3Tier_아키텍처_분석

- 실습가이드

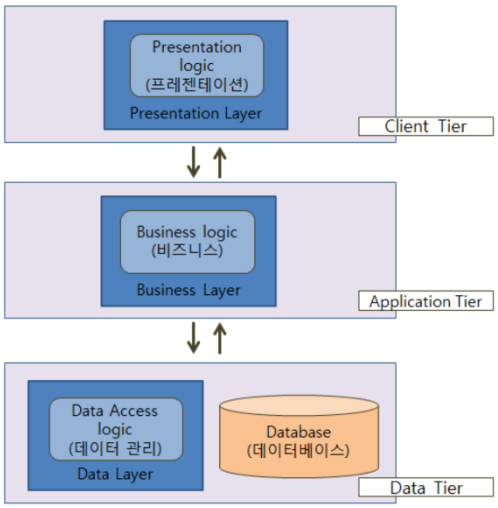
최초 작성일 : 2024/03/05

최종 제출일 : 2024/03/06

김민경

Ⅰ. 3 tier

- 어떠한 플랫폼을 3계층으로 나누어 별도의 논리적/물리적인 장치에 구축 및 운영하는 형태



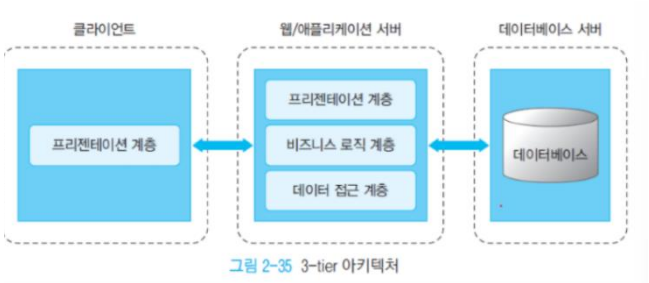


그림 2-35 3-tier 아키텍처

- 웹 서버 운영을 예로 들면 서버 한대에 한꺼번에 모든 기능들을 구축하는 것이 아니라 데이터를 저장하고 읽는 데이터 계층, 데이터를 처리하는 어플리케이션 계층, 그리고 데이터를 표현해주는 클라이언트(프레젠테이션) 계층과 같이 각각 3계층으로 나누어 각각의 기능으로 별도의 논리적/물리적인 장치에서 운영하는 방식

1. 구분

1) 프레젠테이션 계층

- 사용자가 직접 마주하게 되는 계층
- 주로 사용자 인터페이스(인터넷 브라우저 등)를 지원하며 이 계층은 GUI 또는 프론트엔드(front-end) 라고도 부름
- so 이 계층에서는 사용자 인터페이스와 관계없는 데이터를 처리하는 로직은 포함하지x
- 주로 웹 서버를 예시로 둠
- HTML, Javascript, CSS 등이 이 계층에 해당됨

2) 어플리케이션 계층

- 프레젠테이션 계층에서 요청되는 정보를 어떠한 규칙을 바탕으로 처리하고 가공하는 것들을 담당

- 동적인 데이터를 제공함
- 프레젠테이션 계층에서 바라 볼 때에는 서버처럼 동작하고(응답), 데이터 계층에서 바라 볼 때는 클라이언트처럼 행동함(요청)
- 미들웨어(Middleware) 또는 백엔드(back-end)라고도 불림
- 프레젠테이션코드 (예를 들면 HTML, CSS)나 데이터 관리를 위한 코드는 포함하지 x
- 주로 PHP, Java 등이 이 계층에 해당함

3) 데이터 계층

- 데이터베이스와 데이터베이스에 접근하여 데이터를 읽거나 쓰는 것을 관리하는 것을 포함함
- 주로 DBMS (Database Management System)가 이 계층에 해당됨
- 백엔드(back-end)라고도 부름
- 주로 MySQL, MongoDB 등이 이 계층에 해당됨

