8단원. 실무 네트워크 (로드밸런싱·트러블슈팅·클라우드)

> 대상: 전공자 · 정보처리기사 취득자 | 목적: 인프라 직무 기술 면접 대비 중요도: ★★★ 최빈출 · ★★ 자주 출제 · ★ 기본 개념

Q1. 로드밸런서의 알고리즘(라운드로빈, 최소연결, 해시)을 설명하세요. ★★★

답안 **라운드 로빈**은 서버에 순서대로 분배합니다. 단순하지만 서버 성능·부하가 다르면 불균형해질 수 있어 **가중 라운드 로빈**으로 보완합니다. **최소 연결(least connections)**은 현재 활성 연결이 가장 적은 서버로 보내, 처리 시간이 제각각인(장수 연결이 섞인) 워크로드에 유리합니다. **IP/URL 해시**는 출발지 IP나 URL을 해시해 항상 같은 서버로 보내므로 캐시 지역성이나 세션 고정에 좋지만, 서버 증감 시 매핑이 대거 흔들리는 문제가 있어 **consistent hashing**으로 완화합니다. 추가로 **최소 응답시간** 방식도 있습니다. 워크로드 특성(연결 수명, 상태 유무)에 맞춰 고르는 것이 핵심입니다.

관련 개념 가중치, 최소 연결, consistent hashing, 최소 응답시간, 헬스체크 연동

Q2. L4 로드밸런서와 L7 로드밸런서의 차이는 무엇인가요? ★★★

답안 **L4 LB**는 전송 계층(TCP/UDP)에서 IP·포트만 보고 분산합니다. 패킷을 거의 그대로 전달해 매우 빠르고 처리량이 높으며, 어떤 프로토콜이든 다룰 수 있지만 콘텐츠 기반 라우팅은 못 합니다(AWS NLB). **L7 LB**는 응용 계층(HTTP)까지 해석해 **URL 경로·호스트 헤더·쿠키 기반 라우팅**, SSL 종료, 헤더 조작, 압축, 재작성이 가능합니다(AWS ALB, nginx, HAProxy). 대신 매 요청을 파싱해 오버헤드가 큽니다. 마이크로서비스에서 /api/users는 A 서비스, /api/orders는 B 서비스로 보내는 경로 기반 라우팅이 L7의 대표 활용입니다. 선택 기준: 단순 고성능 TCP는 L4, HTTP 지능형 라우팅은 L7.

관련 개념 NLB vs ALB, SSL 종료, 경로/호스트 기반 라우팅, 리버스 프록시

Q3. 로드밸런서 뒤에서 클라이언트 실제 IP를 어떻게 알 수 있나요? ★★★

답안 LB나 프록시를 거치면 서버에는 **LB의 IP**가 출발지로 찍혀, 로그·접근제어·지역 판별이 무의미해집니다. 해결책은 ① **X-Forwarded-For(XFF)** 헤더: L7 프록시가 원 클라이언트 IP를 헤더에 추가합니다(여러 단 거치면 목록이 됨, 신뢰 경계 밖 값은 위조 가능하니 신뢰 프록시가 덮어써야 함). ② **X-Forwarded-Proto**로 원래 스킴(https) 전달. ③ TCP 레벨(L4)에서는 헤더를 못 쓰므로 **Proxy Protocol**로 원 IP를 앞에 붙입니다. 서버·앱은 이 헤더를 신뢰할 프록시에서 온 경우에만 채택하도록 설정해야 하며, 잘못 신뢰하면 IP 위조로 접근제어가 우회됩니다.

