# MSA(Micro Sevice Architecture)

## MSA의 정의

: 애플리케이션을 느슨히 결합된 서비스의 모임으로 구조화하는 서비스 지향 아키텍처 스타일의 일종인 소프트웨어 개발 기법이다.

규모가 작은 자율적인 팀들이 팀별 서비스를 독립적으로 개발, 전개, 규모 확장을 할 수 있게 함으로써 병렬로 개발할 수 있게 한다. 또 지속적인 리팩토링을 통해 개개의 서비스 아키텍처가 하나로 병합될 수 있게 허용한다. MSA는 지속적 배포와 전개(디플로이)를 가능케 한다.

- Highly maintainable and testable (높은 유지보수, 테스트 가능)
- Loosely coupled (느슨한 결합)
- Independently deployable (독립적 배포)
- Organized around business capabilities.(비즈니스 기능 중심)

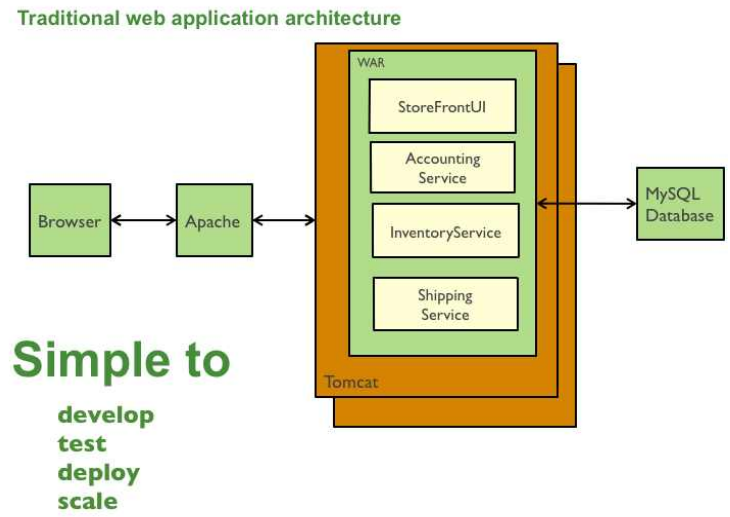
### MSA의 장단점

| MSA 의 장점 | MSA 의 단점 |
|---|---|
| 1. 기능별로 마이크로 서비스를 개발하고, 작업할당을 서비스 단위로 하면 개발자가 해당 부분을 온전히 이해할 수 있다.<br>2. 새로 추가되거나 수정 사항이 있는 마이크로 서비스만 빠르게 배포 가능<br>3. 해당 기능에 맞는 기술 언어등을 선택하여 사용할 수 있다.<br>4. 일부분의 오류가 있으면 해당 기능에만 오류가 발생하고 그 부분만 수정 | 1.관리가 힘들다. 작은 서비스들이 분산되어 있어 모니터링이 힘들다.<br>2. 서로를 호출하여 전체의 서비스가 이루어지기 때문에 까다롭다.<br>3. 통신 관련 오류가 잦을 수 있다.<br>4. 테스트가 불편하다. UI, Gateway 등등 여러개의 서비스를 구동시켜야 한다. |

## 모놀리틱(Monolithic)

: 비즈니스 로직 및 UI 등 서비스를 처리하기 위한 대부분의 컴포넌트가 단일 시스템에서 구동 되는 것을 의미한다.

- 단일 어플리케이션 지향 -> 언어 ,프레임워크 선택에 따라 개발 내용이 달라짐.
- 간단한 테스트, 배포, 스케일
- 대용량 시스템 자체를 경량화하거나 리빌딩할수는 없는 환경(한계점)




