

Building Serverless Applications with Python

Bok, JongSoon
javaexpert@nate.com
<https://github.com/swacademy/fss>

Introduction

- Name
- Address to live
- Career
- Programming, networking, database experience
- Product experience
- Expectations for the course

Prerequisites

- Basic knowledge of Computer Architecture
- Hardware & Software
- Basic Computer Data Structure
- Familiarity with basic Operating Systems functions
- Understanding of the basics of structured programming
- Shell Scripting experience
- Experience programming in C, C++, Java, **Python**
- **AWS Cloud Services** experience

Contents

- AWS Ramp-Up Guide for Serverless
- Learning Brief Python Language
- Understanding of Cloud Computing
- Building Serverless Applications with Python in AWS
- Terms Relevant to Cloud Computing

Reference Sites for Learning Python

- [A Byte of Python](#)
- [Python Tutorials](#)
- [파이썬을 배우는 최고의 방법](#)
- [Python by Luis Solis | ZEEF](#)
- [Python by Alan Richmond | ZEEF](#)
- [Introduction to programming with Python](#)
- [파이썬으로 배우는 알고리즘 트레이닝](#)
- [점프 투 파이썬](#)

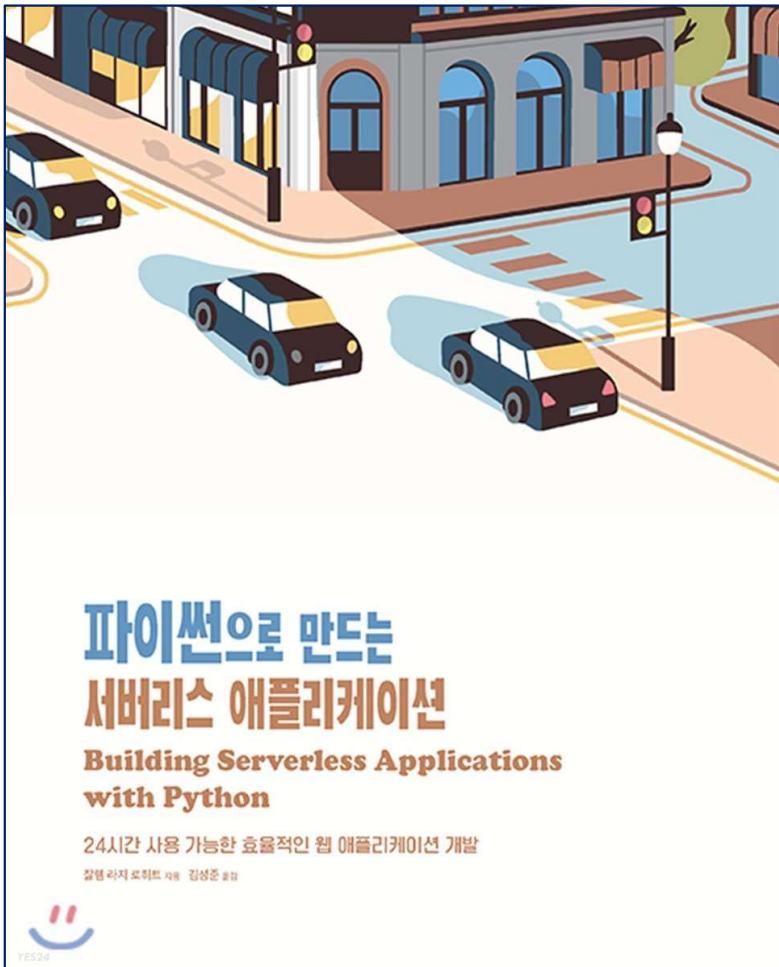
Reference Sites for Learning Python (Cont.)

- [Python 살펴보기](#)
- [Tutorial Campus Python](#)
- [Learning Café Python](#)
- [장고걸스 튜토리얼](#)
- [Python 개발자를 위한 gevent](#)
- [Kivy Planet](#)
- [Non-Programmer's Tutorial for Python 2.6](#)
- [Python Codecademy](#)
- [Python Cheatsheet](#)

Reference Sites for Learning Python (Cont.)

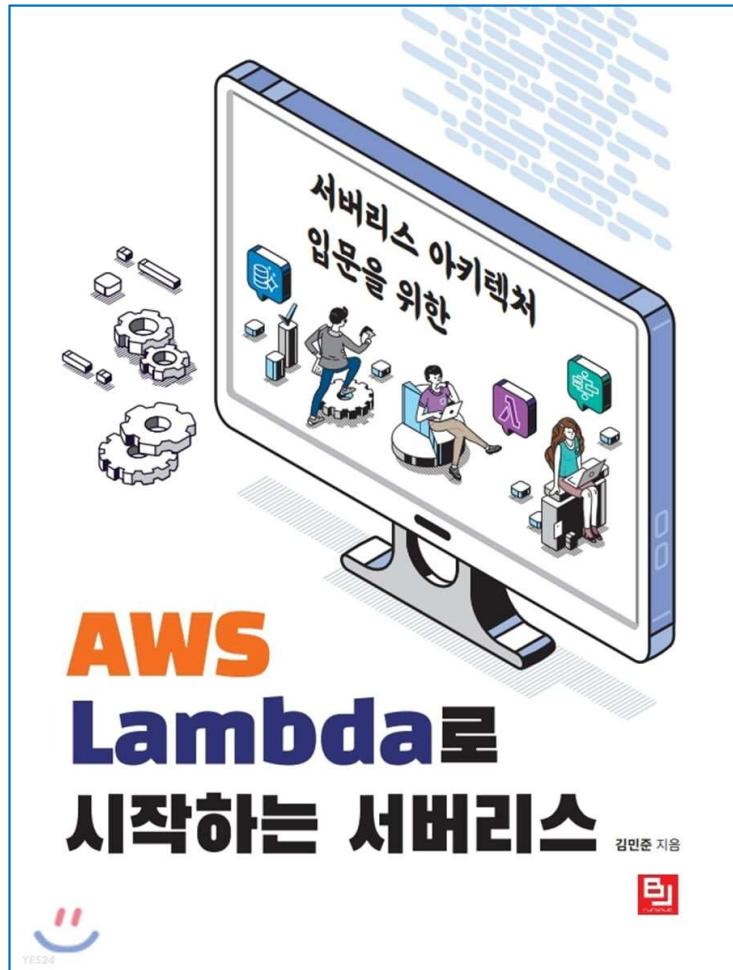
- [Python Imaging Library \(PIL\)](#)
- [될성부른떡잎의 음주 프로그래밍](#)
- [Introduction to Computer Science and Programming Using Python](#)
- [MIT 6.00 컴퓨터 공학과 프로그래밍\(Python\) 오픈 코스](#)
- [Collection Of 51 Free eBooks On Python Programming](#)
- [Tutorials, Python Course](#)
- [Matplotlib tutorial](#)
- [lapee의 computer world](#)
- [Ulismoon' blog](#)
- [Python Programming Tutorials](#)

References



- 파이썬으로 만드는 서비스 애플리케이션
- 잘렘 라자 로히트 저/김성준 역
- 에이콘출판사
- 2018년 10월 29일
- <http://www.yes24.com/Product/Go/ods/65572031>

References



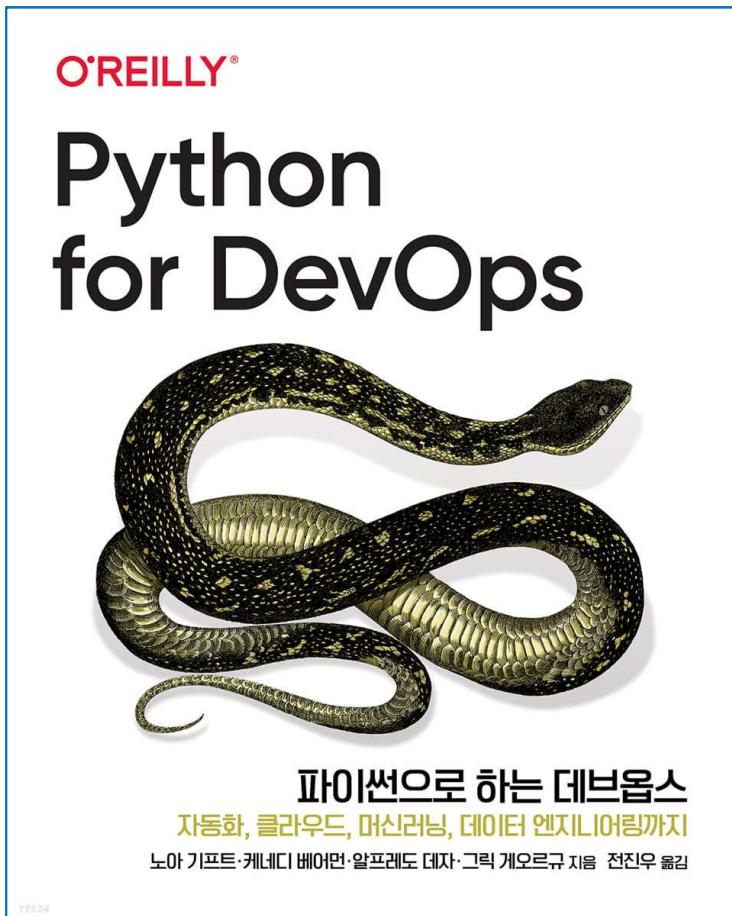
- AWS Lambda로 시작하는 서비스
- 김민주 저
- 비제이퍼블릭
- 2020년 10월 30일
- <http://www.yes24.com/Product/Goods/93982220>

References



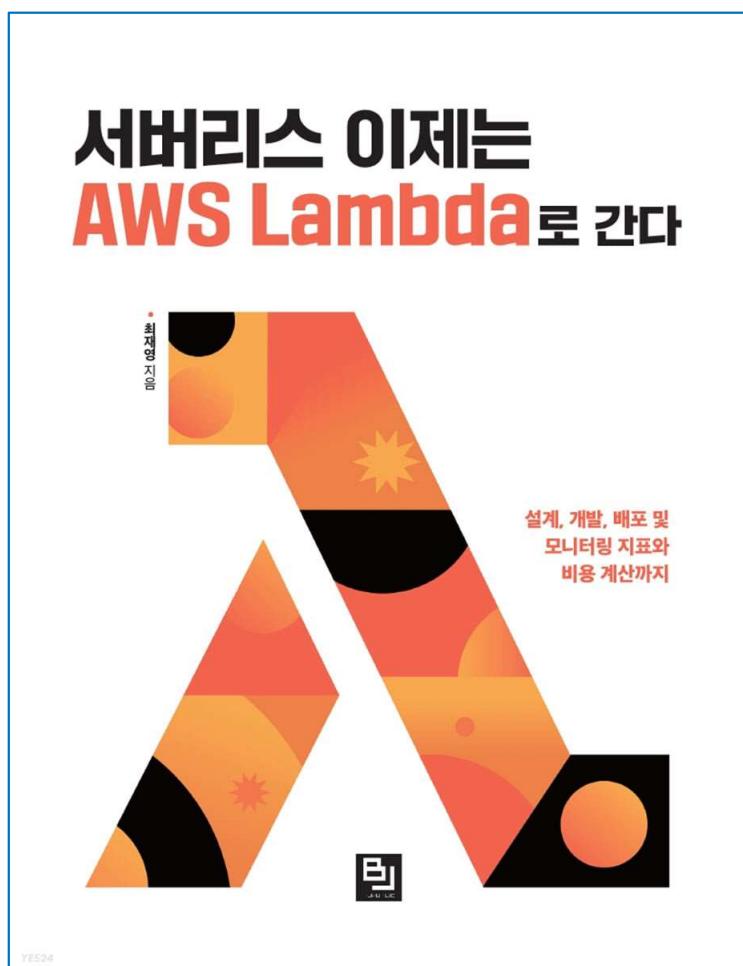
- 서버리스 웹 애플리케이션 구축
- 디에고 자농 저/김경호 외 4명
- 에이콘출판사
- 2018년 2월 23일
- [http://www.yes24.com/Product/Goo
ds/58216884](http://www.yes24.com/Product/Goods/58216884)

References



- 파이썬으로 하는 데브옵스
- 노아 기프트 외 2명 / 전진우 역
- 에이콘출판사
- 2021년 6월 30일
- <http://www.yes24.com/Product/Goods/102181721>

References



- 서비스 이제는 AWS Lambda로 간다
- 최재영 저
- 비제이퍼블릭
- 2022년 11월 21일
- <http://www.yes24.com/Product/Goods/115311643>

References

- Introduction to Serverless Development (Korean)
 - <https://www.aws.training/Details/eLearning?id=64901>
- Architecting Serverless Solutions (Korean)
 - <https://www.aws.training/Details/eLearning?id=70986>

Additional References

■ AWS 온라인 교육 일정

- https://pages.awscloud.com/traincert_alwayslearning_virtualevents_kr.html

■ AWS 교육 및 자격증 개요

- <https://aws.amazon.com/ko/training/>

■ AWS 디지털 교육

- <https://explore.skillbuilder.aws/learn>

■ AWS 교육 및 자격증 소개 자료

- https://pages.awscloud.com/rs/112-TZM-766/images/AWSTrainingAndCertificationOverviewFlyer_KR.pdf

Python in AWS

■ <https://aws.amazon.com/ko/what-is/python/>

The screenshot shows the AWS website's 'What is Python?' page. At the top, there's a navigation bar with links for 제품, 솔루션, 요금, 설명서, 학습하기, 파트너 네트워크, AWS Marketplace, 고객 지원, 이벤트, 자세히 알아보기, and a search icon. On the right, there are links for 문의하기, 지원, 고객지원, 한국어, 내 계정, and a '콘솔에 로그인' button.

The main heading is 'Python이란 무엇인가요?'. Below it is a button labeled '개발자용 AWS에 대해 자세히 알아보기'. The page features four cards:

- 무료 기계 학습 제품 및 서비스 살펴보기**: Shows a gear icon and text about exploring AI and machine learning services.
- 기계 학습 서비스 확인하기**: Shows a brain icon and text about checking out AI services.
- 기계 학습 교육 찾아보기**: Shows a lightbulb icon and text about finding AI education resources.
- 기계 학습 블로그 읽기**: Shows a right-pointing arrow icon and text about reading AI blog posts.

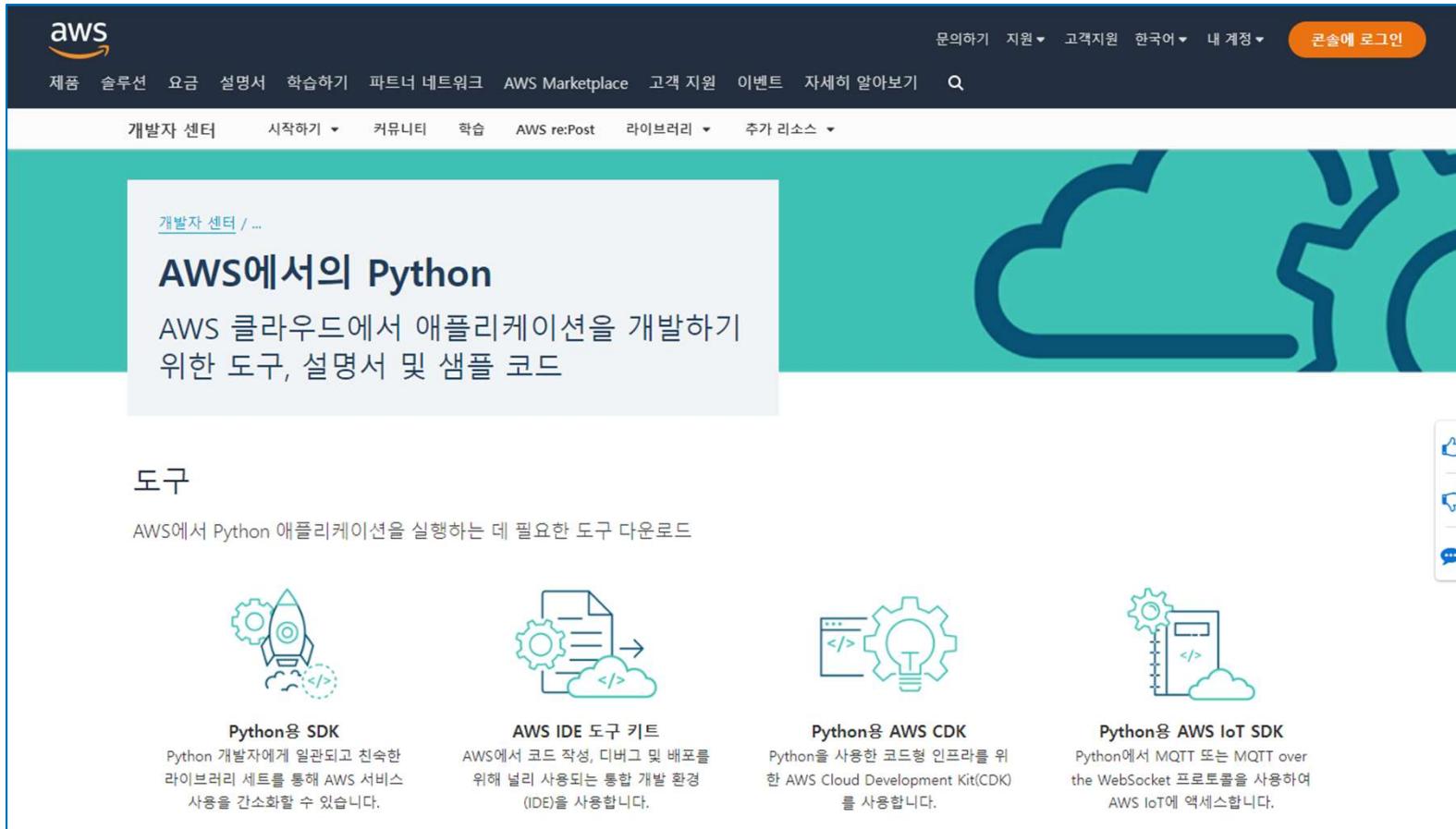
At the bottom, there are two sections:

- Python이란 무엇인가요?**: A question and a detailed answer about Python's use in various fields like web applications, software development, data science, and machine learning.
- Python은 어떻게 사용되나요?**: A question and a detailed answer about Python's versatility across different platforms and its role in system integration.

At the very bottom, there's a footer with the text 'AWS는 아마존의 토지로 모여있거나'.

Python in AWS (Cont.)

■ <https://aws.amazon.com/ko/developer/language/python/>



The screenshot shows the AWS Developer Center page for Python. The top navigation bar includes links for 제품, 슬루션, 요금, 설명서, 학습하기, 파트너 네트워크, AWS Marketplace, 고객 지원, 이벤트, 자세히 알아보기, and a search bar. The main header is "AWS에서의 Python" with the subtext "AWS 클라우드에서 애플리케이션을 개발하기 위한 도구, 설명서 및 샘플 코드". The right side features a large blue cloud icon. Below the header, there's a section titled "도구" (Tools) with four items: "Python용 SDK", "AWS IDE 도구 키트", "Python용 AWS CDK", and "Python용 AWS IoT SDK". Each item has a corresponding icon and a brief description.

개발자 센터 / ...

AWS에서의 Python

AWS 클라우드에서 애플리케이션을 개발하기 위한 도구, 설명서 및 샘플 코드

도구

AWS에서 Python 애플리케이션을 실행하는 데 필요한 도구 다운로드

Python용 SDK
Python 개발자에게 일관되고 친숙한 라이브러리 세트를 통해 AWS 서비스 사용을 간소화할 수 있습니다.

AWS IDE 도구 키트
AWS에서 코드 작성, 디버그 및 배포를 위해 널리 사용되는 통합 개발 환경 (IDE)을 사용합니다.

Python용 AWS CDK
Python을 사용한 코드형 인프라를 위한 AWS Cloud Development Kit(CDK)을 사용합니다.

Python용 AWS IoT SDK
Python에서 MQTT 또는 MQTT over the WebSocket 프로토콜을 사용하여 AWS IoT에 액세스합니다.

Lab Environment

- Apple MacBook or Microsoft Windows 10+
- Google Chrome or Mozilla Firebox
- Adobe Acrobat PDF Reader or Foxit PDF Reader
- Microsoft Visual Studio Code for 64-bit
- Postman
- SSH Client Tools : PuTTY, XSHELL, MobaXterm etc.



AWS Ramp-Up Guide for Serverless



AWS Ramp-Up Guide for Serverless

AWS Lambda, API Gateway for Serverless, 다른 서비스와의 통합을 사용하여 서비스 애플리케이션을 설계, 구축하고 이벤트 중심적 애플리케이션을 개발하는 방법을 학습합니다. 또한, 서비스 애플리케이션을 보안, 확장, 관리하는 방법에 대해서도 알아봅니다. 이러한 기술을 개발하는 데 도움이 되는 디지털 과정과 랩에 관심이 있다면 explore.skillbuilder.aws에서 서비스 학습 플랜을 살펴보십시오. 램프업 가이드에 대한 제안 사항이 있으면 rampupguides@amazon.com으로 연락해주십시오.

AWS 클라우드의 기초 학습

학습 리소스	소요 시간	유형
AWS Ramp-Up Guide: Cloud Essentials		램프업 가이드 »

서비스 기초 학습

학습 리소스	소요 시간	유형
Getting into the Serverless Mindset	30분	디지털 교육 »
AWS Lambda Foundations	1시간	디지털 교육 »

서비스 디자인

학습 리소스	소요 시간	유형
Designing Event-Driven Architectures	2시간	디지털 교육 »
Architecting Serverless Applications	2시간	디지털 교육 »
Amazon API Gateway for Serverless Applications	2시간	디지털 교육 »

고급 서비스 컨셉

학습 리소스	소요 시간	유형
Scaling Serverless Architectures	1시간 30분	디지털 교육 »
Security and Observability for Serverless Applications	1시간 30분	디지털 교육 »
Deploying Serverless Applications	1시간 30분	디지털 교육 »
Developing Serverless Solutions on AWS	3일	디지털 교육 »

https://d1.awsstatic.com/ko_KR/training-and-certification/ramp-up_guides/Ramp-Up_Guide_Serverless.pdf

AWS Ramp-Up Guide for Serverless (Cont.)

추가 리소스 살펴보기	유형
학습 리소스	
Lambda Getting Started	설명서 »
Getting Started with Serverless	워크샵 »
Getting Started with AWS Lambda and Serverless Computing – AWS Online Tech Talks	동영상 »
The Complete AWS SAM	워크샵 »
Serverless tutorials / hands-on activities	워크샵 »
Building microservices with AWS Lambda Video	동영상 »
Serverless architectural patterns and best practices	동영상 »
Best practices for organizing larger serverless applications	블로그 »
Moving to event-driven architectures	동영상 »
Scalable serverless event-driven applications using Amazon SQS & Lambda	동영상 »



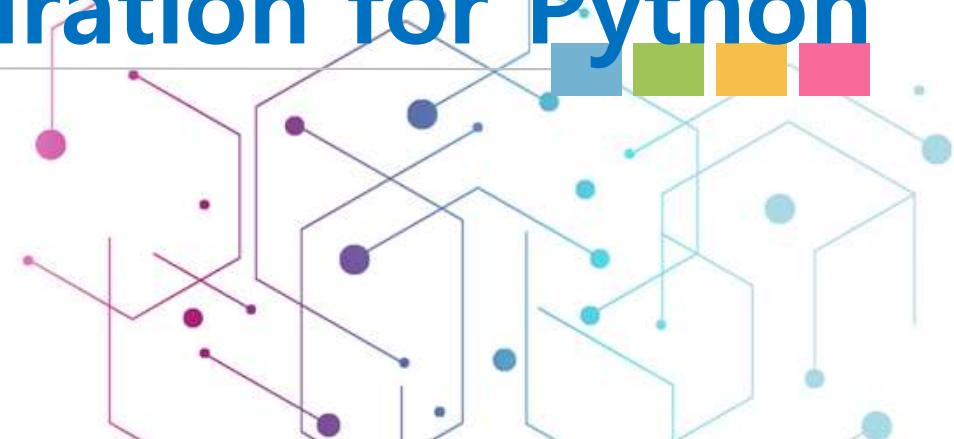
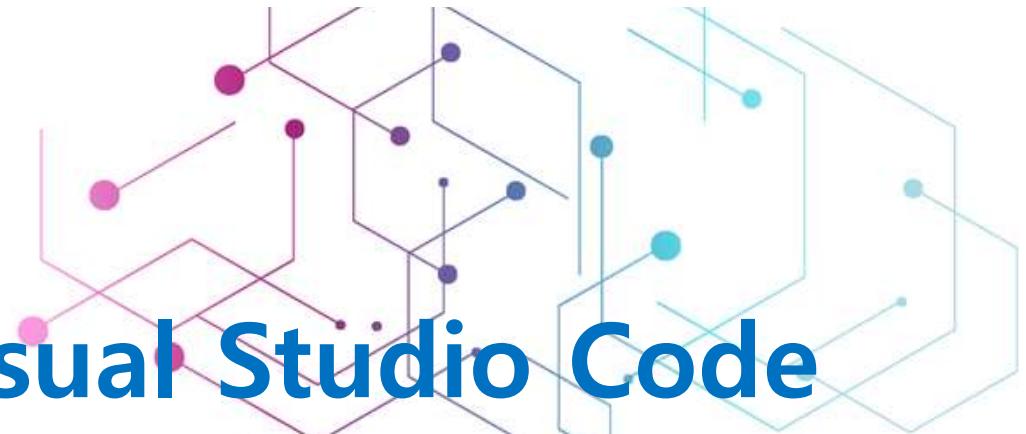
Learning Brief Python Language





Lab. Microsoft Visual Studio Code Installation & Configuration for Python

- ① Visual Studio Code Installation
- ② General Configuration
- ③ Configuration for Python





Lab. Configure Python Programming Environment

-
- ① Creating AWS EC2 Instance
 - ② Remote Access with Visual Studio Code
 - ③ Print 'Hello, World'





Lab. Installation Jupyter Notebook



What is Python?

- In Greek mythology, Python was the serpent, sometimes represented as a dragon, living at the center of the earth, believed by the ancient Greeks to be at Delphi.



Source from [https://en.wikipedia.org/wiki/Python_\(mythology\)](https://en.wikipedia.org/wiki/Python_(mythology))

What is Python? (Cont.)

- Is a general-purpose interpreted, interactive, object-oriented, and high-level programming language.
- Created by Guido van Rossum during 1985- 1990 at the National Research Institute for Mathematics and Computer Science in the Netherlands.
- First released in 1991.
- Is named after a TV Show called '**Monty Python's Flying Circus**' and not after Python-the snake.



Python Popularity

■ PYPL

PYPL Index 10 TOP IDE 10 TOP ODE 10 TOP DB

PYPL PopularitY of Programming Language

 3.4K

Worldwide, Feb 2023 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	27.7 %	-0.7 %
2		Java	16.79 %	-1.3 %
3		JavaScript	9.65 %	+0.6 %
4	↑	C#	6.97 %	-0.5 %
5	↓	C/C++	6.87 %	-0.6 %
6		PHP	5.23 %	-0.8 %
7		R	4.11 %	-0.1 %
8	↑↑	TypeScript	2.83 %	+0.8 %
9		Swift	2.27 %	+0.3 %
10	↓↓	Objective-C	2.25 %	-0.1 %
11	↑↑	Go	1.95 %	+0.7 %
12	↑↑	Rust	1.91 %	+0.9 %
13	↓	Kotlin	1.85 %	+0.2 %
14	↓↓↓	Matlab	1.71 %	-0.1 %

The PYPL PopularitY of Programming Language Index is created by analyzing how often language tutorials are searched on Google.

The more a language tutorial is searched, the more popular the language is assumed to be. It is a leading indicator. The raw data comes from Google Trends.

If you believe in collective wisdom, the PYPL Popularity of Programming Language index can help you decide which language to study, or which one to use in a new software project.

<https://pypl.github.io/PYPL.html>

Python Popularity (Cont.)

■ TIOBE Index

Jan 2023	Jan 2022	Change	Programming Language	Ratings	Change
1	1		 Python	16.36%	+2.78%
2	2		 C	16.26%	+3.82%
3	4	▲	 C++	12.91%	+4.62%
4	3	▼	 Java	12.21%	+1.55%
5	5		 C#	5.73%	+0.05%
6	6		 Visual Basic	4.64%	-0.10%
7	7		 JavaScript	2.87%	+0.78%
8	9	▲	 SQL	2.50%	+0.70%
9	8	▼	 Assembly language	1.60%	-0.25%
10	11	▲	 PHP	1.39%	-0.00%
11	10	▼	 Swift	1.20%	-0.21%
12	13	▲	 Go	1.14%	+0.10%
13	12	▼	 R	1.04%	-0.21%
14	15	▲	 Classic Visual Basic	0.98%	+0.01%
15	16	▲	 MATLAB	0.91%	-0.05%

<https://www.tiobe.com/tiobe-index/>

Starting the Python Interpreter

■ Some simple math stuff

- Type **2 + 2** into the shell and press the **Enter key**.
- Computer should respond with the number **4**.
 - : the sum of **2 + 2**

```
File Edit Shell Debug Options Window Help
Python 3.6.2 (default, Jul 17 2017, 23:14:31)
[GCC 5.4.0 20160609] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 2 + 2
4
>>> |
```

Python Syntax Features

■ Indentation

```
def add5(x):
    return x+5

def dotwrite(ast):
    nodename = getNodeName()
    label=symbol.sym_name.get(int(ast[0]),ast[0])
    print '%s [%s=%s]' % (nodename, label),
    if isinstance(ast[1], str):
        if ast[1].strip():
            print '= %s';' % ast[1]
        else:
            print ''
    else:
        print '';
        children = []
        for n, child in enumerate(ast[1:]):
            children.append(dotwrite(child))
        print ', %s -> {' % nodename
        for name in children:
            print '%s' % name,
```

Python Syntax Features (Cont.)

■ Indentation

Python

```
def factorial(x):
    if x == 0:
        return 1
    else:
        return x * factorial(x - 1)
```

C

```
int factorial(int x)
{
    if(x == 0)
    {
        return 1;
    }
    else
    {
        return x * factorial(x - 1);
    }
}
```

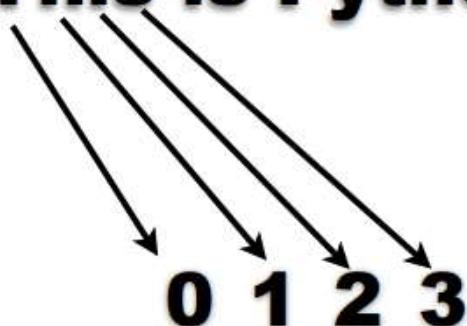
```
int factorial(int x) {
    if(x == 0) {return 1;} else
    {return x * factorial(x - 1); } }
```

Python Syntax Features (Cont.)

■ String Indexing and Slicing

```
>>> t = "This is Python Class, And 1st day"  
>>> t[2]  
'i'
```

This is Python Class, And 1st day



Python Syntax Features (Cont.)

■ String Indexing and Slicing

```
>>> t[2]
'i'
>>> t[6]
's'
>>> t[12]
'o'
>>> t[21]
' '
>>> t[-1]
'y'
```

Python Syntax Features (Cont.)

■ String Indexing and Slicing

```
>>> t[1:3]
'hi'
>>> t[2:10]
'is is Py'
```

Python Syntax Features (Cont.)

■ String Indexing and Slicing

```
>>> t = "2015-03-10 14:21:35"
>>> date = t[:10]
>>> time = t[11:]
>>> print(date)
2015-03-10
>>> print(time)
14:21:35
```

Python Syntax Features (Cont.)

■ String Formatting

```
>>> name = "Python"  
>>> print("This is %s" % name)  
This is Python  
>>> print("This is %d" % 10)  
This is 10
```

Reserved Words

- Cannot use as constants or variables or any other identifier names.
- All the Python keywords contain lowercase letters only.

```
>>>
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
>>>
```

Lines and Indentation

- Python does not use braces(`{ }`) to indicate blocks of code for class and function definitions or flow control.
- Blocks of code are denoted by line indentation, which is rigidly enforced.
- The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount.

```
if True :  
    print ("True")  
else :  
    print ("False")
```

Multi-Line Statements

- Statements in Python typically end with a new line.
- However, allows the use of the line continuation character (\) to denote that the line should continue.

```
>>> total = item_one + \
           item_two + \
           item_three
```

Quotation in Python

- Python accepts single ('), double ("") and triple ('''' or ''''') quotes to denote string literals, as long as the same type of quote starts and ends the string.
- The triple quotes are used to span the string across multiple lines.

```
>>> word = 'word'  
>>> sentence = "This is a sentence."  
>>> paragraph = """This is a paragraph. It is  
made up of multiple lines and sentences."""  
>>>
```

Comments in Python

- A hash sign (#) that is not inside a string literal is the beginning of a comment.
- All characters after the #, up to the end of the physical line, are part of the comment and the Python interpreter ignores them.

```
>>> # First comment
>>> print ("Hello, Python!") # second comment
Hello, Python!
>>>
```

Comments in Python (Cont.)

- You can type a comment on the same line after a statement or expression.

```
>>> name = "Orange" # This is again comment  
>>>
```

- Python does not have multiple-line commenting feature.

```
>>> # This is a comment.  
>>> # This is a comment, too.  
>>> # This is a comment, too.  
>>> # I said that already.  
>>>
```

Waiting for the User

- The following line of the program displays the prompt and, the statement saying “**Press the enter key to exit**”, and then waits for the user to take action.

```
>>>  
>>> input ("\\n\\nPress the enter key to exit.")
```

Press the enter key to exit.

"

```
>>> |
```

Multiple Statements on a Single Line

- The semicolon (;) allows multiple statements on a single line given that no statement starts a new code block.
- Here is a sample snip using the semicolon.

```
>>>  
>>> import sys; x = 'foo' ; sys.stdout.write(x + '\n')  
foo  
4  
>>> |
```

Multiple Statement Groups as Suites

- Groups of individual statements, which make a single code block are called suites in Python.
- Compound or complex statements, such as **if**, **while**, **def**, and **class** require a header line and a suite.
- Header lines begin the statement (with the keyword) and terminate with a colon (`:`) and are followed by one or more lines which make up the suite.

Standard Input & Output

- Inputs through Keyboard.
- The function reads a line from input, converts it to a string (stripping a trailing newline), and returns that.
- Syntax
 - **input(prompt)**
 - **variable = input(prompt)**

```
>>> input()
>>> name = input("name ? ")
>>> print(name)
>>> money = input("How much? ")
>>> print(money)
>>> print(money + 100)
>>> money = int(input("How much ? "))
>>> print(money)
>>> print(money + 100)
```

Standard Input & Output (Cont.)

- Prints the values to a stream, or to `sys.stdout` by default.

- Syntax

- `print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)`
- `file`: a file-like object (stream); defaults to the current `sys.stdout`.
- `sep`: string inserted between values, default a space.
- `end`: string appended after the last value, default a newline.
- `flush`: whether to forcibly flush the stream.

```
>>> print("Hello")
>>> print(100)
>>> print()
>>> print("Good", "morning!")
>>> print(10, 20, 30)
>>> print(1000, "원")
>>> print("Good" + "morning!")
>>> x = 100
>>> y = 200
>>> sum = x + y
>>> print(x, "과", y, "의 합은", sum, "입니다")
```

Standard Input & Output (Cont.)

- Formatted string literals
- We can format the result with the `format()` method.
- Allows you to format selected parts of a string.
- To control such values, add placeholders (curly brackets `{ }`) in the text, and run the values through the `format()` method
- `%` operator

```
name = 'Pete'  
print('Hello %s' % name)
```

Hello Pete

```
num = 5  
print('I have %d apples.' % num)
```

I have 5 apples.

Standard Input & Output (Cont.)

- **str.format()**
- Python 3 introduced a new way to do string formatting that was later back-ported to Python 2.7.

```
name = 'John'  
age = 20  
print("Hello I'm {}, my age is {}.".format(name, age))
```

Hello I'm John, my age is 20.

```
print("Hello I'm {0}, my age is {1}.".format(name, age))
```

Hello I'm John, my age is 20.

Standard Input & Output (Cont.)

- Formatted String Literals or **f-Strings**
- If you are using Python 3.6+, string **f-Strings** are the recommended way to format strings.

```
name = 'Elizabeth'  
print(f'Hello {name}!')
```

Hello Elizabeth!

```
a = 5  
b = 10  
print(f'Five plus ten is {a + b} and not {2 * (a + b)}.')
```

Five plus ten is 15 and not 30.

Standard Input & Output (Cont.)

■ Multiline **f-Strings**

```
name = 'Robert'  
message = 12  
(  
    f'Hi, {name}. '  
    f'You have {message} unread messages.'  
)
```

'Hi, Robert. You have 12 unread messages.'

Standard Input & Output (Cont.)

- The `=` specifier

```
from datetime import datetime  
  
now = datetime.now().strftime('%b/%d/%Y - %H:%M:%S')  
  
print(f'date and time: {now=}')  
date and time: now='Feb/11/2023 - 01:37:21'
```

Standard Input & Output (Cont.)

■ Adding spaces or characters

```
name = 'Robert'

print(f"{name.upper() = }")
name.upper() = 'ROBERT'

print(f"{name.upper() = :}")
name.upper() = ROBERT

print(f"{name.upper() = : ^20}")
name.upper() = ROBERT
```

Standard Input & Output (Cont.)

■ Formatting Digits

- Adding thousands separator

```
a = 10_000_000  
print(f'{a:,}')
```

10,000,000

- Rounding

```
a = 3.1415926  
print(f'{a:.2f}')
```

3.14

- Showing as Percentage

```
a = 0.816562  
print(f'{a:.2%}')
```

81.66%

Standard Input & Output

■ Number formatting table

Number	Format	Output	description
3.1415926	{:.2f}	3.14	Format float 2 decimal places
3.1415926	{:+.2f}	+3.14	Format float 2 decimal places with sign
-1	{:+.2f}	-1.00	Format float 2 decimal places with sign
2.71828	{:.0f}	3	Format float with no decimal places
4	{:0>2d}	04	Pad number with zeros (left padding, width 2)
4	{:x<4d}	4xxx	Pad number with x's (right padding, width 4)
10	{:x<4d}	10xx	Pad number with x's (right padding, width 4)
1000000	{:,}	1,000,000	Number format with comma separator
0.35	{:.2%}	35.00%	Format percentage
1000000000	{:.2e}	1.00e+09	Exponent notation
11	{:11d}	11	Right-aligned (default, width 10)
11	{:<11d}	11	Left-aligned (width 10)
11	{:^11d}	11	Center aligned (width 10)

Standard Input & Output (Cont.)

■ Alignment

```
>>> "{0:<10}".format("hi")
'hi      '
>>> "{0:>10}".format("hi")
'      hi'
>>> "{0:^10}".format("hi")
'    hi    '
>>> "{0:=^10}".format("hi")
'====hi===='
>>> "{0:!<10}".format("hi")
'hi ! ! ! ! ! ! !'
```

Exercise. Python Print



■ Question1

- 두 수 50과 100의 사칙연산을 +,-,*,/를 이용해 구하고 연산결과 값을 화면에 출력하시오.

■ Question2

- 첫번째 변수에 'python is'를 저장하고 두번째 변수에 'easy'를 사용자에게 입력 받아 'python is easy'를 화면에 출력하시오.

■ Question3

- 사각형의 면적을 구하려고 한다. 가로와 세로의 길이를 입력받아서 사각형의 넓이를 계산하는 프로그램을 작성하시오.

• 예

- 가로의 길이 : 30
- 세로의 길이 : 20
- 사각형의 넓이 : 600

Variables, Names, and Objects

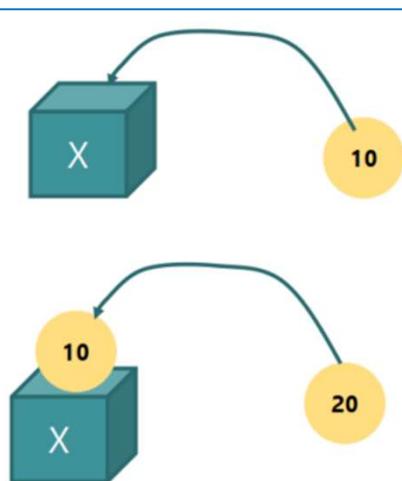
- In Python, everything - *booleans*, *integers*, *floats*, *strings*, even large data structures, functions, and programs - is implemented as an *object*.
- An *object* is like a clear plastic box that contains a piece of data.



Variables, Names, and Objects (Cont.)

- Variables - The values put into variables *can be* changed at any time.
- Constants - The values put into constants *cannot* be changed.

```
>>> x = 10
>>> x = 20
>>> print(x)
>>> x = 100
>>> y = 200
>>> sum = x + y
>>> print(sum)
>>> sum = sum + 100
>>> name = "홍길동"
```



Variables, Names, and Objects (Cont.)

■ Identifier

- Is the name given to variables, classes, methods, etc.

■ Naming Convention

- Identifier cannot be a keyword.
- Identifier is case-sensitive.
- It can have a sequence of letters and digits. However, it must begin with a letter or _. The first letter of an identifier cannot be a digit.
- It's a convention to start an identifier with a letter rather _.
- Whitespaces are not allowed.
- We cannot use special symbols like !, @, #, \$, and so on.

Variables, Names, and Objects (Cont.)

변수 이름	설명
size	가능하다.
cloud9	가능하다. 변수는 영문자, 숫자, _로 이루어진다.
max_size	가능하다. 변수의 중간에 _가 있어도 된다.
_count	가능하다. _가 앞에 붙으면 클래스 내부에서만 사용하는 변수라는 의미도 있다.
6pack	올바르지 않다! 숫자가 앞에 오면 안된다.
mid score	올바르지 않다! 중간에 공백이 있으면 안된다.
class	올바르지 않다! 예약어를 변수의 이름으로 사용할 수 없다.
money#	올바르지 않다! 기호를 변수의 이름으로 사용하면 안 된다.

Variables, Names, and Objects (Cont.)

■ Assigning Values to Variables

- Python variables do not need explicit declaration to reserve memory space.
- The declaration happens automatically when you assign a value to a variable.
- The equal sign (`=`) is used to assign values to variables.
- The operand to the left of the `=` operator is the name of the variable and the operand to the right of the `=` operator is the value stored in the variable.

Variables, Names, and Objects (Cont.)

```
>>> counter = 100 # An integer assignment
>>> miles = 1000.0 # A floating point
>>> name = "John" # a string
>>>
>>> print(counter)
100
>>> print(miles)
1000.0
>>> print(name)
John
>>>
```

Variables, Names, and Objects (Cont.)

■ Multiple Assignment

- Python allows you to assign a single value to several variables simultaneously.

```
>>> a = b = c = 1
```

Variables, Names, and Objects (Cont.)

■ Multiple Assignment

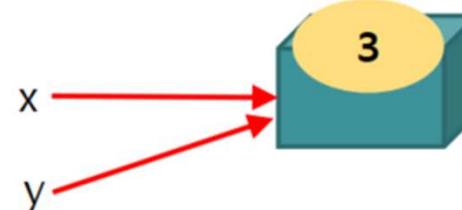
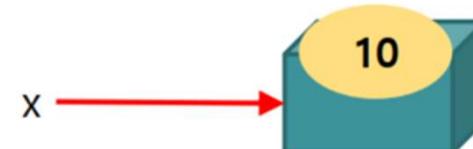
- Here, an integer object is created with the value 1, and all the three variables are assigned to the same memory location.
- You can also assign multiple objects to multiple variables.

```
>>> a, b, c = 1, 2, "John"
>>>
>>> print(a)
1
>>> print(b)
2
>>> print(c)
John
>>>
```

Variables, Names, and Objects (Cont.)

- Everything is Object in Python.
- What is stored in the variable is not the actual value, but the *reference value* of the object.

```
>>> x = 10  
>>> id(x)  
  
>>> x = 3  
>>> y = x  
  
>>> id(x)  
>>> id(y)  
  
>>> y = 10  
>>> id(y)
```



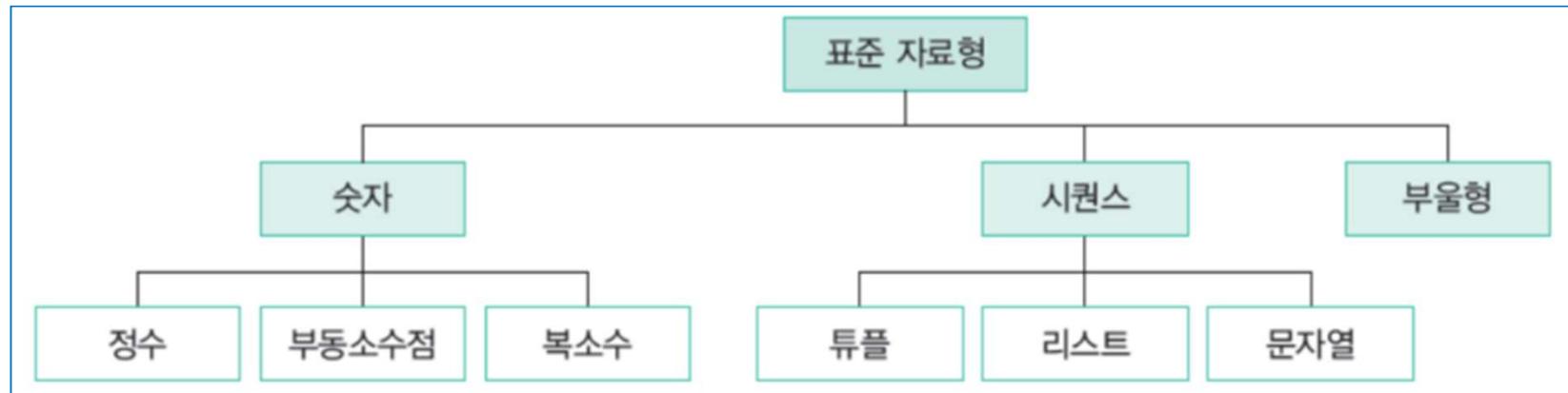
Standard Data Types

- The data stored in memory can be of many types.
- For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters.
- Python has various *standard data type*s that are used to define the operations possible on them and the storage method for each of them.

Standard Data Types (Cont.)

- Python has five *standard data type*s :

- Numbers
- String
- List
- Tuple
- Dictionary



```
>>> type(10)
>>> type(True)
>>> type(12.30)
>>> type("hello")
```

Python Numbers

- Number data types store numeric values.
- Number objects are created when you assign a value to them.

```
>>> var1 = 1
```

```
>>> var2 = 10
```

- You can also delete the reference to a number object by using the **del** statement.

```
>>> del var1[, var2[, var3[...], varN]]]
```

Python Numbers (Cont.)

- Can delete a single object or multiple objects by using the **del** statement.

```
>>> del var
```

```
>>> del var_a, var_b
```

Python Numbers (Cont.)

- Python supports three different numerical types :
 - **int** (signed integers)
 - **float** (floating point real values)
 - **complex** (complex numbers)
- All integers in Python3 are represented as **long** integers.
- Hence, there is no separate number type as long.

Python Numbers (Cont.)

int	float	complex
10	0.0	3.14j
100	15.20	45.j
-786	-21.9	9.322e-36j
080	32.3+e18	.876j
-0490	-90.	-.6545+0j
-0x260	-32.54e100	3e+26j
0x69	70.2-E12	4.53e-7j

A complex number consists of an ordered pair of real floating-point numbers denoted by $x + yj$, where x and y are real numbers and j is the imaginary unit.

Python Numbers (Cont.) – Lab

```
1 print(2 + 3)
2
3 print(2 * 3)
4
5 print(3 / 2)
6
7 print(3 ** 2)
8
9 print(2 + 3 * 4)
10
11 print(2 * 0.1)
12
13 print(0.1 + 0.1)
14
```

Python Numbers (Cont.) – Lab

```
>>> a = 3
>>> b = 123456789
>>> c = 1234567890123456789012345678901234567890
>>> print (a)
3
>>> print (b)
123456789
>>> print (c)
1234567890123456789012345678901234567890
>>> type(c)
<class 'int'>
>>>
>>> f = 22 / 7
>>> f
3.142857142857143
>>> type(f)
<class 'float'>
>>>
```

Python Numbers (Cont.) – Lab

```
>>> hex(0)
'0x0'
>>>
>>> hex(255)
'0xff'
>>>
>>> a = 0xFF
>>> a
255
>>>
>>> b= 0x20
>>> b
32
>>> c = 0x0
>>> c
0
>>>
```

```
>>> bin(0)
'0b0'
>>>
>>> bin(8)
'0b1000'
>>>
>>> bin(32)
'0b100000'
>>>
>>> bin(255)
'0b11111111'
>>>
>>> a = 0b100
>>> a
4
>>> b = 0b1001
>>> b
9
>>> c = 0b11111111
>>> c
255
>>>
```

Python Numbers (Cont.) – Lab

```
>>> oct(8)
'0o10'
>>> oct(10)
'0o12'
>>> oct(64)
'0o100'
>>>
>>> a = 0o10
>>> a
8
>>> b = 0o12
>>> b
10
>>> c = 0o100
>>> c
64
>>>
```

```
>>> a = 1.23 + 0.32
>>> a
1.55
>>> b = 3.0 - 1.5
>>> b
1.5
>>> c = 2.1 * 2.0
>>> c
4.2
>>> d = 4.5 // 2.0
>>> d
2.0
>>> e = 4.5 % 2.0
>>> e
0.5
>>> f = 4.5 / 2.0
>>> f
2.25
>>>
>>> g = 43.2 - 43.1
>>> g
0.100000000000000142
>>> #부동 소수형의 정밀도의 한계
```

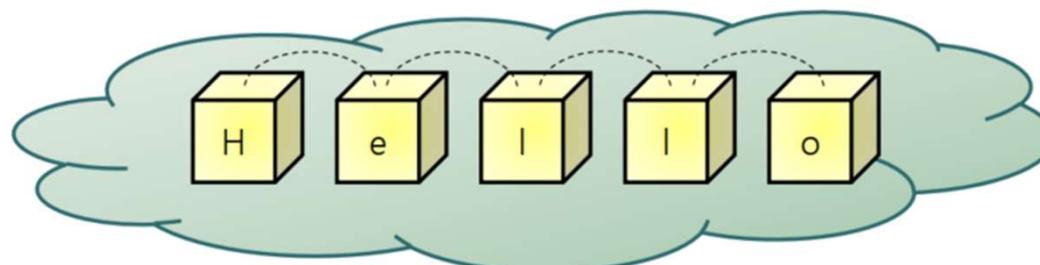
Python Numbers (Cont.) – Lab

```
>>> a = 2 + 3j  
>>> a  
(2+3j)  
>>> type(a)  
<class 'complex'>  
>>>  
>>> a.real  
2.0  
>>> a.imag  
3.0  
>>> a.conjugate()  
(2-3j)  
>>>
```

```
>>> a = (1 + 2j) + (3 + 4j)    # (a + bj) + (c + dj) = (a + c) + (b + d)j  
>>> a  
(4+6j)  
>>>  
>>> b = (1 + 2j) - (3 + 4j)  
>>> b  
(-2-2j)  
>>> c = (1 + 2j) * (3 + 4j)  
>>> c  
(-5+10j)  
>>> d = (1 + 2j) / (3 + 4j)  
>>> d  
(0.44+0.08j)  
>>>
```

Python Strings

- Strings in Python are identified as a contiguous set of characters represented in the *quotation marks*.
- Python allows either pair of *single* or *double quotes*.
- Subsets of strings can be taken using the slice operator (`[]` and `[:]`) with indexes starting at 0 in the beginning of the string and working their way from -1 to the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.



