IP : 인터넷 프로토콜 {

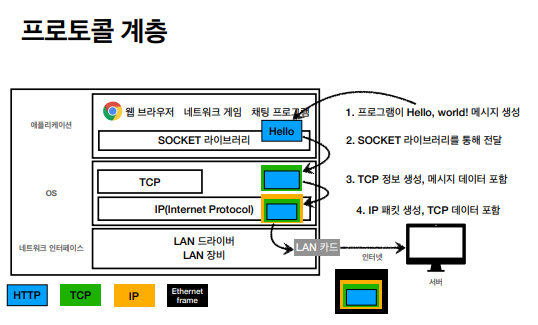
1. 패킷을 받을 대상이 없거나 서비스 불능 상태여도 패킷 전송

2. 패킷이 중간에 사라지면 답이 없음.(중간 서비스가 갑자기 꺼질 수 있으니)

3. 패킷이 순서대로 안올 수 있음

4. 같은 ip를 사용하는 서버에서 통신하는 애플리케이션이 둘 이상이면 이상함

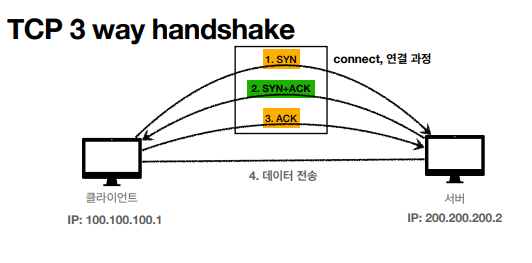
}



IP패킷(출발지, 목적지 IP) 안에 TCP세그먼트(출발지, 목적지 PORT, 전송 제어, 순서, 검증정보..)

TCP 특징 : 가상으로 연결을 미리 해봄(연결 가능 여부 확인), 신뢰할 수 있는 프로토콜

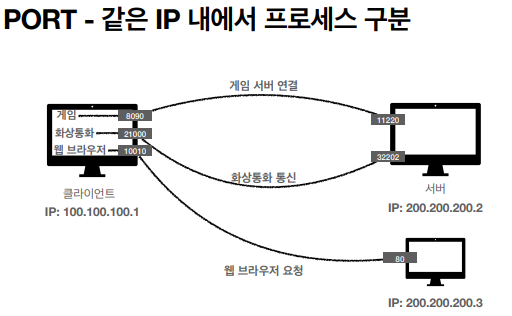
🡪[전송 제어 프로토콜(Transmission Control Protocol)]



SYN: 접속 요청 / ACK: 요청 수락 / 3.ACK에서는 함께 데이터 전송 가능 (이 둘만 연결된걸 앎)

UDP 특징 사용자 데이터그램 프로토콜(User Datagram Protocol)

• 하얀 도화지에 비유(기능이 거의 없음) • 연결지향 - TCP 3 way handshake X • 데이터 전달 보증 X • 순서 보장 X • 데이터 전달 및 순서가 보장되지 않지만, 단순하고 빠름 • 정리 • IP와 거의 같다. +PORT +체크섬 정도만 추가 • 애플리케이션에서 추가 작업 필요 🡪요즘 뜨고있음

