

## 5장 아키텍처

### 아키텍처

아키텍처란 무엇일까? 소프트웨어 시스템에 대해서 이야기 하다보면, “아키텍처가 어떻다”. “최신 아키텍처를 적용했다.” 등 아키텍처에 대한 언급이 많다. 그렇다면, 소프트웨어 아키텍처에 대한 정의는 무엇일까?

<http://www.sei.cmu.edu/architecture/start/glossary/community.cfm> 를 보면, 수많은 아키텍처에 대한 정의가 있다. 지금부터 설명하고자 하는 아키텍처에 대한 정의는 다음과 같다.

“아키텍처는 비즈니스 요구 사항을 만족하는 시스템을 구축하기 위해서 전체 시스템에 대한 구조를 정의한 문서로, 시스템을 구성하는 컴포넌트와, 그 컴포넌트간의 관계, 그리고, 컴포넌트가 다루는 정보(데이터)를 정의한다.”

또한 소프트웨어 아키텍처는 현재의 요구사항뿐 아니라 변화되는 비즈니스 전략에 대응이 가능하도록 장기적인 로드맵을 수용하여 확장가능한 형태로 디자인 되어야 하며 가능하면, 구현 및 사용하고자 하는 조직의 기술 수준, 조직의 규모와 형태 그리고 비즈니스의 형태에 맞춰서 설계 되어야 한다.

### 아키텍처 설계 프로세스

앞에서 아키텍처에 대한 정의는 끝냈다. 그렇다면 실제 아키텍처는 어떤 형태로 설계해야 할까?

아키텍처 설계 방법론은 여러가지가 있으나, 주로 사용되는 프레임워크로는 Zachman Framework, TOGAF, Federal Enterprise Architecture등이 있다. 여기서 소개하는 아키텍처 설계 프로세스는 Open Group <sup>1</sup>에서 만든 TOGAF(The Open Group Architecture Framework) 방법론을 기반으로 하여 프로세스를 정의하였다. TOGAF에 대한 자세한 설명과 템플릿은 <http://www.opengroup.org> 나 <http://www.togaf.org/> 를 참고하기 바란다.

#### 비즈니스 아키텍처 이해

아키텍처는 앞서 정의하였듯이, 비즈니스를 성공적으로 이끌기 위해서, 만들어지는 시스템에 대한 설계다. 목적이 “비즈니스의 성공”에 있기 때문에, 그 비즈니스 자체가 어떤 목표,전략을 가지고 있는지를 이해해야, 목표에 부합하는 아키텍처를 설계할 수 있다.

#### 아키텍처 설계 원칙 정의 (Architecture Principals)

비즈니스 아키텍처가 정의되었으면, 시스템을 설계 하기 위한 기술적인 시스템 아키텍처를 설계

---

<sup>1</sup> <http://www.opengroup.org/>

하기 위한 원칙을 정한다. 품질 속성이나, 기간, 조직 운용론, 기반 기술등 설계의 기본 사상이 되는 원칙을 정의한다.

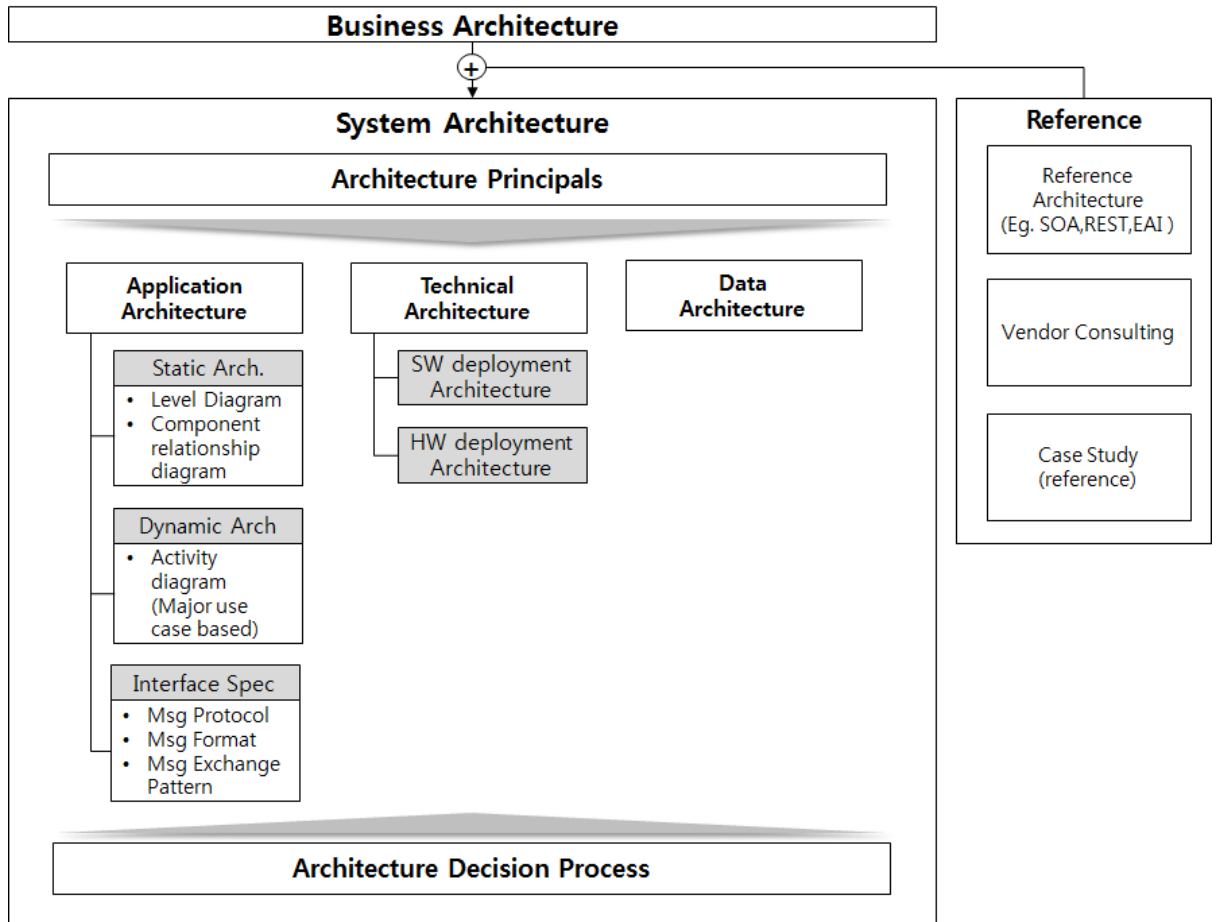
## 시스템 아키텍처의 (System Architecture) 설계

설계 원칙이 정해졌으면, 비즈니스의 요구 사항을 이 설계 원칙에 따라서 설계한다. 시스템 아키텍처는 크게 아래와 같이 3가지로 나뉘어 정의된다.

- ① 애플리케이션 아키텍처 (Application Architecture) : 개발해야하는 애플리케이션 소프트웨어에 대한 아키텍처를 설계한다. 여기에는 컴포넌트의 정의, 컴포넌트 간의 관계, 특정 기능에 대한 컴포넌트간의 호출 흐름, 그리고 컴포넌트간의 통신을 위한 메시지에 대한 규격 정의를 포함한다.
- ② 테크니컬 아키텍처 (Technical Architecture) : 애플리케이션의 배포 구조를 정의한다. 애플리케이션을 배포할 하드웨어의 구조와, 애플리케이션 개발에 사용하는 미들웨어 (DBMS, 웹서버등)등의 배포 구조를 함께 정의한다.
- ③ 데이터 아키텍처 (Data Architecture) : 마지막으로, 애플리케이션에서 다루는 정보(데이터)의 구조와 관계를 정의한다.

이 아키텍처의 설계는 기반 지식이 없는 상태에서는 설계가 어렵다. 물론 경험을 가지고 할 수 있겠지만, 참고할 수 있는 레퍼런스가 있다면 실수나 실패를 줄이고, 시간 또한 단축 시킬 수 있다. 참고자료는 CBD,SOA,EAI와 같은 일반적인 애플리케이션을 개발하기 위해서 패턴화된 아키텍처 스타일을 응용하거나, 유사한 도메인의 CASE STUDY (선행 사례) 기반의 아키텍처, 그리고 솔루션을 사용할 경우, 솔루션 제공사의 컨설팅 서비스를 이용하면, 매우 효율적으로 아키텍처 설계를 할 수 있다.

이 설계 프로세스를 도표화해보면, 다음과 같다.



그러면 이제 부터 각 단계에 대해서 상세하게 설명한다.

## 1. 비지니스 아키텍처의 정의

비지니스 아키텍처는 비지니스 목적을 달성하기 위한 시스템 개발을 위해, 비지니스의 목적과 모델을 이해한다. 이를 위해서 비지니스 모델을 도식화 하고, 기술 조직이 이를 이용하여 비지니스 모델과 목적을 이해한다.

비지니스 아키텍처에는 주로 다음과 같은 내용을 포함한다.

- ① 시스템의 핵심 기능
- ② 시장 현황과 차별화 요소
- ③ 비지니스 로드맵과 일정
- ④ 투자 및 수익 정보
- ⑤ 시스템을 사용하는 사용자 (End User를 포함, 내부 인원 등)에 대한 도메인 모델
- ⑥ 프로세스 (주요 기능에 대한 프로세스)
- ⑦ Business Information (시스템에서 주로 다루는 정보)

추상적인 내용이기 때문에, 비지니스 아키텍처에 어떤 내용이 들어가야 할까? 이해가 어려울 수 있겠지만, 쉽게 예를 들면 “사업계획서”, “시스템 개발 제안서” 와 같은 형태의 문서를 생각하면 된다. 해당 비지니스에 대한 목적이 정의되고, 시스템의 목적과 주요 특성, 구축일정, 비용등이 함

게 정의 된다. 이 문서를 읽었을때, 이 비즈니스를 수행하는 조직(회사)의 모든 인원들이 그 내용에 대해서 이해할 수 있어야 한다. 기술쪽 인원이건, 비즈니스(영업,마케팅,경영)의 원리와 로직을 이해할 수 있어야 한다.

## (1) 서비스 모델의 정의

서비스의 형태를 정의한다. 블로그 서비스를 계획하고 있다면, 블로그 서비스의 비즈니스 모델 정의 예시는 다음과 같다.

“블로그란, 일반적인 사용자가, 자신의 글과 생각을 손쉽게 인터넷에 올릴 수 있는 시스템으로, 1인 미디어의 성격을 갖는다. 개인이 생산한 블로그내의 컨텐츠는 다른 사람들에게 흥미나 지식을 전달하는 가치를 제공함으로써, 블로그 시스템으로 많은 인터넷 사용자의 유입을 유도할 수 있다.

당 서비스에서는 일반 사용자에게 블로그 서비스 시스템을 제공하고, 일반 사용자는 컨텐츠를 제작함으로써, 이 컨텐츠를 통해서 인터넷 사용자의 유입을 유도하고, 블로그 시스템에 광고를 유치함으로써 수익을 창출한다.”