## 모놀리틱의 장단점

| 모놀리식 구조의 장점 | 모놀리식 구조의 단점 |
| --- | --- |
| 1. 어떤 기능이든지 개발되어 있는 환경이 같다.<br>2. 쉽게 고가용성 서버 환경을 만들 수 있다. (같은 애플리케이션으로 하나 더 만들면 됨)<br>3. end-to-end 테스트가 용이함<br>(MSA 같은 경우 테스트에 필요한 서비스를 모두 동작 시켜야함) | 1. 한 프로젝트의 덩치가 너무 커져서 애플리케이션 구동시간이 늘어나고 빌드, 배포 시간도 길어진다.<br>2. 작은 수정사항, 프레임워크 업그레이드 및 고도화를 하려면 전체를 빌드하고 배포해야 한다. 특정 서비스나 기능을 확장 (Scale-out) 할 수 없으며, 전체 어플리케이션이 확장 되어야 한다.<br>3. 많은 양의 코드가 몰려있어 개발자가 모두를 이해할 수 없고 유지보수가 어려움<br>4. 일부분 오류가 전체에 영향을 미친다.<br>5. 전체 기능에 대해 언어 및 프레임워크에 종속적으로 구성된다. (언어, 프레임워크에 종속적)<br>6.코드 패키지 형태의 배포 방식을 취하므로, 개발 -> 상용에 환경 이슈가 발생할 수 있다. |

## 모듈과 컴포넌트, 응집도와 결합도

모듈 : 기능의 단위, 컴포넌트의 집합, 정적구조, 구현단위
컴포넌트 : 런타임에 독립적, 배포, 실행
결합도 : 모듈과 모듈간의 기능적인 응집 정도
응집도 : 모듈 내부의 기능적인 응집 정도
낮은 결합, 높은 응집: 한 구성 요소(component)의 변경이 다른 구성요소의 존재 또는 성능에 가장 적은 영향을 미친다(느슨한 결합, loosed coupled), 높은 응집시스템(high cohension)에서는 코드 가독성과 재사용성이 증가한다.

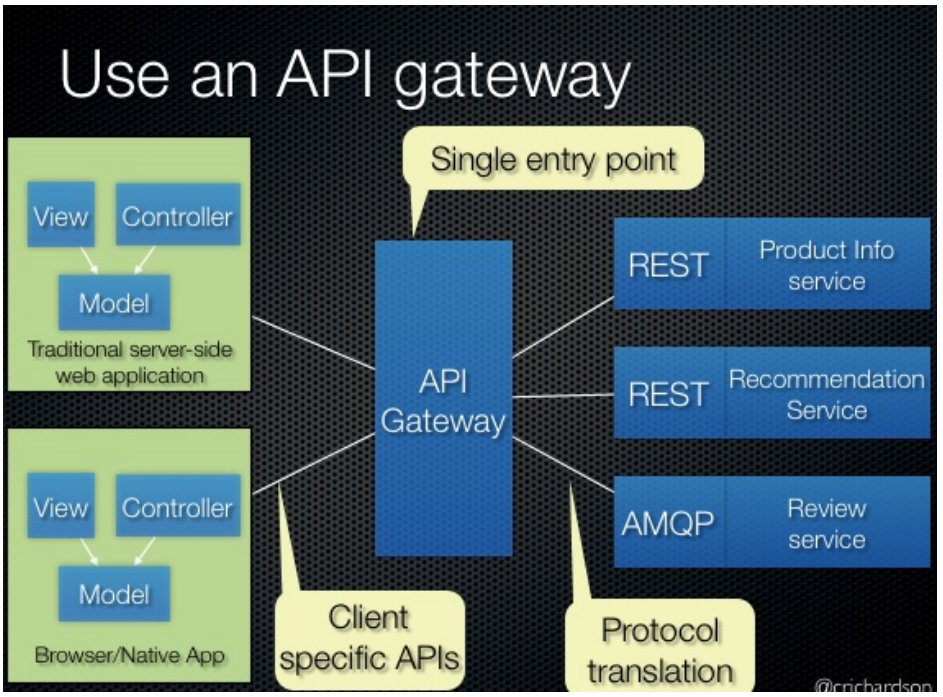
# MSA 구현을 위한 기술 Stack

## API Gateway

: API 를 제공하는 Business Logic 처리를 위한 Node 를 연결하기 위한 통로와 같으며, Nginx Plus / Neflix Zuul / Kong 과 같은 솔루션을 통해 API Gateway 를 구성할 수 있다. API Gateway 는 ReverseProxy 와 같이 URI 를 숨겨주며, 라우팅해주는 작업을 포함해서 통합된 API 환경 자체를 의미한다.
단. MSA 환경에서 API Gateway 가 서비스 및 플랫폼에 따라 다수 연결될 수도 있으며, 이는 플랫폼과 서비스 관점에서 아키텍쳐를 도입 시 고려해야할 포인트이다.
개인적인 판단으로는 Domain to Gateway 가 1:1 형태로 매핑되는게 바람직 하다고 판단.

