#cs #study

목차

- 프록시
- 네트워크 복습

프록시

오리진 서버와 중간 서버

- 오리진 서버 : 자원을 생성하고 클라이언트에게 권한이 있는 응답을 보낼 수 있는 HTTP 서버. 네트워크 내부에서 중간 과정을 생략하고 클라이언트가 응답을 받고자 하는 목적지 서버다.
- 가용성(availablity) : 주어진 기능을 문제 없이 수행하는 시간의 비율. 높으면 고가용성(HA)이라고 별도로 부른다. 오리진 서버가 가용성이 높을 수록 당연하게도 서버 효율성이 좋다.
- 프록시(proxy) : 포워드 프록시(forward proxy)라고 부르기도 하며 클라이언트가 선택한 메시지 대리자. 캐시 저장, 클라이언트 암호화 및 접근 제한 등의 기능 제공.
- 게이트웨이(gateway) : 리버스 프록시(reverse proxy)라고 부르기도 하며 오리진 서버로 오는 요청 메시지를 사전에 받는 문지기 역할을 수행한다. 캐시 저장 및 부하 분산을 위한 로드 밸런서 기능도 보유한다.

고가용성을 유지하기 위한 노력

- 가용성은 전체 가동 시간 중에 실제 가동 시간의 비율이다. 이 때, 실제 가동 시간은 uptime, 가동 불가 시간은 downtime이라 부른다. 가용성이 높아 안정적이라고 평가 받는 서버의 가용성 비율은 보통 99.999% 이상이 목표다.
- 결함 감내(fault tolerance) : 가동 불가 시간이 발생하는 원인은 너무 많다. 해킹, 하드웨어 장애, 천재지변 등 문제의 원인 자체를 제거하는 건 힘들기에 문제가 발생하도 버틸 수 있는 내구성을 강화하는 게 중요하다. 이를 결함 감내 능력이라 하며 대표적인 방법은 서버 다중화다.
- 헬스 체크(health check) : 로드 밸런서를 이용해 연결된 서버에 특정 메시지를 보내서 응답을 기다린다. 일정 시간 내에 원하는 답변을 받지 못하면 서버에 문제가 생긴 것으로 간주하고 해결 조치.
- 로드 밸런싱
  - 서버의 트래픽을 고루 분산시키기 위한 기술. 로드 밸런서를 이용해서 수행하며 하드웨어로는 L4 스위치, L7 스위치가 있으며 소프트웨어로는 대표적으로 Nginx가 있다.
  - 로드 밸런싱 알고리즘
    - 라운드 로빈(round robin) : 서버를 돌아가며 부하를 전달한다.
    - 최소 연결(least connection) : 연결이 적은 서버부터 우선적으로 부하 전달.
    - 그 외, 해시 테이블 또는 무작위로 서버를 선정하여 부하를 전달하는 방법도 있음.
  - 서버의 성능에 따라 가중치를 부여하여 부하 전달 시 서버에 우선순위를 부여할 수 있음.
- 스케일링(scaling)

- 수직적 확장(scale-up) : 기존 인프라 구성 부품을 좀 더 나은 사양으로 교체. 부픔 갈아끼우면 그만이니까 설치와 구성이 쉽지만 유연성이 떨어진다.
  - 예시) RTX3080 -> RTX4080
- 수평적 확장(scale-out) : 기존 부품을 여러 개로 구성하는 것. 부품 하나가 고장나면 다른 부품으로 부하를 가중시켜서 해결 할 수 있으므로 유연성이 높지만 반대로 여러 부품에 대한 설정과 우선순위를 고려하면 설치와 구성 자체가 어려울 수 있다.
  - 예시) RTX3080 1개 -> RTX3080 3개
- 자동 확장(autoscaling) : 자동으로 시스템이 정한 기준을 넘으면 인프라 자원을 확대하는 기술. 클라우드 기반 서비스에서 주로 제공한다.

네트워크 복습

네트워크 정의

- 네트워크는 일종의 지도다. 지도에는 거점이 있고 거점으로 흐르는 길이 있듯이 유무선으로 연결된 통로가 있고 이 통로를 통해 정보를 주고 받는다. 그리고 각 거점은 컴퓨터, 스마트폰 등 실제로 정보를 주고 받을 수 있는 개체이며 각 거점은 네트워크에서 사용하는 주소를 받는다.
- 모든 지도에는 규격이 있고 약속된 기호와 규칙이 있다. 네트워크 또한 마찬가지이며 현대적인 네트워크 규격을 하나의 추상화된 개념도로 만들어 놓은 것이 OSI 7계층이다.

