**1. REST API**

**1.1 REST(RESTful)**

REST(Representational State Transfer)는 프로토콜이나 표준이 아닌 아키텍처 원칙 세트로, 자원을 이름으로 구분하여 해당 자원을 상태를 주고받는 모든 것을 의미한다. 즉, REST는 네트워크 상에서 클라이언트와 서버 사이의 통신 방식 중 하나이다. API 개발자는 REST를 다양한 방식으로 구현할 수 있다. 정리하자면 REST는

1. HTTP URI(Uniform Resource Identifier)를 통해 자원(Resource)을 명시하고,

2. HTTP Method(POST, GET, PUT, DELETE)를 통해

3. 해당 자원(URI)에 대한 CRUD Operation을 적용하는 것을 의미한다.

**1.2 REST 특징**

REST는 다음과 같은 특징들을 가지고 있다.

- Server-Client(서버-클라이언트 구조)

- Stateless(무상태)

- Cacheable(캐시 처리 가능)

- Layered System(계층화)

- Uniform Interface(인터페이스 일관성)

**1.3 REST API란**

REST API(RESTful API)란 REST 아키텍처의 제약 조건을 준수하는 애플리케이션 프로그래밍 인터페이스를 뜻한다.

RESTful API를 통해 요청이 수행될 때 RESTful API는 리소스 상태에 대한 표현을 요청자에게 전송한다. 이 정보 또는 표현은 HTTP: JSON(Javascript Object Notation), HTML, XLT 또는 일반 텍스트를 통해 몇 가지 형식으로 전송된다. JSON은 이름에도 불구하고 사용 언어와 상관이 없을 뿐 아니라 인간과 머신이 모두 읽을 수 있기 때문에 가장 널리 사용된다.

API가 RESTful로 간주되려면 다음 기준을 따라야 한다.

(1) 클라이언트, 서버 및 리소스로 구성되었으며 요청이 HTTP를 통해 관리되는 클라이언트-서버 아키텍처를 가진다.

(2) [스테이트리스(stateless)](https://www.redhat.com/ko/topics/cloud-native-apps/stateful-vs-stateless) 클라이언트-서버 커뮤니케이션: 요청 간에 클라이언트 정보가 저장되지 않으며, 각 요청이 분리되어 있고 서로 연결되어 있지 않다.

(3) 클라이언트-서버 상호 작용을 간소화하는 캐시 가능 데이터

(4) 정보가 표준 형식으로 전송되도록 하기 위한 구성 요소 간 통합 인터페이스. 여기에 필요한 것은 다음과 같다.

- 요청된 리소스가 식별 가능하며 클라이언트에 전송된 표현과 분리되어야 한다.

- 수신한 표현을 통해 클라이언트가 리소스를 조작할 수 있어야 한다.

- 클라이언트에 반환되는 자기 기술적(self-descriptive) 메시지에 클라이언트가 정보를 어떻게 처리해야 할지 설명하는 정보가 충분히 포함되어야 한다.

- 하이퍼미디어: 클라이언트가 리소스에 액세스한 후 하이퍼링크를 사용해 현재 수행 가능한 기타 모든 작업을 찾을 수 있어야 한다.

(5) 요청된 정보를 검색하는 데 관련된 서버(보안, 로드 밸런싱 등을 담당)의 각 유형을 클라이언트가 볼 수 없는 계층 구조로 체계화하는 계층화된 시스템을 가져야한다.

(6) 코드 온디맨드(선택 사항): 요청을 받으면 서버에서 클라이언트로 실행 가능한 코드를 전송하여 클라이언트 기능을 확장할 수 있는 기능을 가진다.

**1.4 REST API의 작동 방식**

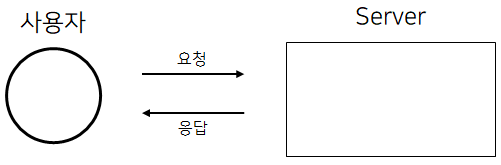
REST API는 웹상에서 사용되는 여러 리소스를 HTTP URI로 표현하고, 해당 리소스에 대한 행위를 HTTP Method로 정의한다. 즉, REST API는 HTTP 요청을 통해 통신함으로써 리소스 내에서 레코드(CRUD 라고도 함)의 작성, 읽기, 업데이트 및 삭제 등의 표준 데이터베이스 기능을 수행합니다. 예를 들어, REST API는 GET 요청을 사용하여 레코드를 검색하고, POST 요청을 사용하여 레코드를 작성하며, PUT 요청을 사용하여 레코드를 업데이트하고, DELETE 요청을 사용하여 레코드를 삭제합니다. 모든 HTTP 메소드는 API 호출에서 사용될 수 있습니다. 잘 디자인된 REST API는 HTTP 기능이 내장된 웹 브라우저에서 실행되는 웹 사이트와 유사합니다.

