



LED 기술문서

≡	조원	김주한, 김효섭, 이호진, 임효경, 최민영, 최한주
📅	날짜	@2024년 3월 18일

1. NAT(Network Address Translation)

1.1 NAT

1.2 주요 사용 사례

1.3 NAT의 장점

1.4 NAT의 단점

1.5 관련 자료

2. SSH(Secure Shell) - Connect

2.1 SSH

2.2 주요 사용 사례

2.3 SSH의 장점

2.4 SSH의 단점

2.5 관련 자료

3. SSH-Keygen(비대칭 키 방식)

3.1 SSH-Keygen

3.2 주요 사용 사례

3.3 SSH-Keygen의 장점

3.4 SSH-Keygen의 단점

3.5 관련 자료

4. Bastion-Host

4.1 Bastion-Host

4.2 주요 사용 사례

4.3 장점

4.4 관련 자료

5. Kubernetes

5.1 Kubernetes

5.2 주요 사용 사례

5.3 장점

6. LoadBalancing

6.1 LoadBalancing

6.2 장점

6.3 로드 밸런싱 알고리즘

6.3.1 정적 로드밸런싱

6.3.1-1 라운드 로빈 방식

6.3.1-2 IP 해시 방식

6.3.2 동적 로드밸런싱

6.3.2-1 최소 연결 방법

6.3.2-2 가중치 기반 최소 연결 방법

6.3.2-3 최소 응답 시간 방법

6.3.2-4 리소스 기반 방법

7. Database Replication

7.1 Replication

7.2 Replication 목적

7.3 Replication 동작원리

7.4 Primary DB의 역할

7.5 Secondary DB의 역할

7.6 주요 사용 사례

8. NFS(Network File System)

8.1 NFS

8.2 작동방식

8.3 장점

8.4 단점

8.5 주요 사용 사례

1. NAT(Network Address Translation)

1.1 NAT

네트워크 패킷의 출발지 IP 주소와 목적지 IP 주소를 변환하는 기술이다. NAT를 통해 사설 네트워크와 공인 네트워크 간의 통신을 가능하게 하고, IP 주소의 부족한 문제를 해결할 수 있다.

DNAT은 목적지 주소 변환을 의미하며, 패킷이 목적지로 전달될 때 목적지 주소를 변환하여 해당 서비스로 패킷을 전달하는 기술이다.

1.2 주요 사용 사례

- 로컬 네트워크에 있는 여러 서버가 하나의 공인 IP 주소를 공유하여 외부에서 접속하는 경우에 주로 사용됨.

1.3 NAT의 장점

1. **IP 주소의 효율적 사용:** NAT는 하나의 공용 IP 주소를 사용하여 여러 개의 사설 IP 주소를 인터넷에 연결할 수 있게 해준다. 이는 IPv4 주소가 부족한 상황에서 매우 유용하며, IP 주소의 사용을 최적화한다.
2. **보안 강화:** NAT는 외부 네트워크에서 내부 네트워크의 특정 기기를 직접 식별하고 접근하기 어렵게 만든다. 이는 사설 네트워크의 보안을 강화하는 데 도움이 된다.
3. **네트워크 구성 변경의 용이성:** 내부 네트워크의 IP 구성을 변경해도, 외부와의 연결에 사용되는 공용 IP는 그대로 유지될 수 있다.. 이는 네트워크 관리를 유연하게 만들어 준다.
4. **포트 포워딩을 통한 서비스 제공:** NAT를 사용하면 특정 포트에 대한 트래픽을 내부 네트워크의 특정 서버로 전달할 수 있다. 이를 통해 내부 서버가 외부에서 접근 가능한 서비스를 제공할 수 있게 된다.

1.4 NAT의 단점


1. **엔드 투 엔드 연결성 저하:** NAT는 데이터 패킷의 출발지와 목적지 주소를 변경함으로써 엔드 투 엔드의 원칙을 침해한다 이는 특정 애플리케이션과 프로토콜에서 호환성 문제를 일으킬 수 있다.
2. **성능 저하:** 모든 패킷의 주소 변환 과정은 추가적인 처리 시간을 요구하며, 이는 네트워크의 성능 저하로 이어질 수 있다.
3. **공용 IP 주소의 한계:** NAT는 공용 IP 주소 하나로 여러 개의 사설 IP 주소를 관리한다. 고성능의 대규모 네트워크에서는 이로 인해 병목 현상이 발생할 수 있다.
4. **트래픽 모니터링 및 로깅의 복잡성 증가:** NAT를 사용하는 경우, 외부 IP 주소와 내부 IP 주소 간의 매핑을 추적해야 한다. 이는 네트워크 트래픽의 모니터링과 로깅을 복잡하게 만들 수 있다.

