

# 개발자필수지식

The Bible of Computer Science

[CS지식의 정석]

version.25.05.11

해당 교안을 무단으로 복제, 업로드, 배포, 도용할 경우

저작권법 제 97조의 저작재산권침해죄에 해당하며

5년이하의 징역 또는 5천만원 이하의 벌금에 처해질 수 있습니다.

강의에 있는 설명이 교안에 전부 들어있지 않고 요약본으로 제공되는 경우는 출판권상 책과  
중복되는 부분이여서 그렇게 제공되는 점 참고부탁드립니다.  
또한 강의에는 책과 중복되는 내용이라도 모두 다루고 있습니다. :)

## 목차

<b>데이터교환형식 #1. JSON과 직렬화와 역직렬화.....</b>	<b>6</b>
JSON.....	6
1.Javascript 객체문법.....	6
2.데이터 + 교환형식.....	6
3.여러언어에서의 쓰임.....	6
4. 단순 배열, 문자열 표현.....	7
JSON 실습#1.....	7
JSON 실습#2.....	9
JSON의 타입.....	9
JSON 직렬화,역직렬화.....	10
JSON의 활용.....	10
<b>데이터포맷 #2. XML.....</b>	<b>11</b>
마크업형태.....	11
구성.....	11
HTML과 XML 비교.....	11
HTML.....	12
JSON.....	12
XML.....	12
sitemap.xml.....	13
<b>API #1 개념.....</b>	<b>13</b>
인터페이스.....	14
ex) 삼성 갤럭시 UI.....	14
ex) 네이버 웹툰.....	14
API의 작동방식.....	15

API의 장점.....	15
API의 종류.....	16
API #2 실습 OPEN API를 이용해 온도예측하기.....	16
index.html.....	17
API #3 실습 Node.js를 이용한 간단한 API구축.....	19
API의 장점을 구현해보자.....	19
node.js 설치.....	19
express 모듈 설치.....	19
a.json.....	20
b.js.....	20
URI 분석.....	20
클라우드 #1 가상머신.....	21
전통적 배포방식.....	21
가상화 배포방식.....	21
클라우드 #2 오프프레미스, 온프레미스.....	23
클라우드.....	23
온프레미스(on-premise)방식.....	24
ex) 네이버의 데이터센터 각.....	25
클라우드 #3 IaaS, PaaS , SaaS.....	25
IaaS.....	25
PaaS.....	25
SaaS.....	26
ex) 구글DOCS.....	27
PaaS와 IaaS 비교.....	27
IaaS.....	27
PaaS.....	27
클라우드 #4 컨테이너와 도커.....	27
컨테이너.....	28
도커.....	28
도커의 활용사례.....	29
CI/CD(Continuous Integration/Delivery & Deployment).....	29

왜 필요할까?.....	29
파이프라인.....	30
빌드.....	30
테스트.....	31
머지.....	31
배포.....	32
툴.....	32
간단한 실습.....	32
<b>Q. 클래스와 객체와 인스턴스의 차이가 뭔가요?.....</b>	<b>33</b>
클래스.....	33
객체.....	33
인스턴스.....	33
실습 - java.....	34
실습 - javascript.....	35
<b>Q. static키워드는 왜 사용하며 단점은 무엇인가요?.....</b>	<b>35</b>
static.....	35
static 실습 - java.....	36
static 실습 - javascript.....	37
static 단점.....	38
<b>Q. 오버로딩과 오버라이딩은 무엇인가요?.....</b>	<b>38</b>
오버로딩(Overloading).....	38
오버라이딩(Overriding).....	40
<b>Q. 추상화란 무엇인가요?.....</b>	<b>41</b>
추상화의 의미.....	41
데이터 추상화.....	41
프로세스 추상화.....	42
ex1) Animal Class.....	42
ex2) MySQL 아키텍처.....	43
<b>Q. 컴파일러와 인터프리터의 차이가 무엇인가요?.....</b>	<b>43</b>
컴파일러.....	44
인터프리터.....	44

JIT 컴파일러.....	45
---------------	----

# 데이터교환형식 #1. JSON과 직렬화와 역직렬화

## JSON

JSON(JavaScript Object Notation)은 Javascript 객체 문법으로 구조화된 데이터교환형식, python, javascript, java 등 여러 언어에서 데이터 교환형식으로 쓰이며 객체문법 말고도 단순 배열, 문자열도 표현 가능합니다.

### 1. Javascript 객체문법

키(key)과 값(value)으로 구성됩니다.

ex) {key : value}

이미 존재하는 키를 중복선언하면 나중에 선언한 해당 키에 대응한 값이 덮어쓰이게 됩니다.

```
{  
    "name" : "kundol",  
    "name" : "king",  
    "name" : "king"  
}
```

```
const fs = require('fs')  
const path = require('path')  
const a = fs.readFileSync(path.join(__dirname, "a.json"))  
console.log(a)  
console.log(JSON.parse(a))
```

### 2. 데이터 + 교환형식

데이터는 추상적인 아이디어에서부터 시작해 구체적인 측정에 이르기까지 다양한 의미로 쓰입니다. 실험, 조사, 관찰 등으로 부터 얻은 사실이나 자료 등을 의미합니다.

### 3. 여러언어에서의 쓰임

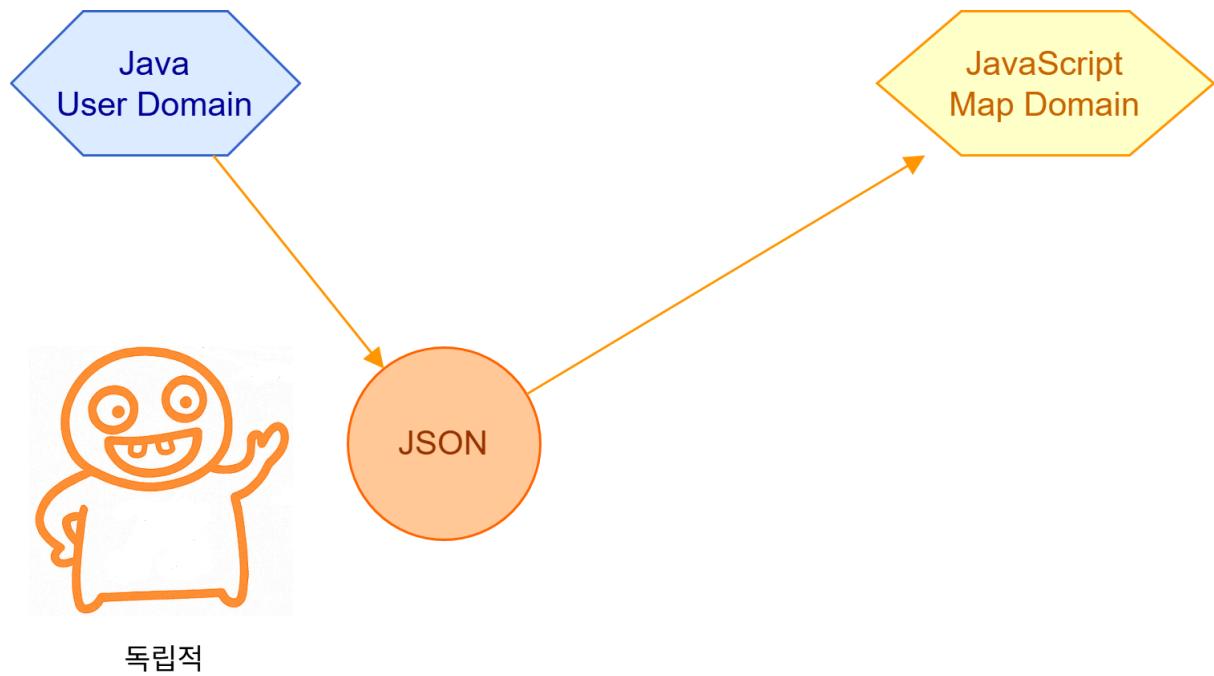
객체, 해시테이블, 딕셔너리 등으로 변환되어 쓰임

ex)

json in javascript = javascript object

json in python = dict

...



[https://www.w3schools.com/python/trypython.asp?filename=demo\\_json](https://www.w3schools.com/python/trypython.asp?filename=demo_json)

#### 4. 단순 배열, 문자열 표현

ex)

단순배열 : [1, 2, 3, 4]

문자열 : “파카파카파카”

<https://www.json.org/json-en.html>

### JSON 실습#1

지브리 OST리스트

- 마녀 배달부 키키 / 따스함에 둘러쌓인다면
- 하울의 움직이는 성 / 세계의 약속

작품명 : 노래 >> name : song

지브리 OST 리스트 : <https://www.youtube.com/watch?v=alLs9S4pwo0>

```
{  
    "지브리OST리스트" : [  
        {  
            "name" : "마녀 배달부 키키",  
            "song" : "따스함에 둘러쌓인다면"  
        },  
        {  
            "name" : "하울의 움직이는 성",  
            "song" : "세계의 약속"  
        }  
    ]  
}
```

앞의 코드처럼 표현이 가능합니다. 배열은 []로 감싸서 표현하죠? JSON도 동일합니다.  
여러개의 지브리OST가 있기 때문에 해당 부분을 대괄호를 기반으로 표현합니다.  
자 근데 여기서 각 객체는 다른 타입을 가지고 있어도 괜찮습니다.

```
{  
    "지브리OST리스트" : [  
        {  
            "name" : "마녀 배달부 키키",  
            "song" : "따스함에 둘러쌓인다면"  
        },  
        {  
            "name" : "하울의 움직이는 성",  
            "song" : 1  
        }  
    ]  
}
```

즉 이렇게 song이 string 또는 1이 되도 상관이 없습니다.  
(하지만 타입은 맞춰주는게 좋습니다.)

```
const a = {  
    "지브리OST리스트" : [  
        {  
            "name" : "마녀 배달부 키키",  
            "song" : "따스함에 둘러쌓인다면"  
        },  
        {  
            "name" : "하울의 움직이는 성",  
            "song" :  
        }  
    ]  
}
```

```

        "song" : "세계의 약속"
    }
]
}
console.log(a.지브리OST리스트[0])
console.log(a.지브리OST리스트[0].name)
console.log(a.지브리OST리스트[0]["song"])
/*
{ name: '마녀 배달부 키키', song: '따스함에 둘러쌓인다면' }
마녀 배달부 키키
따스함에 둘러쌓인다면
*/

```

앞의 코드처럼 배열은 [0], [1] 이런식으로 접근하면 되고 해당 key에 대한 value는 .key 또는 ["key"] 이런식으로 접근해서 빼내면 됩니다.

