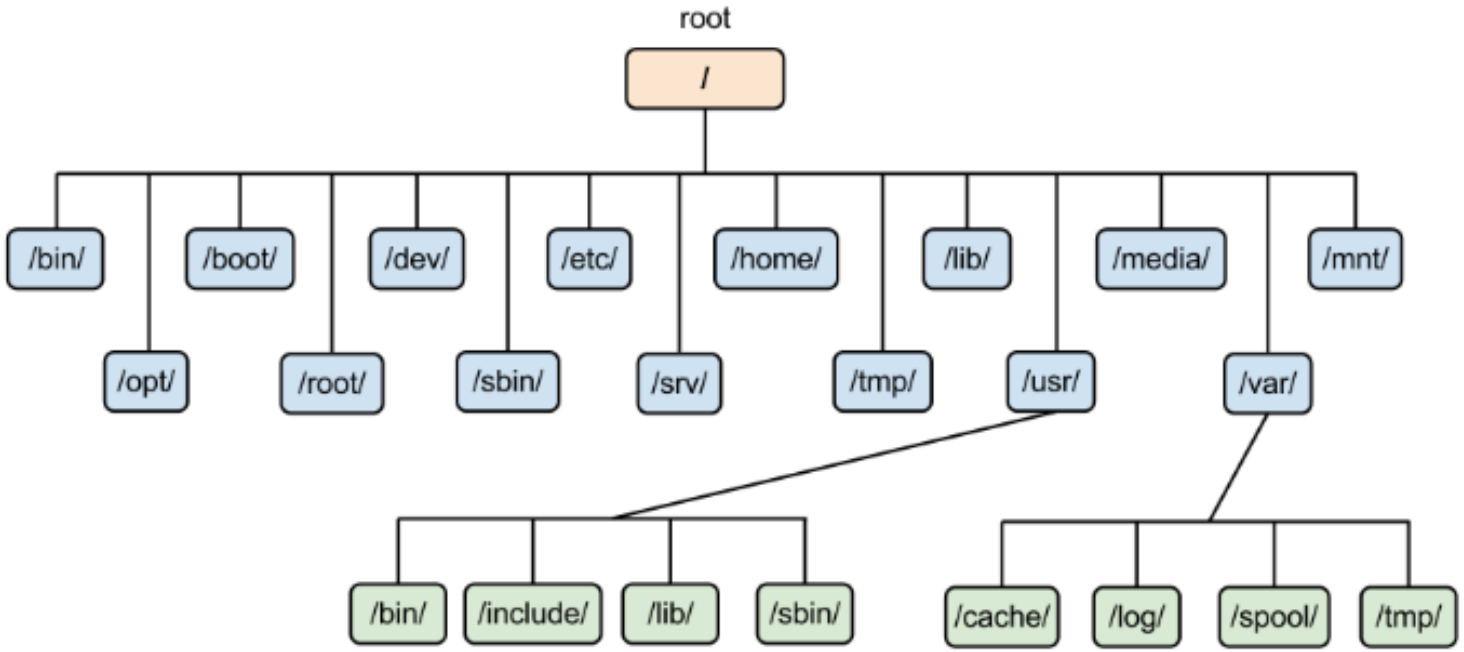
리눅스 배포판

- **리눅스 배포판(Linux Distribution)의 종류**

종류	소개	장점	단점
RedHat	레드햇 사에서 유료로 기술 지원을 하는 엔터프라이즈용 리눅스 RHEL	• 서버의 안정성이 높음 • 지속적인 관리와 최신기술 지원	• 유료
Fedora	레드햇 계열이며 페도라 프로젝트를 통해 개발하는 리눅스	• 레드햇 기술이 탑재됨	• 에러가 잦음
Debian	GNU의 공식 후원을 받는 유일한 배포판, 안정성이 좋아 다른 배포판들의 기반이 됨	• 높은 안정성 • 패키지가 많음	• 오래된 패키지들 • 상용과 호환 어려움
Ubuntu	데비안 계열, 현재 가장 널리 사용되고 있는 배포판 중 하나	• 쉬운 사용성	• 잦은 버전업 • LTS(Long Term Service)
CentOS	레드햇 계열, 레드햇 엔터프라이즈 버전과 완벽히 호환되는 무료 기업용 리눅스	• 무료 레드햇 사용 • 높은 안정성 • 한국에서 가장 많이 사용	• 레드햇 기술지원을 받을 수는 없음
SUSE	해외에서 사용률이 높으며 서버로써 높은 안정성을 보임	• 높은 안정성	• 저장소가 크지 않음
Mint	우분투 기반, 깔끔한 UI와 사용성 지원	• 개인 사용자에 적합한 데스크탑 환경 지원	• 잦은 버전업