Domain : Gateway 매핑

- 다수 URL 이 Gateway 통과 시 유연성/안정성 확보가 어려움
- API Gateway 는 인증과 과금에 처리를 용이하게 하는 측면이 있으므로, 서비스 그룹 관점에서 서비스 그룹에 대한 과금이나 인증을 하는 것이 효과적임
- API Gateway 는 안정성을 위해 쓰로틀 / 쿼터의 기능을 내장할 수 있으므로, 각 서비스 그룹 별 관리의 용이성을 확보해야 함.




## API Gateway Routing

네트워크에서의 라우팅과 동일하며 통신 데이터를 보낼 경로를 선택하는 과정이다. 특정 URL이 어떤 서비스로 연결되는 지에 대한 정의를 나타내며, 최근 RESTful API를 지향하는 서비스의 경우 각 서비스가 의미하는게 URL에 정의된다.

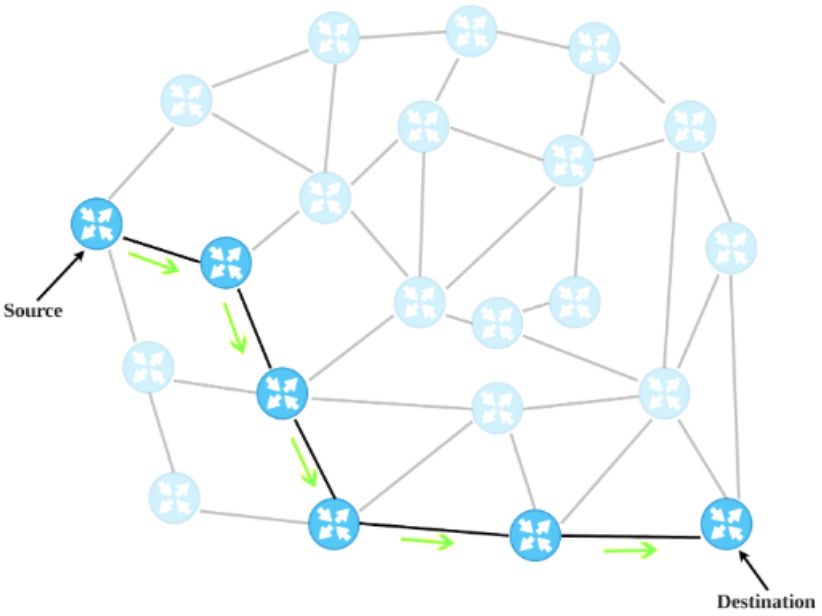
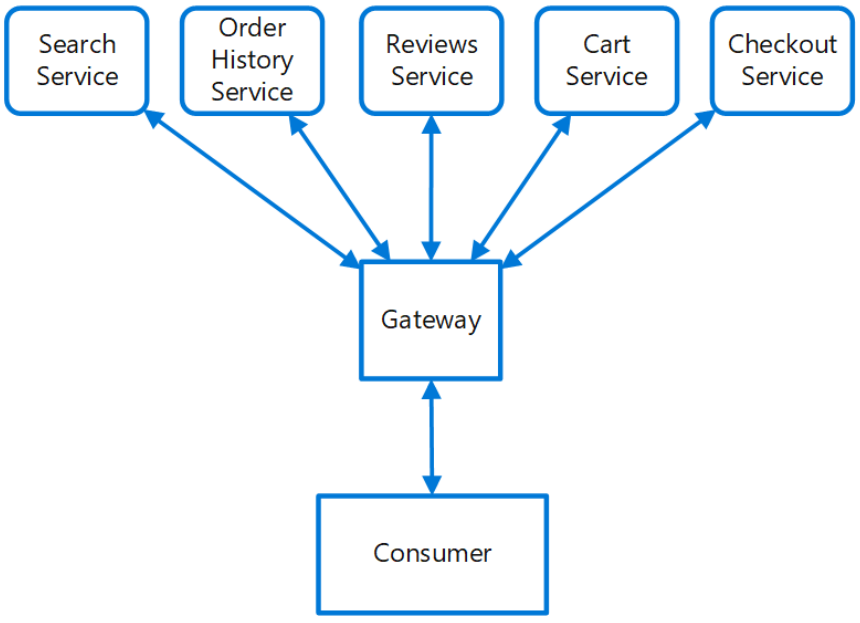