이 비즈니스 모델을 간략하게 도식화 해보면 다음과 같다.



아주 간략하게 설명하였지만, 사업 제안서를 쓴다고 생각하면 된다.

어떤 시스템인지를 설명하고, 어떤 사용자와 관계를 갖으며, 어떠한 가치를 창출해 내며, 어떻게 수익을 창출하는지를 서술해야 한다. 다음은 또 다른 예로, 게임 퍼블리셔와 호스팅 사업자 그리고 소프트웨어 벤더 3사가 협업하여, SNS 게임 벤처 개발 업체를 대상으로 6개월간 투자를 통해서 적은 비용으로 SNS 게임을 개발하여 출시할 수 있도록 하고, SNS 게임이 수익이 실제 수익이 나는 6개월 후 부터 투자 비용을 수거하는 비즈니스 모델과 수익 모델을 정리해놓은 예이다.

## SNS 게임 개발사 지원 모델

### • 플랫폼 사업자 서비스 아키텍처

PAAS 플랫폼 사업자가 제공해야 하는 PAAS 플랫폼의 Conceptual 아키텍처는 다음과 같다.

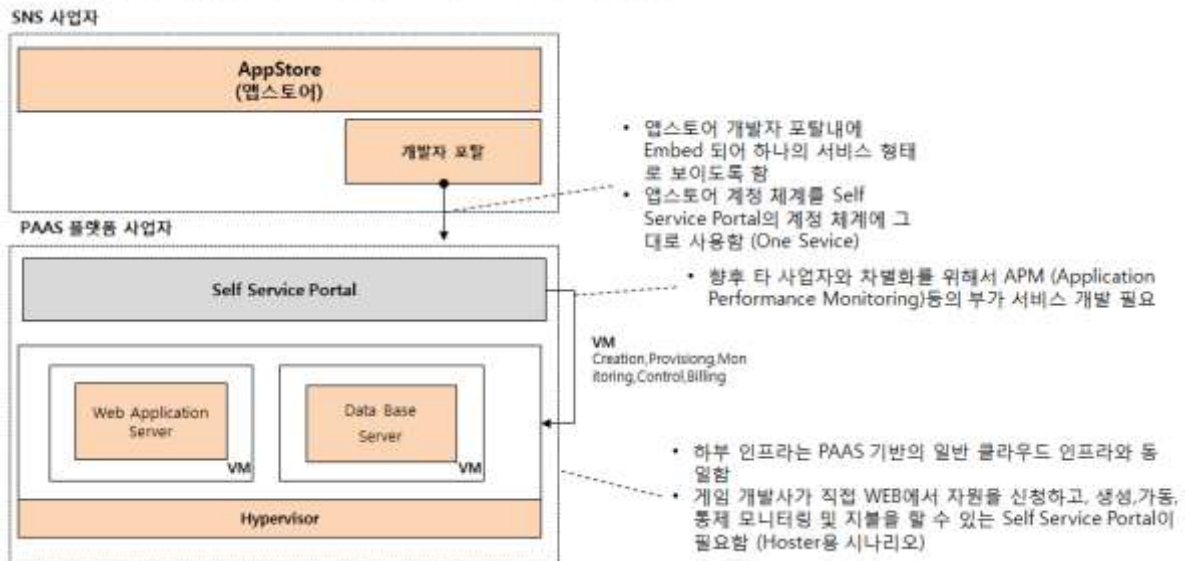


Figure 1 서비스 모델

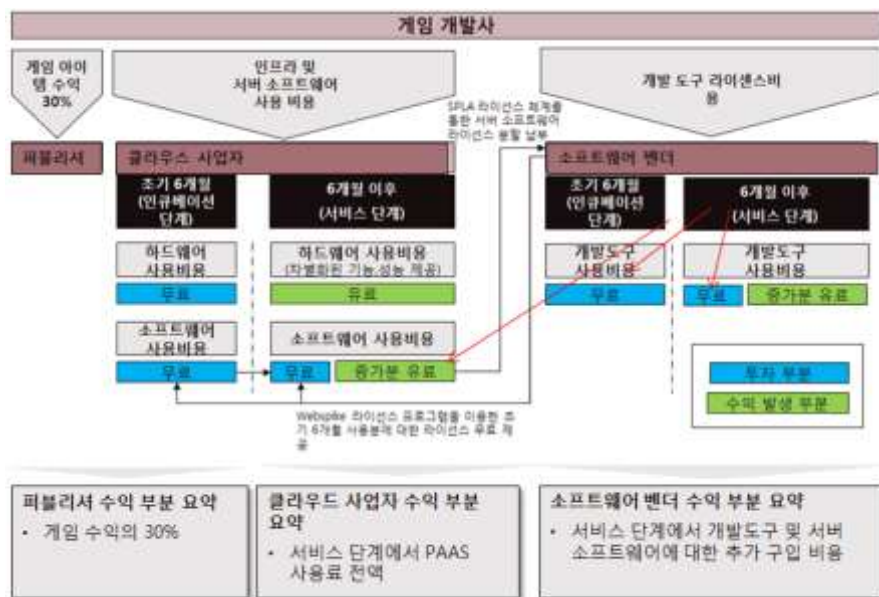


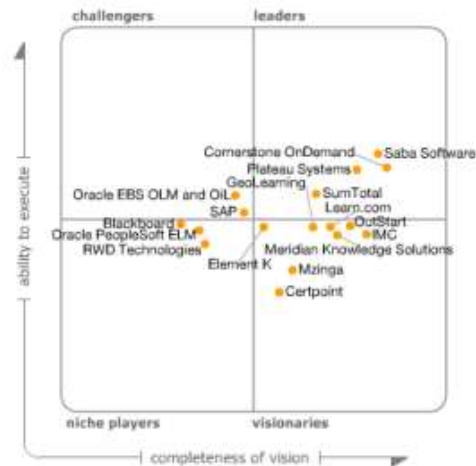
Figure 2 수익 모델

## (2) 시장 현황 분석

비즈니스에 대한 정의와 모델 정의가 끝났으면, 해당 비즈니스에 대한 시장 현황과 경쟁사들을 분석해야 한다. 상당히 비즈니스적인 내용임에도 불구하고 아키텍처 문서에 포함 시키는 이유는, 경쟁 제품이 있을 경우 기능이나 성능들의 비교 대상이 될 수 있기 때문이다. 장단점 분석을 통

해서 조금 더 나은 시스템을 설계할 수 있기 때문이다.

시장 규모, 경쟁사 자료들은 가트너 컨설팅 자료를 참고하면 전체 시장 규모나 선도 업체들을 볼 수 있다. 특히 가트너의 magic quadrant report는 해당 비즈니스의 선도 업체를 잘 리포팅해준다.



**Figure 3 예) LMS(교육 관리 시스템)에 대한 가트너 magic quadrant report**

출처 : <http://technologies.learning-managers.net/technologies/lcms>

경쟁사의 제품 기능 분석은 경쟁사의 홈페이지를 통해서 찾아볼 수 있고, 매출 규모등은 홈페이지에 공시된 IR (Investor Report – 상장 회사의 경우 투자자에게 매 분기별로 비즈니스 사항을 리포팅해야 함) 자료를 참고하면 얻을 수 있다.

이렇게 자료를 수집하여, 자社の 시스템에 대한 기능과 타社 시스템에 대한 기능 비교표를 만들 수 있고 이를 통해서, 차이점과 장단점을 비교해서 집중해야할 기능 항목을 정의하는데 참고할 수 있다.

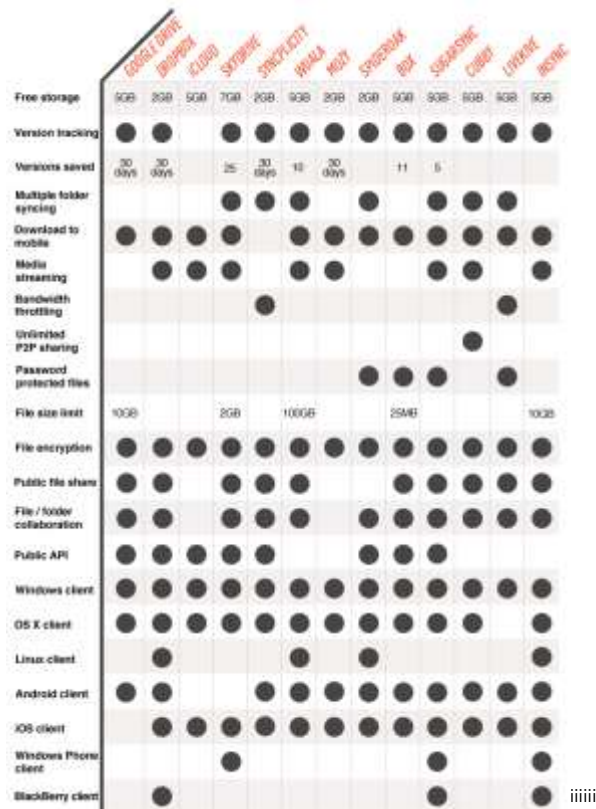


Figure 4 예) 개인 클라우드 스토리지 서비스 기능 비교표

출처 : <http://www.theverge.com/2012/4/24/2954960/google-drive-dropbox-skydrive-sugarsync-cloud-storage-competition>

### (3) 비즈니스 전략

비즈니스 모델과, 시장 현황 그리고 경쟁 제품이 정의 되었으면, 이 상황에서 성공적으로 비즈니스를 수행하기 위한 전략을 정의한다.

주로 차별화 전략으로, 기능적인 차별화, 판매 금액, 서비스, 시장 진입 시간, 원가 절감등이 전략 항목의 예가 될 수 는 있다. 포괄적인 방향성을 정의해야 하는데, 비즈니스 전략은 소프트웨어 시스템의 특징과도 연관이 된다.

예를 들어 원가를 절감하려면 오픈소스를 사용해서 원가를 사용하는 방법도 있고, 시장 진입 시간을 줄이는 서비스를 개발하기 위해서는 클라우드를 이용해서 하드웨어 배포 시간을 줄이는 방법도 있다. 기능적인 차별화를 위해서는 특정 기술에 집중하는 방법도 있다.

이렇게 아키텍처적인 특성은 비즈니스 전략에 영향을 받아서 정의되는 경우가 많다.

### (4) 주요 기능 정의

다음으로, 시스템에 대한 주요 기능을 정의한다. 비즈니스 관점에서의 기능이 되기 때문에, 세세한 기능 보다는 10~20 개 정도의 큰 기능으로 정의하는 것이 좋다. 상세한 기능 정의는 시스템 아키텍처에서 정의 한다. Actor 와 기능 중심의 Use Case Diagram과 같은 도표를 사용하는 것도 유

용하다.