리눅스 구조

- **리눅스 디렉터리 구조**



리눅스 구조

- **리눅스 디렉터리 구조**

디렉토리	내용
/	**기본 계층 모든 파일 시스템 계층의 기본인 루트 디렉토리**
/bin/	**모든 사용자를 위해 단일 사용자 모드에서 사용 가능해야 하는 명령어 바이너리 (예: C:₩Windows₩System32₩)**
/boot	부트 로더 파일
/dev	필요한 장치
/etc	설정파일, 바이너리는 안됨
/etc/opt/	/opt/에 대한 설정 파일
/etc/X11/	X 윈도 시스템의 설정 파일, 버전 11
/etc/sgml/	SGML 설정 파일
/etc/xml/	XML 설정 파일

리눅스 구조

- **리눅스 디렉터리 구조**

디렉토리	내용
/home/	**저장된 파일, 개인 설정, 기타 등을 포함한 사용자의 홈 디렉토리**
/lib/	/bin/과 /sbin/에 있는 바이너리에 필요한 라이브러리
/media/	CD-ROM과 같은 이동식 미디어의 마운트 지점
/mnt/	임시로 마운트된 파일 시스템
/opt/	선택 가능한 응용 소프트웨어 패키지
/proc/	커널과 프로세스 상태를 문서화한 가상 파일 시스템
/root/	루트 사용자의 홈 디렉토리
/sbin/	필수 시스템 바이너리
/srv/	시스템에서 제공되는 사이트 특정 데이터.
/tmp/	임시 파일

리눅스 구조

- **리눅스 디렉터리 구조**

디렉토리	내용
/usr/	읽기 전용 사용자 데이터가 있는 보조 계층 구조.
/usr/bin/	모든 사용자의(단일 사용자 모드에서 필요하지 않은) 비 필수 명령어 바이너리
/usr/include/	표준 include 파일
/usr/lib/	/usr/bin/과 /usr/sbin/에 있는 바이너리를 위한 라이브러리.
/usr/sbin/	비 필수 시스템 바이너리
/usr/share/	아키텍처에 독립적인(공유) 데이터
/usr/src/	소스 코드
/usr/X11R6/	X 윈도 시스템
/usr/local/	로컬 데이터의 3차 계층

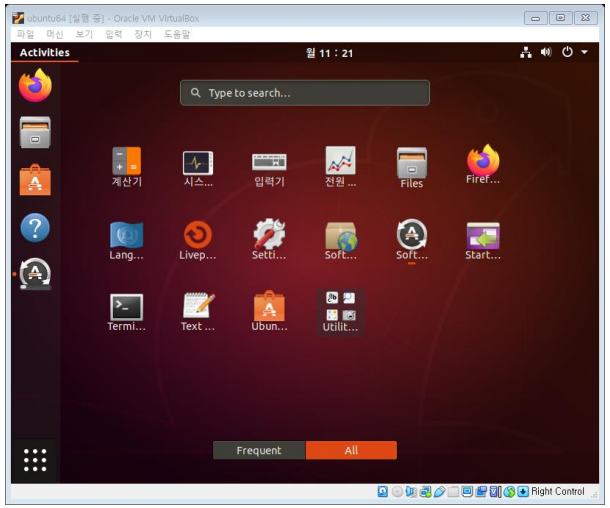
리눅스 구조

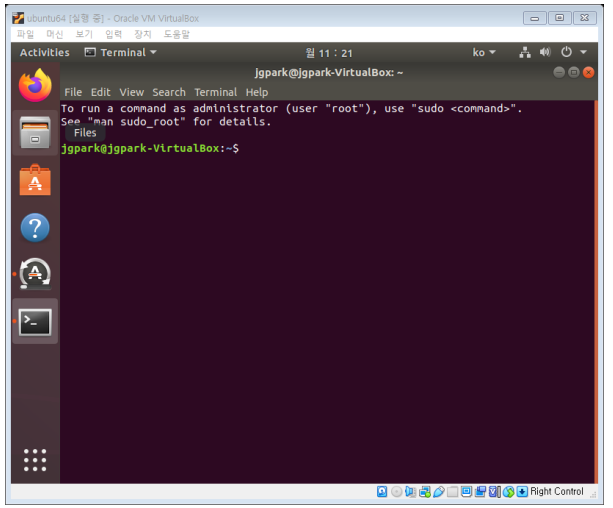
- **리눅스 디렉터리 구조**

디렉토리	내용
/var/	변하기 쉬운 파일
/var/cache/	애플리케이션 캐시 데이터. 이런 데이터는 시간이 걸리는 입출력이나 계산의 결과로 로컬에서 발생
/var/lib/	상태 정보. 프로그램의 실행 중에 수정되는 영구적인 데이터
/var/lock/	잠금 파일. 현재 사용중인 자원을 추적하는 파일
/var/log/	로그 파일
/var/mail/	사용자의 사서함
/var/run/	마지막 부트 때부터 작동하는 시스템에 대한 정보
/var/spool/	처리를 기다리는 작업 스풀
/var/spool/mail/	예전에 사용했으나 현재에는 사용되지 않는 사용자 사서함 위치
/var/tmp/	재부팅 해도 보존되는 임시 파일.