Python Strings (Cont.)



```
>>> s = "Monty python"  
>>> print(s[1])  
>>> print(s[6:10])  
>>> print[:4]  
>>> print[6:]  
>>> print[:]
```

Python Strings (Cont.) - Lab

```
>>> str = 'Hello World'  
>>>  
>>> print (str)          # Prints complete string  
Hello World!  
>>> print (str[0])       # Prints first character of the string  
H  
>>> print (str[2:5])     # Prints characters starting from 3rd to 5th  
llo  
>>> print (str[2:])      # Prints string starting from 3rd character  
llo World!  
>>> print (str * 2)       # Prints string two times  
Hello World!Hello World!  
>>> print (str + "TEST")   # Prints concatenated string  
Hello World!TEST  
>>>
```

Python Strings (Cont.) - Lab

```
1 name = 'Hello, World'
2 print(name.title())
3 print(name.upper())
4 print(name.lower())
5
6 first_name = 'michael'
7 last_name = 'jackson'
8 full_name = first_name + " " + last_name
9 print(full_name)
10 print("Hello, " + full_name.title() + "!")
11
12 print("Languages:\nPython\nC\nJavaScript")
13 print("Languages:\n\tPython\n\tC\n\tJavaScript")
```

Python Strings (Cont.) - Lab

```
1 favorite_language = 'python'  
2 print(favorite_language)  
3  
4 print(favorite_language.rstrip())  
5 print(favorite_language)  
6  
7 favorite_language = favorite_language.rstrip()  
8 print(favorite_language)  
9  
10 favorite_language = ' python '  
11 print(favorite_language.rstrip())  
12  
13 print(favorite_language.lstrip())  
14  
15 print(favorite_language.strip())  
16
```

Python Strings (Cont.) - Lab

```
1 #age = 23
2 message = "Happy " + age + "rd Birthday!"
3 print(message)
4
5
6 age = 23
7 message = "Heppy " + str(age) + "rd Birthday!"
8 print(message)
9
```

Python Lists

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets (`[]`).
- To some extent, lists are similar to arrays in C.
- One of the differences between them is that all the items belonging to a list can be of different data type.

Python Lists (Cont.)

- The values stored in a list can be accessed using the slice operator (`[]` and `[:]`) with indexes starting at 0 in the beginning of the list and working their way to end -1.
- The plus (`+`) sign is the list concatenation operator, and the asterisk (`*`) is the repetition operator.

Python Lists (Cont.) - Lab

```
>>> list = ['abcd', 786, 2.23, 'john', 70.2]
>>> tinylist = [123, 'john']
>>>
>>> print (list)                      # Prints complete list
['abcd', 786, 2.23, 'john', 70.2]
>>> print (list[0])                  # Prints first element of the list
abcd
>>> print (list[1:3])                # Prints elements starting from 2nd till 3rd
[786, 2.23]
>>> print (list[2:])                 # Prints elements starting from 3rd element
[2.23, 'john', 70.2]
>>> print (tinylist * 2)             # Prints list two times
[123, 'john', 123, 'john']
>>> print (list + tinylist)          # Prints concatenated lists
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
>>>
```

Python Tuples

- A tuple is another sequence data type that is similar to the *list*.
- A tuple consists of a number of values separated by commas.
- Lists are enclosed in brackets ([]) and their elements and size can be changed.
- But, tuples are enclosed in parentheses (()) and cannot be updated.
- Tuples can be thought of as *read-only lists*.

Python Tuples (Cont.) - Lab

```
>>> tuple = ('abcd', 786, 2.23, 'john', 70.2)
>>> tinytuple = (123, 'john')
>>>
>>> print (tuple)          # Prints complete tuple
('abcd', 786, 2.23, 'john', 70.2)
>>> print (tuple[0])      # Prints element of the tuple
abcd
>>> print (tuple[1:3])    # Prints elements starting from 2nd till 3rd
(786, 2.23)
>>> print (tuple[2:])     # Prints elements starting from 3rd element
(2.23, 'john', 70.2)
>>> print (tinytuple * 2)  # Prints tuple two times
(123, 'john', 123, 'john')
>>> print (tuple + tinytuple) # Prints concatenated tuple
('abcd', 786, 2.23, 'john', 70.2, 123, 'john')
>>>
```

Python Dictionary

- Is kind of *hash-table* type.
- Works like associative arrays or hashes found in Perl and consist of key-value pairs.
- A dictionary key can be almost any Python type, but are usually numbers or strings.
- Values, on the other hand, can be any arbitrary Python object.
- Is enclosed by curly braces (`{ }`) and values can be assigned and accessed using square braces (`[]`).

Python Dictionary (Cont.) - Lab

```
>>> dict = {}  
>>> dict['one'] = "this is one"  
>>> dict[2] = "This is two"  
>>>  
>>> tinydict = {'name' : 'john', 'code' : 6734, 'dept' : 'sales'}  
>>>  
>>> print (dict['one'])                                # Prints value for 'one' key  
this is one  
>>> print (dict[2])                                    # Prints value for 2 key  
This is two  
>>> print (tinydict)                                 # Prints complete dictionary  
{'name': 'john', 'code': 6734, 'dept': 'sales'}  
>>> print (tinydict.keys())                           # Prints all the keys  
dict_keys(['name', 'code', 'dept'])  
>>> print (tinydict.values())                         # Prints all the values  
dict_values(['john', 6734, 'sales'])  
>>>
```

Data Type Conversion

- Sometimes, may need to perform conversions between the *built-in* types.
- To convert between types, simply use the type-names as a function.
- There are several built-in functions to perform conversion from one data type to another.
- These functions return a new object representing the converted value.

Data Type Conversion (Cont.)

■ `int(x [,base])`

- Converts `x` to an integer.
- The base specifies the base if `x` is a string.

■ `float(x)`

- Converts `x` to a floating-point number.

■ `complex(real [, imag])`

- Creates a complex number.

■ `str(x)`

- Converts object `x` to a string representation.

Data Type Conversion (Cont.)

■ `repr(x)`

- Converts object `x` to an expression string.

■ `eval(str)`

- Evaluates a string and returns an object.

■ `tuple(s)`

- Converts `s` to a tuple.

■ `list(s)`

- Converts `s` to a list.

Data Type Conversion (Cont.)

■ `set(s)`

- Converts `s` to a set.

■ `dict(d)`

- Creates a dictionary. `d` must be a sequence of (key, value) tuples.

■ `frozenset(s)`

- Converts `s` to a frozen set.

■ `chr(x)`

- Converts an integer to a character.

Data Type Conversion (Cont.)

■ **unichr(x)**

- Converts an integer to a Unicode character.

■ **ord(x)**

- Converts a single character to its integer value.

■ **hex(x)**

- Converts an integer to a hexadecimal string.

■ **oct(x)**

- Converts an integer to an octal string.

Data Type Conversion - Lab

```
>>>
>>> int('1234567890')
1234567890
>>>
>>> float('123.4567')
123.4567
>>>
>>> complex('1 + 2j')
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    complex('1 + 2j')
ValueError: complex() arg is a malformed string
>>> complex('1+2j')
(1+2j)
>>>
```

Data Type Conversion - Lab

```
1 #input_test.py
2 print("첫 번째 숫자를 입력하세요 : ")
3 a = input()
4
5 print("두 번째 숫자를 입력하세요 : ")
6 b = input()
7
8 result = int(a) * int(b)
9
10 print("{0} * {1} = {2}".format(a, b, result))
11
```

```
Console >
<terminated> Hello.py [/usr/bin/python3.6]
첫 번째 숫자를 입력하세요 :
5
두 번째 숫자를 입력하세요 :
4
5 * 4 = 20
```

Operators

- Performs a function on one, two, or three operands and returns a result.
- An operator that requires one operand is called a *unary operator*.
- An operator that requires two operands is a *binary operator*.
- A *ternary operator* is one that requires three operands.
- Contains precedence order(top to bottom) along with their *associativity* (left to right or right to left).

Types of Operator

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Operators in Order of Precedence

No.	Operator	Description
1	<code>**</code>	Exponentiation (raise to the power)
2	<code>~, +, -</code>	Complement, unary plus and minus (method names for the last two are <code>+@</code> and <code>-@</code>)
3	<code>*, /, %, //</code>	Multiply, divide, modulo and floor division
4	<code>+, -</code>	Addition and subtraction
5	<code><<, >></code>	Right and left bitwise shift
6	<code>&</code>	Bitwise AND
7	<code>^, </code>	Bitwise exclusive OR and regular OR
8	<code><=, <, >, >=</code>	Comparison operators
9	<code><>, ==, !=</code>	Equality operators
10	<code>=, %=, /=, //=, -=, +=, *=, **=</code>	Assignment operators
11	<code>is, is not</code>	Identity operators
12	<code>in, not in</code>	Membership operators
13	<code>not, or, and</code>	Logical operators

Python Arithmetic Operators

- Assume variable **a** holds the value 10 and variable **b** holds the value 21.

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	$a + b = 31$
- Subtraction	Subtracts right hand operand from left hand operand.	$a - b = -11$
* Multiplication	Multiplies values on either side of the operator	$a * b = 210$
/ Division	Divides left hand operand by right hand operand	$b / a = 2.1$
% Modulus	Divides left hand operand by right hand operand and returns remainder	$b \% a = 1$

Python Arithmetic Operators (Cont.)

- Assume variable **a** holds the value 10 and variable **b** holds the value 20.

Operator	Description	Example
** Exponent	Performs exponential (power) calculation on operators.	$a \text{ } ** \text{ } b = 10^{20}$
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity):	$9 \text{ } // \text{ } 2 = 4$ $9.0 \text{ } // \text{ } 2.0 = 4.0$ $-11 \text{ } // \text{ } 3 = -4$ $-11.0 \text{ } // \text{ } 3 = -4.0$

Python Arithmetic Operators (Cont.)

```
>>> a = 21
>>> b = 10
>>> c = 0
>>>
>>> c = a + b
>>> print ("Line 1 - Value of c is ", c)
Line 1 - Value of c is 31
>>> c = a - b
>>> print ("Line 2 - Value of c is ", c)
Line 2 - Value of c is 11
>>> c = a * b
>>> print ("Line 3 - Value of c is ", c)
Line 3 - Value of c is 210
>>> c = a / b
>>> print ("Line 4 - Value of c is ", c)
Line 4 - Value of c is 2.1
>>> c = a % b
>>> print ("Line 5 - Value of c is ", c)
Line 5 - Value of c is 1
>>>
```

```
>>> a = 2
>>> b = 3
>>> c = a ** b
>>> print ("Line 6 - Value of c is ", c)
Line 6 - Value of c is 8
>>>
>>> a = 10
>>> b = 5
>>> c = a // b
>>> print ("Line 7 - Value of c is ", c)
Line 7 - Value of c is 2
>>>
```

Python Comparison Operators

- Assume variable **a** holds the value 10 and variable **b** holds the value 20

Operator	Description	Example
<code>==</code>	If the values of two operands are equal, then the condition becomes true.	(a <code>==</code> b) is not true.
<code>!=</code>	If values of two operands are not equal, then condition becomes true.	(a <code>!=</code> b) is true.

Python Comparison Operators (Cont.)

- Assume variable **a** holds the value 10 and variable **b** holds the value 20.

Operator	Description	Example
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Python Comparison Operators (Cont.)

```
>>> a = 21  
>>> b = 10  
>>>  
>>> if ( a == b ) :  
        print ("Line 1 - a is equal to b")  
else :  
        print ("Line 1 - a is not equal to b")
```

Line 1 - a is not equal to b

```
>>>  
>>> if (a != b) :  
        print ("Line 2 - a is not equal to b")  
else :  
        print ("Line 2 - a is equal to b")
```

Line 2 - a is not equal to b

```
>>>
```

```
>>> a = 21  
>>> b = 10  
>>>  
>>> if (a < b) :  
        print ("Line 3 - a is less than b")  
else :  
        print ("Line 3 - a is not less than b")
```

Line 3 - a is not less than b

```
>>>  
>>> if (a > b) :  
        print ("Line 4 - a is greater than b")  
else :  
        print ("Line 4 - a is not greater than b")
```

Line 4 - a is greater than b

```
>>>
```

Python Comparison Operators (Cont.)

```
>>> a = 21
>>> b = 10
>>> a, b = b, a # values of a and b swapped. a becomes 10, b becomes 21.
>>>
>>> if (a <= b) :
    print ("Line 5 - a is either less than or equal to b")
else :
    print ("Line 5 - a is neither less than nor equal to b")
```

Line 5 - a is either less than or equal to b

```
>>>
>>> if (b >= a) :
    print ("Line 6 - b is either greater than or equal to b")
else :
    print ("Line 6 - b is neither greater than nor equal to b")
```

Line 6 - b is either greater than or equal to b

```
>>>
```

Python Assignment Operators

- Assume variable **a** holds the value 10 and variable **b** holds the value 20.

Operator	Description	Example
=	Assigns values from right side operands to left side operand.	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand.	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand.	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand.	c *= a is equivalent to c = c * a

Python Assignment Operators (Cont.)

- Assume variable **a** holds the value 10 and variable **b** holds the value 20.

Operator	Description	Example
<code>/=</code> Divide AND	It divides left operand with the right operand and assign the result to left operand.	c <code>/=</code> a is equivalent to c = c / ac <code>/=</code> a is equivalent to c = c / a
<code>%=</code> Modulus AND	It takes modulus using two operands and assign the result to left operand.	c <code>%=</code> a is equivalent to c = c % a
<code>**=</code> Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand.	c <code>**=</code> a is equivalent to c = c ** a
<code>//=</code> Floor Division	It performs floor division on operators and assign value to the left operand.	c <code>//=</code> a is equivalent to c = c // a

Python Assignment Operators (Cont.)

```
>>> a = 21
>>> b = 10
>>> c = 0
>>>
>>> c = a + b
>>> print ("Line 1 - Value of c is ", c)
Line 1 - Value of c is 31
>>>
>>> c += a
>>> print ("Line 2 - Value of c is ", c)
Line 2 - Value of c is 52
>>>
>>> c *= a
>>> print ("Line 3 - Value of c is ", c)
Line 3 - Value of c is 1092
>>>
>>> c /= a
>>> print ("Line 4 - Value of c is ", c)
Line 4 - Value of c is 52.0
>>>
```

```
>>> a = 21
>>> c = 2
>>>
>>> c %= a
>>> print ("Line 5 - Value of c is ", c)
Line 5 - Value of c is 2
>>>
>>> c **= a
>>> print ("Line 6 - Value of c is ", c)
Line 6 - Value of c is 2097152
>>>
>>> c // a
>>> print ("Line 7 - Value of c is ", c)
Line 7 - Value of c is 99864
>>>
```

Python Bitwise Operators

- Bitwise operator works on bits and performs bit-by-bit operation.
Assume if **a** = 60; and **b** = 13.

Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands.	(a & b) (means 0000 1100)
 Binary OR	It copies a bit if it exists in either operand.	(a b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.)

Python Bitwise Operators (Cont.)

- Bitwise operator works on bits and performs bit-by-bit operation.
Assume if **a** = 60; and **b** = 13.

Operator	Description	Example
<code><<</code> Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a <code><<=</code> 240 (means 1111 0000)
<code>>></code> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a <code>>>=</code> 15 (means 0000 1111)

Python Bitwise Operators (Cont.)

```
>>> a = 60                      # 60 = 0011 1100
>>> b = 13                      # 13 = 0000 1101
>>> print ("a = ", a, ":", bin(a), ", b = ", b, ":", bin(b))
a = 60 : 0b111100 , b = 13 : 0b1101
>>>
>>> c = a & b                  # 12 = 0000 1100
>>> print ("Result of AND is ", c, ":", bin(c))
Result of AND is 12 : 0b1100
>>>
>>> c = a | b                  # 61 = 0011 1101
>>> print ("Result of OR is ", c, ":", bin(c))
Result of OR is 61 : 0b111101
>>>
>>> c = a ^ b                  # 49 = 0011 0001
>>> print ("Result of XOR is ", c, ":", bin(c))
Result of XOR is 49 : 0b110001
>>>
```

```
>>> a = 60
>>> c = 0
>>>
>>> c = ~a                      # -61 = 1100 0011
>>> print ("Result of COMPLEMENT is ", c, ":", bin(c))
Result of COMPLEMENT is -61 : -0b111101
>>>
>>> c = a << 2                 # 240 = 1111 0000
>>> print ("Result of LEFT SHIFT is ", c, ":", bin(c))
Result of LEFT SHIFT is 240 : 0b11110000
>>>
>>> c = a >> 2                 # 15 = 0000 1111
>>> print ("Result of RIGHT SHIFT is ", c, ":", bin(c))
Result of RIGHT SHIFT is 15 : 0b1111
>>>
```

Python Logical Operators

- Assume variable **a** holds **True** and variable **b** holds **False**.

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is False.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is True.
not Logical NOT	Used to reverse the logical state of its operand.	not (a and b) is True.

Python Logical Operators (Cont.)

```
>>> a = True
>>> b = False
>>>
>>> print ("a and b is ", (a and b))
a and b is False
>>>
>>> print ("a or b is ", (a or b))
a or b is True
>>>
>>> print ("not(a and b) is ", not(a and b))
not(a and b) is True
>>>
```

Python Membership Operators

- Python's membership operators test for membership in a sequence, such as **strings**, **lists**, or **tuples**. There are two membership operators as explained below:

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not find a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Python Membership Operators (Cont.)

```
>>> a = 10
>>> b = 20
>>> list = [1, 2, 3, 4, 5]
>>>
>>> if ( a in list ) :
    print ("Line 1 - a is available in the given list")
else :
    print ("Line 1 - a is not available in the given list")
```

Line 1 - a is not available in the given list

```
>>> if ( b not in list ) :
    print ("Line 2 - b is not available in the given list")
else :
    print ("Line 2 - b is available in the given list")
```

Line 2 - b is not available in the given list

```
>>>
>>> c = b / a
>>> if (c in list ) :
    print ("Line 3 - a is available in the given list")
else :
    print ("Line 3 - a is not available in the given list")
```

Line 3 - a is available in the given list

Python Identity Operators

- Identity operators compare the memory locations of two objects.
There are two Identity operators as explained below :

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here is results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here is not results in 1 if id(x) is not equal to id(y).

Python Identity Operators (Cont.)

```
>>> a = 20
>>> b = 20
>>>
>>> print ("Line 1 ", ", a = " , a, " : " , id(a), " , b = " , b, " : " , id(b))
Line 1 , a = 20 : 10804416 , b = 20 : 10804416
>>>
>>> if( a is b ) :
    print ("Line 2 - a and b have same identity")
else :
    print ("Line 2 - a and b do not have same identity")
```

Line 2 - a and b have same identity

```
>>>
>>> if ( id(a) == id(b)) :
    print ("Line 3 - a and b have same identity")
else :
    print ("Line 3 - a and b do not have same identity")
```

Line 3 - a and b have same identity

>>>

```
>>> a = 20
>>> b = 30
>>>
>>> print ("Line 4 ", ", a = " , a, " : " , id(a), " , b = " , b, " : " , id(b))
Line 4 , a = 20 : 10804416 , b = 30 : 10804736
>>>
>>> if ( a is not b) :
    print ("Line 5 - a and b do not have same identity")
else :
    print ("Line 5 - a and b have same identity")
```

Line 5 - a and b do not have same identity

>>>

Exercise. Python Operator



■ Question1

- 초단위의 시간을 입력 받아서 몇 분 몇 초인지 계산하여 출력하시오.
- 초단위의 시간을 입력하세요 : 1000
- 1000초는 16분 40초 입니다.

■ Question2

- 가지고 있는 돈으로 가격이 120원인 사탕을 최대한 몇 개까지 살 수 있는지 계산하고, 나머지 돈은 얼마인지 나타내시오.
- 가지고 있는 돈은 얼마입니까 ? : 5000
- 살 수 있는 사탕의 수 : 41개
- 사탕을 구입하고 남은 돈 : 80원

Simple Programming Constructs

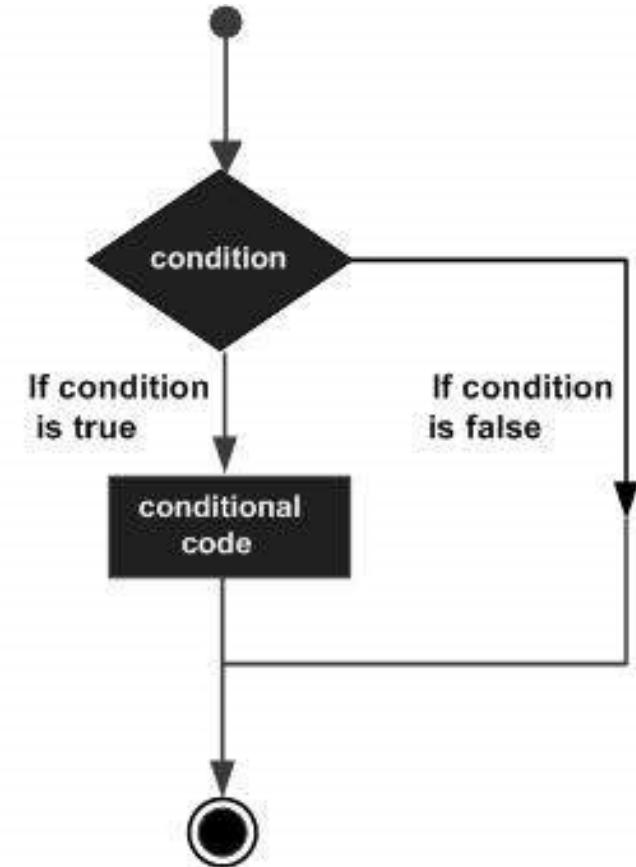
- Conditions(Decision Making) - Decide at runtime *whether to perform* certain statements.
- Loops - Decide at runtime *how many times to perform* certain statements.
- Branches

Decision Making

- Is the anticipation of conditions occurring during the execution of a program and specified actions taken according to the conditions.
- Decision structures evaluate multiple expressions, which produce **TRUE** or **FALSE** as the outcome.
- Need to determine which action to take and which statements to execute if the outcome is **TRUE** or **FALSE** otherwise.

Decision Making (Cont.)

Statement	Description
if statements	An if statement consists of a boolean expression followed by one or more statements.
if...else statements	An if statement can be followed by an optional else statement, which executes when the boolean expression is FALSE .
nested if statements	You can use one if or else if statement inside another if or else if statement(s).



IF Statement

- The **if** statement is similar to that of other languages.
- The **if** statement contains a logical expression using which the data is compared and a decision is made based on the result of the comparison.

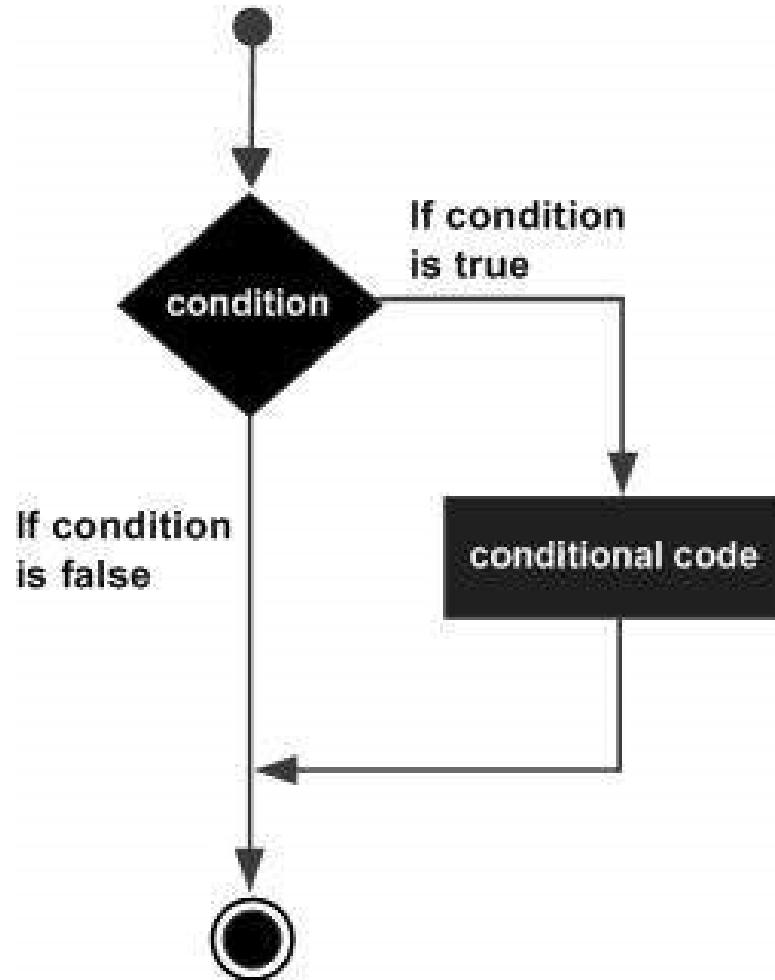
IF Statement (Cont.)

- Syntax

```
if expression :  
    statement(s)
```

- If the boolean expression evaluates to **TRUE**, then the block of statement(s) inside the if statement is executed.
- In Python, statements in a block are uniformly indented after the **:** symbol.
- If boolean expression evaluates to **FALSE**, then the first set of code after the end of block is executed.

IF Statement (Cont.)



IF Statement (Cont.)

```
>>> var1 = 100
>>> if var1 :
    print ("1 - Got a true expression value")
    print ( var1 )

1 - Got a true expression value
100
>>>
>>> var2 = 0
>>> if var2 :
    print ("2 - Got a true expression value")
    print ( var2 )

>>> print ("Good bye!")
Good bye!
>>>
```

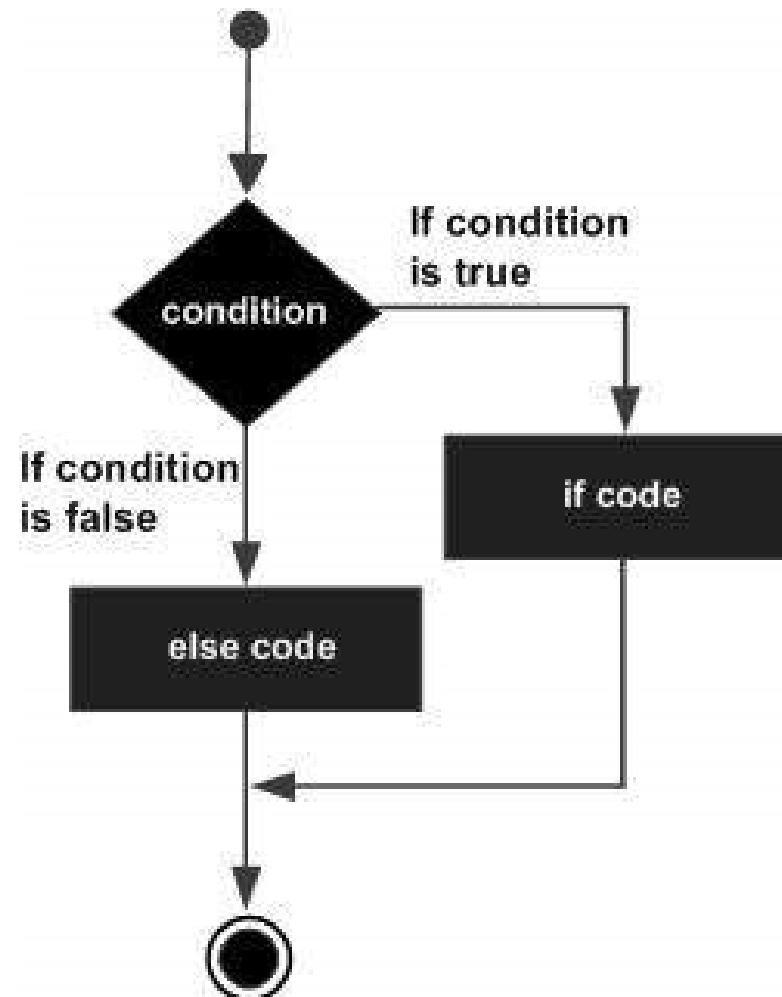
IF ... ELSE ... Statements

- An **else** statement can be combined with an **if** statement.
- An **else** statement contains a block of code that executes **if** the conditional expression in the **if** statement resolves to 0 or a **FALSE** value.
- The **else** statement is an optional statement and there could be at the most only one else statement following **if**.

IF ... ELSE ... Statements (Cont.)

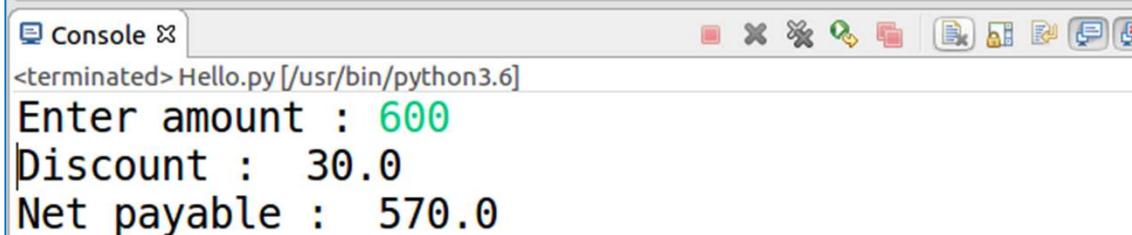
■ Syntax

```
if expression :  
    statement(s)  
  
else :  
    statement(s)
```



IF ... ELSE ... Statements (Cont.)

```
2 amount = int(input("Enter amount : "))
3
4 if amount < 1000 :
5     discount = amount * 0.05
6     print ("Discount : ", discount)
7 else :
8     discount = amount * 0.10
9     print ("Discount : ", discount)
10
11 print ("Net payable : ", amount - discount)
```



The screenshot shows a Python IDE interface with a code editor and a terminal window. The code editor contains the provided Python script. The terminal window is titled 'Console' and shows the output of running the script. The user enters '600' as the amount, and the program calculates a 10% discount of 60.0, resulting in a net payable amount of 540.0.

```
Console >
<terminated> Hello.py [/usr/bin/python3.6]
Enter amount : 600
Discount : 60.0
Net payable : 540.0
```

Exercise. Python if...else...



■ Question1

- 사용자로부터 두 수를 입력 받아 둘 중에서 큰 수를 출력하는 프로그램
- 예)첫번째 정수 : 10
- 두번째 정수 : 20
- 10과 20 중 큰 수는 20

■ Question2

- 항공사에서는 짐을 부칠 때 20kg이 넘어가면 20000원의 요금을 받는다. 사용자로부터 짐의 무게를 입력 받고 지불해야 할 금액을 계산하는 프로그램
- 예)짐의 무게는 ? : 18
- 짐에 대한 수수료는 없습니다. 감사합니다.
- 짐의 무게는 ? : 25
- 짐에 대한 요금은 20000원입니다. 감사합니다.

Exercise. Python if...else...



■ Question3

- 물건을 구입할 때 구입액이 10만원 이상이면 5%의 할인을 받을 수 있다. 사용자에게 물건 구입금액을 물어보고 할인금액과 최종 지불금액을 출력하는 프로그램
- 예) 물건을 구입한 금액은 ? : 110000
- 물건 구입시 할인금액 : 5500원
- 최종 지불금액 : 104500원

Exercise. Python if...else...



■ Question4

- 사용자에게 근무 시간과 시간당 임금을 물어본다. 기본 근무 시간은 주당 40시간이다. 40시간을 초과하면 1.5배의 임금을 받는다. 총 임금을 계산하는 프로그램을 만드시오.
- 예)근무시간 : 45
- 시간당 임금 : 10000
- 총 임금은 475,000원



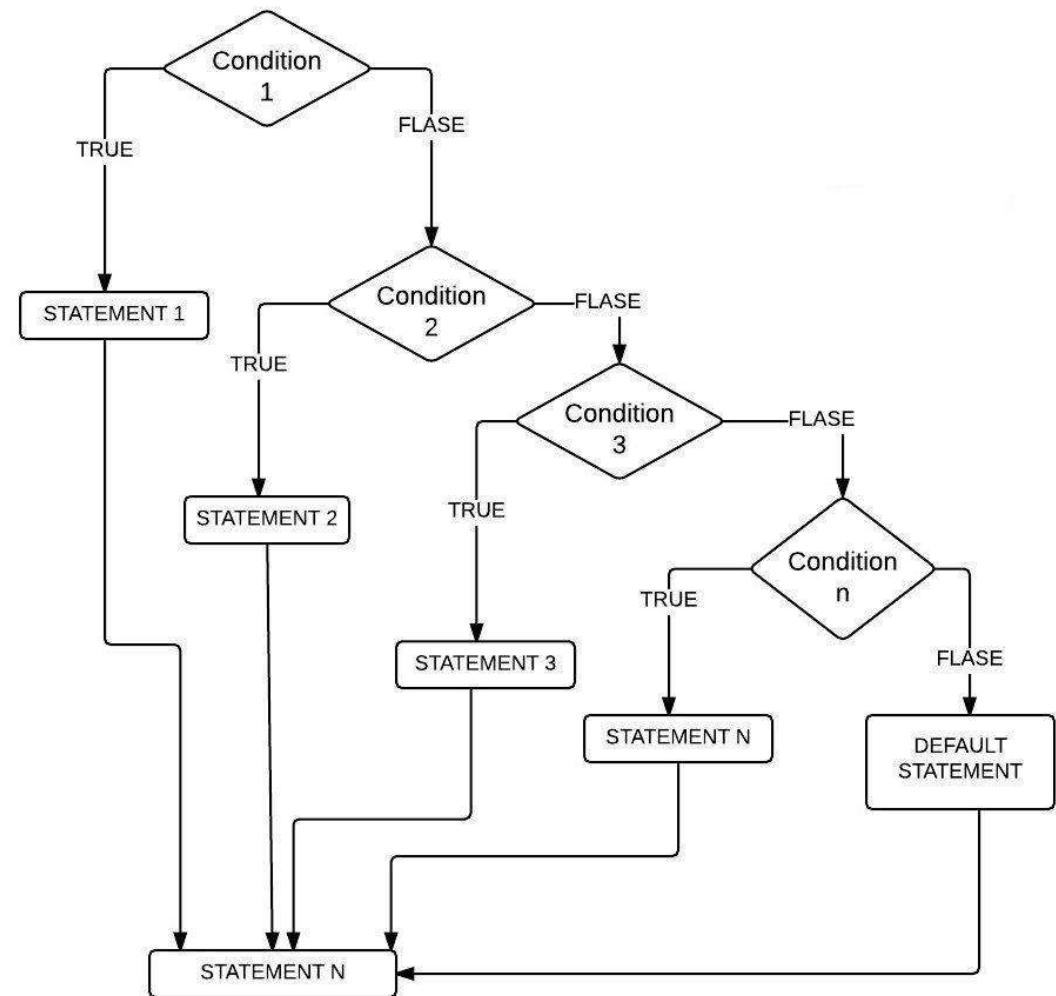
IF ... ELIF ... ELSE Statements

- The **elif** statement allows to check multiple expressions for **TRUE** and execute a block of code as soon as one of the conditions evaluates to **TRUE**.
- Similar to the **else**, the **elif** statement is optional.
- However, unlike **else**, for which there can be at the most one statement, there can be an arbitrary number of **elif** statements following an **if**.

IF ... ELIF ... ELSE Statements (Cont.)

■ Syntax

```
if expression1 :  
    statement(s)  
  
elif expression2 :  
    statement(s)  
  
elif expression3 :  
    statement(s)  
  
else :  
    statement(s)
```



IF ... ELIF ... ELSE Statements (Cont.)

- Core Python does not provide **switch** or **case** statements as in other languages.
- But can use **if...elif...** statements to simulate **switch case**.

IF ... ELIF ... ELSE Statements (Cont.)

```
2 amount = int(input("Enter amount : "))
3
4 if amount < 1000 :
5     discount = amount * 0.05
6     print ("Discount : ", discount)
7 elif amount < 5000 :
8     discount = amount * 0.10
9     print ("Discount : ", discount)
10 else :
11     discount = amount * 0.15
12     print ("Discount : ", discount)
13
14 print ("Net payable : ", amount - discount)
```

The screenshot shows a code editor with a Python script named 'Hello.py'. Below the editor is a terminal window titled 'Console' showing the execution and output of the script.

```
Console >
<terminated> Hello.py [/usr/bin/python3.6]
Enter amount : 3000
Discount : 300.0
Net payable : 2700.0
```

Nested IF Statements

- There may be a situation when want to check for another condition after a condition resolves to true.
- In such a situation, can use the ***nested if*** construct.
- In a ***nested if*** construct, can have an ***if...elif...else*** construct inside another ***if...elif...else*** construct.

Nested IF Statements (Cont.)

- Syntax

```
if expression1 :  
    statement(s)  
    if expression2 :  
        statement(s)  
    elif expression3 :  
        statement(s)  
    else :  
        statement(s)  
elif expression4 :  
    statement(s)  
else :  
    statement(s)
```

Nested IF Statements (Cont.)

```
2 num = int(input("Enter number : "))
3
4 if num % 2 == 0 :
5     if num % 3 == 0 :
6         print ("Divisible by 3 and 2")
7     else :
8         print ("Divisible by 2 not divisible by 3")
9 else :
10    if num % 3 == 0 :
11        print ("Divisible by 3 not divisible by 2")
12    else :
13        print ("Not divisible by 2 not divisible by 3")
14
15
```

The screenshot shows a Python code editor with a script named 'Hello.py' running in a terminal window. The code uses nested if statements to check if a user-specified number is divisible by both 2 and 3. The terminal output shows the program prompting for a number, receiving the input '8', and then printing the message 'Divisible by 2 not divisible by 3'.

```
Console > <terminated> Hello.py [/usr/bin/python3.6]
Enter number : 8
Divisible by 2 not divisible by 3
```

Single Statement Suites

- If the suite of an **if** clause consists only of a single line, it may go on the same line as the header statement.

```
>>> var = 100
>>>
>>> if ( var == 100 ) : print ("Value of expression is 100")
Value of expression is 100
>>> print ("Good bye!")
Good bye!
>>>
```

if - Lab

- What is the result of the following code ?

```
>>> a = "Life is too short, you need python"
>>>
>>> if "wife" in a : print("wife")
elif "python" in a and "you" not in a : print ("python")
elif "shirt" not in a : print ("shirt")
elif "need" in a : print ("need")
else : print ("none")
```

Exercise. Python if...elif...else...



■ Question1

- 항공사에서는 짐을 부칠 때 20kg이 넘어가면 추가 요금을 받는다. 사용자로부터 짐의 무게를 입력받고 지불해야 할 금액을 계산하는 프로그램을 작성하시오.
- 0 ~ 20kg : 무료, 20 ~ 30kg : kg당 1000원 추가요금, 30kg 초과 : 짐을 부칠 수 없음.
- 예) 짐의 무게 : 23
- 짐에 대한 요금은 3000원입니다. 감사합니다.
- 짐의 무게 : 33
- 짐을 부칠 수 없습니다. 감사합니다.

Exercise. Python if...elif...else...



■ Question2

- 사용자가 선택하는 도형의 면적을 계산하는 프로그램
- 예) 1 : 삼각형 2 : 사각형 3 : 사다리꼴 4 : 원
- 면적을 계산할 도형 선택 : 1
- 밑변 : 10
- 높이 : 50
- 삼각형의 면적 : 250

- 1 : 삼각형 2 : 사각형 3 : 사다리꼴 4 : 원
- 면적을 계산할 도형 선택 : 2
- 가로 : 10
- 세로 10
- 사각형의 면적 : 100

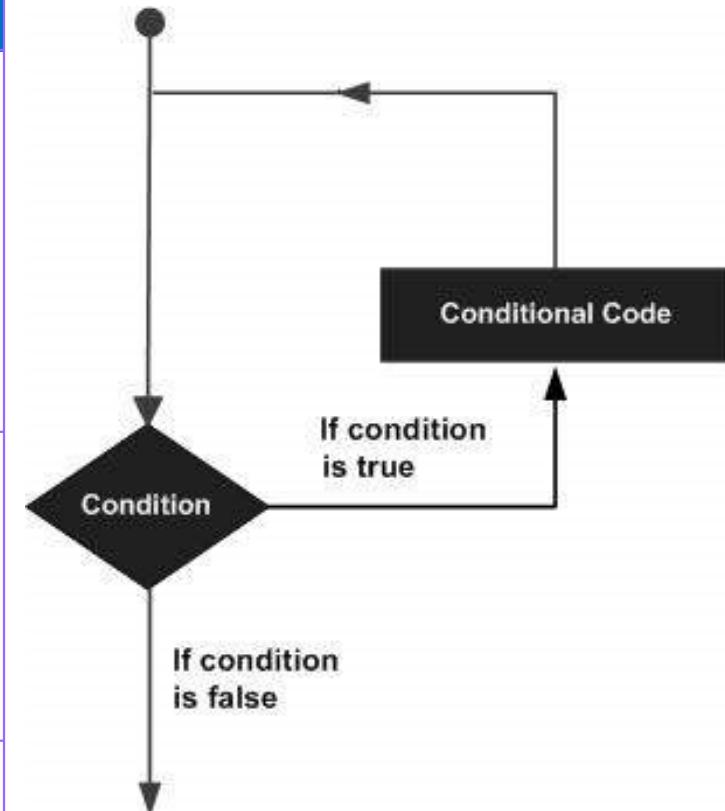


Loops

- In general, statements are executed sequentially.
- The first statement in a function is executed first, followed by the second, and so on.
- There may be a situation when need to execute a block of code several number of times.
- Programming languages provide various control structures that allow more complicated execution paths.
- A loop statement allows to execute a statement or group of statements multiple times.

Loops (Cont.)

Statement	Description
while loop	Repeats a statement or group of statements while a given condition is TRUE . It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
nested loop	You can use one or more loop inside any another while, or for loop.



WHILE Loop Statements

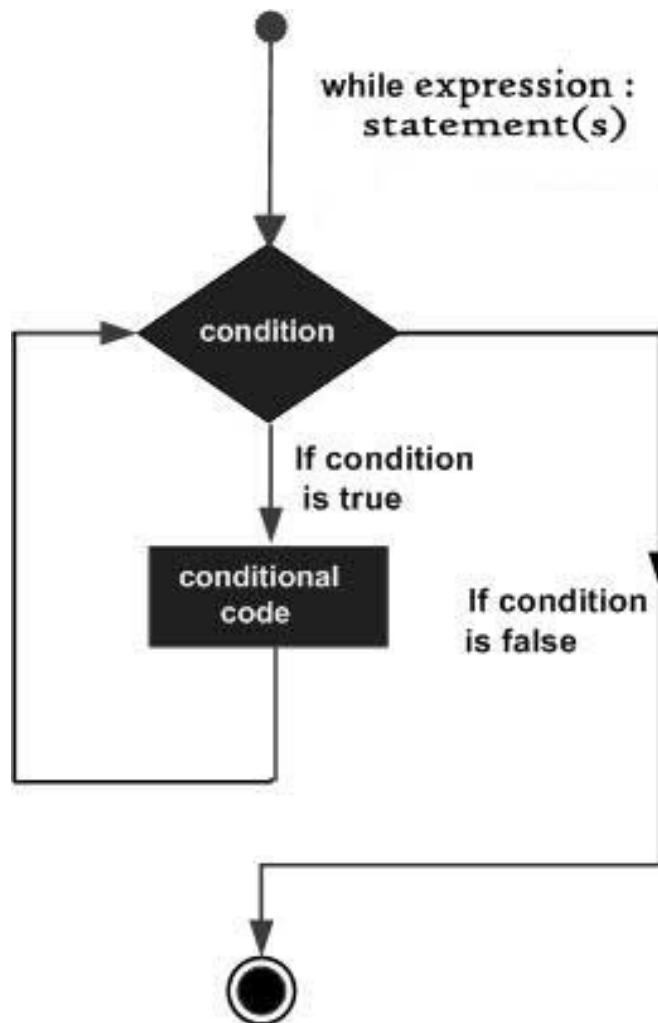
- Repeatedly executes a target statement as long as a given condition is *true*.
- Syntax

```
while expression :  
    statement(s)
```

WHILE Loop Statements (Cont.)

- Statement(s) may be a single statement or a block of statements with uniform indent.
- The condition may be any expression, and true is any non-zero value.
- The loop iterates while the condition is true.
- When the condition becomes false, program control passes to the line immediately following the loop.

WHILE Loop Statements (Cont.)



WHILE Loop Statements (Cont.)

```
>>> count = 0  
>>> while (count < 9 ) :  
    print ("The count is : ", count)  
    count = count + 1
```

```
The count is : 0  
The count is : 1  
The count is : 2  
The count is : 3  
The count is : 4  
The count is : 5  
The count is : 6  
The count is : 7  
The count is : 8  
>>>
```

The Infinite Loop

- A loop becomes infinite loop if a condition never becomes **FALSE**.
- Must be cautious when using while loops because of the possibility that this condition never resolves to a **FALSE** value.
- This results in a loop that never ends.
- Such a loop is called an infinite loop.

The Infinite Loop (Cont.)

```
>>> var = 1
>>> while var == 1 :          # This constructs an infinite loop
    num = int (input ("Enter a number : "))
    print ("You entered : ", num)
```

```
Enter a number : 5
You entered : 5
Enter a number : 7
You entered : 7
Enter a number : 9
You entered : 9
Enter a number : 11
You entered : 11
Enter a number : 13
You entered : 13
Enter a number : 15
You entered : 15
Enter a number :|
```

Using else Statement with Loops

- Python supports having an **else** statement associated with a loop statement.
 - If the **else** statement is used with a **for** loop, the **else** statement is executed when the loop has exhausted iterating the list.
 - If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes *false*.

Using else Statement with Loops

```
>>> count = 0
>>> while count < 5 :
    print (count, " is less than 5")
    count = count + 1
else :
    print (count, " is not less than 5")
```

```
0  is less than 5
1  is less than 5
2  is less than 5
3  is less than 5
4  is less than 5
5  is not less than 5
>>>
```

Single Statement Suites

- Similar to the **if** statement syntax, if **while** clause consists only of a single statement, it may be placed on the same line as the **while** header.

while - Lab

- **while**문을 이용하여 아래와 같이 별(*)을 표시하는 프로그램을 완성 하시오.

*

**

FOR Loop Statements

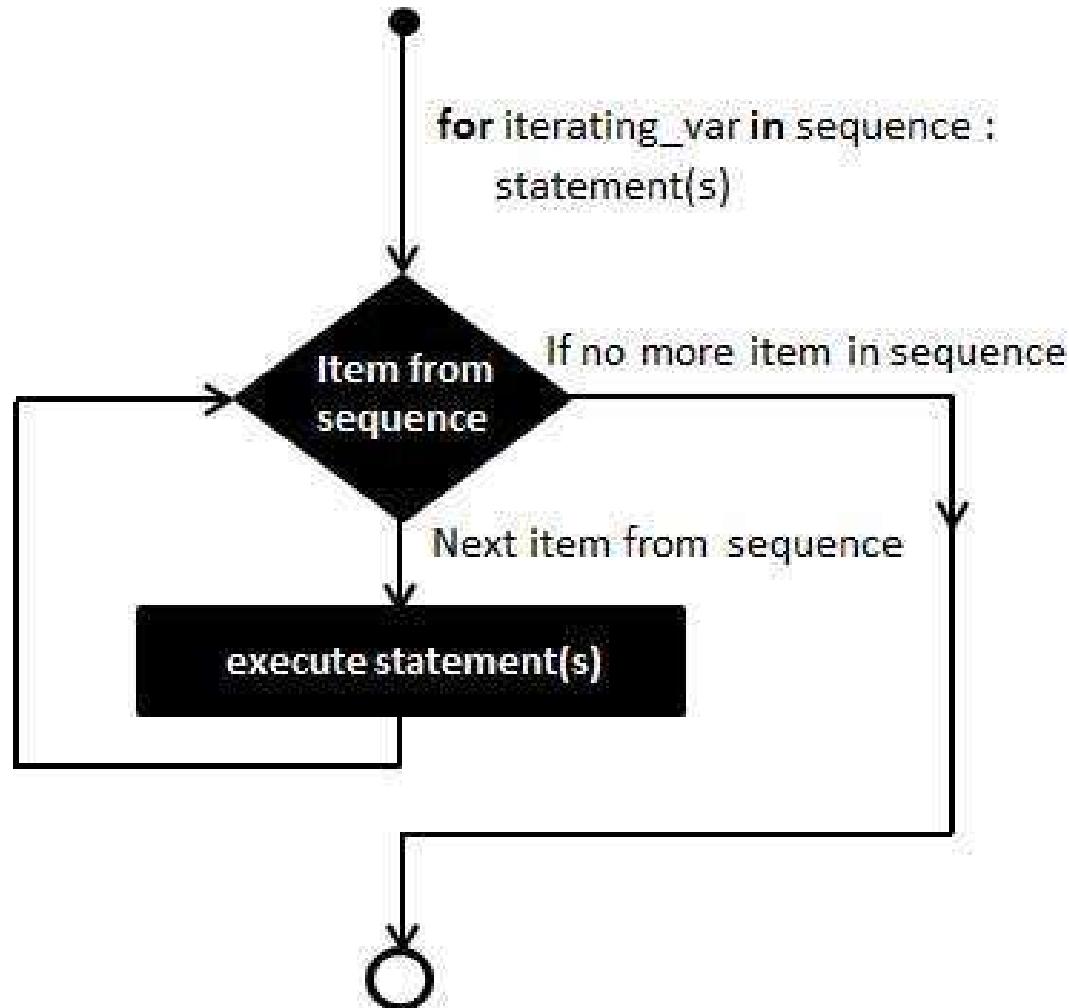
- Has the ability to iterate over the items of any sequence, such as a **list** or a **string**.
- Syntax

```
for iterating_var in sequene :  
    statement(s)
```

FOR Loop Statements (Cont.)

- If a sequence contains an expression list, it is evaluated first.
- Then, the first item in the sequence is assigned to the iterating variable **iterating_var**.
- Next, the statements block is executed.
- Each item in the list is assigned to **iterating_var**, and the statement(s) block is executed until the entire sequence is exhausted.

FOR Loop Statements (Cont.)



The range() function

- The built-in function `range()` is the right function to iterate over a sequence of numbers.
- It generates an iterator of arithmetic progressions.

```
>>> range(5)
range(0, 5)
>>>
>>> list(range(5))
[0, 1, 2, 3, 4]
>>>
```

The range() function (Cont.)

- **range()** generates an iterator to progress integers starting with 0 up to n-1.
- To obtain a list object of the sequence, it is typecasted to **list()**.
- Now this list can be iterated using the for statement.

```
>>> for var in list(range(5)) :  
    print (var)  
  
0  
1  
2  
3  
4  
>>>
```

FOR Loop Statements (Cont.)

```
>>> for letter in 'Python':           # traversal of a string sequence
    print ("Current Letter : ", letter)

Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
>>>
>>> fruits = ['Banana', 'Apple', 'Mango']
>>> for fruit in fruits:             # traversal of List sequence
    print ("Current fruit : ", fruit)

Current fruit : Banana
Current fruit : Apple
Current fruit : Mango
>>>
```

Iterating by Sequence Index

- An alternative way of iterating through each item is by index offset into the sequence itself.

```
>>> fruits = ['Banana', 'Apple', 'Mango']
>>> for index in range(len(fruits)) :
    print ("Current fruit : ", fruits[index])
```

```
Current fruit : Banana
Current fruit : Apple
Current fruit : Mango
>>>
```

Using else Statement with Loops

- Python supports having an **else** statement associated with a loop statement.
 - If the **else** statement is used with a **for** loop, the **else** block is executed only if **for** loops terminates normally (and not by encountering **break** statement).
 - If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false.

Using else Statement with Loops (Cont.)

```
>>> numbers = [11, 33, 55, 39, 55, 75, 37, 21, 23, 41, 13]
>>>
>>> for num in numbers :
    if num % 2 == 0 :
        print ("The list contains an even number")
        break
else :
    print ("The list does not contain even number")
```

The list does not contain even number

```
>>>
```

Nested Loops

- Python programming language allows the usage of one loop inside another loop.
- Syntax

```
for iterating_var in sequence :  
    for iterating_var in sequence :  
        statement(s)  
    statement(s)
```

Nested Loops (Cont.)

