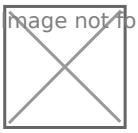
Network

- DNS
- HTTPS
- HTTP/HTTPS and TCP 프로토콜 데이터 전송 및 응답



DNS

레코드

- 레코드는 어디로 route할건지 매핑 명시 하는 것

A

- IPv4
- from example.com to 0.0.0.0

AAAA

- IPv6
- from example.com to ...

CNAME

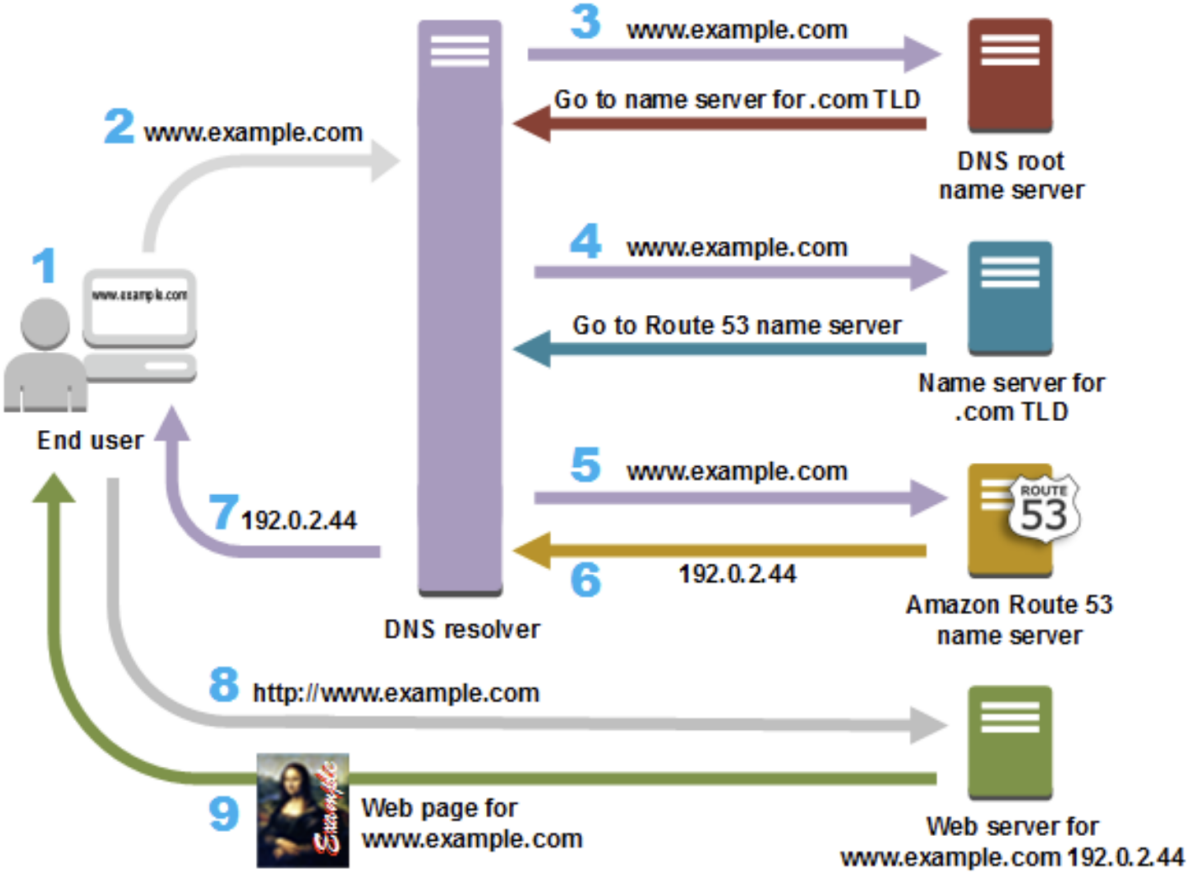
- from domain to domain

네임 서버

- 어떤 네임서버 쓸껀지 현재 도메인의 DNS 호스팅 제공자에게 명시해줘야 함

Image not found or type unknown

클라이언트가 내 도메인에 오기까지



https://aws.amazon.com/route53/what-is-dns/#:~:text=The%20Internet's%20DNS%20system%20works,These%20requests%20are%20called%20queries.

```
# mac□□ resolver □□ □□ □□

scutil --dns
```

```
resolver #1
  search domain[0] : openvpn
  nameserver[0] : 210.220.1.1
  nameserver[1] : 219.250.1.1
  nameserver[2] : 8.8.8.8
  flags    : Request A records
  reach    : 0x00000002 (Reachable)

(...)
```