1.5 관련 자료

- CISCO의 공식 페이지에서도 내부 사설 IP의 주소를 숨길 수 있는 NAT의 기능에 대해 소개하고 있음.

NAT를 사용하여 CTC의 실제 IP 주소를 숨겨 ONS 15454로 세션 설정

이 문서에서는 CTC(Cisco Transport Controller)와 ONS 15454 간에 세션을 설정하기 위한 NAT(Network Address Translation)의 샘플 컨피그레이션을 제공합니다. CTC가 방화벽 내에 있을 때 컨피그레이션은 NAT를 통해 CTC의 실제 IP 주소를 숨깁니다.

 https://www.cisco.com/c/ko_kr/support/docs/optical-networking/ons-15454-sdh-multiservice-provisioning-platform-mspp/65122-15454toctc-nat.html

https://www.cisco.com/c/ko_kr/support/docs/optical-networking/ons-15454-sdh-multiservice-provisioning-platform-mspp/65122-15454toctc-nat.html

2. SSH(Secure Shell) - Connect

2.1 SSH

네트워크 프로토콜을 사용하여 두 컴퓨터 간에 암호화된 통신 채널을 설정하는 방법으로, 이를 통해 사용자는 안전하게 원격 컴퓨터에 로그인하고, 파일을 전송하거나 네트워크 서비스를 안전하게 사용할 수 있다. SSH는 주로 원격 서버 관리 및 안전한 파일 전송에 사용된다.

2.2 주요 사용 사례

- 원격 서버에 접속하여 파일 전송, 실행, 관리 등을 수행하는 경우에 사용됨.
- 원격으로 실행되는 애플리케이션 또는 서비스를 모니터링하고 관리하는 경우에 사용됨.
- 보안 강화를 위해 원격 접속 시 SSH를 사용하는 경우에 사용됨.

2.3 SSH의 장점

1. **보안 강화** : 원격 서버에 접속하여 파일 전송, 실행, 관리 등을 수행하는 경우, 원격으로 실행되는 애플리케이션 또는 서비스를 모니터링하고 관리하는 경우에 도움이 된다.
2. **원격 제어** : 원격 시스템에 명령을 실행하고 관리할 수 있다.
3. **다양한 운영 체제 지원** : 다양한 운영 체제에서 SSH를 지원하므로 다양한 환경에서 사용할 수 있다.

2.4 SSH의 단점

1. 설정과 관리가 복잡할 수 있다. 특히, 공개 키 인증 같은 보다 안전한 인증 방식을 사용할 때, 키 관리가 부담스러울 수 있다.
2. 초기 설정 때 호스트 키 검증 과정에서 사용자가 서버의 공개 키를 수동으로 검증해야 하는 경우가 있는데, 이 과정을 소홀히 하면 중간자 공격에 취약해질 수 있다.
3. SSH 연결은 암호화 과정 때문에 약간의 지연이 발생할 수 있어, 매우 높은 실시간 성능이 필요한 애플리케이션에는 적합하지 않을 수 있다.
4. 비밀번호를 사용한 인증 방식은 브루트 포스 공격에 취약할 수 있어, 가능하면 피하는 게 좋다.
5. 네트워크 구성이나 방화벽 설정에 따라 SSH 접속이 차단될 수도 있고, 이럴 때는 추가적인 설정이나 터널링 기법을 사용해야 한다.
6. 공개 키 인증을 사용할 때, 개인 키를 잃어버리면 접속할 수 없게 되므로, 키 관리에 신경 써야 한다.

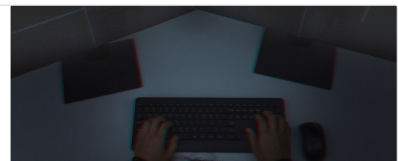
2.5 관련 자료

- SSH연결이 브루트 포스 공격에 당해 악성 채굴 코드 등을 심는 보안사고가 발생한 사례

무작위 대입, 채굴 악성코드 등 해커 공격에 SSH 서비스 '몸살'

SSH(Secure Shell)를 타깃으로 한 무작위 대입 시도가 지속적으로 발견되고 있어 SSH 서비스 운영자들의 주의가 요구된다. SSH는 보통 리눅스 서버를 원격에서 접속하는데 사용되는 서비스다. 그런데 해커가 이러한 서비스를 노리고 무작위 대입 시도를 한 후, 채굴 악성코드 등을 심어 악의적인 행위를 하는 것으로 드러났다.

 <https://m.boanews.com/html/detail.html?idx=109910>



3. SSH-Keygen(비대칭 키 방식)

3.1 SSH-Keygen

SSH 클라이언트에서 사용자의 공개 및 비밀 키를 생성하는 데 사용되는 명령 줄 도구다. 이 키는 SSH 프로토콜을 사용하여 안전한 원격 접속에 사용된다. SSH 키 쌍은 공개 키와 비밀 키로 구성되며, 비밀 키는 사용자가 보유하고 있어야 하며 공개 키는 원격 서버에 저장되어 있어야 한다.