주요 기능을 정의할때, 꼭 들어가야 하는 것중의 하나는 이 시스템을 사용하는 “사용자” 와 “시스템”간의 상호작용(Interaction)을 정의해야 한다.

## (5) 비지니스 로드맵

비지니스 로드맵은 해당 비지니스의 장기적인 일정을 서술한다.

출시 일정, 시장 공략 계획등을 포함한다.

비지니스 로드맵은 시스템의 개발 및 출시 일정과도 연관이 된다. 블로그나 SNS와 같은 순수 소프트웨어 서비스일 경우에는 비교적 개발 일정 조정이 가능한 편이지만, 스마트폰이나 TV와 연동되는 소프트웨어나 공장 자동화 같은 소프트웨어의 경우에는 제조부분과 연동이 되어 있기 때문에 마케팅, 광고, 물류등의 다른 비지니스 유닛과 연결이 된다. 그래서 개발 일정이 지연될 경우 큰 손해를 입을 수 있다. 이런 이유로 비지니스 로드맵을 정확하게 알아야 개발 일정을 조정할 수 있고, 필요에 따라 기능을 조정하거나 단계적인 출시 일정등을 고려할 수 있다.

다음은 블로그 시스템에 대한 로드맵 예제이다.

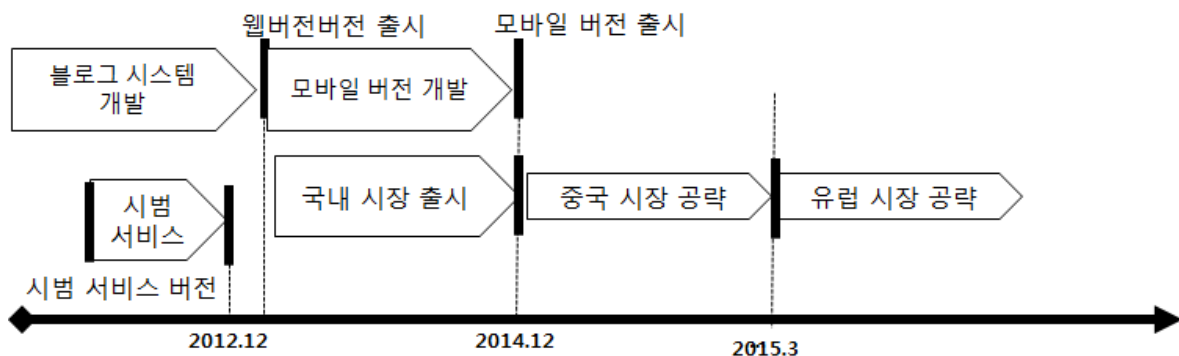


Figure 5 블로그 시스템 비지니스 로드맵 예시

아주 간단하게 표현하였지만, 실제 로드맵 작성시에는 반드시 제품의 출시 일정과 버전별 릴리즈 계획등이 정교하게 포함되어야 한다.

지금까지 상당히 복잡한 내용을 설명하였다. 비지니스 아키텍처는, 기술 아키텍처를 비지니스 목적에 부합시키기 위해서, 비지니스를 이해하는 것을 목적으로 작성하는 문서이다. 직접 기술 아키텍처가 이런 내용을 작성하기는 어렵다. 아키텍처 설계가 들어간다는 것은, 이미 위에서 서술한 대부분의 모델이 어느정도 분석이 끝났다는 것이다. 적어도 어떤 비지니스를 할 것이며, 시장 상황 등은 정의가 되었을 것이며, 비지니스 전략 역시 마케팅이나 경영쪽에서 작성해야 하는 내용이니 크게 걱정을 할 필요는 없다.

이미 작성된 내용을 다시 기술팀 입장에서 재 정의하고 문서를 작성함으로써 비지니스를 이해한다.



## 2. 아키텍처 설계 원칙 (Architecture Principal) 정의

비지니스를 분석했으면, 설계 해야 하는 시스템의 설계 원칙 (Architecture Principal)을 정한다. 이는 비지니스의 요구 사항에 의해서 원칙이 세워진다.

예를 들어 “경영 악화에 의해서 IT에 투자를 많이 하지 않는다”면, Outsourcing 등을 고려할 수 있고, 아키텍처 역시 외주 개발팀에 의해서 분산 개발이 용이한 아키텍처 스타일이 되어야 한다. 또는 “SNS에서의 사용자의 반응을 실시간으로 분석해서 제품 마케팅에 활용한다” 와 같은 비지니스 요구 사항이 있을 경우에는 “SNS의 데이터를 실시간으로 분석할 수 있는 빅데이터 기능을 아키텍처에 반영해야 한다.”

이러한 설계 원칙을 Architecture Principal이라고 하고, 아키텍처 디자인의 기본 원칙이 된다.

Architecture Principal은 보통 7~15개 정도를 정의하는 것이 적절하다. 요구사항 변경이나, 기술적인 난이도, 조직의 숙련도등 많은 요인에 의해서 아키텍처를 변경해야할 상황이 발생하는데, 이때, 판단의 기준이 되는 것이 Architecture Principal이다.

다음은 클라우드 컴퓨팅 시스템 구축을 위한 Architecture Principal의 예이다.

### Architecture Principals

- **Fundamental Principals which drives Cloud Architecture Design**

1. Infinite capacity
2. Continuous availability
3. Predictability
4. Take a service provider's approach to delivering infrastructure
5. Resiliency over redundancy mind set
6. Minimize human involvement
7. Optimize resource usage
8. Incentive desired resource consumption behavior

### 5.Resiliency over redundancy mind set

- **From the provider's perspective, focus on maintaining service availability through resiliency, rather than redundancy**

- Objective

- Reduce redundancy at the infrastructure level which is highly costly
- Eliminate duplicated redundancy which is typical across several layers of the stack
- Move toward resilience model which is more cost effective
- Push resilience up the stack; designing application for resilience is less costly than redundant infrastructure

### 3. 시스템 아키텍처의 설계

비즈니스 모델의 파악이 끝나고, 아키텍처의 기본 설계 원칙이 정해졌으면 시스템 아키텍처 설계에 들어간다. 시스템 아키텍처는 비즈니스 아키텍처와, 아키텍처 설계 원칙을 기본으로 해서 설계를 하고, 아키텍처 스타일이나 CASE STUDY 등의 레퍼런스 자료를 이용해서 설계를 한다.



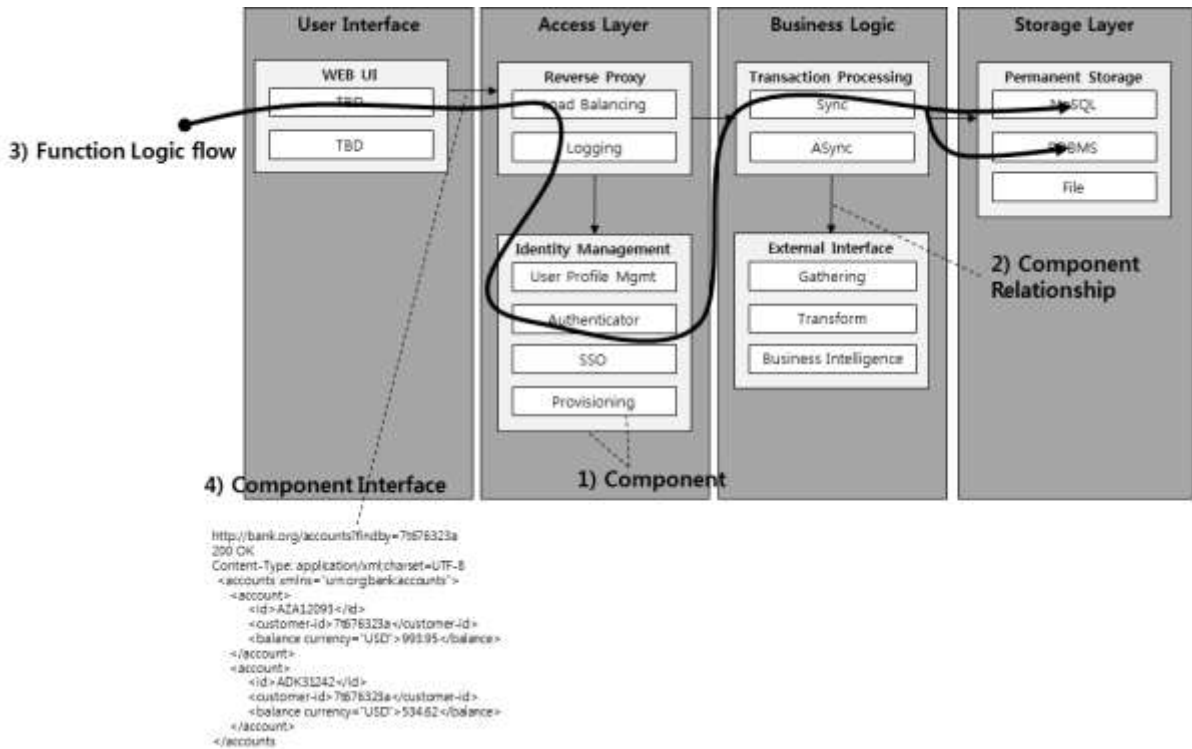
#### (1) 애플리케이션 아키텍처

애플리케이션 아키텍처는 직접 개발하는 소프트웨어 애플리케이션에 대한 아키텍처 구조이다.

애플리케이션은 크게 4가지 요소로 표현할 수 있다.

애플리케이션을 구성하는 컴포넌트, 컴포넌트의 상호 관계, 특정 기능을 수행하기 위한 컴포넌트 간의 호출 순서 그리고 각 컴포넌트간의 호출을 위한 통신 방식 즉 인터페이스에 대한 패턴과 프로토콜로 이루어진다.

이 4가지 구성 요소를 아키텍처 문서로 분류해보면, 애플리케이션 아키텍처의 구조는 컴포넌트들의 구성과, 각 컴포넌트들의 상호 연관 관계를 표현하는 정적인 아키텍처 (Static Architecture)와, 특정 기능에 대해서, 컴포넌트 간의 호출 흐름을 표현하는 동적인 아키텍처 (Dynamic Architecture)와 컴포넌트간의 통신 규격을 정의하는 인터페이스 정의서 (Interface Defintion)로 정의 된다.



< 그림. Application Architecture 구성 요소 >

이제 부터 각 구성 요소를 표현하는 아키텍처 다이어그램들에 대해서 설명한다.

## Static Architecture

### 컴포넌트 구성의 표현

시스템을 구성하는 컴포넌트들을 표현한다. 계층별로(Layered) 표현하며, 각 컴포넌트에 대한 설명을 간략하게 서술한다. 컴포넌트를 break down하여 보통 Level0~Level3까지 설계하고, 더 상세한 부분은 각 컴포넌트의 상세 설계 문서에서 기술한다.