- The syntax for a nested **while** loop statement in Python programming language is as follow:
- Syntax

```
while expression :  
    while expression :  
        statement(s)  
    statement(s)
```

Nested Loops (Cont.)

```
>>> import sys  
>>> for i in range(1, 11):  
    for j in range(1, 11):  
        k = i * j  
        print(k, end=' ')  
    print()  
  
1 2 3 4 5 6 7 8 9 10  
2 4 6 8 10 12 14 16 18 20  
3 6 9 12 15 18 21 24 27 30  
4 8 12 16 20 24 28 32 36 40  
5 10 15 20 25 30 35 40 45 50  
6 12 18 24 30 36 42 48 54 60  
7 14 21 28 35 42 49 56 63 70  
8 16 24 32 40 48 56 64 72 80  
9 18 27 36 45 54 63 72 81 90  
10 20 30 40 50 60 70 80 90 100  
>>>
```

for - Lab

- A학급에 총 10명의 학생이 있다. 이 학생들의 중간고사 점수는 아래와 같다.

70, 60, 55, 75, 95, 90, 80, 80, 85, 100

for 문을 이용하여 A 학급의 평균 점수를 구해 보시오.

Branch(Loop Control) Statements

- Change the execution from its normal sequence.
- When the execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Statement	Description
break	Terminates the loop statement and transfers execution to the statement immediately following the loop.
continue	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
pass	The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

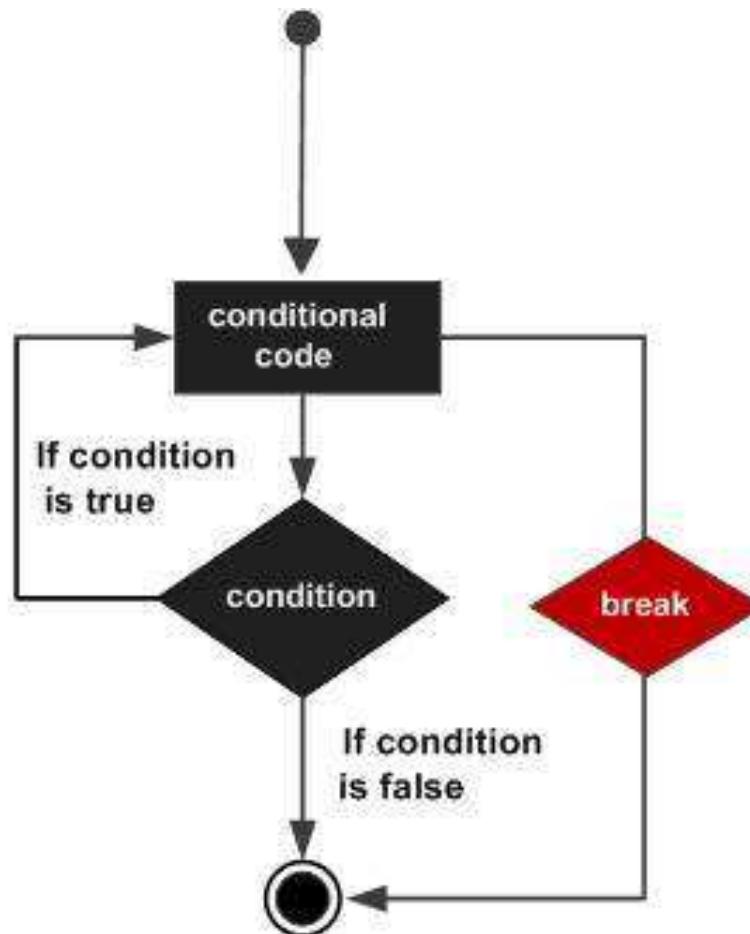
BREAK Statement

- Is used for premature termination of the current loop.
- After abandoning the loop, execution at the next statement is resumed, just like the traditional break statement in C.
- Is when some external condition is triggered requiring a hasty exit from a loop.
- The break statement can be used in both **while** and **for** loops.
- If you are using nested loops, the **break** statement stops the execution of the innermost loop and starts executing the next line of the code after the block.

BREAK Statement (Cont.)

■ Syntax

break



BREAK Statement (Cont.)

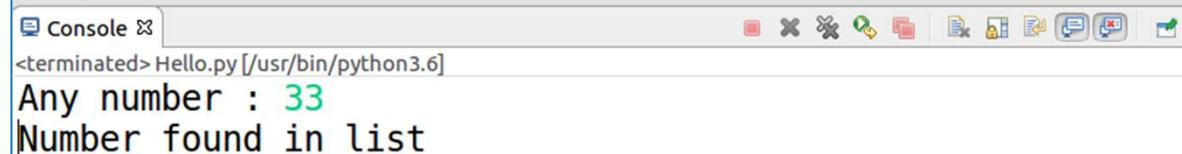
```
>>> for letter in 'Python' :          # First example
    if letter == 'h' :
        break
    print ("Current Letter : ", letter)
```

```
Current Letter : P
Current Letter : y
Current Letter : t
>>>
```

```
>>> var = 10                      # Second example
>>> while var > 0 :
    print ("Current variable value : ", var)
    var = var - 1
    if var == 5 :
        break;
```

```
Current variable value : 10
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
>>>
```

```
2 no = int(input( 'Any number : '))
3 numbers = [11, 33, 55, 39, 55, 75, 37, 21, 23, 41, 13]
4
5 for num in numbers :
6     if num == no :
7         print ( 'Number found in list')
8         break
9 else :
10    print ( 'Number not found in list')
```



The screenshot shows a Python IDE interface with a console window. The console window has a title bar labeled "Console" and a status bar indicating "<terminated> Hello.py [/usr/bin/python3.6]". The main area of the window displays the following text:
Any number : 33
Number found in list

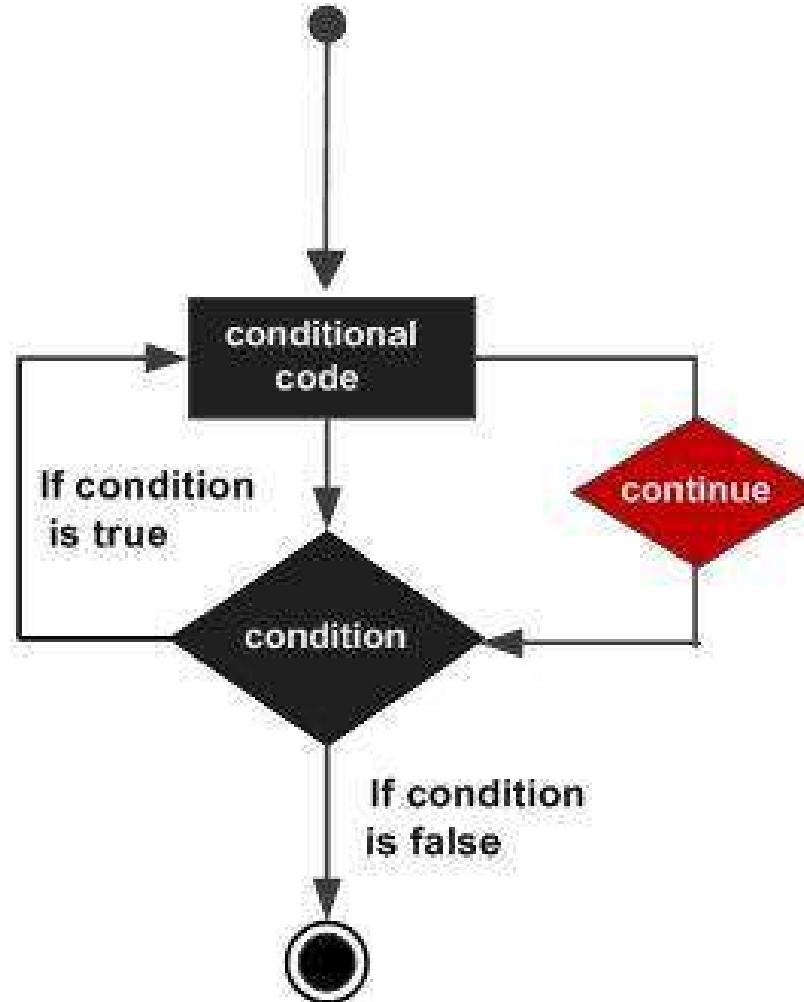
CONTINUE Statement

- Returns the control to the beginning of the current loop.
- When encountered, the loop starts next iteration without executing the remaining statements in the current iteration.
- The continue statement can be used in both **while** and **for** loops.

CONTINUE Statement (Cont.)

■ Syntax

continue



CONTINUE Statement (Cont.)

```
>>> for letter in 'Python':           # First Example
    if letter == 'h':
        continue
    print ('Current Letter : ', letter)
```

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
>>>
```

```
>>> var = 10                         # Second Example
>>> while var > 0 :
    var = var - 1
    if var == 5 :
        continue
    print ('Current variable value : ', var)
```

```
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Current variable value : 4
Current variable value : 3
Current variable value : 2
Current variable value : 1
Current variable value : 0
>>>
```

PASS Statement

- Is used when a statement is required syntactically but you do not want any command or code to execute.
- Is a null operation.
- Nothing happens when it executes.
- Is also useful in places where your code will eventually go, but has not been written yet i.e. in stubs.
- Syntax

pass

PASS Statement (Cont.)

```
>>> for letter in 'Python' :  
    if letter == 'h' :  
        pass  
        print ('This is pass block')  
    print ('Current Letter : ', letter)
```

```
Current Letter : P  
Current Letter : y  
Current Letter : t  
This is pass block  
Current Letter : h  
Current Letter : o  
Current Letter : n  
>>>
```

Exercise. Python Loop



■ Question1

- 1부터 50까지 자연수 중에서 3의 배수의 총합을 출력하시오.



■ Question2

- 다음과 같이 출력하는 프로그램을 작성하시오.

A	B	C	D	E
f	g	h	i	j
K	L	M	N	O
p	q	r	s	t
U	V	W	X	Y
z				

Functions

- Is a block of organized, reusable code that is used to perform a single, related action.
- Provide better modularity for application and a high degree of code reusing.
- As already know, Python gives many built-in functions like **print()**, etc.
- But can also create own functions.
- These functions are called *user-defined* functions.

Defining a Function

- Can define functions to provide the required functionality.
- Syntax

```
def function_name (parameters ) :  
    """function_docstring"""  
    function_suite  
    return [expression]
```

- Here are simple rules to define a function in Python.

Defining a Function (Cont.)

- Function blocks begin with the keyword **def** followed by the function name and parentheses (()).
- Any input parameters or arguments should be placed within these parentheses.
- Can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or **docstring**.

Defining a Function (Cont.)

- The code block within every function starts with a colon (:) and is indented.
- The statement `return [expression]` exits a function, optionally passing back an expression to the caller.
- A return statement with no arguments is the same as `return None`.

Defining a Function (Cont.)

- The following function takes a string as input parameter and prints it on standard screen.
- Example

```
def printme( str ) :  
    """This prints a passed string into this  
function """  
    print (str)  
    return
```

Calling a Function

- Defining a function gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.
- Once the basic structure of a function is finalized, can execute it by calling it from another function or directly from the Python prompt.

Calling a Function (Cont.)

```
1 # Function definition is here
2
3 def printme (str):
4     "This prints a passed string into this function"
5     print (str)
6     return
7
8 # Now you can call printme function
9 printme("This is first call to the user defined function!")
10 printme("Again second call to the same function")
```

Console ✘

<terminated> demo.py [/usr/bin/python3.6]

This is first call to the user defined function!
Again second call to the same function

Pass by Reference vs Value

- All parameters (arguments) in the Python language are *passed by reference*.
- It means if change what a parameter refers to within a function, the change also reflects back in the calling function.

Pass by Reference vs Value (Cont.)

```
1 # Function definition is here
2
3 def changeme (mylist):
4     "This changes a passed list into this function"
5     print ( "Values inside the function before change : ", mylist)
6     mylist[2] = 50
7     print ( "Values inside the function after change : ", mylist)
8     return
9
10 # Now you can call changeme function
11 mylist = [10, 20, 30]
12 changeme(mylist)
13 print ( "Values outside the function : ", mylist)
```

```
Console ✘
<terminated> demo.py [/usr/bin/python3.6]
Values inside the function before change : [10, 20, 30]
Values inside the function after change : [10, 20, 50]
Values outside the function : [10, 20, 50]
```

Pass by Reference vs Value (Cont.)

- The parameter **mylist** is *local* to the function **changeme**.
- Changing **mylist** within the function does *not* affect **mylist**.
- The function accomplishes nothing and finally this would produce the next slide result.

Pass by Reference vs Value (Cont.)

```
1 # Function definition is here
2
3 def changeme (mylist):
4     "This changes a passed list into this function"
5     mylist = [1, 2, 3, 4] # This would assign new reference in mylist
6     print ("Values inside the function : ", mylist)
7     return
8
9 # Now you can call changeme function
10 mylist = [10, 20, 30]
11 changeme(mylist)
12 print ("Values outside the function : ", mylist)
```

Console

<terminated> demo.py [/usr/bin/python3.6]

Values inside the function : [1, 2, 3, 4]
Values outside the function : [10, 20, 30]

Function Arguments

- Can call a function by using the following types of formal arguments :
 - Required arguments
 - Keyword arguments
 - Default arguments
 - Variable-length arguments

Required Arguments

- Are the arguments passed to a function in correct positional order.
- Here, the number of arguments in the function call should match exactly with the function definition.
- To call the function **printme()**, definitely need to pass one argument, otherwise it gives a syntax error.

Required Arguments (Cont.)

```
1 # Function definition is here
2
3 def printme (str):
4     print (str)
5
6
7 # Now you can call changeme function
8 printme()
```

Console ✘

<terminated> demo.py [/usr/bin/python3.6]

Traceback (most recent call last):

File "/home/instructor/PythonHome/0829/demo.py", line 8, in <module>
 printme()

TypeError: printme() missing 1 required positional argument: 'str'

Keyword Arguments

- Are related to the function calls.
- When use keyword arguments in a function call, the caller identifies the arguments by the parameter name.
- Allows to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters.
- Can also make keyword calls to the `printme()` function in the following ways.

Keyword Arguments (Cont.)

```
1 # Function definition is here
2
3 def printme (str):
4     print (str)
5
6
7 # Now you can call changeme function
8 printme( str = "My string")
```

Console ×

<terminated> demo.py [/usr/bin/python3.6]

My string

Keyword Arguments (Cont.)

```
1 # Function definition is here
2
3 def printinfo (name, age):
4     print ( "Name : ", name)
5     print ( "Age : ", age)
6
7
8 # Now you can call changeme function
9 printinfo ( age = 50, name = "Miki")
```

```
Console ✘
<terminated> demo.py [/usr/bin/python3.6]
Name : Miki
Age : 50
```

Default Arguments

- Is an argument that assumes a default value if a value is not provided in the function call for that argument.
- The next slide example gives an idea on default arguments, it prints default **age** if it is not passed.

Default Arguments (Cont.)

```
1 # Function definition is here
2
3 def printinfo (name, age = 35):
4     print ("Name : ", name)
5     print ("Age : ", age)
6
7
8 # Now you can call printinfo function
9 printinfo ( age = 50, name = "Miki")
10 printinfo ( name = "Sally")
```

Console

<terminated> demo.py [/usr/bin/python3.6]

Name : Miki

Age : 50

Name : Sally

Age : 35

Variable-length Arguments

- May need to process a function for more arguments than specified while defining the function.
- Are called *variable-length* arguments.
- Are not named in the function definition, unlike required and default arguments.

Variable-length Arguments (Cont.)

- Syntax for a function with non-keyword variable arguments is given below :

```
def function_name ([formal_args,] *var_args_tuple) :  
    function_suite  
    return [expression]
```

Variable-length Arguments (Cont.)

- An asterisk (*) is placed before the variable name that holds the values of all nonkeyword variable arguments.
- This *tuple* remains empty if no additional arguments are specified during the function call.

Variable-length Arguments (Cont.)

```
1 # Function definition is here
2
3 def printinfo ( arg1, *vartuple):
4     print ( "Output is : ")
5     print (arg1)
6     for var in vartuple :
7         print (var)
8
9
10 # Now you can call printinfo function
11 printinfo ( 10 )
12 printinfo ( 70, 60, 50)
```

Console

<terminated> demo.py [/usr/bin/python3.6]

```
Output is :
10
Output is :
70
60
50
```

The Anonymous Functions

- These functions are called anonymous because they are not declared in the standard manner by using the **def** keyword.
- Can use the **lambda** keyword to create small anonymous functions.

The Anonymous Functions (Cont.)

- Lambda forms can take any number of arguments but return just one value in the form of an expression.
- They cannot contain commands or multiple expressions.
- An anonymous function cannot be a direct call to print because **lambda** requires an expression.

The Anonymous Functions (Cont.)

- Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.
- Although it appears that lambdas are a one-line version of a function, they are not equivalent to inline statements in **C** or **C++**, whose purpose is to stack allocation by passing function, during invocation for performance reasons.

The Anonymous Functions (Cont.)

■ Syntax

- The syntax of **lambda** functions contains only a single statement.

```
lambda [arg1 [, arg2, .... Argn]] : expression
```

The Anonymous Functions (Cont.)

```
1 # Function definition is here
2
3 sum = lambda arg1, arg2 : arg1 + arg2
4
5
6 # Now you can call printinfo function
7 print ("Value of total : ", sum(10 ,20))
8 print ("Value of total : ", sum(20, 20))
```

```
Console ✘
<terminated> demo.py [/usr/bin/python3.6]
Value of total :  30
Value of total :  40
```

The return Statement

- The statement `return [expression]` exits a function, optionally passing back an expression to the caller.
- A `return` statement with no arguments is the same as `return None`.

The return Statement (Cont.)

```
1 # Function definition is here
2
3 def sum (arg1, arg2):
4     #Add both the parameters and return them.
5     total = arg1 + arg2
6     print ("Inside the function : ", total)
7     return total
8
9
10 # Now you can call printinfo function
11 total = sum(10, 20)
12 print ("Outside the function : ", total)
```

Console ✘

<terminated> demo.py [/usr/bin/python3.6]

Inside the function : 30
Outside the function : 30

Scope of Variables

- All variables in a program may not be accessible at all locations in that program.
- This depends on where have declared a variable.
- The scope of a variable determines the portion of the program where you can access a particular identifier.
- There are two basic scopes of variables in Python :
 - Global variables
 - Local variables

Global vs. Local variables

- Variables that are defined inside a function body have a *local* scope, and those defined outside have a *global* scope.
- This means that *local* variables can be accessed only inside the function in which they are declared, whereas *global* variables can be accessed throughout the program body by all functions.
- When call a function, the variables declared inside it are brought into scope.

Global vs. Local variables (Cont.)

```
1 # Function definition is here
2
3 total = 0          # This is global variable.
4
5 # Function definition is here
6 def sum (arg1, arg2):
7     # Add both the parameters and return them.
8     total = arg1 + arg2    # here total is local variable
9     print ("Inside the function local total : ", total)
10    return total
11
12
13 # Now you can call sum function
14 sum (10, 20)
15 print ("Outside the function global total : ", total)
```

Console

<terminated> demo.py [/usr/bin/python3.6]

Inside the function local total : 30
Outside the function global total : 0

Lab

```
1
2 def greet_user() :
3     """간단한 환영 인사를 표시한다"""
4     print ('Hello!')
5
6 greet_user()
7
```

Lab (Cont.)

```
1
2 def greet_user( username ) :
3     """간단한 환영 인사를 표시한다"""
4     print ( 'Hello, ' + username.title() + ' ! ')
5
6 greet_user( 'Jesse' )
7
```

Lab (Cont.)

```
1 def add_number(n1, n2):  
2     ret = n1 + n2  
3     return ret  
4  
5 def add_txt(t1, t2):  
6     print(t1 + t2)  
7  
8 ans = add_number(10, 15)  
9 print(ans)                  # 25가 출력됨  
10 text1 = 'Hello '  
11 text2 = 'World'  
12 add_txt(text1, text2)       # 'Hello World'가 출력됨  
13
```

Lab (Cont.)

```
1
2 def describe_pet(animal_type, pet_name):
3     """애완동물에 관한 정보를 출력한다"""
4     print ('\nI have a ' + animal_type + '.')
5     print ('My ' + animal_type + "'s name is " + pet_name.title() + ".")
6
7 describe_pet('hamster', 'harry')
8 describe_pet('dog', 'willie')
9
```

Lab (Cont.)

```
1
2odef describe_pet (animal_type,  pet_name):
3    """애완동물에 관한 정보를 출력한다"""
4    print ( '\nI have a ' + animal_type + '.')
5    print ( "My " + animal_type + "'s name is " + pet_name.title() + ".")
6
7 describe_pet( 'harry',  'hamster')
8
```

Lab (Cont.)

```
1
2 def describe_pet(animal_type, pet_name):
3     """애완동물에 관한 정보를 출력한다"""
4     print ('\nI have a ' + animal_type + '.')
5     print ("My " + animal_type + "'s name is " + pet_name.title() + ".")
6
7 describe_pet(animal_type = 'hamster', pet_name = 'harry')
8 describe_pet(pet_name = 'harry', animal_type = 'hamster')
9
```

Lab (Cont.)

```
1
2 def describe_pet (pet_name, animal_type = 'dog'):
3     """애완동물에 관한 정보를 출력한다"""
4     print ( '\nI have a ' + animal_type + '.')
5     print ( "My " + animal_type + "'s name is " + pet_name.title() + ".")
6
7 describe_pet(pet_name = 'willie')
8
```

Lab (Cont.)

```
1
2 def describe_pet (pet_name, animal_type = 'dog'):
3     """애완동물에 관한 정보를 출력한다"""
4     print ( '\nI have a ' + animal_type + '.')
5     print ( "My " + animal_type + "'s name is " + pet_name.title() + ".")
6
7 # 월리라는 개
8 describe_pet( 'willie')
9 describe_pet(pet_name = 'willie')
10
11 # 햄스터 해리
12 describe_pet( 'harry', 'hamster')
13 describe_pet(pet_name = 'harry', animal_type = 'hamster')
14 describe_pet(animal_type = 'hamster', pet_name = 'harry')
15
```

Lab (Cont.)

```
1
2 def build_person( first_name, last_name):
3     """사람에 관한 정보 딕셔너리를 반환하기"""
4     person = { 'first' : first_name, 'last' : last_name}
5     return person
6
7
8 musician = build_person( 'Jimi', 'Hendrix')
9 print (musician)
10
```

Lab (Cont.)

```
1
2 def build_person( first_name, last_name, age = None):
3     """사람에 관한 정보 딕셔너리를 반환하기"""
4     person = { 'first' : first_name, 'last' : last_name}
5     if age : person[ 'age' ] = age
6     return person
7
8
9 musician = build_person('Jimi', 'Hendrix', age = 27)
10 print (musician)
11
```

Lab (Cont.)

```
1
2 def make_pizza(*toppings):
3     """주문 받은 토픽 리스트 출력하기"""
4     print (toppings)
5
6 make_pizza( 'pepperoni' )
7 make_pizza( 'mushrooms' , 'green peppers' , 'extra cheese' )
8
```

Lab (Cont.)

```
1
2odef make_pizza(*toppings):
3    """만들려고 하는 피자를 요약한다"""
4    print ("\nMaking a pizza with the following toppings :")
5    for topping in toppings :
6        print("- " + topping)
7
8 make_pizza( 'pepperoni' )
9 make_pizza( 'mushrooms' , 'green peppers' , 'extra cheese' )
10
```

Lab (Cont.)

```
1 def add_txt(t1, t2='Python'):
2     print(t1 + ' : ' + t2)
3
4 add_txt('Best language : ')
5 add_txt(t2 = 'Java', t1='Good Language : ')
6
7 def func1(*args):
8     print(args)
9
10 def func2(width, height, **kwargs):
11     print(kwargs)
12
13 func1()                      # 빈 튜플 () 이 출력됨
14 func1(3, 5, 1, 5)            # (3, 5, 1, 5)가 출력됨
15
16 func2(10, 20)                # 빈 사전 {}이 출력됨
17 func2(10, 20, depth=50, color='blue') # {'depth':50, 'color':'blue'} 이 출력됨
18
```

Lab (Cont.)

```
1 param = 10
2 strdata = '전역변수'
3
4 def func1():
5     strdata = '지역변수'
6     print(strdata)
7
8 def func2(param):
9     param = 1
10
11 def func3():
12     global param
13     param = 50
14
15 func1()                      # '지역변수'가 출력됨
16 print(strdata)                # '전역변수'가 출력됨
17 print(param)                  # 10이 출력됨
18 func2(param)
19 print(param)                  # 10이 출력됨
20 func3()
21 print(param)                  # 50이 출력됨
22
```

Lab (Cont.)

```
1 def reverse(x, y, z):  
2     return z, y, x  
3  
4 ret = reverse(1, 2, 3)  
5 print(ret)                      # (3, 2, 1)이 출력됨  
6  
7 r1, r2, r3 = reverse('a', 'b', 'c')  
8 print(r1); print(r2); print(r3)    # 'c', 'b', 'a' 순으로 출력됨  
9
```

Lab (Cont.)

```
1 a = [1,2,3]
2 b = a
3 a[0] = 100
4 print (a)
5 print (b)
6
7 print( "id(a) = ", id(a), ", id(b) = ", id(b))
8
```

Lab (Cont.)

```
1 a = [1,2,3]
2 b = a[:]
3 a[0] = 100
4 print (a)
5 print (b)
6
7 print( "id(a) = ", id(a), ", id(b) = ", id(b))
8
```

Lab (Cont.)

```
1 import copy  
2  
3 a = [1,2,3]  
4 b = copy.deepcopy(a)  
5 a[0] = 100  
6 print (a)  
7 print (b)  
8  
9 print("id(a) = ", id(a), ", id(b) = ", id(b))  
10
```

Lab (Cont.)

```
1 def change(var):  
2     print ("Before change in function : ", var)  
3     var[0] = 100  
4     print ("After change in function : ", var)  
5  
6 a = [1, 2, 3]  
7 print ("Before call function : ", a)  
8 change(a)  
9 print ("After call function : ", a)  
10
```

Lab (Cont.)

```
1 def change(var):  
2     print ("Before change in function : ", var)  
3     var[0] = 100  
4     print ("After change in function : ", var)  
5  
6 a = [1, 2, 3]  
7 print ("Before call function : ", a)  
8 change(a[:])  
9 print ("After call function : ", a)  
10
```

Lab (Cont.)

```
2 def times(a, b):  
3     return a * b  
4  
5 print (times)  
6 print (times(10 ,10))  
7  
8 print(globals())  
9  
10 myTimes = times  
11  
12 r = myTimes(10, 10)  
13 print (r)  
14  
15 print (globals())  
16
```

```
2 g = lambda x, y : x * y  
3  
4 print (g(2, 3))  
5  
6 print ((lambda x : x * x)(3))  
7  
8 print (globals())  
9
```

Exercise. Python Function



■ Question1

- 함수 이름 : isPrime()
- 인자로 들어온 숫자가 소수인지 아닌지 판별하는 함수를 작성하시오.

■ Question2

- 함수 이름 : printPrime()
- 2부터 30개의 소수를 출력하는 함수를 작성하시오.

Python Files

- General text file
- CSV
- Excel
- JSON
- XML
- HTML



Lab. Using Text File in Python

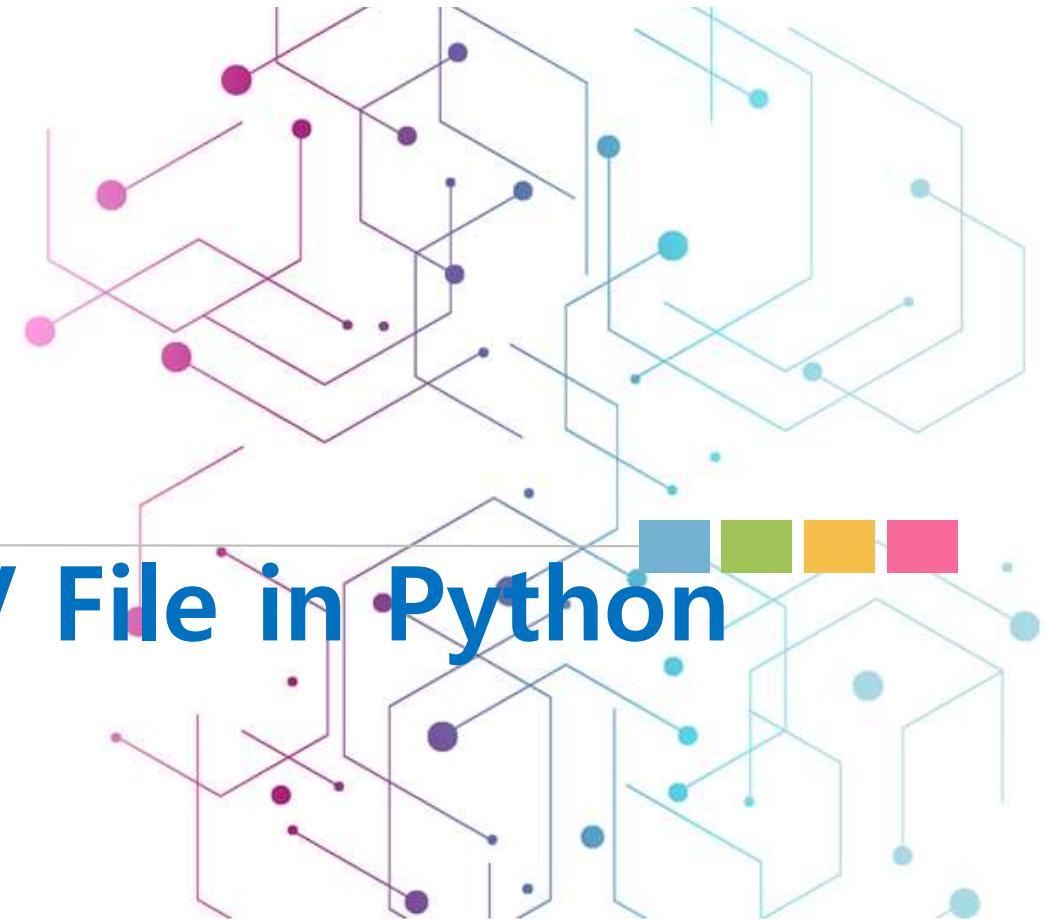


CSV File

- Comma-Separate-Values
- 값에 대한 유형이 없고, 모든 것이 문자열
- 글꼴 크기 또는 색상에 대한 설정이 없다.
- 여러 개의 Worksheet가 없다.
- Cell 너비와 높이를 지정할 수 없다.
- 병합된 cell을 가질 수 없다.
- Image 또는 chart를 포함할 수 없다.
- Python에서는 **csv** module 또는 pandas 이용.



Lab. Using CSV File in Python



JSON File

- Stands for JavaScript Object Notation
- Is lightweight data interchange format.
- Is language independent.
- Is "self-describing"(human readable) and easy to understand.
- Is an easier to use alternative to XML.
- Python **json** module
 - Python Data \leftrightarrow JSON

JSON File (Cont.)

■ JSON과 Python 자료형 Mapping

JSON	설명	Python
object	순서가 없는 key와 값의 쌍	dict
array	순서가 있는 sequence들의 값	list
string	문자열	str
number(int)	double-precision in JavaScript	int
number(float)	floating-point format in JavaScript	float
true, false	Boolean	True, False
null	empty	None

JSON File (Cont.)

■ JSON Encoding

- Python Object(dict, list, tuple 등) → JSON 문자열
- `import json`
- JSON의 `dumps()` 사용

■ JSON Decoding

- JSON 문자열 → Python Object
- `import json`
- JSON의 `loads()`



Lab. Using JSON File in Python



HTML File

- HyperText Markup Language
- Python's **urllib3** module 이용
- **HTTPError** (a subclass of **URLError**)
 - Is useful when handling exotic HTTP errors.
- 외부 **requests** module 이용
 - \$ **pip install requests**



Lab. Using HTML File in Python



Python Database Programming

- Must download a separate DB API module for each database.
- The DB API provides a minimal standard for working with databases.
 - Importing the API module.
 - Acquiring a connection with the database.
 - Issuing SQL statements and stored procedures.
 - Closing the connection

Python Database Programming (Cont.)

- Python has an in-built support for **SQLite**.
 - DB-API 2.0 interface for SQLite databases
 - <https://docs.python.org/3.7/library/sqlite3.html>
- MySQL / MariaDB
 - <https://pypi.org/project/PyMySQL/>
 - <https://github.com/PyMySQL/PyMySQL/blob/master/example.py>
 - <https://dev.mysql.com/doc/connector-python/en/>
 - **pymysql** module / **mysql-connector-python** module

Python Database Programming (Cont.)

■ Oracle

- <https://www.oracle.com/database/technologies/appdev/python.html>
- https://oracle.github.io/python-cx_Oracle/
- Installation **cx_Oracle**

```
$ pip install cx_Oracle --upgrade
```

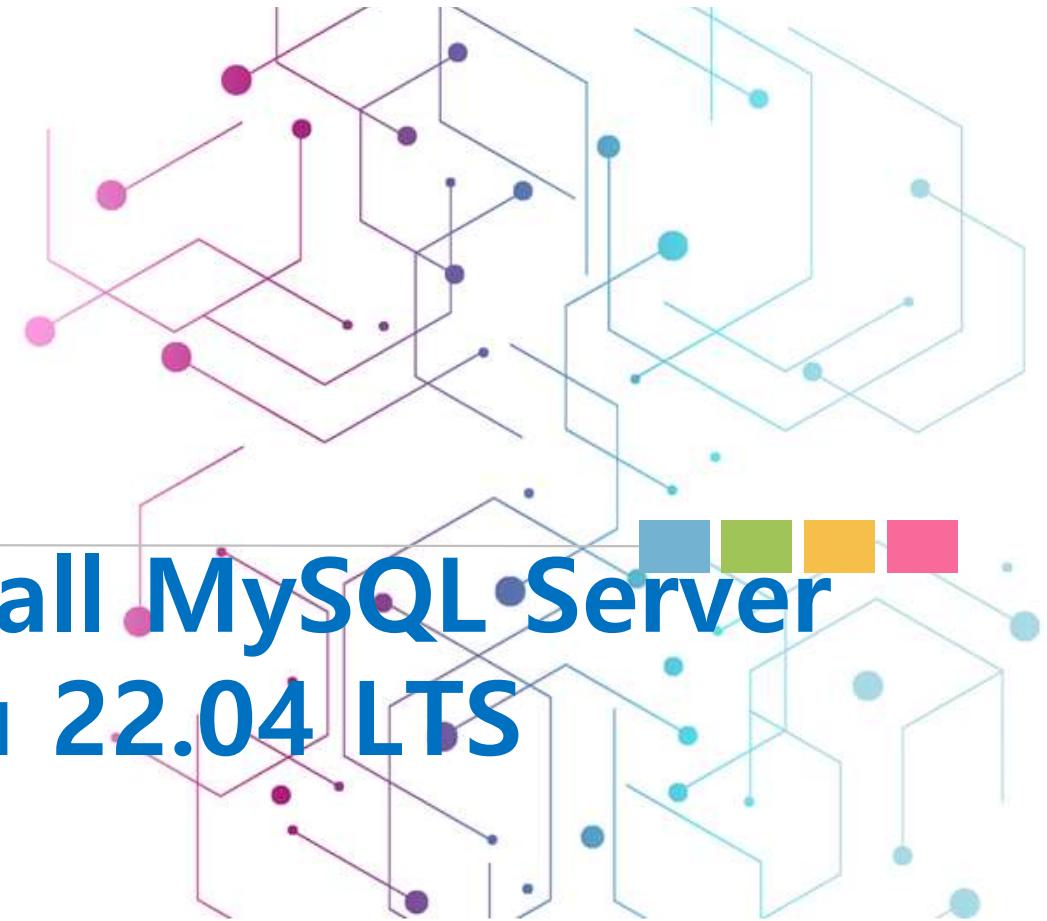
■ MongoDB

- NOSQL
- Installation **pymongo**

```
$ pip install pymongo
```

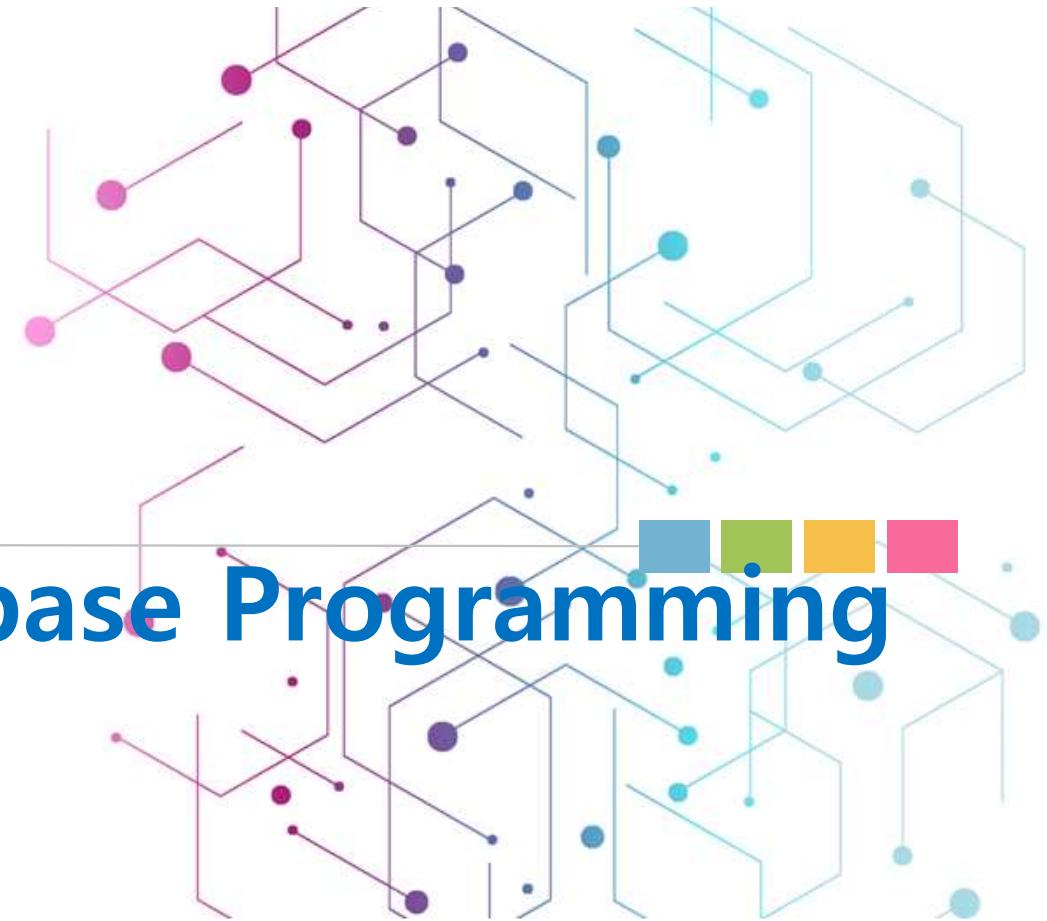


Lab. How to Install MySQL Server on Ubuntu 22.04 LTS





Lab. Python Database Programming



웹 데이터 수집 종류

- 뉴스 기사 크롤링
 - RSS 서비스(예: <http://news.jtbc.joins.com/Etc/RssService.aspx>)
 - XML 형식
- SNS 데이터 크롤링
 - 트위터, 댓글 등
 - Text 형식 또는 JSON 형식
- 웹 데이터 수집
 - 웹 페이지 URL에서 원하는 정보를 수집
 - HTML 형식
- 공공데이터 수집
 - Open API를 이용
 - URL이 존재
 - CSV 형식, 정형화된 데이터
 - Pandas 패키지의 `read_csv("url")`을 이용
 - JSON 형식
 - Pandas 패키지의 `read_json("url")`을 이용

데이터를 선택적으로 가져옴

전체 데이터를 가져옴

Crawling & Scraping

■ Crawling

- Web site의 data를 그대로 취득하는 것

■ Scraping

- Crawling하여 모든 취득한 data에서 필요한 것만 추출하거나 변환하는 처리 포함.

requests

- Python에서 HTTP 요청을 만들기 위한 사실상의 표준
- 학습할 내용
 - 가장 일반적인 HTTP 메소드를 사용하여 요청하기
 - 쿼리 문자열 및 메시지 본문을 사용하여 사용자의 요청과 데이터를 변경하기
 - 요청 및 응답에서 데이터 검사하기
 - 인증 된 요청 만들기
 - 응용 프로그램이 백업 또는 속도 저하를 막을 수 있도록 요청을 구성하기

requests module

- pip 명령으로 쉽게 설치
 - **pip install requests**
- requests가 설치되면 응용 프로그램에서 사용
 - **import requests**

GET 요청

- GET은 HTTP 요청(Request) 방식 중 하나
- **requests.get()** : 지정된 resource에서 데이터를 가져옴

```
1 import requests
```

```
1 requests.get('https://api.github.com')
```

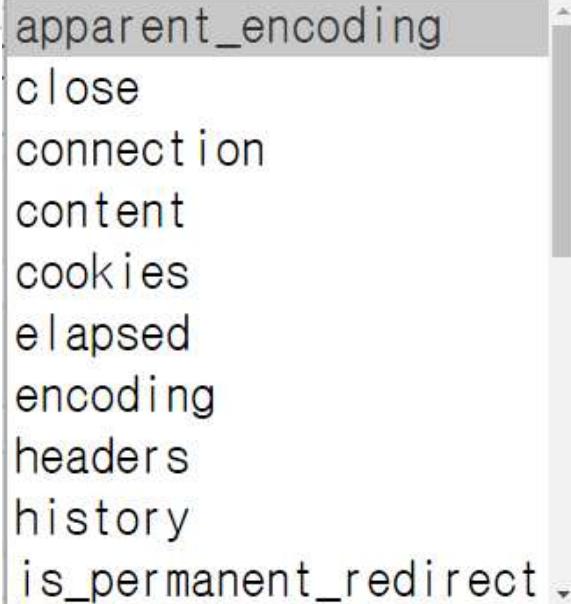
<Response [200]>

Response 객체

- 응답(Response)은 요청의 결과를 저장한 객체

```
1 response = requests.get('https://api.github.com')
```

```
1 response.
```



Status Code

- HTTP 응답 메시지의 상태 라인에는 상태 코드와 상태 메시지를 포함하고 있음

상태 코드	의미	내 용
100번 영역	정보전송	임시적인 응답을 나타내는 것은 Status-Line과 선택적인 헤더들로 구성되어 있고, 빈 줄로 끝을 맺는다.
200번 영역	성공	클라이언트의 요구가 성공적으로 수신되어 처리되었음을 의미 200(성공), 201(POST요청 처리), 204(전송할 데이터 없음)
300번 영역	리다이렉션	해당 요구사항을 처리하기 위해서는 사용자 에이전트에 의해 수행되어야 할 추가적인 동작이 있음을 의미
400번 영역	클라이언트 측 오류	클라이언트가 서버에게 보내는 요구 메시지를 완전히 처리하지 못한 경우 와 같이 클라이언트에서 오류가 발생한 경우에 사용 401(사용자인증), 403(접근권한 없음), 404(요청한 URL 없음)
500번 영역	서버측 오류	서버 자체에서 발생된 오류상황이나 요구사항을 제대로 처리할 수 없을 때 사용.

Status Code (Cont.)

```
1 response.status_code
```

200

```
1 if response.status_code == 200:  
2     print('Success!')  
3 elif response.status_code == 404:  
4     print('Not Found.')
```

상태코드 정보를 사용하여 코드에서 의사 결정을 내릴 수 있음

Success!

Response 인스턴스를 사용하면 상태 코드가 200에서 400 사이 이면 True로 평가되고 그렇지 않으면 False로 평가

```
1 if response:  
2     print('Success!')  
3 else:  
4     print('An error has occurred.')
```

Success!

Content

- GET 요청의 응답의 메시지 본문에는 중요한 정보가 포함되어있는 경우가 많음
- 응답 내용을 바이트 단위로 보려면 **.content**를 사용

```
1 response = requests.get('https://api.github.com')
2 response.content
```

```
b'{"current_user_url": "https://api.github.com/user", "current_user_authorizations_html_url": "https://github.com/settings/connections/applications{/client_id}", "authorizations_url": "https://api.github.com/authorizations", "code_search_url": "https://api.github.com/search/code?q={query}{&page,per_page,sort,order}", "commit_search_url": "https://api.github.com/search/commits?q={query}{&page,per_page,sort,order}"}
```

Content (Cont.)

- **.text** 속성에 액세스 할 때 문자열 인코딩을 지정할 수 있음
 - 내부적으로 추론해서 자동 지정됨

```
1 response = requests.get('https://api.github.com')
2 response.encoding = 'utf-8'
3 response.text
```

```
'{"current_user_url": "https://api.github.com/user", "current_user_authorizations_html_url": "https://github.com/settings/connections/applications/f4f3e013-1234-44d7-8924-77ac1f8ef33"}'
```

```
1 response = requests.get('http://javaspecialist.co.kr')
2 response.content
```

```
1 response = requests.get('http://javaspecialist.co.kr/emp')
2 response.text
```

```
b'<!DOCTYPE html>\n<html manifest="index.manifest">\n<head>\n<meta charset="utf-8"\n      name="viewport" content="width=device-width, initial-scale=1.0"\n      name="robots" content="index, follow"\n      http-equiv="X-UA-Compatible" content="IE=edge"\n      name="google-site-verification" content="RPK2fDgygxieiosW4a4cE8zola1193rlzAf90zQjTvc"\n      <title>WxecWx9eWx90WxbWxb0Wx94WxecWxa0Wx84WxbWxacWxb8WxeaxWxb0Wx80WxeaxWxb7Wxb8Wxb7Wxa3Wxb9</title>\n<!-- Favicon -->
```

Content (Cont.)

- **.text** 속성에 액세스 할 때 문자열 인코딩을 지정할 수 있음

```
1 response = requests.get('https://api.github.com')
2 response.json()
```

```
{'current_user_url': 'https://api.github.com/user',
 'current_user_authorizations_html_url': 'https://github.com/settings/connections/applications{/client_id}',
 'authorizations_url': 'https://api.github.com/authorizations',
 'code_search_url': 'https://api.github.com/search/code?q={query}{&page,per_page,sort,order}',
 'commit_search_url': 'https://api.github.com/search/commits?q={query}{&page,per_page,sort,order}',
```

Response Headers

```
response.headers
```

```
{'Date': 'Sat, 08 Jun 2019 11:52:45 GMT', 'Content-Type':  
    'application/json; charset=utf-8', 'Transfer-Encoding': 'chunked',  
    'Server': 'GitHub.com', 'Status': '200 OK', 'X-RateLimit-Limit': '60',  
    'X-RateLimit-Remaining': '58', 'X-RateLimit-Reset': '1559998358',  
    'Cache-Control': 'public, max-age=60, s-maxage=60', 'Vary': 'Accept,  
    Accept-Encoding', 'ETag': 'W/"7dc470913f1fe9bb6c7355b50a0737bc"',  
    'X-GitHub-Media-Type': 'github.v3; format=json',  
    'Access-Control-Expose-Headers': 'ETag, Link, Location, Retry-After,  
    X-GitHub-OTP, X-RateLimit-Limit, X-RateLimit-Remaining, X-RateLimit-Reset,  
    X-OAuth-Scopes, X-Accepted-OAuth-Scopes, X-Poll-Interval,  
    X-GitHub-Media-Type', 'Access-Control-Allow-Origin': '*',  
    'Strict-Transport-Security': 'max-age=31536000; includeSubdomains;  
    preload', 'X-Frame-Options': 'deny', 'X-Content-Type-Options': 'nosniff',  
    'X-XSS-Protection': '1; mode=block', 'Referrer-Policy':  
    'origin-when-cross-origin, strict-origin-when-cross-origin',  
    'Content-Security-Policy': "default-src 'none'", 'Content-Encoding':  
    'gzip', 'X-GitHub-Request-Id': 'DC9A:300B:149E1B4:18FE584:5CFBA18D'}
```

```
1 response.headers['Content-Type']
```

```
'application/json; charset=utf-8'
```

Request Parameter 사용하기

```
1 import requests  
2  
3 # 깃허브 저장소에서 검색 파라미터 사용하기  
4 response = requests.get(  
5     'https://api.github.com/search/repositories',  
6     params={'q': 'requests+language:python'}  
7 )  
8  
9 # 'requests' 저장소의 속성 검사  
10 json_response = response.json()  
11 repository = json_response['items'][0]  
12 print(f'Repository name: {repository["name"]}')  
13 print(f'Repository description: {repository["description"]}')
```

params 매개변수를 이용
해서 tuple 형식으로 지정
가능

Repository name: requests

Repository description: Python HTTP Requests for Humans™



Request Headers

```
1 import requests
2
3 response = requests.get(
4     'https://api.github.com/search/repositories',
5     params={'q': 'requests+language:python'},
6     headers={'Accept': 'application/vnd.github.v3.text-match+json'},
7 )
8
9 json_response = response.json()
10 repository = json_response['items'][0]
11 print(f'Text matches: {repository["text_matches"]}')
```

get() 메소드에 headers 매개
변수를 사용하여 HTTP 헤더
를 딕셔너리 형식으로 전달

```
Text matches: [{object_url: 'https://api.github.com/repositories/1362490', object_type: 'Repository', property: 'description', fragment: 'Python HTTP Requests for Humans™ 🌟🎨🌟', matches: [{text: 'Requests', indices: [12, 20]}]}]
```

다른 HTTP 요청 방법

```
1 requests.post('https://httpbin.org/post', data={'key':'value'})
```

```
<Response [200]>
```

```
1 requests.put('https://httpbin.org/put', data={'key':'value'})
```

```
<Response [200]>
```

```
1 requests.delete('https://httpbin.org/delete')
```

```
<Response [200]>
```

```
1 requests.head('https://httpbin.org/get')
```

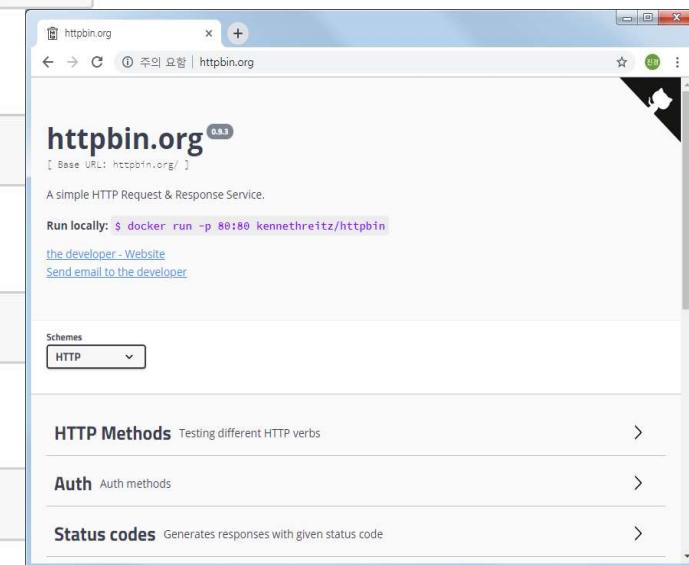
```
<Response [200]>
```

```
1 requests.patch('https://httpbin.org/patch', data={'key':'value'})
```

```
<Response [200]>
```

```
1 requests.options('https://httpbin.org/get')
```

```
<Response [200]>
```



httpbin은 테스트 요청을 받아들이고 요청에 대한 데이터로 응답하는 서비스

메시지 본문

- POST, PUT 요청은 메시지 본문을 통해 데이터를 전달
- 요청을 사용하여 결과 데이터를 해당 함수의 data 매개변수로 전달

```
1 requests.post('https://httpbin.org/post', data={'key': 'value'})
```

<Response [200]>

데이터를 dict로 보낼 수 있음

```
1 requests.post('https://httpbin.org/post', data=[('key', 'value')])
```

<Response [200]>

데이터를 tuple 목록으로 보낼 수도 있음

메시지 본문 (Cont.)

- JSON 데이터의 경우 json 매개변수 이용
- Content-Type 헤더에 '**application/json**' 이 추가됨

```
1 response = requests.post('https://httpbin.org/post',
                           json={'key': 'value'})
2
3 json_response = response.json()
4 json_response['data']
```

```
'{"key": "value"}'
```

```
1 json_response['headers']['Content-Type']
```

```
'application/json'
```

Request 검사하기

- **.request** 속성에 액세스하여 **PreparedRequest**를 볼 수 있음