3.2 주요 사용 사례

- SSH 접속을 위한 키 쌍을 생성하는 경우에 사용한다.
- 원격 서버에 공개 키 등록하는 경우에 사용한다.
- SSH 인증을 사용하여 보안 강화하는 경우에 사용한다.

3.3 SSH-Keygen의 장점

1. **강력한 보안** : SSH 키를 사용하여 비밀 키를 암호화하고 인증하는 데 사용되므로 보안성이 높다.
2. **비밀 키 보안** : 사용자가 개인 키를 안전하게 보관함으로써 외부로부터의 침입을 방지한다.

3.4 SSH-Keygen의 단점

1. **키 관리 번거로움** : 사용자가 생성한 키 쌍(공개 키와 개인 키)을 안전하게 관리해야 한다. 개인 키를 잃어버리면 접근 권한을 잃게 되고, 공개 키는 서버에 제대로 등록되어야 한다.
2. **보안 위험** : 개인 키가 노출되면, 해당 키를 가진 누구나 사용자의 계정에 접근할 수 있다. 따라서 개인 키는 반드시 안전한 곳에 보관해야 하며, 가능하면 passphrase로 추가 보호해야 한다.
3. **복잡한 초기 설정** : 처음 SSH-Keygen을 사용하여 키를 생성하고, 서버에 공개 키를 등록하는 과정이 초보자에게는 다소 복잡하게 느껴질 수 있다.
4. **서버 설정 요구** : 공개 키 인증 방식을 사용하기 위해서는 서버 측에서도 SSH 서비스가 해당 방식을 지원하도록 설정되어 있어야 한다. 이 설정이 잘못되었거나 지원하지 않는 경우 접속할 수 없다.
5. **passphrase 분실 시 접근 불가** : 개인 키를 passphrase로 보호하는 경우, 이를 잊어버리면 키를 사용할 수 없게 되어 접근이 불가능해진다. passphrase 없이 키를 생성할 수도 있지만, 이는 보안상 좋지 않은 선택이다

3.5 관련 자료

비공개 서버에 접속하기 위해서는 인증절차를 거쳐야 한다. 기존에 비밀번호를 네트워크를 통해 보내는 비밀번호 인증은 네트워크 상에서 ID/비밀번호가 그대로 노출되는 문제가 있고, 접속할 때마다 입력해야 하는 번거로움이 있다.

이와 다르게 SSH 키는 이와 달리 공개키 암호 방식을 사용하여 서버에서 인증 받을 수 있다.

CUBRID | 블로그 - SSH 공개키 인증을 사용하여 암호 없이 편리하게 원격 호스트에 접속하기-

SSH 키는 공개 키 암호화 방식 및 인증 확인 응답 인증을 사용하는 SSH 서버에 대해 자체 식별하는 방식입니다. 비공개 서버에 접속하기 위해서는 인증절차를 거쳐야 하는데요, 기존에 비밀번호를 네트워크를 통해 보내는 비밀번호 인증은 네트워크 상에서 ID/비밀번호가 그대로 노출되는 문제가 있고, 접속할 때마다 입력해야 하는 번거로움이 있습니다. SSH 키는 이와 달리 공개키 암호 방식을 사용하여 서버에서 인증받을 ...

<https://www.cubrid.com/blog/3825015>

4. Bastion-Host

4.1 Bastion-Host

외부와 내부 네트워크 사이에 위치한 중간 호스트로, 외부에서 내부 네트워크로 접속하는 경로를 통제하고 관리하는 데 사용된다. 보통 Bastion Host는 특별히 보안이 강화된 서버로 구성되며, 접근 권한이 엄격히 제어된다.

4.2 주요 사용 사례

- 외부에서 내부 네트워크로의 안전한 접근 경로가 필요할 때 사용한다.
- 보안 강화를 위해 외부에서 내부 서버에 접속하는 경로를 제한 및 모니터링하는 경우에 사용한다.
- VPN 접속, SSH 터널링 등을 통해 내부 서버에 접근하는 보안 요구 사항 충족하기 위해 사용한다.

4.3 장점

1. **보안 강화** : 외부와 내부 네트워크 사이에 추가적인 보안 계층을 제공하여 내부 시스템을 보호한다.
2. **접근 제어** : 접근 권한을 엄격히 제어하여 불필요한 접근을 방지하고 감시할 수 있다.
3. **로그 기록** : 접속 이력을 기록하고 감시하여 보안 사고 대응에 도움을 준다.