Level 0의 경우에는 일반적인 계층 (Layer or Tier)를 정의한다. 주로, SOA(Service Oriented Architecture), CBD, 3-Tier Client Server와 같은 Architecture Style에 영향을 많이 받는다. 이러한 Architecture Style은 이미 정해진 계층 구조를 가지고 있기 때문에, 설계 하고자 하는 아키텍처가 이러한 Reference Architecture에 영향을 받았다면 상당히 유사한 구조의 계층 구조가 나오게 된다.

아래는 요즘 일반적으로 사용되는 OPEN API나 SOA 기반의 아키텍처에 대한 계층 구조와 각 컴포넌트의 기능을 Level 0 기준으로 표현하면 다음과 같다.

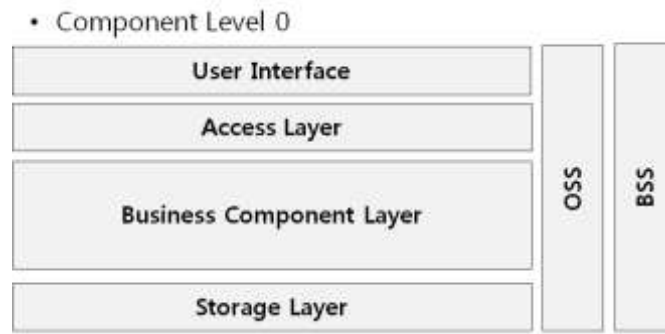


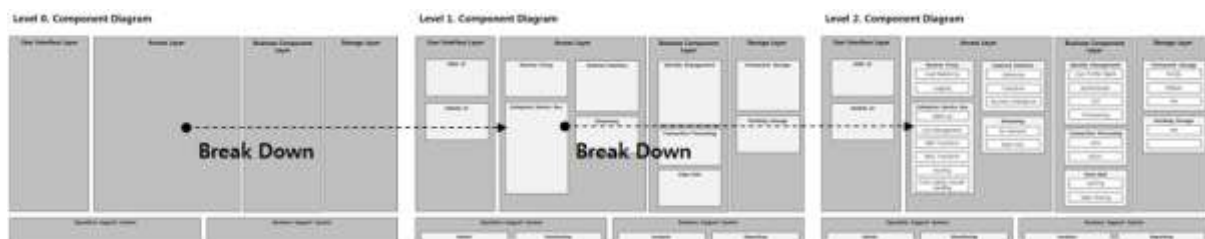
Figure 6. Level 0 Component Diagram

• Component Level 0 Description

Level 0	Description
User Interface	웹,모바일등 사용자 인터페이스 계층
Access Layer	비즈니스 로직을 OPEN API를 이용하여 외부로 서비스 하는 계층 외부 시스템과의 인터페이스를 제공
Business Component Layer	비즈니스 로직을 구현하는 계층
Storage Layer	데이터를 저장하는 계층
OSS	운영,모니터링 관리를 위한 계층
BSS	비즈니스를 위한 지표 분석 리포팅 서비스

Figure 7. Level 0 Component Description

Level 0가 정해졌으면, 각 블록을 Break down해서 Level1,2,3 단계로 아래 그림과 같이 정의해나간다. 마찬가지로, 각 Level별로 정의된 컴포넌트에 대해서도 Component Description을 서술한다.



이 단계까지 디자인을 하면, 전체 시스템이 어떤 구조(Structure)를 가지는지 정의가 된다

특히 블록들은 Level1이나 2정도에서 각 개발 UNIT으로 분리될 수 가 있다. 예를 들어 Business Component Layer내에, Component A,B,C는 A팀이 개발, D,E,F는 B팀이 개발하고 UI Layer에서 Mobile은 C팀이 개발, WEB UI는 D팀이 개발하는 식으로 나뉘어 질 수 있다.

이는 향후에 팀 모델링과 TASK 기반의 개발 프로세스 관리등과도 연계성을 가지게 된다. 개발 TASK를 상위 레벨 (Level 1정도) 블록으로 Categorize(분류)하여 관리하면 시스템에 대한 개발 진척도를 가시화하여 관리할 수 있다.

또 다른 예를 보자, 아래 그림은 통신사의 서비스 Delivery 플랫폼의 상위 아키텍처이다. 앞서 설명한 계층화된 컴포넌트 다이어그램의 구조로 서술이 되어 있다. 각 컴포넌트들은 기능적인면으로 서술이 되어 있다. 약간 특이한 점은 맨 아래, Consumer,Provider, Network & Gateway Layer와 같은 내부 소프트웨어 컴포넌트가 아닌 부분이 서술되어 있는데, 이렇게 시스템 아키텍처 설명에

필요하다면 외부 시스템에 대한 연계 구조를 함께 포함하는 것도 구조 설명에 도움이 된다.

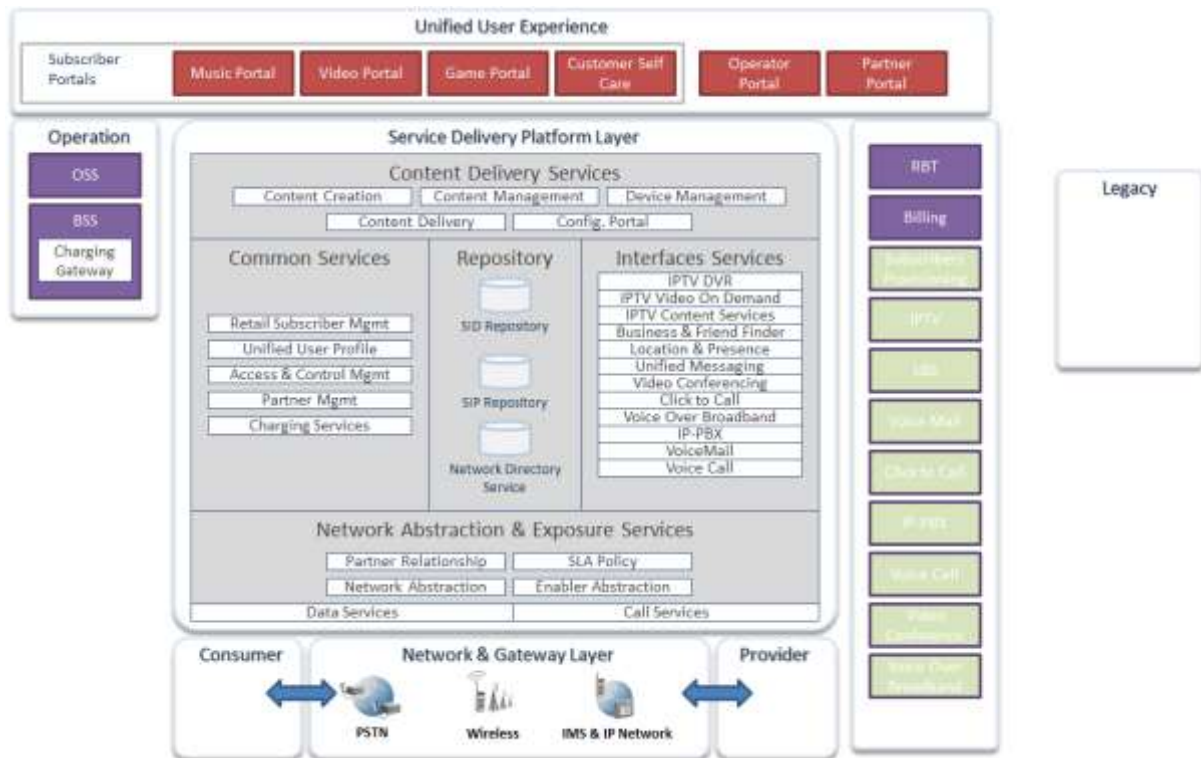
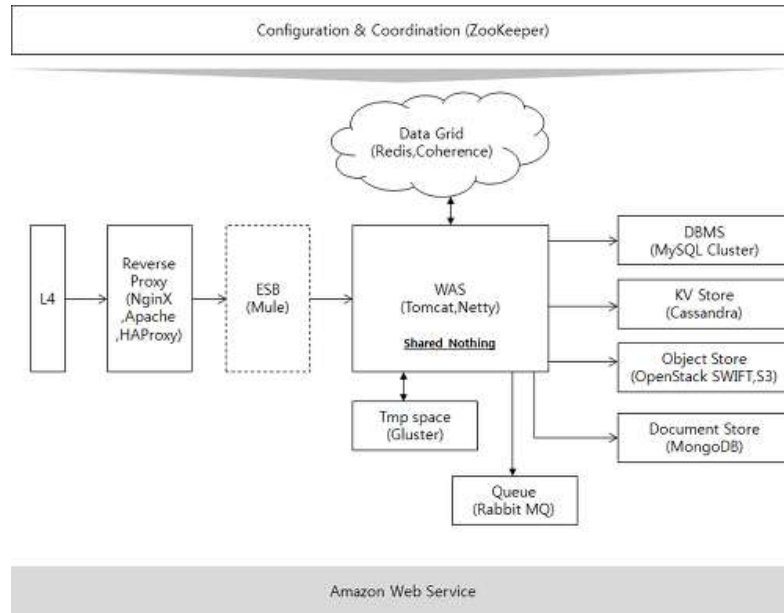


Figure 8 통신사의 통합 서비스 Delivery 플랫폼 상위 아키텍처

## 컴포넌트간의 관계 표현

컴포넌트 구성을 정의했으면, 이 컴포넌트들이 상호 유기적으로 어떤 관계를 갖는지를 정의해야 한다. 이 관계는 컴포넌트간의 호출 구조를 정의하면 된다. 함수의 호출 관계를 연상하면 이해가 쉽게 될 것이다. 아래는 일반적인 웹 시스템에 대한 컴포넌트간의 관계를 표현한 다이어그램이다.



**Figure 9 컴포넌트간 관계 표시도**

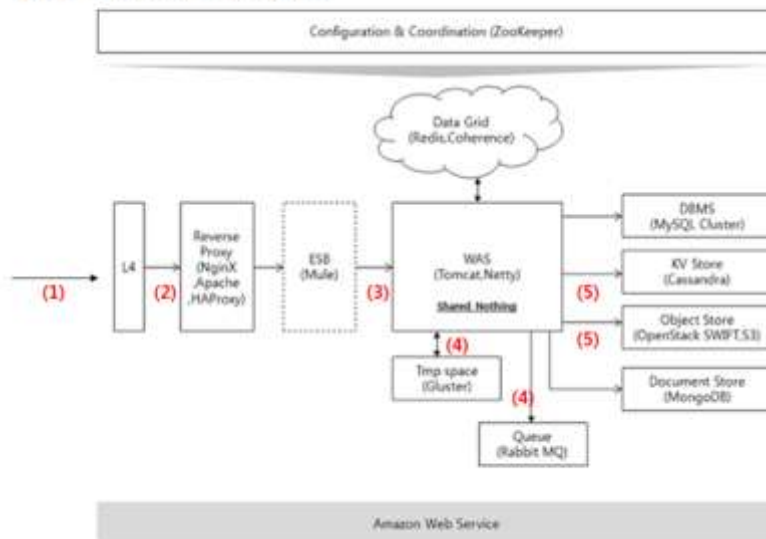
이렇게 Static Architecture를 디자인할때, 흔히 혼동 되는 사항이 컴포넌트의 명명과 분리를 어떻게 할것인가인데, 컴포넌트 명을 "컨텐츠 관리","사용자 관리","학습 시스템","매출 관리 시스템" 과 같은 비즈니스 단위로 분리할 수 도 있고, "Transaction Processing", "Load Balancing";과 같이 기술적인 단위로 분리할 수 도 있다. 디자인시 여기서 많은 혼동을 유발하는데, 가이드는 상위 계층의 분류법은 비즈니스 위주를 사용하고, 하위 디테일로 갈 수 록, 기술적인 단위로 분리하는 것이 좋다.