## JSON 실습#2

이름 : 쁘돌

좋아하는 것 : 아령, 바나나

아령 : 10kg, 육각형

바나나 : 초록색

```
{
  "name" : "kundol",
  "like" : {
    "아령" : {
      "weight" : "10kg",
      "feature" : "육각형"
    },
    "바나나" : {
      "color" : "초록색"
    }
  }
}
```

## JSON의 타입

javascript object와 유사합니다만 undefined, 메서드 등을 포함할 수 없습니다.

- 수(Number)

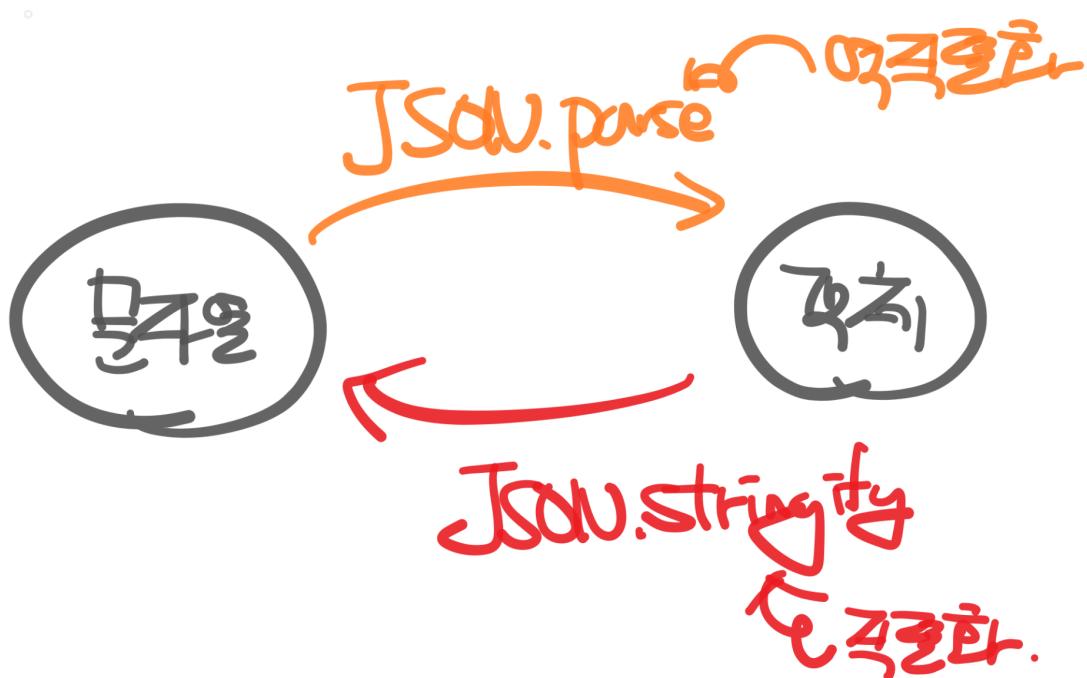
- 문자열(String):
- 참/거짓(Boolean)
- 배열(Array)
- 객체(Object)
- null

## JSON 직렬화, 역직렬화

직렬화란 외부의 시스템에서도 사용할 수 있도록 바이트(byte) 형태로 데이터를 변환하는 기술이며 역직렬화는 반대를 의미

JSON.parse()

JSON.stringify()



## JSON의 활용

JSON은 프로그래밍 언어와 프레임워크 등에 독립적이므로, 서로 다른 시스템간에 데이터를 교환하기에 좋습니다. 주로 API의 반환형태, 시스템을 구성하는 설정파일에 활용됩니다.

ex) 업비트의 API

<https://docs.upbit.com/reference>

ex) package.json

<https://docs.npmjs.com/cli/v9/configuring-npm/package-json>

## 데이터포맷 #2. XML

XML(Extensible Markup Language)은 마크업 형태를 쓰는 데이터교환형식입니다.

### 마크업형태

마크업(markup)은 태그 등을 이용하여 문서나 데이터의 구조를 나타내는 방법입니다.  
(속성부여도 가능)

### 구성

1. 프롤로그 : 버전, 인코딩
2. 루트요소(단 하나만)
3. 하위 요소들

```
<?xml version="1.0" encoding="UTF-8"?>
<OSTList>
    <OST like="1">
        <name>마녀 배달부 키키</name> <song>따스함에 둘러쌓인다면</song>
    </OST>
    <OST like="2">
        <name>하울의 움직이는 성</name> <song>세계의 약속</song>
    </OST>
</OSTList>
```

## HTML과 XML 비교

- 1.HTML의 용도는 데이터를 표시 / XML은 데이터를 저장 및 전송
- 2.HTML에는 미리 정의된 태그가 있지만 사용자는 XML에서 고유한 태그를 만들고 정의 가능
- 3.XML은 대/소문자를 구분하지만 HTML은 구분하지 않습니다. <book> 대신 <Book>으로 태그를 작성하면 XML 구문 분석기에서 오류가 발생합니다.

## HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <p></p>
    <div></div>
</body>
</html>
```

[참고] html 자체적으로는 고유한 태그를 만들지는 못하지만, 자바스크립트의 웹 컴포넌트(Web Components) API를 활용하면 고유한 태그를 만들 수 있음.

```
<!DOCTYPE html>
<html lang="ko">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Custom Element Example</title>
</head>

<body>
    <my-element data-name="큰돌"></my-element>
    <my-element data-name="CS"></my-element>
    <script>
        class MyElement extends HTMLElement {
            connectedCallback() {
                const name = this.getAttribute('data-name') ||
'이름 없음';
                this.innerHTML =
                    <div style="border: 1px solid #ccc; padding: 10px;
margin: 5px;">
                        <h3>안녕하세요, ${name}님!</h3>
                        <p>이것은 커스텀 엘리먼트 예제입니다.</p>
                    </div>
            }
        }
    </script>
</body>
</html>
```

```

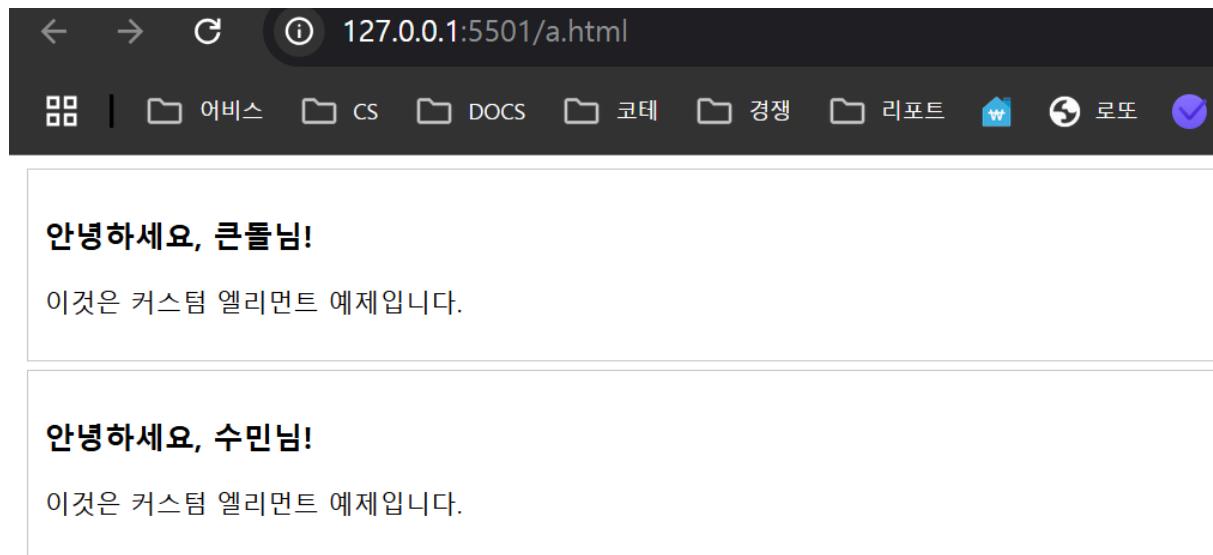
        </div>
    ;
}

}

// 커스텀 엘리먼트 등록
customElements.define('my-element', MyElement);
</script>
</body>

</html>

```



## JSON과 XML 비교

JSON과 비교했을 때 닫힌 태그가 계속해서 들어가기 때문에 JSON과 비교하면 무겁습니다. 또한 Javascript Object로 변환하기 위해서 JSON보다는 더 많은 노력이 필요합니다.(JSON은 JSON.parse() 면 됩니다.)

### JSON

```
{
    "지브리OST리스트" : [
        {
            "name" : "마녀 배달부 키키",
            "song" : "따스함에 둘러쌓인다면"
        },
    ]
}
```

```
        {
            "name" : "하울의 움직이는 성",
            "song" : "세계의 약속"
        }
    ]
}
```

## XML

```
<?xml version="1.0" encoding="UTF-8"?>
<OSTList>
    <OST>
        <name>마녀 배달부 키키</name> <song>따스함에 둘러쌓인다면</song>
    </OST>
    <OST>
        <name>하울의 움직이는 성</name> <song>세계의 약속</song>
    </OST>
</OSTList>
```

```
const fs = require('fs')
const path = require('path')
var parser = require('xml2json');
let a = fs.readFileSync(path.join(__dirname, "a.xml"))
a = parser.toJson(a)
console.log(a)
```

## sitemap.xml

xml은 대표적으로 sitemap.xml에 쓰입니다.

sitemap.xml은 서비스 내의 모든 페이지들을 리스트업한 데이터입니다.

사이트가 매우 크거나 서로 링크가 종속적으로 연결되지 않은 경우 크롤러가 일부 페이지를 누락하는 일이 있는데 이를 sitemap.xml이 방지하고 모든 페이지들을 크롤링할 수 있도록 해줍니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
    <url>
        <loc>http://www.example.com/foo.html</loc>
        <lastmod>2018-06-04</lastmod>
    </url>
```

```
<url>
  <loc>http://www.example.com/abc.html</loc>
  <lastmod>2018-06-04</lastmod>
</url>
</urlset>
```

## API #1 개념

API (API, Application Programming Interface)는 둘 이상의 컴퓨터 프로그램이 서로 통신하는 방법이자 컴퓨터 사이에 있는 중계 계층을 의미합니다.