**Figure 1:** Destination-based routing along the shortest path

## URL 기반 Routing

api.growthsoft.co.kr/{service}/{Resource}

상기와 같은 샘플 URL 에서는 service 를 통해, API Gateway 를 통과 후, 실제 Backend Application 의 접근을 나타낸다. 위와 같은 형태의 Gateway 를 가장 많이 활용하며, AWS 의 API Gateway 와 Lambda 연결 시에도 URI Resource 를 활용하여 사용한다. (게이트웨이를 통한 API Service 요청 - URI 기반)




애플리케이션, 서비스 또는 배포 집합 앞에 게이트웨이를 배치합니다. 애플리케이션 계층 7 라우팅을 사용하여 요청을 적절한 인스턴스로 라우트합니다.

이 패턴을 사용하면 클라이언트 애플리케이션이 단일 엔드포인트만 알고 통신하면 됩니다. 서비스를 통합하거나 분해하는 경우 클라이언트를 업데이트할 필요가 없습니다. 게이트웨이를 계속 요청할 수 있으며 라우팅만 변경됩니다.

또한 게이트웨이를 통해 클라이언트의 백 엔드 서비스를 추상화하여 게이트웨이 뒤에 있는 백 엔드 서비스의 변경을 허용하는 동시에 클라이언트 호출을 단순하게 유지할 수 있습니다. 예상 클라이언트 동작을 처리해야 하는 어떤 서비스로든 클라이언트 호출을 라우트할 수 있으므로 클라이언트를 변경하지 않고 게이트웨이 뒤에 서비스를 추가, 분할 및 재구성할 수 있습니다.
(출처 : https://docs.microsoft.com/ko-kr/azure/architecture/patterns/gateway-routing )

AWS API Gateway (Sample URL)
`http://petstore-demo-endpoint.execute-api.com/petstore/pets`

AWS 의 경우 SubDomain 을 이용하는 서비스가 다양하게 되어 있음. API GW 처리 대신 SubDomain 및 URI 기반으로 Region 및 Service Routing 처리를 하고 있음.

ap-northeast-2.console.aws.amazon.com
ap-northeast-2.console.aws.amazon.com/**ec2**/v2/home?region=ap-northeast-2#Home
ap-northeast-2.console.aws.amazon.com/**eks**/home?region=ap-northeast-2#/home
ap-northeast-2.console.aws.amazon.com/**lambda**/home?region=ap-northeast-2#/begin
**s3.console.**aws.amazon.com

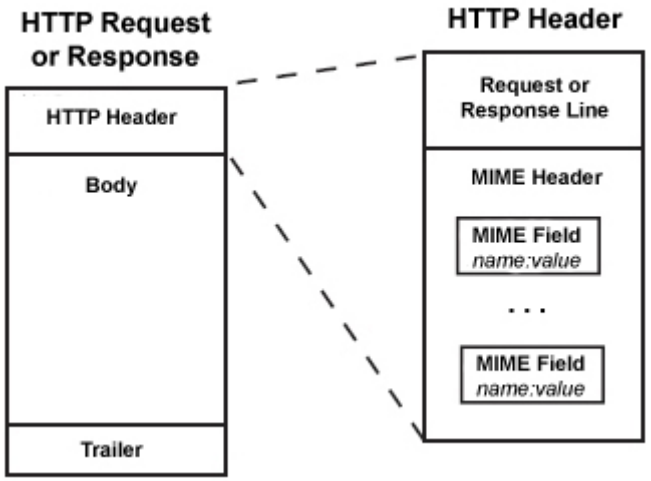
EndPoint 정의 : 상기 URL 을 보면 EndPoint 는 aws.amazon.com 으로 대표 도메인 즉 최종 연결 Entry 로 정의 될 수 있음을 알 수 있다. Endpoint 는 Backend Application 으로 직접 연결되는 경우도 존재하지만, 일반적으로 API Gateway 를 EndPoint 로 지정하고 Backend Application 의 경우 Service URL 혹은 Subdomain 기반으로 Routing 처리를 한다.