- resolver □□ □□□ □□

    - Q) resolver 여러개 있는 이유는 뭐지?
- nameserver[0], [1], [2]는 설정된 resolver의 주소

레퍼런스

- AWS LightSail DNS 설정 문서

HTTPS

http 데이터 주고 받는 과정

https 데이터 주고 받는 과정

HTTP/HTTPS and TCP 프로토콜 데이

> 요약
>
> 1. HTTP 는 TCP 상위 프로토콜로 stateless해서 한번 3-way handshake 하고, 요청하고 응답받으면 통신이 종료되는것이다
> 2. HTTP 없이 TCP 프로토콜로만 연결한다는 것은 단순히 바이트 스트림을 주고받을 수 있는 상태를 말하는 것이다. 이때 바이트 스트림을 주고받을 수 있는 매개체로 [

1. 개요

인터넷 통신에서 데이터 전송 및 응답을 담당하는 주요 프로토콜로는 HTTP/HTTPS와 TCP가 있습니다. 이 문서에서는 두 프로토콜의 차이점, 동작 방식, 그리고 자바를 이용한 간단한 예제

2. HTTP/HTTPS

2.1 정의

- **HTTP (HyperText Transfer Protocol)**: TCP 위에서 동작하는 응용 계층 프로토콜로, 주로 웹 브라우저와 서버 간의 통신에 사용됩니다.
- **HTTPS (HTTP Secure)**: HTTP에 보안(SSL/TLS)을 추가한 프로토콜로, 데이터 전송 시 암호화를 제공합니다.

2.2 특징

- **Stateless**: 각 요청과 응답은 독립적이며, 서버는 클라이언트의 이전 요청 상태를 기억하지 않음
- 요청-응답 방식: 클라이언트가 요청을 보내면 서버가 응답을 반환하고, 그 후 연결이 종료

2.3 동작 과정

1. **3-way handshake**: TCP를 통해 연결 설정.
   - TCP 패킷 주고 받음.
2. 요청 전송: 클라이언트가 서버로 HTTP 요청 전송.
3. 응답 생성: 서버가 요청을 처리하고 HTTP 응답 반환.
4. 연결 종료: 응답 완료 후 연결 종료.

3. TCP

3.1 정의

- **TCP (Transmission Control Protocol)**: 신뢰성 있는 데이터 전송을 보장하는 전송 계층 프로토콜로, 데이터를 바이트 스트림 형태로 전송합니다.

3.2 특징

- 신뢰성: 데이터의 손실, 중복, 순서 오류 등을 방지
- 스트림 지향: 바이트 스트림으로 데이터를 처리
- 양방향 통신: 데이터 전송이 가능

3.3 동작 과정

1. 소켓 생성: 클라이언트와 서버가 각각 소켓을 생성합니
2. 연결 요청: 클라이언트가 서버에 연결 요청, 서버는 이를 수락
3. 데이터 전송: 스트림 형태로 데이터를 주고 받음
4. 연결 종료: 데이터 전송이 완료되면 소켓을 닫아 연결을 종료

4. 자바 예제

4.1 서버 코드 (Server.java)

```
import java.io.*;
import java.net.*;

public class Server {
    public static void main(String[] args) {
        int port = 12345;

        try (ServerSocket serverSocket = new ServerSocket(port)) {
            System.out.println("Server is listening on port " + port);

            while (true) {
                Socket socket = serverSocket.accept();
                System.out.println("New client connected");

                new ServerThread(socket).start();
            }

        } catch (IOException ex) {
            System.out.println("Server exception: " + ex.getMessage());
            ex.printStackTrace();
        }
    }
}
```

```
class ServerThread extends Thread {
    private Socket socket;

    public ServerThread(Socket socket) {
        this.socket = socket;
    }

    public void run() {
        try (InputStream input = socket.getInputStream();
             BufferedReader reader = new BufferedReader(new InputStreamReader(input))) {

            String message;

            while ((message = reader.readLine()) != null) {
                System.out.println("Received: " + message);
            }

        } catch (IOException ex) {
            System.out.println("Server thread exception: " + ex.getMessage());
            ex.printStackTrace();
        }
    }
}
```

4.2 □□□□ □□ (Client.java)

```
import java.io.*;
import java.net.*;

public class Client {
    public static void main(String[] args) {
        String hostname = "localhost";
        int port = 12345;

        try (Socket socket = new Socket(hostname, port)) {

            OutputStream output = socket.getOutputStream();
            PrintWriter writer = new PrintWriter(output, true);
```

```
            writer.println("Hello, Server");

        } catch (UnknownHostException ex) {
            System.out.println("Server not found: " + ex.getMessage());
        } catch (IOException ex) {
            System.out.println("I/O error: " + ex.getMessage());
        }
    }
}
```

5. 정리

- **HTTP/HTTPS**는 TCP 기반의 상위 프로토콜로, 요청-응답 방식의 stateless 통신을 수행합니다.
- **TCP** 소켓은 HTTP 없이 순수한 TCP를 사용하여 바이트 스트림을 소켓을 통해 주고받는 방식입니다.
- **TCP** 소켓은 실시간, 스트림 지향, 양방향 통신을 제공합니다.