**2. HTTP 통신**

**2.1 HTTP**

HTTP(Hyper Text Transfer Protocol)는 HTML 파일을 전송하는 프로토콜이라는 의미를 가진다. TCP와 UDP를 사용하여 웹 브라우저에서 통신이 일어나며, 초기에는 HTML 파일을 전송하려는 목적으로 만들어졌으나 현재는 JSON, image 파일 등 또한 전송한다.

**2.2 HTTP 통신 방식**



HTTP 통신은 클라이언트에서 서버로 요청을 보내고 서버가 응답하는 방식으로 통신이 이루어진다. 요청을 보내고 응답을 받을 때 그 정보들을 패킷(Packet)이라는 작은 조각에 실어서 보내게 된다. 응답에는 클라이언트의 요청에 따른 결과를 반환한다. HTTP 통신은 클라이언트의 요청이 있을 때 서버가 응답하는 단방향 통신 특성을 가진다. 즉, 서버가 클라이언트로 요청을 보낼 수는 없다.

요청을 보낼 때, 내용을 기다리는 시간과 함께 연결하는 시간이 들어간다. 초기에는 서버는 응답한 후 클라이언트(사용자)의 Connection을 곧바로 끊어버렸으나, 최근에는 성능상의 이유(Connection을 맺고 끊는 비용이 비싸다)로 Keep Alive 옵션을 통해 일정 기간 동안 클라이언트와 Connection을 유지하는 방식으로 통신이 가능해졌다.

**3. HTTP 통신 과정**

**3.1 브라우저에 URL을 입력했을 때 서버에서 응답하는 과정**

**(1) 브라우저 주소창에 URL을 입력**

URL을 입력받은 브라우저는 먼저 URL의 구조를 해석한다. 어떤 프로토콜로 어느 도메인으로 어떤 포트로 보낼지 해석하는 것이다.



프로토콜(Protocol)은 네트워크 상으로 통신을 할 때 어떤 프로토콜을 사용하여 해당 URL에 요청할 것인지를 명시한다. 가운데의 URL은 해당 웹사이트의 위치(Locator)를 식별하기 위해 사용된다. 마지막으로 포트(Port)는 논리적인 접속 장소를 알려준다. 주로 전송 계층에서 사용된다. 명시적으로 포트를 선언하지 않았다면 브라우저에서는 설정된 기본값을 이용해 요청하게 된다. HTTP라면 80 포트를 HTTPS라면 443 포트를 기본값으로 요청한다.

**(2) 브라우저가 URL의 IP 주소를 찾기 위해 캐시에서 DNS 기록을 확인**

DNS(Domain Name System)는 웹사이트의 IP 주소와 도메인 주소를 연결해주는 시스템이다. DNS의 주요 목적은 쉽게 사이트 주소를 찾을 수 있도록 도와주는 것이다.

DNS 기록을 찾기 위해서, 브라우저는 네 개의 캐시를 확인한다.

* 첫 번째, DNS 쿼리는 우선 브라우저 캐시를 확인한다. 브라우저는 내가 이전에 방문한 웹 사이트의 DNS 기록을 일정 기간 동안 저장하고 있다.
* 두 번째, 브라우저는 OS 캐시를 확인한다. 브라우저 캐시에 원하는 DNS 레코드가 없다면, 브라우저가 내 컴퓨터 OS에 시스템 호출을 통해 DNS 기록을 가져온다.
* 세 번째, 브라우저는 라우터 캐시를 확인한다. 만약 컴퓨터에도 원하는 DNS 레코드가 없다면, 브라우저는 라우터에서 DNS 기록을 저장한 캐시를 확인한다.
* 마지막으로, ISP(Internet Service Provider) 캐시를 확인한다. 만약 위 모든 단계에서 DNS 기록을 찾지 못한다면, 브라우저는 ISP에서 DNS 기록을 찾는다. ISP는 DNS 서버를 가지고 있는데, 해당 서버에서 DNS 기록 캐시를 검색할 수 있다.