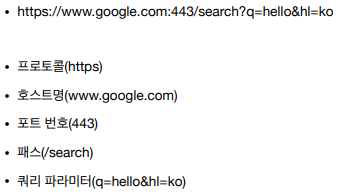


패킷에 port,IP 다 담는게 TCP/IP 패킷

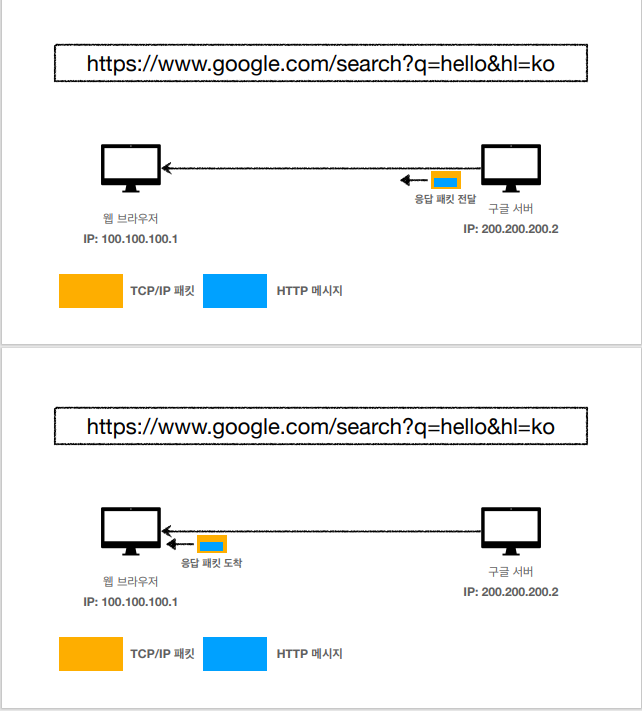
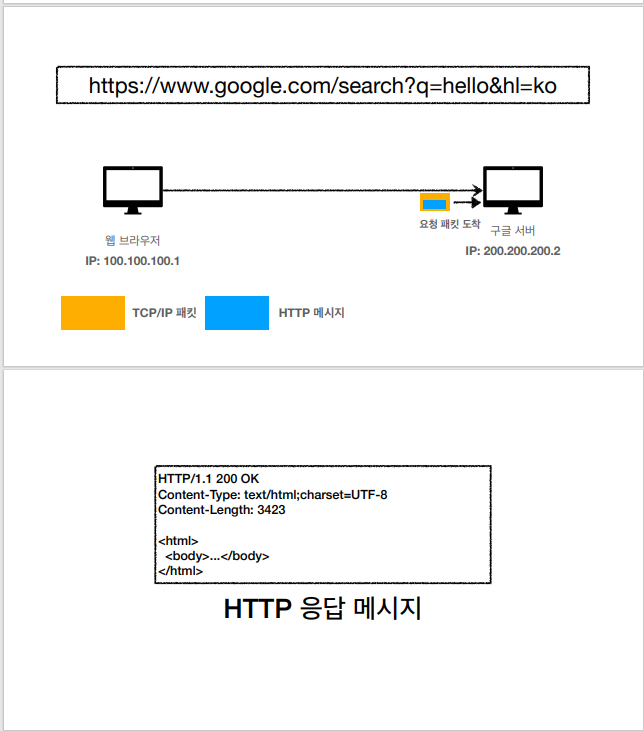