OSI 7계층

- 물리 계층(physics layer, 1계층) : 허브로 정의하는 계층. 전기 신호를 받아 일차적으로 정보로 변환하는 곳이다. 하드웨어에 전자가 진입하면 0과 1의 비트 신호로 변환하여 "허브" 장치에 연결된 기기에 전역으로 비트 신호를 뿌리거나 반대로 신호를 수용한다.
- 데이터 링크 계층(datalink layer, 2계층) : 스위치로 정의하는 계층. 허브는 전기 신호를 비트 신호로 변환하는 일 밖에 하지 않기에 연결된 기기를 각각 구분하지 못한다. 또한, 송신과 수신을 양쪽 다 동시에 수행할 수 없기에 잘못 꼬이면 충돌이 발생할 수 있다. 이를 극복하기 위해 2계층은 동시 송수신이 가능하며 각 기기들을 NIC를 이용하여 MAC 주소로 개체를 구분하게 만들었다.
- 네트워크 계층(network layer, 3계층) : 라우터로 정의하는 계층. 2계층의 단점은 대규모 네트워크를 구성할 때 모든 MAC 주소를 기억하기는 힘든 점이다. 또한, MAC 주소는 NIC 장비란 하드웨어에 귀속되는 만큼 스위치 중심으로 계층적 구조를 만들 기도 힘들다. 그렇기에 주소를 쉽게 관리하기 위해 별도의 추상화된 주소 체계를 만든 것이 IP다. 3계층에서는 각 네트워크 그룹을 라우터로 분리하여 라우터에서 자신이 속한 IP 그룹의 주소 테이블을 갖고 있다. 이를 통해 라우터 별로 주소를 그룹으로 묶어서 주소 특정하기 쉽고 경로 설정이 용이하게 만든 것이다.
- 전송 계층(transport layer, 4계층) : TCP/UDP로 정의하는 계층. 네트워크 계층에서 주소까지 특정했으니 이제 정보 전송도 편해졌다. 하지만 내가 열심히 만들어 놓은 정보가 잘 목적지까지 갔는 지 검증하는 과정이 없는 게 문제다. 그렇기에 정보의 검증을 위해 전송 계층이 생겼으며 목적지를 명확히 하기 위해 신규 주소로 포트가 추가되었다.
  - 포트는 목적지인 네트워트 기기에서 어떤 프로그램이 정보를 받을 것인지 규정한다. 운영체제를 공부하면서 알았겠지만 컴퓨터 내부에서 정보를 원하는 프로세스는 넘친다. 그렇기에 어느 프로세스가 정보를 받을 지 알아야 정확한 곳에 정보를 줄 수 있다. 각 프로그램에 번호를 부여한 것이 포트이며 우리는 이 포트를 통해 정확히 어느 프로세스에 네트워크 정보가 도달하는 지 안다.

    - 목적지 규정 후에는 정확히 정보가 목적지까지 전달되었는 지 확인이 필요하다. 이 때, 신뢰성 검증을 하기 위해 쓰는 프로토콜이 TCP와 UDP다. 자, 네트워크를 지도라고 하고 TCP와 UDP를 집배원, 정보를 배송 상품이라 가정하자. TCP 집배원은 정확히 해당 목적지까지 물품을 전달하고 중간에 하자가 생기면 재배송 요청까지 수행한다. UDP 집배원은 목적지까지 전송하지 않고 목적지에 속한 지역을 관할하는 우체국이나 관리사무소로 물품을 인계한다. 즉, TCP는 정보의 검증을 몇 단계 씩 수행하기에 신뢰성이 높지만 그만큼 시간이 걸리고 UDP는 전송 경로가 짧고 간소화된 만큼 빠르지만 신뢰성은 TCP보다 떨어진다. 어느 쪽을 사용할 지는 신뢰성과 신속성 중 어느 쪽을 더 중시해야 되는 상황인 지 파악할 필요가 있다.
- 세션 계층(session layer, 5계층) : 이제 정보 시 신뢰성 검증 수단까지 마련했다. 5계층에서는 이제 언제 연결하고 언제까지 연결을 지속할 것인지 정한다. 가령, 네트워크에서 사용자끼리 채팅창을 열고 대화할 때 언제까지 채팅을 유지하면서 메시지를 서로 나눌 것인지 정하고 연결을 관리하는 게 세션이다. 마치 물품 배송 시작부터 끝까지 집배원의 배송 경로가 기록되는 것과 비슷하다.
    - 연결 과정 전반을 관리하는 건 곧 연결 중에 이루어지는 정보 처리도 관여하는 것이다. 즉, 전송 계층이 현재 전달되는 정보의 신뢰성을 검증하면 세션 계층은 그 위에서 정보를 주고 받는 전체 흐름을 관장한다.
    - 정보의 흐름을 관장하기에 중간 지점마다 정보를 동기화할 수 있는 영역을 설정하고 오류가 발생한 경우에는 가장 가까운 동기화 지점을 끌고 와서 복구한다. 네트워크를 통해 실시간으로 동영상을 볼 때 중간에 잠깐 버퍼 전송 문제가 발생해도 곧바로 그 지점에서 영상을 재생할 수 있는 게 이 계층이 있기 때문이다.(아쉽게도 현실의 집배원은 배송 경로 중간에 물품이 망가지더라도 물품을 복구할 수 없다. 물품을 복구할 수 있는 3D 프린터와 설계도라도 들고 다니지 않는 이상에야)
- 표현 계층(expression layer, 6계층) : 정보 전달 시 맺고 끊음까지 관장하게 되었다. 이제 정보를 전달받은 사용자가 무슨 정보인지 해석할 수 있어야 한다. 6계층은 정보 해석법에 대해 논한다.
    - 다시 택배로 예시를 들어보자. 우리는 당연하게도 배송을 시키면 해외 배송이 아닌 국내 배송이라면 출발지와 목적지 전부 한국어로 쓰여 있다. 또한, 배송 물품을 어떻게 사용하는 지 물품이 무엇인 지 알고 있다. 왜냐하면 그렇게 보내기로 약속했기 때문이다.(혹시라도 컴퓨터를 주문했는데 벽돌이 왔는가? 네트워크에서도 시키지 않은 악성 코드가 넘어오는 경우가 있다!)
    - 표현 계층에서도 어떤 방식으로 보내는 정보를 해석해야 하는 지 알려준다.
- 응용 계층(application layer, 7계층) : 이제 정보를 해석하는 것도 가능하다. 그렇다면 이제 그 정보를 원하는 대로 사용하는 일만 남았다. 7계층은 직접적으로 정보를 처리하며 여기서 가장 대표적인 프로토콜이 HTTP다.