관련 개념 X-Forwarded-For/Proto, Proxy Protocol, 신뢰 프록시 설정, IP 스푸핑 위험

Q4. 헬스체크는 왜 중요하고 어떻게 설계하나요? ★★

답안 LB는 헬스체크로 각 서버의 상태를 감시해 **비정상 서버를 자동으로 제외**하고 정상 서버로만 트래픽을 보냅니다. **얕은 체크**(TCP 포트 열림, HTTP 200)는 가볍지만 "포트는 살아있는데 DB 연결이 끊긴" 상태를 놓칩니다. **깊은 체크**(/

health가 DB·의존 서비스까지 점검)는 정확하지만, 공통 의존성(DB)이 흔들리면 모든 서버가 동시에 unhealthy가 되어 전체 장애가 될 수 있어 균형이 필요합니다. 파라미터로 주기, 타임아웃, 임계 횟수(연속 몇 번 실패해야 제외)를 조정합니다. 배포 시에는 헬스체크 통과 후 트래픽을 받게 해(그레이스풀) 초기화 중 요청 실패를 막습니다.

관련 개념 얕은/깊은 체크, /health 엔드포인트, 그레이스풀 셧다운/스타트업, 캐스케이딩 장애

Q5. 포워드 프록시와 리버스 프록시의 차이는? ★★

답안 포워드 프록시는 **클라이언트 앞**에 서서 내부 사용자의 외부 요청을 대신 수행합니다. 사내 인터넷 접근 제어, 캐싱, 익명화가 목적이며 서버는 프록시를 클라이언트로 인식합니다. **리버스 프록시**는 **서버 앞**에 서서 외부 요청을 받아 내부 서버로 전달합니다. SSL 종료, 로드밸런싱, 캐싱, 압축, WAF, 백엔드 은닉이 목적이며 클라이언트는 프록시를 서버로 인식합니다(nginx, HAProxy, CDN 엣지). "누구를 대리하느냐"(포워드=클라이언트, 리버스=서버)가 구분의 핵심입니다.

관련 개념 nginx/HAProxy, SSL 종료, 백엔드 은닉, API 게이트웨이

Q6. CDN은 어떻게 동작하고 어떤 이점을 주나요? ★★★

답안 CDN은 전 세계 **엣지 서버**에 콘텐츠를 캐싱해, 사용자가 지리적으로 가까운 엣지에서 받게 합니다. 사용자를 가까운 엣지로 보내는 것은 애니캐스트나 DNS(GSLB)가 담당합니다. 이점은 ① **지연 감소**(RTT 단축), ② **원본 서버 부하 경감**(캐시 히트는 원본까지 안 감), ③ **대역폭 절감**, ④ **DDoS 흡수**와 WAF 제공입니다. 정적 콘텐츠(이미지, JS, 비디오)가 주 대상이지만, 동적 콘텐츠도 엣지까지의 최적화된 연결과 마이크로캐싱으로 가속합니다. 운영 포인트는 **캐시 무효화**(배포 시 오래된 자산 문제 → 파일명 해싱), Cache-Control 설계, 캐시 히트율 모니터링입니다.

관련 개념 엣지 서버, 오리진, 캐시 히트율, 캐시 무효화/퍼지, 엣지 컴퓨팅

Q7. "사이트가 느려요"라는 신고를 받으면 어떻게 접근하겠습니까? ★★★

답안 계층적으로 문제를 분리합니다. ① **범위 확인**: 전체인지 특정 사용자·지역·페이지인지(모니터링·다른 지역에서 재현). ② **클라이언트→서버 경로 분해**: 브라우저 개발자도구로 DNS/연결/TTFB/다운로드/렌더링 중 어디가 느린지 봅니다. TTFB가 크면 서버·네트워크, 다운로드가 크면 대역폭·자산 크기, 렌더링이 크면 프론트엔드. ③ **네트워크 구간**: ping/mtr로 RTT·손실, traceroute로 느린 홉 파악. ④ **서버 측**: LB→웹→WAS→DB 각 구간 지연(APM), CPU·메모리·커넥션 풀·슬로 쿼리 확인. ⑤ **최근 변경**: 배포·트래픽 급증·설정 변경 이력. 핵심은 추측하지 말고 **구간별로 측정해 병목을 좁히는 것**입니다.