```
1 response = requests.post('https://httpbin.org/post',
2                           json={'key': 'value'})
```

```
1 response.request.headers['Content-Type']
```

'application/json'

```
1 response.request.url
```

'https://httpbin.org/post'

```
1 response.request.body
```

b'{"key": "value"}'

Authentication

- 인증 정보를 전달할 수 있는 **auth** 매개변수 제공

```
1 from getpass import getpass  
2 requests.get('https://api.github.com/user',  
3 auth=('[REDACTED]', getpass()))
```

.....

아이디는 본인의 것으로...

<Response [200]>

```
1 requests.get('https://api.github.com/user')
```

<Response [401]>

자격증명 없이 요청할 경우

SSL 인증서

- SSL 인증서 확인을 사용하지 않으려면 요청 함수의 **verify** 매개변수로 **False**를 전달

```
1 requests.get('https://api.github.com', verify=False)
```

```
C:\Users\user\Anaconda3\lib\site-packages\urllib3\connectionpool.py:847:  
InsecureRequestWarning: Unverified HTTPS request is being made. Adding ce  
rtificate verification is strongly advised. See: https://urllib3.readthed  
ocs.io/en/latest/advanced-usage.html#ssl-warnings  
InsecureRequestWarning)
```

```
<Response [200]>
```

Timeout

■ 요청의 시간 초과를 설정하려면 **timeout** 매개변수 사용

```
1 requests.get('https://api.github.com', timeout=1)
```

<Response [200]>

```
1 import requests
2 from requests.exceptions import Timeout
3
4 try:
5     response = requests.get('https://api.github.com', timeout=1)
6 except Timeout:
7     print('요청 시간을 초과했습니다.')
8 else:
9     print('요청이 정상 처리되었습니다.)
```

프로그램에서 시간 초과 예외를 잡고 이에 따라 응답을 처리할 수 있음

요청이 정상 처리되었습니다.

Session 객체

- 요청 방법에 대한 통제를 세밀하게 조정하거나 요청 성능을 향상시켜야 하는 경우에는 **Session** 객체를 직접 사용

```
1 import requests
2 from getpass import getpass
3
4 with requests.Session() as session:
5     session.auth = ('hjk7902', getpass())
6     response = session.get('https://api.github.com/user')
7
8 print(response.headers)
9 print(response.json())
```

```
...
{'Date': 'Mon, 22 Jul 2019 02:55:02 GMT', 'Content-Type': 'application/json; charset=utf-8', 'Transfer-Encoding': 'chunked', 'Server': 'GitHub.com', 'Status': '200 OK', 'X-RateLimit-Limit': '5000', 'X-RateLimit-Remaining': '4999', 'X-RateLimit-Reset': '1563767702', 'Cache-Control': 'private, max-age=60, s-maxage=60', 'Vary': 'Accept, Authorization, Cookie, X-GitHub-OTP, Accept-Encoding', 'ETag': 'W/"f6c828a183c7dc0bb6fd6b55c18503a'}
```

재시도 제한

- 전송 어댑터를 만들고 **max_retries** 매개변수를 설정한 다음 기존 세션에 마운트

```
1 import requests
2 from requests.adapters import HTTPAdapter
3 from requests.exceptions import ConnectionError
4
5 github_adapter = HTTPAdapter(max_retries=3)
6
7 session = requests.Session()
8
9 session.mount('https://api.github.com', github_adapter)
10
11 try:
12     session.get('https://api.github.com')
13 except ConnectionError as ce:
14     print(ce)
```

requests Exercise. 환율 정보 가져오기

```
1 import requests  
2 from bs4 import BeautifulSoup
```

```
1 url = 'https://finance.naver.com/marketindex/' # 환율 정보  
2 market_index = requests.get(url)
```

```
1 market_index
```

```
<Response [200]>
```

```
1 soup = BeautifulSoup(market_index.content, "html.parser")
```

```
1 price = soup.select_one("div.head_info > span.value")  
2 print(price)
```

```
<span class="value">1,189.70</span>
```

```
1 print("usd/krw=", price.text)
```

```
usd/krw= 1,189.70
```

requests Exercise. 영화 랭킹 출력하기

```
1 import requests  
2 from bs4 import BeautifulSoup
```

```
1 url = 'https://movie.naver.com/movie/sdb/rank/rmovie.nhn'  
2 movie_ranking = requests.get(url)  
3 movie_ranking
```

<Response [200]>

```
1 soup = BeautifulSoup(movie_ranking.content, 'html.parser')
```

```
1 title_list = soup.select('div.tit3 > a')  
2 title_list[0]
```

악인전

```
1 for i, title in enumerate(title_list):  
2     print('{}위: {}'.format(i+1, title.text))
```



Lab. Using requests Module



Beautiful Soup

- Beautiful Soup은 Screen-scraping Project를 위해 설계된 Python Library.
- 구문 분석, 트리 탐색, 검색 및 수정을 위한 몇 가지 간단한 방법과 Python 관용 구를 제공하며 문서를 분석하고 필요한 것을 추출하는 도구
- 들어오는 문서를 Unicode로 보내고 문서를 UTF-8로 자동 변환
- Beautiful Soup은 **lxml** 및 **html5lib**과 같은 Python Parser Library를 사용할 수 있음
 - 이를 이용하면 속도를 개선시키거나 호환성이 높은 응용프로그램을 만들 수 있음
- 공식 사이트
 - <https://www.crummy.com/software/BeautifulSoup/>
- Documentation
 - <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

Parser Libraries

Parser	사용법	장점	단점
Python의 html.parser	<code>BeautifulSoup(markup,"html.parser")</code>	<ul style="list-style-type: none">보통 속도Python 2.7.3, 3.2.2 이상 버전에서 호환됨	<ul style="list-style-type: none">Python 2.7.3 또는 3.2.2 이전버전에서 호환되지 않음
lxml's HTML parser	<code>BeautifulSoup(markup,"lxml")</code>	<ul style="list-style-type: none">매우 빠름호환성	<ul style="list-style-type: none">외부 C에 의존함
lxml's XML parser	<code>BeautifulSoup(markup,"lxml-xml")</code> <code>BeautifulSoup(markup, "xml")</code>	<ul style="list-style-type: none">매우 빠름	<ul style="list-style-type: none">XML Parser만 지원외부 C에 의존함
html5lib	<code>BeautifulSoup(markup,"html5lib")</code>	<ul style="list-style-type: none">호환성이 높음Web Browser와 같은 방식으로 page parsing유효한 HTML5를 만들	<ul style="list-style-type: none">매우 느림

Selector API

- Beautiful Soup은 가장 일반적으로 사용되는 CSS 선택자 지원
- Beautiful Soup의 Selector API는 `select()` 와 `select_one()`, `find()` 등
- `select()` 와 `select_one()` 만 알아도 원하는 요소를 찾기에 충분
- `soup.select("CSS 선택자")`
 - CSS 선택자에 해당하는 모든 요소를 반환
- `soup.select_one("CSS 선택자")`
 - CSS 선택자에 맞는 오직 첫 번째 태그 요소만 반환
- <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

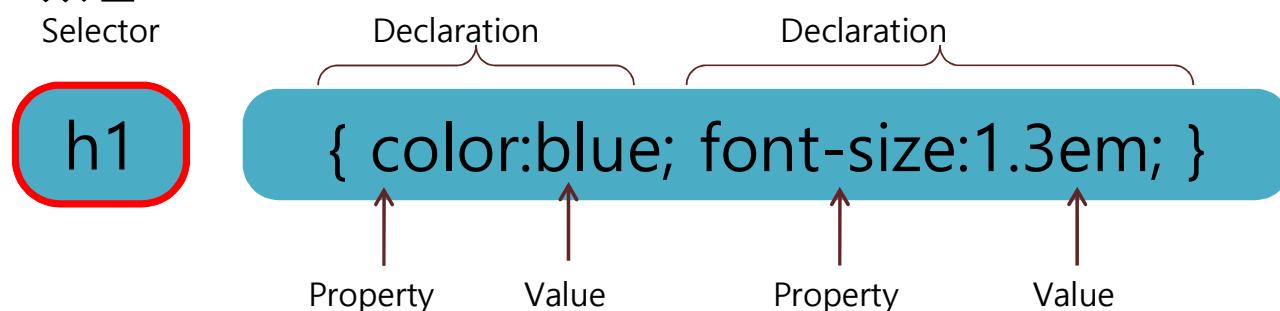
CSS Selector

■ CSS(Cascading Style Sheet)

- CSS는 문서의 contents와 layout, 글꼴 및 시각적 요소들로 표현되는 문서의 외관(design)을 분리하기 위한 목적으로 만들어졌음

■ CSS 선택자

- CSS를 이용해서 HTML 문서에 시각적 요소를 부여할 때 문서 내의 어느 요소에 부여할지를 결정하기 위해 사용하는 것
- CSS 선택자(Selector)는 HTML 문서의 태그 이름, class 속성, id 속성 등을 이용해서 작성할 수 있음



실습을 위한 준비

```
1 import requests  
2 from requests_file import FileAdapter  
3  
4 s = requests.Session()  
5 s.mount('file://', FileAdapter())
```

```
1 res = s.get('file:///sample.html')
```

```
1 res
```

```
1 <html>  
2 <head><title>HTML Sample</title>  
3 </head>  
4 <body>  
5   <h1>Hello CSS</h1>  
6   <div id="subject">선택자</div>  
7   <div class="contents">선택자를 어떻게 작성하느냐에 따라  
8     <span>다른 <b>요소가 반환</b></span> 됩니다.</div>  
9   <div>CSS 선택자는 다양한 곳에서 <b>활용</b>됩니다.</div>  
10 </body>  
11 </html>
```

<Response [200]>

```
1 from bs4 import BeautifulSoup
```

```
1 soup = BeautifulSoup(res.content, "html.parser")
```

```
1 el = soup.select_one("h1")  
2 print(el)
```

<h1>Hello CSS</h1>

선택자 종류

- Tag 선택자 ("element")
 - Tag 선택자는 일반적으로 스타일 정의하고 싶은 html 태그 이름을 사용
 - 요소 안의 텍스트는 **text**, tag 이름은 **name** 그리고 tag의 속성들은 **attrs**를 이용해 조회
 - **soup.select("h1")**
- 다중(그룹) 선택자 ("selector1, selector2, selectorN")
 - 선택자를 ", "(comma)로 분리하여 선언하면 여러 개 선택자 적용.
 - 해당하는 모든 선택자의 요소를 찾기 때문에 **select_one()** 아닌 **select()** 메서드를 이용
 - **soup.select("h1, p")**

선택자 종류 (Cont.)

■ 자손 선택자 ("ancestor descendant")

- 요소가 자손 관계가 있을 때 적용시키기 위한 선택자.
- 선택자와 선택자 사이를 공백으로 띄우고 나열
- `soup.select("div b")`

■ 자식 선택자 ("parent > child")

- 선택자와 선택자 사이에 `>`를 입력하며 반드시 부모자식간의 관계에만 스타일이 적용되도록 함.
- 두 단계 이상 건너뛴 관계에서는 자식 선택자가 작동하지 않음
- `soup.select("div > b")`

선택자 종류 (Cont.)

■ 클래스(class) 선택자 (" . class ")

- HTML 문서에서 class 속성의 값과 일치하는 요소를 선택
- 선택자 이름 앞에 ". "을 이용하여 선언.
- `soup.select("div.contents")`

■ 아이디(id) 선택자 ("# id ")

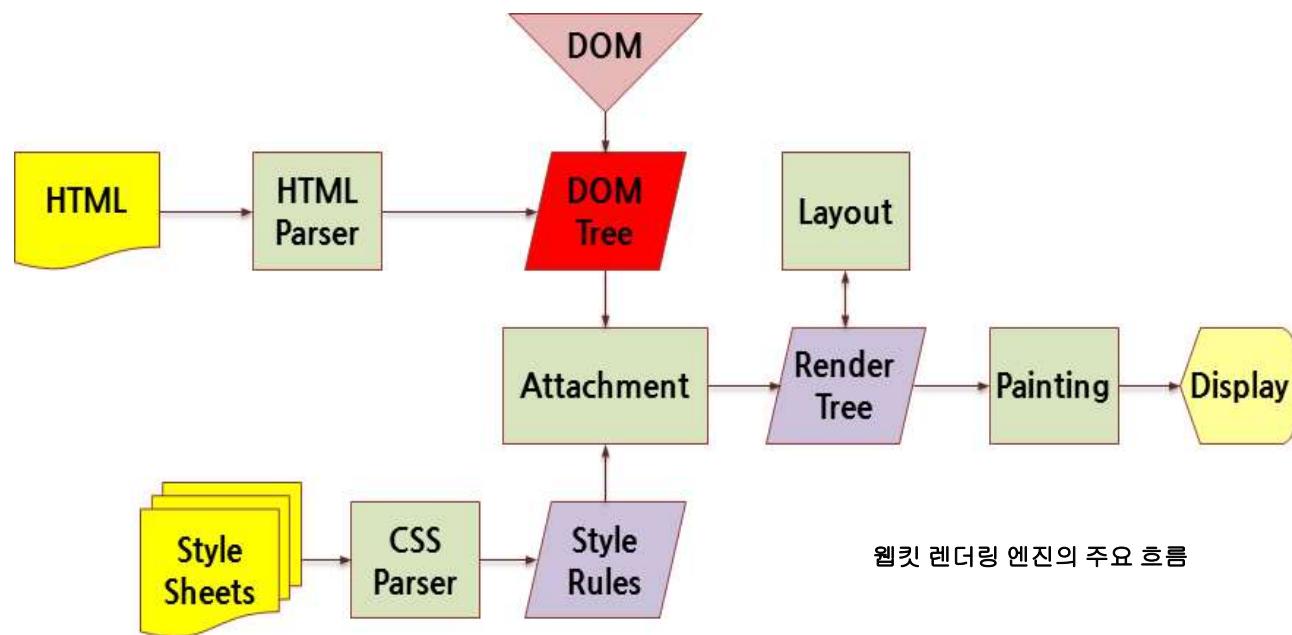
- HTML 문서에서 id 속성의 값과 일치하는 요소를 선택
- id 선택자는 #으로 정의합니다.
- `soup.select_one("#subject")`

■ 속성 선택자 [name="value"]

- 특정한 속성을 갖는 요소만 선택.
- 속성 선택자는 [와] 사이에 속성의 이름과 값을 지정
- `soup.select_one("[id=subject]")`

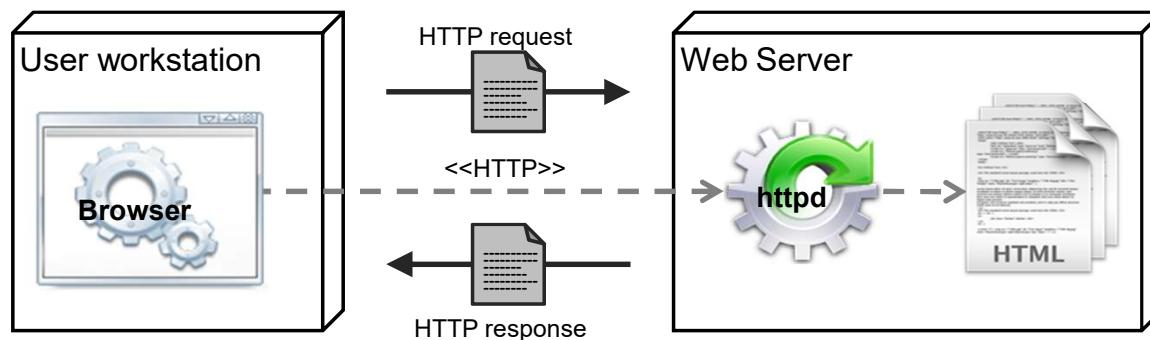
DOM(Document Object Model, 문서 객체 모델)

- HTML 문서를 parsing해서 만들어진 객체
- HTML 문서에서 원하는 요소를 찾기 위해서는 요소를 찾는 메소드를 실행시키기 전에 먼저 DOM이 만들어져 있어야 함



HTTP Request와 HTTP Response

- Browser가 서버의 페이지를 요청하면 서버는 해당 파일을 찾은 다음 HTTP응답을 통해 클라이언트에 전송
- Browser는 응답된 페이지를 해석하여 화면에 보여줌
- request : 사용자가 원하는 파일 또는 리소스의 위치와 브라우저에 관한 정보를 포함
- response : 요청한 자원에 관한 정보를 가지고 있으며, 일반적으로 텍스트 형태이며, 그래픽 등을 바이너리 정보를 포함할 수도 있음.



HTTP Message

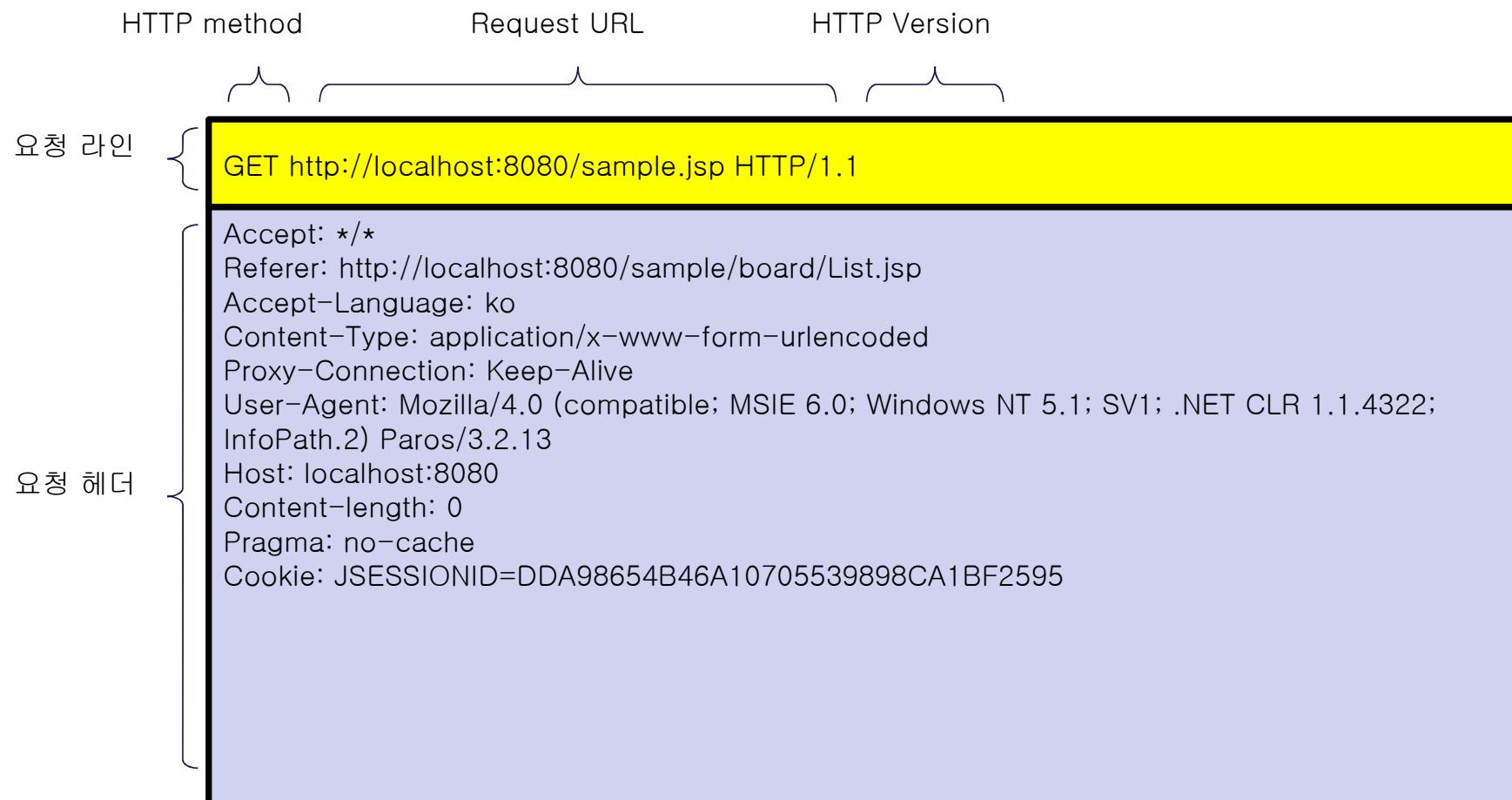
- HTTP Message는 크게 요청과 응답 메시지로 구분
- HTTP Message는 크게 라인, header, body 영역으로 나뉘며, 또 여기서 header와 body 영역은 공백으로 구분됨
- Body 영역을 제외한 나머지 영역(요청, header)의 각 라인은 연속된 CR/LF(\r\n) 문자로 구분



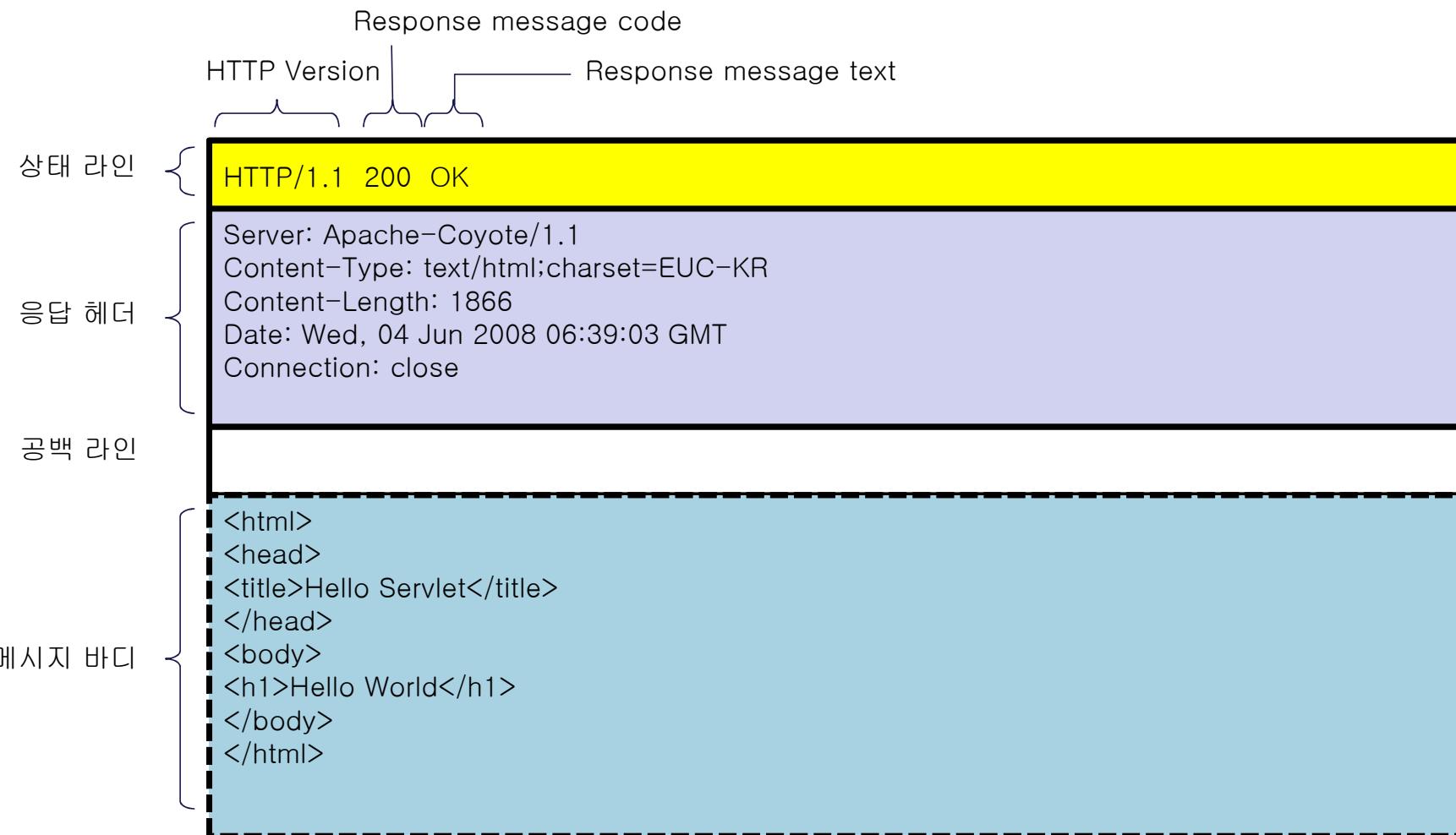
HTTP 요청 방식

요청 방식	설명
OPTIONS	서버 자원 및 기능검색을 위한 요청. 보통 Web 서버가 어떤 Method를 지원하는지 알기 위해 사용
GET	요청하는 주소(URL)에 의해 서버의 자원을 요청. 서버에 전달되는 해당 데이터(파라미터)가 요청 URL에 포함.
HEAD	바디 영역을 제외한 서버 정보를 요청. 요청하는 URI의 자원 유/무를 알아내기 위해 사용.
POST	GET 요청과는 달리 해당 데이터(파라미터)가 요청 URL에 포함되지 않으며 요청 바디 영역에 포함.
PUT	바디 영역의 내용을 실제 서버 URI 위치에 파일로 저장.
DELETE	서버에 저장된 자원을 삭제.
TRACE	요청 자원이 수신되는 경로를 보여줌.
CONNECT	프록시와 같은 중간 서버에 터널을 형성하여 요청하기 위해 사용

HTTP GET Request Message



HTTP Response Message





Lab. Using Beautiful Soup

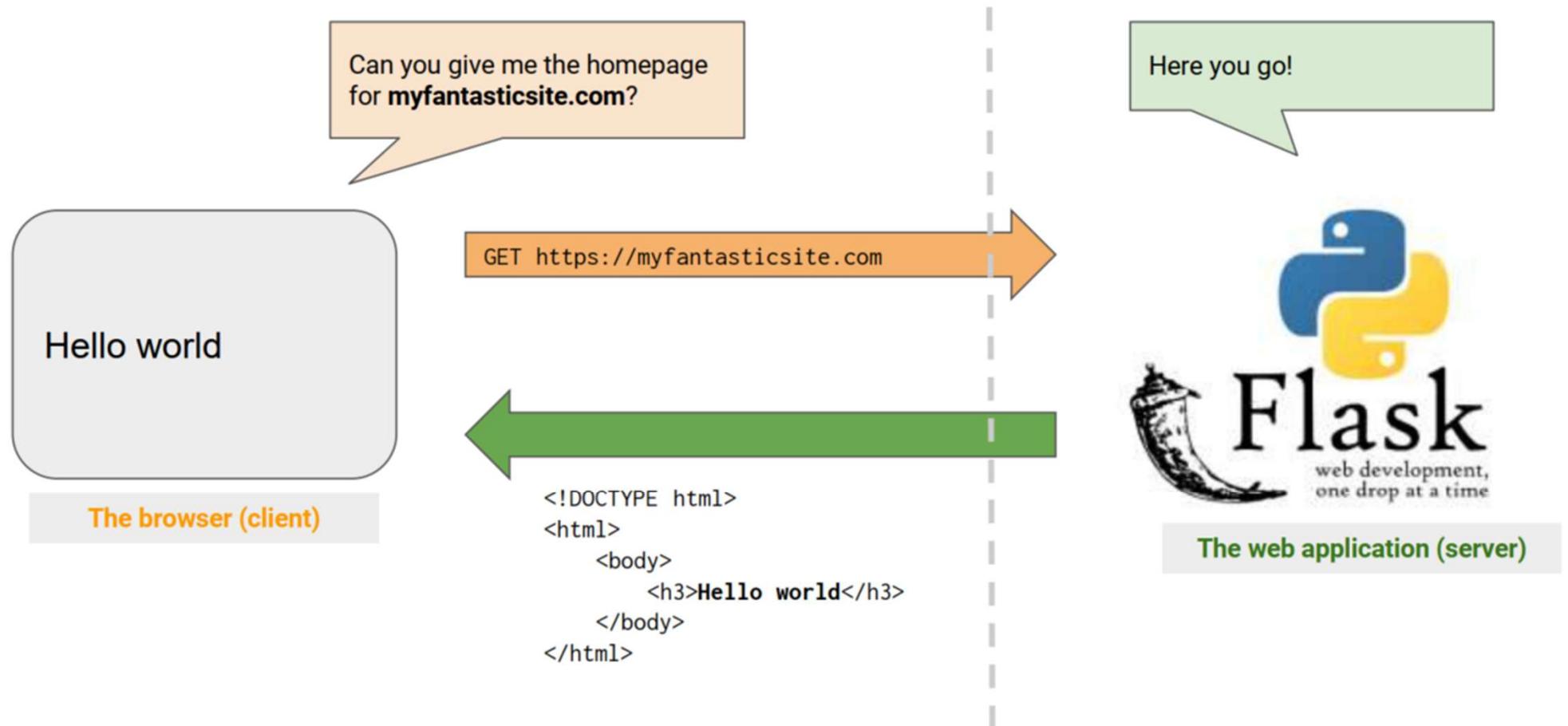


Flask Web Library

- Browser가 서버의 페이지를 요청하면 서버는 해당 파일을 찾은 다음 HTTP응답을 통해 클라이언트에 전송
- Browser는 응답된 페이지를 해석하여 화면에 보여줌
- request : 사용자가 원하는 파일 또는 리소스의 위치와 브라우저에 관한 정보를 포함
- response : 요청한 자원에 관한 정보를 가지고 있으며, 일반적으로 텍스트 형태이며, 그래픽 등을 바이너리 정보를 포함할 수도 있음.



Framework for building web applications in Python



Before Writing a Flask Application



- Install Python
 - Python comes with a package manager called **pip**
- Install *virtualenv*
 - **pip install virtualenv**

```
1 lab1-ubuntu-ec2 × +  
ubuntu@ip-172-16-14-144:~$ sudo apt install python3-venv pip -y  
Reading package lists... Done  
Building dependency tree... Done  
Reading state information... Done  
Note, selecting 'python3-pip' instead of 'pip'  
The following additional packages will be installed:  
  build-essential bzip2 cpp cpp-11 dpkg-dev fakeroot fontconfig-config fo  
  gcc-11-base javascript-common libalgorithm-diff-perl libalgorithm-diff-  
  libatomic1 libc-dev-bin libc-devtools libc6-dev libcc1-0 libcrypt-dev l  
  libfakeroot libfile-fcntllock-perl libfontconfig1 libgcc-11-dev libgd3  
  libjpeg-turbo8 libjpeg8 libjs-jquery libjs-sphinxdoc libjs-underscore l
```

Before Writing a Flask Application (Cont.)



- Create a directory for your web application
 - mkdir `hello`
- Activate `virtualenv` inside your application directory
 - `python -m venv venv`

```
● 1 lab1-ubuntu-ec2 × +  
ubuntu@ip-172-16-14-144:~/WebHome$ python3 -m venv venv  
  
● 1 lab1-ubuntu-ec2 × +  
ubuntu@ip-172-16-14-144:~/WebHome$ source venv/bin/activate  
(venv) ubuntu@ip-172-16-14-144:~/WebHome$ █
```

Before Writing a Flask Application (Cont.)



■ pip install flask

```
1 lab1-ubuntu-ec2 × +  
(venv) ubuntu@ip-172-16-14-144:~/WebHome$ pip install flask  
Collecting flask  
  Downloading Flask-2.2.2-py3-none-any.whl (101 kB)  
    101.5/101.5 KB 2.9 MB/s eta 0:00:00  
Collecting Werkzeug>=2.2.2  
  Downloading Werkzeug-2.2.2-py3-none-any.whl (232 kB)  
    232.7/232.7 KB 11.4 MB/s eta 0:00:00  
Collecting Jinja2>=3.0  
  Downloading Jinja2-3.1.2-py3-none-any.whl (133 kB)  
    133.1/133.1 KB 11.9 MB/s eta 0:00:00  
Collecting click>=8.0  
  Downloading click-8.1.3-py3-none-any.whl (96 kB)  
    96.6/96.6 KB 10.7 MB/s eta 0:00:00  
Collecting itsdangerous>=2.0  
  Downloading itsdangerous-2.1.2-py3-none-any.whl (15 kB)  
Collecting MarkupSafe>=2.0  
  Downloading MarkupSafe-2.1.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (25 kB)  
Installing collected packages: MarkupSafe, itsdangerous, click, Werkzeug, Jinja2, flask  
Successfully installed Jinja2-3.1.2 MarkupSafe-2.1.2 Werkzeug-2.2.2 click-8.1.3 flask-2.2.2 itsdangerous-2.1.2  
(venv) ubuntu@ip-172-16-14-144:~/WebHome$
```

Writing First Flask Application



- Create a file called **main.py** in your project directory.
- ... and edit it in your favorite text editor

```
from flask import Flask

app = Flask(__name__)

app.run(debug=True)
```

Running Flask Application



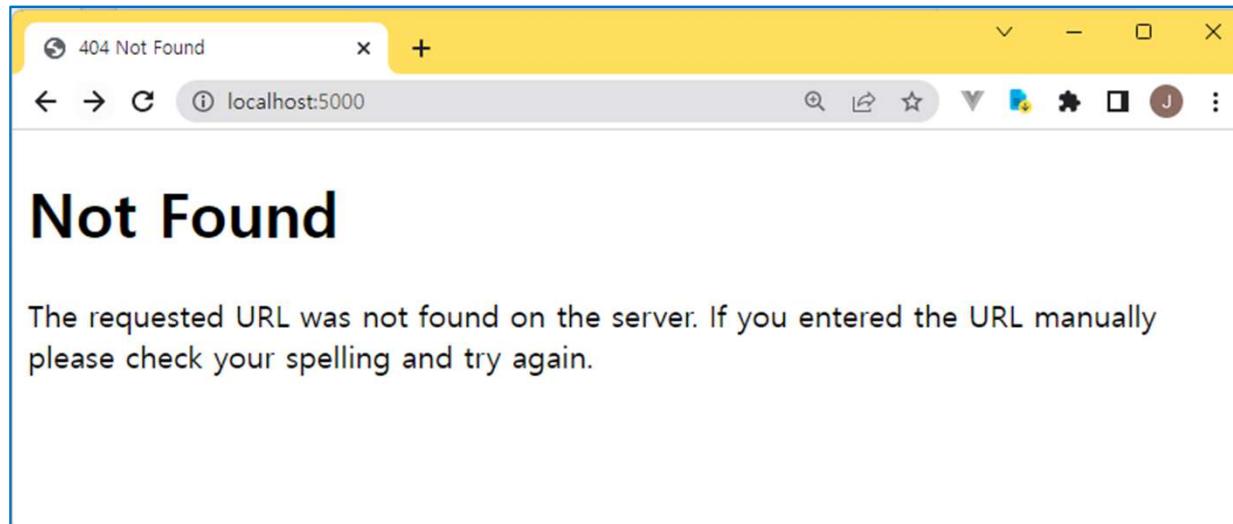
- Run **main.py**, and visit <http://localhost:5000/> in your web browser.
- Note the **/**.
- That's the route

```
● (flaskvenv) ubuntu@ip-172-31-9-79:~/hello$ nano main.py
○ (flaskvenv) ubuntu@ip-172-31-9-79:~/hello$ python main.py
  * Serving Flask app 'main'
  * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
  * Running on http://127.0.0.1:5000
Press CTRL+C to quit
  * Restarting with stat
  * Debugger is active!
  * Debugger PIN: 105-210-353
```

Running Flask Application



- The web server is running at **http://localhost:5000**, but you didn't tell it what to return when someone visits the **/** route.
- To do that, you have to add a function for the **/** route
- Flask uses Python **decorators** to define routes.
- There can be any number of routes in your application.



Defining Routes



C:\cygwin64\home\rohit\tmp\hello\main.py - Notepad++

```
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
main.py X

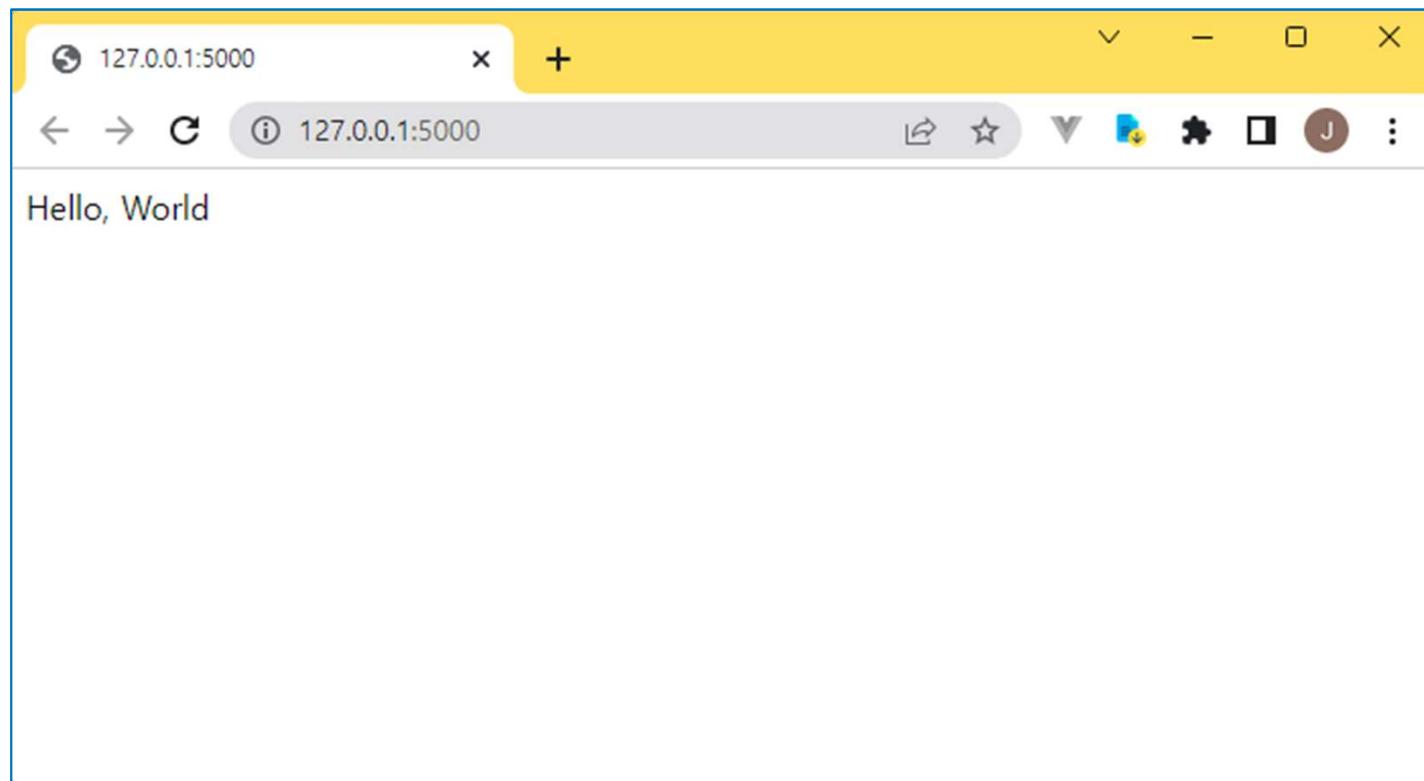
1 from flask import Flask
2
3 app = Flask(__name__)
4
5 @app.route("/")
6 def hello():
7     return "Hello world"
8
9 app.run(debug=True)
10
```

Python file length : 123 lines : 10 Ln : 10 Col : 1 Sel : 0 | 0 Unix (LF) UTF-8 INS

Annotations for the code:

- Line 5: `@app.route("/")` ① When someone visits the / route . . .
- Line 6: `def hello():` ② . . . , run this function . . .
- Line 7: `return "Hello world"` ③ . . . , and return "Hello world"

Defining Routes (Cont.)



Static File & Templates



```
./static:  
images  
  
./static/images:  
flask.png  
  
./templates:  
hello.html
```

```
from flask import Flask, render_template  
  
app = Flask(__name__, static_folder='static', template_folder='templates')  
  
@app.route("/")  
def hello():  
    return render_template('hello.html')  
  