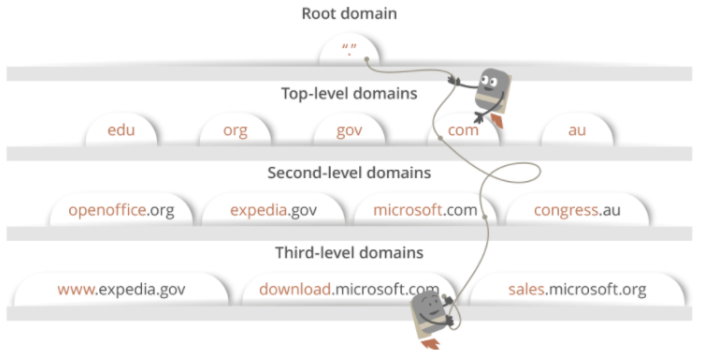
**(3) 만약 요청한 URL이 캐시에 없다면, ISP의 DNS 서버가 DNS 쿼리로 URL을 호스팅하는 서버의 IP 주소를 찾는다**

example.com이라는 주소로는 컴퓨터끼리 통신할 수 없다. 이를 인터넷 상에서 컴퓨터가 읽을 수 있는 IP 주소로 변환해야 서로 통신이 가능하게 된다. 우선 브라우저에서는 자신의 로컬 hosts 파일과 브라우저 캐시에 해당 URL이 존재하는지 확인한다. 존재하지 않는다면 도메인 주소를 IP 주소로 변환해주는 DNS 서버에 요청하여 해당 URL을 IP 주소로 변환한다. DNS 서버는 도메인 네임과 해당하는 IP 주소를 관리하고 있는 서버이다.

앞에서 언급했듯이 내 컴퓨터가 example.com을 호스트하는 서버와 연결하려면 example.com의 IP 주소가 필요하다. DNS 쿼리의 목적은 웹 사이트에 대한 올바른 IP 주소를 찾을 때까지 인터넷에서 여러 DNS 서버를 검색하는 것이다. 필요한 IP 주소를 찾거나, 찾을 수 없다는 오류 응답을 반환할 때까지 한 DNS 서버에서 다른 DNS 서버로 검색이 반복적으로 계속되기 때문에 이 유형의 검색을 재귀적 질의(Recursive Query)라고 한다.

ISP의 DNS 서버를 DNS 리커서(DNS Recursor)라고 부르는데, DNS 리커서는 인터넷의 다른 DNS 서버에 답변을 요청하여 의도된 도메인 이름의 적절한 IP 주소를 찾는 일을 담당한다. 다른 DNS 서버는 웹사이트 도메인 이름의 도메인 아키텍처를 기반으로 DNS 검색을 수행하므로 네임 서버(Name Server)라고 한다.

그렇다면 도메인 아키텍처란 어떻게 구성되어 있을까?



오늘날 많은 웹사이트 URL은 3차 도메인, 2차 도메인 및 최상위 도메인(TLD: Top Level Domain)으로 이뤄진다. 각 단계에는 DNS 룩업(Lookup) 도중에 쿼리되는 고유한 네임 서버가 있다. DNS 룩업이란 DNS 서버에서 인터넷 도메인 이름을 사용해 IP 주소를 알아내는 과정이다. DNS 조회 과정을 정리하면 다음과 같다.

1. 사용자가 웹 브라우저에 example.com을 입력하면, 쿼리가 인터넷으로 이동하고 DNS 재귀 확인자(DNS Recursive Resolver)가 이를 수신한다. 재귀 확인자는 URL이 궁극적으로 필요한 IP 주소로 변환되도록 하는 일련의 쿼리를 시작한다.

2. 이어서 재귀 확인자가 DNS 루트 이름 서버(.)를 쿼리한다.

3. 다음으로, 루트 서버가 도메인에 대한 정보를 저장하는 최상위 도메인(TLD) DNS 서버(예: .com)의 주소로 확인자에 응답한다. example.com을 검색할 경우의 요청은 .com TLD를 가리킵니다.

4. 확인자가 .com TLD에 요청한다.

5. TLD 서버가 도메인 이름 서버(example.com)의 IP 주소로 응답한다.

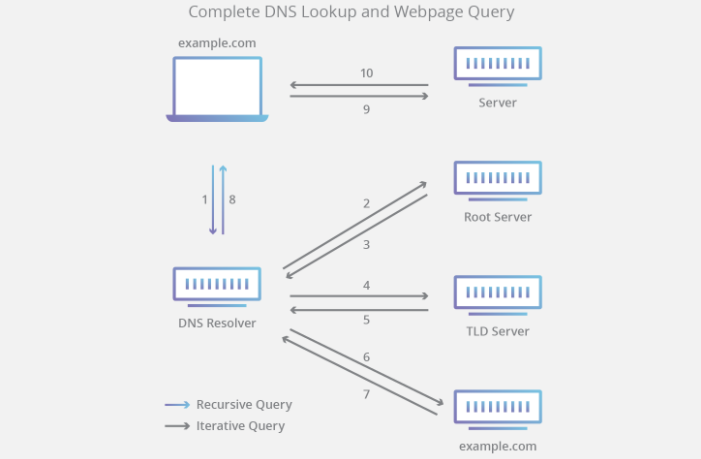
6. 마지막으로, 재귀 확인자가 도메인의 이름 서버로 쿼리를 보낸다.