## Dynamic Architecture

컴포넌트의 종류와 구조가 파악되었으면, 이제 부터 주요 시나리오에 대해서, 각 시나리오를 처리하기 위한 흐름을 표시한다. 컴포넌트 관계도를 이용하여, 각 시나리오에 대한 호출 흐름에만 번호를 적어서 순차적으로 설명할 수 도 있고, 시나리오의 특성상 낮은 레벨의 컴포넌트가 필요하거나, 기타 기술적인 사항이 있으면 추가로 표시하도록 한다.

## Dynamic Architecture

### File Upload Scenario



- ① 클라이언트가 HTTP 방식으로 파일에 대한 파일 메타 정보와 함께, 파일을 업로드 한다.
  - ② Reverse Proxy를 통해서 request를 뒤에 연결되어 있는 Web Application Server (WAS)로 전달한다. 파일 업로드 처리 서버는 N개 이상으로 클러스터링 구성이 되어 있으며, 이 Reverse Proxy에서 항상적으로 살아 있는 서버중에서 시스템 부하가 제일 낮은 서버를 찾아서 request를 전달한다.
  - ③ WAS로 업로드 된 파일은, 웹네일 추출등을 위해서, 물리적 파일은 Tmp space에 우선 저장되고, 파일 처리에 필요한 헤더 정보 및 메타 정보 (사용자명, 파일 이름, 파일 업로드 위치)등은 Queue에 저장되어 비동기 처리가 된다.
- : (중략)

애플리케이션 아키텍처는 소프트웨어 컴퍼넌트간의 구조를 표현하지만, 경우에 따라서 설명을 돕기 위해서는 하드웨어나 네트워크에 대한 구성이나, 소프트웨어 솔루션을 참고 사항으로 표현할 수 있다.

위의 그림에도, L4 스위치등이 표현되어 있고, 각 컴포넌트의 구성 미들웨어들이 표현되어 있다. 이는 시스템을 구성하는 하드웨어 및 솔루션 컴포넌트를 표현하여 이들 특성에 따른 아키텍처 흐름을 표현하기 위함이다.

### Detail Architecture

Static Architecture와 Dynamic Architecture를 정의하였으면, 전체 시스템의 구조와 흐름을 표현한다. 큰 그림 설명을 통해서 전체 시스템에 대략적인 설명은 하였고 기능별 상세 디자인등은 별도의 디자인 문서에서 진행한다.

여기서 고려해야할 사항은, 아키텍처는 전체 시스템의 큰 그림의 설계이고, 각 컴포넌트 개발팀은 컴포넌트에 대한 상세 설계와 구현 및 테스트를 담당한다. 그래서 아키텍처에 정의된 설계 사상이 개발팀에까지 연속적으로 잘 전달되는 것이 필요한데, 아키텍처 설계상의 특징이나 사상을 담은 설계의 경우에는 별도로 상세 디자인을 포함한 "Detail Architecture"라는 것을 작성하여, 문서에 포함하도록 한다.

다음은 동영상을 병렬로 인코딩 하는 시스템에 대한 상세 아키텍처이다. 예제로 사용하는 용도라

서 한장 정도만 첨부하였지만, 상세 디자인 수준으로 자세하게 서술해야 한다.

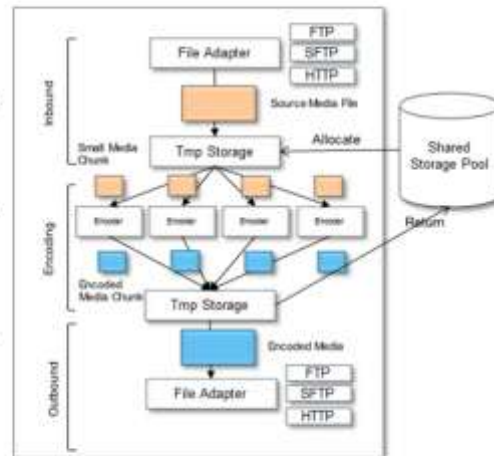
## Transcoding Component Architecture

### Encoding workflow (Parallel Encoding) detail

- 1) File Adapter gets source file.
- 2) File Adapter allocate temp storage area to store source file.
- 3) The source file is stored into the temp storage area.
- 4) The source file is spited to multiple small chunk.
- 5) Multiple Encoders are invoked and each encoder encode the small chunks. After all the chunks has been encoded, it is merged one file. (Similar to Map & Reduce). By using parallel processing it enables us to maximize hardware resource utilization.
- 6) After finishing the encoding, encoded result file is transferred into destination by using File Adapter.

### Tmp Storage Consideration

Tmp Storage is just used to store source file and encoded result file – (Temporary working space)  
Requirement is provide high performance IO but reliability is not required. (If IO fail is occurred just retry it.)  
It is recommended using Local Disk in physical server.



## Interface Definition (인터페이스 정의서)

인터페이스 정의서에서는 각 컴포넌트들이 통신하는 방법을 정의하며, Protocol, Message format 그리고, Message Exchange Pattern (MEP) 3가지 항목으로 구성된다.

### Protocol

Protocol은 통신 프로토콜이다, JSON/HTTP 기반의 REST, SOAP 기반의 웹서비스, 구글의 Protocol Buffer와 같은 통신 프로토콜이 해당한다.

### Message Format

다음은 Message Format으로, Protocol에 실려서 전송되는 메시지에 대한 포맷(스키마)를 정의한다. SOAP 기반의 웹서비스에서는 WSDL등이 해당하며, 이해하기 쉬운 것으로는 OPEN API의 명세서등이 있다. 여기서 정의 하는 Message Format은 큰틀의 아키텍처 부분에 대한 설계이지 개별 인터페이스에 대해 각각 Message Format을 정의하는 것이 아니다.

메세지의 전체적인 구조를 기술하는 것으로, 메세지의 헤더, 전체 구조, 에러 코드등을 정의한다.

메세지 구조 설계시, 구글이나 Amazon과 같이 OPEN API를 제공하는 해외 유수의 서비스나, 솔루션들의 API 명세 문서를 참고하면 많은 도움이 된다.

예) Amazon Web Service 클라우드 서비스의 Simple Storage Service (S3)의 OPEN API 명세서



<http://docs.amazonwebservices.com/AmazonS3/2006-03-01/API/RESTBucketGET.html>

Message format에는 request와 response 양쪽을 모두 정의해야 하며, 특히 response에는 정상적인 응답뿐만 아니라 예외 사항(에러)에 대한 Message format을 포함하여 정의해야 하며, 특히 장애에 대해서는 에러코드를 정의해야 한다.

### Message Exchange Pattern

마지막으로는 Message Exchange Pattern (MEP)인데, 메시지를 호출하는 방법을 정의한다. 우리가 일반적 예로, 프로그래밍에서 사용하는 호출 형태는 Synchronous 형태로 request가 가면 return이 올때 까지 기다리는 호출 패턴이다. 윈도우즈나, 모바일 앱에서 버튼을 누를 때 특정 함수를 호출하게 하는 것은 Event Handling 패턴이다.

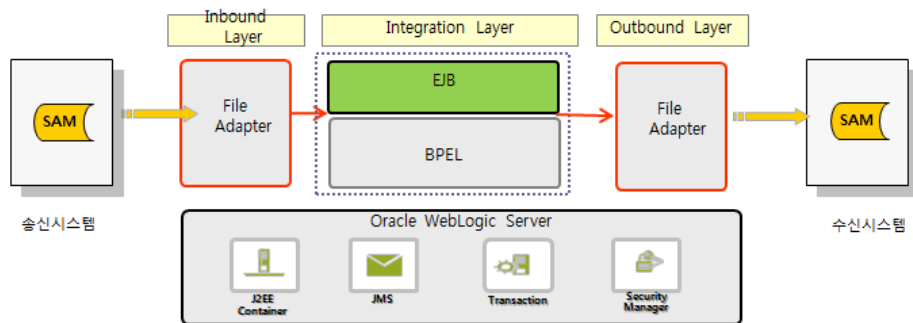
일반적으로 프로그래밍에서 사용하는 API에 대한 함수 호출 패턴이 MEP이다.

대부분의 인터페이스는 동기식 (Synchronous) 방식을 사용하는데, 이 방식은 필수적으로 사용되고 난이도가 있는 부분은, 비동기식 (Asynchronous) 호출 패턴으로, Message Queue를 이용한 메세징 패턴이다. 동기식의 경우 호출 후, 성공 실패 여부를 바로 리턴받기 때문에 그리 구현이 어렵지 않지만, 비동기식의 경우 호출 한 후, 비동기적으로 서버에서 처리한 후에 리턴을 해야 하는데, 호출한 클라이언트가 응답을 대기하지 않기 때문에 에러 처리가 쉽지 않다. 특히 에러가 났을때에 대한 처리 방식 - "재시도, 무시, 관리자에게 알림" 등의 에러 처리 정책을 정의해야 하는데, 이러한 에러 처리를 하는 패턴을 "Error Hospital"이라고 한다. (에러가 나면 병원에게서 치료를 하고 온다는 의미로). 이 "Error Hospital"의 구현은 난이도가 높고, 시간이 많이 걸리기 때문에 특별히 신경을 써야 하는 부분이기도 하며, 비동기식의 경우 클라이언트와 서버의 관계가 M:N의 multiplicity를 가지기 때문에 복잡도가 비교적 매우 높다.

인터페이스 관점에서는 RPC 형태의 함수 호출만 생각하면 안된다. 컴포넌트간을 연계하는 방법은 RPC 뿐만 아니라, FTP를 사용한 FILE 인터페이스 방법, DBMS의 테이블을 복사하는 테이블 인터페이스 방식등 여러 가지 방식이 있기 때문에, 인터페이스 아키텍처 정의에 의해서 유연적으로 설계를 해야한다.

다음은 EAI 시스템을 이용하여, 두 회사간의 인터페이스를 하는 아키텍처이다. FILE을 이용하여 메시지를 전달하는 아키텍처이다.