## Header / Message 기반 Routing

Domain 기반 혹은 RESTful API 가 나오기 전에 사용하던 Route 방식이며, Domain / URI 기반과 다른 점은 어플리케이션이 메시지를 수신 받은 이후에 Route 되는 특징이 있다.
일반적인 L7 (H/W) 기준에서 Routing 수행 시 헤더나 URI 를 인지하여 Route 를 하는데, API Gateway 도 HW 관점에서 L7 과 유사한 역할을 수행한다고 볼 수 있다. 다만, SSL 과 같은 암호화 인증서 방식에서는 Header / Body 정보를 탐색하기 위해 리소스를 추가로 사용하기 때문에 최근 RESTful API 관점에서는 Header / Message 기반의 Routing 을 대표적으로 사용하진 않으며, 특수성이 있는 어플리케이션에 한하여 사용한다.(보안이 필요한 서비스에 사용)
EX) 은행 레거시 송금 시스템은 TCP Byte 기반의 통신을 위해 전문의 Header 의 수 Byte 를 송금 / 수신 / 여신 등의 행위로 사용한다. 즉 TCP 특성상 연결을 맺어놓은 상태에서 소량의 Byte 처리를 해야하는 경우에는, 아직도 Header 기반의 Routing 모듈을 사용하기도 함.

상기 AWS 의 URL Pattern 을 보면 Query String 으로 Region 정보가 있으나, Header / Message 기반은 Header 의 내용을 파악하거나, Body 내용을 파악해야 하므로 overhead 가 존재한다. 따라서, 보안성이 극도로 민감한 서비스를 제외하고는 일반적인 RESTful API 에 URI 생성 규칙과 Query String 등을 활용한다.

(즉, URL 라우팅방식은 암호화가 되지 않고, 요청을 빠르게 할 수 있고, 반대로 Header/Message기반은 Http 메세지의 헤더, 바디를 읽어야만 요청 할 곳의 정보를 알 수 있기때문에 overhead가 발생함.)