2. 3tier 필요한 이유

- **분리와 추상화**
  - 각 계층이 독립적으로 역할을 수행함으로써, 변경이나 유지보수가 한 계층에 국한되고, 다른 계층에 미치는 영향을 최소화함
  - ex) 데이터베이스 스키마를 변경해도 프레젠테이션 레이어는 영향을 받지 x
- **보안 강화**
  - 데이터와 비즈니스 로직이 사용자 인터페이스와 분리되어 있기 때문에, 시스템을 보다 안전하게 만들 수 있음
  - ex) 직접적인 데이터베이스 접근을 차단하고, 비즈니스 계층을 통해서만 데이터 접근을 허용하여 보안을 강화할 수 있음
- **확장성**
  - 트래픽이나 데이터가 증가함에 따라, 각 계층을 독립적으로 확장할 수 있음
  - ex) 사용자 수가 증가할 경우 프레젠테이션 계층에 서버를 추가하거나, 데이터베이스 성능을 향상시키기 위해 데이터 계층을 확장할 수 있음
- **유지보수성**

- 잘 정의된 인터페이스와 계층 간의 분리로 인해, 시스템의 각 부분을 더 쉽게 관리하고 업데이트할 수 있음

- 한 계층의 변경이 다른 계층에 미치는 영향이 제한되므로 유지보수 작업이 단순화됨

- **재사용성**

- 비즈니스 로직이 중앙집중화되어 있어, 다른 프로젝트나 애플리케이션에서도 같은 비즈니스 계층을 재사용할 수 있음

- 코드 중복을 줄이고, 개발 시간을 단축하는 데 도움이 됨

- **테스트와 디버깅의 용이성**

- 각 계층을 독립적으로 테스트할 수 있으므로, 문제 발생 시 디버깅이 쉬워짐

- ex) 데이터베이스 쿼리 문제가 있다면 데이터 계층만을 대상으로 테스트하여 빠르게 해결할 수 있음

- **시장 대응 속도**

- 3-Tier 아키텍처는 다양한 전문가(프론트엔드 개발자, 백엔드 개발자, 데이터베이스 관리자 등)가 동시에 다른 부분을 개발할 수 있게 해줌으로써, 시장 변화나 사용자 요구에 빠르게 대응할 수 있도록 도와줌

Ⅱ. N Tier

1. 1 Tier

- 하나의 물리적인 컴퓨터 또는 서버에 3가지의 다른 기능으로 함께 구현한 방식
- so 물리적인 장비를 새로운 장비로 변경하고자 하는 경우에는 모든 구성을 함께 변경해야 함

2. 2 Tier

- 클라이언트 계층과 데이터 계층의 물리적인 컴퓨터 또는 서버로 구분하여 클라이언트 계층에서의 변경이나 데이터베이스의 변경 시 서로 영향을 받지x

3. 3 Tier

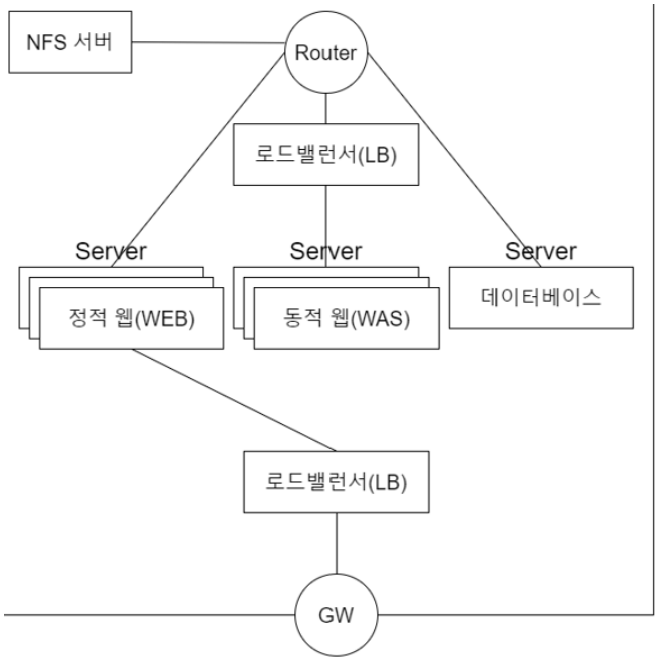
- 클라이언트 계층, 어플리케이션 계층, 데이터 계층으로 서버를 모두 물리적으로 나누어 구성하