### 3.1 File 연계 (Inbound/Outbound)



#### 1. 개요

EAI Inbound/Outbound Layer에서의 File Adapter 를 사용하여 송신시스템과 수신시스템간의 File 연계를 위한 인터페이스 구성 설계

#### 2. 설계 내용

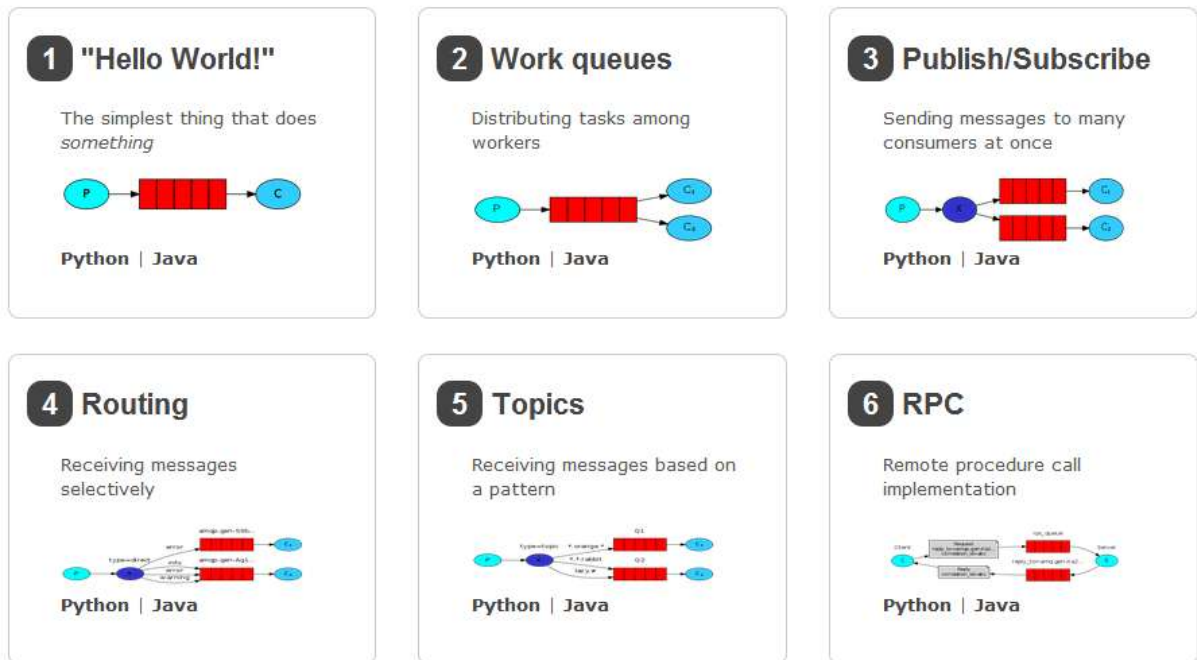
##### 2.1 처리순서

- 1) EAI 시스템이 송신시스템에 생성된 SAM File을 인지하여 미리 BPEL 에 정의된 수신시스템에 전달한다.

#### 3. 환경설정

File Adapter 정의 시 송신시스템과 File 정보, 수신시스템과 File 정보를 미리 정의한다.

이외에도, Message Queue를 이용하여, 비동기 호출 방식의 MEP 구현 방식이다. 이런 구현 방식은 Message Queue 솔루션들의 문서를 보면, 아키텍처 스타일을 참고할 수 있는데, 아래 그림은 Message Queue 미들웨어인, RabbitMQ에서 소개한 MEP 패턴이다.



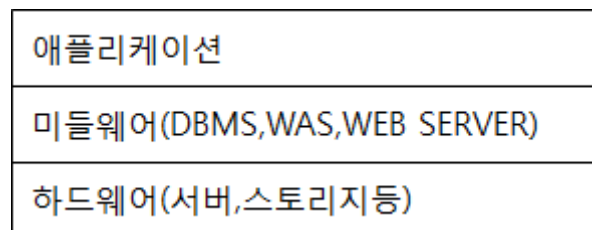
출처 : <http://www.rabbitmq.com/getstarted.html>

MEP 만 가지고도 책한권이 나올만큼 많은 분량이기 때문에, 여기서는 상세한 설명을 하지 않는다. MEP에 대한 상세한 설계는 미들웨어나, EAI 아키텍처, OPEN API 아키텍처 스타일등을 참고하

고, 서적으로는 “Enterprise Integration Patterns”<sup>2</sup>를 참고하면 조금 더 다양한 패턴을 참고할 수 있다.

## (2) 테크니컬 아키텍처

테크니컬 아키텍처에는 개발한 애플리케이션을 배포할 솔루션(RDBMS, WAS등의 미들웨어)과 하드웨어에 대한 구조를 정의한다.



시스템이 동작하기 위해서는 애플리케이션 소프트웨어 뿐만 아니라 소프트웨어 애플리케이션이 동작하기 위한 기본적인 미들웨어 (DBMS,웹서버 등)와 이 미들웨어와 애플리케이션을 호스팅할 하드웨어로 이루어진다.

테크니컬 아키텍처에서는 애플리케이션에서 하드웨어까지 연관 구조를 정의한다.

### Hardware deployment Architecture

하드웨어의 배포 구조와 하드웨어를 아키텍처를 구성하는 각각의 컴포넌트, 서버 RACK, 서버, 네트워크 장비, 스토리지에 대한 디자인을 정의한다. 각 서버에 대한 특성등은 다른 하드웨어와 인프라를 다루는 챕터에서 구체적으로 설명하도록 한다.

### 서버 디자인 아키텍처

개별 하드웨어 서버의 디자인을 정의한다. 서버 하드웨어에서 정의해야 하는 내용은

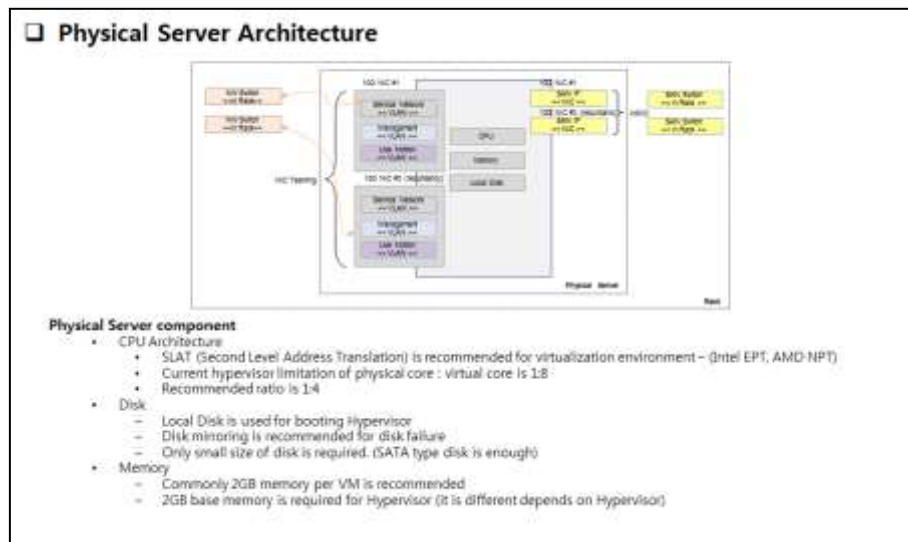
- CPU의 클럭 속도, Hyper Threading 등의 CPU 관련 기술들을 선택하여 설계한다.
- 네트워크 구성 부분에서는 NIC (network interface card)에 대한 디자인을 고려하는데, 네트워크는 단순히 하나의 NIC를 구성하는 것이 아니라 용도에 따라 관리용 인터페이스, 외부 서비스용, 내부 통신용, 스토리지 네트워크 연결등을 애플리케이션의 특성에 맞도록 분리해서 배포한다. 또한 성능이나 안정성등을 고려하여 bonding (두개의 NIC를 논리적으로 하나로 묶는 방법)등을 고려해야 한다.
- 메모리 부분에 있어서도, 서버의 처리능력에 맞게 알맞은 용량을 선정해야 한다. 예를 들어 하나의 WAS가 2 core에서 가장 최적의 성능이 나온다고 가정하였을때, 서버가 12core로 구성되어 있다면, 32 bit JVM에서 작동하는 WAS의 경우 최적의 메모리 사이즈

---

<sup>2</sup> Addison Wesley : Gregor Hohpe , Bobby Woolf, ISBN-10: 0321200683

는 1.5G~2G정도가 된다. 12core이기 때문에, 2 core를 OS의 기본 동작에 사용한다고 가정하면, 나머지 10 core가 WAS에 사용되기 때문에 총 5개의 WAS 인스턴스를 기동할 수 있고, 필요 메모리량은 5개의 WAS 인스턴스가 각 2GB의 메모리를 소요한다면, 10 GB의 메모리가 필요하며, OS가 기본적으로 사용하는 메모리를 2GB로 가정하면 총 12GB가 필요하다.

- 애플리케이션의 형태를 고려하여 로컬 디스크를 설계해야 하는데, TP성의 처리만 할 경우 많은 디스크 용량을 필요로 하지 않기 때문에 작은 수의 디스크만을 사용하면 되지만, 만약 많은 데이터를 저장하는 RDBMS같은 솔루션의 경우 많은 디스크를 사용해야 하기 때문에 서버가 충분한 물리적인 공간을 가지고 있어야 하며, 성능과 안정성 (이중화)를 고려하여 RAID 구성을 결정해야 한다.
- 이러한 내용들을 고려하여, 내부에 충분한 공간 확보를 해야 하는데, NIC를 꼽을 PCI 슬롯, 디스크를 꼽을 수 있는 공간등을 고려해야 하며, 서버 케이스의 크기를 결정해야 한다. (1U,2U,4U) 이러한 모든 장비에 전력을 제공할 수 있는 power supply 에 대한 용량도 함께 설계해야 한다.
- 아울러 서버의 집적도등을 고려하여 서버의 타입 (서버형, 랙마운트, 블레이드)등을 결정해야 한다.

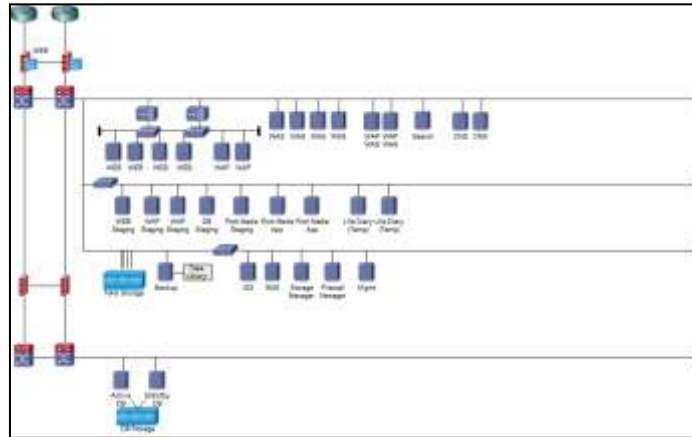


## 네트워크 디자인 아키텍처

서버간의 네트워크 디자인을 정의한다.