예를 들어 A라는 컴퓨터가 요청을 하고 B라는 컴퓨터가 응답을 한다고 했을 때의 어떻게 통신할 것인지, 어떠한 데이터를 주고 받을 건지 등에 대한 방법(HTTP, HTTPS 프로토콜을 사용할 것인지, GET, POST 등의 방식 등..)이 정의된 중계계층을 말합니다.

참고로 API는 과거부터 발전되어온 용어로 라이브러리 및 프레임워크를 설명하는 명세서, 웹상에서 WEB API, Web Socket API 등을 가리키는데 현재를 기준으로 API라고 할 때 보통 WEB API를 기준으로 설명합니다.

## 인터페이스

인터페이스(interface)는 서로 다른 두 개의 시스템, 장치 사이에서 정보나 신호를 주고받는 경우의 접점이나 경계면입니다. 이를 통해 해당 컴퓨터의 내부서버가 어떻게 구현되어있는지는 상관없이 인터페이스를 통해 통신 등이 가능합니다.

### ex) 삼성 갤럭시 UI

사용자 인터페이스(UI)를 생각해볼까요? UI는 컴퓨터와 사람을 연결합니다. 다음 그림은 삼성 갤럭시의 UI입니다.



이러한 핸드폰의 화면을 기반으로 사용자는 휴대폰과의 상호작용을 할 수 있습니다. 앱을 실행하거나 등을 할 수 있는 것이죠.

### ex) 네이버 웹툰

저희는 네이버의 웹툰의 서버가 어떻게 되어있는지. 데이터베이스가 어떻게 되어있는지 알지 못합니다. 그러나 이러한 인터페이스를 기반으로 웹툰의 서비스를 즐길 수 있습니다.

요일전체 월요웹툰 화요웹툰 수요웹툰 목요웹툰 **금요웹툰** 토요웹툰 일요웹툰 매일+웹툰

금요 추천 웹툰 TODAY : 2022.09.16



절대복종  
4화 - 넌 내꺼야!  
흔



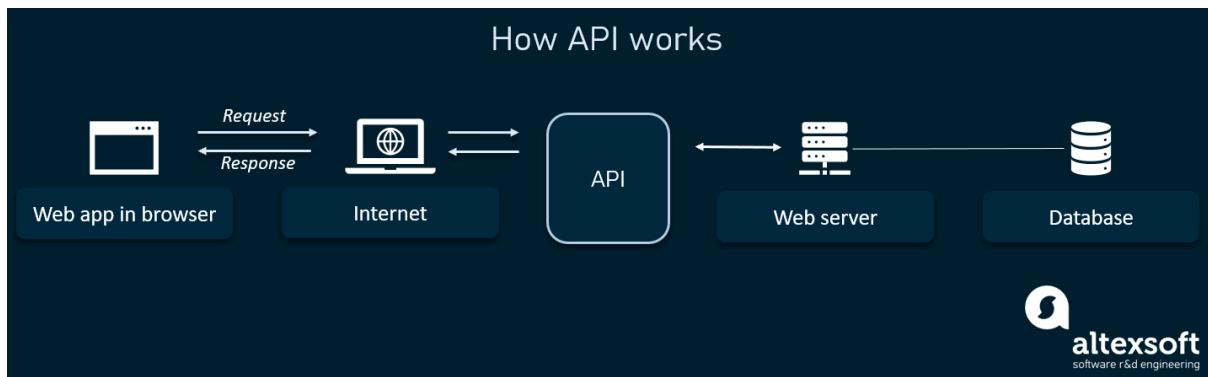
여름의 너에게  
5화  
라라뮤



지옥 키우기  
1화  
세하

## API의 작동방식

API는 다음과 같이 중계계층으로 끼어져있어서 작동을 합니다.



사용자가 브라우저를 통해서 서버에 요청을 하게 되면 API가 중계계층역할을 하며 요청을 처리하는 것을 볼 수 있습니다. 직접 서버의 데이터베이스에 접근 하는 것을 방지하기도 하는 것을 볼 수 있습니다.

ex) <https://www.naver.com/>

네이버의 화면은 여러 API들의 응답값 등으로 이루어져 있습니다.

## API의 장점

- 제공자는 서비스의 중요한 부분을 드러내지 않아도 됩니다. 예를 들어 DB 설계 구조나 드러내고 싶지 않은 데이터베이스의 테이블 정보, 서버의 상수값 등을 드러내지 않고 드러내고 싶은 부분만을 드러낼 수 있습니다.
- 사용자는 해당 서비스가 어떻게 구현되는지 알 필요없이 필요한 정보만을 받을 수 있습니다.
- OPEN API의 경우 앱 개발 프로세스를 단순화 시키고 시간과 비용을 절약할 수 있습니다. ex) 로그인 : <https://developers.naver.com/products/login/api/api.md>
- 내부 프로세스가 수정되었을 때 API를 매번 수정하는 것이 아닌 API가 수정이 안 되게 만들 수 있습니다. 이를 통해 내부 DB, 서버의 로직이 변경이 되어도 매번 사용자가 앱을 업데이트하는 일은 줄어들 수 있습니다. ex) DB튜닝
- 제공자는 데이터를 한곳에 모을 수 있습니다. 예를 들어 콘돌예스24라는 책을 파는 쇼핑몰을 만들었다고 하면 해당 사이트에 방문하는 방문자, 어떤 특정한 것을 클릭하는 사용자에 대한 이벤트를 집계하고 싶을 때 해당 API를 만들고 해당 이벤트가 발생하면 해당 API를 호출하게 만들면 해당 데이터를 한 곳에 모을 수 있습니다. ex) yes24의 베스트셀러, 검색페이지에서의 사용자이벤트

## API의 종류

- private : 내부적으로 사용됩니다. 주로 해시키를 하드코딩해놓고 이를 기반으로 서버와 서버간의 통신합니다. 이는 비즈니스 파트너와도 사용될 수 있습니다. 비밀스럽게 해당 파트너와 해시키를 공유해 통신합니다.
- public : 모든 사람이 사용할 수 있습니다. 많은 트래픽을 방지하기 위해 하루 요청수의 제한, 계정당 몇개 등으로 관리합니다.

## 네이버 오픈 API 목록 [🔗](#)

네이버 오픈API 목록 및 안내입니다.

API명	설명	호출제한
검색	네이버 블로그, 이미지, 웹, 뉴스, 백과사전, 책, 카페, 지식iN 등 검색	25,000회/일
네이버 로그인	외부 사이트에서 네이버 로그인 기능 구현	없음

네이버 검색의 경우 일일 25000회로 제한을 걸어놓은 모습.

<https://help.naver.com/service/30015/contents/17309?lang=ko>

## API #2 실습 OPEN API를 이용해 온도예측하기

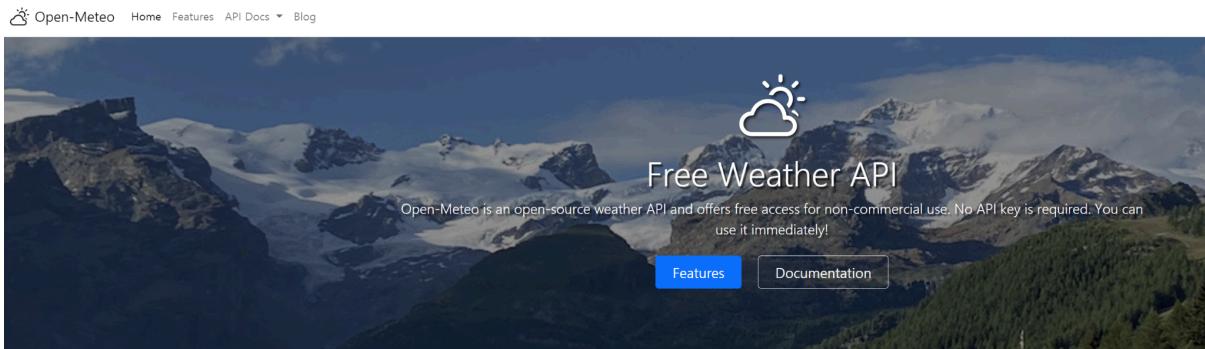
날씨예측이라는 것은 원래 수치모델, 인공지능모델이 필요함.

그런거 없이 OPEN API를 통해서 만들 수 있음.

소스코드 : <https://github.com/wnghdcjfe/csnote/tree/main/ch0>

<https://www.chartjs.org/>

[https://open-meteo.com/en/docs#latitude=37.57&longitude=126.98&hourly=temperature\\_2m&current\\_weather=true&timezone=Asia%2FTokyo](https://open-meteo.com/en/docs#latitude=37.57&longitude=126.98&hourly=temperature_2m&current_weather=true&timezone=Asia%2FTokyo)



## index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
    <style>
        canvas {
            width: 100% !important;
        }
    </style>
</head>

<body>
    <canvas id="myChart"></canvas>
</body>
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
<script>
    const OPEN_API =
"https://api.open-meteo.com/v1/forecast?latitude=37.57&longitude=1
26.98&hourly=temperature_2m&current_weather=true&timezone=Asia%2FT
okyo"
    const draw = (res) => {
        const opt = {
            year: 'numeric',
            month: 'numeric',
            day: 'numeric',
            hour: 'numeric'
        }
    }
</script>
```

```

    res.hourly.time = res.hourly.time.map(e => {
      return new Intl.DateTimeFormat("ko-KR", opt).format(new
Date(e))
    })
    const data = {
      labels: res.hourly.time,
      datasets: [{
        label: '서울의 온도차트',
        data: res.hourly.temperature_2m,
        borderColor: 'rgb(255, 99, 132)',
        backgroundColor: 'rgba(255, 99, 132, 0.5)',
        pointStyle: 'circle',
        pointRadius: 10,
        pointHoverRadius: 15
      }]
    }
    const ctx =
document.getElementById('myChart').getContext('2d');
    const myChart = new Chart(ctx, {
      type: 'line',
      data: data
    });
  }
  window.onload = async () => {
    const ret = await fetch(OPEN_API).then(res => res.json())
    draw(ret)
  }
/*
 Load : 스타일시트 및 이미지와 같은 모든 종속 리소스를 포함하여 전체
페이지가 로드될 때를 말합니다.
*/
</script>