app.run(debug=True)
```

Static File & Templates (Cont.)



Resources



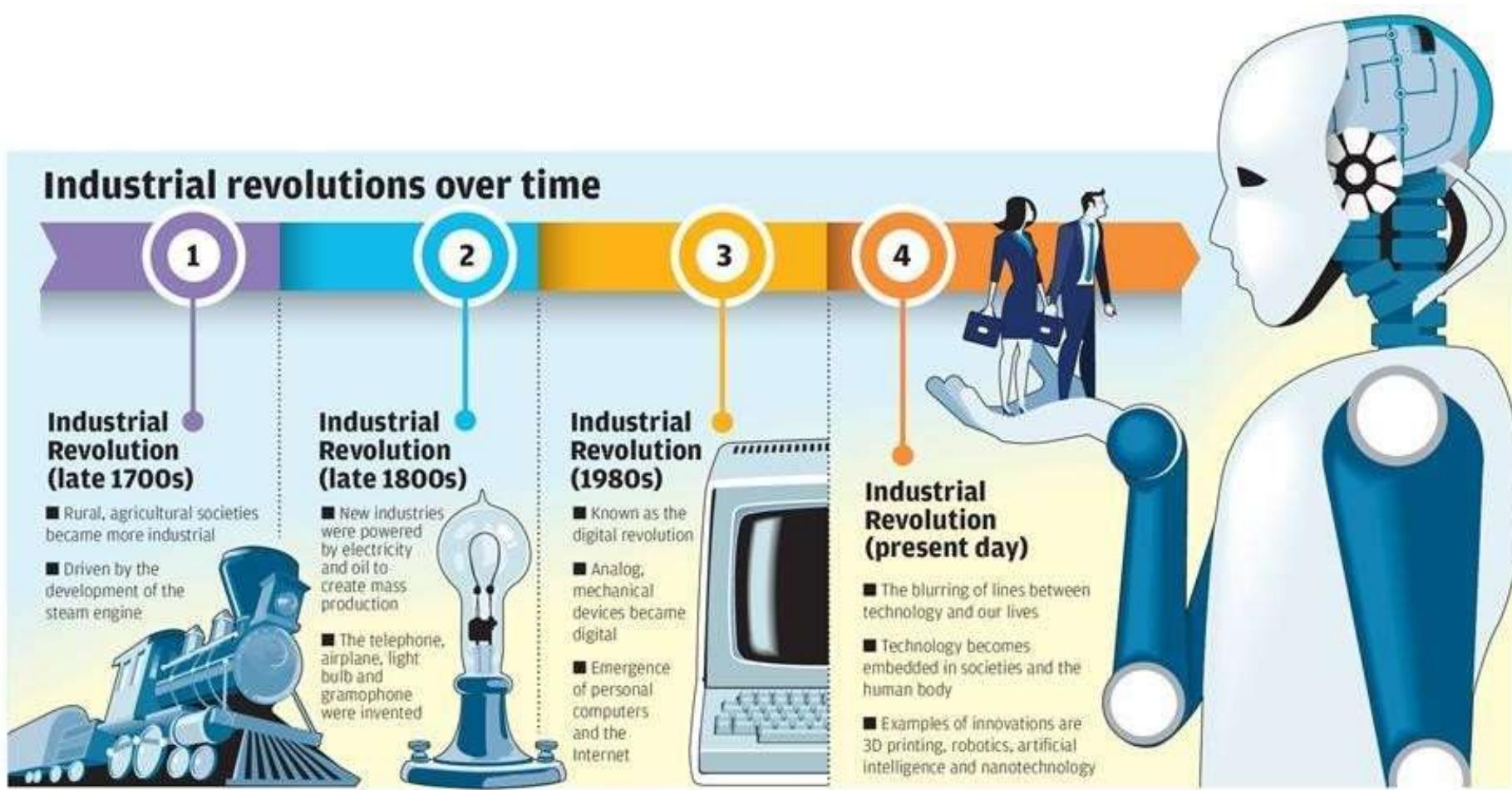
- Flask documentation
 - <https://palletsprojects.com/p/flask/>
- The Flask Mega Tutorial
 - <https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world>
- Explore Flask eBook
 - <https://exploreflask.com/en/latest/>
- More Flask resources
 - <https://www.fullstackpython.com/flask.html>



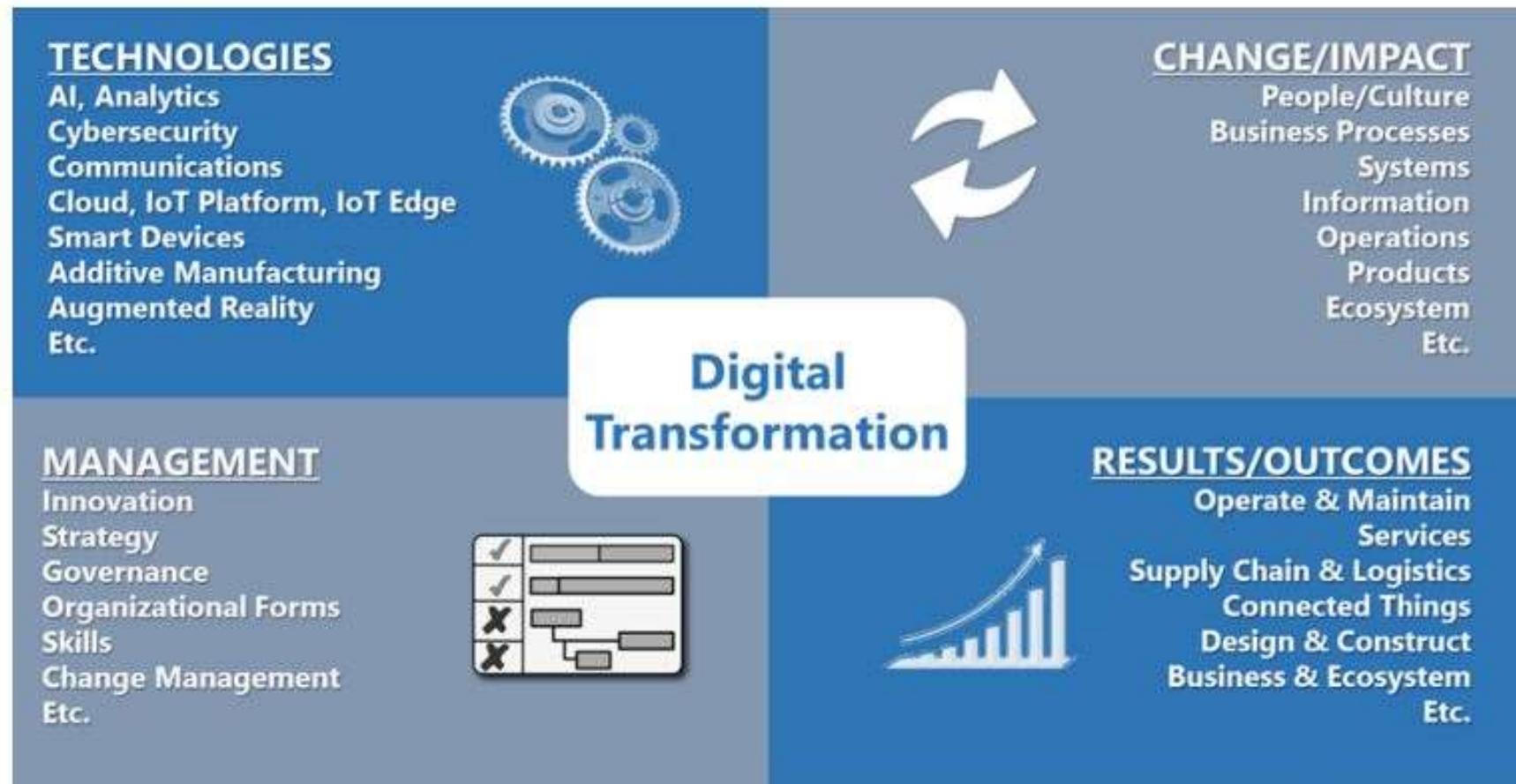
Understanding of Cloud Computing



4th Industrial Revolution



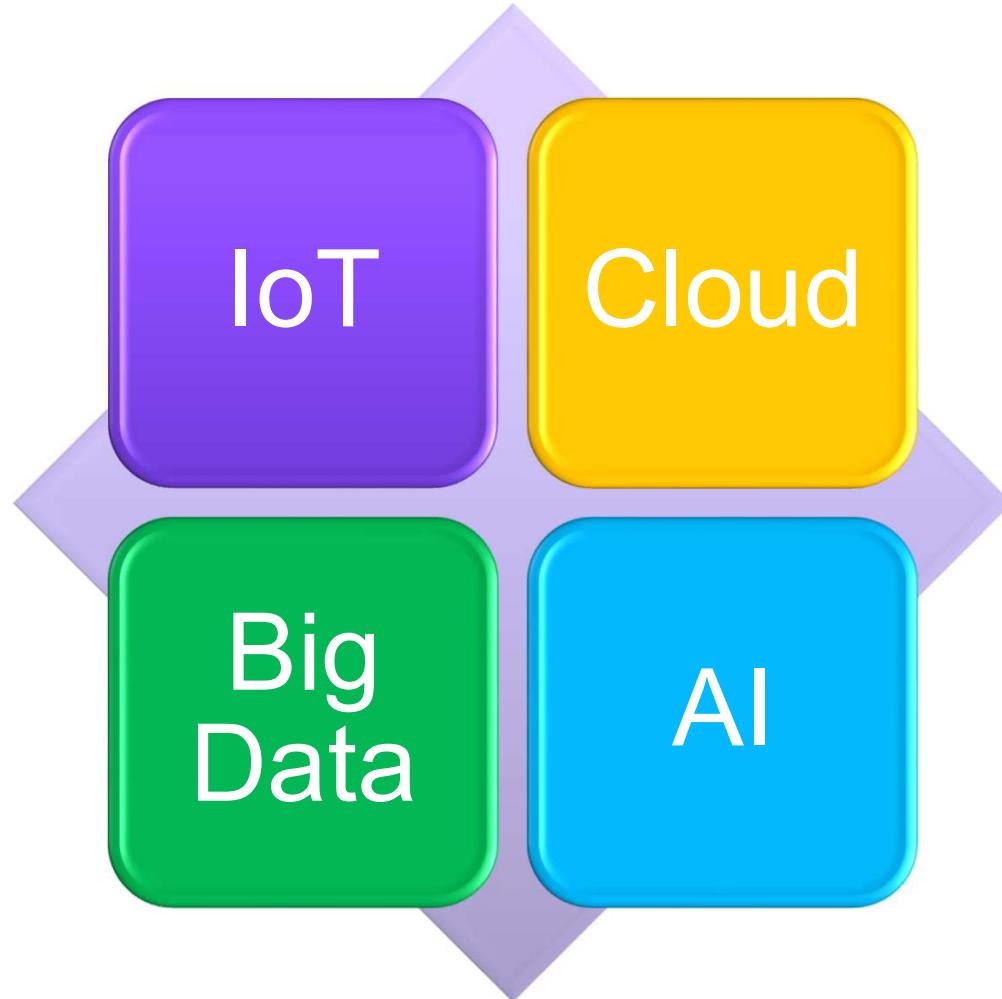
Digital Transformation



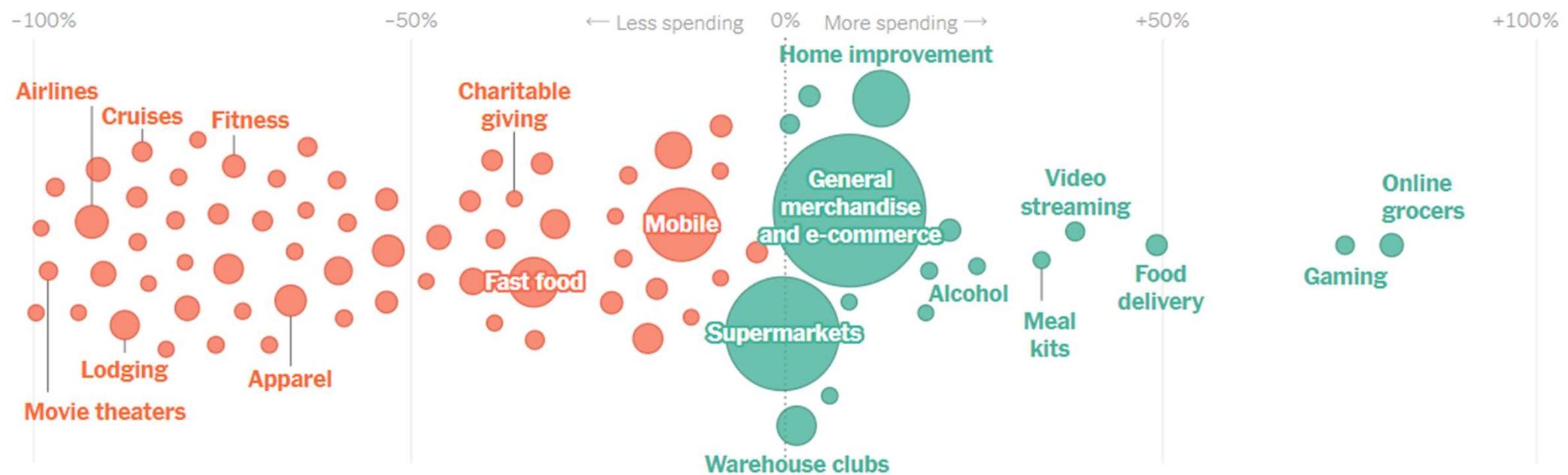
Four Dimensions to Consider When Developing a Digital Transformation Strategy

<https://www.arcweb.com/blog/industrial-digital-transformation-snapshot-strategies-success>

Digital Transformation



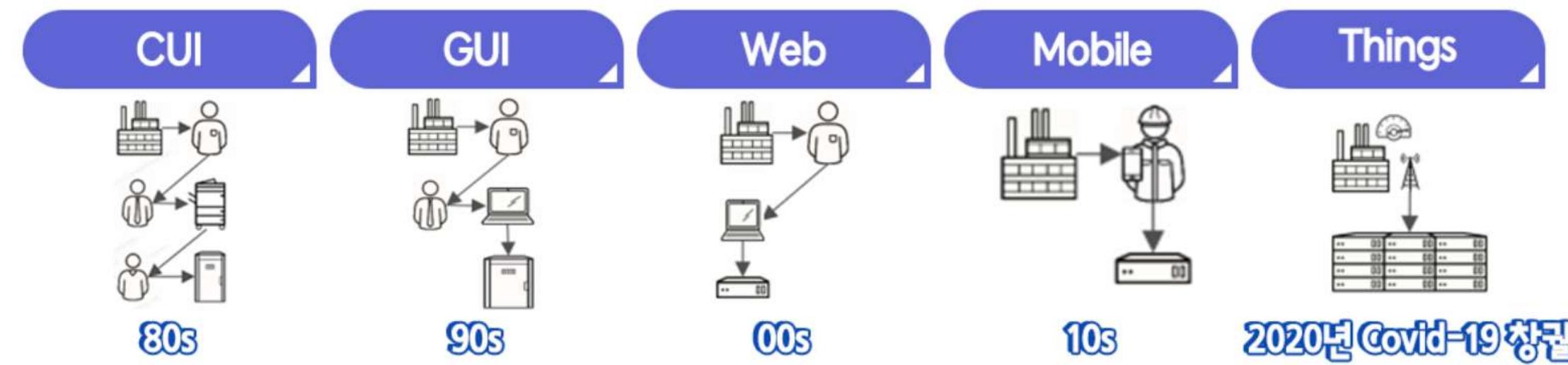
Post Pandemic(COVID-19) and Business Trend



Change in spending from 2019 for the week ending April 1. Bubbles are sized by industry sales.

<https://www.nytimes.com/interactive/2020/04/11/business/economy/coronavirus-us-economy-spending.html>

Post Pandemic(COVID-19) and Business Trend



e-koreatech, "CI/CD를 통한 DevOps 엔지니어링 이해", 2회차

Legacy Enterprise IT Architecture



The Evolution of Data Centers – Enterprise Data Centers(EDCs)

■ Challenges:

- Difficult to build, involves complex Ops management
- Hard to scale and make adjustments, involves long launch cycles

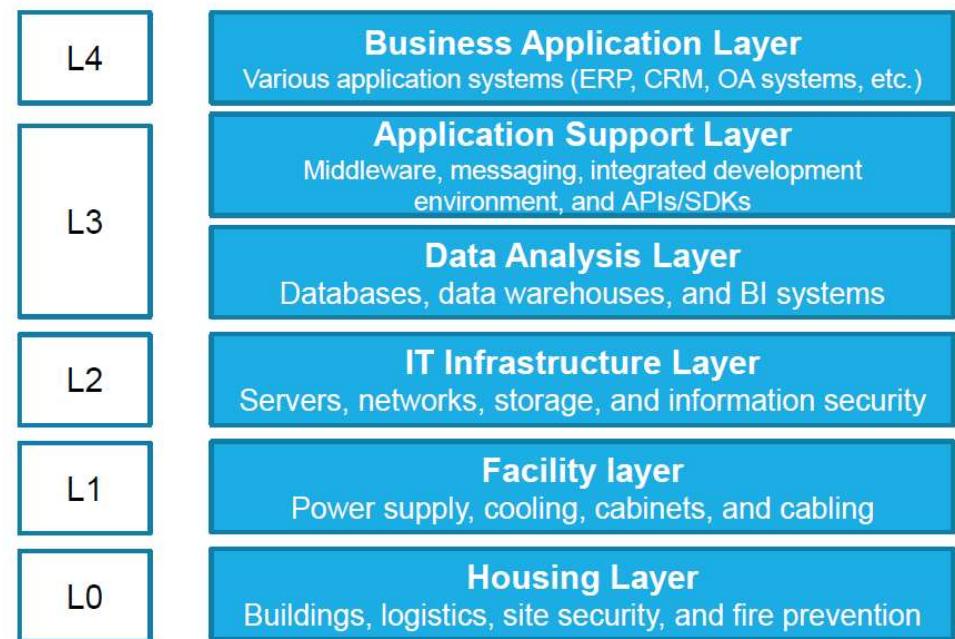
■ High TCO

- $\text{TCO} = \text{CapEx} + \text{OpEx} + \text{OppCost}$

■ Uncertain TVO

- $\text{TVO} = \text{Business value and benefits from IT}$

Data Center Layers: L0-L4



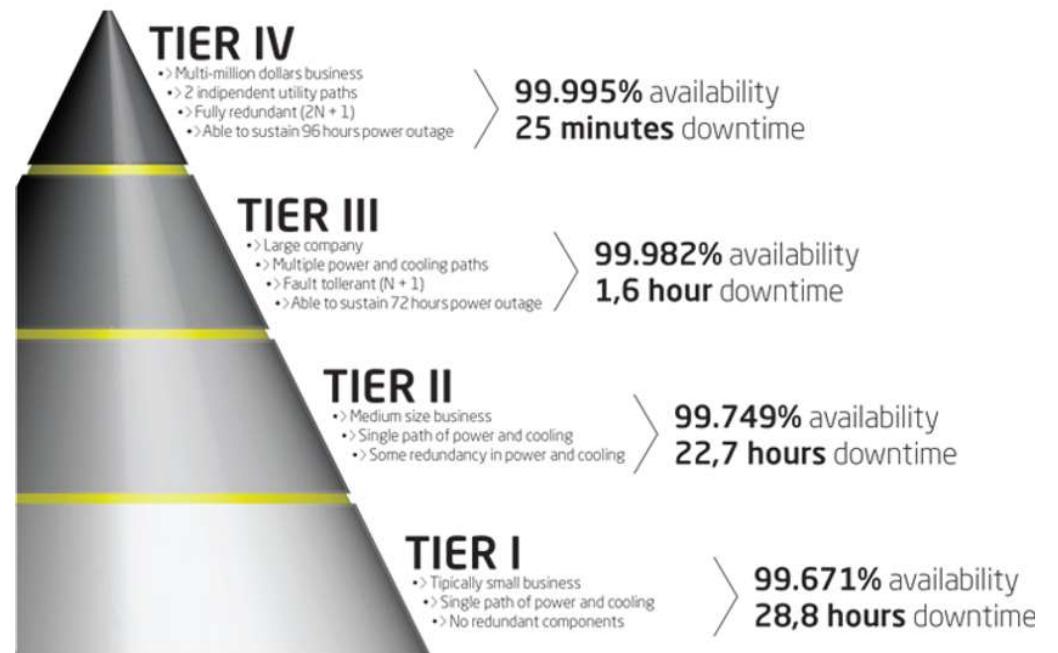
The Evolution of Data Centers – Enterprise Data Centers(EDCs)

■ IDC Tiers : T1 – T4

- Reliability and security
- Ops management capabilities
- Infrastructure availability

$$\text{Availability} = \frac{\text{Promised service time} - \text{downtime}}{\text{Promised service time}} \times 100\%$$

■ Tencent Cloud data centers must be above **T3**.



<https://www.netari.com/post/2014/02/04/what-to-look-for-in-a-data-center-understanding-tier-levels-industry-standards>

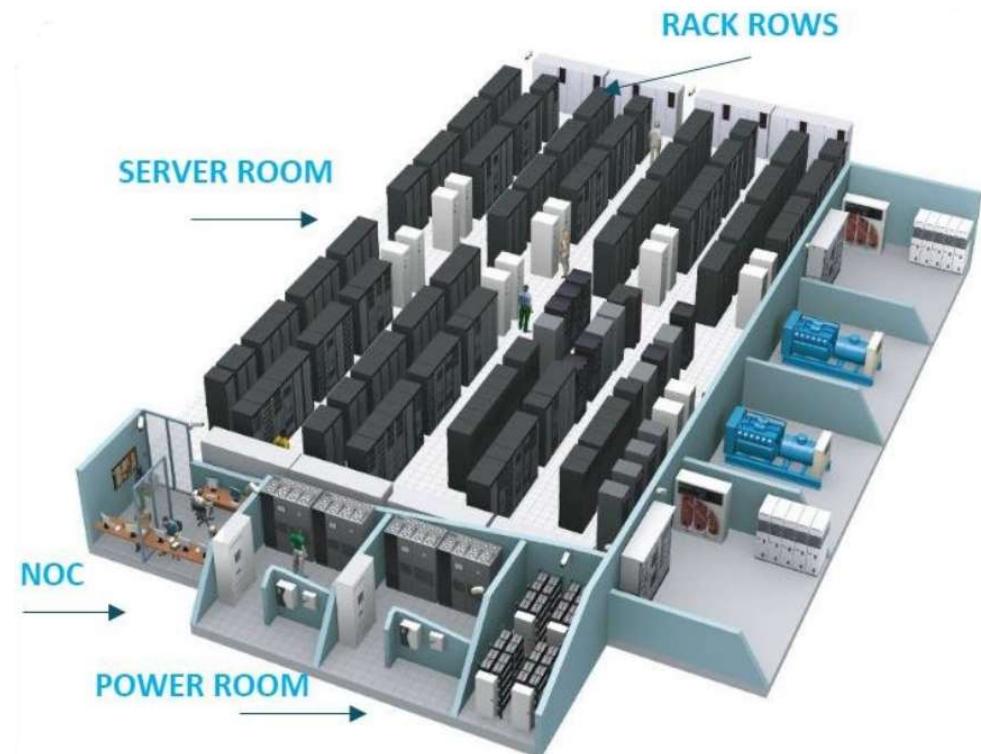
The Evolution of Data Centers – Self-built Internet Data Centers(IDC)

■ Advantages :

- Planning and construction : IDC design, civil engineering...
- Deployment: servers, storage...
- Ops: monitoring, alarming, security Ops...
- Business system deployment: security Ops, availability, and reliability

■ Disadvantage:

- High costs



https://www.researchgate.net/figure/Typical-layout-of-a-Data-center-arranged-by-three-main-areas-server-room-power-room_fig2_343722092

The Evolution of Data Centers – Hosted/Rented IDC

- Two types of leasing of IDC resources such as storage, servers, and bandwidth : hosting and renting
- Advantages of hosted/rented IDCs over self-built EDCs:
 - Lower costs
 - Faster IDC launch
 - Carrier-grade reliability
 - Standardization
 - Ops management

Responsible Entity	Hosted	Rented
The ISP provides:	Facilities, Bandwidth, Power	Facilities, Bandwidth, Power, Hardware, Management, Maintenance
The enterprise provides:	Hardware, Management, Maintenance, Business systems	Business systems

Cloud Computing

- Service on Demand, *Pay-as-you-go*
- Origin : More and more, companies will fulfill their IT requirements simply by purchasing fee-based *Web services* from third parties – similar to the way they currently buy electric power or telecommunications services.
 - 『IT Doesn't Matter』, Nicholas Carr, 2003
- Proposal : in 2006, Google CEO *Eric Schmidt* proposed the concept of cloud computing. AWS was founded in 2006, marking the birth of cloud computing.



Cloud Computing

- Service on Demand, *Pay-as-you-go*
- Origin : More and more, companies will fulfill their IT requirements simply by purchasing fee-based *Web services* from third parties – similar to the way they currently buy electric power or telecommunications services.
 - 『IT Doesn't Matter』, Nicholas Carr, 2003
- Proposal : in 2006, Google CEO *Eric Schmidt* proposed the concept of cloud computing. AWS was founded in 2006, marking the birth of cloud computing.



Cloud Computing (Cont.)



인터넷 기술을 활용하여 다수의 고객들로부터 높은 수준의 확장성을 가진 자원들을 서비스로 제공받는 컴퓨팅의 한 형태



표준화된 IT 기반 기능들이 IP 네트워크를 통해 제공되며, 언제나 접근이 허용되고 수요의 변화에 따라 가변적이며 사용량이나 광고에 기반한 과금 모형을 제공하는 웹 또는 프로그램적인 인터페이스를 제공하는 컴퓨팅

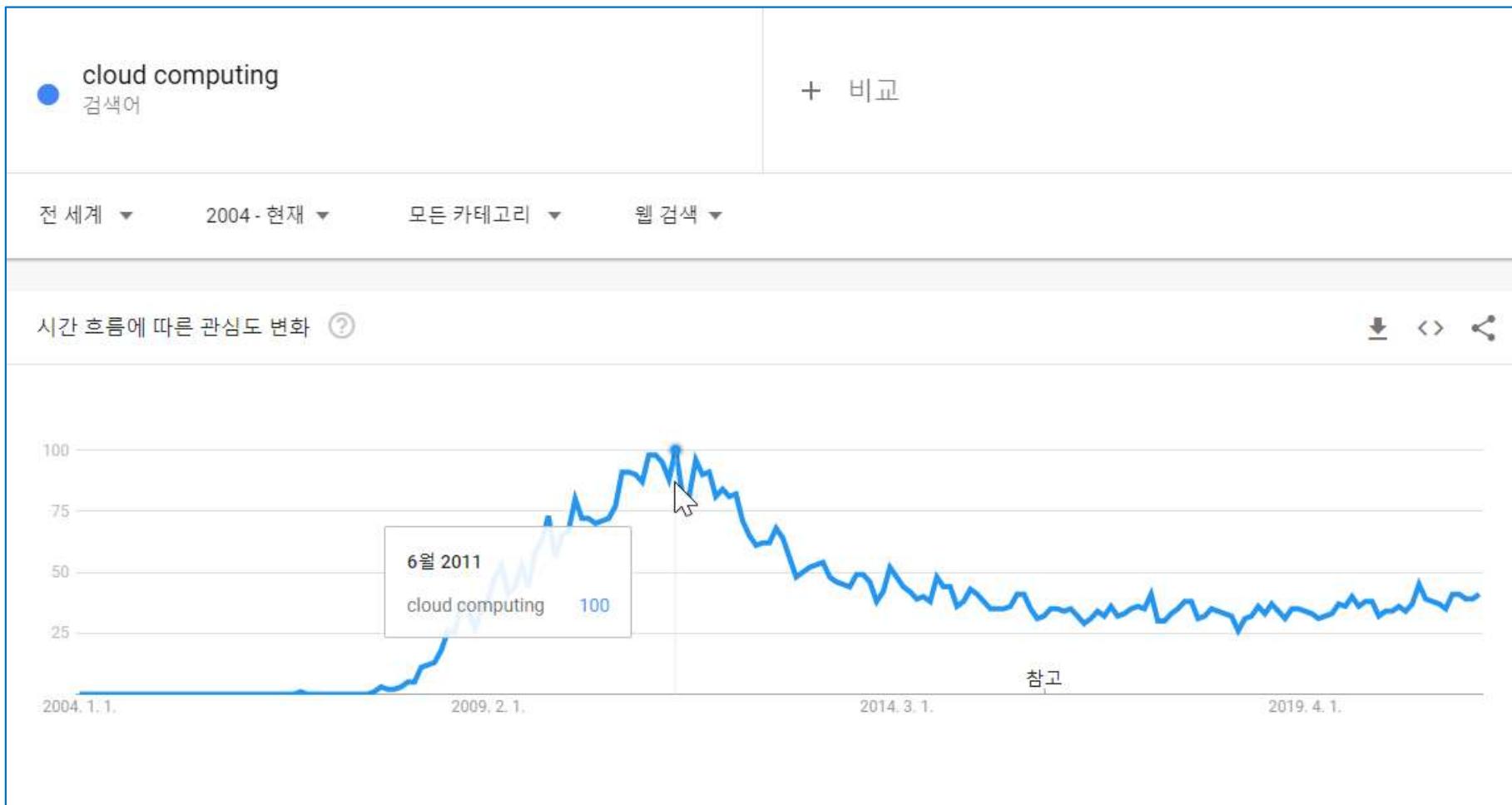


인터넷에 기반한 개발과 컴퓨터 기술의 활용을 뜻하는 것으로 인터넷을 통해서 동적으로 규모화 가능한 가상적 자원들이 제공되는 컴퓨팅



웹 기반 어플리케이션을 활용하여 대용량 데이터베이스를 인터넷 가상 공간에서 분산처리하고 이 데이터를 PC, 휴대 전화, 노트북 PC, PDA 등 다양한 단말기에서 불러오거나 가공할 수 있게 하는 환경

Cloud Computing (Cont.)



<https://trends.google.com/trends/explore?date=all&q=cloud%20computing>

Cloud Computing (Cont.)



Special Publication 800-145

2. The NIST Definition of Cloud Computing

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.

**Recommendations of the National Institute
of Standards and Technology**

Peter Mell
Timothy Grance

Cloud Computing (Cont.)

Features	Description
Massive scale	A public cloud often has hundreds of thousands or even millions of servers; a private cloud can have hundreds to thousands of servers.
High reliability	Multi-replica fault tolerance provides high reliability.
Isolation of tenants	Multiple tenants share the underlying hardware resources, but are logically isolated at the upper layers.
Elastic scaling	Dynamic scaling helps clients cope with the growth of their applications and their user base.
Service on demand	Cloud provides a large pool of resources that clients can purchase on demand.
Monitorable and measurable resources	Cloud platforms provide features for monitoring and measuring resources.
Low costs	Users only pay only for the resources they use, not the entire infrastructure.

Cloud Computing (Cont.)

*Cloud computing is the **on-demand** delivery of **IT resources** over the **Internet** with **pay-as-you-go** pricing. Instead of buying, owning, and maintaining physical data centers and servers, you can access technology services, such as computing power, storage, and databases, on an as-needed basis from a cloud provider like Amazon Web Services (AWS).*

클라우드 컴퓨팅이란 인터넷에서 종량 요금제 방식으로 클라우드 서비스 플랫폼을 통해 컴퓨팅 파워, 데이터베이스, 스토리지, 애플리케이션, 기타 IT 리소스를 온디맨드로 제공하는 서비스를 말합니다.

-Amazon Web Services 개요

Cloud Computing (Cont.)

Item	EDCs	Traditional IDCs	Cloud Computing
Rental Scope	None	L0, L1, and part of L2	L0 – L4
Overall Costs	High	Moderate	Low
Launch Cycle	Long	Moderate	Very short
Ops Management	Complicated	Moderate	Simple
Scalability	Difficult	Moderate	Elastic scaling
Independence and Controllability	High	Moderate	Public cloud / private cloud

Cloud Computing

비대면 /
Online

Mobile / Web

Credit Card

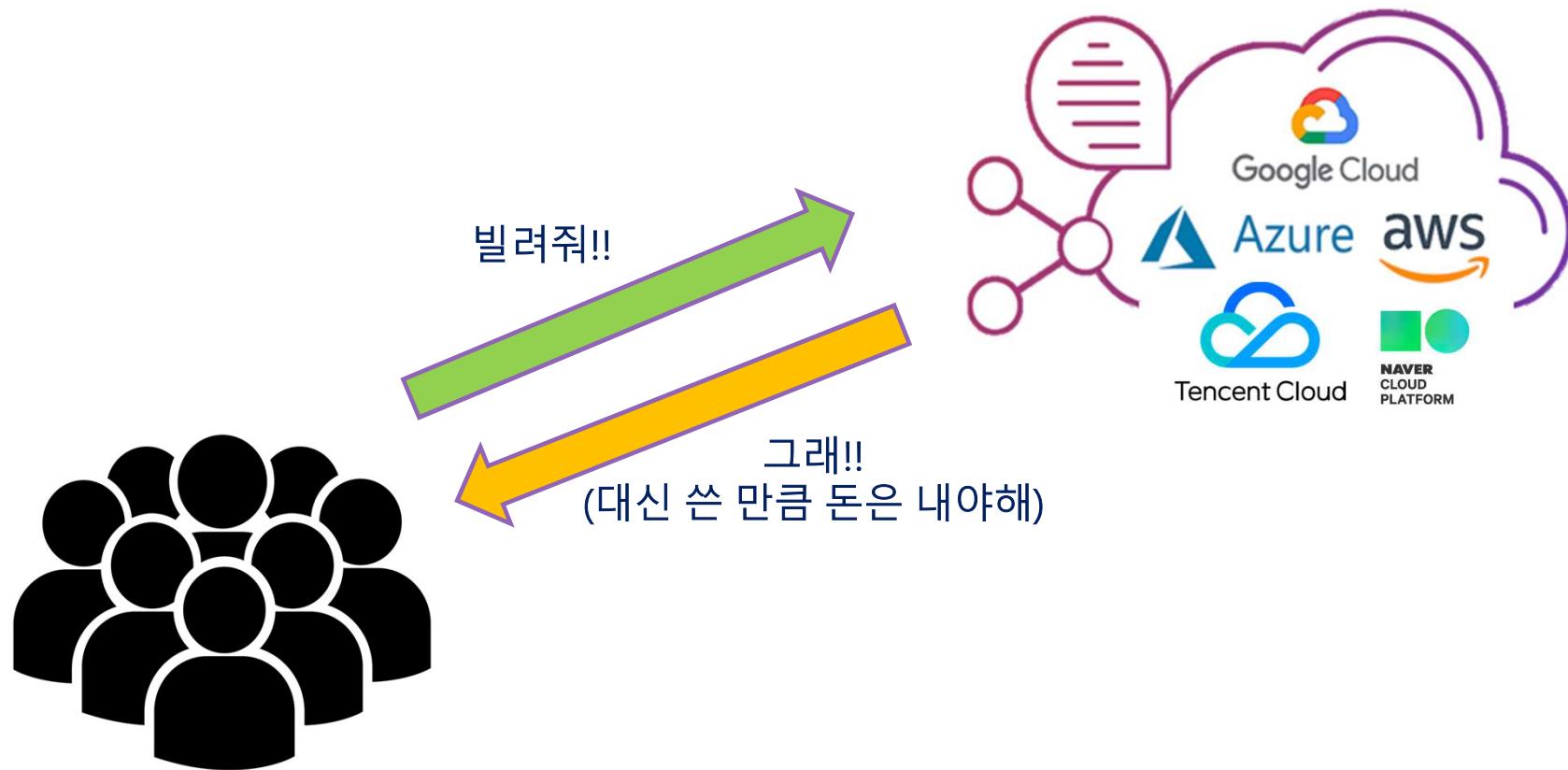
IT 공급 / 주
문 Service

CRM / AI-
based
Service

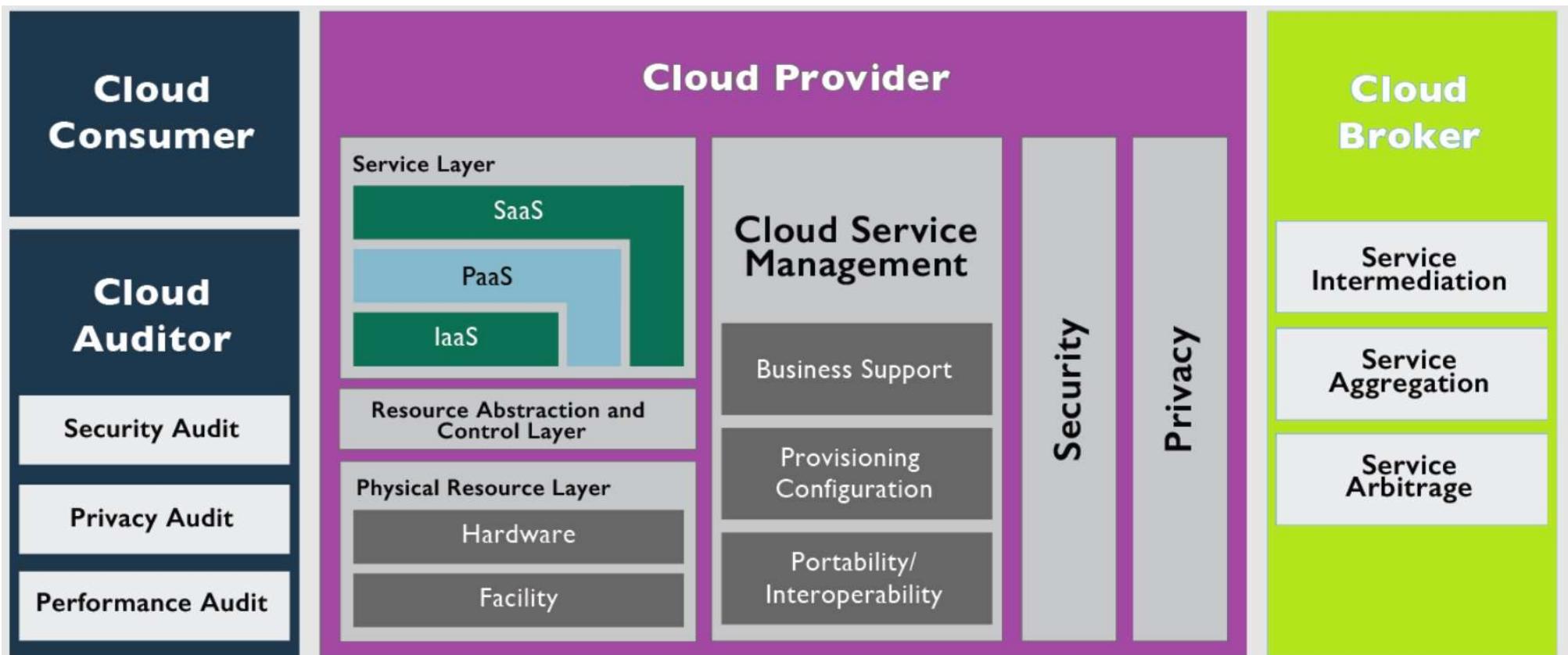
장바구니
System

신속 배송
System

Cloud Computing

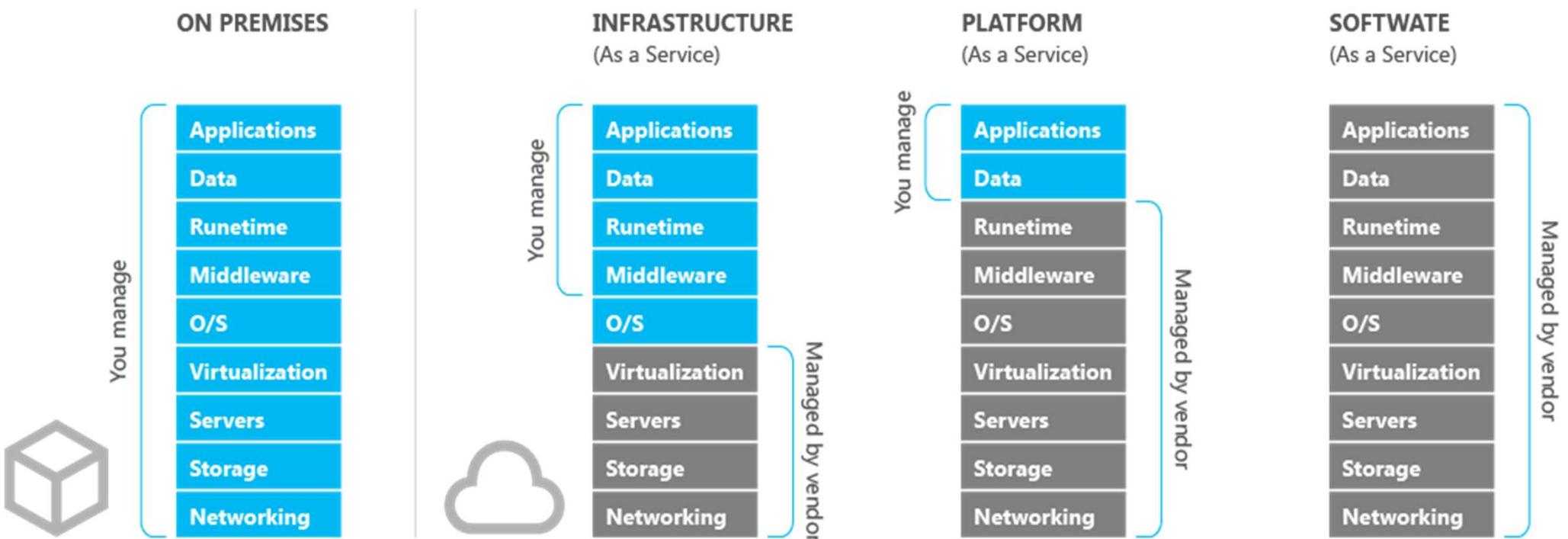


Cloud Computing – Reference Model



<https://cyberrisk-countermeasures.info/2021/12/08/nist-cloud-computing-reference-architecture-and-taxonomy%EF%BF%BC/>

Cloud Computing – Reference Model (Cont.)



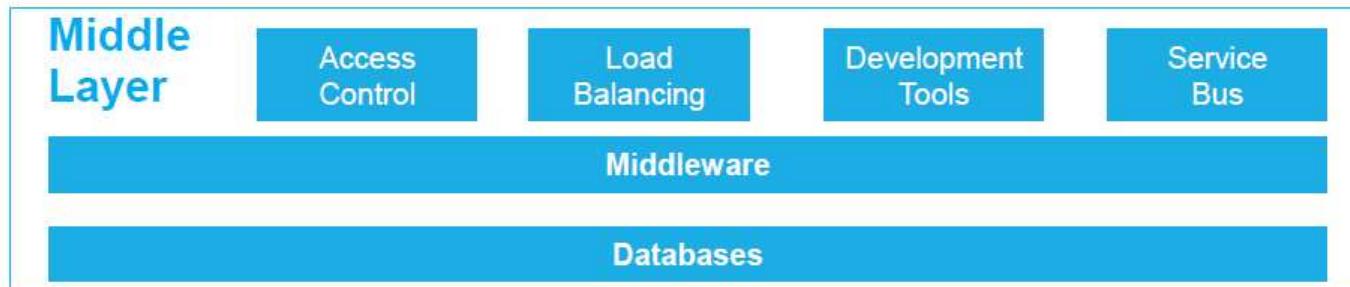
<https://velog.io/@aonee/%ED%81%B4%EB%9D%BC%EC%9A%B0%EB%93%9C%EB%9E%80-nzrij240>

Cloud Computing – Reference Model (Cont.)

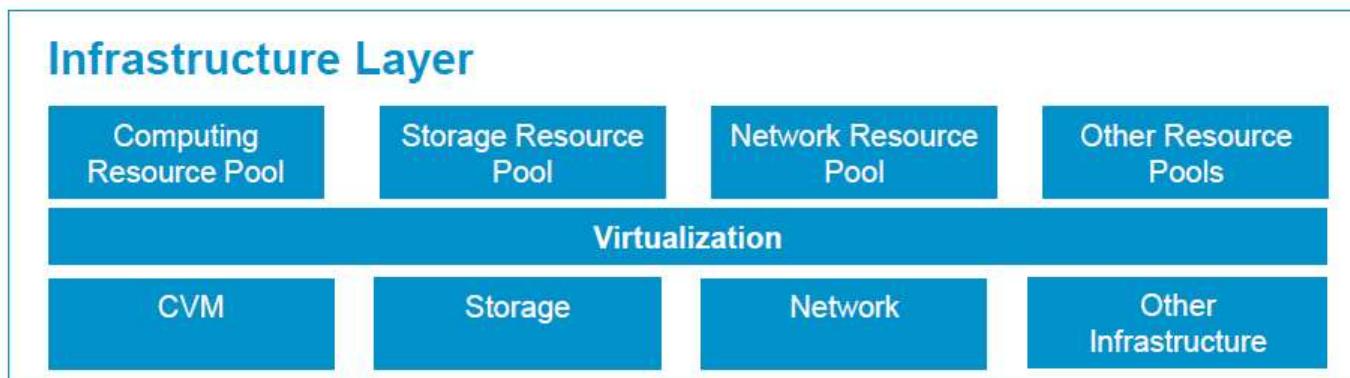
SaaS



PaaS



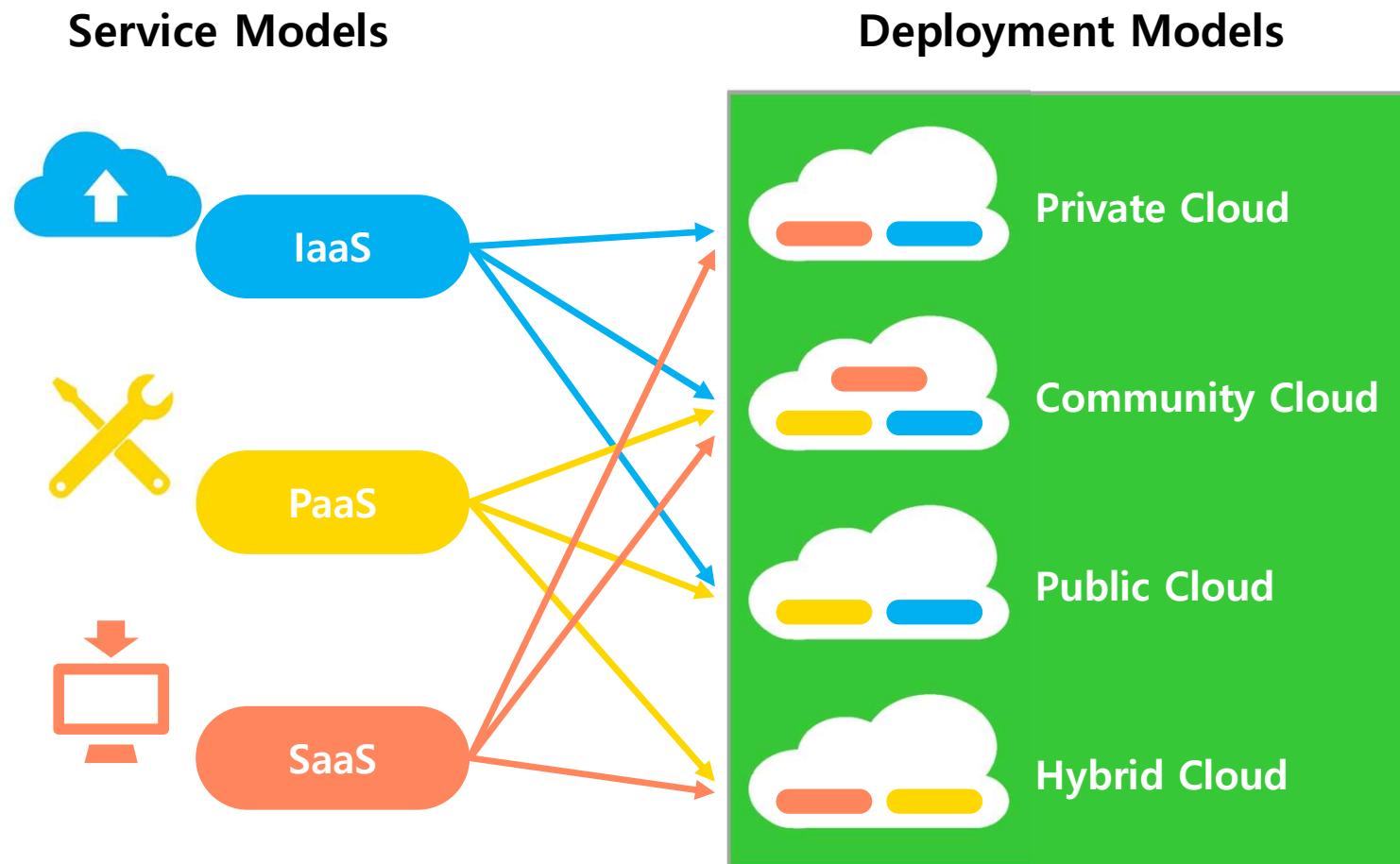