- 외부 네트워크와 연결하기 위한 라우터
- 네트워크의 백본이 되는 L2 스위치
- 그리고 로드밸런싱과, 기타 애플리케이션의 특성에 따른 네트워크 서비스를 제공하는 L4,L7 스위치

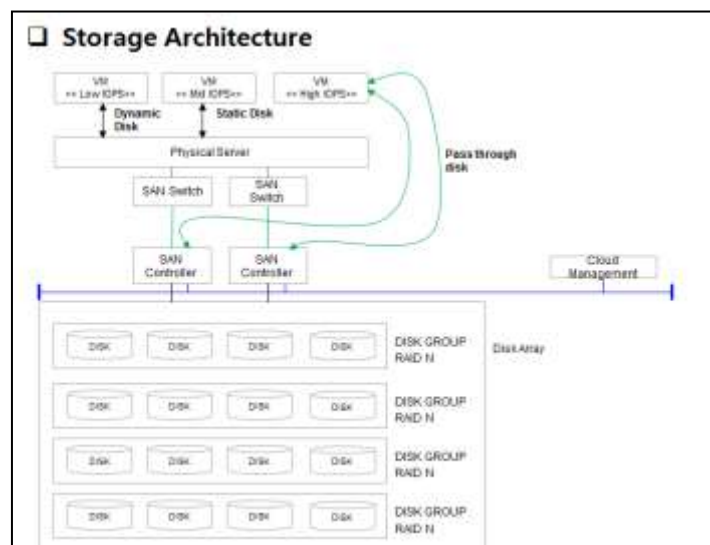
- 보안을 위한 침입 탐지 시스템 IPS와 방화벽
- 내부 IP를 사용하기 위한 NAT (Network Address Translation) 장비
- 그리고 LAN (Local Area Network)에 대한 구성등을 정의한다.



## 스토리지 디자인 아키텍처

외부 스토리지 (디스크)에 대한 아키텍처를 정의한다.

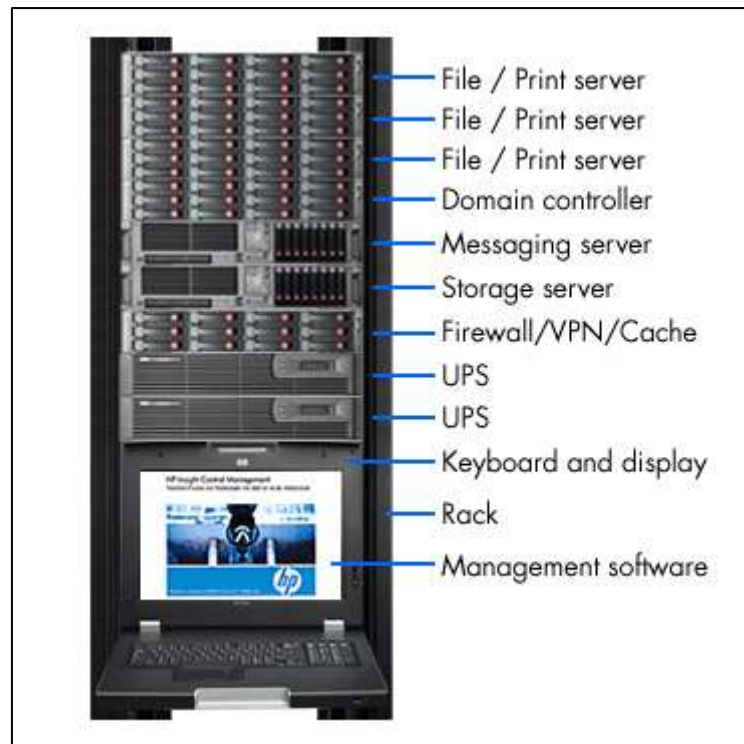
- NFS (Network File System)과 같은 공유 파일 시스템, SAN (Storage Area Network)과 같은 스토리지 타입을 정의하고
- 스토리지를 관리하는 Controller, 어떤 물리적인 디스크를 사용할것인지 (SAS,SATA, DISK 회전속도)에 대한 디자인과 RAID 구성등을 정의한다.
- 그리고 스토리지를 연결하는 네트워크 구성등에 대해서 정의한다.



## 랙 디자인 아키텍처

시스템의 규모가 RACK (서버를 집적해서 데이터 센터에 배치 시키는 일종의 서버용 케이스)에 서버를 배치하는 구성을 디자인 한다. 랙의 위치에 따른 케이블링의 위치, 랙의 전력 용량, 랙을 컨트롤 하기 위한 KVM (키보드와 화면이 달려있는 콘솔로, 여러대의 서버를 스위칭하며, 서버를 컨트롤 할 수 있다.), 필요에 따라 UPS(무정전 전원 공급 장치)등의 설계를 포함해서 고려한다.

특히 대규모 숫자의 랙을 설계할 경우에는 서버들의 배치는 랙의 전력 용량과 함께, 발열량을 고려하여 장비를 배치해야 한다.



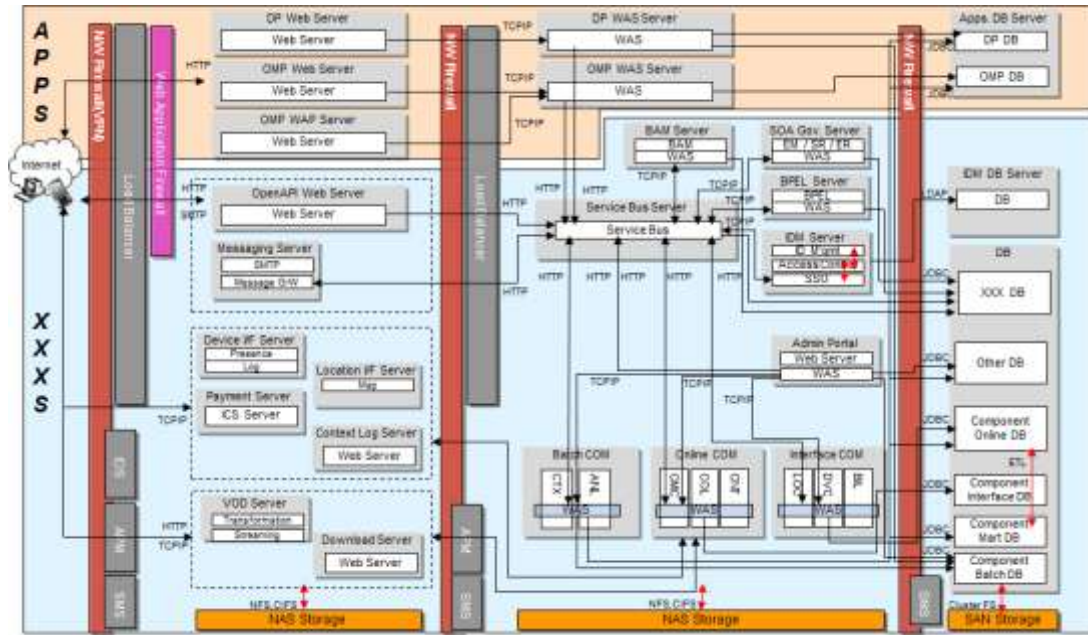
[ HP RACK 디자인 ]

출처 : <http://www.hp.com/sbso/productivity/howto/rack-server/plan-it.html>

## Solution deployment Architecture

솔루션은 애플리케이션이 동작하기 위한 미들웨어를 정의하며, Solution deployment Architecture 는 이에 대한 배포 구조를 정의한다.

솔루션 아키텍처는 각 솔루션의 특성을 많이 반영된다. 특히 클러스터링 기능이나 HA와 같은 이중화 기능을 제공하는 경우에는 조금 더 많은 특성을 요구로 한다. 클러스터링을 위한 전용 네트워크 인터페이스나, HA의 경우 상태 정보를 공유하기 위한 공유 디스크를 사용하는 것 RDBMS의 경우 트랜잭션 로그를 빠른 디스크에 분리해서 배치 하는 등의 케이스가 있다. 이렇게 솔루션들은 각 솔루션 자체의 배포 구조에 대한 것들에 대한 설계 뿐만 아니라 하드웨어에 대한 요구 사항이나 구성들에 대한 권고 사항을 가지고 있기 때문에 solution deployment architecture 는 solution의 배포 구조뿐만 아니라, 하드웨어 설계 아키텍처에까지 영향을 준다.



### (3) 데이터 아키텍처

데이터 아키텍처는 시스템에서 저장하고 다루는 정보에 대한 정의와 관리 구조에 대한 아키텍처이다. 데이터 아키텍처에서 표현해야 하는 주요 내용들은 다음과 같다.

#### 데이터 모델링

데이터 모델링은 시스템에서 다루는 정보를 구성하는 개체와 그 개체들의 관계를 정의한다. 쉽게 정의하면 DB 설계에 사용하는 ERD (Entity Relationship Diagram)을 생각하면 가장 명확하다. 이 단계에서는 각 Entity를 판별하고, 각 Entity에 대한 관계를 정의한다.

데이터 모델링은 두가지 단계로 이루어져야 한다.

#### Conceptual Modeling

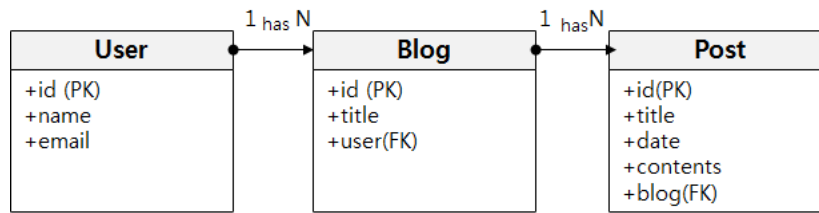
시스템을 이루는 정보를 표현하는 개체와, 개체간의 관계와 multiplicity 정의한다. 여기에는 개체들이 가지고 있는 구체적인 속성 (Attribute)이나 Key 등은 정의하지 않는다.



#### Logical Modeling

Conceptual Modeling에서 추가하여, 각 Entity에 Attribute를 표현하고, 아울러 Primary Key와 ForeignKey를 정의한다.





## Implementation Model

Logical Model을 물리적인 테이블로 설계한다. 각 Attribute에 대해서 Table Column으로 맵핑하고, Index 정의와, 각 Column에 대한 데이터 타입을 정의한다. Data Architecture 설계에서는 Implementation Model까지 설계하지 않고, Logical Model까지만 설계한다. Implementation Model은 상세 설계 단계에서 수행한다. Implementation Model은 구현에 사용될 솔루션의 특성에 맞게 데이터를 디자인하며, 솔루션에 추가 기능 (Stored Procedure), XML Document 지원 등의 기능을 반영하여 설계한다.



