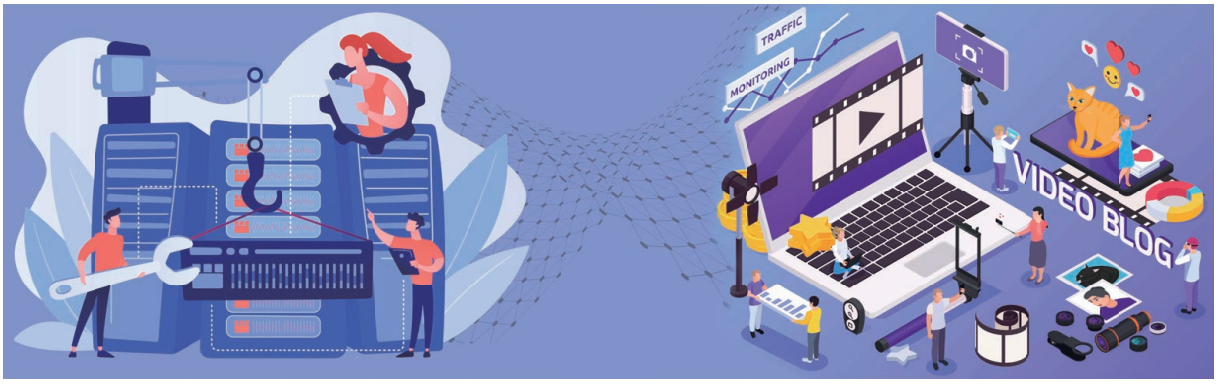
테크놀로지 리더를 위한
Media & IT(Information Technology)

#7. 시스템 아키텍처와 미디어

강자원
컴퓨터시스템응용기술사
/ KBS MNC(Media Network Center)팀



연재 목차

만약 누군가가 “시스템 아키텍처가 도대체 뭐에요?”라고 물어본다면 어떻게 대답해야 할까? 사실 시스템 아키텍처라고 하면 어떤 것인지 정확한 개념을 떠올리기 어려울 수 있다. 하지만 멀티미디어의 다양하고 복잡한 소프트웨어 시스템을 설계하고 관리하기 위해서 효과적인 시스템 아키텍처는 필수다. 시스템이 해야 할 일을 결정하고, 필요한 주요 하부 시스템과 이들의 연결 방식을 파악하고, 계속 분해해 나가면서 상세 내용을 알게 되면 이를 이용해 개발팀이 각 하부 시스템을 확장하고 이들을 통합해 원하는 시스템을 만들어 낸다. 그러나 이러한 패턴이 최근 몇 년간 변화하고 있고 그 속도도 빨라지고 있다. 우선, 막연하게 알고 있는 시스템 아키텍처란 무엇인지부터 알아보자.

시스템 아키텍처란 뭘까?

모든 IT 시스템을 운영 및 기획, 구축하는 부서라면 시스템 아키텍처를 보유하고 있다. 일부 조직은 아키텍처가 좀 더 신중히 구체화 되어 있다. 그러나 아키텍처가 우연의 산물인 경우가 너무나 많다. 전체적인 계획 없이 시간이 지나면서 축적된 형태인 것이다. 아키텍처란 목표하는 대상에 대하여 그 구성과 동작 원리, 구성요소 간의 관계 및 시스템 외부 환경과의 관계 등을 설명하는 설계도 혹은 청사진을 말한다. 예를 들어, 집을 짓는다고 할 때 어떤 모양으로 짓고, 몇 층으로 할지, 방이 어디에 위치할지 등 전체적인 설계도를 만드는 것이다. 시스템 아키텍처란, 컴퓨터 프로그램이나 시스템의 구조와 구성요소들이 어떻게 조직되어 동작하며 상호작용하는지에 대한 계획이나 설계다. 시스템 아키텍처는 마치 건물의 설계도와 비슷한 역할을 한다. 애플리케이션을 개발할 때, 이 애플리케이션이 어떤 모듈들로 구성되고, 각 모듈이 어떤 기능을 수행하며, 어떤 방식으로 데이터가 흐르는지를 결정하는 것이 시스템 아키텍처다. 이렇게 시스템 아키텍처는 시스템의 구조와 동작을 계획하고 설계함으로써 복잡한 시스템을 더욱 효율적으로 관리하고 개발할 수 있게 도와준다. 조금 더 쉽게 설명해 보자.