CLI

- **CLI(Command Line Interface, 명령줄 인터페이스)**
  - 터미널을 통해 사용자와 컴퓨터가 상호작용하는 방식
  - 명령은 키보드 등을 통해 문자열의 형태로 입력
  - 결과는 문자열 등의 형태로 출력
- **GUI(Graphical User Interface, 그래픽 유저 인터페이스)**
  - 아이콘이나 메뉴 버튼 등으로 명령어를 대신하고 결과 또한 문자열 뿐만 아니라 이미지의 형태로 출력되는 사용자 인터페이스





리눅스 기본 명령어

- **디렉터리 관련**

명령어	내용	옵션
pwd	현재 위치 출력	
cd	디렉토리 이동	
ls	현재 디렉토리 내 파일 출력	-l, -a
mkdir	새 디렉토리 생성	
rmdir	디렉토리 삭제	-P

리눅스 기본 명령어

- **파일 관련**

명령어	내용	옵션
touch	파일의 최근 업데이트 일자 변경	
cp	파일 복사	
mv	파일 이동	
rm	파일 삭제	-f, -I, -r, -d
file	파일 타입 확인	
find	조건에 맞는 특정 파일 찾기	

리눅스 기본 명령어

- **출력 관련**

명령어	내용	옵션
echo	명령어 다음 내용을 출력	
cat	파일 합치기	
more	출력 결과를 단위 별로 끊어서 출력	
grep	파일에서 해당 내용을 찾아서 출력	-w
>, >>	리다이렉션, 덮어쓰기 / 추가	
man	명령어 매뉴얼 출력	
clear	콘솔 화면 정리	

리눅스 기본 명령어

- **네트워크 관련**

명령어	내용	옵션
ping	목적지에 핑을 보내고 결과 확인	
ifconfig	네트워크 인터페이스 설정 확인	
netstat	네트워크 상태 확인	
traceroute	목적지까지 경로 확인	
route	라우팅 테이블 상태 확인	

리눅스 기본 명령어

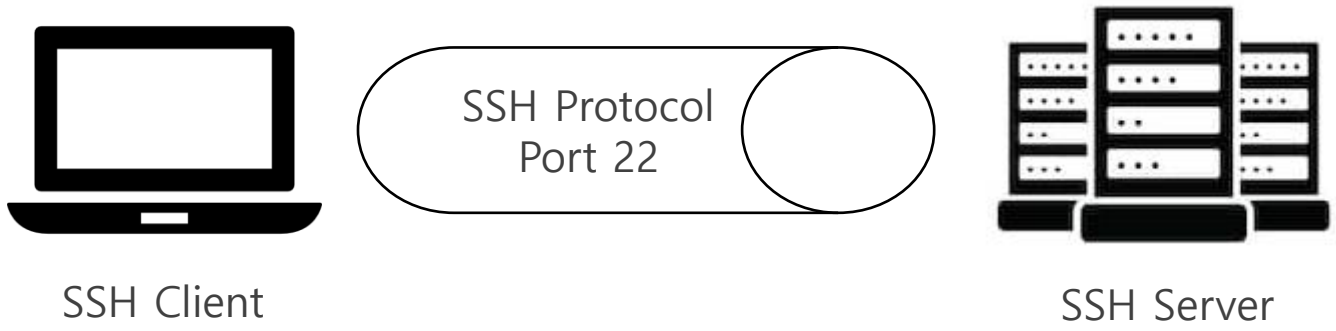
- **그 외 기본 명령어**

명령어	내용	옵션
date	현재 시각 출력	
ps	현재 실행되고 있는 프로세스 목록 출력	-a
kill	특정 프로세스 종료	
tar, bzip2, gzip	압축 관련 명령어	
chmod	파일 또는 디렉토리 권한 수정, chown, chgrp	
crontab	반복적인 작업 수행 명령어	
apt	데비안 계열에서 패키지 관리 명령어	