7. 이제, example.com의 IP 주소가 이름 서버에서 확인자에게 반환된다.

8. 이어서, DNS 재귀 확인자가 처음 요청한 도메인의 IP 주소로 웹 브라우저에 응답한다. DNS 조회의 8단계를 거쳐 example.com의 IP 주소가 반환되면, 이제 브라우저가 웹 페이지를 요청할 수 있다.

9. 브라우저가 IP 주소로 HTTP 요청을 보낸다.

10. 해당 IP의 서버가 브라우저에서 렌더링할 웹 페이지를 반환한다.



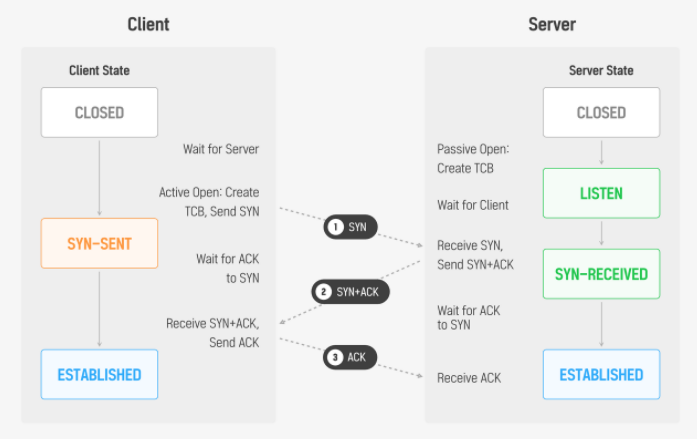
**(4) ARP를 통해 IP 주소를 MAC 주소로 변환**

실질적인 통신을 하기 위해서는 논리 주소인 IP 주소를 물리 주소인 MAC(Media Access Control) 주소로 변환해야 한다. 이를 위해 해당 네트워크 내에서 ARP(Address Resolution Protocol)를 브로드 캐스팅한다. 해당 IP 주소를 가지고 있는 노드는 자신의 MAC 주소를 응답한다. ARP는 IP 주소를 MAC 주소로 변환해주는 프로토콜이다.

**(5) 브라우저가 해당 서버와 TCP 연결을 시작**

브라우저가 올바른 IP 주소를 수신하면 IP 주소와 일치하는 서버와 연결해 정보를 전송한다. 브라우저는 인터넷 프로토콜(IP)을 사용하여 이러한 연결을 구축한다. 사용할 수 있는 여러가지 인터넷 프로토콜이 있지만, 일반적으로 HTTP 요청에서는 TCP(Transmission Control Protocol)라는 전송 제어 프로토콜을 사용한다.

클라이언트와 서버 간에 데이터 패킷을 전송하려면 TCP 연결을 해야 한다. 이 연결은 TCP/IP 3-way handshake라는 연결 과정을 통해 이뤄진다. 이 방식은 서로의 통신을 위한 포트를 확인하고 연결하기 위하여 3번의 요청/응답 후에 연결이 되는 것을 말한다.



1. 클라이언트에서 서버에 연결 요청을 하기 위해 SYN 데이터를 보낸다.

2. 서버에서 해당 포트는 LISTEN 상태에서 SYN 데이터를 받고 SYN\_RCV로 상태가 변경된다.

3. 그리고 요청을 정상적으로 받았다는 대답(ACK)와 클라이언트도 포트를 열어 달라는 SYN을 같이 보낸다.

4. 클라이언트에서는 SYN+ACK를 받고 ESTABLISHED로 상태를 변경하고 서버에 ACK를 전송한다.

5. ACK를 받은 서버는 상태를 ESTABLISHED로 변경한다.

위와 같이 3번의 통신이 정상적으로 이루어지면, 서로의 포트가 ESTABLISHED 되면서 데이터 전송을 위한 TCP 연결이 설정된다.