IaaS



Management Layer



Cloud Computing – Reference Model (Cont.)



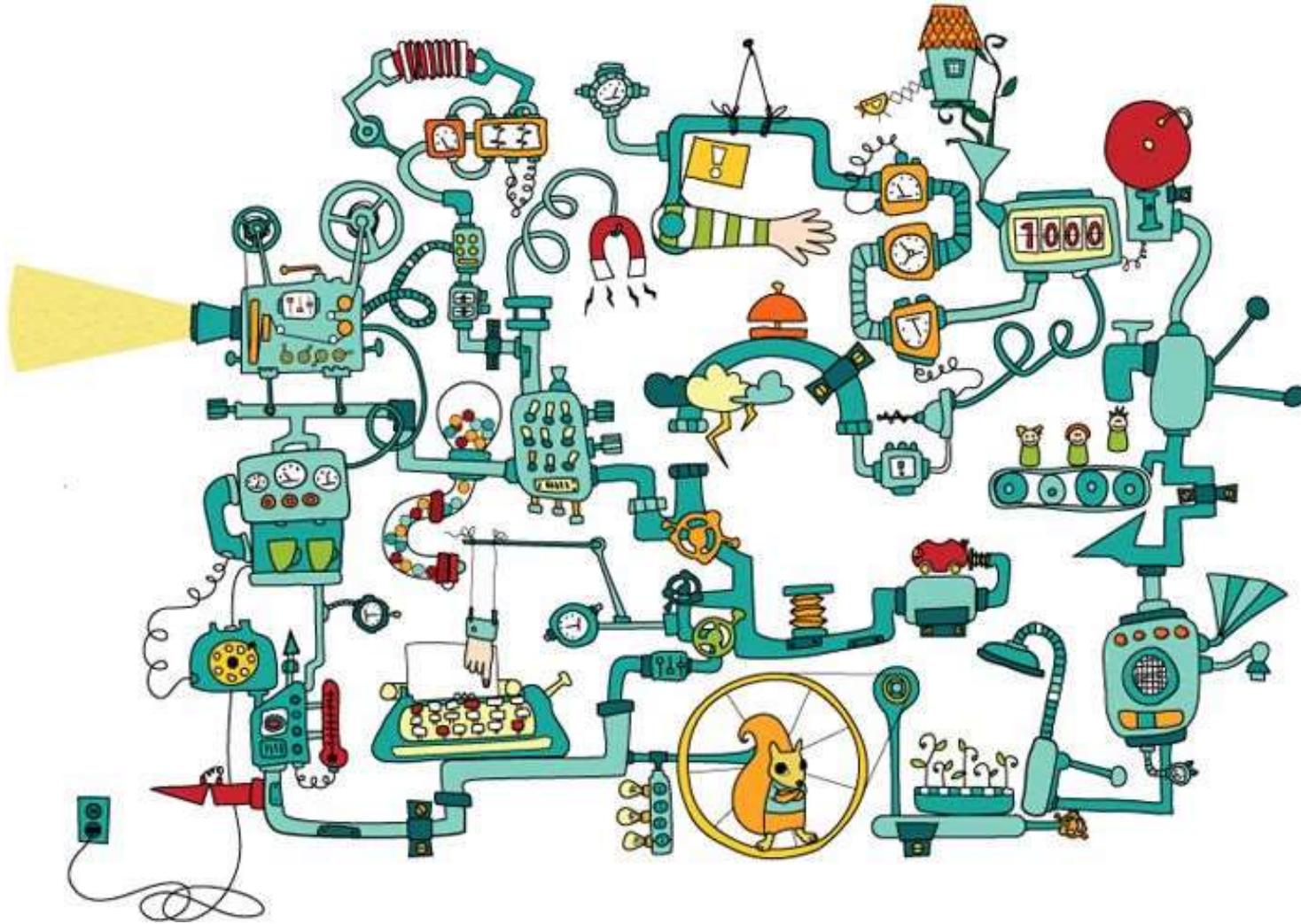
Cloud Computing – Reference Model (Cont.)

구분	장점	단점
퍼블릭 클라우드	<ul style="list-style-type: none">초기 투자비용 없음융통성 있는 사용량 조절	<ul style="list-style-type: none">서비스 제공자 기업의 의존도가 높음
프라이빗 클라우드	<ul style="list-style-type: none">기존 IT 자원을 활용 가능행위추적 용이	<ul style="list-style-type: none">초기 투자비용이 많이 소요
하이브리드 클라우드	<ul style="list-style-type: none">기존 IT 자원을 활용 가능서비스 구성변경 용이	<ul style="list-style-type: none">운용비와 도입비용 증가
커뮤니티 클라우드	<ul style="list-style-type: none">초기 투자비용 없음융통성 있는 사용량 조절	<ul style="list-style-type: none">서비스 제공자 기업의 의존도가 높음
공통정보보호 요구사항	외부에서 내부(클라우드) 시스템 접속이 이루어져 함에 따라 통신구간 암호화, 내부 시스템 보호를 위한 방화벽, 침입방지 시스템 구축 등 주요 보호조치 필요	

Legacy Enterprise IT Architecture

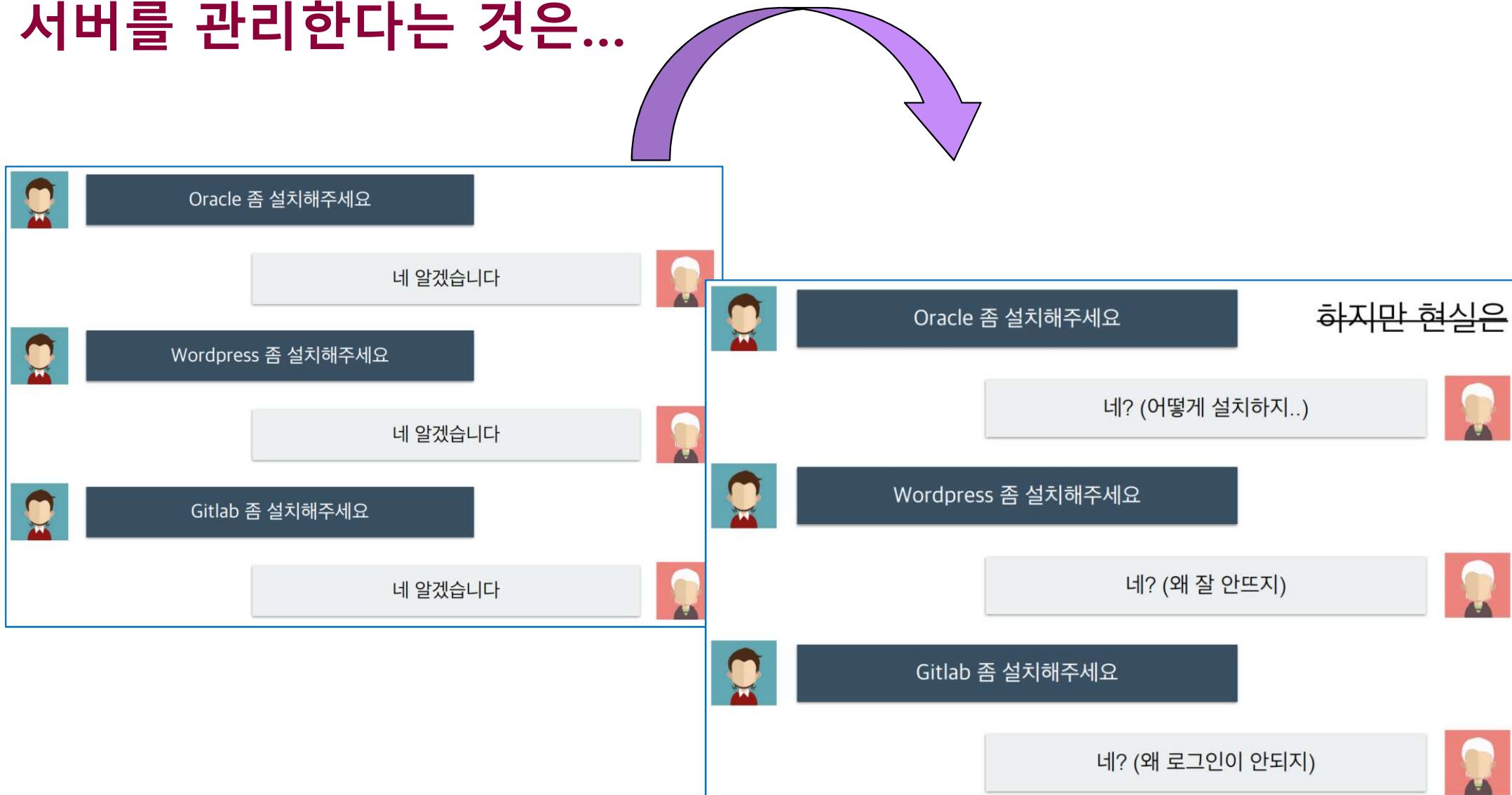


서버를 관리한다는 것은...



<https://subicura.com/2017/01/19/docker-guide-for-beginners-1.html>

서버를 관리한다는 것은...



서버를 관리한다는 것은...



이제 AWS를 쓰기로 했습니다!



이제 Azure를 쓰기로 했습니다!



이제 Google Cloud를 쓰기로 했습니다!

계속해서 바뀌는 **서버 환경** 😕



Node.js를 쓰기로 했습니다!



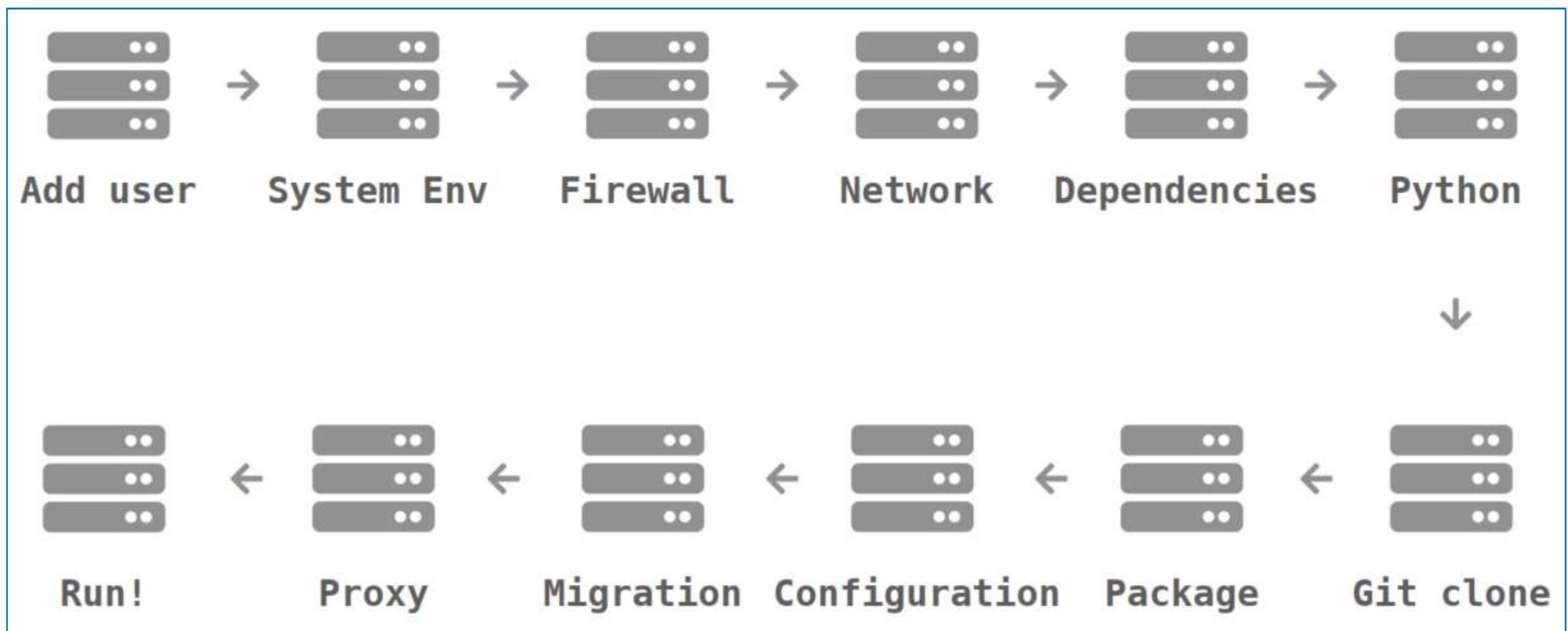
Python을 쓰기로 했습니다!



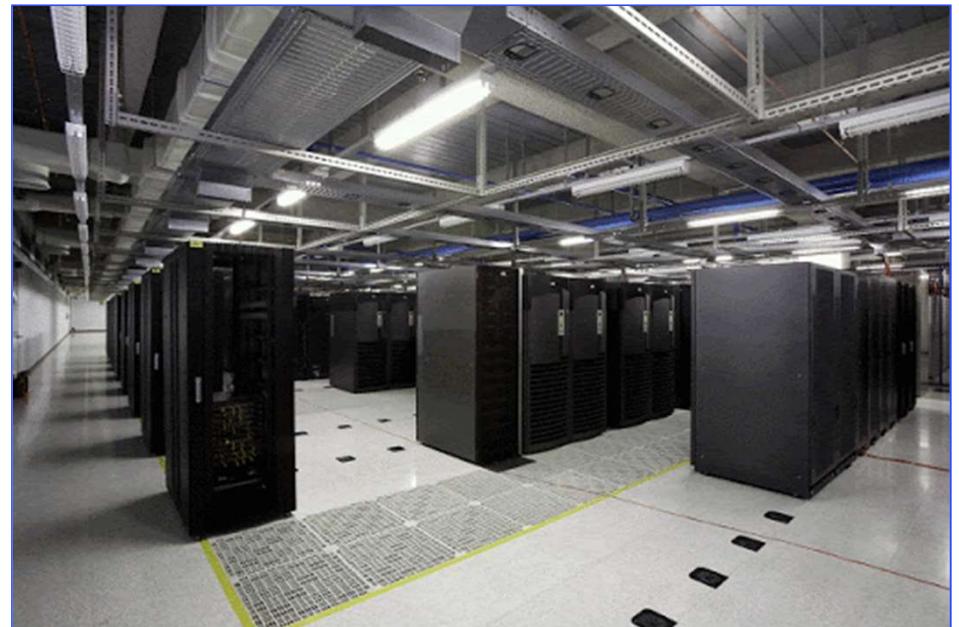
Ruby를 쓰기로 했습니다!

계속해서 바뀌는 **개발 환경** 😱

서버를 관리한다는 것은...



Virtualization

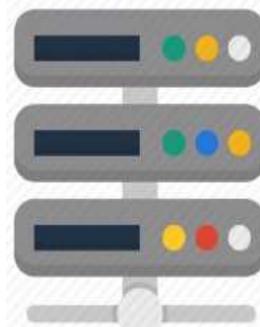


Virtualization

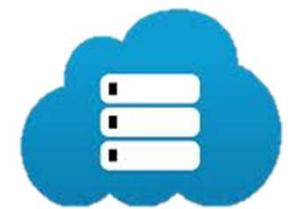
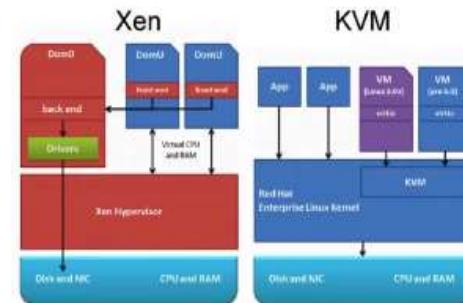
Data Centers



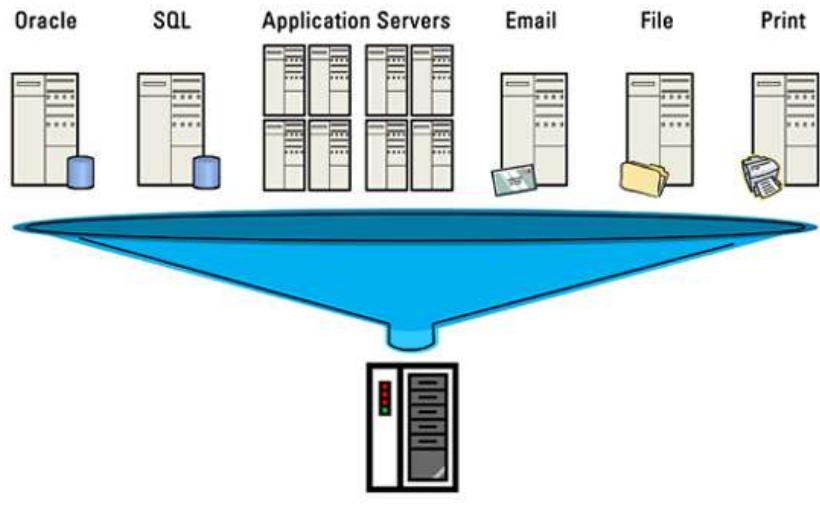
Server Racks



Virtualization



Virtualization



- V12n
- 물리적인 하드웨어 장치**를**
- 논리적인 객체**로**
- 추상화하는 기술
- Is the act of creating a virtual(rather than actual) version of something, including virtual computer hardware platforms, storage devices, and computer network resources.

Container



컨테이너

위키백과, 우리 모두의 백과사전.



다른 뜻에 대해서는 [컨테이너 \(동음이의\)](#) 문서를 참조하십시오.

컨테이너(영어: Container)는 철판으로 만들어져 재사용이 가능한 규격화된 통으로 화물을 옮길 때 쓴다.

1950년대 상용화 되고 그후 점차 널리 쓰게 되었으며, 짐 꾸리기에 편하고 운반이 쉬우며 보관에도 좋은 점 때문에 전 세계적으로 널리 퍼지게 되었다.^[1]

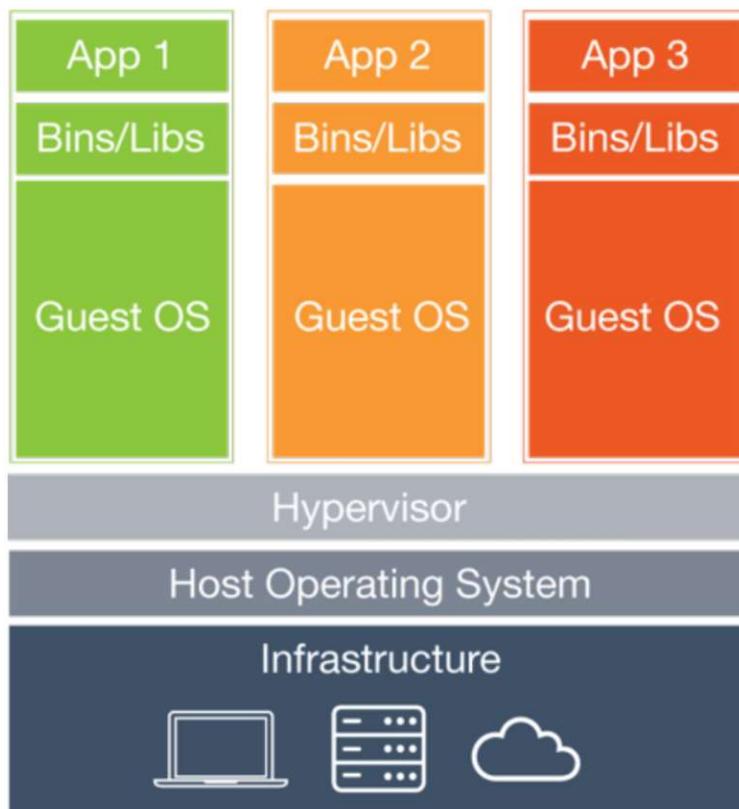
Container



WHY CONTAINERS?

<https://www.youtube.com/watch?v=n-JwAM6XF88>

Container (Cont.)



가상머신이 있는데, 왜 컨테이너인가?

→ 성능차이

CPU, RAM 할당하고, OS 넣고...

성능을 높이는 방법으로...

→ 가상 머신의 일을 줄이자...

가상 하드웨어를 또 생성?

→ 하이퍼바이저 빼자...

OS위에 또 OS?

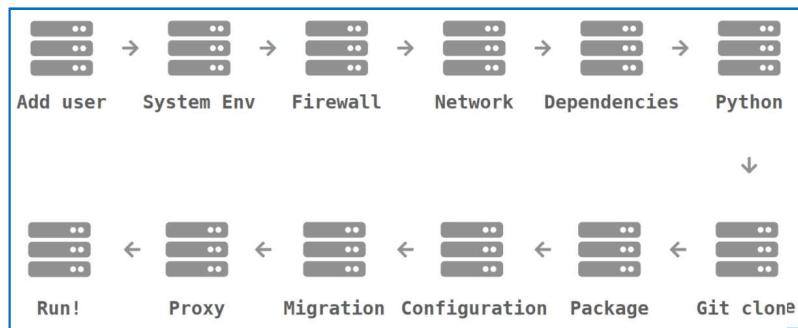
→ Guest OS 빼자...

자원 할당은 OS Kernel이 하자.

→ LXC가 하자.

다시 처음으로 돌아가서, 서버를 관리하는 것은...

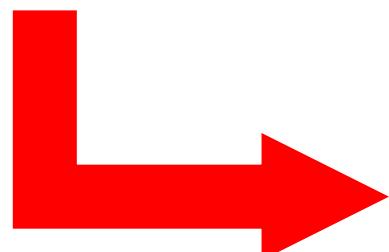
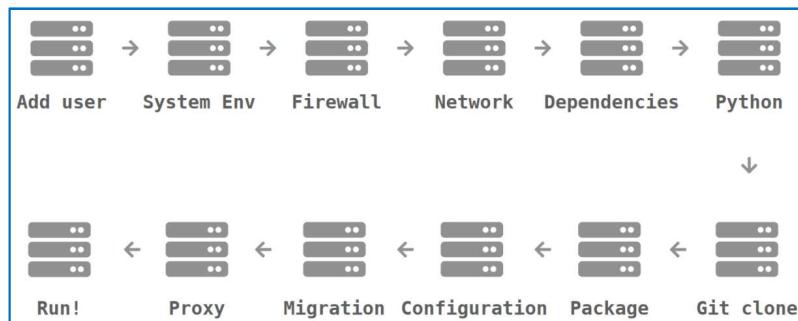
1번째 방법은...



**문서화를
잘하자.**

다시 처음으로 돌아가서, 서버를 관리한다는 것은...

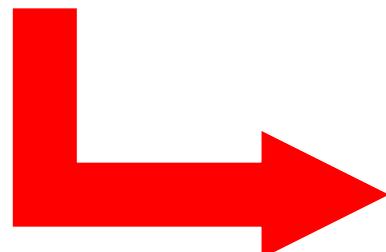
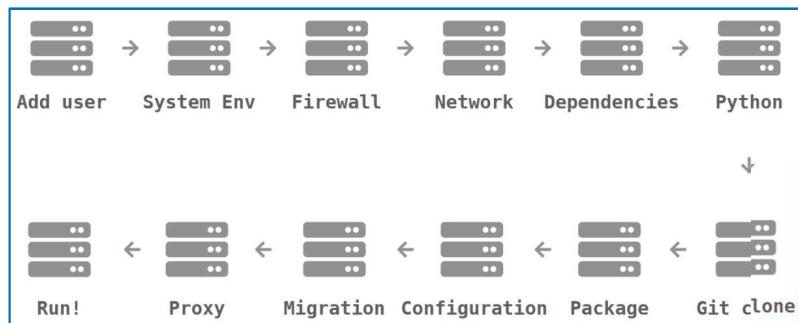
2번째 방법은...



툴을 잘
쓰자.

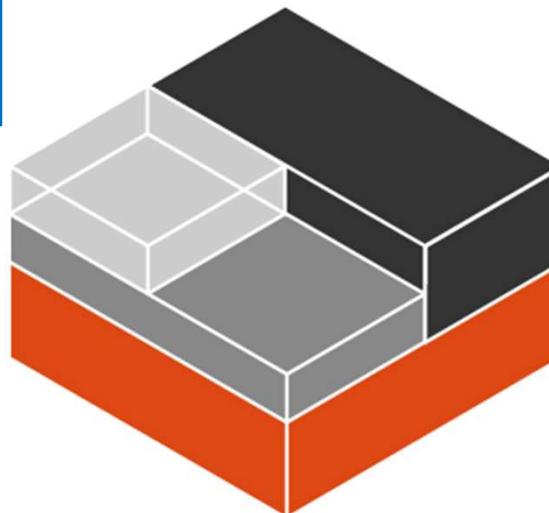
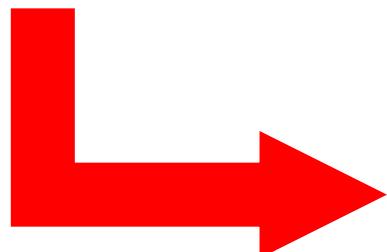
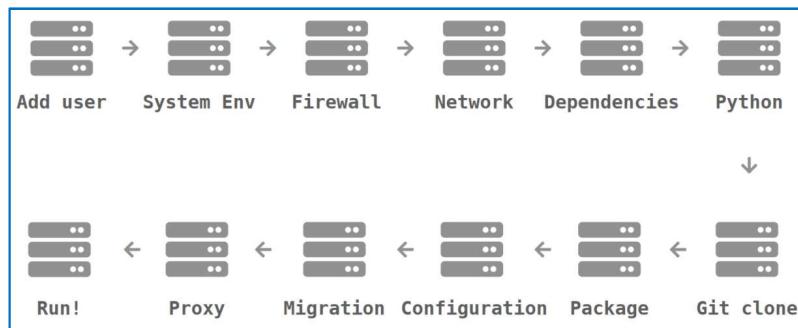
다시 처음으로 돌아가서, 서버를 관리하는 것은...

3번째 방법은...



다시 처음으로 돌아가서, 서버를 관리하는 것은...

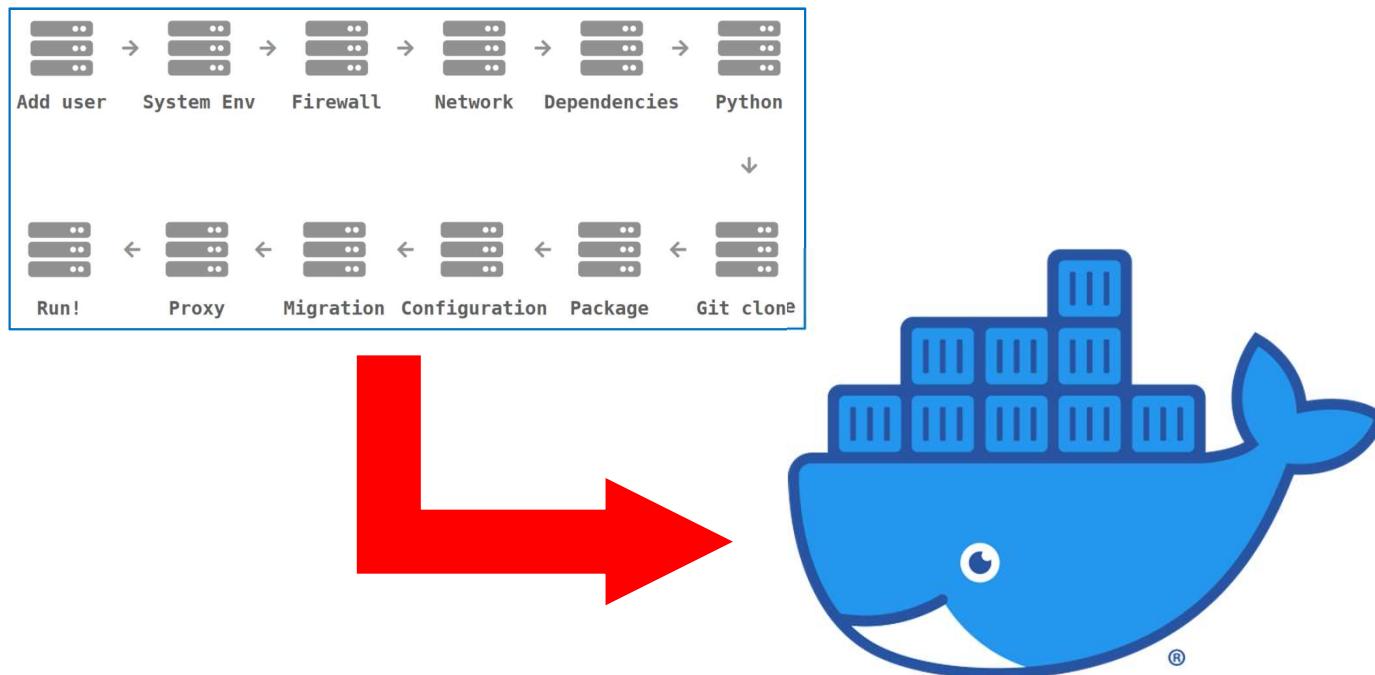
4번째 방법은...



**리눅스 기능을
이용한 빠르고
효율적인 서버
관리**

다시 처음으로 돌아가서, 서버를 관리하는 것은...

5번째 방법은...

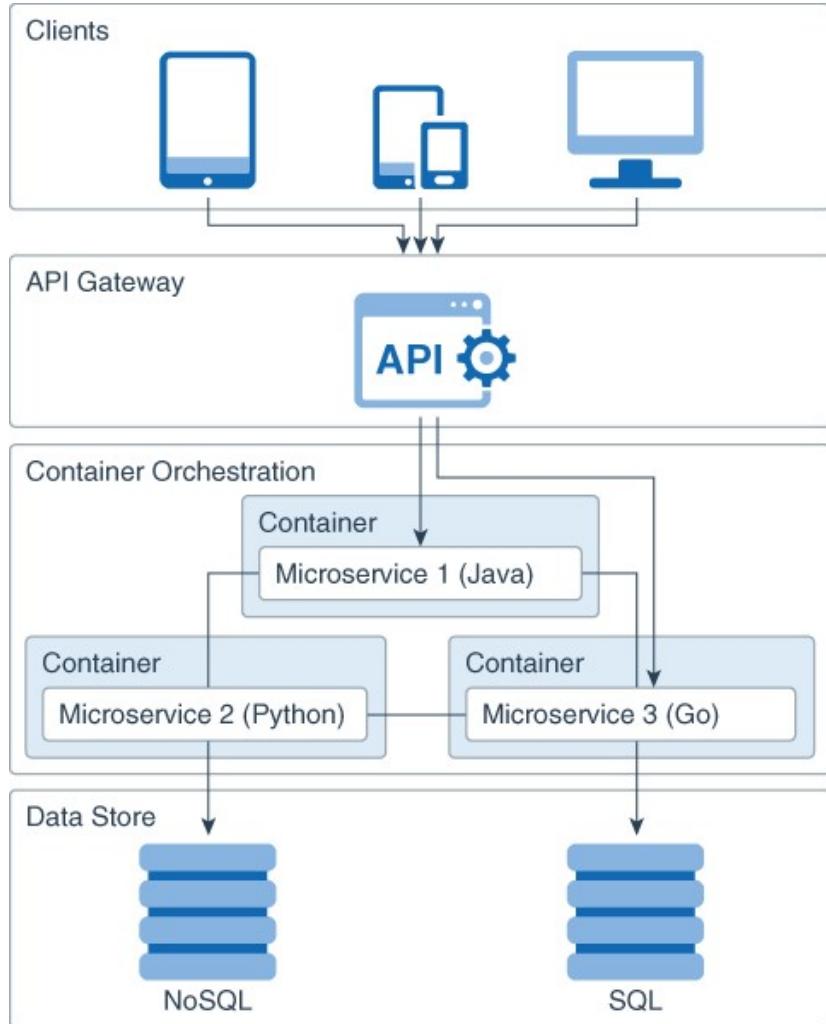


**어렵고 복잡한
기능을 사용하기
쉽게!!!**

Legacy Enterprise IT Architecture

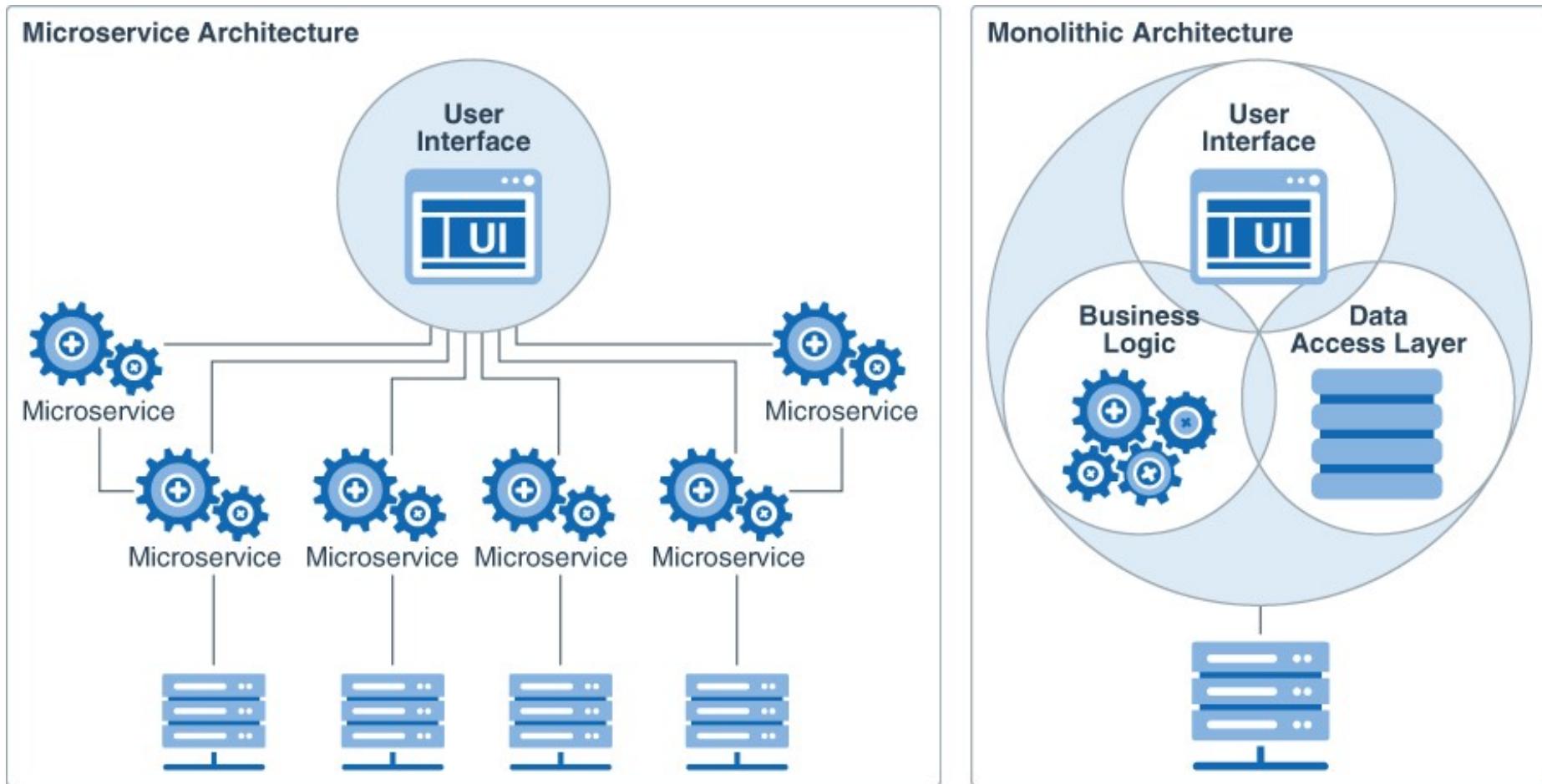


Microservices



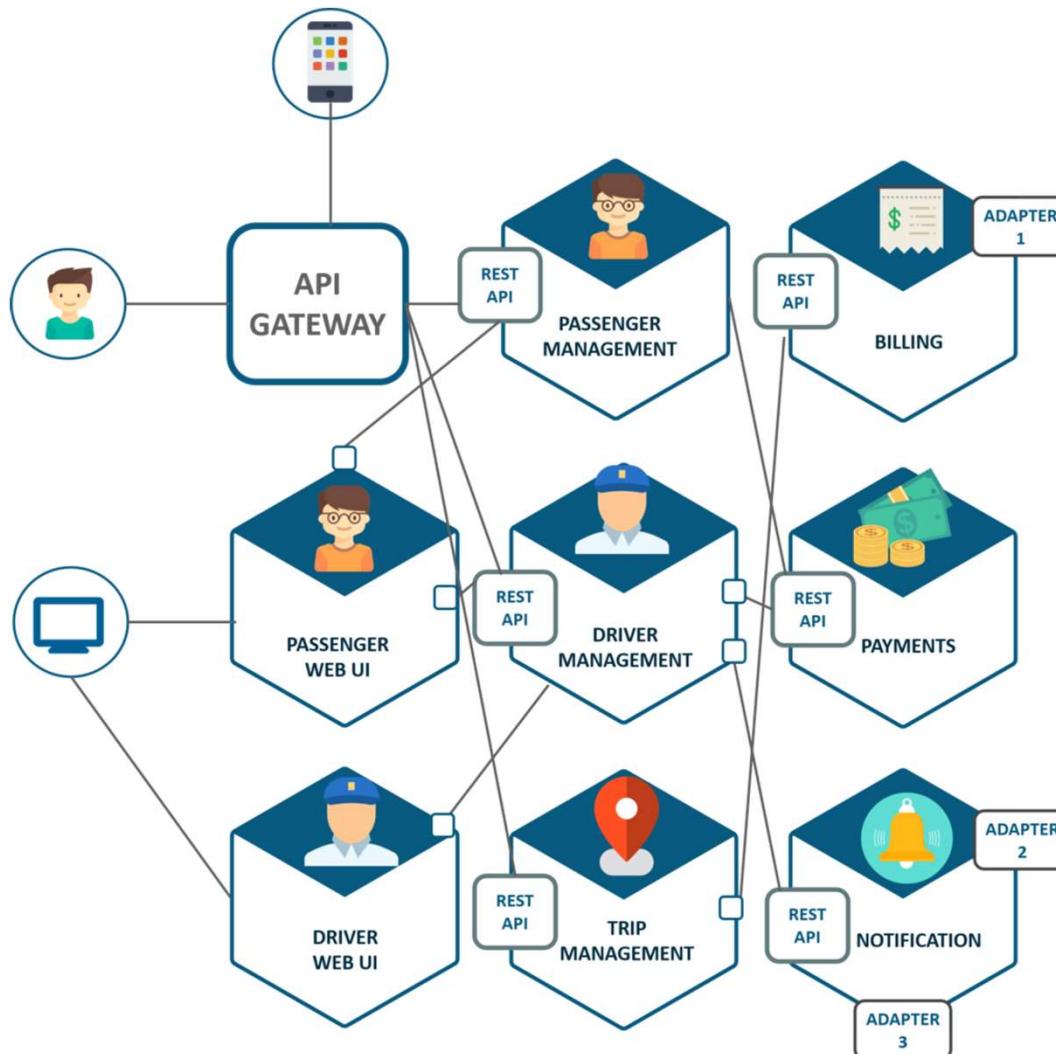
- Design an application :
 - Is multilanguage
 - Easily scalable
 - Easy to maintain and deploy
 - Highly available
 - Minimizes failures
- Each microservice owns :
 - A simple task
 - Communicates with the clients or with other microservices
 - Using lightweight communication mechanisms such as REST API requests.

Microservices



<https://docs.oracle.com/en/solutions/learn-architect-microservice/#GUID-1A9ECC2B-F7E6-430F-8EDA-911712467953>

Microservices

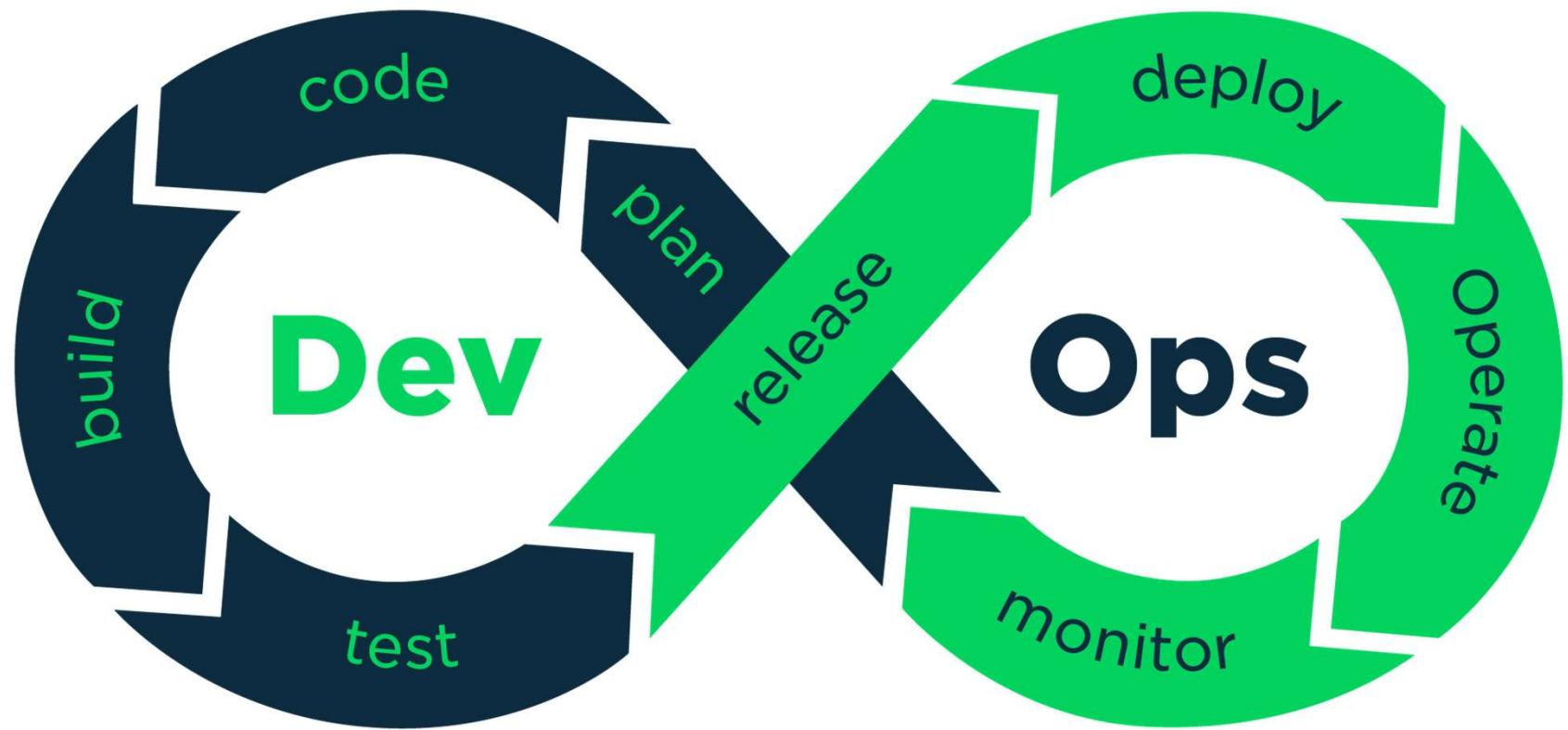


<https://aionys.com/how-to-benefit-from-microservices-architecture-implementation/>

Legacy Enterprise IT Architecture



DevOps



<https://devopedia.org/devops>

DevOps

“DevOps is
development
and operations
collaboration”

“DevOps
is using
automation”

“DevOps
is **small**
deployments”

It's DevOps!

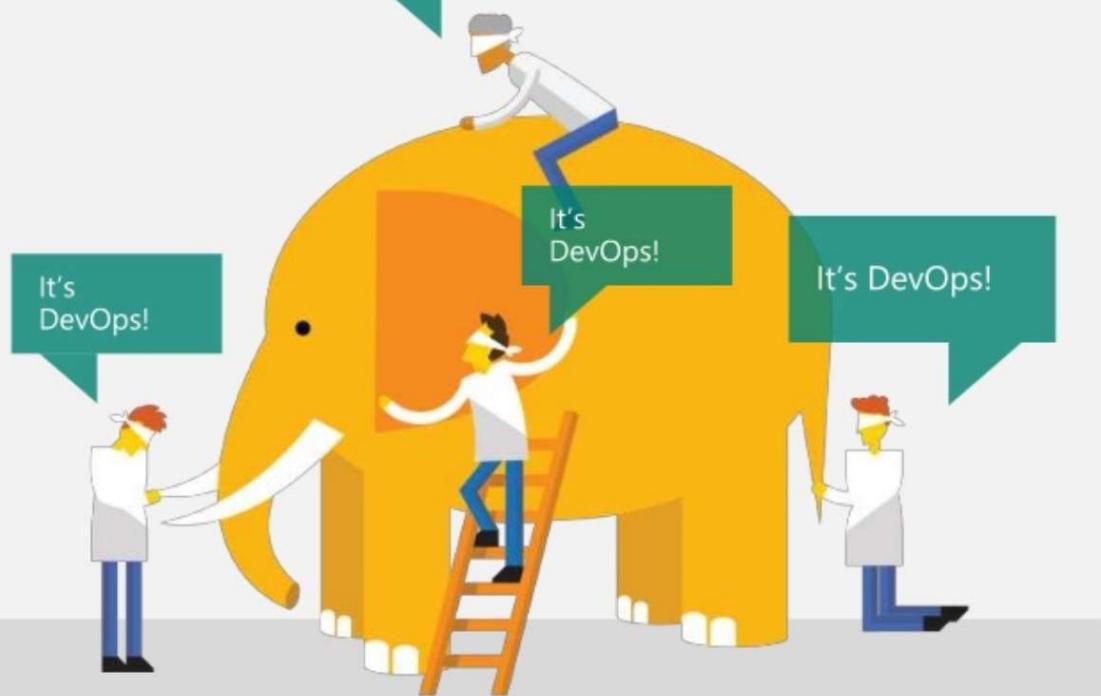
It's DevOps!

It's DevOps!

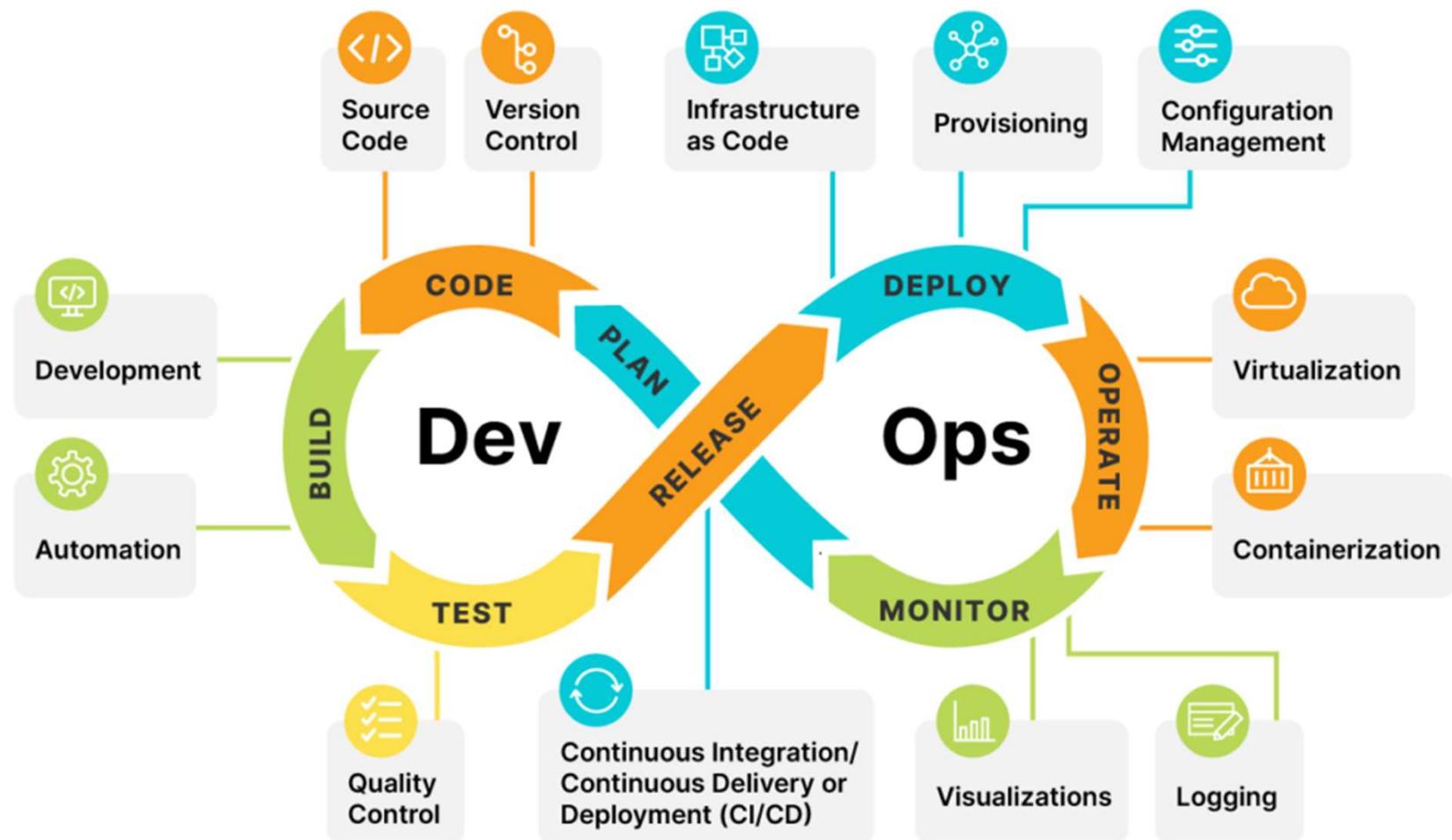
“DevOps is
treating your
infrastructure
as code”

“DevOps
is feature
switches”

“Kanban
for Ops?”

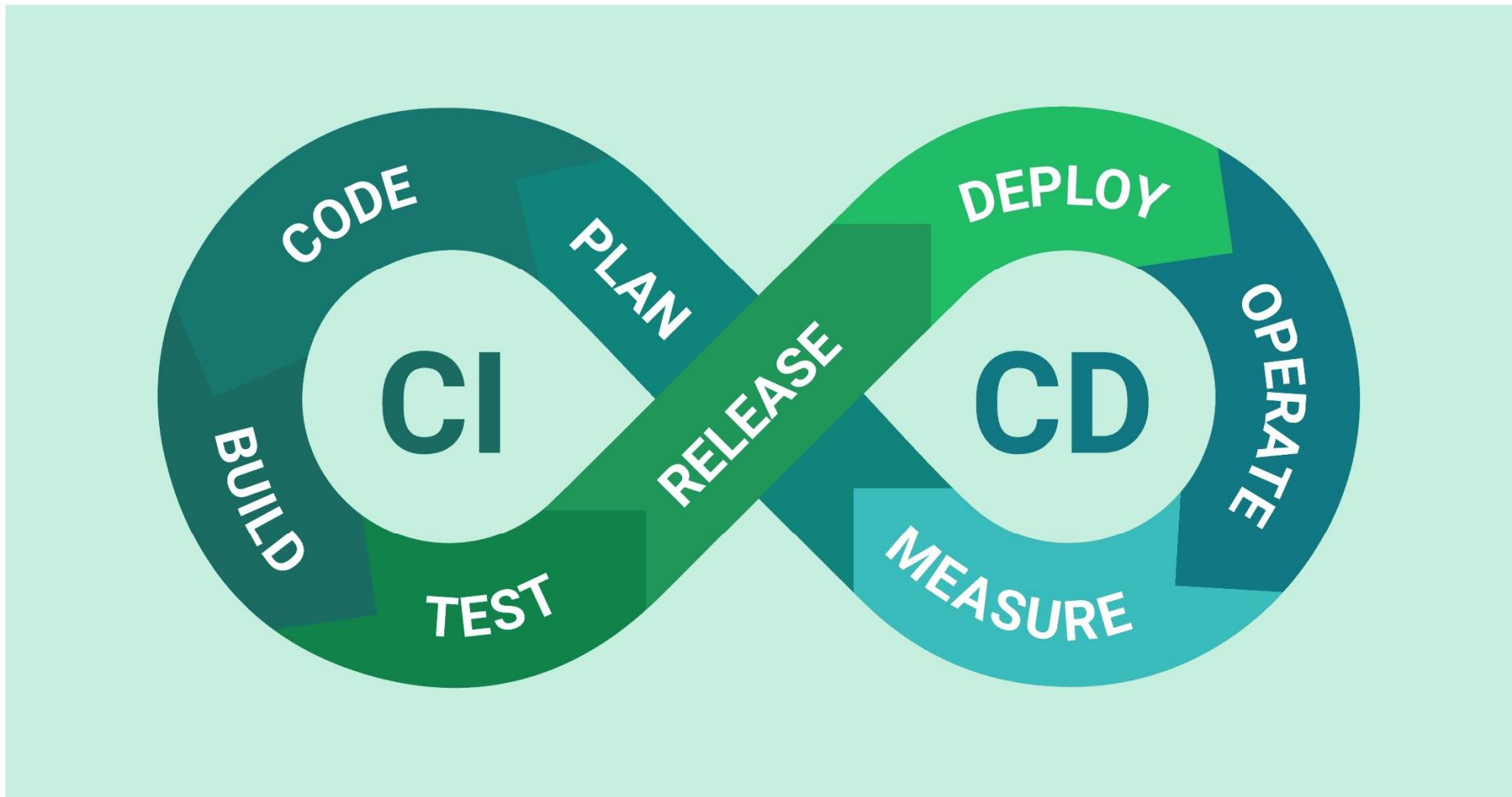


DevOps



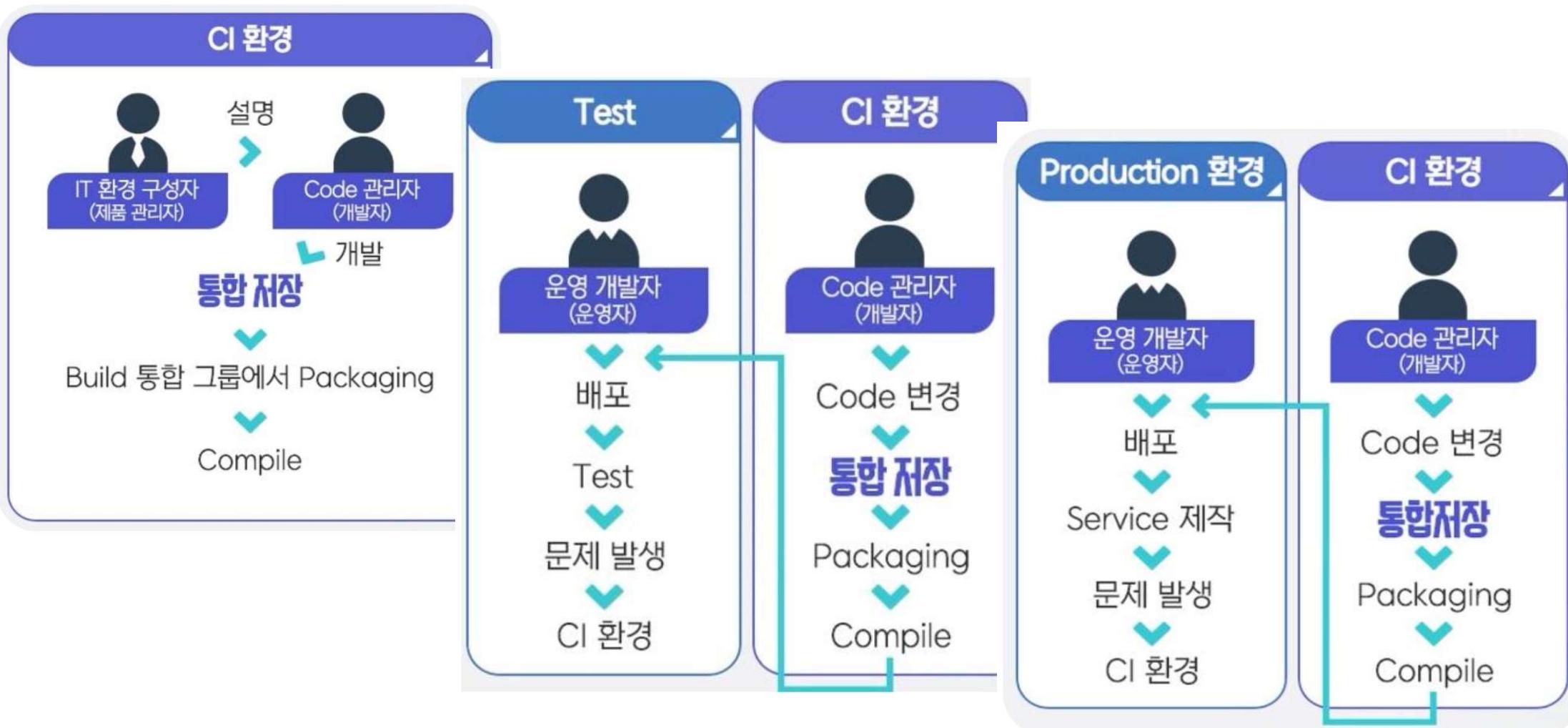
<https://orangematter.solarwinds.com/2022/03/21/what-is-devops/>

CI/CD

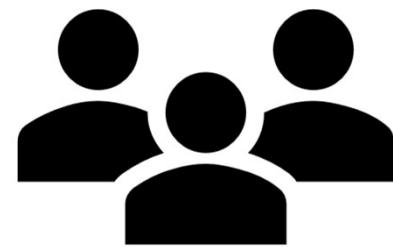
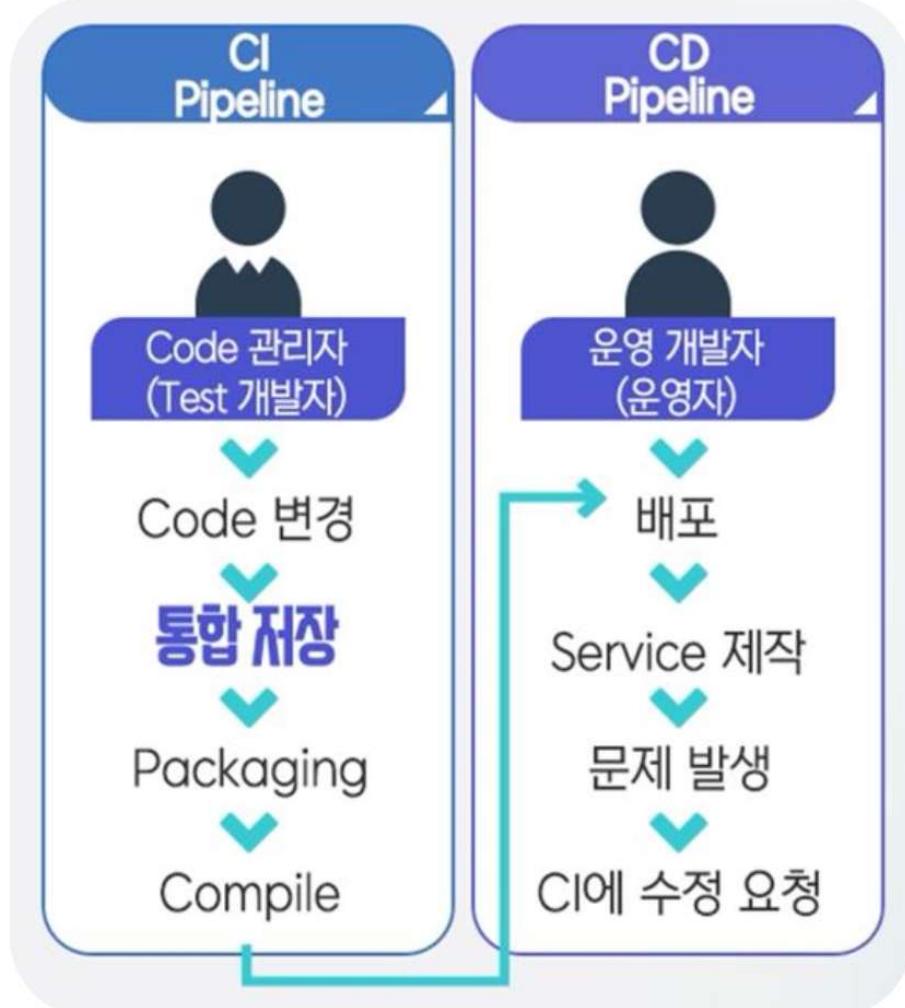


<https://velog.io/@cham/CICD-CICD%EB%9E%80>

CI/CD



CI/CD



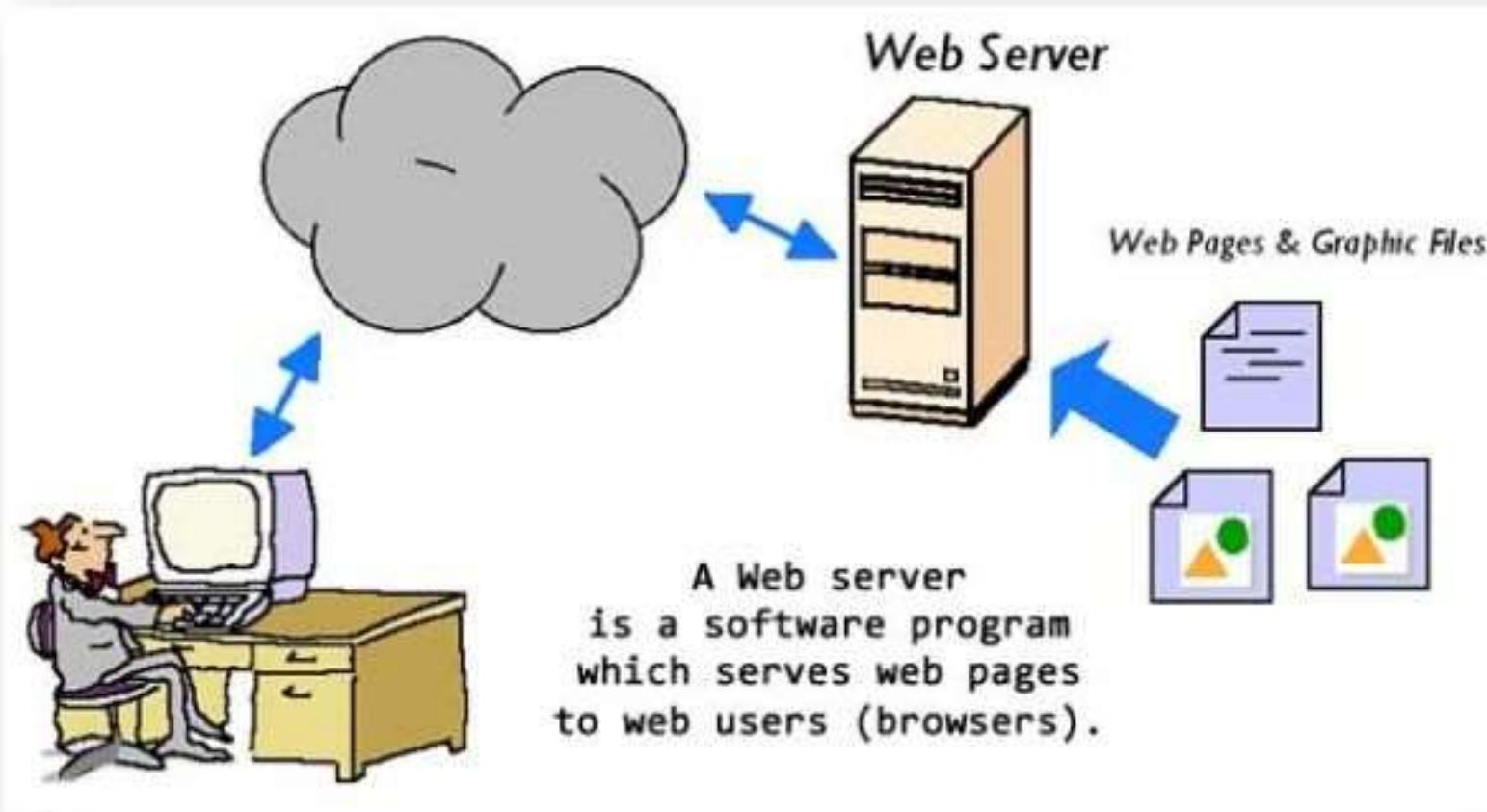
SRE(Site Reliability Engineering)
or DevOps Engineer



Building Serverless Applications with Python in AWS

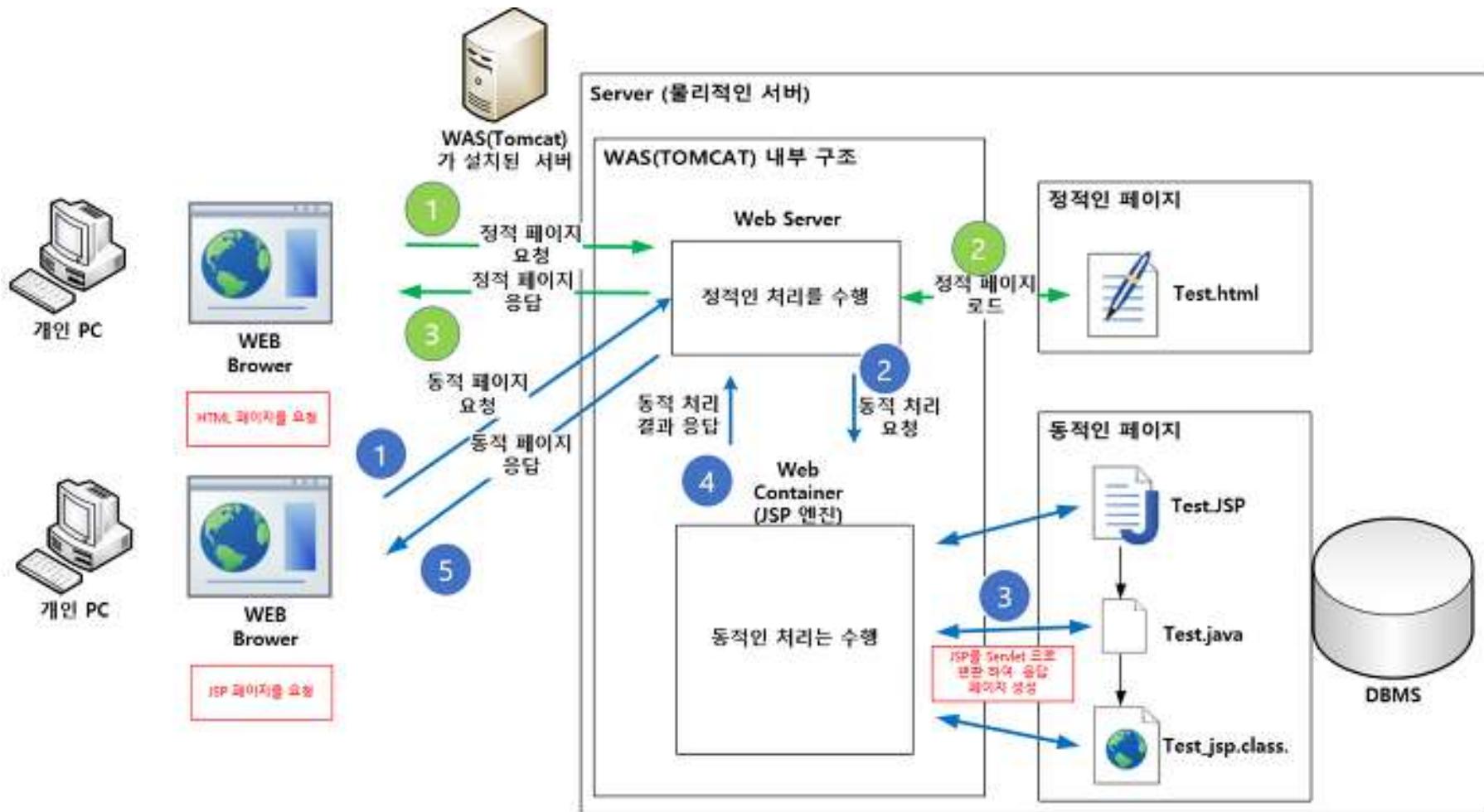


Server-Less vs Server-Oriented



<https://dyncs.data.blog/2020/08/17/what-is-a-web-server/>

Server-Less vs Server-Oriented

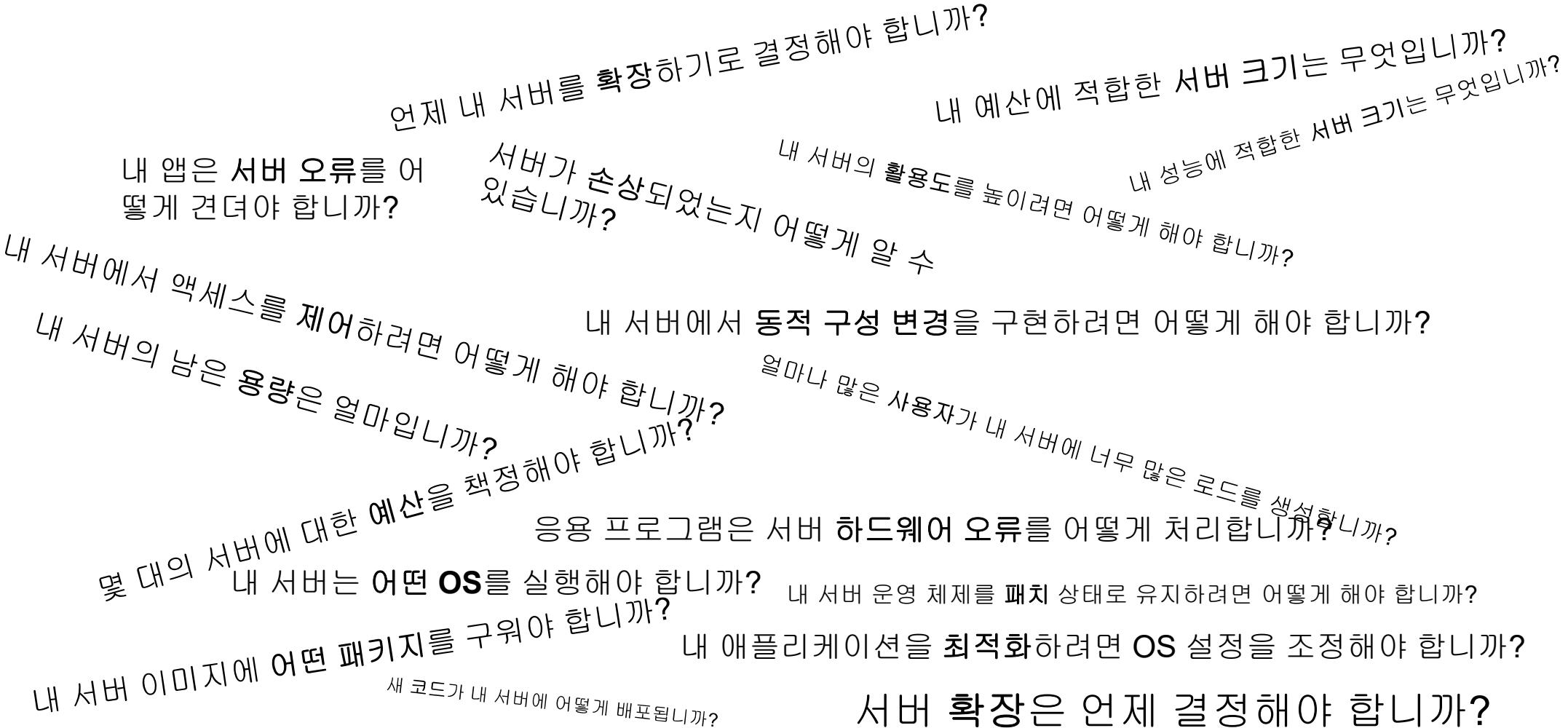


Server-Less vs Server-Oriented



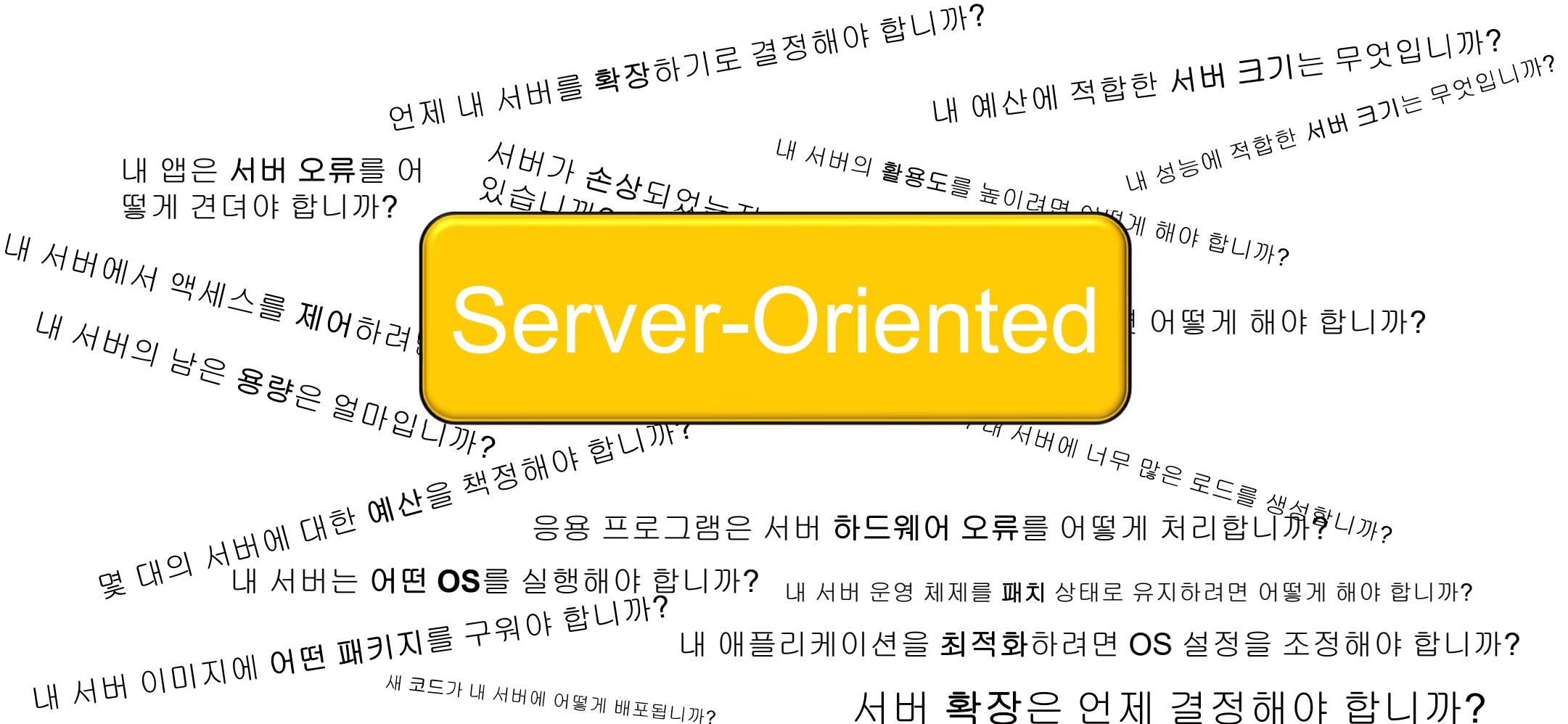
<https://www.crn.com/news/data-center/google-unveils-new-750m-data-center-as-part-of-9-5b-goal>

Server-Less vs Server-Oriented



Server-Less vs Server-Oriented

Server-Oriented



Server-Less vs Server-Oriented



<https://stackify.com/function-as-a-service-serverless-architecture/>

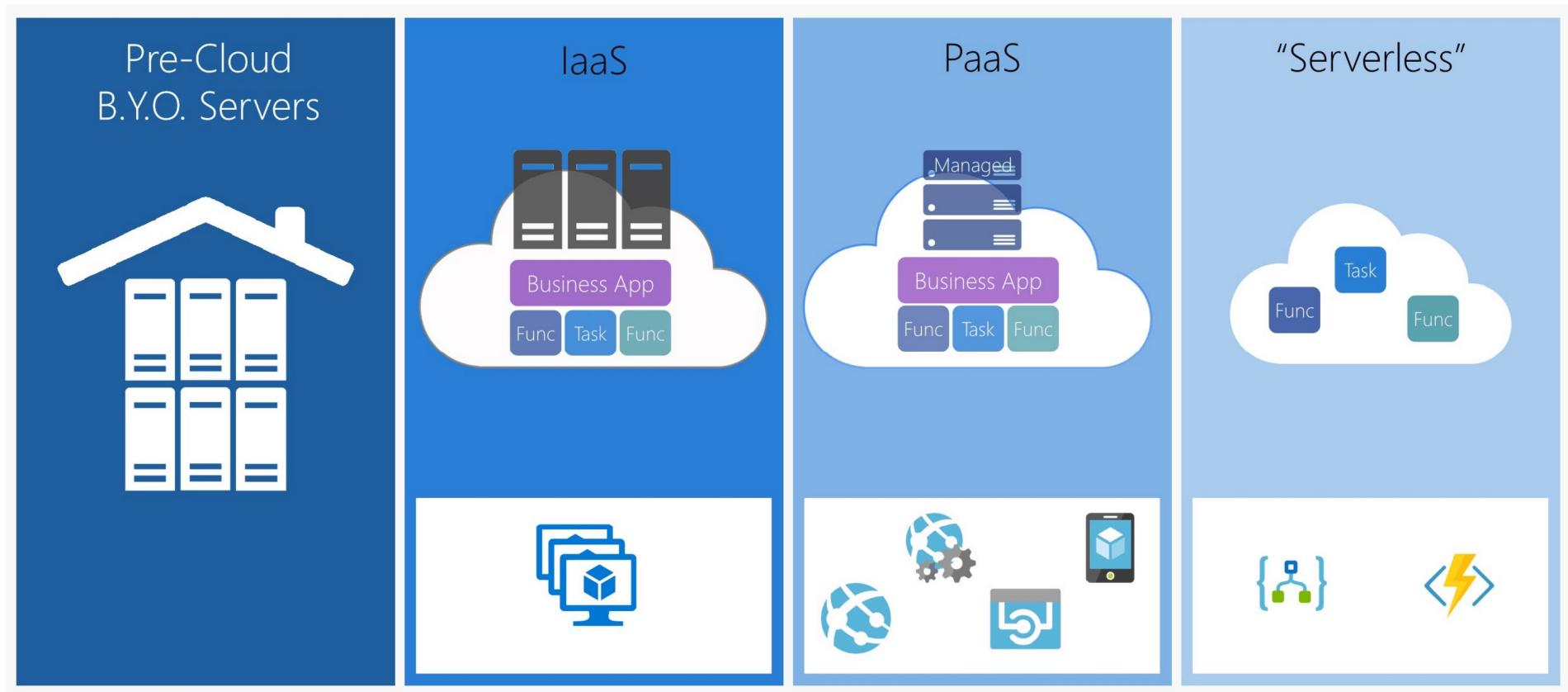
Serverless

“**Serverless**” is a **misnomer** in the sense that servers are **still** used by cloud service providers to execute code for developers. However, developers of serverless applications are **not concerned with** capacity planning, configuration, management, maintenance, fault tolerance, or scaling of containers, VMs, or physical servers.

Serverless computing **does not hold resources** in volatile memory; computing is rather done in short bursts with the results persisted to storage. When an app is **not in use**, there are **no computing resources** allocated to the app. Pricing is based on the **actual amount of resources** consumed by an application. It can be a form of utility computing.

Serverless computing can simplify the process of **deploying code** into production. Serverless code can be used in conjunction with code deployed in traditional styles, such as microservices or monoliths. Alternatively, applications can be written to be purely serverless and use **no provisioned servers** at all. This should not be confused with computing or networking models that do not require an actual server to function, such as peer-to-peer (P2P).

Serverless



<https://stackify.com/function-as-a-service-serverless-architecture/>

Serverless

Base64 Encoded HTML5 content in the URL as a query param

The Cloud Fn. Decodes the base64 string and renders the html

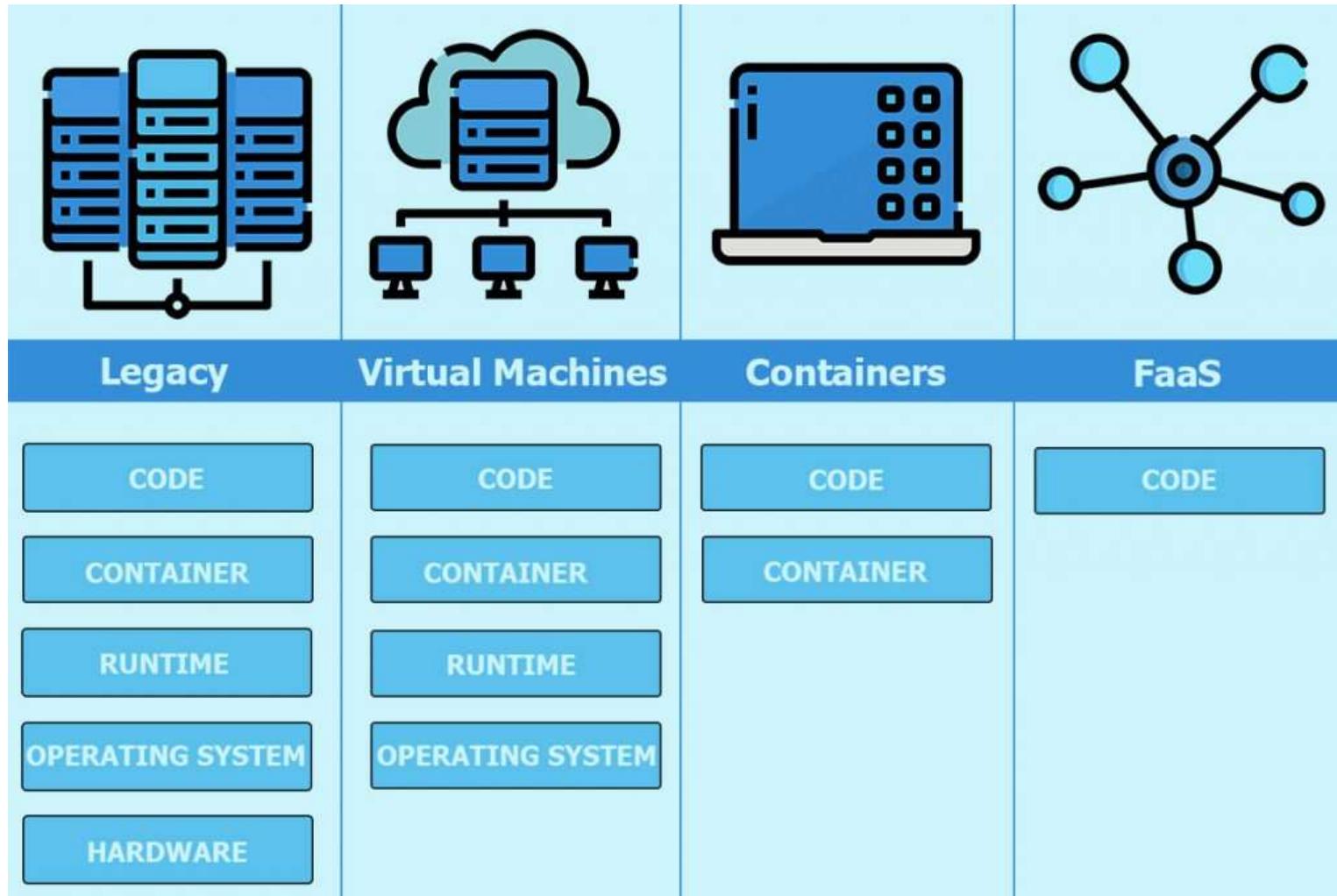


https://<faas_http_trigger_url>/<function_name>/?q=<base64_encoded_html5_content>

Ex. <https://a46b334a-6c53-4e7d-944f-e8758537b4c5.ingress.live.shoot.live.k8s-hana.ondemand.com/f1/?q=PGg2Pg0KTG10dGx1IFBvdGF0bw0KPC9oNj4NCjxoMT4NCkJpZyBQb3RhdG8NCjwvaDE+>

<https://www.validatek.com/technologies/function-service-faas>

Serverless



<https://www.webapper.com/case-for-functions-as-a-service/>

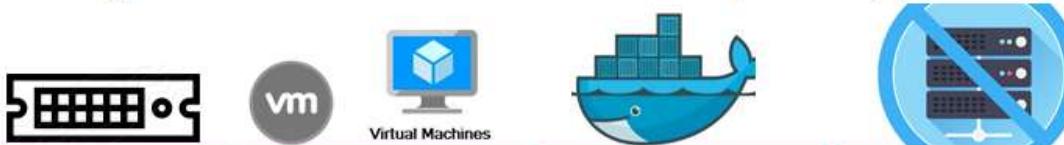
Serverless

- Serverless architecture
- Coupling vs Decoupling
- Event-based architecture design

*Serverless computing, also known as **function as a service (FaaS)**, is a cloud computing and code execution model in which the cloud provider fully manages starting and stopping of a function's container **platform as a service (PaaS)**.*