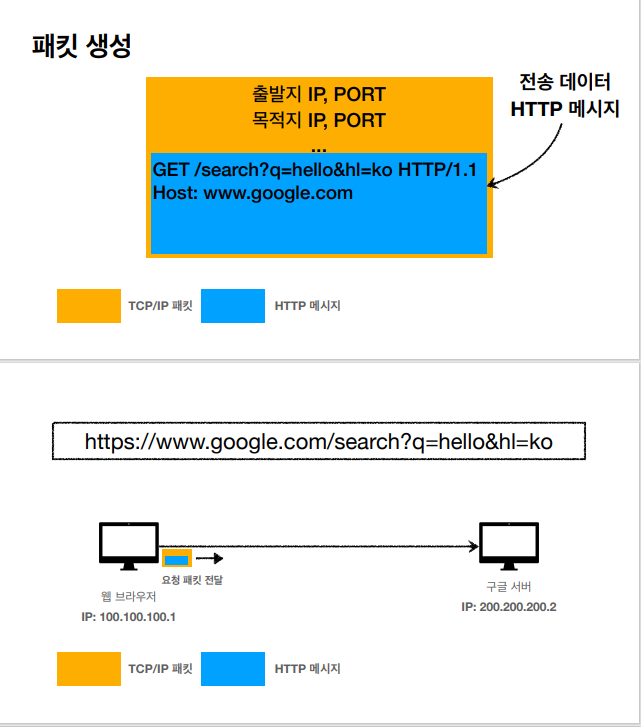
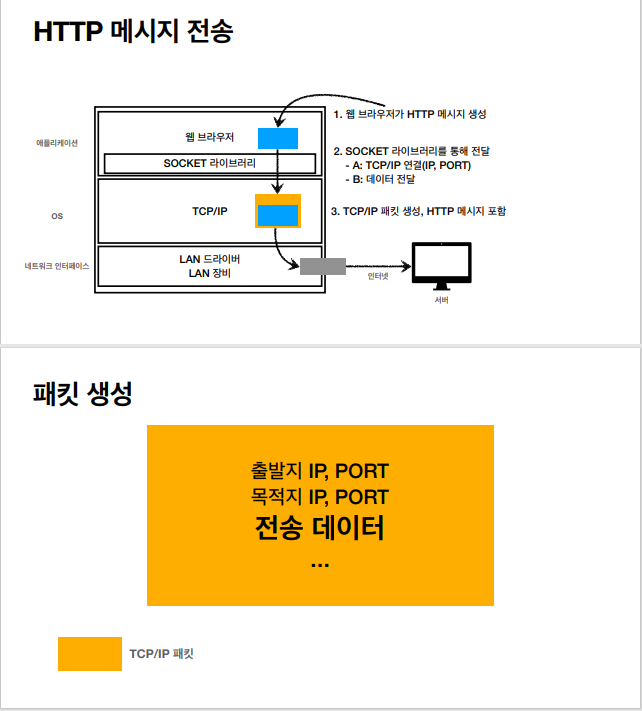
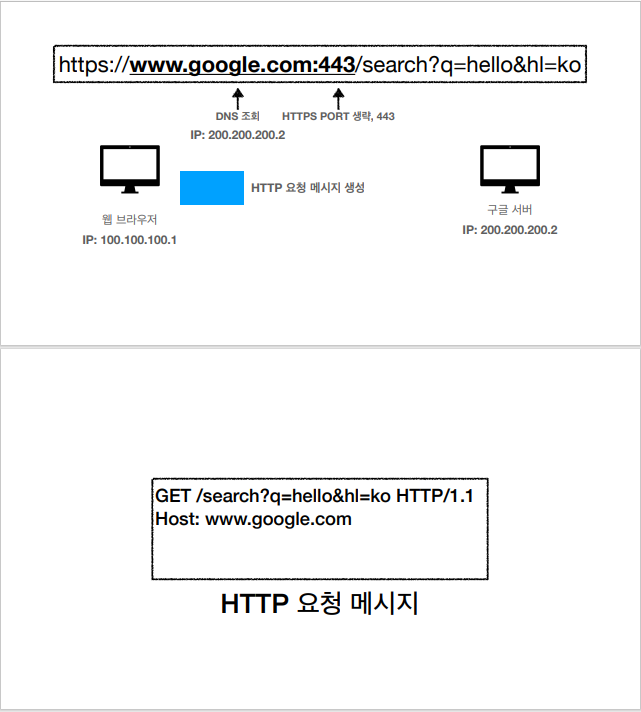
PORT: 같은 아이피안에서 통신할 애플리케이션을 구분하기 위해 사용