포스트잇은 소프트웨어의 핵심 구성요소다. 만약 포스트잇과 연결된 가위를 칼로 교체해야 한다면 어떻게 해야 할까? 끈으로 연결되어 있으므로 펜, 잉크병, 테이프 등과 연결된 끈을 풀고 다시 가위를 칼로 교체한 후 다시 끈으로 묶어야 한다. 그런데, 이렇게 가위를 칼로 바꿨더니 펜과 테이프가 "나는 칼이 아니라 가위가 필요해"라고 말한다. 헉! 다시 펜과 테이프를 칼과 연결할 수 있도록 변경해야 하고 이러한 변경은 연결된 개체에 연쇄적으로 영향을 주게 된다.

자, 다음 그림을 보자. 소프트웨어의 아키텍처를 적용해 정리해 보니, 화면들이 잘 정리되었다. 이제 가위를 교체하려면 가위의 끈을 풀고 칼로 연결하면 된다. 참 쉬워졌다. 아키텍처의 첫 번째 목적은 소프트웨어를 쉽게 변경할 수 있는 구조로 설계해 유지보수를 쉽게 하는 것이다. 하지만, 아직도 문제는 있어 보인다. 지금의 구조에서 포스트잇은 끈으로 묶여있지 않지만, 핵심 구성요소가 끈으로 묶여 다른 구성요소들과 연결되어 있을 경우 핵심구성요소의 변경이 쉽지 않을 수 있다. 이런 경우를 의존성이라 말한다. 아키텍처는 의존성을 약하게 가질 수 있는 구조로 설계해야 운영이 수월하다.

일반적인 시스템 아키텍처는 어떻게 구성되어 있을까?

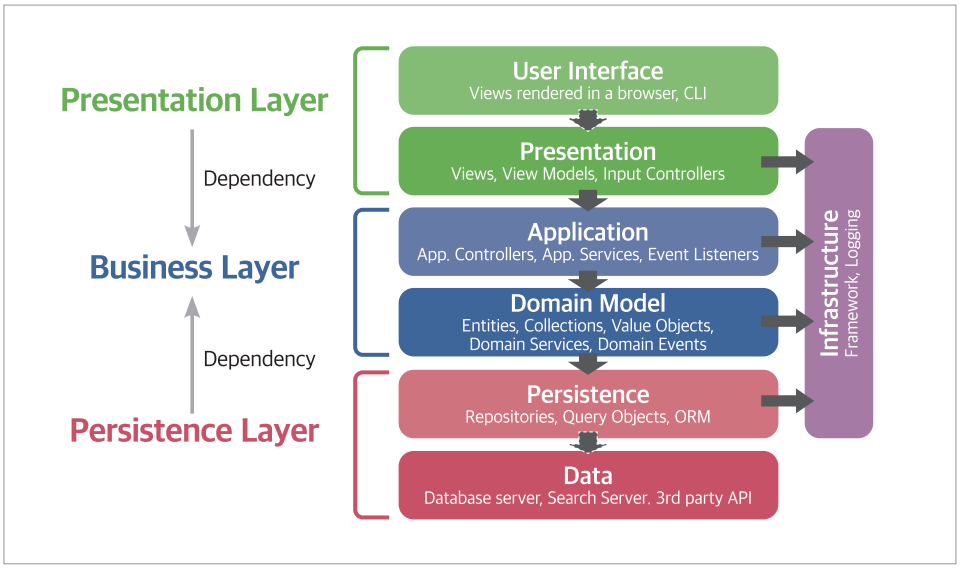


그림 1. 계층형 시스템 아키텍처

시스템을 설계할 때 어떤 아키텍처 스타일을 적용할 것인가에 따라 아키텍처의 구성요소는 달라질 수 있다. 전통적인 시스템 아키텍처의 구성을 설명한다면 다음과 같은 구조로 스트리밍 미디어 콘텐츠를 제공하는 시스템, 예를 들어 유튜브를 예시로 이야기해 보겠다.