4.4 관련 자료

- 실제 사용 사례

GCP, AWS, Azure 대표적인 3종류의 CSP의 공식 문서에도 나와 있는 베스천 호스트 사용 사례

베스천 호스트를 사용하여 비공개 클러스터에 원격으로 액세스 | Kubernetes Engine | Google Cloud

이 튜토리얼에서는 베스천 호스트를 사용하여 인터넷을 통해 Google Kubernetes Engine(GKE)에서 비공개 클러스터에 액세스하는 방법을 보여줍니다.

<https://cloud.google.com/kubernetes-engine/docs/tutorials/private-cluster-bastion?hl=ko>



Azure Bastion - 완전 관리형 RDP/SSH | Microsoft Azure

Azure Bastion은 Azure Portal을 통해 직접 가상 머신에 대한 보안 RDP(원격 데스크톱 프로토콜) 및 SSH(Secure Shell) 프로토콜 액세스를 제공합니다.

<https://azure.microsoft.com/ko-kr/products/azure-bastion>

튜토리얼: Linux Bastion Host를 사용한 프라이빗 네트워크 액세스 구성 - Amazon Managed Workflows for Apache Airflow

Amazon Managed Workflows for Apache Airflow를 사용하여 Apache Airflow 워크플로를 쉽게 구성, 배포 및 관리할 수 있는 환경을 만드는 방법에 대해 알아봅니다.

https://docs.aws.amazon.com/ko_kr/mwaa/latest/userguide/tutorials-private-network-bastion.html



5. Kubernetes

5.1 Kubernetes

컨테이너화된 애플리케이션을 자동화하고 관리하기 위한 오픈소스 플랫폼이다. Google에서 개발되었으며 현재는 Cloud Native Computing Foundation(CNCF)의 호스팅 프로젝트로 관리되고 있다. 쿠버네티스는 대규모의 컨테이너화된 애플리케이션을 효율적으로 배포, 확장 및 관리할 수 있도록 설계되었다.

5.2 주요 사용 사례

- 컨테이너 오케스트레이션: 쿠버네티스는 컨테이너화된 애플리케이션을 자동으로 배포하고 관리하는데 사용됨.
- 여러 호스트에 걸쳐 실행되는 컨테이너를 효율적으로 관리하여 자동으로 복제, 로드 밸런싱, 다중 클라우드 및 하이브리드 클라우드 환경: 쿠버네티스는 다양한 클라우드 환경에서 동작할 수 있어 벤더 락인(vendor lock-in) 문제를 해결할 수 있음.
- 온프레미스와 클라우드 간의 워크로드를 쉽게 마이크로서비스 아키텍처: 쿠버네티스는 각각 독립적으로 배포되고 확장 가능한 마이크로서비스 아키텍처를 구축하는 데 사용됨. 서비스 간의 결합도가 낮아져 유연성과 확장성을 개선할 수 있음.
- CI/CD(Continuous Integration/Continuous Deployment): 쿠버네티스는 CI/CD 파이프라인에서 배포 및 테스트 자동화를 지원하여 애플리케이션 개발과 배포 과정을 자동화하고 효율적으로 관리할 수 있음.

5.3 장점

1. 확장성: 쿠버네티스는 수천 개의 컨테이너화된 애플리케이션을 효율적으로 관리할 수 있는 높은 확장성을 제공합니다.
2. 가용성: 서비스 중단 없이 애플리케이션을 업데이트하고 롤백하는 기능을 제공하여 가용성을 향상시킵니다.
3. 자동화: 쿠버네티스는 많은 작업을 자동화하여 운영 부담을 줄입니다. 예를 들어, 자동 스케일링, 자동 복구 및 자동 로드 밸런싱 등이 있습니다.
4. 유연성: 다양한 클라우드 및 온프레미스 환경에서 사용할 수 있으며 다양한 언어 및 프레임워크를 지원합니다.
5. 커뮤니티 및 생태계: 쿠버네티스는 커뮤니티가 활발하게 참여하고 있어 지속적인 개선과 다양한 플러그인 및 도구를 활용할 수 있는 풍부한 생태계를 갖추고 있습니다. 쿠버네티스는 복잡한 애플리케이션 환경을 효율적으로 관리하고 배포하는 데 필수적인 도구로 자리 잡고 있으며, 다양한 산업 및 기업에서 광범위하게 채택되고 있습니다.

6. LoadBalancing

6.1 LoadBalancing

애플리케이션을 지원하는 리소스 풀 전체에 네트워크 트래픽을 균등하게 배포하는 방법이다. 최신 애플리케이션은 수백만 명의 사용자를 동시에 처리하고 정확한 텍스트, 비디오, 이미지 및 기타 데이터를 빠르고 안정적인 방식으로 각 사용자에게 반환해야 한다. 이렇게 많은 양의 트래픽을 처리하기 위해 대부분의 애플리케이션에는 데이터가 중복되는 리소스 서버가 많이 있다. 로드 밸런서는 사용자와 서버 그룹 사이에 위치하며 보이지 않는 촉진자 역할을 하여 모든 리소스 서버가 동일하게 사용되도록 하는 디바이스이다.