DNS: 도메인 네임 시스템 🡪 IP주소를 일일이 기억하기 힘드니까 도메인 주소를 통해 접근

URI :=: URL(location)+URN(name)



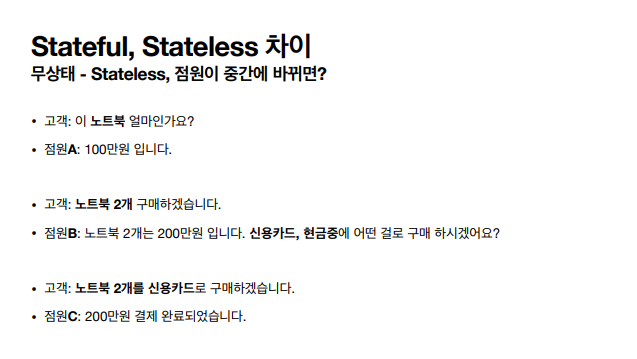
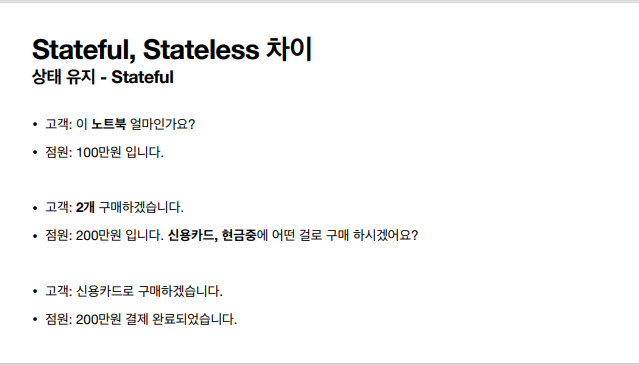
https와 http의 차이점???



HTTP: Hyper Text Transfer Protocol

1) 클라이언트 – 서버 구조의 이점 : 클라는 받아온 정보를 ui로 그리기에 집중, 서버는 비즈니스 로직에만 집중. 만약 트래픽이 폭발적으로 증가했을시 백엔드는 서버에 대해서만 해결하면 됌.

2) 무상태 프로토콜 : stateful <-> stateless 차이점.



Stateless의 장점 : 상태 유지 시에는 중간에 다른 점원으로 바뀌면 안되지만, 무상태면 중간에 다른 점원으로 바뀌어도 된다. 🡪 무한한 서버 증설 가능.

``\*한계: 로그인한 사용자의 경우 로그인 했다는 상태를 서버에 유지🡪 쿠키, 세션등을 사용!

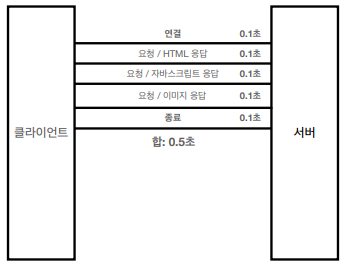
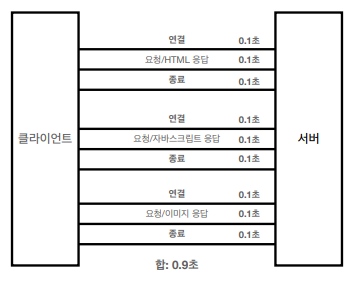
상태유지는 최소한만 사용하자.

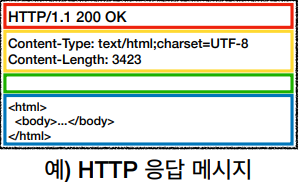
3) 비 연결성(connectionless) : 연결을 유지하지 않는 모델.

1시간 동안 수천명이 서비스를 사용해도 실제 서버에서 동시에 처리하는 요청은 수십개 이하임;

* 볼일 보고 바로 끊어버리면 서버 자원을 매우 효율적으로 사용할 수 있음.

한계 : 3 way handshake 시간 발생; 🡪 HTTP 지속 연결(Persistent Connections) 사용!





시작 라인(빨강)

1. HTTP 메서드 : GET/POST/PUT/DELETE 1. HTTP 버전, 상태 코드, 이유 문구
2. 절대경로 3.http 버전

헤 더(노랑)

HTTP 전송에 필요한 모든 부가정보 • 예) 메시지 바디의 내용, 메시지 바디의 크기, 압축, 인증, 요청 클라이언트(브라우저) 정보, 서버 애플리케이션 정보, 캐시 관리 정보 …

**“API**(Application Programming Interface, 응용 프로그램 프로그래밍 인터페이스)는 응용 프로그램에서 사용할 수 있도록, 운영 체제나 프로그래밍 언어가 제공하는 기능을 제어할 수 있게 만든 인터페이스를 뜻한다.”