</html>

```

## API #3 실습 Node.js를 이용한 간단한 API구축

### API의 장점을 구현해보자.

제공자의 경우 API를 만들게 되면 내부 프로세스가 수정되었을 때 매번 수정하는 것이 아닌 API가 수정이 안되게끔 만들 수 있다. 또한 내부가 변경이 되어도 사용자에게 영향을 주지 않고 변경이 가능합니다.

### node.js 설치

Node.js는 비동기적 이벤트 주도 방식, 논블로킹 I/O 모델을 사용하는 구글의 V8 엔진을 장착한 자바스크립트 런타임입니다.

런타임이란 프로그램이 실행될 때 그 프로그램이 머무는 공간을 의미합니다. 브라우저는 자바스크립트 런타임이기도 합니다. 브라우저라는 공간 안에서 자바스크립트로 만든 프로그램을 실행할 수 있기 때문입니다. 이와 동일하게 Node.js 또한 자바스크립트 런타임이므로 자바스크립트로 만든 프로그램을 실행할 수 있습니다. 다시 말해, 자바스크립트로 만든 게임, 알고리즘, 서버 등 많은 것들을 실행할 수 있습니다. 그중 우리가 이번에 만들고 실행해볼 것은 서버입니다.

<https://nodejs.org/en/>



Node.js® is an open-source, cross-platform JavaScript runtime environment.

Node.js assessment of OpenSSL 3.0.7 security advisory

Download for Windows (x64)

18.12.1 LTS

Recommended For Most Users

19.3.0 Current

Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#)    [Other Downloads](#) | [Changelog](#) | [API Docs](#)

### express 모듈 설치

```
npm install express
```

express는 Node.js에서 동작하는 웹 프레임워크 중 많이 사용되며 라우팅 설정, 미들웨어 설정, 정적자원서버 설정 등이 쉬운 프레임워크입니다.

라우팅이란 URI(또는 경로) 및 특정한 HTTP 요청 메소드(GET, POST 등)인 특정 엔드포인트에 대한 클라이언트 요청에 애플리케이션이 응답하는 방법을 결정하는 것을 말함.

### a.json

```
{  
    "name" : "kundol",  
    "tall" : "200"  
}
```

### b.js

```
const express = require('express')  
const app = express()  
const port = 3000  
const fs = require('fs')  
app.get('/', (req, res) => {  
    const f = JSON.parse(fs.readFileSync("a.json",  
{encoding:"utf-8"}))  
    const data = {  
        "name" : f.name  
    }  
    res.send(data)  
})  
  