각 계층에서 정보를 포장하는 방식

- 1계층은 정보의 원시 데이터, 2계층은 출발지와 목적지의 MAC주소, 3계층은 출발지와 목적지의 IP주소 등. 각 계층마다 전송하고자 하는 정보가 존재한다. 그렇다면 목적지에서 정보를 분석하려면 당연하게도 출발지와 목적지가 정보를 포장하고 푸는 방식에 대한 공통 규칙이 있어야 한다.
- 1계층은 전기 신호를 비트 신호로 해석한 원시적인 데이터 밖에 없으므로 추상화된 정보 집합 구조가 없다. 이는 2계층부터 형성되는 데 2계층부터 7계층까지 전부 정보를 헤더, 페이로드, 트레일러로 나누어서 포장 및 분해한다.
- 헤더는 페이로드에 들어간 데이터의 크기, 해석방식, 송수신지 식별 정보 등 각 계층마다 필요한 정보를 넣는다. 페이로드는 목적지로 전달할 본문 + 이전 계층의 정보를 포함한다. 그렇기에 이전 계층의 헤더 + 페이로드 + 트레일러가 전부 페이로드에 들어간다. 트레일러에는 부가 정보를 넣으며 주로 오

류 검출용 정보를 포함한다. 트레일러 부분의 내용물을 본문과 비교하여 정보 손실이 있는 지 검증하는 식이다.

- 위와 같이 이전 계층의 헤더 + 페이로드 + 트레일러가 한묶음으로 현 계층의 페이로드로 들어가는 걸 캡슐화라 한다. 반대로 페이로드에 묶인 정보를 풀어 헤쳐서 분석하는 걸 역캡슐화라 한다.
- 캡슐화 과정
  - HTTP 프로토콜을 통해서 서버로 정보 요청 -> 세션에서 요청된 서버로의 연결 확인 -> TCP 프로토콜을 통해 정보 검증용 비트 구성 -> IP주소를 확인하고 라우터로 이동하여 목적지 주소로 전송 -> 공유기(스위치)를 통해 스위치에 묶여있는 기기 중에서 해당 IP주소와 매칭되는 MAC 주소 확인 -> 허브를 통해 디지털 신호를 전기 신호로 변환하여 입력 장치를 통해 정보 해석
- 역캡슐화 과정
  - 전기 신호를 디지털 신호로 전환하여 NIC로 정보 전달 -> MAC 주소 확인하고 FCS 검사 -> IP 주소 검사 -> TCP 오류 비트를 통해 신뢰성 검증 -> 세션을 통해 연결 및 정보 전송 여부 확인 -> 표현 계층 헤더에 포함된 인코딩 방식을 확인하고 정보 해석 -> 해석된 정보를 통해 정보 처리하고 필요하면 HTTP 프로토콜을 통해 응답 메시지 전달 준비

Quiz

- 철수가 게임을 하다가 같은 팀원이랑 시비가 붙게 되었다. 그 팀원이 현란한 언변으로 철수를 농락하고 철수는 분을 못 참고 욕을 날리려던 찰나 팀원과 연결이 끊어졌다. 머릿속에서 꺼내지 못한 욕들이 계류하며 철수의 분이 더 커졌으나 어쩌겠는가. 이 때, OSI 7계층 중, 어느 계층에서 접속 종료 문제가 발생한 걸 알 수 있는가?
- 영희는 클라우드 엔지니어다. 고객이 현재 체계를 최대한 유지한 선에서 가용성을 높이고 서버에서 피치 못한 오류가 발생할 시에 신속한 대응을 할 수 있는 체계를 요구했다. 별도의 확장 방식 없이 단일 소규모 서버를 운영 중인 회사의 요구를 수용하여 영희가 제시할 수 있는 의견을 작성하시오.