<https://dydrlaks.medium.com/api-%EB%9E%80-c0fd6222d34c>

URI는 리소스만 식별. 리소스와 해당 리소스를 대상으로 하는 행위를 분리

리소스 : 회원 / 행위: 조회, 등록, 삭제, 변경 (메서드를 쓰기 때문에)

1. GET : 리소스 조회 / 서버에 전달하고 싶은 데이터는 쿼리로 전달
2. POST : 요청 데이터 처리 (ex: html form, 글쓰기, 댓글 달기, 주문 생성--)
   1. DB에 없는 새 리소스 생성
   2. 데이터 생성 및 변경을 넘어 프로세스를 처리해야 할 때
   3. 다른 메서드로 처리하기 어려운 경우 ex)JSON 으로 조회 데이터를 넘길때
3. PUT : POST는 어디에 API가 갈지 모르는데 PUT은 리소스 위치를 알고 있어야함.

!주의사항: username 필드는 사라짐

1. PATCH : 리소스의 부분적인 데이터만 변경하고 싶을시 사용.

클라이언트->서버로 데이터 전송

1. 쿼리 파라미터를 통한 데이터 전송 : GET, 정렬 필터(검색어)
2. 메시지 바디를 통한 데이터 전송 : POST, PUT, PATCH

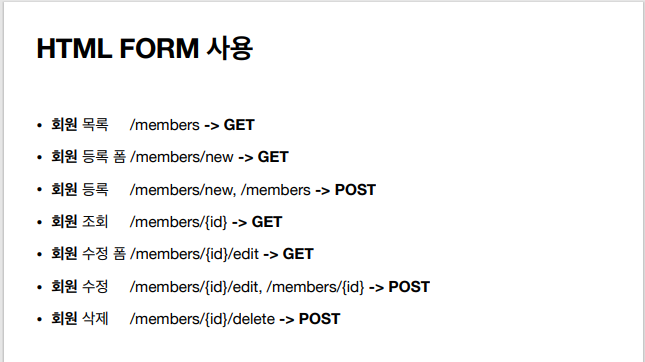
정적, 동적 데이터 조회 (GET) / HTML Form (POST) / HTTP API 데이터 전송.

http api -> 서버 <-> 서버 통신할 때, js로 통신할 때.

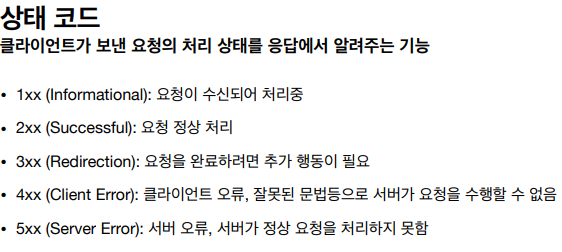
->Content-Type: application/json을 주로 사용 (사실상 표준) • TEXT, XML, JSON 등등

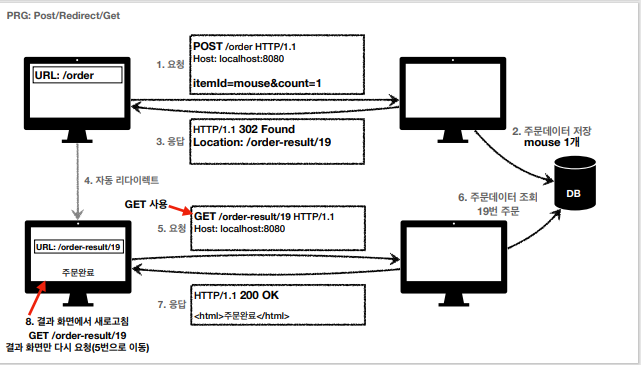
POST가 신규 자원을 등록 할 때, 클라이언트는 등록될 리소스의 URI를 모름.

* 서버가 새로 등록된 리소스 URI를 생성함.
* PUT은 클라가 리소스 URI를 알고 있어야함...



HTML FORM에서 /new, /edit, /delete 등의 컨트롤 URI를 사용하는 이유: form에는 get,post만 지원.