3. VSCode(SSH) 개발환경 준비

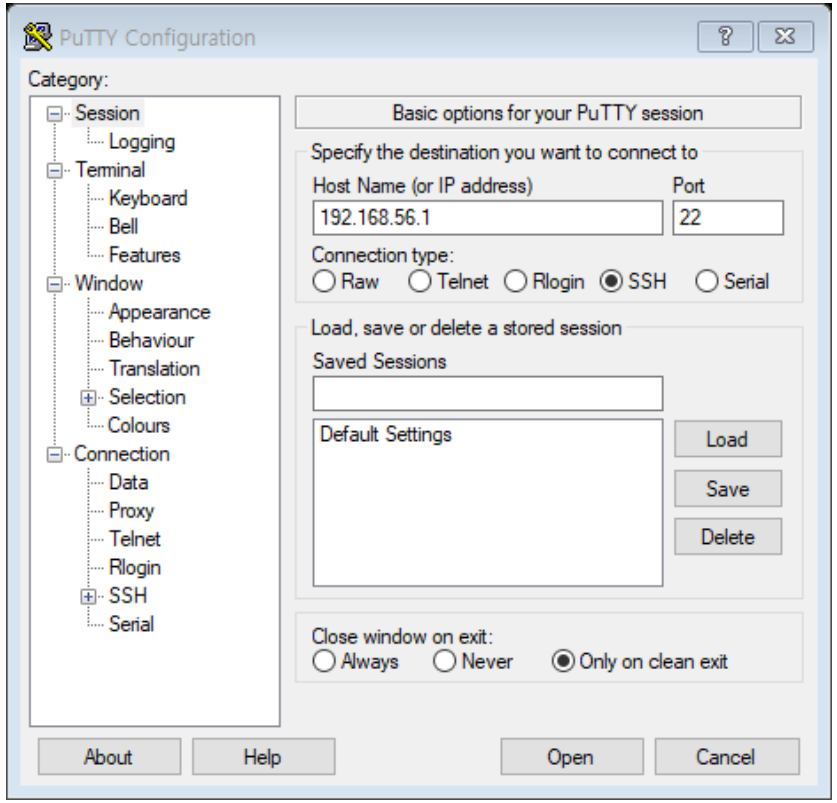
SSH

- **시큐어 셸(SSH, Secure Shell)**
  - 시큐어 셸은 네트워크로 다른 시스템에 원격으로로그인하여 명령을 내릴 수 있는 프로그램이나 프로토콜
  - 주요기능
    - 보안 데이터 전송
    - 원격 제어