Presentation Layer (User Interface)

Presentation layer는 유튜브에 접속하면 보이는 화면이다. 컴퓨터에서 보이는 화면이나 스마트TV에서 접속해 클라이언트 시스템과 직접적으로 연결되는 부분이다. 웹사이트에서는 UI 부분에 해당하고 백엔드 API에서는 엔드포인트 부분에 해당한다. 그러므로 Presentation layer에서 API의 엔드포인트들을 정의하고 전송된 HTTP 요청(request)을 읽어 들이는 로직을 구현한다. 하지만 그 이상의 역할은 담당하지 않는다. 실제 시스템이 구현하는 비즈니스 로직은 다음 레이어로 넘기게 된다.

* Presentation Layer 역할 : EndPoint, Authentication, JSON Translation

Business Layer (Business Logic)

Business Layer는 이름 그대로 비즈니스 로직을 구현하는 부분이다. 도메인 레이어라고도 하는데, 실제 시스템이 구현해야 하는 로직을 이 레이어에서 구현하게 된다. 시스템에서 서비스를 제공하는 가장 핵심적인 부분이라 할 수 있다. Presentation Layer의 UI는 Business Layer의 업무 로직에 업무처리를 요청하고 다시 Business Layer는 Persistence Layer를 통해 데이터를 조회 후 그 결과를 UI로 전달해 사용자에게 표시하는 방식이다. 쉽게 설명하면, 유튜브에서 원하는 영상을 선택하면 선택된 영상을 보여주기 위해 처리되는 중간 과정을 담당하는 부분이 된다. 그러기 위해서는 사용자 인증, 향후 추천 알고리즘을 사용하기 위한 데이터의 전 처리 등에 해당하는 부분이다.

* Business Layer 역할 : Business Logic, Validation, Authorisation, Event Listeners

Persistence Layer (Data Access)

Persistence Layer는 데이터베이스와 관련된 로직을 구현하는 부분이다. Business Layer에서 필요한 데이터를 생성, 수정, 읽기 등을 처리하여 실제로 데이터베이스에서 데이터를 저장, 수정, 읽어 들이기를 하는 역할을 한다. 데이터베이스뿐 아니라 데이터를 처리하고 저장하는 인프라 부분의 로직처리까지도 해당한다. Business Layer에서 요청한 영상 콘텐츠의 위치를 찾고 실제 스토리지에 저장된 콘텐츠를 스트리밍 서버로 전달하는 역할 등을 한다.

* Persistence Layer 역할 : Storage Logic, Database, 3rd Party API

시스템 아키텍처는 왜 중요할까?

아키텍처는 다양한 영역과 관련된 의사결정의 결과물이며, 이후 이어질 활동에 대한 기준이 된다

한번 아키텍처가 만들어진 후에 이후에 이어질 활동과 단절되는 것이 아니라, 활동에 대한 기준이 된다. 그 형태는 상세 설계에 대한 원칙이 될 수도 있고, 운영 준비에 대한 약속이 될 수도 있다. 다음 릴리즈를 준비하거나 기능을 고도화할 때, 아키텍처의 견고함은 경제의 개념과 연관될 수 있다. 여기서 견고함이란 변화를 이겨내는 힘이 아니라, 스스로 변화할 힘을 의미한다. 외부 요인에 의해서 비자발적으로 변화를 '당하는 것'이 아니라, 외부 요인을 파악하고, 요인에 대한 영향도를 분석하여 자발적으로 변화할 수 있는 '유연성'이 바로 견고함의 의미이다. 견고함과 유연함을 반대의 의미라고 생각할 수 있으나, 견고함의 극한이 유연함의 극한과 통한다.

단순하게 구조화하고, 설계하라. 그래야 운영비용을 감소시킬 수 있다

시스템이 안정적으로 운영되고, 궁극적으론 운영비를 절감하면서 오랫동안 잘 사용하기 위해서는 복잡성의 문제를 해결해야만 한다. 복잡성을 절감하는 것은 시스템의 생존과 연결된다. 아키텍처는 복잡성을 이겨낼 수 있는 유연한 구조로 전환되어야 하며, 그 구조는 아키텍처로, 아키텍처를 만들어 내는 과정인 설계를 잘해야 한다는 것이다. 기술의 발전으로든 사용자의 증가든 어떤 원인으로 인해 복잡성이 증가하게 되고 이로 인해 아키텍처를 변경해야 하는 일은 생길 것이다. 이때, 어김없이 경제의 개념과 상충하게 되는데 언제나 경제 쪽이 늘 이긴다. 그래서, 아키텍처는 복잡성을 절감하는 방향으로 설계되어야 한다.