app.listen(port, () => {  
    console.log(`http://127.0.0.1:${port}`)  
})
```

## URI 분석

http://127.0.0.1:3000

프로토콜 : HTTP

IP : 127.0.0.1 (루프백IP)

port : 3000

port는 사실 숨겨져있습니다.

https port : 443이 기본포트

http port : 80이 기본포트

ex) www.naver.com

# 클라우드 #1 가상머신

클라우드를 배워보기 앞서 클라우드의 기반 기술인 가상머신에 대해 배워보겠습니다.

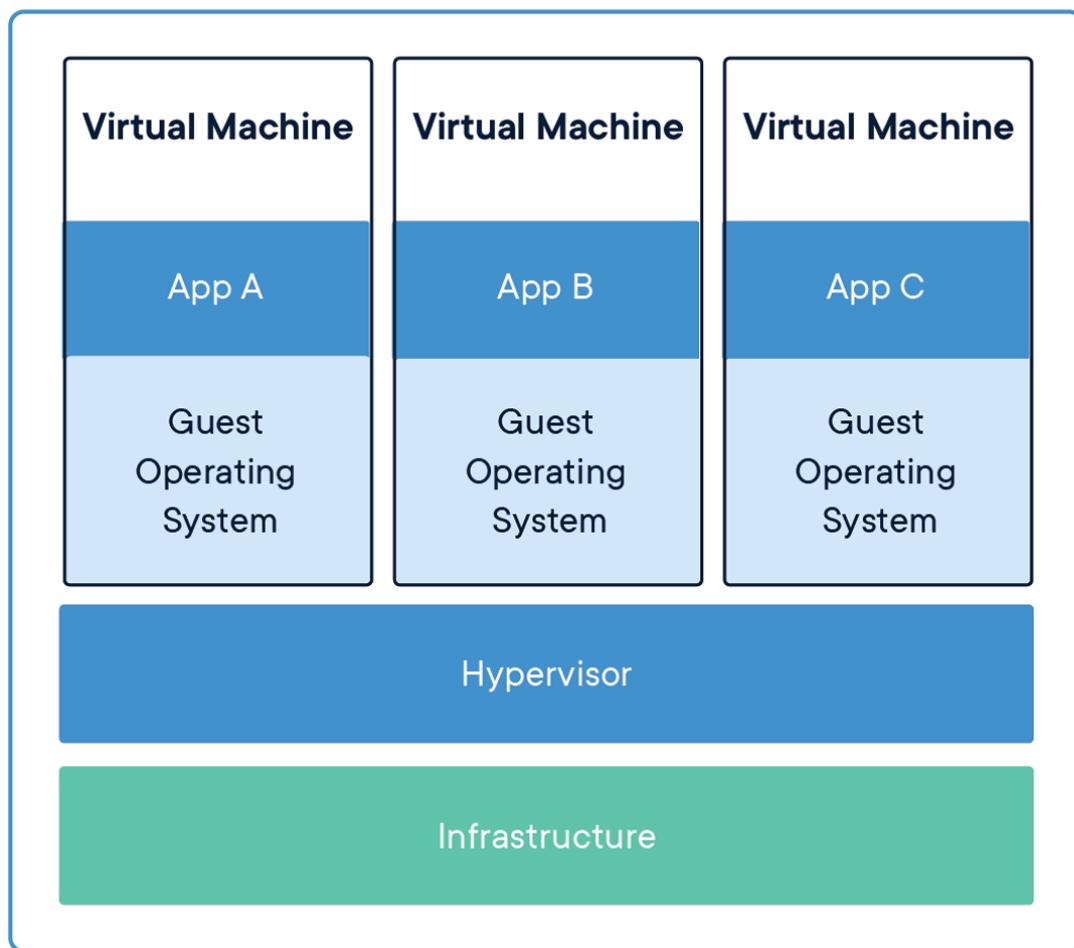
## 전통적 배포방식

물리적인 컴퓨터 한 대에 하나의 OS를 깔고 여러 가지 프로그램을 설치하는 방식. 계정을 나눠 여러명의 사용자가 이용할 수 있도록 할 수 있지만 어떤 프로그램을 설치했을 때 다른 앱에 영향을 미칩니다.

## 가상화 배포방식

가상머신을 기반으로 배포하는 것을 말합니다. 가상머신이란 컴퓨터의 하드웨어를 소프트웨어적으로 구현한 것을 말합니다.

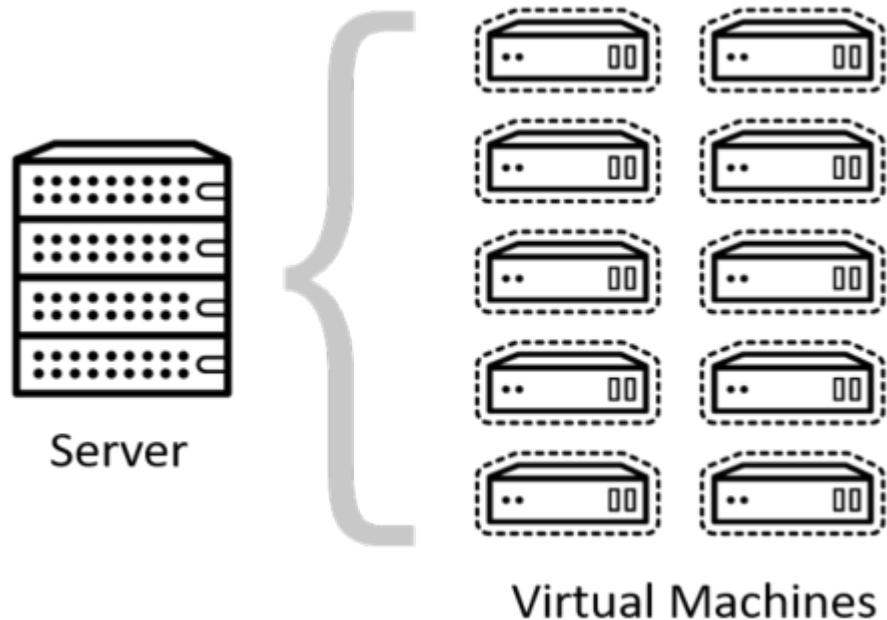
계정을 나누는 것이 아니라 한대의 컴퓨터를 가지고 여러개의 OS를 구동할 수 있게 되며 CPU, RAM을 물리적으로 갈아끼는 것이 아니라 설정만으로 이를 수행할 수 있게 되었습니다.



중간에 있는 하이퍼바이저는 하나의 시스템 상에서 가상 컴퓨터를 여러 개 구동할 수 있도록

해 주는 중간 계층을 의미하며 이 위에 여러개의 가상머신을 구축할 수 있고 가상머신 위에 OS 그리고 그 위에 앱이 올라가는 형태로 가상머신을 독립적으로 수행할 수 있습니다.

클라우드는 이러한 가상화라는 기술 때문에 한대의 하드웨어로 여러명의 사용자들에게 독립적으로 클라우드 서비스를 할 수 있습니다.



이렇게 독립적으로 가상머신이 구축되어 서로 전혀 상호작용하지 않으며 한 가상 머신위의 프로그램은 다른 가상머신위의 프로그램에서 볼 수 없는 형태를 샌드박스되었다라고도 합니다.

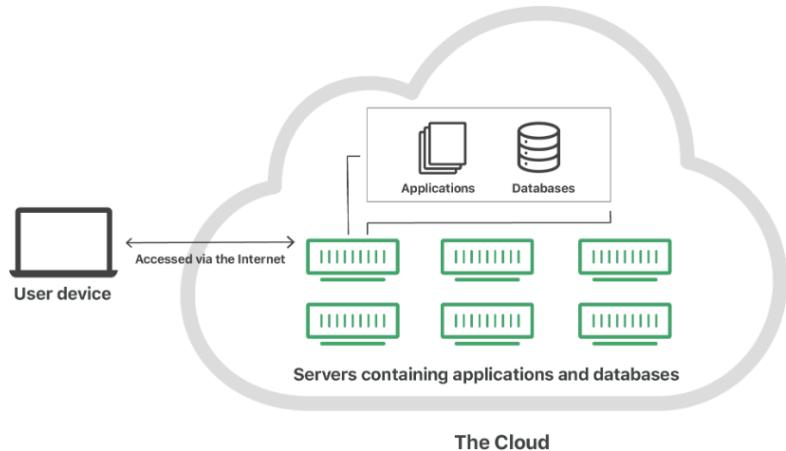
다만 OS가 공유가 안되기 때문에 가상머신에 일일히 OS를 설치해야 하는 단점이 있습니다.

## 클라우드 #2 오프프레미스, 온프레미스

### 클라우드

클라우드 서비스는 내가 아닌 다른 회사의 공급자가 호스팅하고 인터넷을 통해 사용자에게 제공되는 인프라, 플랫폼 또는 소프트웨어를 말합니다.

이를 이용하면 자체 인프라나 하드웨어 설치 없이도 애플리케이션과 리소스에 쉽고 싸게 이용이 가능합니다.



[분산된 서버를 기반으로 클라우드를 이용하는 모습]

이렇게 되면 서버를 직접 구매할 때 고려해야 할 전력, 위치, 서버 세팅, 확장성을 고민하지 않고 서비스 운영에만 집중할 수 있습니다. 이를 오프프레미스(off-premise) 방식이라고 합니다.

### 온프레미스(on-premise)방식

이와는 반대로 온프레미스방식은 기업이나 개인이 자체 시설에서 보유하고 직접 유지 관리하는 프라이빗 데이터 센터(IDC)을 의미합니다.



## ex) 네이버의 데이터센터 각



업계 관계자들이 꼽는 춘천의 최대 장점은 '프리쿨링'을 통한 전기 절약이 가능하다는 점이다. 프리쿨링은 서버실 온도를 조절하기 위한 냉각수를 전기가 아닌 외부 찬 공기를 이용해 만드는 방식이다. 산간 지방에 위치한 춘천은 연중 최대 6개월까지 프리쿨링이 가능하다.

- 데이터센터 냉각시설의 효율화를 위해 강원도로 자리잡음.

## 클라우드 #3 IaaS, PaaS , SaaS

### IaaS

IaaS(Infrastructure-as-a-Service)는 인프라형 클라우드서비스입니다.

클라우드가 단지 인프라를 제공합니다. node.js, MongoDB 등을 개발자가 직접 설치해야 하는 대신 특정 서비스에 종속되지 않습니다.

ex) AWS의 EC2, NCP 등

ex) NCP 사용법

<https://blog.naver.com/jhc9639/221737799510>

### PaaS

PaaS(Platform-as-a-Service)는 플랫폼형 클라우드 서비스입니다.

클라우드가 플랫폼을 제공합니다. Node.js, MongoDB 등이 설치되어있으며 그저 클릭을 통해 해당 서비스를 이용할 수 있습니다. 모니터링, CI/CD가 제공됩니다.

ex) heroku

The screenshot shows the Heroku dashboard for the app 'aviss-signal'. At the top, there's a navigation bar with 'Personal' (dropdown), 'aviss-signal' (app name), 'GitHub' (repo link), and buttons for 'Open app' and 'More'. Below the header, there are tabs for 'Overview', 'Resources', 'Deploy', 'Metrics', 'Activity', 'Access', and 'Settings'. A banner at the top says 'Get a complete visualization of your app in a team-based continuous delivery environment with [Heroku Pipelines](#)' with 'Hide' and 'Create a Heroku Pipeline' buttons. The main content area has sections for 'Installed add-ons (\$0.00/month)', 'Dyno formation (\$0.00/month)', and 'Collaborator activity'. On the right, there's a 'Latest activity' log showing deployment and build logs from a user named 'jhc9639@naver.com'.

ex) heroku의 예 : 자유롭게 클릭 몇번으로 여러가지 서비스들을 설치가 가능함.

<https://elements.heroku.com/addons/rediscloud>

The screenshot shows the Heroku Elements page for RedisCloud, specifically the 'Data Stores' section. On the left, there's a sidebar with 'ADD-ON CATEGORIES' including Data Stores, Data Store Utilities, Monitoring, Logging, Email/SMS, Caching, Errors and Exceptions, Content Management, Search, Metrics and Analytics, Testing, Messaging and Queueing, Network Services, Alerts and Notifications, User Management, Development Tools, Security, Dynos, and Content. The main area is titled 'Data Stores Choose where to store your data.' and contains a grid of eight service cards:

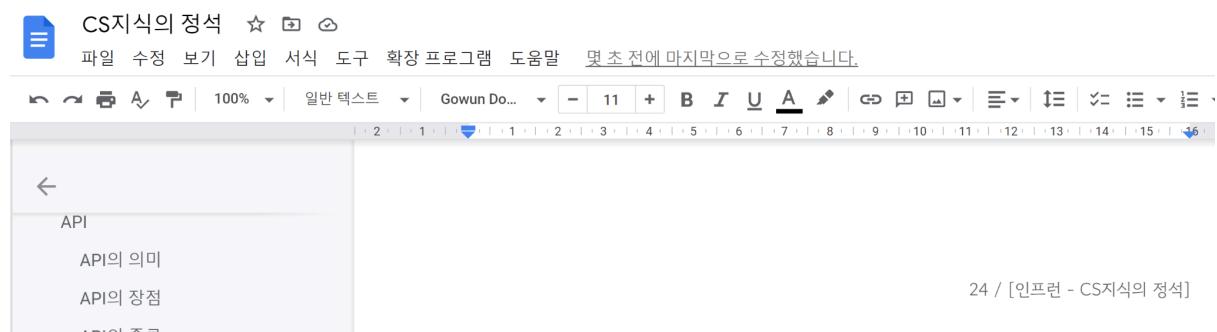
- Stackhero for InfluxDB** (BETA): InfluxDB on dedicated instances, up-to-date versions and attractive prices.
- Bucketeer**: Use Amazon S3 from your Heroku application.
- RethinkDB Cloud** (BETA): Cloud-hosted elastic RethinkDB.
- Stackhero for MariaDB**: MariaDB on dedicated instances, up-to-date versions and super attractive prices.
- SentinelDB** (BETA): A privacy by design, GDPR-compliant database with per-record encryption.
- Redis Enterprise Cloud**: From the creators of Redis. Enterprise-Class Redis for Developers (w/ Free plan).
- CloudKarafka**: Message streaming as a service powered by Apache Kafka.
- Crunchy Bridge** (BETA): Crunchy Bridge is a fully managed Postgres service from the Postgres Experts.

## SaaS

SaaS(Software as a Service)는 서비스형 클라우드서비스입니다.

완전한 서비스를 클라우드서비스로부터 제공받아 사용합니다.

## ex) 구글DOCS



구글 DOCS의 경우 클라우드를 통해 다른 컴퓨터에서도 쉽게 작업. 다른 사람과의 실시간 공유작업이 가능합니다.