6.2 장점

1. 성능 향상 : 응답 시간을 늘리고 네트워크 지연 시간을 줄여 애플리케이션 성능을 향상 시킨다. 서버 간에 로드를 균등하게 배포하여 애플리케이션 성능 향상 클라이언트 요청을 지리적으로 더 가까운 서버로 리디렉션하여 지연 시간 단축 시킨다. 결과적으로 물리적 및 가상 컴퓨팅 리소스의 신뢰성 및 성능 보장한다.
2. 애플리케이션 가용성 : 서버 장애 또는 유지 관리로 인해 애플리케이션 가동 중지 시간이 늘어 방문자가 애플리케이션을 사용할 수 없게 될 수 있다. 로드 밸런서는 서버 문제를 자동으로 감지하고 클라이언트 트래픽을 사용 가능한 서버로 리디렉션하여 시스템의 내결함성을 높입니다. 로드 밸런싱을 사용하여 다음 작업을 더 쉽게 수행할 수 있다. 애플리케이션 가동 중지 없이 애플리케이션 서버 유지 관리 또는 업그레이드 실행 및 백업 사이트에 자동 재해 복구 제공할 수 있습니다. 상태 확인을 수행하고 가동 중지를 유발할 수 있는 문제를 방지할 수 있습니다.
3. 애플리케이션 확장성 : 로드 밸런서를 사용하여 여러 서버 간에 네트워크 트래픽을 지능적으로 전달할 수 있다. 로드 밸런싱이 다음을 수행하므로 애플리케이션에서 수천 개의 클라이언트 요청을 처리할 수 있다.
4. 한 서버에서 트래픽 병목 현상 방지 : 필요한 경우 다른 서버를 추가하거나 제거할 수 있도록 애플리케이션 트래픽을 예측할 수 있다.
5. 애플리케이션 보안 : 로드 밸런서에는 인터넷 애플리케이션에 또 다른 보안 계층을 추가할 수 있는 보안 기능이 내장되어 있다. 이는 공격자가 서버 장애를 일으키는 수 백만 개의 동시 요청으로 애플리케이션 서버를 가득 채우는 분산 서비스 거부 공격을 처리하는 데 유용한 도구다.

6.3 로드 밸런싱 알고리즘

로드 밸런서가 서로 다른 클라이언트 요청 각각에 가장 적합한 서버를 결정하기 위해 따르는 규칙 세트로, 로드 밸런싱 알고리즘은 크게 2가지 범주로 나뉜다.

6.3.1 정적 로드밸런싱

정적 로드 밸런싱은 서버나 네트워크 리소스 사이에 클라이언트 요청을 분배하는 방법 중 하나다. 이 방식에서는 트래픽의 양이나 서버의 현재 부하 상태와 관계없이 미리 정해진 규칙이나 알고리즘에 따라 트래픽을 분배한다. 정적 로드 밸런싱은 주로 서버의 성능, 용량 등과 같은 고정된 속성을 기반으로 작동한다.

6.3.1-1 라운드 로빈 방식

라운드 로빈 방식은 여러 대상이 있을 때, 각각의 대상에게 동일한 시간 또는 기회를 순서대로 할당하는 방식이다. 이 방법은 주로 컴퓨터 네트워킹이나 운영 체제에서 프로세스 스케줄링, 작업 할당에 사용된다. 라운드 로빈은 공정성을 중시하며, 모든 대상이 동일한 주의를 받도록 보장한다. 예를 들어, 여러 프로세스가 CPU 시간을 요구할 때, 라운드 로빈 스케줄러는 각 프로세스에게 일정 시간(타임 슬라이스라고도 함) 동안 CPU를 사용할 기회를 순서대로 제공한다. 이렇게 하면 한 프로세스가 CPU를 독점하는 것을 방지하고 모든 프로세스가 일정량의 처리 시간을 얻게 된다. 네트워킹에서도 비슷한 원리로, 라운드 로빈 DNS는 여러 서버 중 하나에게 순차적으로 네트워크 요청을 분산시키는 방법으로 사용된다.

6.3.1-2 IP 해시 방식

클라이언트의 IP 주소를 기준으로 해시 값을 계산한다. 그리고 이 해시 값을 사용해서 특정 서버에 요청을 할당한다. 예를 들어, 웹 서버 그룹에 요청이 들어올 때, 클라이언트의 IP 주소를 해싱하여 그 결과에 따라 요청을 처리할 서버를 결정한다. 이 방법은 같은 클라이언트의 요청이 항상 같은 서버로 전송되도록 보장한다. 즉, 사용자마다 고유한 서버가 지정되어 일관된 서비스 경험을 제공할 수 있다. IP 해시 방식은 클라이언트와 서버 간의 세션 유지가 중요한 경우 유용하다.