Serverless

Advantages of Serverless Computing



	Bare Metal	VM	Container	Serverless
Boot Time	~20 mins	~2 mins	2 secs	~0.0003 secs
App deployment lifecycle	Deploy in Weeks Live for years	Deploy in minutes Live for weeks	Deploy in Seconds Live for minutes/hours	Deploy in milliseconds Live for seconds
Development Complexity	Need to know: 1. Hardware 2. OS 3. Runtime Environment 4. Application code	Need to know: 1. OS 2. Runtime Environment 3. Application code	Need to know: 1. Runtime Environment 2. Application code	Need to know: 1. Application code
Investment	Buy/rent dedicated server	Rent a dedicated VM, on a shared server	Rent Containers, pay for the actual runtime	Pay for compute resources used during runtime
Scaling	Takes months Should be approved by a panel of experts	Takes hours Should be approved by administrators	Takes seconds Policy driven scaling	Takes milliseconds Scaling is event driven

https://3.bp.blogspot.com/-WfMUSEeaR84/WfFBaEVKdjl/AAAAAAAAMyM/_FIHqv5TaOlli4fzlna90jHCqpO6CJB9ACLcBGAs/s320/Advantages%2Bof%2BServerless%2BComputing.png

Serverless

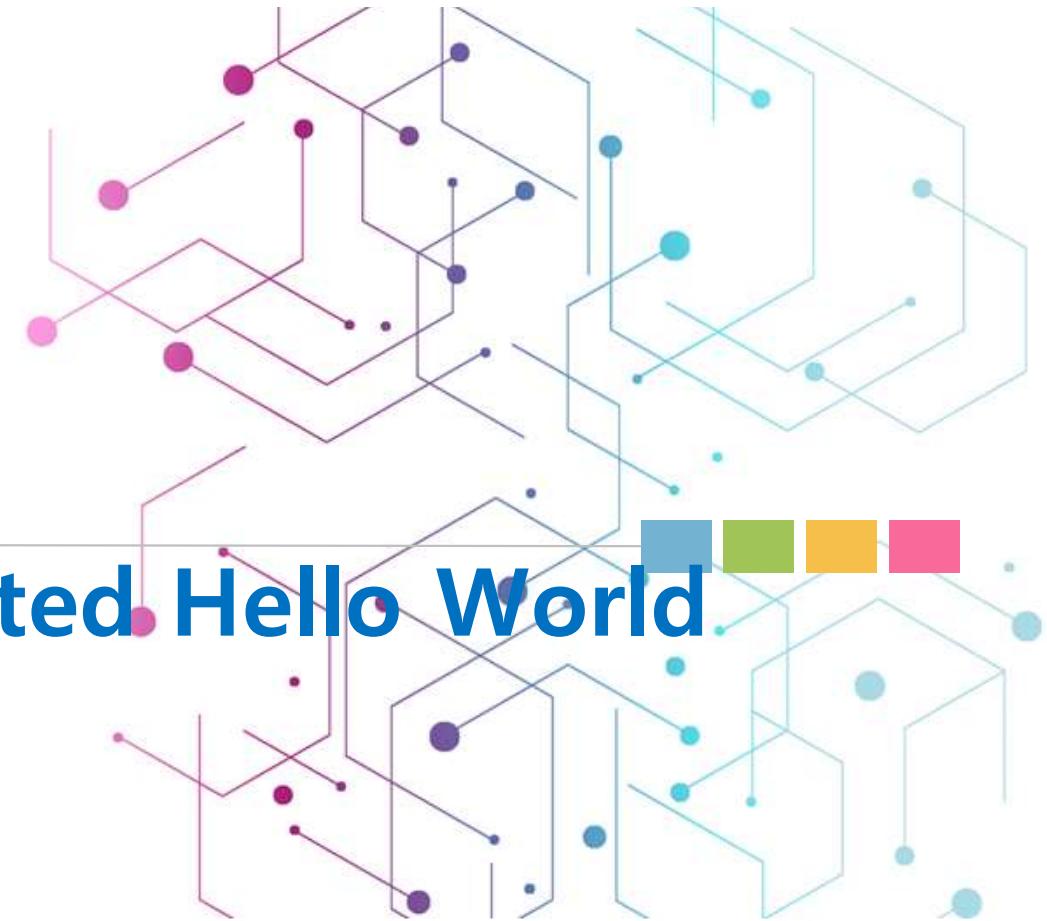
- 가격
 - 실제 사용량에 대해서만 비용 청구
- 애플리케이션의 품질에 집중
 - 서버에 집중할 필요없이 애플리케이션 품질 향상에 좀 더 집중 가능
- 높은 가용성과 유연한 확장
 - 요청이 들어올 때만 실행되고
 - 동적으로 자원이 할당
 - 스케일링에 대한 고민 불필요.

• Cold Start

- 상시 대기가 아닌 요청이 들어올 때 시작
- 클라우드 제공 플랫폼에 더욱 종속적
 - 타 플랫폼 이전이 힘듦
- 긴 시간이 필요한 작업에 불리
 - 동영상 업로드, 데이터 백업 등.



Lab. Server-Oriented Hello World

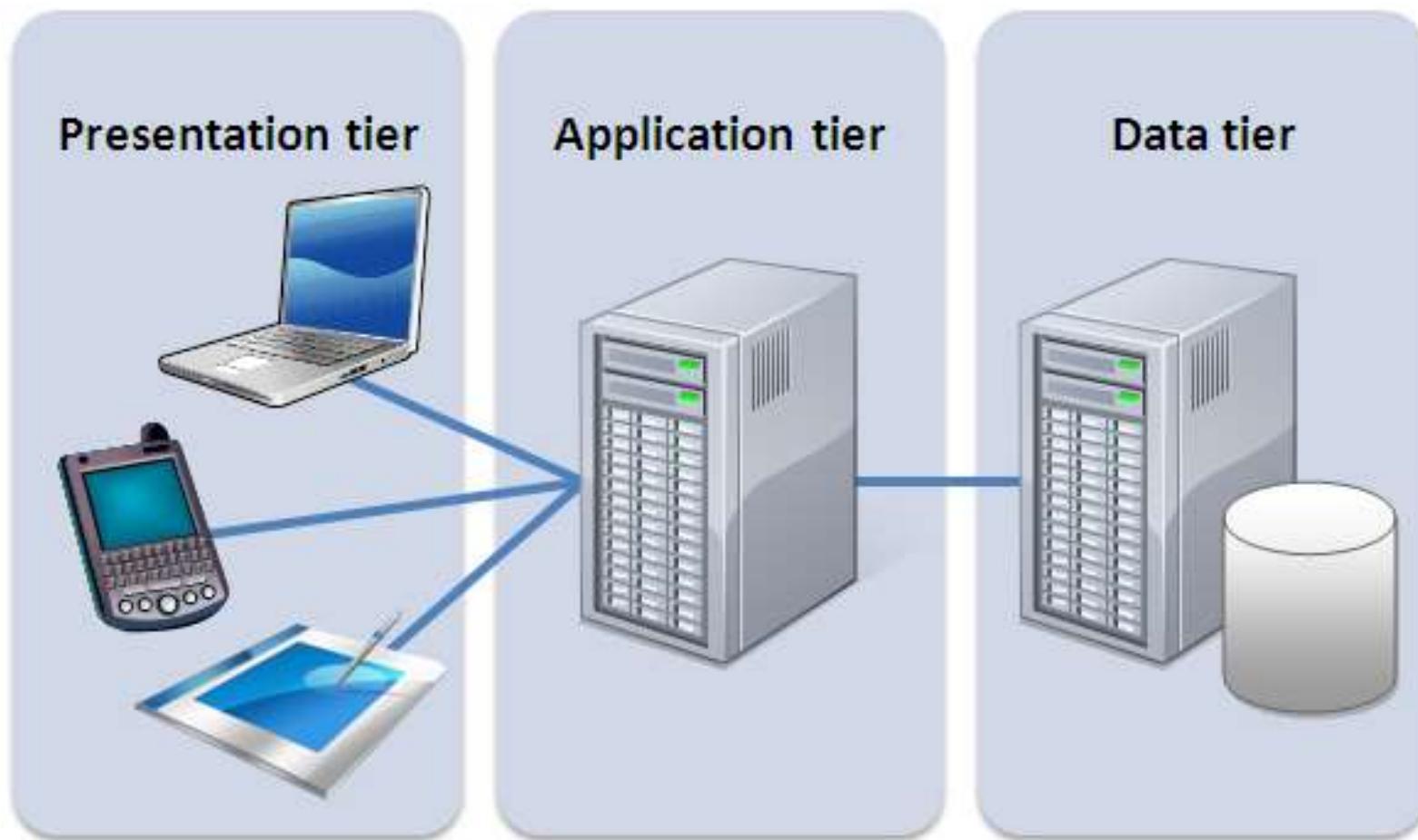




Lab. Serverless Hello World



Server-Oriented in AWS



<https://managementmania.com/en/three-tier-architecture>

Server-Oriented in AWS (Cont.)

- 고객에게 두 수를 입력 받아 그 합을 반환하는 웹 서비스를 개발할 때 개발 과정
 - Python의 Flask 또는 Django Framework을 사용하여 두 수를 입력 받고, 그 합을 반환하는 HTTP API Backend Server 개발
 - React 또는 Vue.js 등의 적당한 Frontend Framework를 선택하여 Frontend Page 개발
 - Backend Server에서 Frontend Page를 제공하는 코드 개발
- 배포 과정
 - Cloud 업체로부터 적당한 가사 서버를 할당받는다.
 - Frontend Page와 Backend Code를 서버에 복제하고 기동한다.

Server-Oriented in AWS (Cont.)

- 고객이 해당 서버의 IP 주소를 통해 접근하기 보다 도메인을 부여하는 과정
 - Domain 제공 업체로부터 Domain을 구입한다.
 - Name Server를 구축하거나 Domain 제공 업체에서 제공하는 서비스를 이용하여 구축한 가상 서버의 IP 주소를 연결한다.
- HTTP가 아닌 HTTPS로 서비스를 제공하는 과정
 - LetsEncrypt와 같은 곳에서 SSL 인증서를 발급받는다.
 - Backend Server가 인증서를 사용하여 HTTPS 서비스가 가능하도록 Code를 수정한다.
 - 수정한 Code를 가상 서버에 복제하고 다시 기동한다.
 - 인증서는 만료 기간이 있으므로 만료 직전에 인증서를 다시 발급받고 가상 서버를 갱신하는 작업을 주기적으로 해야 한다.

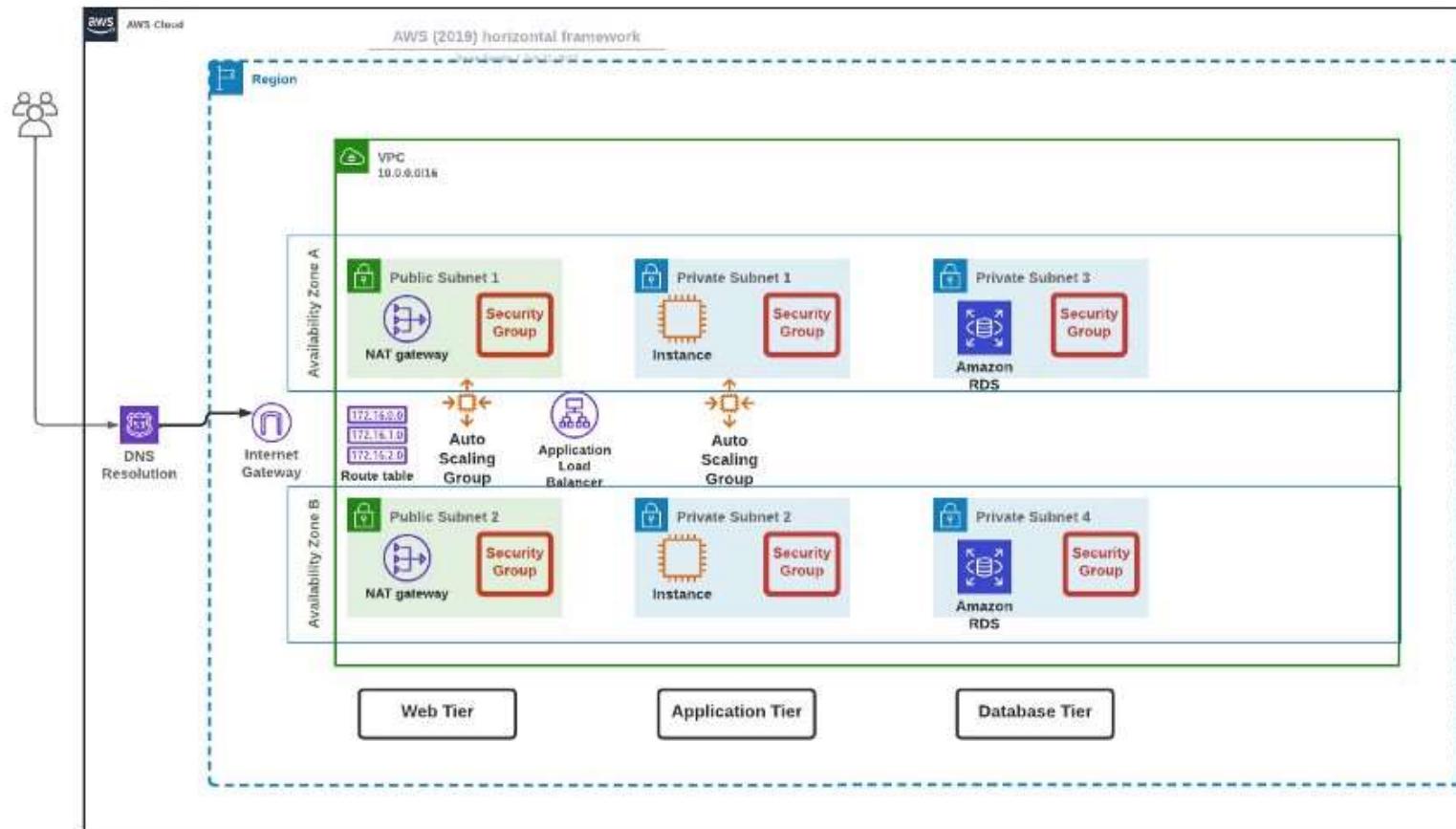
Server-Oriented in AWS (Cont.)

- 서비스 도중 오류가 발생할 경우 배포된 서버에 접근하여 Log 확인을 하는 과정
 - 가상 서버에 접속하여 Backend Server가 남긴 log 파일을 확인한다.
 - 가상 서버의 디스크 공간에는 한계가 있으므로 오래된 log 파일을 압축하여 보관하거나 삭제하는 로직을 추가한다.
- 서비스가 정장 작동하고 있는지 확인하기 위해 Monitoring이 필요하다.
 - Backend Server의 응답 중 정상과 비정상 응답수를 세어서 지표 서버에 보고 한다.
 - 지표 서버로부터 수치를 시각화하는 Dashboard를 구축한다.
 - 비정상 응답수가 일정 이상일 경우 경보를 보내는 기능을 추가한다.

Server-Oriented in AWS (Cont.)

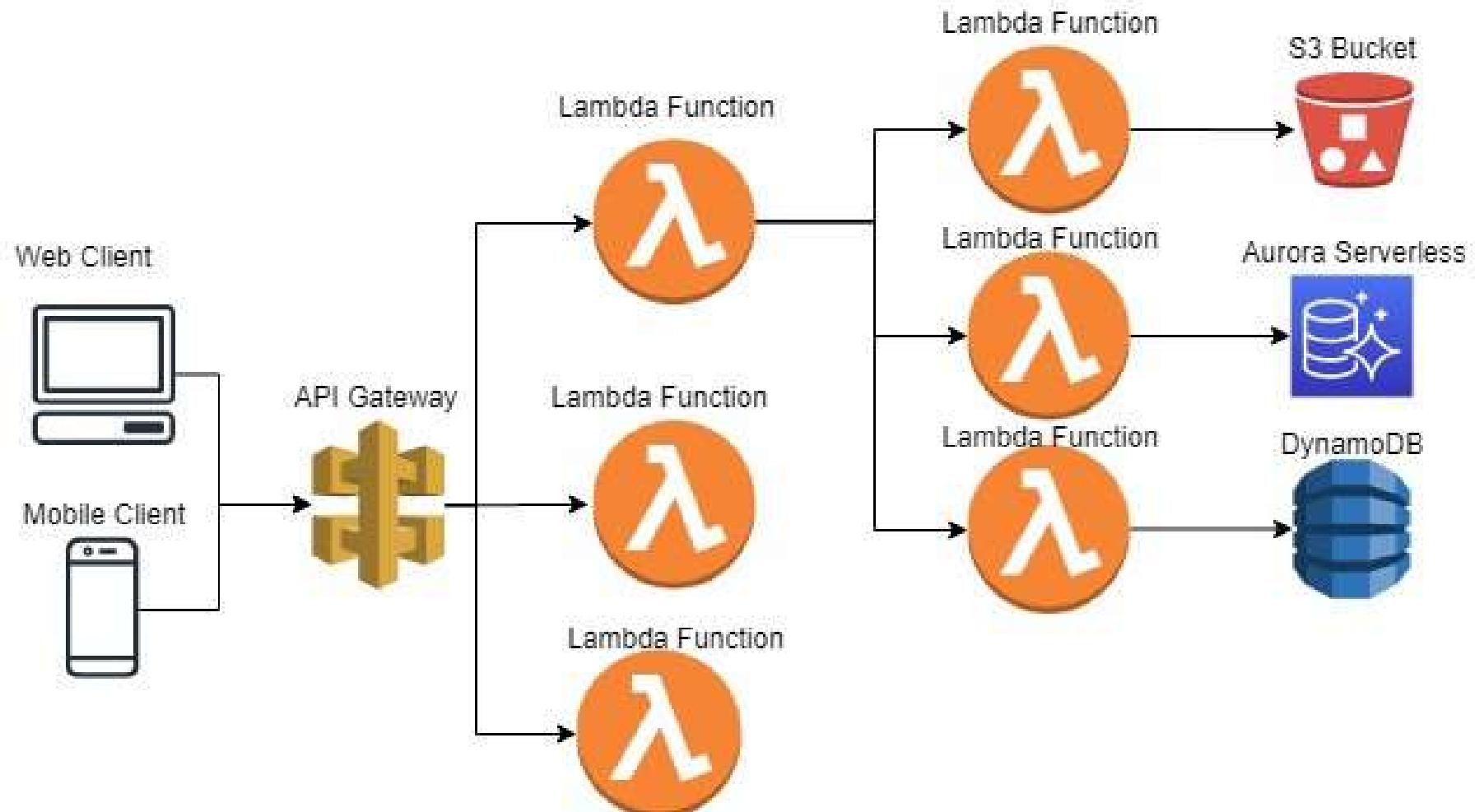
- 갑자기 고객이 많아져서 서비스가 응답할 수 있는 수준 이상의 요청이 발생하게 될 경우 서버 증설을 하는 과정
 - 필요한 만큼 가상 서버를 더 많이 할당한다.
 - 모든 가상 서버에 Frontend Page와 Backend Code를 복제하고 기동한다.
 - 가상 서버에 요청을 분배하는 Load Balancer를 설정하고 여기에 Domain을 연결한다.

Server-Oriented in AWS (Cont.)



<https://medium.com/@bryanJR/3-tier-architecture-on-aws-626173d15ba0>

Serverless in AWS



<https://levelup.gitconnected.com/deploying-microservices-using-serverless-architecture-cf7d1570950>

Serverless in AWS (Cont.)

- Backend Server Code → **AWS Lambda**
- AWS Lambda 실행시 Event 생성 → **Amazon API Gateway** 사용
- Frontend Page 저장소 → **Amazon S3**
- Amazon S3에서 정적으로 제공하는 파일을 CDN을 통해 고객에게 더 빠르게 전달 → **Amazon CloudFront**
- API Gateway와 CloudFront를 위한 Domain 지정 → **Amazon Route 53**
- API Gateway와 CloudFront는 기본적으로 HTTPS를 사용

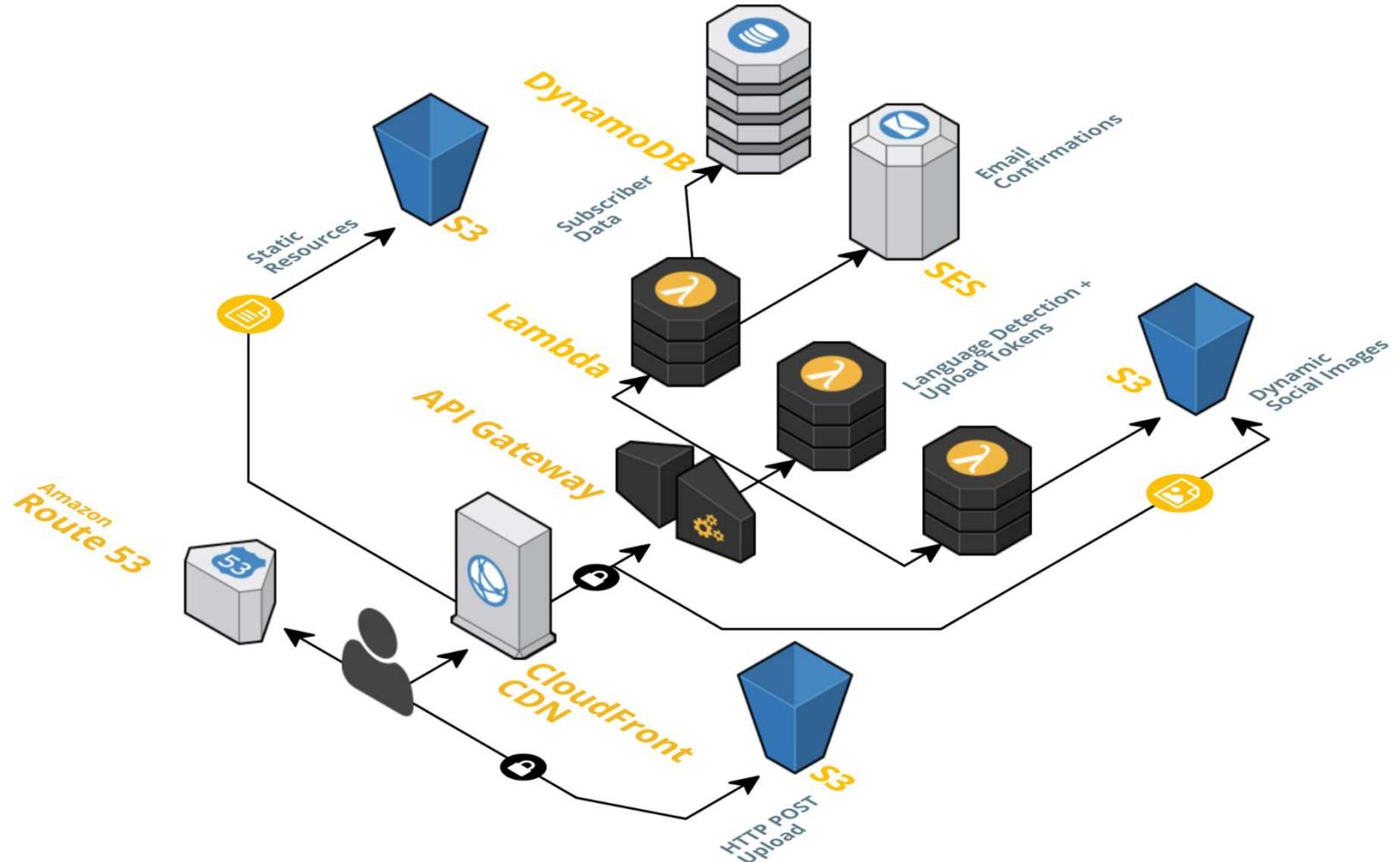
Serverless in AWS (Cont.)

- Lambda, API Gateway, CloudFront에서 발생하는 Log → **Amazon CloudWatch**
- CloudWatch는 AWS 각 서비스의 지표 정보를 Dashboard로 구축
- API Gateway와 Lambda는 요청 횟수당 사용 금액을 계산하는 종량제
- 요청량이 갑자기 증가해도 더 많은 Lambda가 동시에 실행되어 요청을 동시에 처리 가능. ← 기본 1천 개 Lambda가 동시에 처리 가능

Serverless in AWS (Cont.)

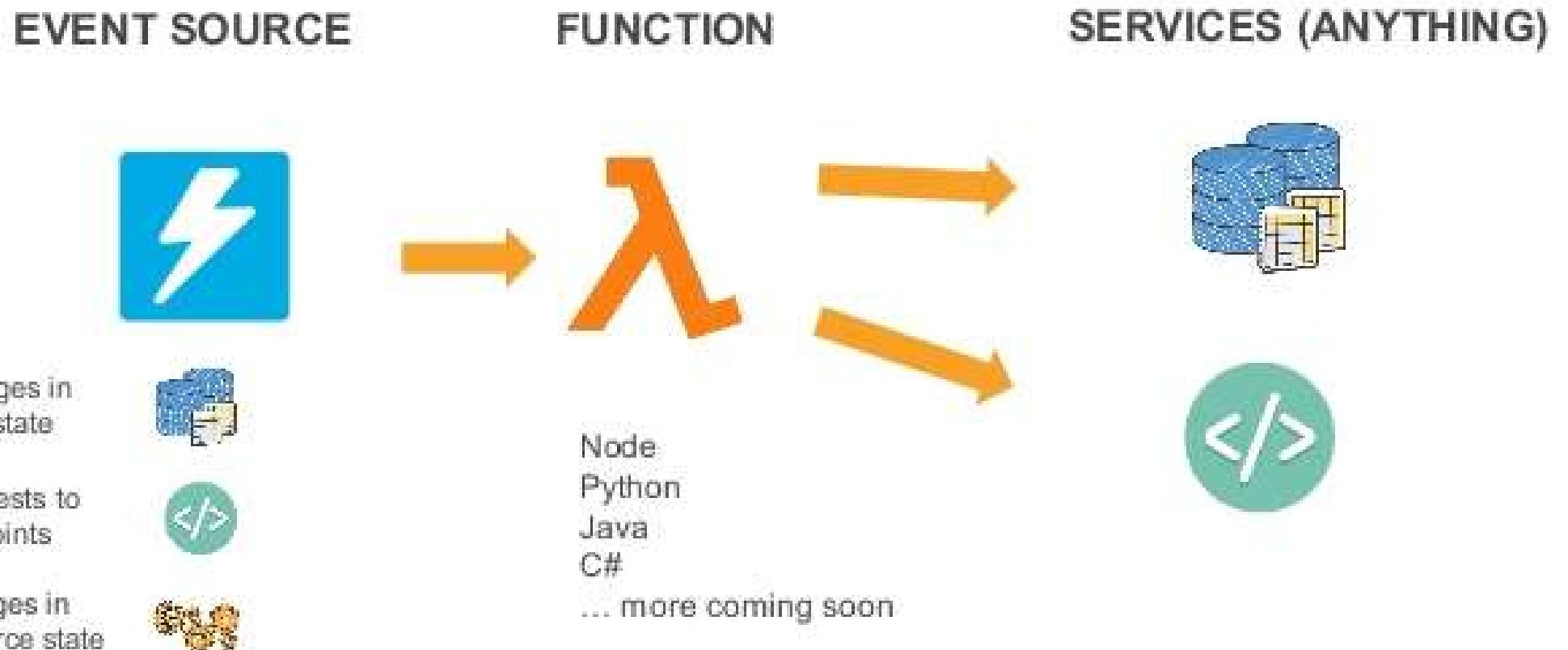
- Infra → CloudFormation으로 관리하는 API Gateway, Lambda, S3, CloudFront, CloudWatch, Route 53
- Backend Code → AWS Lambda
- Frontend Page → Amazon S3

Serverless in AWS (Cont.)



<https://www.stackery.io/blog/serverless-use-cases/>

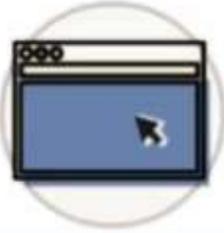
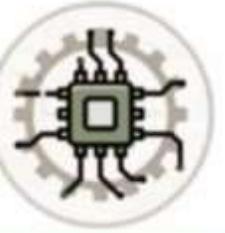
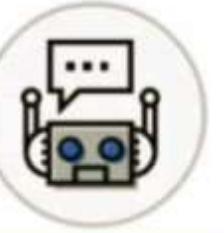
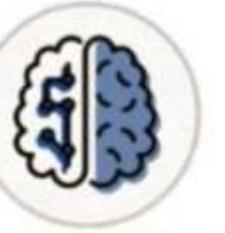
Serverless in AWS (Cont.)



Serverless in AWS (Cont.)

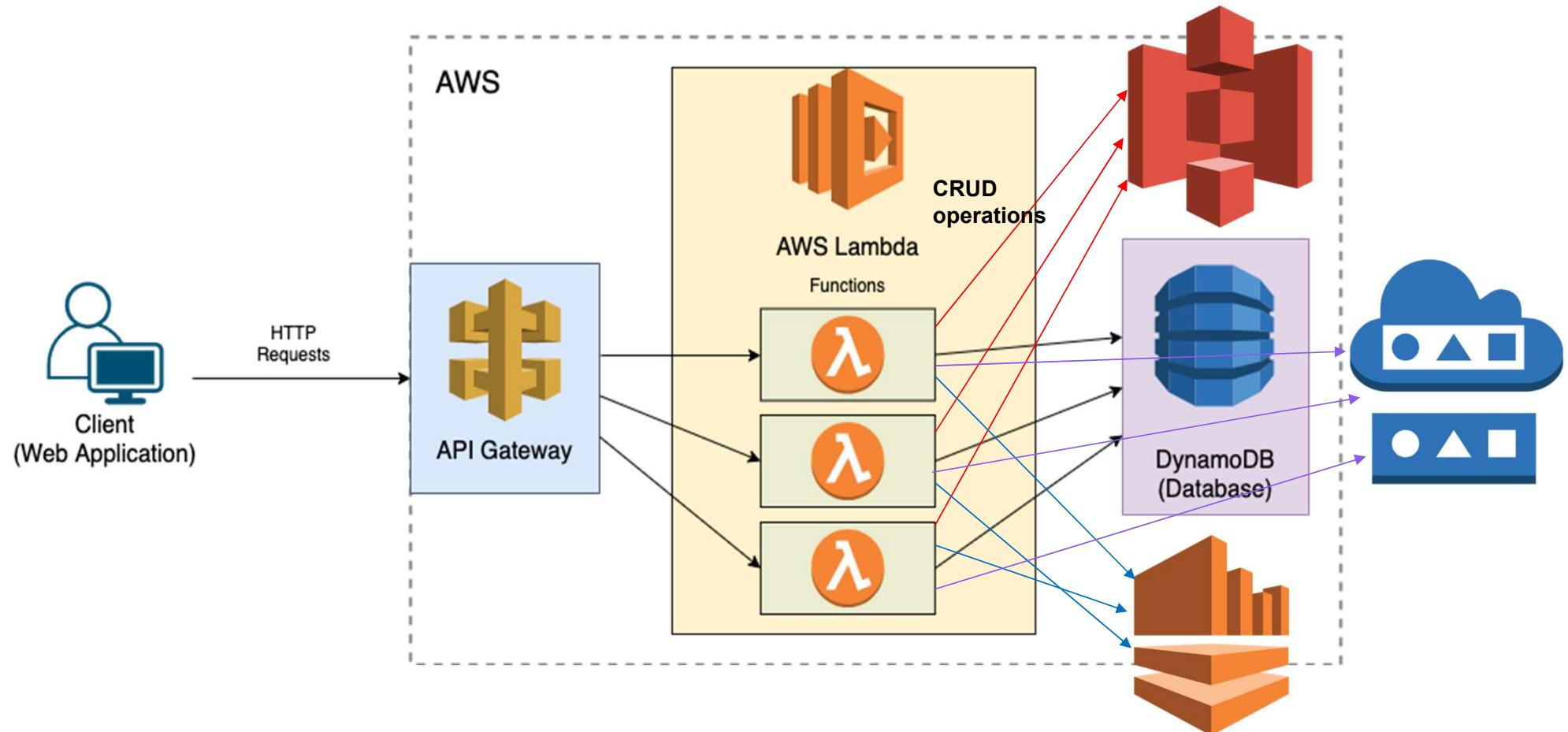


Serverless in AWS (Cont.)

					
Web Applications <ul style="list-style-type: none">• Static websites• Complex web apps• Packages for Flask and Express	Backends <ul style="list-style-type: none">• Apps & services• Mobile• IoT	Data Processing <ul style="list-style-type: none">• Real time• MapReduce• Batch	Chatbots <ul style="list-style-type: none">• Powering chatbot logic	Amazon Alexa <ul style="list-style-type: none">• Powering voice-enabled apps• Alexa Skills Kit	IT Automation <ul style="list-style-type: none">• Policy engines• Extending AWS services• Infrastructure management

<https://kruschecompany.com/serverless-architecture-for-modern-apps-providers-and-caveats/>

Serverless Use Cases – Web Applications and Backends

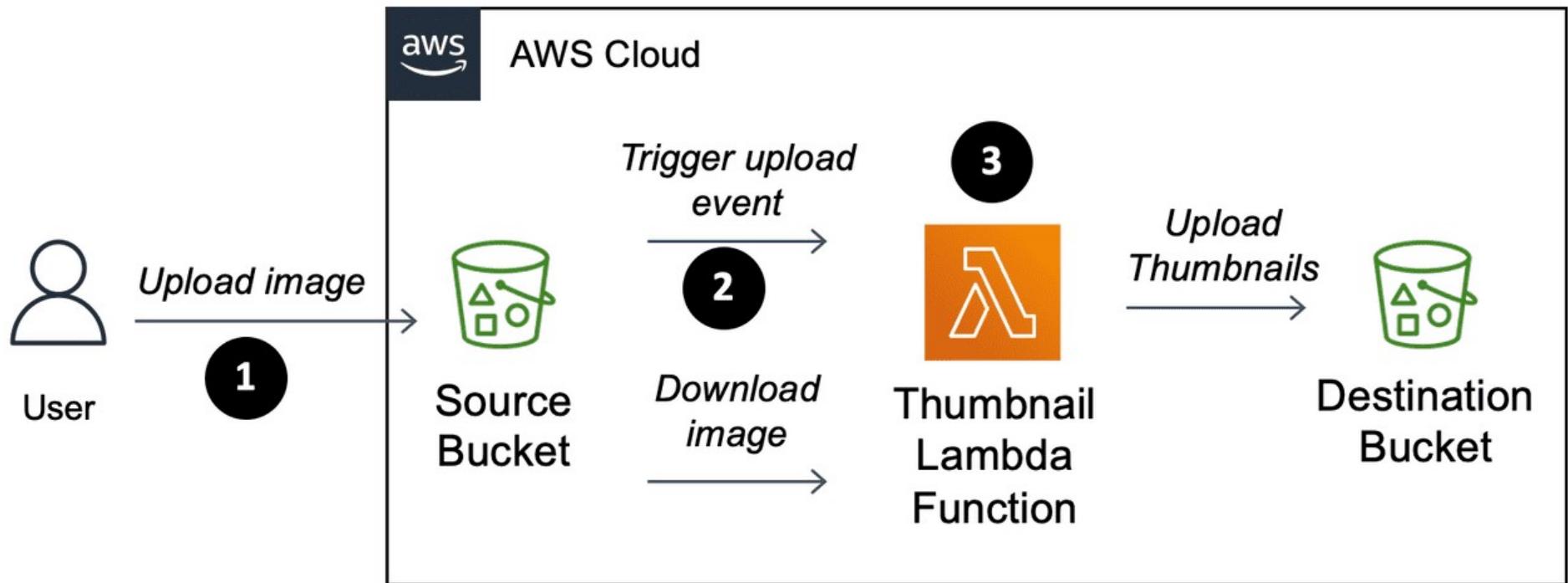


<https://medium.com/@connielok/building-a-serverless-back-end-with-aws-5bb3642a3f4>

Serverless Use Cases – Backend Order Processing



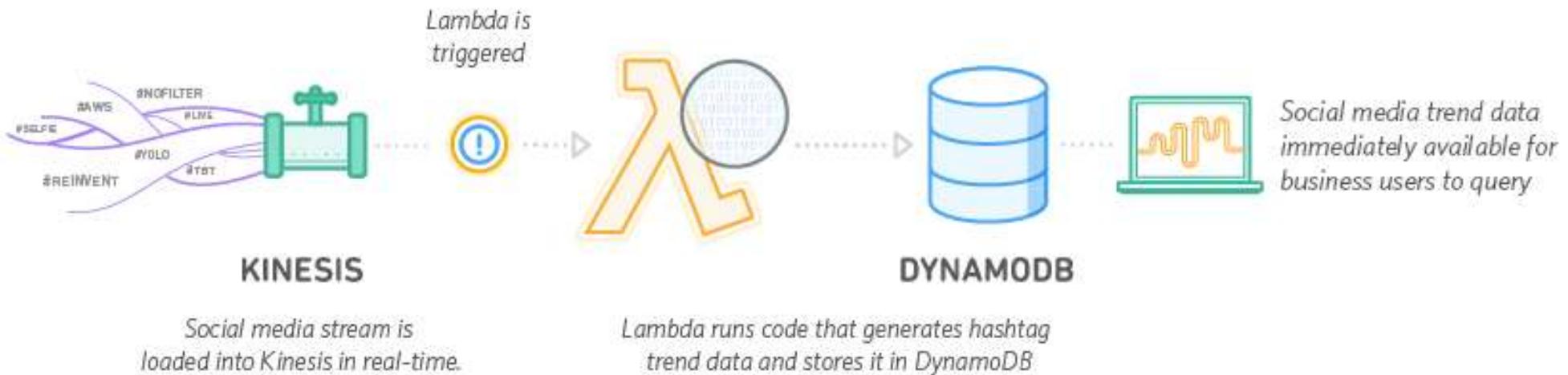
Serverless Use Cases – Image Thumbnail Generation



<https://lukstei.com/2021-11-21-creating-a-serverless-thumbnail-service-with-aws-cdk/>

Serverless Use Cases – Data Processing

Example: Analysis of Streaming Social Media Data

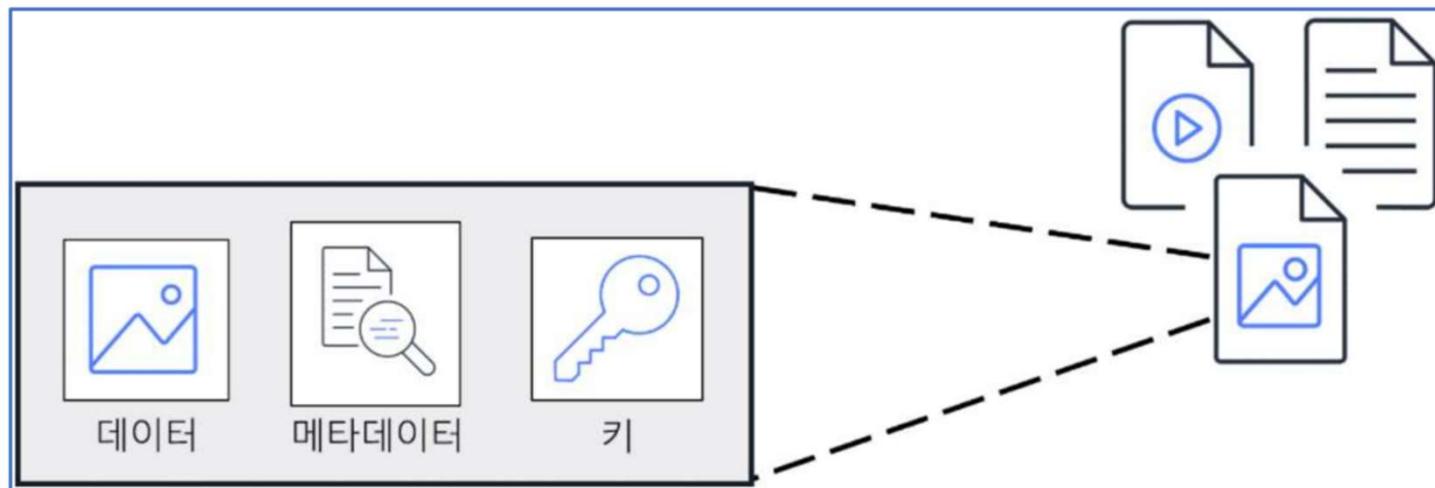


<https://aws.amazon.com/ko/lambda/data-processing/>

Amazon Simple Storage Service – S3

- Object Storage

- In object storage, each object consists of **data**, **metadata**, and a **key**.
- The **data** might be an image, video, text document, or any other type of file.
- **Metadata** contains information about what the data is, how it is used, the object size, and so on.
- An object's **key** is its unique identifier.



Amazon Simple Storage Service – S3 (Cont.)

- Is a service that provides object-level storage.
- Stores data as objects in buckets.
- You can upload any type of file to Amazon S3, such as images, videos, text files, and so on.
- For example, you might use Amazon S3 to store backup files, media files for a website, or archived documents.
- Offers unlimited storage space.
- The maximum file size for an object in Amazon S3 is 5 TB.
- When you upload a file to Amazon S3, you can set permissions to control visibility and access to it.
- You can also use the Amazon S3 versioning feature to track changes to your objects over time.





Lab. Create Amazon S3 buckets and Manage



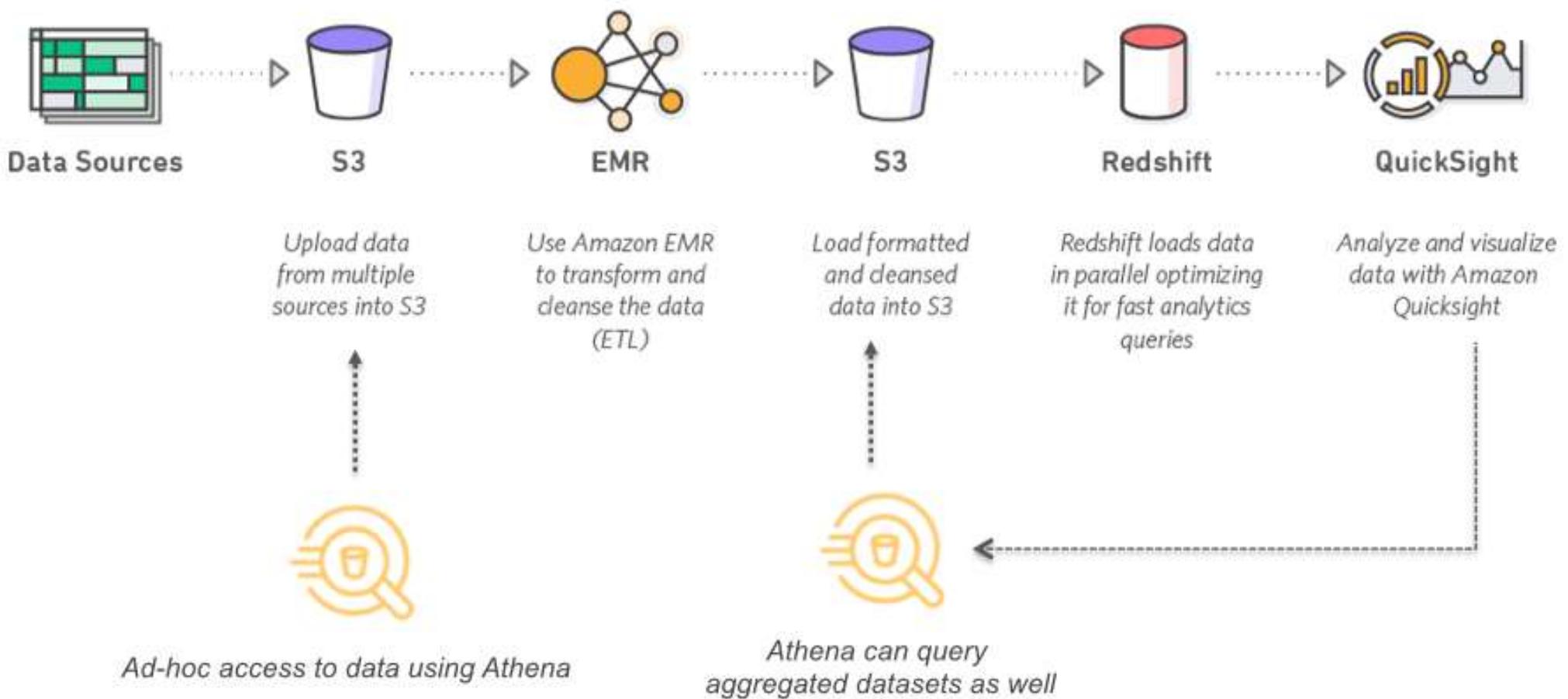
Amazon Athena

- Is an interactive query service.
- S3를 SQL 기반의 언어로 조회
- 별도의 Provisioning 또는 Server없이 동작 → Serverless
- For example :
 - 여러 Log 파일이 저장된 S3에서 필요한 데이터를 조회
 - 정형화된 Metadata 또는 저장 데이터 조회
 - Event 데이터에서 필요한 정보 추출

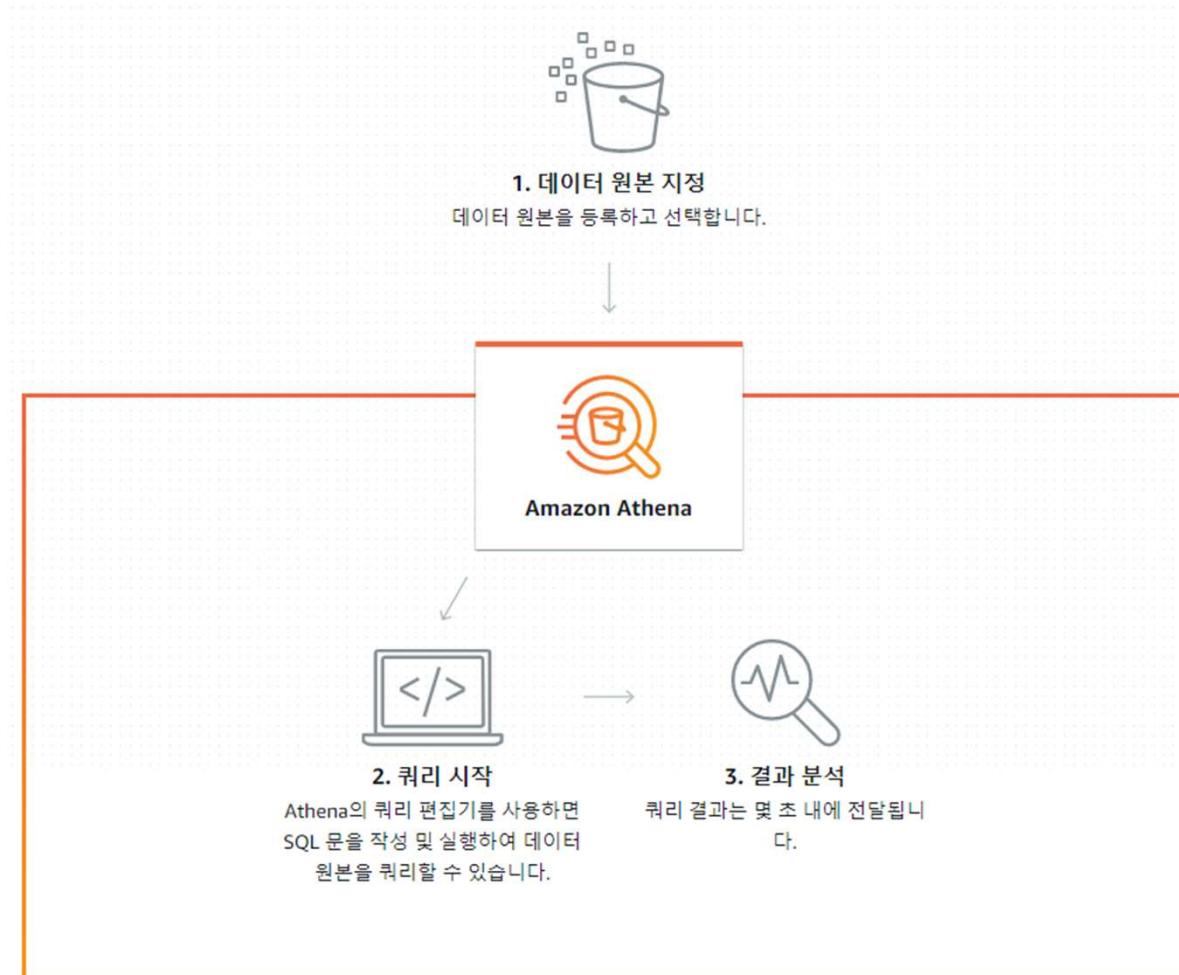


Amazon Athena

Amazon Athena (Cont.)



Amazon Athena (Cont.)





Lab. Using AWS Athena

Amazon Relational Database Service(RDS)



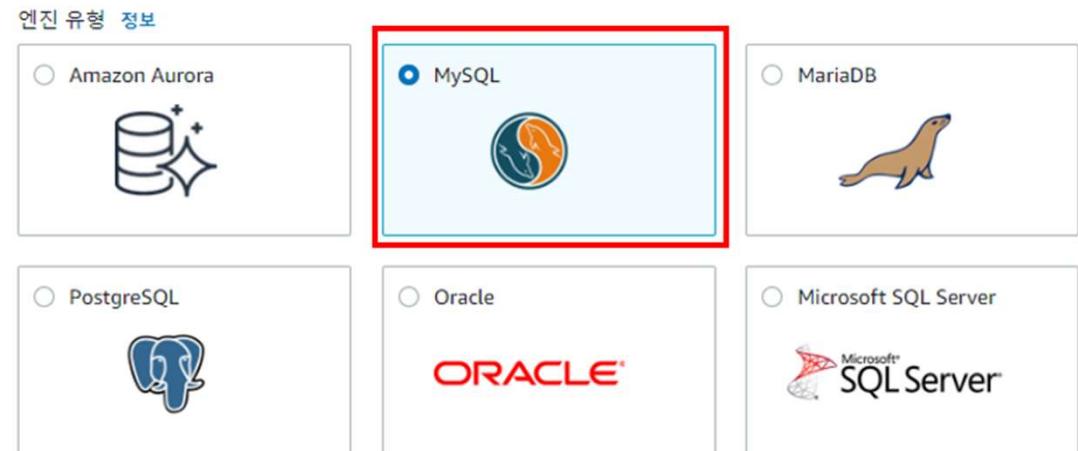
