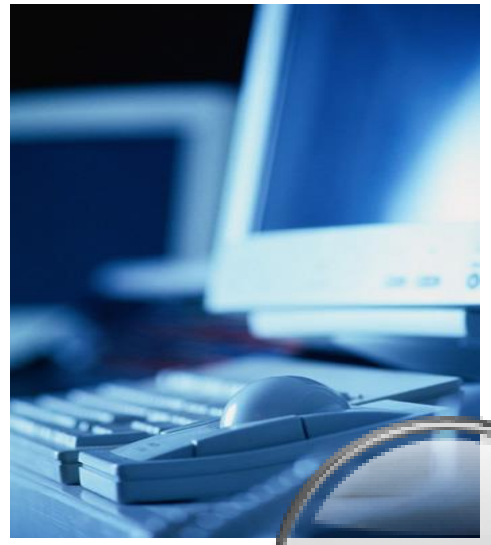
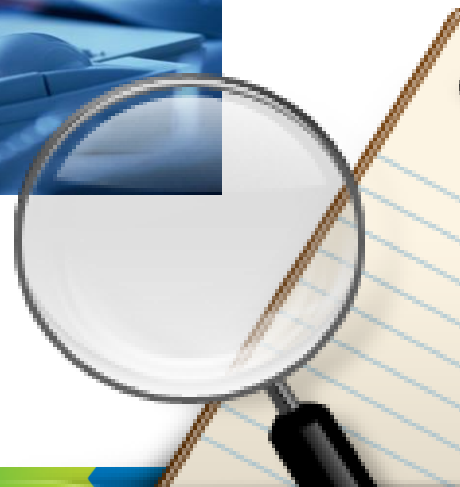
쿠버네티스 마이크로서비스아키텍처 프로젝트

- MSA 아키텍처 소개
- flask를 활용한 웹페이질 렌더링
- RestfulAPI 설계하기
- flask를 활용한 RestfulAPI 구성





MSA 아키텍처 소개

MSA 아키텍처 소개

MSA의 특징

출처: https://waspro.tistory.com/429

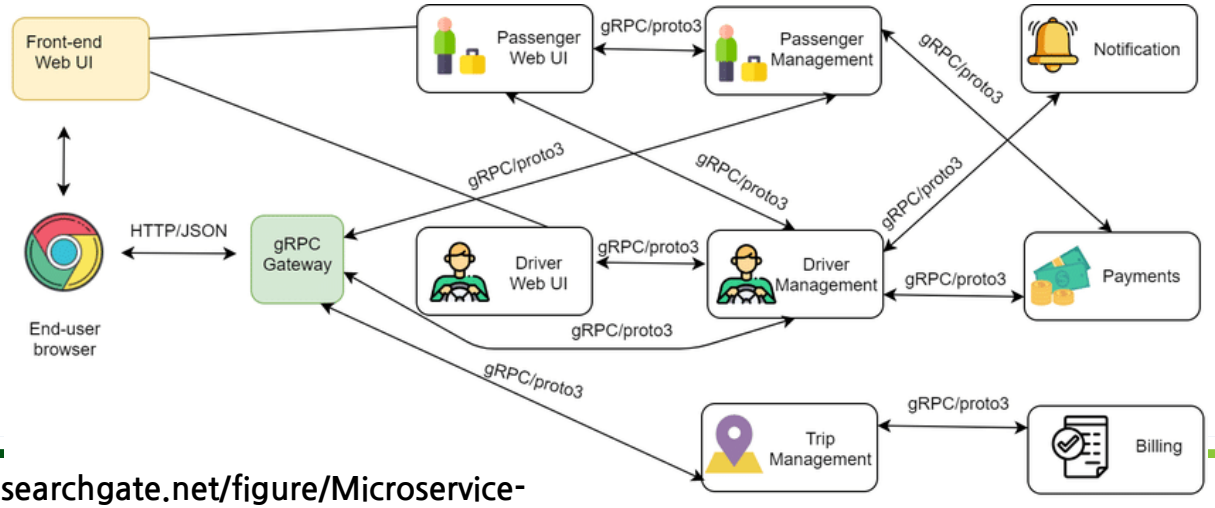
① 애플리케이션 로직을 각자 책임이 명확한 작은 컴포넌트들로 분해하고 이들을 조합해서 솔루션 제공

② 각 컴포넌트는 작은 책임 영역을 담당하고 완전히 상호 독립적으로 배포
마이크로서비스는 비즈니스 영역의 한 부분에서만 책임을 담당
여러 애플리케이션에서 재사용할 수 있어야 함