정적 로드 밸런싱은 서버 간에 비슷한 성능과 용량이 있을 때 잘 작동한다. 그러나 서버의 부하가 크게 변하는 동적인 환경에서는 최적의 효율을 내기 어렵다. 이런 경우, 서버의 현재 부하 상태를 실시간으로 모니터링하고, 이를 기반으로 트래픽을 분배하는 동적 로드 밸런싱 방식이 더 적합할 수 있다.

6.3.2 동적 로드밸런싱

서버와 네트워크 장치들 사이에서 트래픽을 조절하는 것이다. 이는 더 나은 성능과 안정성을 제공한다. 보통 로드 밸런서는 여러 대의 서버 사이에서 트래픽을 분산시키는 역할을 한다. 동적 로드 밸런싱은 이것을 자동화하여 서버의 상태나 트래픽 패턴을 분석하여 효율적으로 트래픽을 관리한다. 이는 더 효율적인 리소스 활용과 사용자들에게 더 나은 서비스를 제공한다.

6.3.2-1 최소 연결 방법

최소 연결 방법은 동적 로드 밸런싱 알고리즘 중 하나이다. 이 방법은 클라이언트 요청을 처리할 때 현재 가장 적은 연결을 가진 서버에게 요청을 전달하는 것이다.

이 방법은 서버의 현재 부하를 고려하여 트래픽을 분산시키므로, 서버의 부하가 골고루 분배되고 각 서버의 성능을 최대한 활용할 수 있다. 이는 사용자에게 더 나은 성능과 안정성을 제공하며, 전체 시스템의 효율성을 높일 수 있다.

6.3.2-2 가중치 기반 최소 연결 방법

가중치 기반 최소 연결 방법은 최소 연결 방법과 유사하지만, 각 서버에 가중치를 부여하여 트래픽을 분산하는 방식이다. 서버의 가중치는 서버의 성능, 용량 또는 특정 요구 사항에 따라 설정된다.

이 방법은 각 서버의 상대적인 성능을 고려하여 트래픽을 분산하기 때문에 효율적인 로드 밸런싱을 제공한다. 가중치가 높은 서버에는 더 많은 트래픽이 전달되고, 가중치가 낮은 서버에는 더 적은 트래픽이 전달된다. 이를 통해 서버의 부하를 골고루 분산시키면서도 각 서버의 성능을 최대한 활용할 수 있다.

6.3.2-3 최소 응답 시간 방법

클라이언트 요청을 처리하는 서버 중에서 응답 시간이 가장 짧은 서버에게 트래픽을 전달하는 방식이다. 이 방법을 사용하면 서버의 응답 시간을 모니터링하고, 가장 빠르게 응답하는 서버에게 요청을 라우팅하여 사용자에게 최상의 경험을 제공할 수 있다.

이는 사용자들이 서비스에 빠르게 접근할 수 있도록 하며, 서버 부하를 균형있게 분산시키는 데 도움이 된다. 따라서 최소 응답 시간 방법은 사용자 경험을 최적화하고 시스템 성능을 향상 시키는 데 유용하다.

6.3.2-4 리소스 기반 방법

서버의 리소스 상태를 기반으로 트래픽을 분산하는 방식이다. 예를 들어, CPU 사용량, 메모리 사용량, 디스크 I/O 등과 같은 서버의 리소스 상태를 모니터링하고, 가장 가용성이 높은 서버에게 트래픽을 라우팅한다.

이 방법을 통해 로드 밸런서는 서버의 현재 상태를 실시간으로 파악하여 효율적으로 트래픽을 관리할 수 있다. 따라서 서버의 부하를 최소화하고 성능을 최적화할 수 있다. 리소스 기반 방법은 서버의 가용성과 성능을 유지하면서 시스템의 안정성을 높이는 데 유용하다.

7. Database Replication

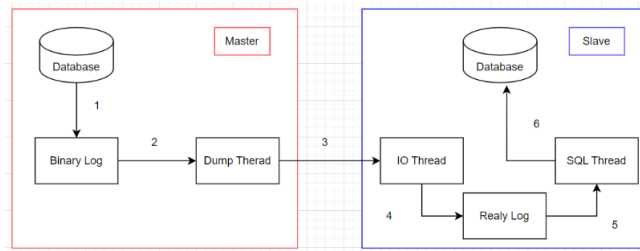
7.1 Replication

데이터 리플리케이션은 실시간 복제본 데이터베이스 서버를 운용하는 것을 의미한다. 기준이 되는 서버를 Primary(Master)라 하고, Primary(master)서버와 동일한 내용을 갖는 다른 서버를 Replica라 한다.