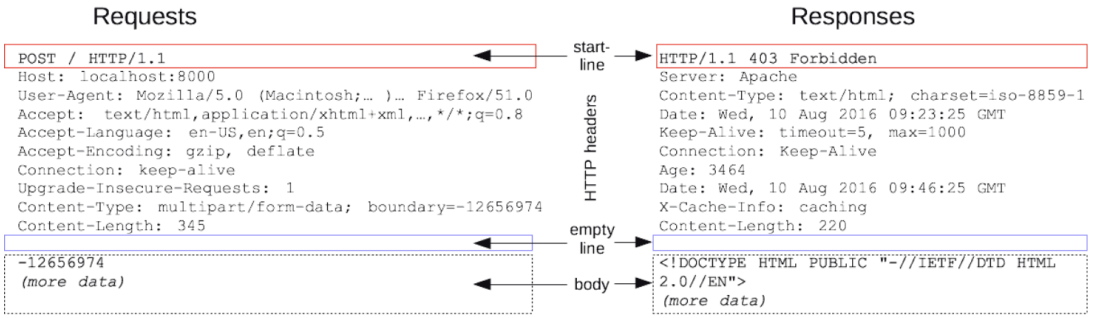

**[HTTP Header / Body]**

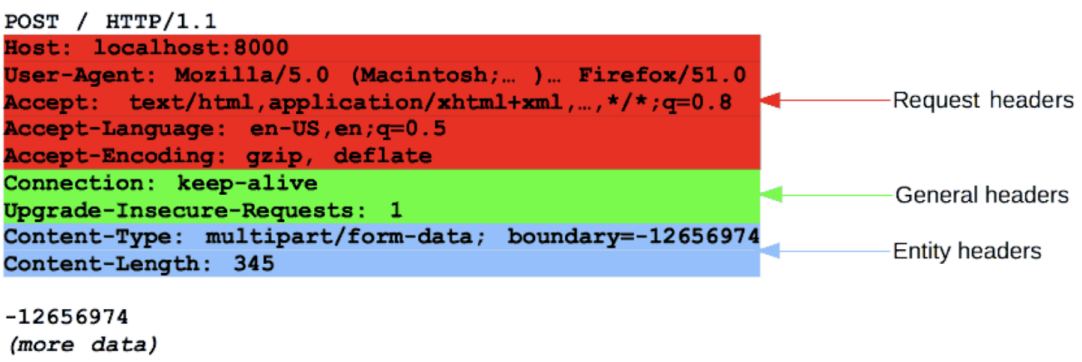
```
https://example.com/over/there?name=ferret
```

** Query String ?
?name=ferret 이부분! (보안 안됨)

Message 기반 Routing 의 경우 Message Broker (MQ) 와 같은 구조를 지니지 않는 형태를 의미한다. 즉 여기서 의미하는 Message 는 HTTP Request 의 Body 부분에 해당 되는 내용으로 Body 전문 내에 { Service : Vm } 과 같은 형태를 의미한다.

** Http message 구조



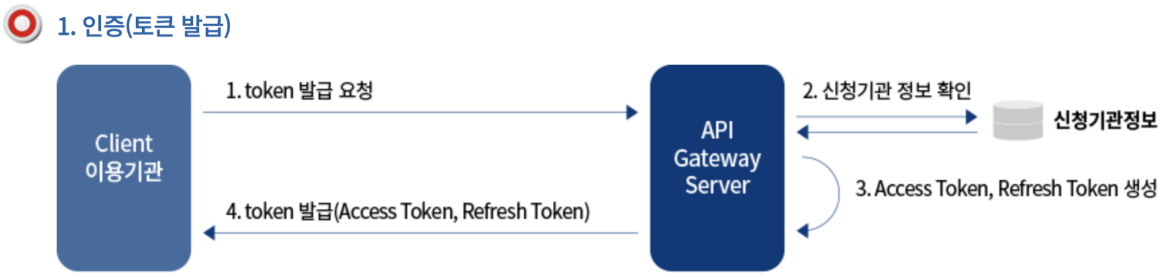



## API Gateway Token ( JWT Json Web Token )

: API의 경로에 대해 JWT 권한 부여자를 구성하면 API Gateway는 클라이언트가 API 요청과 함께 제출하는 JWT를 검증합니다. API Gateway는 토큰 검증 및 선택적으로 토큰의 범위를 기반으로 요청을 허용하거나 거부합니다. 경로의 범위를 구성하는 경우 토큰에 경로의 범위 중 하나 이상이 포함되어야 합니다.
API의 각 경로에 대해 고유한 권한 부여자를 구성하거나, 여러 경로에 대해 동일한 권한 부여자를 사용할 수 있습니다.

토큰은 비공개 시크릿 키 또는 공개/비공개 키를 사용하여 서명된다. 이를테면 서버는 "관리자로 로그인됨"이라는 클레임이 있는 토큰을 생성하여 이를 클라이언트에 제공할 수 있다. 그러면 클라이언트는 해당 토큰을 사용하여 관리자로 로그인됨을 증명한다. 이 토큰들은 한쪽 당사자의 비공개 키(일반적으로 서버의 비공개 키)에 의해 서명이 가능하며 이로써 해당 당사자는 최종적으로 토큰이 적법한지를 확인할 수 있다.