③ 마이크로서비스는 몇가지 기본 원칙에 기반
소비자와 서비스 제공자 사이의 데이터 교환을 위해 HTTP와 JSON 같은 경량 통신 프로토콜 사용

④ 애플리케이션은 항상 기술 중립적 프로토콜을 사용해 통신하므로 서비스 구현 기술과는 무관
마이크로서비스 기반의 애플리케이션을 다양한 언어와 기술로 구축 가능

⑤ 작고 독립적이며 분산된 마이크로서비스를 사용해 조직
명확히 정의된 책임 영역을 담당하는 소규모 팀을 보유
이 팀들은 애플리케이션 출시처럼 하나의 목표를 향해 일함//자기가 개발하는 서비스만 책임



그림출처: https://www.researchgate.net/figure/Microservice-architecture-of-a-taxi-hailing-system-35_fig2_349134846

MSA 아키텍처 소개

MSA의 목적

출처: https://waspro.tistory.com/429

- 마이크로서비스를 달성하기 위해서는 많은 노력과 비용이 수반
- 따라서 MSA를 적용하고자 할 경우 명확한 목적을 갖고 고려해야 함

- 1) 마이크로서비스 아키텍처를 통해 달성하고자 하는 목표는 무엇인가?
  - 명백한 목표 예를 들어 기존 대비 1.5배 빠른 성능, 이벤트에도 중단되지 않는 가용성 높은 서비스 등 결과를 기준으로 설명할 수 있어야 하며, 시스템의 end-user에게 이점을 제공하는 방식으로 설명될 수 있어야 한다.

- 2) 마이크로 서비스 사용에 대한 대안을 고려했습니까?
  - 마이크로서비스가 제공하는 것과 동일한 이점을 얻을 수 있는 다른 많은 방법이 있다. 때로는 MSA를 고려하지 않고도 적용 가능한 방법이 다양하게 존재하므로, 이에 대한 고려를 선행한 후 MSA를 선택하는 것이 좋다.

MSA 아키텍처 소개

MSA의 장점

출처: https://waspro.tistory.com/429

① 작은 서비스들로 나누고, 각 서비스를 독립적으로 만듦 → loosely-coupled(약결합)
② 대용량 분산 환경에 적합
③ 복잡도 감소
④ 유연한 배포
⑤ 재사용성 → 확장성
⑥ 서비스별 hw/sw 플랫폼/기술의 도입 및 확장이 자유로움
⑦ 개발자가 이해하기 쉽고 개발/운영 매 단계의 생산성이 높음
⑧ 지속적인 개발/디플로이가 biz capability 단위로 관련된 소수의 인원의 책임하에 이뤄짐
⑨ Fault isolation (장애 허용 시스템) 특성이 좋음

MSA 아키텍처 소개

MSA의 단점

출처: https://waspro.tistory.com/429

단점	보완방법
장애추적, 모니터링, 메시징 처리가 어렵다.	Sleuth 등과 ELK, EFK, Splunk 등의 서비스를 연동하여 사용하는 방안 고려
여러 서비스에 걸쳐져 있는 Feature의 경우, 트랜잭션을 다루기 어렵다.	보상 트랜잭션 또는 부분적으로 composite 서비스로의 병합 고려
여러 서비스에 걸쳐져 있는 Feature의 경우, 테스팅이 복잡하다.	테스팅 계획 및 방법에 노력 투자
서비스 간 Dependency가 있는 경우 릴리즈가 까다롭다.	관련 개발조직 간 roll-out 계획 마련 및 dependency의 관리체계 수립
서비스 개수가 많고 유동적이기 때문에 CI/CD 및 서비스 관리 상의 문제가 발생할 수 있다.	서비스 레지스트리, 모니터링, 개발/디플로이 자동화 기술 고려 (PaaS 고려)
모놀리스 시스템을 마이크로서비스 전환할 때 큰 고동이 수반될 수 있다.	B2C 웹 또는 SaaS 어플리케이션은 처음부터 마이크로서비스 및 서비스 별 조직을 고려하여 구성

flask를 활용한 웹페이지 렌더링

flask를 활용한 웹페이지 렌더링

Flask?