는 방식

- 이에 각각의 계층에서 변화가 일어나더라도 서로 영향을 받지 않고 독립적으로 운영됨

Ⅲ. 3 Tier 분석

1. 온프레미스 3 Tier



- 각 tier를 **다른 네트워크에서 구현하더라도, 이들은 연결되어 하나의 서비스로 동작해야하기 때문에 라우터를 통해 연결**되어야 함
- 외부에서 접근할 때는 로드밸런싱 되어 web으로 접속됨
- 사용자가 web에서 데이터를 수정하거나 데이터에 접근할 때

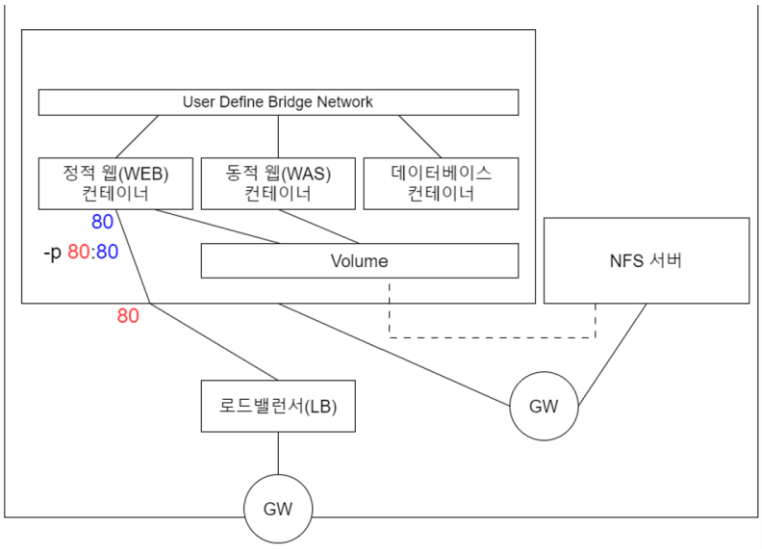
1-1. 장점

- **커스터마이징 :** 시스템의 모든 측면을 사용자의 필요에 따라 맞춤 설정할 수 있음
- **성능 :** 자체 하드웨어에서 실행되므로 네트워크 지연이 최소화되며, 성능을 최적화할 수 있음
- **보안 :** 상대적으로 데이터와 애플리케이션의 보안을 내부적으로 완벽히 제어할 수 있음
- **통합 :** 기존 it 인프라와 통합이 용이함

1-2. 단점

- **비용 :** 높은 초기 투자비용이 요구되며, 유지보수 및 업그레이드에 지속적인 비용이 발생함
- **유연성 부족:** 빠른 스케일링이 어렵고 늘어나는 요구사항에 신속하게 대응하기 어려울 수 있음
- **자원 활용:** 자원이 고정되어 있어 불필요한 유휴상태(컴퓨터 시스템이 사용 가능한 상태이나 실제적인 작업이 x는 시간)가 발생할 수 있음