**토큰 : 토큰은 크기가 작고[2] URL 안전으로 설계되어 있으며[3] 특히 웹 브라우저 통합 인증(SSO) 컨텍스트에 유용하다.




# API GateWay 기능

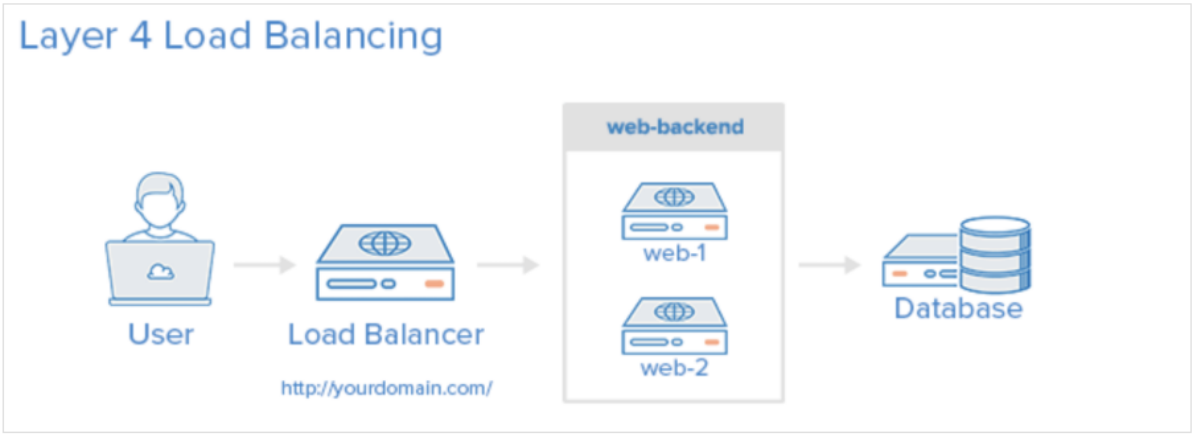
### 1. 로드밸런싱

: 컴퓨터 네트워크 기술의 일종으로 둘 혹은 셋이상의 중앙처리장치 혹은 저장장치와 같은 컴퓨터 자원들에게 작업을 나누는 것을 의미한다. 이로써 가용성 및 응답시간을 최적화 시킬 수 있다.
다수 클라이언트 접근을 처리하기 위한 Network 장비 OSI 7 계층에 의해 TCP Level 의 경우 L4 를 사용하며, Application Level 처리를 위한 L7 을 사용함. 최근 장비의 경우 L4 ~ L7 의 경계가 사실상 거의 없어지는 추세로 전환 되고 있음.

- 부하 분산의 기능(트래픽 자동 분산)
- 어플리케이션 및 서버의 라우팅 기능 (URL / HEADER 등) / 포트포워딩 기능
- SSL 인증서 처리 기능

- 서버의 주기적인 Health check 기능 및 Fail Over(장애조치) / Fail Back(장애복구) 기능
  TCP/UDP 분석 가능, 방화벽 역할

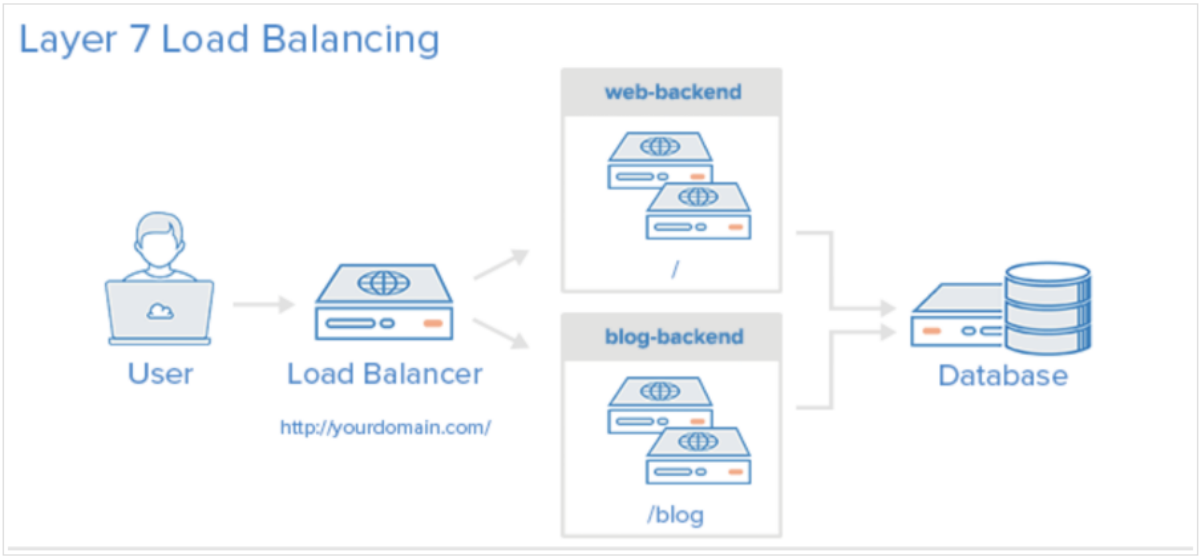
하드웨어가 제공하는 레벨까지 SW LB(API Gateway) 가 처리 성능을 제공하진 않지만, 기능이 더 다양하므로, Case 에 맞게 선별적으로 사용이 가능함.