## PaaS와 IaaS 비교

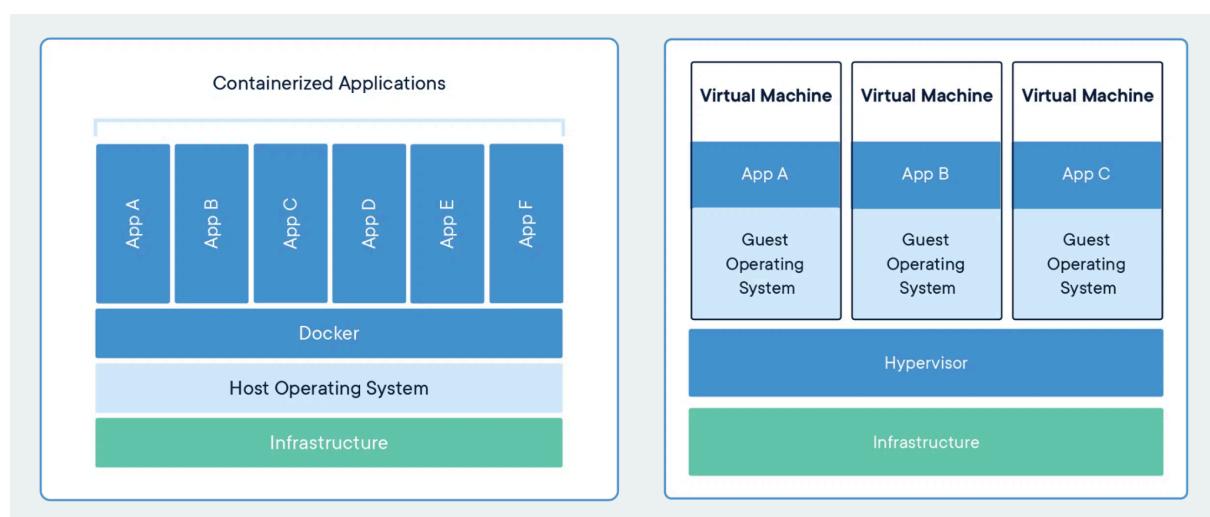
### IaaS

- 유연하며 플랫폼에 종속되지 않습니다.
- 이식성이 높음
- 운영비 효율 낮음

### PaaS

- 유연하지 않으며 플랫폼에 종속됩니다.
- 이식성은 낮음
- 운영비 효율 좋음

## 클라우드 #4 컨테이너와 도커



## 컨테이너

컨테이너는 애플리케이션이 한 컴퓨팅 환경에서 다른 컴퓨팅 환경으로 빠르고 안정적으로 실행되도록 코드와 모든 종속성을 패키징하는 소프트웨어의 표준 단위입니다.

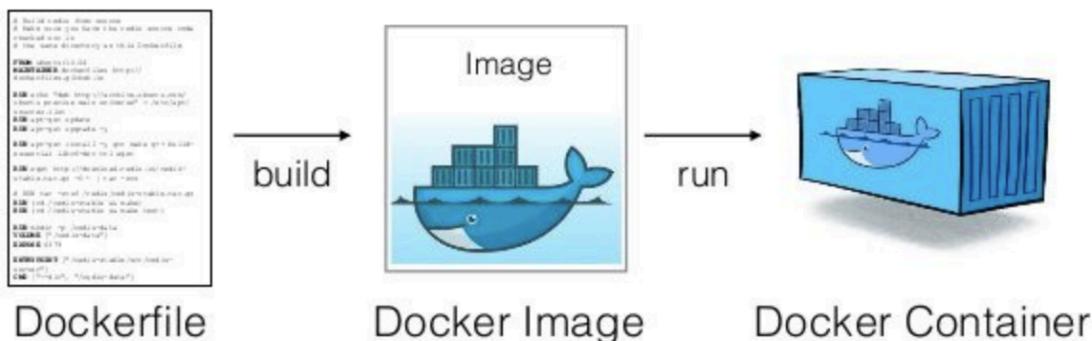
컨테이너는 OS를 공유하기 때문에 빠르고, 경량화되어있으며 격리성도 훌륭합니다. 그러나 OS에 문제가 생기면 다른 앱에도 영향을 미칠 수 있습니다.

## 도커

도커는 컨테이너 필요한 기능을 거의 모두 제공하는 플랫폼입니다.

애플리케이션 구동에 필요한 환경설정관련 절차를 도커파일에 작성하고 그 후 빌드를 하면 도커이미지가 생성이 되고 도커이미지를 실행시키면 도커컨테이너가 만들어집니다.

그 후 도커컨테이너에 설정된 프로그램, 데이터 등이 실제 컴퓨팅자원 위에서 돌아가게 됩니다.



즉, 도커는 다음과 같은 과정을 거쳐 컨테이너를 만듭니다.

1. 도커파일 : 패키지, 환경변수설정 등을 기록한 파일입니다. 이를 빌드해 도커이미지로 변환합니다.

2. 도커이미지 : 컨테이너 실행에 필요한 파일과 설정값, 데이터 등을 포함된 상태값이며 불변합니다. 하나의 이미지에서 여러개의 컨테이너를 생성할 수 있으며 컨테이너의 상태와는 무관하게 이미지는 그대로 존재합니다.

예를 통해 1대의 서버에 환경설정해야 한다면 미리 만들어 놓은 이미지를 다운받아서 이를 통해 컨테이너만 만들면 끝입니다.

3. 도커컨테이너 : 컨테이너가 실행시키면 도커이미지에 설정된 프로그램, 데이터 등이 실제 컴퓨팅자원과 연결됩니다.

## 도커의 활용사례

이렇게 만들어지는 도커컨테이너를 기반으로 클라우드에 컨테이너배포방식으로 서비스가 많이 운영되고 있습니다.

2014년 구글에서 발표한 자료에 따르면 구글에서 만드는 서비스들 대부분이 도커 컨테이너에 기반하여 쓰이고 있으며 매주 약 20억개의 서비스가 운영된다고 합니다.

## Google and Containers

**Everything** at Google runs in a container.

Internal usage:

- Resource isolation and predictability
- Quality of Services
  - batch vs. latency sensitive serving
- Overcommitment (not for GCE)
- Resource Accounting

We start over 2 billion containers per week.

## CI/CD(Continuous Integration/Delivery & Deployment)

개발자가 코드를 짬다면 그 다음 해야 할일은 무엇일까요?

바로 지속적으로 코드를 합치고 코드를 배포해야 합니다.

이를 CI/CD(Continuous Integration/Delivery & Deployment)라고 합니다.

## 왜 필요할까?

혼자가 아닌 수많은 개발자가 코드를 합치고 배포를 계속해서 시스템 없이 수동으로 한다면 이런 일이 발생하게 됩니다.

- dev 서버에 누가 배포했나요? 제 환경에서 갑자기 안되는데요?
- 이 함수 테스트 안하고 배포했나요? 해당 부분에서 에러 뜨는 거 같아요.

여러 명의 개발자가 동시에 개발을 하게 될 것이고 이는 앞과 같은 문제가 발생되게 됩니다.

이를 수동으로 하나하나 해결할 수는 없습니다. 이를 위해 CI/CD라는 개념이 도래했고 이를 쉽게 해주는 툴 등이 나오게 되었습니다.

## 파이프라인

코드구축부터 시작해서 배포까지의 일련의 과정들을 CI/CD 파이프라인이라고 합니다.

총 3가지의 단계로 구성됩니다.



continuous integration : 코드를 빌드하고 테스트하고 합칩니다.

continuous delivery : 해당 레퍼지토리에 릴리스합니다.

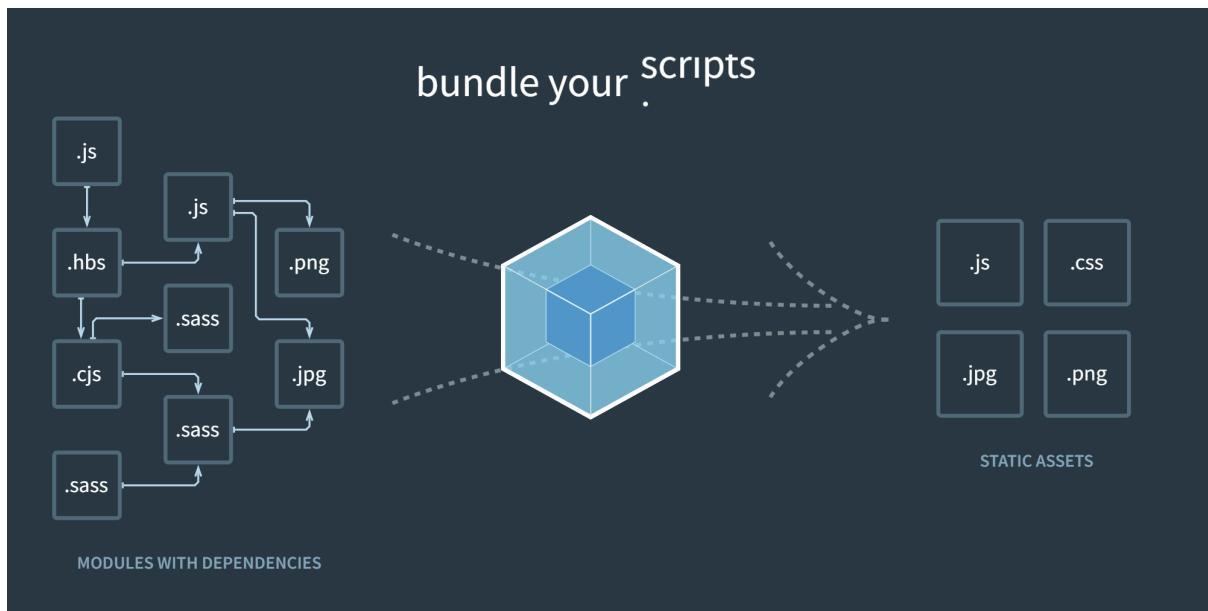
continuous deployment : 이를 프로덕션, 즉 실제 서비스에 배포합니다.

파이프라인이 주는 장점은 코드배포까지 좀 더 체계적으로 만드는 점과 테스트가 강제된다는 점입니다. 파이프라인 자체내에 테스트가 있기 때문에 테스트 없으면 코드 머지자체가 안되게 만들 수도 있기 때문이죠.

## 빌드

대표적인 예로 webpack이 있습니다.

<https://webpack.js.org/>



ex) <https://vuejs.org/>

## 테스트

테스트는 함수 등 작은 단위를 테스팅하는 단위테스트, 모듈을 통합할 때 테스트하는 통합테스트, 사용자가 서비스를 사용하는 상황을 가정해서 테스트하는 엔드투엔드테스트가 대표적입니다.

테스트를 위한 대표적인 프레임워크로는 mocha가 있습니다.



Mocha is a feature-rich JavaScript test framework running on [Node.js](#) and in the browser, making asynchronous testing *simple* and *fun*. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases. Hosted on [GitHub](#).

이외에 코드 보안 테스트도 포함합니다.

## 머지

git이나 svn을 이용해 머지를 합니다. 요새는 그냥 git을 쓴다고 보시면 됩니다.

작은 프로젝트 같은 경우 충돌을 최소화하기 위해 어떤 폴더는 해당 개발자만 맡는다고 할 수도 있습니다.

**충돌이라는 것은 대부분 일어나기 때문에 조금 더 작은 단위로 충돌이 일어나게 하는게 중요합니다.**

한달동안 코드를 짜서 배포하는게 아니라 작은 이슈단위로 나눠서 보통 머지를 합니다. 그렇다고 해서 너무 아토믹하게 작은 단위로는 하지는 않고 작은 issue단위를 기반으로 머지를하게 됩니다.

만약 충돌시에는 서로 화면공유하면서 합의하에 충돌을 해결하는게 제일 좋습니다.

ex) mocha

<https://github.com/mochajs/mocha/pull/4511>

## 배포

배포는 그저 사용자를 위한 서비스를 배포할 수도 있다고 생각할 수 있지만 그뿐만이 아닌 내부적으로 QA엔지니어나 관리자페이지를 위한 배포, 데이터웨어하우스로부터 데이터를 가공해서 백엔드개발자를 위한 배포 등을 포함합니다.

## 툴

github action, genkins, circle ci가 유명하며 heroku를 통해 CI, CD 설정 없이 자동 가능 참고로 heroku + github action 으로 설정도 가능.

## 간단한 실습

The screenshot shows the AWS Lambda console interface. At the top, it displays the user's personal icon, the function name 'temp454', and the GitHub repository 'wnghdjcfe/temp' with the main branch selected. Below this, a navigation bar includes links for Overview, Resources, Deploy, Metrics, Activity (which is underlined), Access, and Settings. The main content area is titled 'Activity Feed' and lists two recent events:

- A deployment event by 'jhc9639@naver.com' with commit hash '50f6b1b6' just now, with a link to 'Compare diff'.
- A build event by 'jhc9639@naver.com' that succeeded just now, with a link to 'View build log'.

## Q. 클래스와 객체와 인스턴스의 차이가 뭔가요?

java 테스팅사이트 : <https://www.jdoodle.com/online-java-compiler/>

javascript 테스팅사이트 : <https://www.programiz.com/javascript/online-compiler/>

### 클래스

클래스(class)란 객체(object)를 만들어 내기 위한 틀이며 만들어 낸 객체의 속성과 메서드의 집합을 담아놓은 것

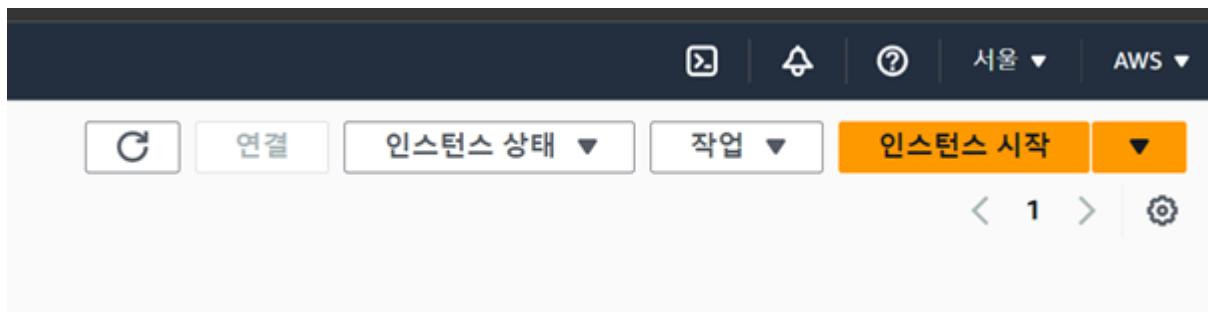
### 객체

객체란 클래스로부터 만들어지는 실제, 클래스로 선언된 변수를 객체라 한다.

### 인스턴스

인스턴스(instance)란 객체가 메모리에 할당이 된 상태이며 런타임에 구동되는 객체를 말합니다. 객체와 같은 의미로 쓰이기도 합니다.

참고로 인스턴스는 AWS의 클라우드의 가상서버라는 말로도 쓰입니다.



## 실습 - java