2. 도커 3 Tier



- **User Define Bridge Network :** 현재 사용자 정의 브리지 네트워크를 통해 컨테이너 간 통신을 관리함

2-1. 장점

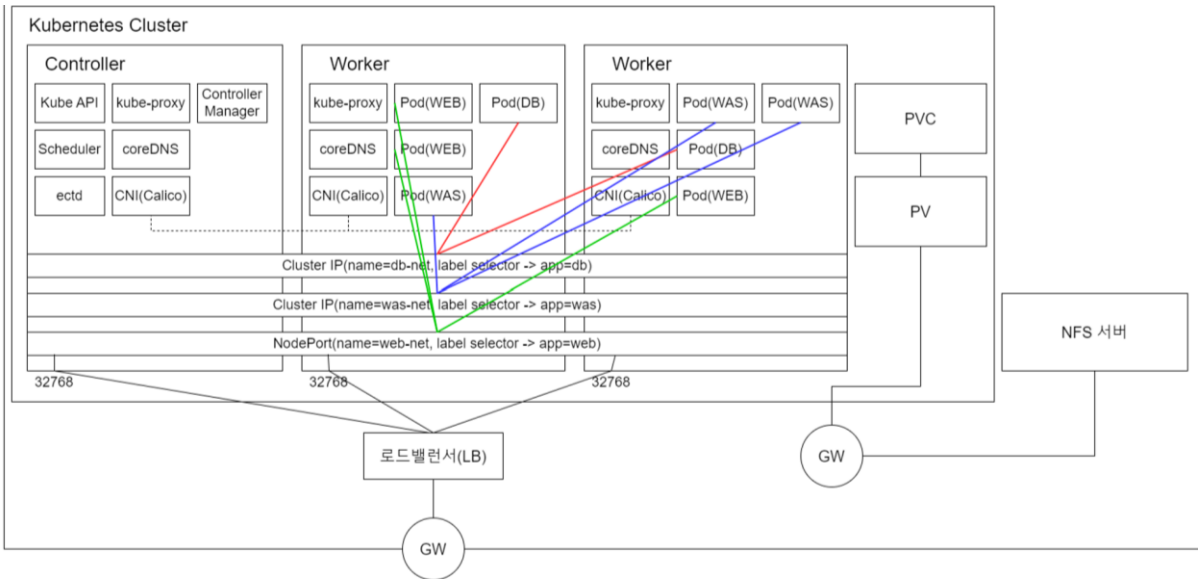
- **배포 속도:** 컨테이너 기술을 사용하여 신속하게 배포할 수 있음
- **환경 일관성:** 개발부터 운영까지 동일한 환경에서 애플리케이션을 실행할 수 있음
- **리소스 활용:** 가상화된 환경에서 리소스를 효율적으로 사용할 수 있음
- **모듈성:** 컨테이너를 통해 각 서비스를 독립적으로 관리하고 업데이트할 수 있음
- **비용 효율성:** 물리적 인프라에 비해 유지보수 비용을 절감할 수 있음

2-2. 단점

- **보안 취약성:** 컨테이너는 가상 머신에 비해 보안이 덜 견고할 수 있음

- **영속성 관리:** 상태를 유지하는 애플리케이션에 대한 데이터 관리가 더 복잡할 수 있음
- **네트워킹 복잡성:** 컨테이너 간의 네트워크 구성과 통신이 복잡할 수 있음
- **리소스 한계:** 컨테이너는 호스트 시스템의 자원에 의존하므로, 리소스가 제한적일 수 있음

3. 쿠버네티스 3 Tier



- 각 노드의 Web Server 역할 pod들은 NodePort와 연결되어있고, WAS Server pod와 DB Server pod는 ClusterIP와 연결되어있음
- 이러한 구성을 통해, **외부에서는 WebServer에만 접근 가능**

WAS 및 DB 서버는 클러스터 내부에서만 접근 가능하도록 하여 서비스에 따라 접근 제어가 가능함

- but, pod 간의 통신이 노드를 경유하기 때문에 네트워크 오버헤드에 주의해야하함, nodeport는 외부에서 직접 접근이 가능하므로 보안 측면에서 주의가 필요함

3-1. 장점

- 자동화된 운영: 쿠버네티스는 자동 스케일링, 자가 치유, 롤아웃 및 롤백 등의 기능을 제공하여 운영을 자동함
- **고가용성:** 서비스 중단 없이 애플리케이션을 업데이트하고 유지보수할 수 있음
- **확장성:** 클러스터의 리소스에 맞게 파드를 자동으로 확장/축소할 수 있음
- **로드 밸런싱과 서비스 발견:** 내부적으로 로드 밸런서를 사용하여 트래픽을 분산하고, 서비스를

쉽게 발견하고 연결할 수 있음

• **영속적 스토리지 관리:** Volume을 통해 상태를 유지하는 애플리케이션의 데이터 관리를 쉽게 할 수 있음

3-2. 단점

• **학습 곡선:** 쿠버네티스는 배우고 사용하기 복잡하며, 초기 설정이 매우 까다로울 수 있음

• **자원 소모:** 쿠버네티스 자체가 자원을 소모하며, 작은 규모의 프로젝트에는 과도할 수 있음

• **관리 복잡성:** 클러스터의 상태를 유지하고 모니터링하는 것이 복잡하며, 전문 지식을 요구

출처

서희님 실습가이드