관련 개념 TTFB, 개발자도구 워터폴, mtr, APM, 병목 구간 분리

Q8. 네트워크 트러블슈팅 도구(ping, traceroute, dig, tcpdump, ss)의 용도를 설명하세요. ★★★

답안 ping은 도달성·RTT·손실 확인(단, ICMP 차단 시 무의미). **traceroute/mtr**는 경로상 각 홉의 지연·손실을 확인해 어디서 끊기는지 파악(mtr은 연속 측정). **dig/nslookup**은 DNS 조회로 이름 해석 문제 진단(dig +trace로 위임 경로

추적). **tcpdump/Wireshark**는 실제 패킷을 캡처해 핸드셰이크, 재전송, RST, TLS 오류 등을 눈으로 확인하는 최종 병기. **ss(netstat 후속)**는 로컬 소켓 상태(LISTEN 여부, 연결 상태 분포, 큐 길이) 확인. **curl -v**는 HTTP·TLS 단계별 진단. 문제 유형에 따라 "이름 해석은 dig, 경로는 mtr, 실제로 오가는 건 tcpdump, 내 소켓은 ss"로 도구를 매핑해 답하면 실무 감각을 보여줄 수 있습니다.

관련 개념 mtr, dig +trace, tcpdump 필터, ss -tanp, Wireshark

Q9. 3-tier 아키텍처의 네트워크 구성을 설명하세요. (인프라 연계) ★★

답안 웹(프레젠테이션) - 애플리케이션(WAS) - DB 3계층을 네트워크로 분리하는 표준 설계입니다. 인터넷 → 로드밸런서 → **퍼블릭/DMZ 서브넷의 웹 계층** → **프라이빗 서브넷의 앱 계층** → **더 격리된 프라이빗 서브넷의 DB 계층** 순으로, 각 계층 사이에 방화벽/보안 그룹으로 **필요한 포트만** 허용합니다(웹은 443만 인바운드, 앱은 웹에서 오는 8080만, DB는 앱에서 오는 3306만). DB는 인터넷 아웃바운드도 차단합니다. 목적은 **최소 권한**과 **침해 확산 차단**(웹이 뚫려도 DB에 직접 못 감)입니다. 클라우드에서는 서브넷·보안 그룹·NACL로 이 경계를 구현합니다.

관련 개념 DMZ, 계층별 서브넷, 최소 권한, 배스천 호스트, 심층 방어

Q10. 클라우드 네트워킹 핵심 요소(VPC, 서브넷, IGW, NAT GW, 피어링)를 설명하세요. ★★★

답안 VPC는 클라우드 안의 격리된 사설 네트워크(CIDR 지정)입니다. **서브넷**은 VPC를 가용영역·용도별로 나눈 것으로, 라우트 테이블에 인터넷 경로가 있으면 퍼블릭, 없으면 프라이빗입니다. **IGW(인터넷 게이트웨이)**는 VPC를 인터넷에 연결하며, 퍼블릭 서브넷의 공인 IP 인스턴스가 양방향 통신하게 합니다. **NAT 게이트웨이**는 프라이빗 서브넷 인스턴스가 **아웃바운드만** 인터넷에 나가게 합니다(패치·API 호출용, 인바운드는 불가). **VPC 피어링/Transit Gateway**는 VPC 간(또는 온프레미스와) 사설 연결을 제공하며, 이때 CIDR가 겹치면 안 됩니다. 이 요소들의 조합이 클라우드 인프라 면접의 핵심 주제입니다.

관련 개념 VPC/서브넷/라우트 테이블, IGW vs NAT GW, VPC 피어링/TGW, VPC 엔드포인트(프라이빗 링크)

Q11. TCP 연결이 간헐적으로 끊기는 문제, 원인 후보들을 나열해보세요. ★★

답안 ① **NAT/방화벽 세션 타임아웃**: 유휴 장수 연결을 중간 장비가 조용히 정리 → keepalive를 타임아웃보다 짧게. ② **LB 유휴 타임아웃**: LB가 무통신 연결을 끊음(웹소켓·DB 연결에 흔함) → 타임아웃 상향 또는 애플리케이션 keepalive. ③ **MTU/PMTUD 블랙홀**: 큰 패킷만 실패 → ping -s 테스트, MSS 클램핑. ④ **비대칭 라우팅**으로 상태 기반 방화벽이 응답 차단. ⑤ **간헐적 패킷 손실**(불량 케이블·과부하 링크) → mtr 장시간 관찰. ⑥ 서버 측 **커넥션 풀 만료**나 리소스 고갈. ⑦ IP 충돌·DHCP 임대 문제. tcpdump로 끊기는 순간의 FIN/RST 여부를 보면 "누가 먼저 끊었는지"를 특정할 수 있어 원인 범위를 크게 좁힙니다.