7.2 Replication 목적

Replication은 목적은 여러가지가 있는데, 시스템 중단을 최소화하고 연속적인 서비스를 제공하는고가용성 제공과 마스터 데이터 베이스의 데이터가 손상되거나 손실된 경우 슬레이브로부터 복구할 수 있는 안정성 및 복원력 향상, 그리고 슬레이브 데이터베이스를 백업서버로 데이터를 안전하게 보관할 수 있다.

7.3 Replication 동작원리



그림과 같이, 마스터에서 데이터 변경이 일어나면 마스터에 데이터베이스에 반영이 된다. 그 후 마스터에서 변경된 이력을 Binary Log에 기록 후 관련 이벤트를 슬레이브에 전달한다.

슬레이브는 IO_THREAD에서 마스터 이벤트를 감지하고 마스터의 덤프 스레드에게 데이터를 요청하고, 마스터의 덤프 스레드는 바이너리 로그의 기록을 읽어 IO 스레드에게 전달해준다. 그 후 IO 스레드는 전달받은 데이터를 릴레이 로그에 저장 하고, SQL 스레드는 릴레이 로그를 읽고 슬레이브의 데이터베이스에 반영이 된다.

7.4 Primary DB의 역할

Primary DB 노드는 데이터베이스 복제에서 중심이 되는 역할을 맡는다. 이 노드는 쓰기 작업을 처리하고, 이러한 변경 사항을 다른 복제된 노드에 전파한다. 즉, 모든 쓰기 작업은 주로 Primary DB 노드에서 이루어지며, 다른 복제된 노드들은 이러한 변경 사항을 복제하여 동기화한다.

Primary DB 노드는 데이터베이스의 실시간 업데이트 및 변경을 관리하고, 데이터의 일관성을 유지한다. 따라서 응용 프로그램이나 사용자들이 데이터를 쓰거나 변경할 때, 이러한 변경 사항은 주로 Primary DB 노드에서 처리된다.

일반적으로 Primary DB 노드는 가장 최신의 데이터를 가지고 있으며, 다른 복제된 노드들은 이를 기반으로 자신의 데이터를 동기화한다. 이러한 방식으로 Primary DB 노드는 데이터의 중심적인 역할을 수행하며, 데이터의 일관성과 신뢰성을 유지한다.

7.5 Secondary DB의 역할

Secondary DB 노드는 Master DB 노드로부터 데이터를 복제받는 노드이다. 이 노드는 주로 읽기 작업을 처리한다. Master DB 노드의 데이터 변경 사항을 복제하여 동기화하므로, 일반적으로 Secondary DB 노드는 Master DB 노드와 동일한 데이터를 가지고 있다.

Secondary DB 노드는 주로 읽기 요청을 처리하는 데 사용된다. 쓰기 작업은 Master DB 노드에서 처리되므로, Secondary DB 노드는 읽기 작업에 대한 별도의 부하를 처리할 수 있다. 이는 전체 시스템의 성능을 향상시키고, 사용자에게 빠른 응답 속도를 제공하는 데 도움이 된다.

또한, Secondary DB 노드는 Master DB 노드의 부하가 증가하거나 장애가 발생할 경우 대비하여 장애 복구 및 고가용성을 제공한다. 즉, Secondary DB 노드는 데이터베이스 시스템의 안정성과 가용성을 높이는 데 기여한다.

요약하면, Secondary DB 노드는 Master DB 노드의 데이터를 복제하여 읽기 작업을 처리하고, 장애 복구 및 고가용성을 제공한다. 이를 통해 전체 데이터베이스 시스템의 성능과 안정성을 향상 시킨다.

7.6 주요 사용 사례

- **읽기 부하 분산:** 데이터베이스 Replica를 사용하면 읽기 작업을 Primary DB 노드와 Replica 노드로 분산시킬 수 있다. 이를 통해 Primary DB 노드의 부하를 줄이고 전체 시스템의 응답 시간을 향상시킬 수 있다.
- **고가용성 및 장애 복구:** Replica 노드는 Primary DB 노드의 복제본이므로, Primary DB 노드에 장애가 발생했을 때 데이터의 손실을 방지하고 시스템의 가용성을 유지할 수 있다. 장애 복구 시에는 Replica 노드 중 하나가 Primary로 승격되어 서비스를 계속할 수 있다.
- **백업:** Replica 노드는 Primary DB 노드의 데이터를 복제하므로, 일종의 실시간 백업으로 활용할 수 있다. 필요한 경우 복제본을 사용하여 데이터를 복원할 수 있다.
- **지리적 분산:** 데이터베이스 Replica를 여러 지역에 배치함으로써 지리적으로 분산된 사용자에게 빠른 응답 시간을 제공할 수 있다. 각 지역에 Replica를 배치하여 사용자에게 가까운 서버에서 데이터를 읽을 수 있도록 할 수 있다.
- **테스트 및 개발:** Replica 노드는 Primary DB 노드의 복제본이므로, 테스트 및 개발 환경에서 안전하게 사용할 수 있다. 실제 데이터를 기반한 테스트 및 개발 작업을 수행할 수 있다.