L4 설정

- VIP (Virtual IP) : 192.168.0.100 / PORT : 80 / Algorithm : RR (Round Robin)
- RIP (Real IP) : 192.168.0.101 / 192.168.0.102 PORT : 80

** Round Robin
: 세션을 각 서버에 순차적으로 맺어주는 방식, 단순히 순서에 따라 세션을 할당하므로 경우에 따라 경로별로 같은 처리량이 보장되지 않음




L7 설정

- VIP (Virtual IP) : 192.168.0.100 / PORT : 80 / Algorithm : RR (Round Robin)
  Protocol : http
  Source URL : api.seowon.com
- RIP (Real IP) : 192.168.0.101 / 192.168.0.102 PORT : 80

- VIP (Virtual IP) : 192.168.0.100 / PORT : 80 / Algorithm : RR (Round Robin)
  Protocol : http
  Source URL : www.seowon.com
- RIP (Real IP) : 192.168.0.101 / 192.168.0.102 PORT : 80800

2. 메시지 변환
3. 프로토콜 변환
4. 메시지 호출 패턴 변환
5. 메시지 어그리게이션
6. 로깅 / 미터링
7. QoS : 사용자 및 서비스 Level QoS 보장

오케스트레이션

MSA 개발할때의 고려사항
1. 팀 자율성 향상(언어의 구애 받지 않음)
2. 시장 출시 시간 단축
3. 부하를 다루기 위한 비용 효율적인 확장(특정 서비스에 대한 scale-out)
4. 견고성 향상(단일 모듈에 대한 단일 책임원칙

URL 기반 Routing
api.growthsoft.co.kr/{service}/{Resource} => URL로 라우팅하기
Header / Message 기반 Routing

API Gateway 주요기능
로드밸런싱
: network 장비 TCP level 의 경우 : L4
IP 포트로 접근해서 TCP로 보내줌
L4 설정
- VIP (Virtual IP) : 192.168.0.100 / PORT : 80 / Algorithm : RR(Round Robin)
- RIP(Real IP) : 192.168.0.101 / 192.168.0.102 PORT : 80

L7 설정
- VIP(Virtual IP) : 192.168.0.100 / PORT : 80 / Algorithm
- Protocol : http
- Source URL: www.seowon.com

동기 : 요청을 보내고 응답 대기(처리 결과 기다림) (하나의 리퀘스트, 하나의 리스폰스)
비동기 : 요청을 보내고 응답이 오던말던 자기 일을 하는 것(처리 결과 안기다림) (여러개의 리퀘스트, 여러개의 리스폰스)

블록/논블록 I/O
:

클라이언트 사이드 랜더링(CSR)
: 초기 서버에 불러오는
서버 사이드 랜더링(SSR)
: 서버쪽에서 jsp / Servlet 랜더링을 함(장점: 클라이언트쪽에서 개발할 게 없음)

디자인패턴
: saga 패턴,

gof 디자인패턴(개발방식의 패턴방식)