```
public class Person {
    // 멤버변수(속성)
    String name;
    int IQ;
    int str;
    // constructor
    public Person(String name, int IQ, int str){
        this.name = name;
        this.IQ = IQ;
        this.str = str;
    }
    public Person(){
        this.name = "alanwalker";
        this.IQ = 100;
        this.str = 100;
    }
    // 메서드
    public void levelup(){
        this.IQ = this.IQ + 1;
        this.str = this.str + 1;
        System.out.println(this.name + "의 지능과 힘이 증가했습니다!
" + this.IQ + " / " + this.str);
    }
    public static void main(String[] args) {
        Person a = new Person(); // 객체 >> 인스턴스
        a.levelup();

        Person b; // 객체
        b = new Person("홍철", 1, 1000); // 인스턴스
        b.levelup();
    }
}
```

## 실습 - javascript

```
class Person {
    constructor(name = "alanwalker", IQ = 100, str = 100) {
        this.name = name;
        this.IQ = IQ;
        this.str = str;
    }

    levelup() {
        this.IQ++;
        this.str++;
        console.log(this.name + "의 지능과 힘이 증가했습니다! " +
this.IQ + " / " + this.str);
    }
}

const a = new Person();
a.levelup();

const b = new Person("홍철", 1, 1000);
b.levelup();
```

## Q. static키워드는 왜 사용하며 단점은 무엇인가요?

static의 개념등에 대해서는 java를 기준으로 설명하며 이후 javascript로도 실습을 진행하며 설명하겠습니다.

### static

static 키워드는 클래스의 인스턴스가 아닌 클래스에 속하며 클래스의 변수, 메서드 등을 공유하는데 사용됩니다. 이를 통해 해당 클래스로 만들어지는 객체사이에서 중복되는 메서드, 속성을 효율적으로 정의할 때 쓰이며 단순히 전역변수가 아니라 클래스내의 static 키워드로 선언하여 이 클래스의 객체들끼리 사용되는 메서드 또는 속성이다. 라는 것을 나타내주는 명시성이라는 장점이 생기기 때문에 씁니다.

## static 실습 - java

```
public class Person {
    // 멤버변수(속성)
    String name;
    int IQ;
    int str;
    private static final String GUDOC = "큰돌의 터전";
    // constructor
    public Person(String name, int IQ, int str){
        this.name = name;
        this.IQ = IQ;
        this.str = str;
    }
    public Person(){
        this.name = "alanwalker";
        this.IQ = 100;
        this.str = 100;
    }
    // 메서드
    public void levelup(){
        this.IQ = this.IQ + 1;
        this.str = this.str + 1;
        System.out.println(this.name + "의 지능과 힘이 증가했습니다!
" + this.IQ + " / " + this.str);
    }
    // public void talk(Person a, Person b){
    //     System.out.println(a.name + " & " + b.name + "이 대화를
    시작했다!");
    // }
    private static void talk(Person a, Person b){
        System.out.println(a.name + " & " + b.name + "이 대화를
    시작했다!");
    }
    public static void main(String[] args) {
        Person a = new Person(); // 객체 >> 인스턴스
        a.levelup();

        Person b; // 객체
        b = new Person("큰돌", 1000, 1); // 인스턴스
        b.levelup();
    }
}
```

```

        Person.talk(a, b);
        System.out.println(Person.GUDOC);
    }
}

```

[참고] final은 상수를 정의할 때 쓰입니다. 이를 통해 재할당이 불가능하게 만듭니다.

## static 실습 - javascript

```

class Person {
    constructor(name = 'alanwalker', IQ = 100, str = 100) {
        this.name = name;
        this.IQ = IQ;
        this.str = str;
    }

    levelup() {
        this.IQ += 1;
        this.str += 1;
        console.log(this.name + "의 지능과 힘이 증가했습니다! " +
this.IQ + " / " + this.str);
    }

    static talk(a, b) {
        console.log(a.name + " & " + b.name + "이 대화를
시작했다!");
    }
    static GUDOC = "큰돌의 터전";
}

const a = new Person();
a.levelup();

const b = new Person("큰돌", 1000, 1);
b.levelup();
Person.talk(a, b);
console.log(Person.GUDOC);

```

## static 단점

static 키워드로 선언된 변수, 블록, 메서드 등은 선언과 동시에 미리 heap영역이 아닌 Method area 메모리 영역에 할당이 되며 프로그램이 종료 될 때까지 GC에 의해 메모리가 회수되지 않기 때문에 만약 클래스가 객체로 쓰이지 않는다면 메모리 낭비를 불러올 수 있습니다.

## Q. 오버로딩과 오버라이딩은 무엇인가요?

### 오버로딩(Overloading)

오버로딩은 이름이 같아도 매개변수 개수, 타입, 순서를 다르게 해서 같은 이름으로도 여러 개의 함수를 정의할 수 있는 것을 말합니다. 이는 프로그램의 유연성을 높이고 결과적으로 코드를 깔끔하게 하는 효과가 있으며 같은 클래스 내에서 사용합니다.

ex1)

```
class Calculator{
    void multiply(int a, int b){
        System.out.println("결과는 : "+(a * b) + "입니다.");
    }
    void multiply(int a, int b,int c){
        System.out.println("결과는 : "+(a * b * c) + "입니다.");
    }
    void multiply(double a, double b){
        System.out.println("결과는 : "+(a * b) + "입니다.");
    }
}
public class MyClass {
    public static void main(String args[]) {
        int a = 1;
        int b = 2;
        int d = 4;
        Calculator c = new Calculator();
        c.multiply(a, b);
        c.multiply(a, b, d);
        double aa = 1.2;
        double bb = 1.4;
```

```
    c.multiply(aa, bb);

}
```

ex2)

```
class Person{
    void pay(String a, int b){
        System.out.println(a + "가 "+ b + "원만큼 계산합니다. ");
    }
    void pay(int a, String b){
        System.out.println(b + "가 "+ a + "원만큼 계산합니다. ");
    }
}
public class MyClass {
    public static void main(String args[]) {
        Person c = new Person();
        c.pay("영주", 1000000000);
        c.pay(1000000000, "영주");

    }
}
```

ex3)

C++에서도 쓰이기도 함. (이처럼 다른 언어에서도 쓰임, 되는 언어가 있고 안되는 언어가 있음.)

```
#include<bits/stdc++.h>
using namespace std;
string a = "aaa";
string b = string("aaa");
string c = string(3, 'a');
int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
```

```

cout.tie(NULL);
cout << a << '\n';
cout << b << '\n';
cout << c << '\n';
}
/*
aaa
aaa
aaa
*/

```

## 오버라이딩(Overriding)

상위 클래스가 가지고 있는 메서드를 하위 클래스가 재정의를 하는 것을 말합니다. 상속 관계 클래스에서 사용되며 static, final로 선언한 메서드는 오버라이딩이 불가능합니다.

ex1)

```

class Animal{
    void eat(){
        System.out.println("먹습니다.");
    }
}

class Person extends Animal{
    @Override
    void eat(){
        System.out.println("사람처럼 먹습니다. ");
    }
}

public class MyClass {
    public static void main(String args[]) {
        Person a = new Person();
        a.eat();
    }
}

```

ex2)

interface를 기반으로 구체적인 함수 정의는 하지 않고 이를 기반으로 여러개의 하위 클래스를 만들어 오버라이딩을 하기도 합니다.