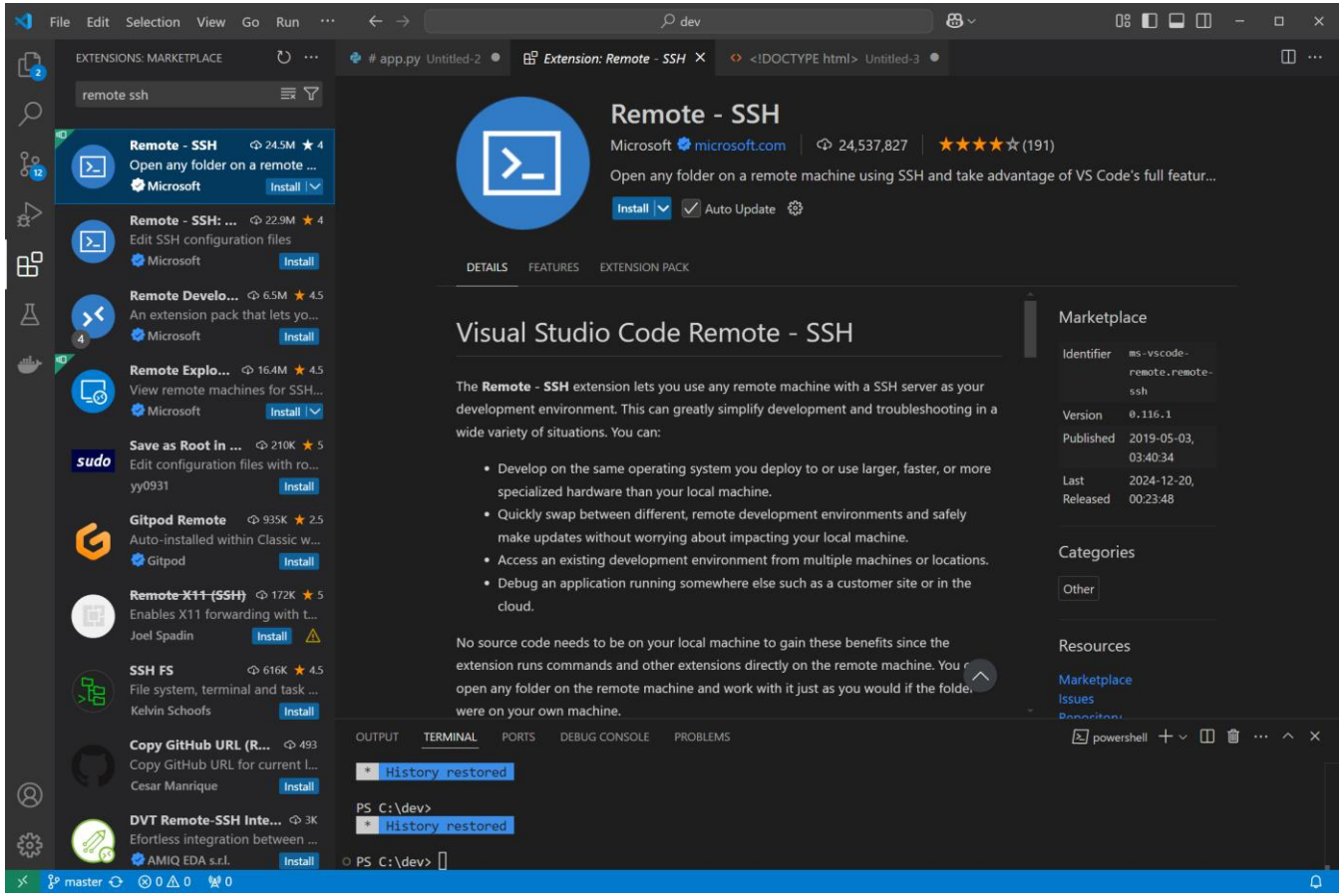
PuTTY

- **PuTTY**
  - **SSH 프로토콜로 서버에 접속할 수 있는 SSH 클라이언트 프로그램**



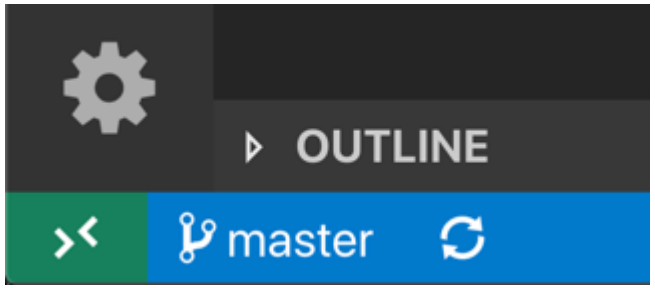
VSCode Remote SSH

- VSCode에서 SSH 연결을 통해 서버에 접속하여 바로 개발



VSCode Remote SSH

1. VSCode 좌측 하단의 녹색 버튼을 눌러 “Connect to Host” 실행
2. + Add New SSH Host 실행
3. 그 이후 뜨는 명령어 창에 아래와 같이 저장
   - ssh *your_account*@116.124.191.174:10022
4. 설정을 다 했으면 다시 “Connect to Host”를 실행하여 방금 추가한 서버에 접속
5. 암호까지 입력하고 나면 서버에 접속하여 사용 가능



4. Python Flask 웹서버 기초

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

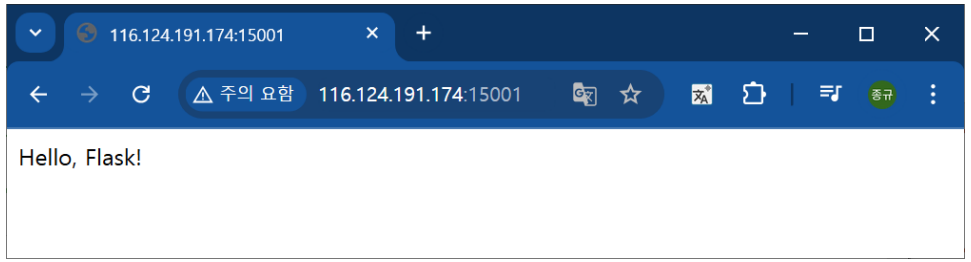
- 프로젝트 디렉터리 생성
  - $ mkdir ex1
- 파이썬 가상환경 설정
  - 가상환경을 사용하여 프로젝트간 종속성 충돌을 방지
  - 가상환경 생성
    - $ python3 –m venv venv
  - 가상환경 활성화
    - $ source venv/bin/activate
  - 가상환경 비활성화
    - $ deactivate
- 가상환경에 플라스크 설치
  - 가상환경 활성화 상태에서
    - (venv) $ pip install flask

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



5. HTML과 CSS

플라스크 프로젝트 기본 구조

- **플라스크 프로젝트 기본 구조**

```
flaskProject/
 ├──── static/          # CSS, JavaScript, 이미지 파일
 ├──── templates/       # HTML 템플릿
 ├──── venv/            # 가상 환경 디렉터리
 └──── app.py           # Flask 애플리케이션 코드
```

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**

👋 나를 소개합니다

안녕하세요! **Flask**와 *HTML*을 함께 배우고 있습니다.

줄바꿈은 이렇게 사용해요.
한 줄 더!