시스템 아키텍처, 어떻게 단순하게 잘 설계할 수 있을까?

여기서, 마이크로서비스 아키텍처(MSA, Micro Service Architecture)의 개념을 설명해 볼 수 있겠다. 최근 아마존, 넷플릭스, 우버부터 우아한 형제들과 쿠팡까지 도입한 사실이 알려지면서 화제가 됐다. 보수적이라는 국내의 금융업계도 MSA 전환 사실이나 적용 계획을 올해 초 밝혔다. 가장 최근엔 방송 미디어 관련 업계에서는 LGU+에서 업계 최초로 IPTV 플랫폼에 'MSA' 기술을 적용했다고 밝혔다. 미디어 플랫폼 운영 안정성을 높이고 IPTV 품질을 개선하여 고객 불편을 최소화하려는 시도다.

그림 2. 연합뉴스 2023.06.01.자 기사

그렇다면, 도대체 MSA가 무엇이길래? 업계 최초이니 이런 말들을 하는 것일까? MSA란, 서로 영향을 주지 않는 독립적으로 상용 환경에 배포 가능한 각각의 기능을 수행하는 작은 서비스 구조를 말한다. 대규모 소프트웨어 프로젝트를 마이크로 단위의 모듈로 분리하여 loosely-coupled 한 구조로 만들고 API를 통해 서로 통신하는 구조다. 쉽게 말해서, 하나의 큰 애플리케이션을 여러 개의 작은 애플리케이션으로 쪼개어 변경과 조합이 가능하도록 만든 아키텍처다.

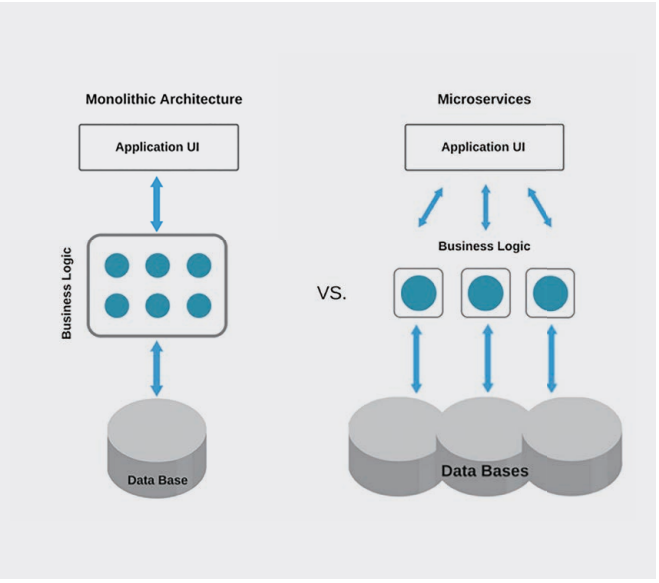


그림 3. 모놀로식 아키텍처와 마이크로서비스 아키텍처의 비교

그러니까, 무슨 소리냐하면... 대부분 우리가 방송제작 환경에서 사용하는 시스템들은 소프트웨어의 모든 구성요소가 한 프로젝트에 통합된 형태이다. 웹 형태로 개발된 미디어 콘텐츠 관리 시스템을 예로 들면, 이런 웹 형태의 프로그램을 개발하기 위해 각각 모듈별로 개발하고, 개발이 완료된 웹 애플리케이션을 하나의 결과물로 패키징하여 배포되는 형태를 말한다. 이런 애플리케이션을 모놀리식 애플리케이션이라 하며, 웹의 경우 자바 애플리케이션을 포장하는데 사용되는 포맷인 WAR(Web application ARchive) 파일로 빌드되어 WAS(Web Application Server)에 배포하는 형태를 말한다.