- 파이썬의 대표적인 웹 개발 프레임워크 중 하나
- 마이크로 프레임워크, 가볍고 간단
- 지정한 라이브러리와 패키지만 설치됨 => 효율성, 자유도가 높음

Flask

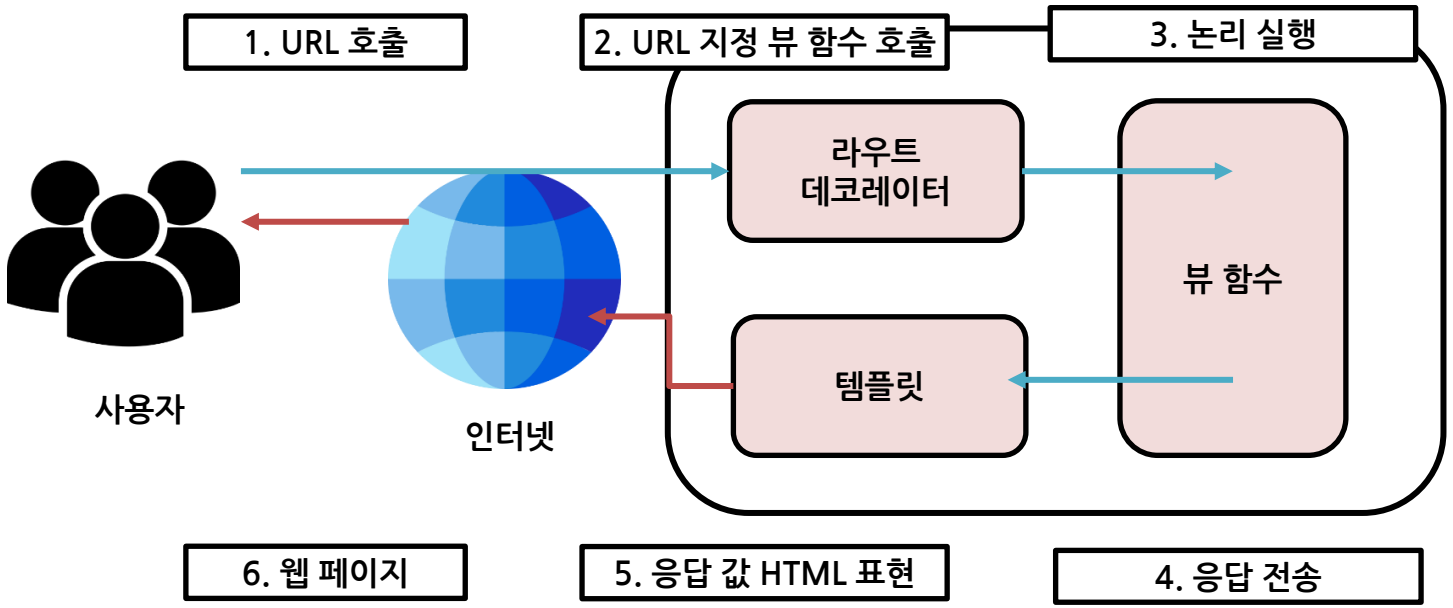
web development,
one drop at a time

"Micro" does not mean that your whole web application has to fit into a single Python file

공식 홈페이지
https://flask.palletsprojects.com/en/1.1.x/

플라스크 앱 구조

- 플라스크 앱은 다음 과정을 통해 호출
  1. 특정 URL 호출
  2. 뷰 함수 호출
  3. 논리 실행
  4. 논리 결과 응답 전송
  5. 응답 값 HTML 표현
  6. 클라이언트 전달



flask를 활용한 웹페이지 렌더링

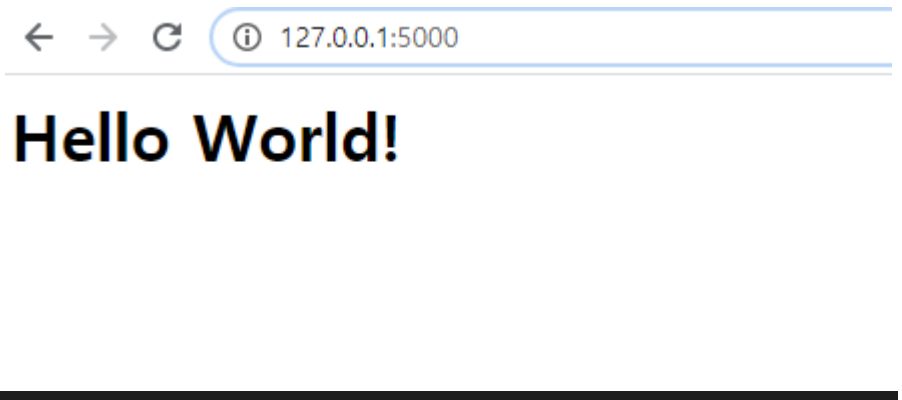
플라스크 시작하기

- **파이썬 웹 마이크로 프레임워크, 백 엔드 서버 기능**
  - WSGI : Web Server Gateway Interface, CGI의 업그레이드형 (성능 업!)
  - Werkzug + Jinja2 템플릿 엔진 구성
  - 기본 5000번 포트 사용
  - 웹 페이지 문자열 출력 예제