- Enables you to run relational databases in the AWS Cloud.
- Is a *managed service* that automates tasks such as hardware provisioning, database setup, patching, and backups.
- With these capabilities, can spend less time completing administrative tasks and more time using data to innovate your applications.
- You can integrate Amazon RDS with other services to fulfill your business and operational needs, such as using AWS Lambda to query your database from a serverless application.

Amazon Relational Database Service(RDS) (Cont.)



- Amazon RDS database engines

- Amazon Aurora
- PostgreSQL
- MySQL
- MariaDB
- Oracle Database
- Microsoft SQL Server



ID	Product name	Size	Price
1	Medium roast ground coffee	12 oz.	\$5.30
2	Dark roast ground coffee	20 oz.	\$9.27



Lab. Using RDS for MySQL



Amazon DynamoDB



- **Nonrelational** databases
- In a nonrelational database, a table is a place where you can store and query data.
- Nonrelational databases are sometimes referred to as **NoSQL databases** because they use structures other than rows and columns to organize data.
- One type of structural approach for nonrelational databases is **key-value** pairs.
- With **key-value** pairs, data is organized into items (**keys**), and items have attributes (**values**).

Amazon DynamoDB (Cont.)



- You can think of attributes as being different features of your data.
- In a *key-value* database, you can add or remove attributes from items in the table at any time.
- Additionally, not every item in the table has to have the same attributes.

Key	Value
1	<p>Name: John Doe</p> <p>Address: 123 Any Street</p> <p>Favorite drink: Medium latte</p>
2	<p>Name: Mary Major</p> <p>Address: 100 Main Street</p> <p>Birthday: July 5, 1994</p>

Amazon DynamoDB (Cont.)



- Amazon DynamoDB is a *key-value* database service.
- It delivers single-digit millisecond performance at any scale.
- **Serverless:**
 - Do not have to provision, patch, or manage servers.
 - Do not have to install, maintain, or operate software.
- **Automatic scaling:**
 - As the size of your database shrinks or grows, automatically scales to adjust for changes in capacity while maintaining consistent performance.
 - This makes it a suitable choice for use cases that require high performance while scaling.
- Refer to https://www.youtube.com/watch?v=U_GJYMUjiwA



Lab. Using Amazon DynamoDB

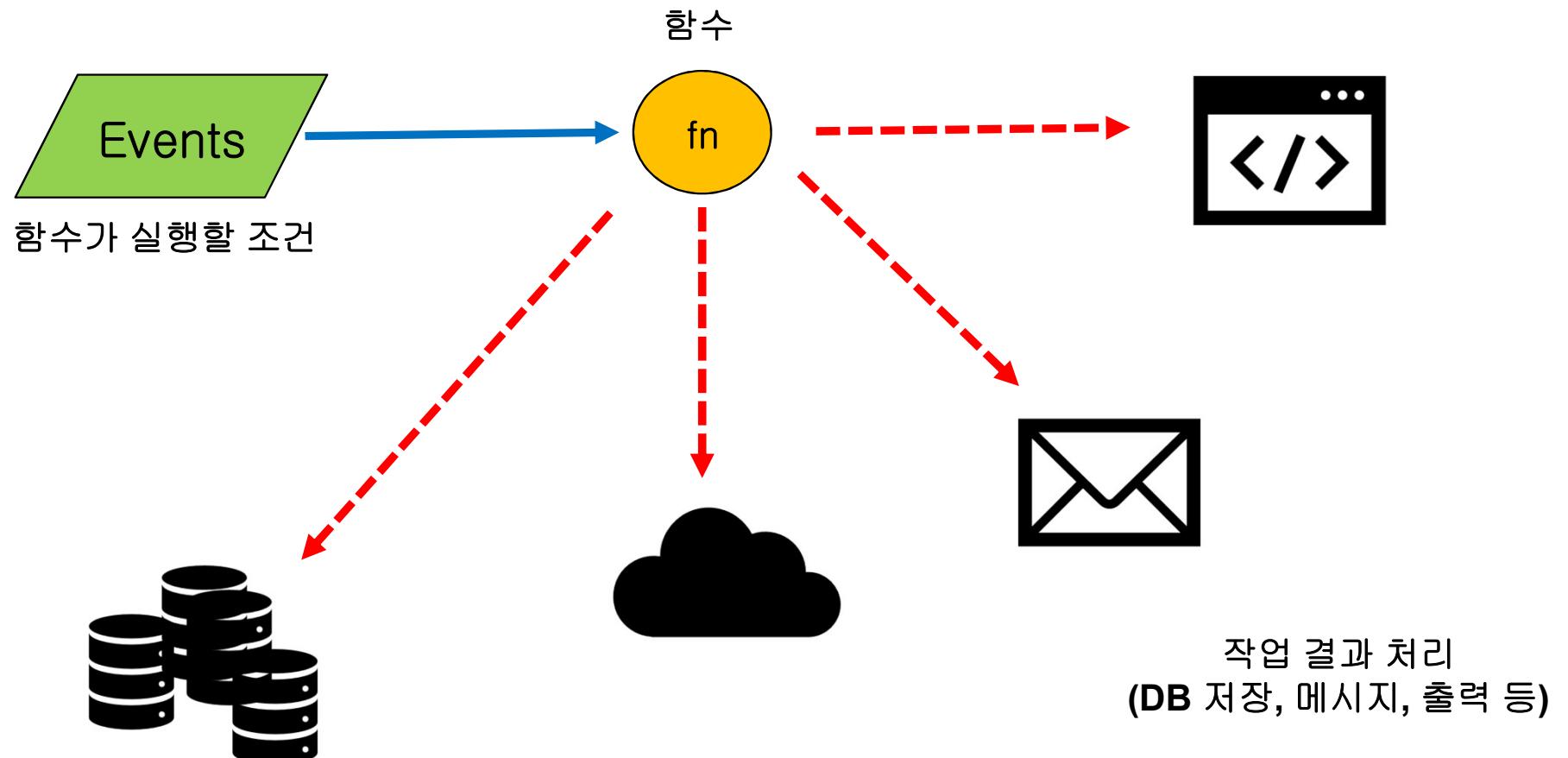


AWS Lambda

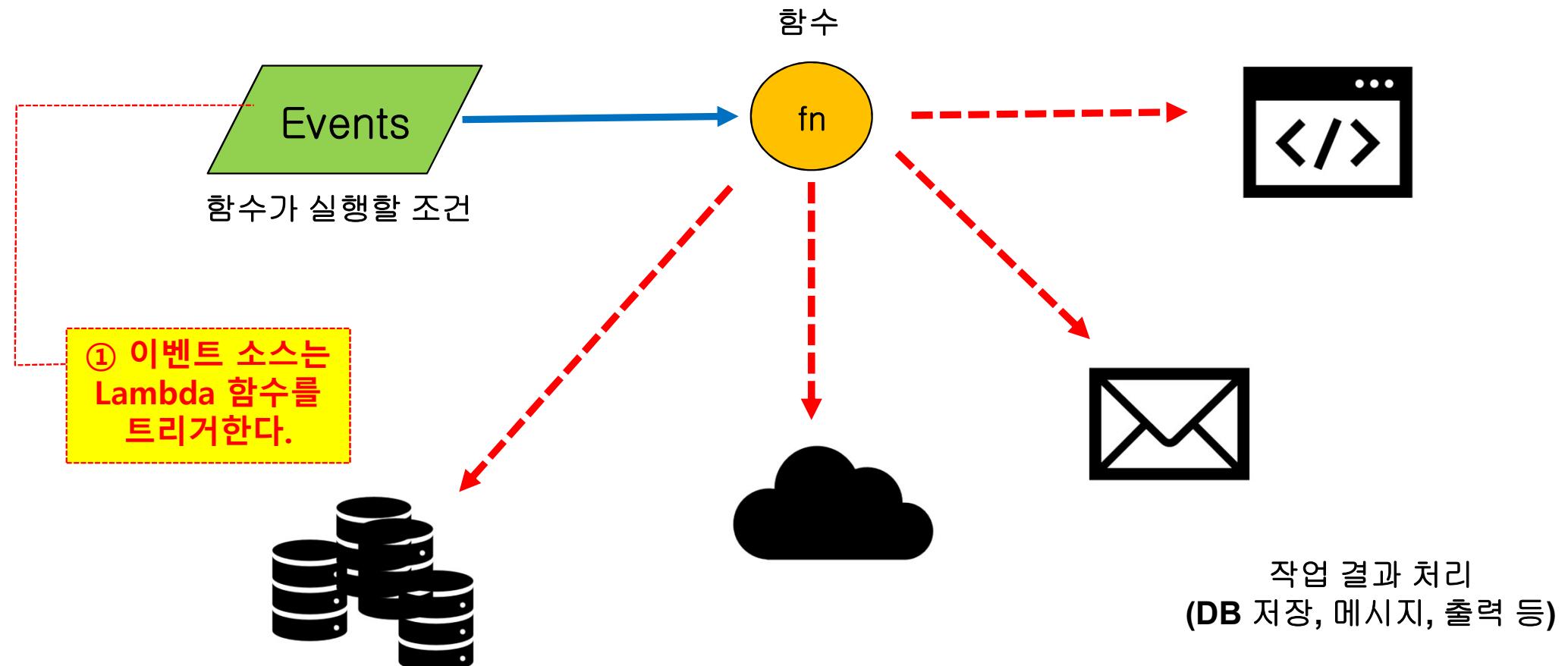


- Is an event-driven, serverless computing platform provided by Amazon as a part of Amazon Web Services.
- Is a computing service that runs code in response to events and automatically manages the computing resources required by that code.
- Was introduced on November 13, 2014.
- Node.js, Python, Java, Go, Ruby and C# (through .NET) are all officially supported as of 2018.
- Refer to [AWS Lambda Developer Guide](#)

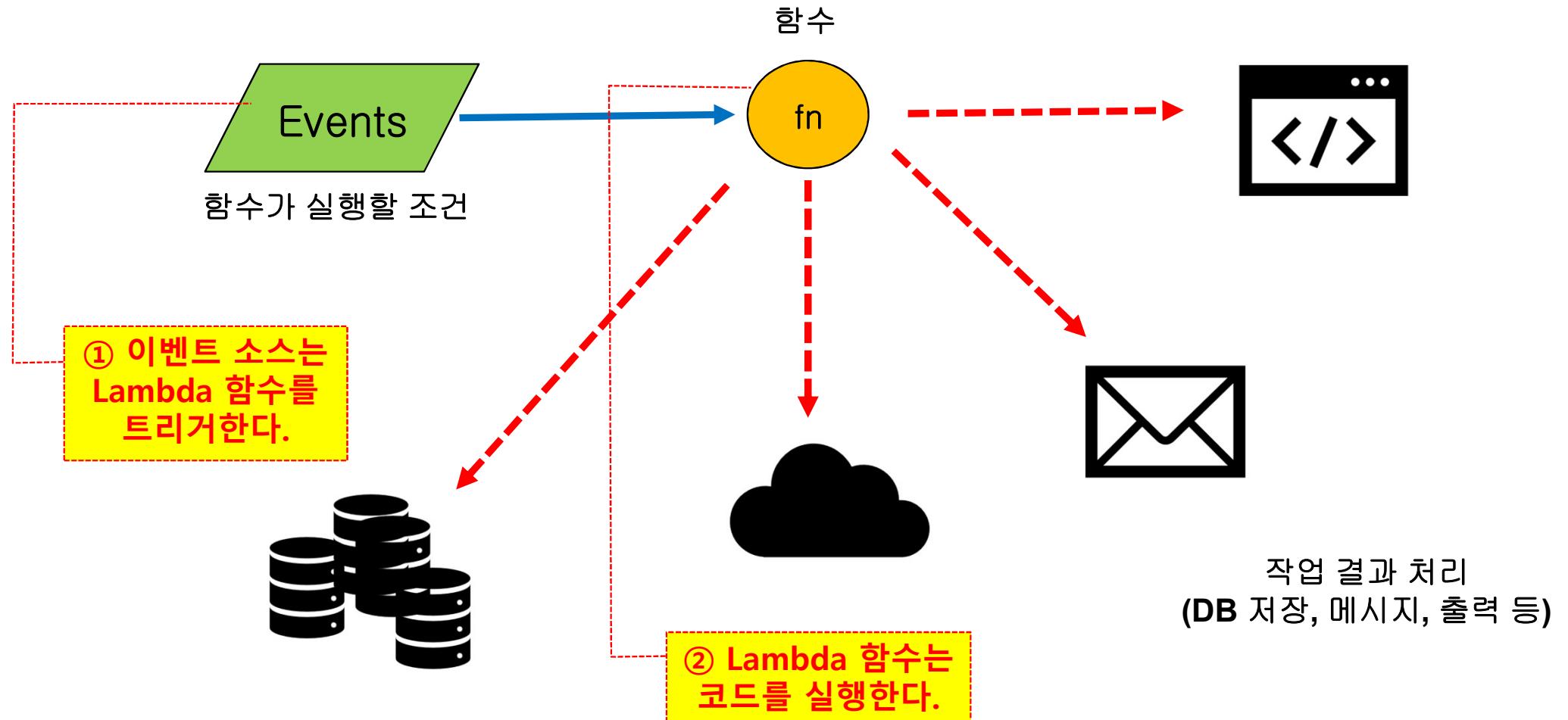
AWS Lambda Action



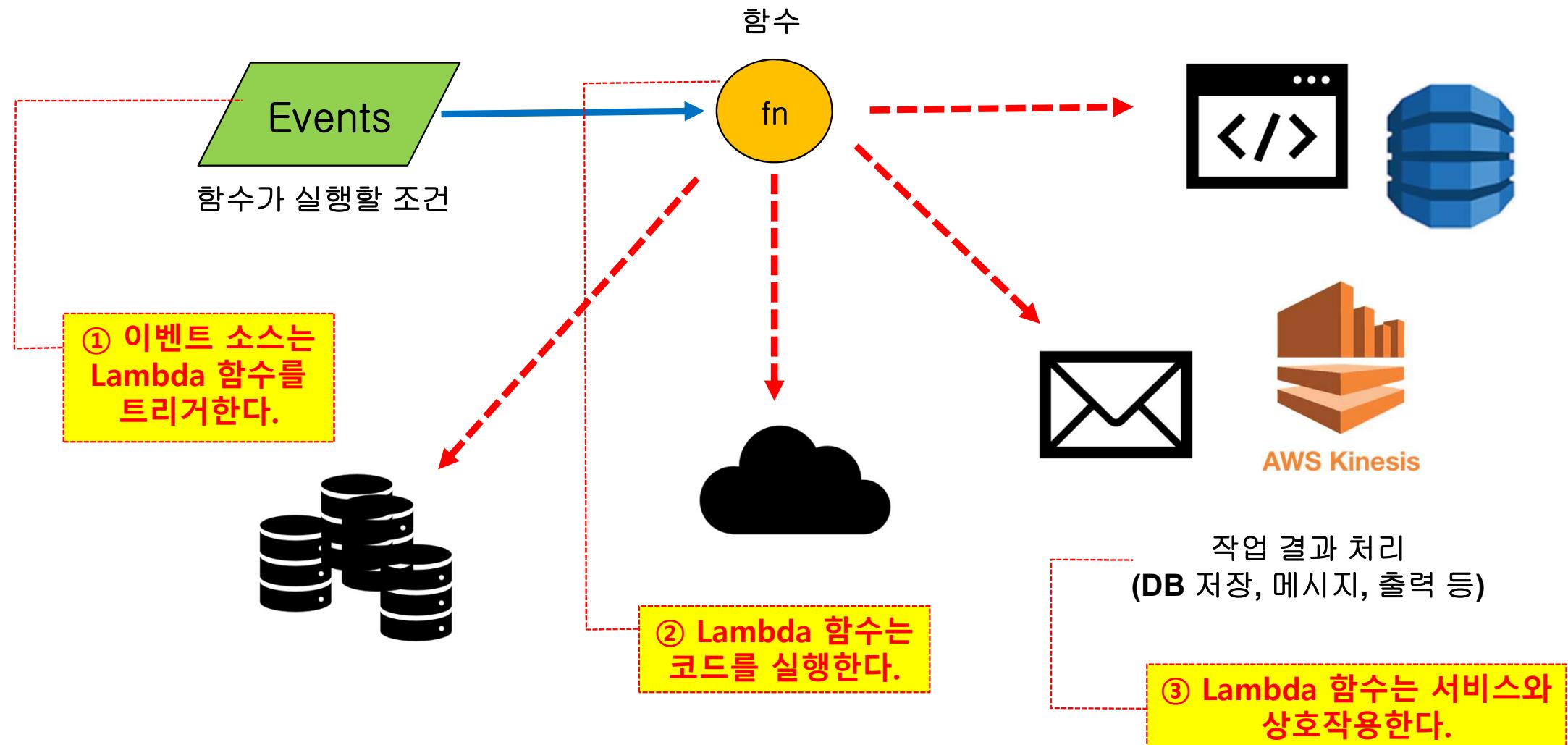
AWS Lambda Action



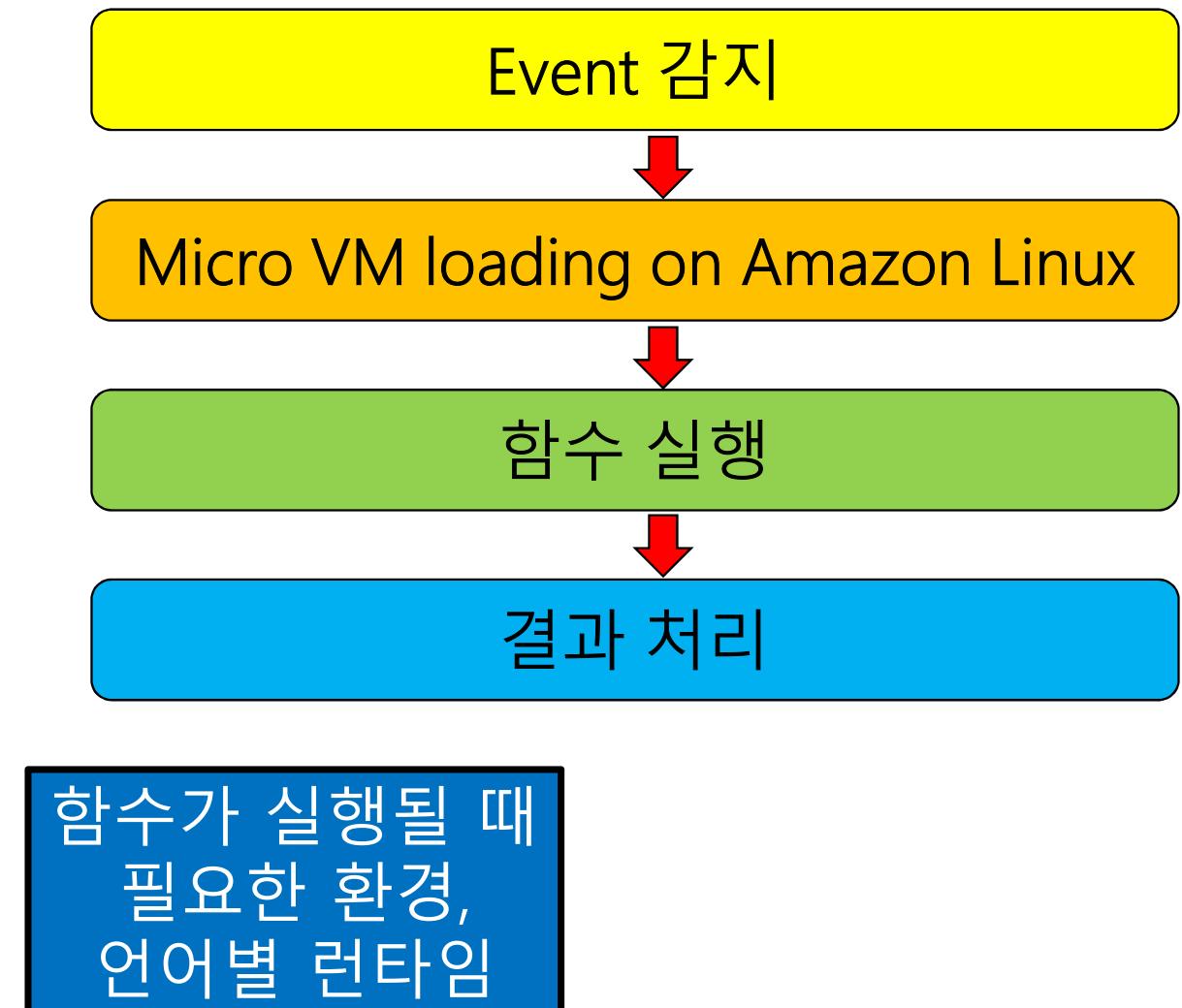
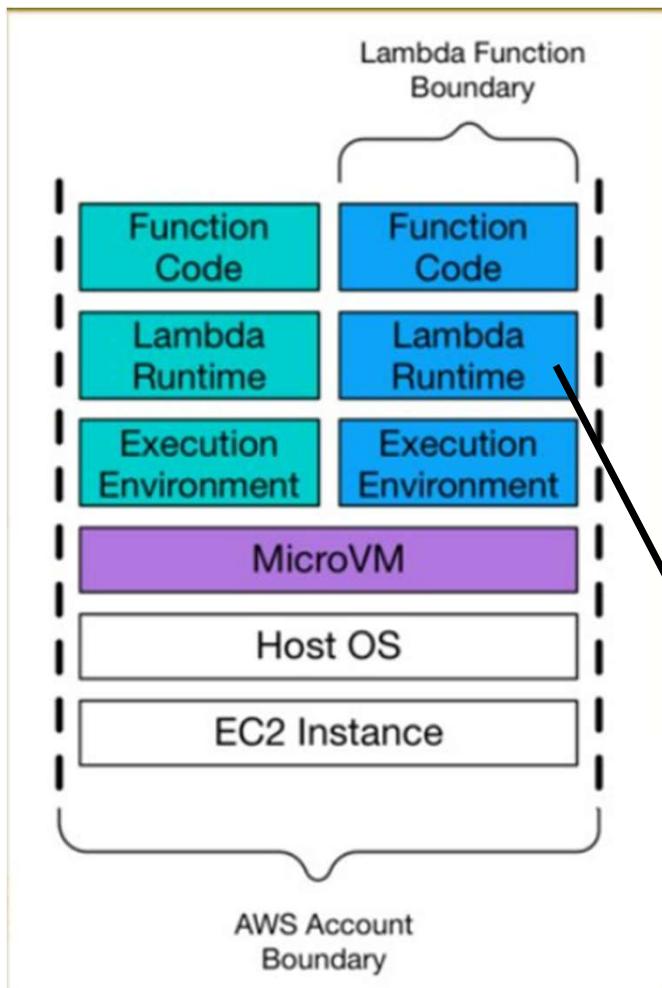
AWS Lambda Action



AWS Lambda Action



AWS Lambda 내부



AWS Lambda's Runtime

- AWS Lambda Service와 작성한 함수 코드 사이에 위치
- Event와 Context 정보 등 응답을 중계해 주는 역할
- 기본적으로 제공되는 런타임 환경과 직접 작성 가능한 필요한 런타임 환경
 - Amazon Linux 2
 - Image – Custom
 - Linux kernel – 4.14
 - Amazon Linux
 - Image – amzn-ami-hvm-2018.03.0.20220802.0-x86_64-gp2
 - Linux kernel – 4.14

<https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html>

AWS Lambda's Runtime (Cont.)

- Supported Runtimes

- Node.js
 - Node.js 18, Node.js 16, Node.js 14, Node.js 12(Deprecation at Mar 31, 2023)
- Python
 - Python 3.9, Python 3.8, Python 3.7
- Java
 - Java 11, Java 8
- .NET
 - .NET Core 3.1(Deprecation at Mar 31, 2023), .NET 6, .NET 5
- Go
 - Go 1.x
- Ruby
 - Ruby 2.7

<https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html>

AWS Lambda's (Trigger) Event

- The triggers page of AWS Lambda looks like this:

The screenshot shows the 'Triggers' page in the AWS Lambda console. At the top, there is a breadcrumb navigation 'Lambda > 추가 트리거' and a title '트리거 추가'. Below this, a sub-header '트리거 구성 정보' is visible. A search bar labeled '소스 선택' is present. A scrollable list of trigger sources is displayed, each with a small icon and a brief description. The sources listed are: Alexa, Apache Kafka, API Gateway, Application Load Balancer, AWS IoT, CloudWatch Logs, CodeCommit, Cognito Sync Trigger, DynamoDB, EventBridge(CloudWatch Events), Kinesis, and MQ.

Trigger Source	Description
Alexa	alexa iot
Apache Kafka	analytics stream
API Gateway	api application-services aws serverless
Application Load Balancer	aws load-balancing
AWS IoT	aws devices iot
CloudWatch Logs	aws logging management-tools
CodeCommit	aws developer-tools git
Cognito Sync Trigger	authentication aws identity mobile-services sync
DynamoDB	aws database nosql
EventBridge(CloudWatch Events)	aws events management-tools
Kinesis	analytics aws streaming
MQ	



Lab. Lambda, API Gateway





Lab. Lambda, API Gateway Version Control





Lab. DynamoDB, Lambda, IAM, SNS



Terms relevant to Cloud Computing



IT Resources



물리적서버



가상서버



소프트웨어



서비스

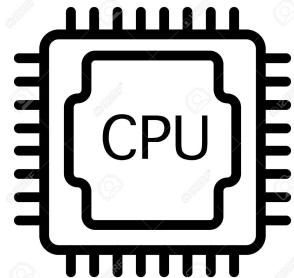


저장 장치



네트워크 장치

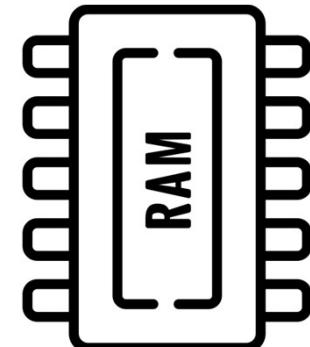
IT Resources (Cont.)



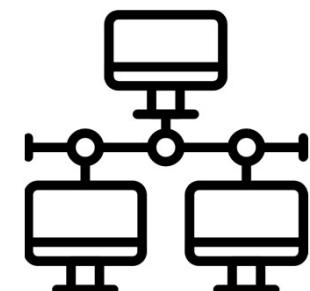
컴퓨팅 자원



네트워크 자원

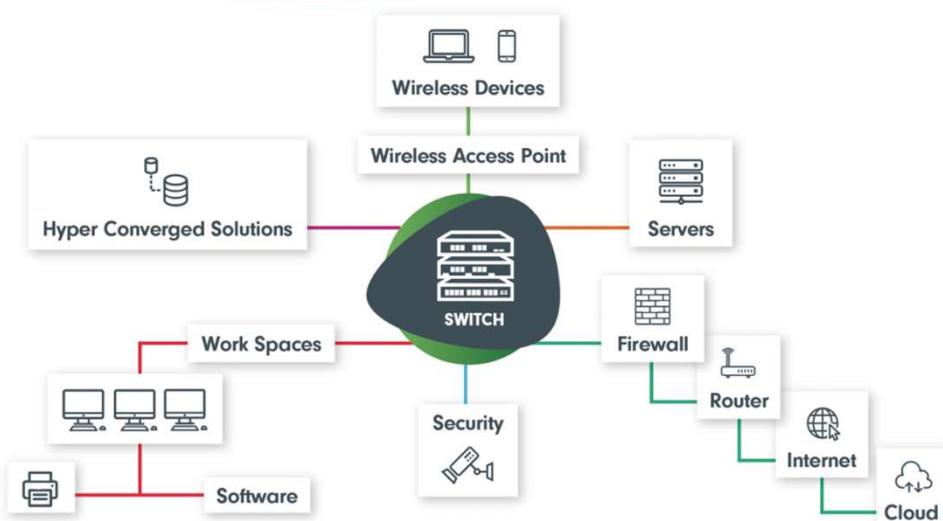


보안 자원



Infrastructure

IT Infrastructure



- 인프라
- 기반 시설
- IT 서비스의 기반이 되는
- 시스템 구조
 - 하드웨어와 네트워크 장비
 - 장비를 제어하기 위한 시스템 소프트웨어도 포함

<https://www.stonegroup.co.uk/insights/what-makes-up-an-it-infrastructure/>

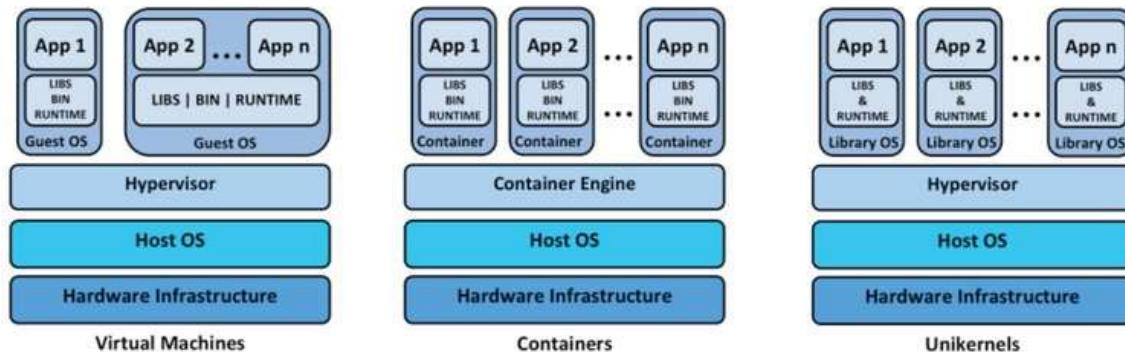
Platform



<https://medium.com/swlh/the-platform-edge-dbc541320fa7>

- 플랫폼
- 기차 플랫폼 또는 무대 강단
- 상생 생태계
- 판매자와 구매자 양쪽을
- 하나의 場으로 끌어들여
- 새로운 가치를 창출하도록 만드는 모델

Virtual Machines & Virtualization



<https://www.sdxcentral.com/edge/definitions/mec-virtualization/>

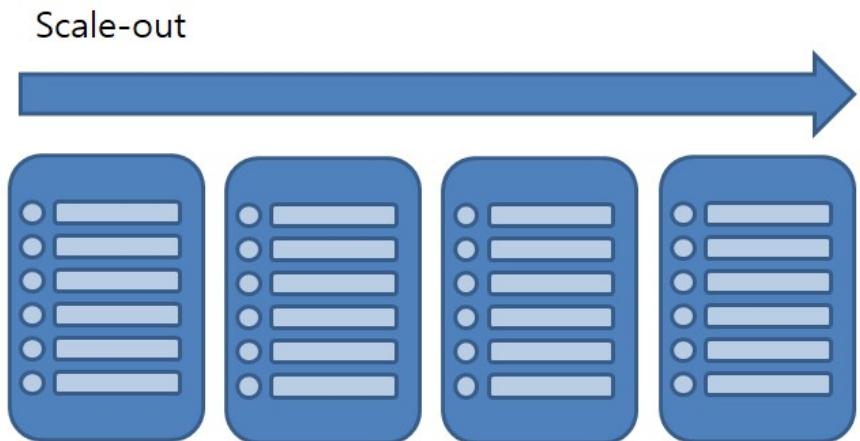
• 가상 머신

- 하나의 물리적 서버 상에
- 하이퍼바이저(Hypervisor)라는 소프트웨어가 여러 개의 가상 머신을 생성하여 제공

• 가상화

- 하나의 물리적 서버 상에
- 하나 이상의 가상 머신을 생성하여
- 복수 개의 논리적 서버를 운영하는 기술

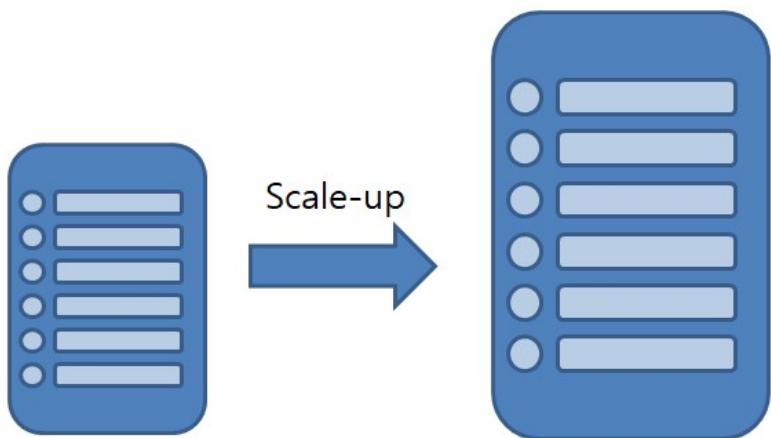
수평 확장



<https://toma0912.tistory.com/87>

- Horizontal Scaling
- 동일한 유형의 IT자원의 할당과 반납을 통한 확장과 축소
 - Scale Out : 자원의 수평적 할당
 - Scale In : 자원의 수평적 반납

수직 확장



<https://toma0912.tistory.com/87>

- Vertical Scaling
- IT 자원의 수요 증가/감소를 처리할 수 있는 능력
 - Scale Up : 기존 IT 자원을 고사양 용량의 다른 자원으로 대체한 경우
 - Scale Down : 저사양 용량의 IT자원을 다른 자원으로 대체한 경우

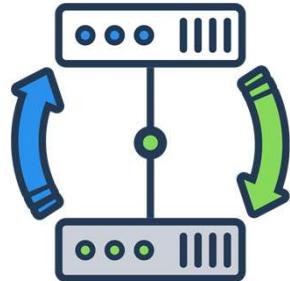
Data Center



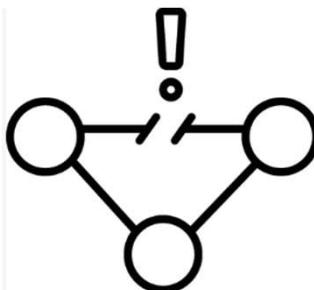
<https://www.ciokorea.com/news/39204>

- 서버 컴퓨터와 네트워크 회선 등을 제공하는 건물이나 시설
- 서버 호텔
- Application의 Server를 Hosting하는 실제 시설
- 운영에 필요한 Infra
 - Computing System을 위한 Hardware
 - Networking 장비
 - 전원공급장치
 - 전기 시스템
 - 백업 발전기
 - 환경 제어장치(에어컨, 냉각장치, 팬 등...)
 - 운영 인력
 - 기타 인프라

High Availability vs. Fault Tolerance

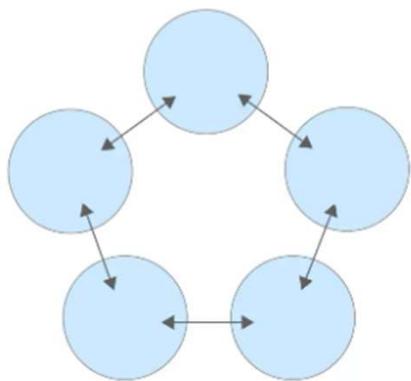


- 고가용성(High Availability)
 - 장애 상황을 해결하고 서비스를 지속할 수 있는 능력
 - 장애 상황의 준비가 되어 있는 Architecture 필요

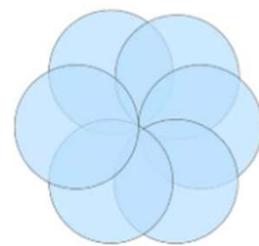


- 장애 내구성 or 내결함성(Fault Tolerance)
 - 장애 상황에도 서비스를 지속할 수 있는 능력
 - 장애 상황에 영향을 받지 않는 Architecture 필요

Tight Coupling vs. Loose Coupling



Loose Coupling



Tight Coupling

- Tight Coupling

- 다른 주체에 대해 단단하게 얹힌 상태
- 주체끼리 높은 의존성을 가지고 있어서 변경이 쉽지 않음.

- Loose Coupling

- 다른 요소에 대해 얹히지 않고 연결되어 있는 상태
- 주체끼리 낮은 의존성을 가지고 있어서 쉽게 변경할 수 있고 유연함.