관련 개념 세션 타임아웃, keepalive, PMTUD 블랙홀, 비대칭 라우팅, RST 분석

Q12. 마이크로서비스에서 서비스 디스커버리와 서비스 메시가 필요한 이유는? ★★

답안 MSA에서는 서비스 인스턴스가 오토스케일링·배포로 **동적으로 뜨고 지므로** IP를 하드코딩할 수 없습니다. **서비스 디스커버리**(Consul, etcd, 쿠버네티스 DNS)는 서비스 이름으로 현재 살아있는 인스턴스를 찾게 해줍니다. 호출 측이 조회하는 클라이언트 사이드와, LB·프록시가 대신하는 서버 사이드 방식이 있습니다. **서비스 메시**(Istio, Linkerd)는 각 파드에 사이드카 프록시(Envoy)를 붙여 서비스 간 트래픽의 **로드밸런싱·재시도·서킷 브레이커·mTLS 암호화·관측성**을 애플리케이션 코드 수정 없이 인프라 계층에서 제공합니다. 네트워크 복잡성이 앱에서 플랫폼으로 옮겨간 것이 핵심입니다.

관련 개념 서비스 디스커버리, 쿠버네티스 Service/DNS, 사이드카(Envoy), mTLS, 서킷 브레이커

Q13. QoS(서비스 품질)란 무엇이고 언제 필요한가요? ★

답안 QoS는 한정된 대역폭에서 **트래픽에 우선순위를 부여**해 중요한 트래픽의 품질을 보장하는 기술입니다. 음성(VoIP)·영상회의는 지연·지터에 민감하므로 대용량 파일 전송보다 우선 처리해야 합니다. 기법으로 트래픽 분류·마킹(DSCP), 우선순위 큐잉, 대역폭 예약, 트래픽 셰이핑(초과분 지연)·폴리싱(초과분 폐기)이 있습니다. 기업망·통신사망에서 중요하며, 인터넷 전체에는 적용이 어렵습니다(그래서 실시간 앱은 애플리케이션 차원의 적응형 코덱·버퍼로 대응). 클라우드에서는 대역폭·IOPS 등에 대한 리소스 보장·제한으로 유사 개념이 나타납니다.

관련 개념 DSCP 마킹, 우선순위 큐, 셰이핑 vs 폴리싱, 버퍼블로트

Q14. 서킷 브레이커와 재시도, 타임아웃이 네트워크 안정성에 왜 중요한가요? ★★

답안 분산 시스템에서 한 서비스의 장애가 호출 사슬을 타고 전체로 번지는 **캐스케이딩 장애**를 막기 위해서입니다. **타임아웃**이 없으면 느린 응답을 무한정 기다리다 스레드·커넥션이 고갈됩니다(반드시 설정). **재시도**는 일시적 오류를 극복하지만, 무분별하면 이미 힘든 서버에 부하를 가중(재시도 폭풍)하므로 **지수 백오프+지터**와 멱등성 확인이 필요합니다. **서킷 브레이커**는 실패율이 임계치를 넘으면 회로를 '열어' 일정 시간 호출을 즉시 차단(빠른 실패)해 죽어가는 서비스에 회복 시간을 주고, 반쯤 열림 상태로 조금씩 재시도하며 복구를 확인합니다. 이 세 가지가 함께 회복 탄력성(resilience)을 만듭니다.

관련 개념 캐스케이딩 장애, 지수 백오프+지터, 재시도 폭풍, 서킷 브레이커 상태(closed/open/half-open), 벌크헤드