🎯 오늘의 일정

- HTML 실습
- CSS 적용
- Flask와 연결

📅 주간 시간표

요일	활동
월요일	파이썬 복습
화요일	Flask 실습

📷 나의 이미지

ex2-3
form 데이터 전송

- **ex2-3 디렉터리 구조**

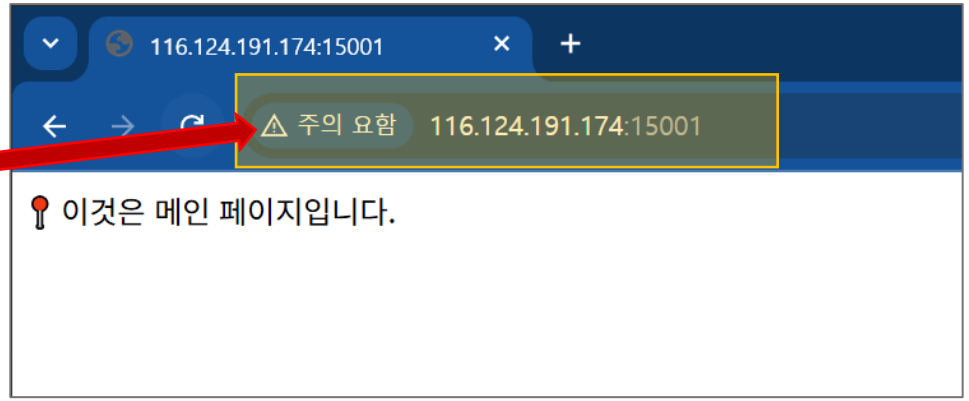
```
ex2-3/
 ├──── venv/          # 파이썬 가상환경
 ├──── app.py         # Flask 애플리케이션 실행 파일
 ├──── static/        # 정적 파일(CSS, JS, 이미지 등)
 │     └──── style.css      # 외부 CSS 파일
 └──── templates/           # HTML 템플릿 디렉터리
       ├──── form.html      # 입력 폼 페이지
       └──── result.html    # 입력값 출력 결과 페이지
```

6. 라우팅(Routing)

ex3 – 라우팅(1/3)

- 라우팅
  - 라우팅은 URL 경로와 기능을 연결하는 과정

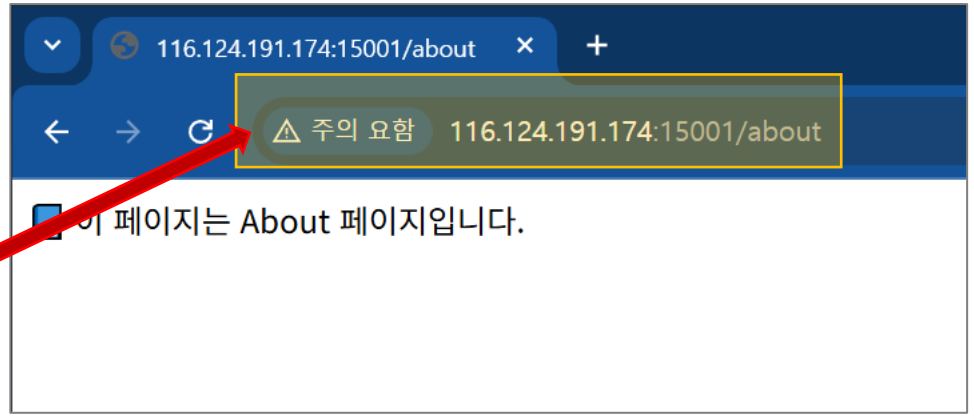
```python
from flask import Flask, render_template

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

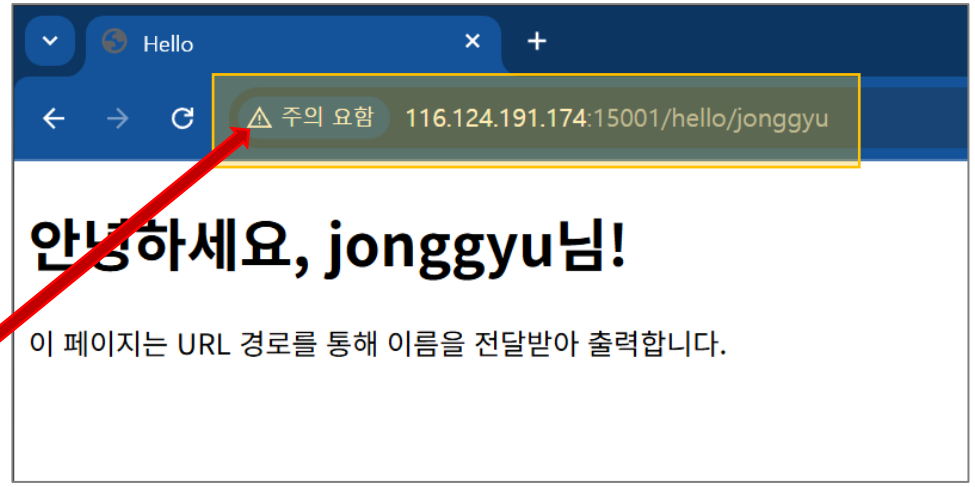
```python
from flask import Flask, render_template