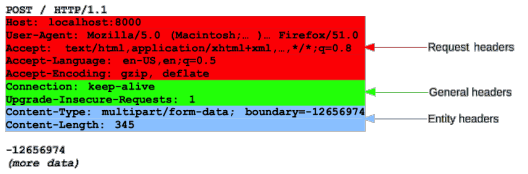


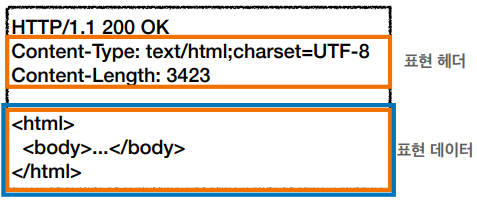
Prg 이후 리다이렉트 해도 URL이 이미 POST -> GET으로 리다이렉트 됨 , 새로 고침해도 GET으로 결과조회

• 302 Found -> GET으로 변할 수 있음 • 307 Temporary Redirect -> 메서드가 변하면 안됨

• 303 See Other -> 메서드가 GET으로 변경 • 304 Not Modified -> 캐시 재사용

HTTP 헤더 : HTTP 전송에 필요한 모든 부가정보





표현 : Content – Type : 미디어 타입, 문자 인코딩

Ex ) text/html; charset-utf-8 application/json image/png

Content-Encoding : 데이터를 읽는 쪽에서 인코딩 헤더의 정보로 압축 해제

협상 : 클라이언트가 선호하는 표현 요청

Accept: 클라가 선호하는 미디어 타입 전달

Accept-Language: 클라가 선호하는 자연 언어 요청

전송 방식 : 압축 전송시에는 Content-Encoding을 보내줘야한다.

분할 전송시에는 Content-Length X

일반 정보 : referrer : 유입 경로 / User-Agent : 클라 앱 정보(브라우저 정보 등등)

Host : 요청한 호스트 정보(도메인) 🡪 한 서버가 여러 개 도메인 돌리는 경우가 있어서 어떤 도메인(애플리케이션)에서 요청한건지 알아야 그에 대한 응답을 줄 수 있기 때문.

쿠키

http는 무상태 프로토콜이기 때문에 쿠키가 없다면 사용자 정보를 매 주소에 get 쿼리로 전달해야함.   
사용처 : 로그인 세션, 광고 추적

생명주기 : expires, max-age

캐시

서버에서 캐시를 담아서(유효시간 포함) 브라우저에 보내면, 브라우저 캐시에 캐시 정보를 저장함.

그곳에서 캐시가 유효한지 검증, 매우 빠르며; 비싼 네트워크 사용량 줄일 수 있음.

If ) 캐시 유효 시간이 초과해서 서버에 다시 요청시 브라우저 캐시에 저장된 캐시를 사용하는 방법 : 검증 헤더;

서버가 보내준 Last-Modified: 2020년 11월 10일 10:00:00 로 보내왔다면 유효 시간 지나면

if-modified-since: 2020년 11월 10일 10:00:00 1. 만약 바뀌었다면 다시 HTTP BODY 받음

2.안바뀌었다면 HTTP/1.1 304 Not Modified 대신 HTTP BODY가 비어있어서 가벼움.

Last-Modified 대신에 Etag 사용 가능 (해시값 사용할 수 있음)

단순히 Etag가 같으면 놔두고 다르면 받기(클라이언트는 캐시 매커니즘을 모름)

프록시 캐시 : 캐시를 담고 있는 중간 서버, 원 서버까지 가면 너무 느릴 수 있으니 중간에서 관리한다. 하지만 Cache-Control: no-cache 처럼 • 데이터는 캐시해도 되지만, 항상 원(origin) 서버에 검증하고 사용하는 제약이 있을 수 있다.

웹 브라우저가 알아서 캐시하는 것 까지 막고 싶다면

• Cache-Control: no-cache, no-store, must-revalidate • Pragma: no-cache

no-cache 와 must-revalidate 차이점 : 노캐시는 원 서버가 검증을 할 수 없으면 그냥 오래된 결과라도 보여주자 / must는 검증 못하면 504 오류 띄워버려야됌 , 중요한 정보이기 때문.