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
  return "<h1>Hello World!</h1>"

if __name__== "__main__":
  app.run()
```




```
* Serving Flask app "test" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

사용하기 쉬운 REST API

사용하기 쉬운 REST API

REST API 기본 개념

- Representational State Transfer의 약자로 자원을 이름으로 구분하여 주고 받는 모든 것을 의미
- HTTP URL로 자원을 명시하고 HTTP Method로 자원의 CRUD 오퍼레이션을 적용

- 장점
  - HTTP 표준 프로토콜에 따르는 모든 프로토콜에서 사용 가능하며 디자인 문제 최소화, 서버와 클라이언트 역할 분리 등이 있음

- 단점
  - 구형 브라우저가 모든 메서드를 지원하지는 않아 사용할 수 있는 메서드가 제한적임

- 특징
  1. 서버와 클라이언트 구조
  2. 무상태, 클라이언트 context를 서버에 저장하지 않아 구현이 단순하며 클라이언트 요청만 단순 처리
  3. 캐시 처리가 가능
  4. 계층화, 클라이언트는 REST API 서버만 호출하며 다중 계층으로 구성

REST API 설계 기본 규칙

REST API는 도큐먼트, 컬렉션, 스토어로 구성

- 도큐먼트 : 데이터베이스 레코드,
- 컬렉션 : 서버에서 관리하는 리소스, 디렉터리
- 스토어 : 클라이언트에서 관리하는 리소스 저장소

REST API 설계 규칙

1. 슬래시 구분자(/)는 계층 관계를 나타내는데 사용한다.
2. URI 마지막 문자로 슬래시(/)를 포함하지 않으며 URI 경로의 마지막에는 슬래시(/)를 사용하지 않는다.
3. 하이픈(-)은 URI 가독성을 높이는데 사용한다.
4. 언더바(_)는 URI에 사용하지 않는다.
5. URI 경로에는 소문자가 적합하다.
6. 파일확장자는 URI에 포함하지 않는다.
7. 리소스 간에는 연관 관계가 있는 경우 /리소스명/리소스 ID/관계가 있는 다른 리소스명으로 표기한다.

사용하기 쉬운 REST API

REST API 잘 설계하기

- 처음부터 설계를 잘해야 함
- 팀 멤버와 유저들도 제대로 활용할 수 있음
- API는 기본적인 CRUD를 활용
- 명확한 패턴이 필요
- URL에서 동사를 사용하지 않아야 함(대신에 HTTP request method를 사용)
- 자동차 판매점 API 예제
  - 자동차를 추가
  - 자동차를 검색
  - 자동차 엔진이나 옵션, 소개글 등 조회

잘못된 설계

```
/createCar
/seeCars
/getCar/Bentz
/getEngines/Bentz/e-class
/deleteCar/Bentz
/updateCar/Bentz
/getTopRatedCars
/findCarsFromThisYear
```

→

제대로된 설계

```
POST /cars
GET /cars
GET /cars/bentz
GET /cars/bentz/e-class
DELETE /cars/bentz
PUT /cars/bentz
GET /cars?min_rating=9.8
GET /cars?release_date=2021
```

→

압축

```
GET     /cars
POST    /cars/bentz
PUT     /cars/bentz/e-class
DELETE
```

flask를 활용한 REST API 구성

사용하기 쉬운 REST API

REST API 잘 설계하기

GET	/cars
POST	/cars/bentz
PUT	/cars/bentz/e-class
DELETE	

```python
from flask_restful import Resource
from flask_restful import reqparse

class Minus(Resource):
    def get(self):
        try:
            parser = reqparse.RequestParser()
            parser.add_argument('x', required=True, type=int, help='x cannot be blank')
            parser.add_argument('y', required=True, type=int, help='y cannot be blank')
            args = parser.parse_args()
            result = args['x'] - args['y']
            return {'result': result}
        except Exception as e:
            return {'error': str(e)}

from flask import Flask
from flask_restful import Api

app = Flask('My First App')
api = Api(app)
api.add_resource(Minus, '/minus')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8000, debug=True)
```