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)
```

7. REST API

HTTP 메서드

- **HTTP 메서드**
  - **클라이언트가 서버로 보내는 HTTP 요청할 때, 요청의 성격이 무엇인지 알리기 위해 HTTP 메서드를 같이 보냄**
  - **주요 메서드로는 GET, POST 등이 있음**
- **HTTP 응답코드**
  - **서버는 클라이언트의 서버 요청을 수행한 후 그 결과를 응답코드와 함께 보냄**
  - **대표적인 응답코드는 200(성공), 400(잘못된 요청), 404(페이지 없음), 500(내부 서버 오류) 등이 있음**

메서드	설명
GET	URL(URI) 형식으로 서버에 요청
POST	HTTP의 Body에 내용을 담아서 새로운 정보 생성을 서버에 요청
HEAD	GET과 비슷하며 실제 문서가 아닌 문서 정보(Header)만 요청
OPTIONS	서버가 지원하는 HTTP 메서드에 대한 질의
PUT	POST와 비슷하나 새로운 정보가 아닌 주로 기존 정보를 업데이트 요청
DELETE	정보의 제거 요청
TRACE	요청 리소스가 수신되는 경로를 보임
CONNECT	프락시 서버 같이 중간 서버 경유 요청

HTTP 메서드

- **HTTP 응답코드(1/2)**

종류	내용
100번대	(정보) 요청을 받았으며 프로세스를 계속함
200번대	(성공) 요청을 받았고 수행하여 완료함
300번대	(리다이렉션) 요청 완료를 위해 추가적으로 작업이 필요함
400번대	(클라이언트 오류) 요청이 잘못 되거나 처리 불가
500번대	(서버 오류) 서버에 문제가 생겨서 처리 불가

HTTP 메서드

- **HTTP 응답코드(2/2)**

응답 코드	의미	설명
200	OK	요청이 처리됨
201	Created	서버가 새 리소스를 작성함
204	No Content	요청은 성공했으나 컨텐츠가 없음
301	Moved Permanently	요청한 페이지는 영구히 이동하였음
302	Moved Temporarily	요청한 페이지는 임시로 이동하였음
400	Bad Request	서버가 요청을 인식 못 함(잘못된 요청)
401	Unauthorized	인증이 필요한 요청
403	Forbidden	서버가 요청을 금지함
404	Not Found	서버가 요청한 페이지를 찾을 수 없음
500	Internal Server Error	서버에 오류가 발생하여 요청 수행 불가
503	Service Unavailable	서버 과부하 등의 이유로 서버를 이용할 수 없음

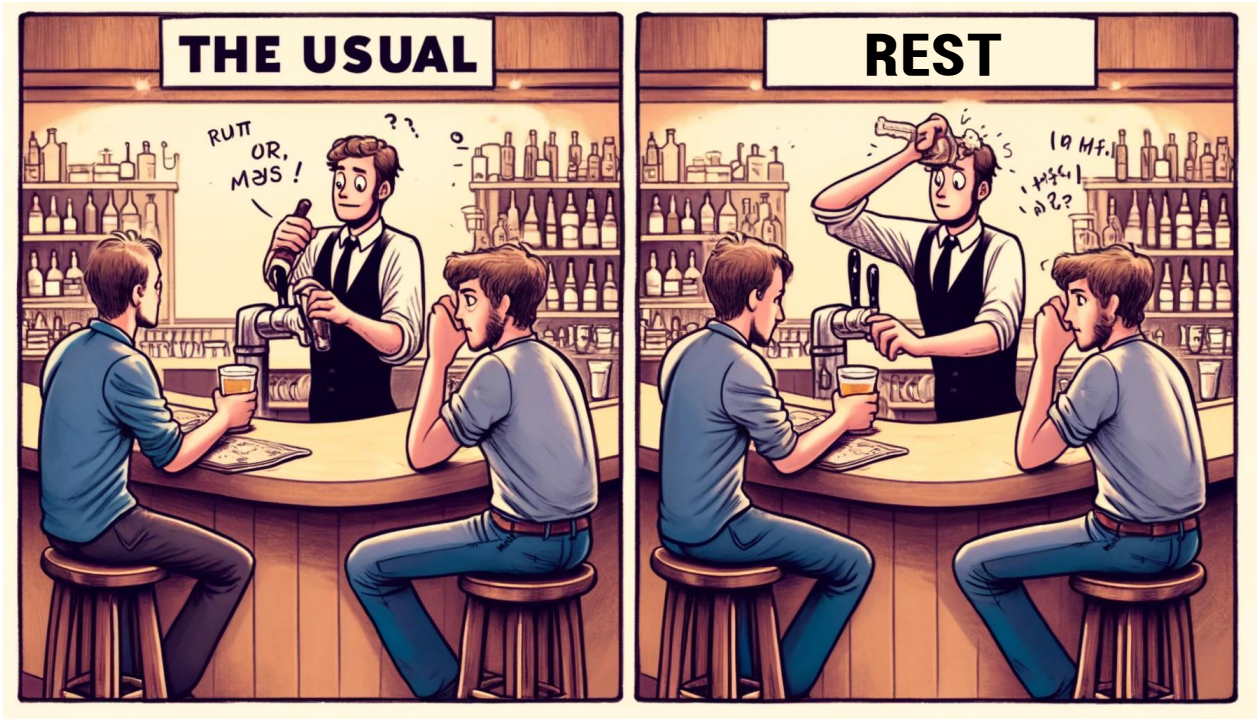
REST

- **REST(Representational State Transfer)**
  - **REST는 웹을 위한 네트워크 아키텍처 스타일로 아래와 같은 특성을 가짐**
  - **RESTful 하다는 것은 REST 아키텍처 원칙을 준수한다는 것**

특성	내용
Client-Server Architecture	클라이언트와 서버는 서로 독립적으로 작동해야 하며, 이를 통해 각각의 부분을 독립적으로 발전시킬 수 있음
Stateless	각 요청은 독립적이어야 하며, 서버는 클라이언트의 상태 정보를 저장하지 않아야 함. 모든 필요한 정보는 요청과 함께 전송되어야 함