8. NFS(Network File System)

8.1 NFS

NFS(Network File System)는 Sun Microsystems에서 개발된 분산 파일 시스템이다. 클라이언트-서버 아키텍처를 기반으로 하며, 파일 및 디렉터리를 공유하여 여러 시스템 간에 데이터를 공유할 수 있다. NFS는 네트워크를 통해 파일 시스템을 마치 로컬 파일 시스템처럼 액세스할 수 있도록 해준다.

네트워크에 파일을 저장하는 메커니즘으로 사용자가 원격 컴퓨터에 있는 파일 및 디렉토리에 액세스할 수 있고 해당 파일 및 디렉토리가 로컬에 있는 것처럼 처리하도록 허용하는 분산 파일 시스템이다. 예를 들어, 사용자는 운영 체제 명령을 사용하여 원격 파일 및 디렉토리에 대한 파일 속성을 작성, 제거, 읽기, 쓰기, 설정할 수 있다.

8.2 작동방식

NFS는 1984년 Sun Microsystems에서 처음 개발했지만, 현재는 국제 인터넷 표준화 기구에서 유지 관리하고 있습니다. NFS는 Linux 운영 체제와 macOS를 포함한 Unix 시스템용으로 설계되었습니다. NFS에서 클라이언트는 원격 프로시저 호출(RPC)을 사용하여 원격 NFS 서버에 파일 또는 디렉토리를 요청합니다. 파일 또는 디렉토리를 사용할 수 있고 클라이언트에 올바른 액세스 권한이 있는 경우 서버는 해당 파일 또는 디렉토리를 클라이언트에 마운트합니다. 클라이언트는 가상 연결을 통해 파일 작업을 수행합니다. NFS 버전 4부터, 파일 충돌 해결을 위한 NFS 잠금 관리자는 더 이상 별도의 서비스가 아니며 프로토콜의 일부입니다.

8.3 장점

1. **간편한 파일 공유:** NFS를 사용하면 여러 컴퓨터 간에 파일 및 디렉토리를 쉽게 공유할 수 있다.
2. **네트워크 효율성:** 파일을 로컬 파일 시스템처럼 읽고 쓸 수 있으므로 네트워크 트래픽을 줄일 수 있다.
3. **중앙 집중식 관리:** 파일 시스템을 중앙 집중식으로 관리하여 관리가 용이하다.
4. **성능:** 최신 NFS 버전은 성능을 향상시키는 여러 기능을 제공한다.

8.4 단점

1. **보안:** 기본적으로 NFS는 암호화되지 않으므로 데이터 보안에 취약할 수 있다.
2. **인증:** NFS는 일반적으로 클라이언트 시스템의 IP 주소 또는 호스트 이름을 기반으로 공유 파일에 대한 액세스를 제어하는 호스트 기반 인증을 사용한다. 내장된 보안 메커니즘이 제한적이므로, 개방형 네트워크에서는 문제가 될 수 있다.
3. **신뢰성:** 네트워크 연결에 따라 NFS 공유가 불안정할 수 있다.
4. **권한 관리:** 파일 및 디렉토리에 대한 권한 관리가 복잡할 수 있다.
5. **파일 잠금:** NFS는 상태 비저장 설계를 사용하므로 서버에서 열린 파일을 추적하지 않는다. 따라서 파일 잠금은 클라이언트에서 처리되며 여러 클라이언트가 동일한 파일에 동시에 데이터를 쓰려고 할 때 충돌이 발생한다.

8.5 주요 사용 사례

- **서버 백업:** 클라이언트에서 NFS를 사용하여 서버의 데이터를 백업하는 데 활용할 수 있다.
- **소규모 파일 서버:** NFS를 사용하여 소규모 파일 서버를 구축하여 여러 사용자가 파일을 공유하는 데 활용할 수 있다.
- **분산 작업:** 여러 컴퓨터에서 동시에 작업하는 경우 NFS를 사용하여 파일을 공유하고 데이터를 동기화하는 데 활용할 수 있다.
- **간단한 파일 공유:** 여러 사용자가 파일을 공유하고 수정하는 데 활용된다.
- **클라우드 환경:** 클라우드 서비스에서 데이터를 공유하고 백업하는 데 활용된다.
- **서버 클러스터:** 서버 클러스터 간에 데이터를 공유하고 동기화하는 데 활용된다.