```
interface Animal{
    public void eat();
}

class Person implements Animal{
    @Override
    public void eat(){
        System.out.println("사람처럼 먹습니다. ");
    }
}

public class MyClass {  

    public static void main(String args[]) {
        Person a = new Person();
        a.eat();
    }
}
```

## Q. 추상화란 무엇인가요?

### 추상화의 의미

프로그래밍에서의 추상화는 복잡한 데이터, 구조, 시스템 등으로부터 핵심만을 가려내 덜 자세하게 만드는 것 또는 세부사항, 절차 등을 감추고 인터페이스 등을 만드는 것으로 복잡도를 낮추는 방법을 말합니다.

추상화는 데이터, 프로세스 추상화 크게 2가지로 나눠집니다.

### 데이터 추상화

어떠한 데이터들의 공통점을 모으고 차이점은 버립니다. 예를 들어 고양이, 강아지, 원숭이 등의 객체들의 공통적인 특징을 묶어 동물이라는 카테고리로 카테고리화 시킵니다.

## 프로세스 추상화

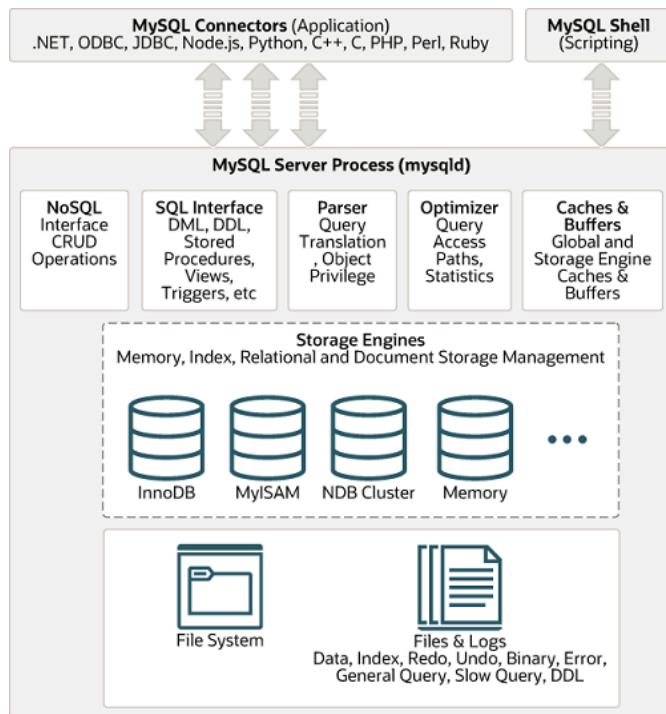
어떠한 내부 프로세스를 숨기는 것을 말합니다. 예를 들어 데이터베이스가 어떻게 데이터를 저장하는지는 모르지만 단순하게 insert, upsert 등의 쿼리로 데이터를 저장할 수 있습니다.

### ex1) Animal Class

```
abstract class Animal {  
    public abstract void animalSound();  
    public void sleep() {  
        System.out.println("zzz");  
    }  
}  
  
class Pig extends Animal {  
    public void animalSound() {  
        System.out.println("꿀꿀꿀~");  
    }  
}  
class Dog extends Animal {  
    public void animalSound() {  
        System.out.println("왈왈~");  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Pig a = new Pig();  
        a.animalSound();  
        a.sleep();  
        Dog b = new Dog();  
        b.animalSound();  
        b.sleep();  
    }  
}
```

## ex2) MySQL 아키텍처

코드뿐만 아니라 실제 시스템에서도 추상화는 이뤄져있습니다. 대표적인 예로 데이터베이스 시스템이 있습니다. 데이터베이스 내의 내부프로세스와는 상관없이 “쿼리”를 통해 접근할 수 있습니다.



## Q. 컴파일러와 인터프리터의 차이가 무엇인가요?

컴파일러와 인터프리터는 프로그래밍 언어로 작성된 코드를 컴퓨터가 이해할 수 있는 기계어로 변환하는 과정에 관여하는 프로그램입니다.

### [참고] 기계어

기계어 코드(Machine Code)는 컴퓨터 프로세서가 직접 이해하고 실행할 수 있는, 가장 기본적인 형태의 명령어 집합입니다. 이러한 코드는 0과 1, 즉 이진수(binary)로 표현되며, 컴퓨터의 중앙 처리 장치(CPU)가 직접 실행할 수 있는 유일한 언어이자 가장 낮은 수준의 언어이며 이는 기계어가 컴퓨터 하드웨어와 직접적으로 소통할 수 있는 언어를 의미합니다.



Figure: Compiler



Figure: Interpreter

## 컴파일러

- 전체 변환: 소스 코드의 전체를 읽어 한 번에 기계어로 변환합니다. 변환 과정을 거친 후, 생성된 기계어 코드를 실행합니다.
- 속도: 컴파일 과정 자체는 시간이 걸리지만, 변환된 코드는 직접 실행되므로 실행 시간은 빠릅니다.
- 사용 예: C, C++, Go, Rust 등의 언어가 컴파일러를 사용합니다.
- 코드를 수정했을 때 컴파일과정이 필요합니다.
- ex) C++ 실습

### [참고] 컴파일과정

고수준의 소스코드를 전처리, 컴파일러, 어셈블러, 링커의 과정을 거쳐 저수준언어로 만들고 실행할 수 있는 프로그램을 만드는 과정

## 인터프리터

- 한 줄씩 변환: 소스 코드를 한 줄씩 읽어가며 바로 기계어로 변환하고 실행합니다.
- 속도: 컴파일 단계가 없으므로 초기 시작은 빠르지만, 전체 코드 실행 시간은 컴파일러를 사용할 때보다 느릴 수 있습니다. 코드를 실행할 때마다 변환 과정을 거치기 때문입니다.
- 사용 예: Python이 대표적으로 인터프리터 방식을 사용합니다.
- 코드를 수정했을 때 컴파일과정이 필요하지 않습니다.
- ex) Python 실습

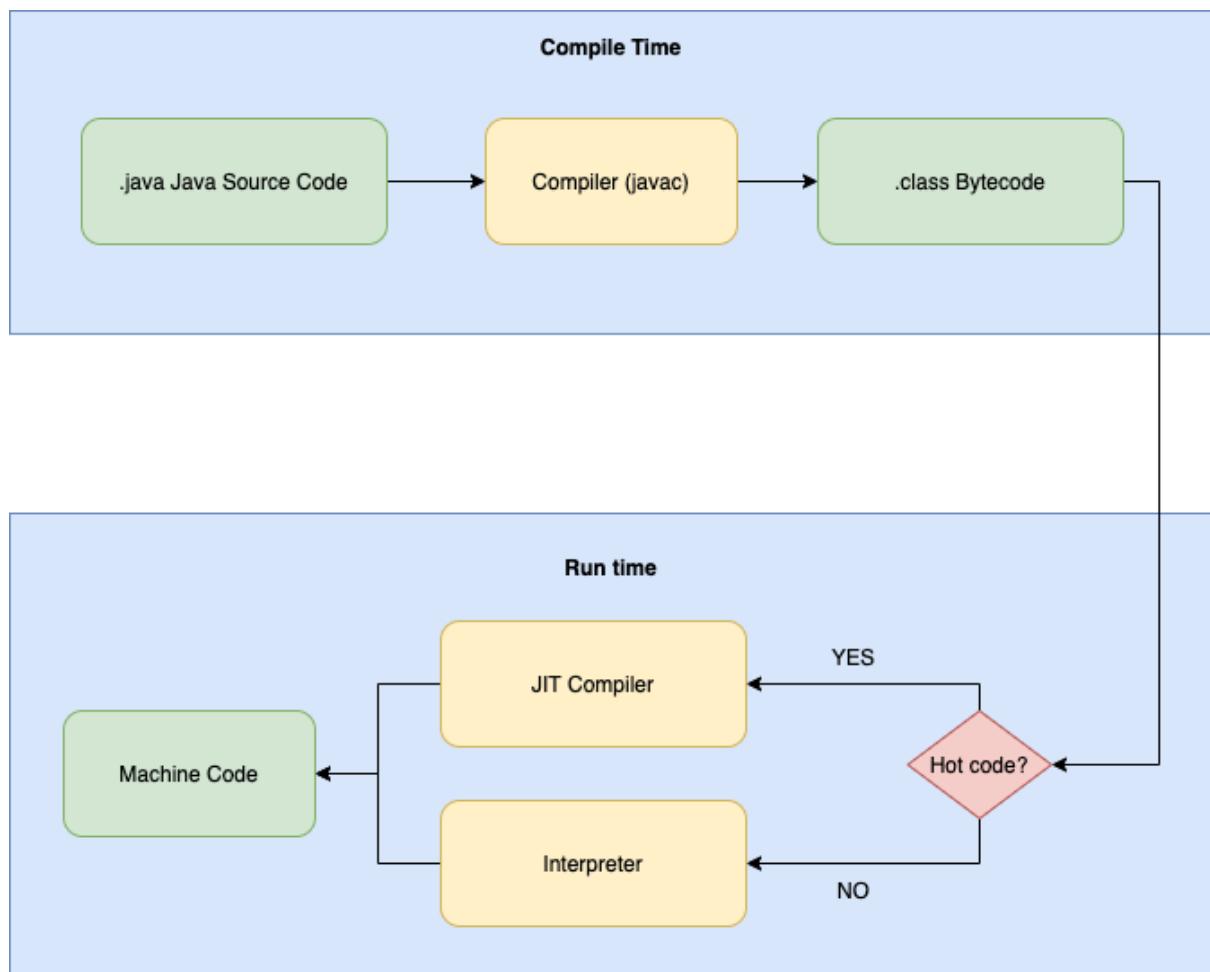
모두 고수준의 언어를 기계어로 변환한다는 공통점이 있습니다.

## JIT 컴파일러

JIT 컴파일러는 인터프리터와 컴파일러의 중간 형태라고 볼 수 있습니다.

1. 코드 분석: 실행 시점에 프로그램 코드를 분석하여, 어떤 부분이 가장 자주 실행되는지(hot spot) 판단합니다.(즉, 실행전 코드를 분석하지 않습니다.)
2. 동적 컴파일: 분석 결과에 기반하여, 자주 실행되는 코드(hot spot)만을 선별적으로 기계어로 변환합니다. 이 과정은 프로그램 실행 중에 실시간으로 이루어집니다.
3. 최적화: 컴파일 과정에서 다양한 최적화 기법을 적용합니다. (ex. 메모리 접근 패턴을 분석, 가비지 컬렉션의 오버헤드 최소화 등)
4. 실행: 컴파일된 기계어 코드를 실행합니다. 프로그램이 계속 실행되면서 새로운 핫 스팟이 발견되면, 해당 부분도 JIT 컴파일을 통해 최적화됩니다.

ex) JVM(자바가상머신), .NET, V8(node.js 엔진)



자주 사용되는 코드(hot code)의 실행 속도가 크게 향상되는 것이라는 장점과 컴파일 된 코드를 메모리에 저장해 캐싱하기 때문에 인터프리터 방식에 비해 더 많은 메모리를 소비한다는 단점이 있습니다.

여기까지 오시느라 수고하셨습니다.  
질문 있으시면 언제든지 질문 부탁드립니다.  
좋은 수강평과 별점 5점은 제가 큰 힘이 됩니다. :)  
감사합니다.