Cacheable	클라이언트는 응답을 캐시할 수 있어야 하며, 서버 응답은 캐싱 가능 여부를 명시해야 함
Uniform Interface	인터페이스의 통일성을 통해 시스템의 각 부분을 독립적으로 개선할 수 있음. 이를 위해 리소스 식별, 자기 서술적 메시지, 하이퍼미디어 등의 원칙을 따름
Layered System	클라이언트는 서버가 직접적으로 연결되어 있는지, 중간 서버를 통해 연결되어 있는지 알 수 없어야 함
Code on Demand (Optional)	서버가 클라이언트에 실행 가능한 코드를 전송할 수 있어야 함(옵션)

REST (Stateless)

- **Stateless**
  - 클라이언트와 서버 간의 각 통신은 완전히 독립적이어야 하며, 서버는 클라이언트의 상태나 세션 정보를 저장하면 안 됨
  - 대신, 각 요청은 필요한 모든 정보를 포함해야 하고, 서버는 그 요청만을 처리하여 응답해야 함
  - 요청이 서로 의존적이지 않기 때문에, 각 요청은 이전의 다른 요청들로부터 독립적



REST (Stateless)

- State를 저장 안 함
  - 독립성
    - 각 요청이 필요한 모든 정보를 포함하므로, 서버는 이전 요청의 상태를 기억할 필요가 없어서 서버 설계를 단순화하고, 다른 서버로 요청을 자유롭게 전달할 수 있게 함
  - 확장성
    - 서버가 상태 정보를 유지하지 않기 때문에, 어떤 서버에 요청이 배치되더라도 모든 서버가 동일하게 요청을 처리할 수 있음
    - 이는 서버 클러스터의 확장성을 증가시키며, 서비스의 가용성과 장애 허용성을 향상시킴
  - 성능
    - 각 요청이 독립적으로 처리되므로, 서버는 상태를 관리하기 위한 추가적인 리소스를 사용할 필요가 없어 서버의 처리 성능을 최적화함
- State를 저장함
  - 복잡성 증가
    - 서버가 클라이언트의 상태를 추적하려면 추가적인 로직과 저장 공간이 필요하여, 서버의 복잡성을 증가시키고, 오류 발생 가능성을 높임
  - 확장성 제한
    - 서버가 상태 정보를 저장하면, 특정 클라이언트의 요청을 처리할 수 있는 서버가 제한될 수 있어 로드 밸런싱을 어렵게 만들고, 서버 확장을 복잡하게 할 수 있음
  - 리소스 사용 증가
    - 상태 정보를 저장하고 관리하기 위해 추가 서버 리소스가 필요하므로 전체 시스템의 성능에 부담을 줌

세션과 쿠키

- 세션(Session)
  - 세션은 서버 측에서 사용자 데이터를 저장
  - 세션 ID는 일반적으로 쿠키를 통해 클라이언트에 저장되며, 이 ID를 사용하여 서버는 세션 데이터에 접근하고 사용자를 식별
  - 세션은 주로 로그인 상태 유지에 사용
- 쿠키(Cookies)
  - 클라이언트 측에서 작은 데이터 조각을 저장
  - 이 데이터는 웹 브라우저에 저장되며, 사용자가 웹사이트를 다시 방문할 때마다 웹 서버로 전송
  - 쿠키는 사용자의 선호, 세션 트래킹, 사용자 식별 정보 등을 저장
- 그럼 세션과 쿠키를 쓰면 RESTful 하지 않은가...?
  - 그렇다
  - 원칙적으로는 세션과 쿠키는 REST 아키텍처를 위반하는 것이지만, 사용자 편의를 위해 실제 서비스에서는 세션과 쿠키를 사용하고 있음
  - 따라서 세션과 쿠키를 쓰더라도 RESTful 하다고 표현하는 경우가 많음

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