**(6) 브라우저가 웹서버에 HTTP 요청을 보낸다**

TCP 연결이 설정되면 데이터 전송이 시작된다. 브라우저는 example.com 웹 페이지를 요청하는 GET 요청을 보낼 것이다. 만약 자격 증명(CREDENTIALS)을 입력하거나 form을 제출하는 경우 POST 요청을 사용할 수 있다. 이 요청에는 브라우저 식별(User-Agent 헤더), 수락할 요청 유형(Accept 헤더) 및 추가 요청을 위해 TCP 연결을 유지하라는 연결 헤더와 같은 추가 정보도 포함된다. 또한 브라우저가 이 도메인에 대해 저장한 쿠키에서 가져온 정보도 전달한다.

**(7) 서버가 요청을 처리하고 응답(Response)을 보낸다**

서버에는 웹 서버가 포함되어 있는데, 이는 브라우저로부터 요청을 수신하고, 해당 내용을 request handler에 전달하여 응답을 읽고 생성하는 역할을 한다. Request handler를 요청, 요청의 헤더 및 쿠키를 읽고 필요한 경우 서버의 정보를 업데이트하는 프로그램이다(NET, PHP, Ruby, ASP 등으로 작성됨). 그런 다음 응답을 특정 포맷(JSON, XML, HTML)으로 작성한다.

**(8) 서버가 HTTP 응답을 보낸다**

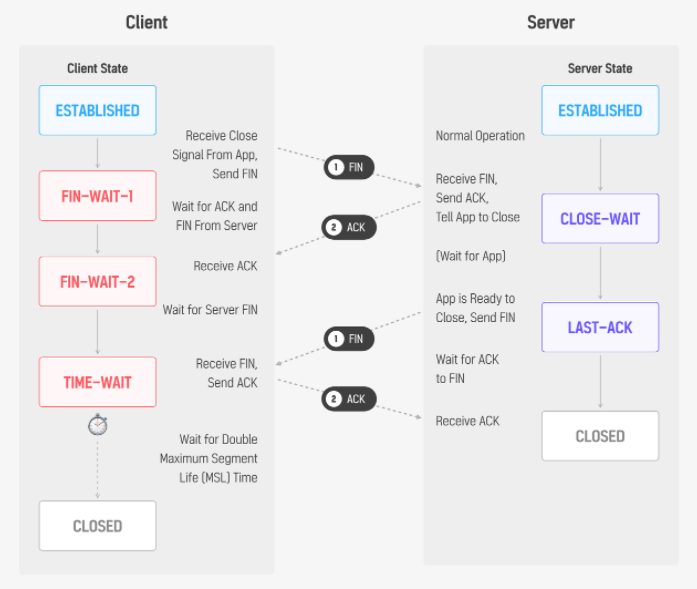
서버 응답에는 요청한 웹페이지와 함께 상태 코드(status code), 압축 유형(Content-Encoding), 페이지 캐싱 방법(Cache-Control), 설정한 쿠키, 개인 정보 등이 포함된다.

**(9) 브라우저가 HTML 컨텐츠를 보여준다**

브라우저는 응답 받은 HTML을 화면에 단계별로 표시한다. 첫째, HTML 골격을 렌더링한다. 그런 다음 HTML 태그를 확인하고 이미지, CSS 스타일시트, 자바스크립트 파일 등과 같은 웹 페이지의 추가 요소에 대한 GET 요청을 보낸다. 정적 파일(Static File)은 브라우저에서 캐싱되므로 다음에 페이지를 방문할 때 다시 가져올 필요가 없다. 그리고 마지막으로 example.com 페이지가 브라우저에 나타난다.

**(10) 웹서버와 TCP 연결 해제**

연결 해제는 4-way handshake 방식을 통해 이루어진다.



1. 먼저 클라이언트는 FIN을 보내고 FIN\_WAIT1 상태로 대기한다.

2. 서버는 CLOSE\_WAIT로 바꾸고 응답 ACK를 전달한다. 동시에 해당 포트에 연결되어 있는 어플리케이션에 CLOSE를 요청한다.

3. ACK를 받은 클라이언트는 상태를 FIN\_WAIT2로 변경한다.

4. CLOSE 요청을 받은 서버 어플리케이션은 종료 프로세스를 진행하고 FIN을 클라이언트에 보내 LAST\_ACK 상태로 바꾼다.

5. FIN을 받은 클라이언트는 ACK를 서버에 다시 전송하고 TIME\_WAIT로 상태를 바꾼다. TIME\_WAIT에서 일정 시간이 지나면 CLOSED 된다. ACK를 받은 서버도 포트를 CLOSED로 닫는다.