그러나, MSA는 API를 통해서만 상호작용할 수 있다. 즉, 마이크로서비스는 서비스의 end-point(접근점)를 API 형태로 외부에 노출하고, 실질적인 세부 사항은 모두 추상화한다. 내부의 구현 로직, 아키텍처와 프로그래밍 언어, 데이터베이스, 품질 유지 체계와 같은 기술적인 사항들은 서비스 API에 의해 철저하게 가려진다. 각각의 서비스는 모듈화가 되어 있으며 이러한 모듈끼리는 RPC 또는 message-driven API 등을 이용하여 통신한다. 이러한 MSA는 각각 개별의 서비스 개발을 빠르게 하며, 유지보수도 쉽게 할 수 있도록 한다. 대부분 MSA를 적용하려는 이유는 아래와 같다.

- 하나의 서비스 장애가 다른 서비스로 전파되지 않는다.
- 코드 복잡성 및 의존성을 제거하여 영향도 파악을 용이하게 한다.
- 서비스의 고가용성과 확장성을 확보한다.

그러면, 미디어 업계에서는 MSA를 어떻게 적용해야 할까? 꼭 필요할까?

기존에 우리가 지금도 사용하는 '백엔드(Backend)' 개발 방식은 간단한 서비스로 시작하며 간편 개발이나 배포에는 용이하나, 점점 복잡한 기능이 추가될수록 연관된 모듈 모두 수정을 해야 한다는 어려움이 있다. 하지만 MSA로 전환하면 원래 하나의 모듈로 작동하던 구조를 최대한 독립적인 작은 모듈(마이크로서비스)로 분리하여, 관련된 일부분만 설계, 개발, 검증이 가능하다. 신기능에 한정해 베타 테스트를 제공하거나, 고객 피드백을 더 빠르게 반영할 수 있는 애자일(Agile) 프로세스가 가능하게 되는 것이다. 또 예상치 못한 트래픽 증가 시에도 마이크로서비스 증설과 DB 서버 증설을 5분 내로 하도록 하는 가상화 기술을 적용해 안정적인 서비스 품질을 유지할 수 있다.

MSA와 같은 모듈형 아키텍처 스타일은 클라우드 기반 환경에 적합해 높은 인기를 구가하고 있다. 특히 도커(Docker), 쿠버네티스(Kubernetes) 등과 같은 컨테이너 기반의 플랫폼과 조합이 잘 어우러지면서 클라우드 플랫폼과 MSA는 서로 끌어주고 밀어주면서 발전하고 있다. MSA는 아마존, 넷플릭스, 이베이와 같은 글로벌 서비스 기업들이 사용할 만큼 강력한 아키텍처이다. 하지만 막상 MSA를 적용하려고 하면 다음과 같은 난관에 부딪히게 될 것이다.

개발 복잡도와 숙련도

분산 시스템 개발은 일반 개발보다 복잡하다. 독립적인 서비스이기 때문에 각 모듈의 인터페이스를 신중하게 처리해야 한다. 요청에 응답하지 않게 될 경우의 방어 코드도 작성해야 하며 호출 대기 시간이 일정 수준을 넘기면 복잡한 상황이 발생할 수 있다. 또한 동기적인 처리방식인 REST 통신으로 인한 제약이 발생할 수 있다.

트랜잭션 관리

MSA는 Database Per Service라는 새로운 요구사항으로 분산된 서비스마다 분리된 DB 간의 트랜잭션 관리가 어려울 수 있다. 또한 반 정규화된 데이터의 동기 처리도 신경을 써야 한다.

통합 테스트 어려움

MSA 기반의 애플리케이션을 테스트하는 것은 번거로울 수 있다. 테스트를 시작하기 전에 의존성이 있는 서비스를 미리 확인해야 한다.

배포 복잡도

MSA의 배포는 복잡할 수 있다. 각 서비스 간의 조정이 필요할 수 있다.

MSA를 도입하다 보면 기존에 없던 새로운 문제점들이 발생한다. 이런 부분들은 자동화 도구를 사용하거나 비즈니스 상황에 맞게 수정된 아키텍처를 적용하여 해결할 수 있다. 그럼에도 불구하고 시스템과 서비스의 확장성을 고려하고 운영의 효율과 유지보수를 생각하면 단점보다는 장점이 많은 아키텍처다. 그리고, 이런 것들은 점점 더 보완하는 기술들이 나오고 있다. 서비스에 대한 유연한 확장성이 필요하다면 시스템을 기획 및 구축할 때 MSA 아키텍처 설계를 고민해 봐야 할 것이다.