TABLE : USER

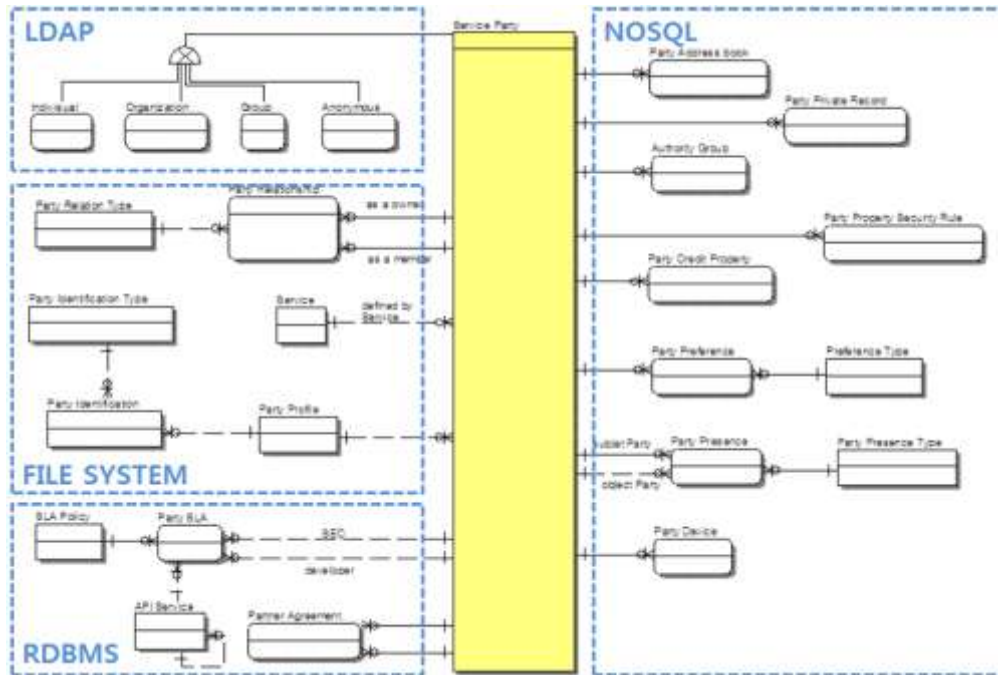
column	data type	Index
id	VARCHAR(255)	PK
name	VARCHAR(255)	
email	VARCHAR(255)	

## 데이터 저장소

Data Architecture에서 정의하는 데이터는 RDBMS에 들어가는 데이터 뿐만 아니라, 파일, 메타 정보 등의 모든 형태의 데이터 아키텍처를 정의한다. 그래서 RDBMS에 대한 설계뿐만 아니라, 데이터의 특성에 맞는 데이터 저장소를 선택해야 한다.

대표적인 데이터 저장소 타입으로는 LDAP, RDBMS, FILE SYSTEM, NO SQL, ISAM 있으며, 데이터 모델의 Logical Model에서 정의한 개체들을 각 저장소 타입에 맞게 재 배치해야 한다. 어디에 데이터를 저장할지를 결정한 후에, 각 데이터 저장소 특성에 맞게 상세 설계 단계에서 Implementation Model을 설계한다.





## 애플리케이션 컴포넌트와 맵핑

데이터의 종류와 각 속성이 파악되고, 저장소가 결정 되었으면, 각 데이터를 사용하는 애플리케이션 컴포넌트와의 관계를 정의한다. 되도록이면, 각 데이터 개체가 단일 애플리케이션에 의해서 다뤄지도록 설계를 하는 것이 좋다.

동일한 데이터 개체를 여러개의 애플리케이션에서 동시 업데이트를 할때는 일관성의 문제가 발생할 수 있다. 트랜잭션 보장 능력이 있는 RDBMS의 경우에는 큰 문제가 없을 수 있겠으나, 파일 시스템이나 NOSQL과 같이 트랜잭션에 대한 개념이 없는 저장소의 경우 데이터 불 일치를 유발할 수 있다.

그리고 가능하다면, 하나의 기능을 구현하는데, 데이터가 같은 저장소에 있는 것이 유리하다. 예를 들어 사용자 정보를 LDAP과 RDBMS에 나눠서 저장한다면, "사용자 생성"과 같은 기능은 하나의 트랜잭션에서 LDAP과 RDBMS 양쪽을 동시에 업데이트 해야 하며, 두 개의 데이터 저장소를 하나의 트랜잭션으로 묶는 분산 트랜잭션 (Distributed Transaction)의 개념으로 구현을 해야 한다. 이러한 저장소들이 분산 트랜잭션 표준 (XA)등을 지원한다면 일관성 보장이 다소 쉽겠지만, NoSQL과 같이 분산 트랜잭션을 지원하지 않는 시스템이 트랜잭션에 참여하게 되면, 애플리케이션 계층에서 트랜잭션에 대한 보장 로직을 별도로 구현해야 하기 때문이다.

## 데이터 관리 프로세스

어떤 데이터를 어디에 저장하고, 누가 사용할지가 정해졌으면, 이 데이터에 대한 관리 정책을 정해야 한다.

**Security** 데이터에 대한 암호화, 데이터 베이스 시스템으로의 접근 통제를 위한 인증과 인가에 대한 정책을 정의한다.

**Life cycle** 데이터의 생성에서 부터 폐기까지의 데이터 생명 주기를 정의한다. 특히 이 영역에서는 데이터에 대한 백업 정책등이 정의되어야 한다.

**Sharing & Integration** 애플리케이션 시스템 간의 데이터를 공유하거나, 서로 통합을 위해서 데이터를 전송하기 위한 아키텍처를 정의한다. EAI나 Data Integration 관점에 대한 부분이나 MDM (Master Data Management) 부분이 이 영역에 해당한다.

**Analysis & reporting** 시스템에 축적된 데이터를 기반으로 분석과 리포팅 서비스를 제공할 수 있는 아키텍처를 정의한다. OLAP이나 BI같은 데이터 분석 리포팅이 이 영역에 해당한다.

#### 4. Architecture Decision Process

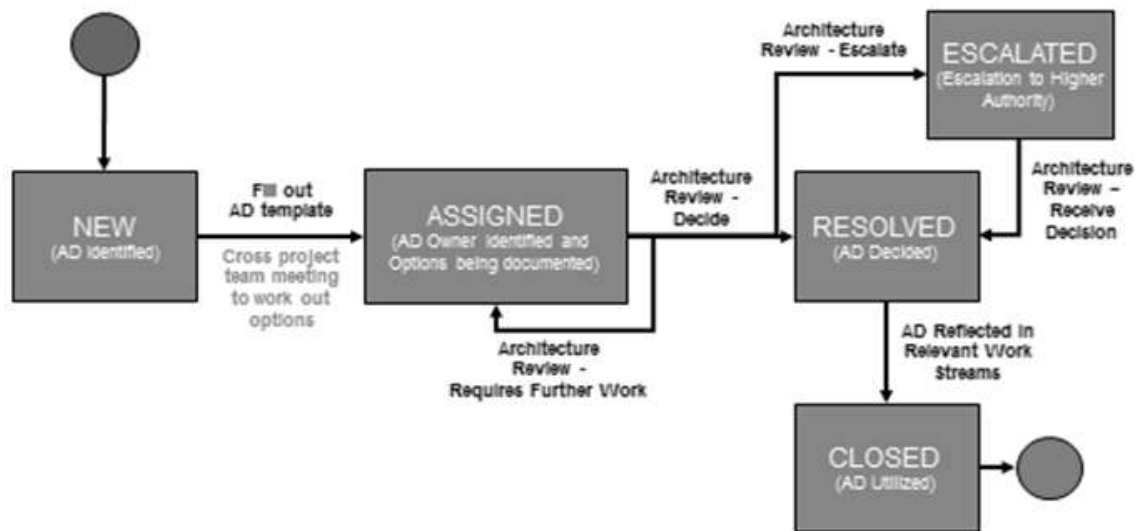
Architecture Decision Process (이하 AD)는 아키텍처 설계 과정에서, 문제를 해결하기 위해서 다양한 아키텍처 옵션이 존재 하는 경우 아키텍처에 따라서 비즈니스에 영향을 주는 일정, 기간, 비용에 영향을 주거나, 구현에서 기술적인 이유나 일정, 조직의 기술 보유 능력 또는 회사의 전략에 따라서 아키텍처의 선택이 필요할 경우, 아키텍처 옵션에 대한 의사결정을 하는 프로세스 이다.

일반적으로 아키텍처는 해당 분야의 담당 아키텍트가 결정하는 구조를 가지지만, 그에 대한 파급 효과가 클 경우 AD Process를 사용하는데, AD Process는 아키텍트의 단독적인 결정이 아니라 아키텍처 의사 결정을 위한 Architecture Committee 라는 조직을 운영하여, 이 Committee에서 의사 결정을 한다. Committee는 아키텍트등의 기술자로만 이루어진 것이 아니라 필요에 따라 경영진, 프로젝트 관리자(PM)나 관리 조직(PMO), 실무 개발 PM 들로 구성된다.

AD Process를 사용하는 이유는 전체 조직적인 관점 (기술,비즈니스,프로젝트 관리)에서 특정 아키텍처의 **장단점을 기반으로** 아키텍처를 결정하여 향후 발생 가능한 위험을 최소화 하고 위험 관리 대책을 마련하기 위함이다.

#### AD Process 정의

AD Process는 일반적으로 다음과 같은 절차를 따른다





- ① [NEW] 실무팀 (개발,영업,QA,운영)이 요건 만족을 위해서 다른팀과 회의를 통해서 요건 변경의 필요성이 있음을 파악하고, 변경 방법 옵션을 정의하고 각각의 장단점을 AD 템플릿 포맷에 맞게 작성 한다.
- ② [ASSIGNED] 아키텍처팀이 해당 AD를 검토후 판단하여, 결정이 가능한 수준이면 결정을 한후, 실무팀에 결정 사항을 통보한다.
- ③ [ESCALATED] 아키텍처팀의 결정에 실무팀이 수락이 안되거나 또는 해당 결정 사항이 시스템 개발이나 비즈니스에 지대한 영향이 있다고 판단되었을 때는 *Architecture Committee\** 로 해당 AD를 escalation한다.
- ④ [Resolved] AD에 대해서 하나의 옵션으로 결정이 난후에, 아키텍처적인 파급효과등을 파악하여 검토한 후, 실무팀에 결정 사항을 통보한다.

## AD Template

AD Process를 진행하기 위해서는 AD이 필요한 아키텍처 Option에 대한 설명과 장단점, 그리고 AD가 필요한 이유를 서술해야 한다. 이는 문서로 작성되어 Commiitee등에서 의사 결정의 기본 자료로 활용되며, Commiitee는 기술 인력 뿐만 아니라 비즈니스와 관리인력이 참여하는 만큼 기술 인력 이외의 사람들도 이해할 수 있는 수준으로 작성되어야 하며, 이 모든 내용을 간략하게 1장에 작성해야 한다. 상세한 설명이 필요한 경우에는 별첨 문서를 따로 작성하여 첨부하도록 한다.

아래는 이러한 내용을 담은 AD Template 예제이다.

## AD I. Architecture Decision Item Title

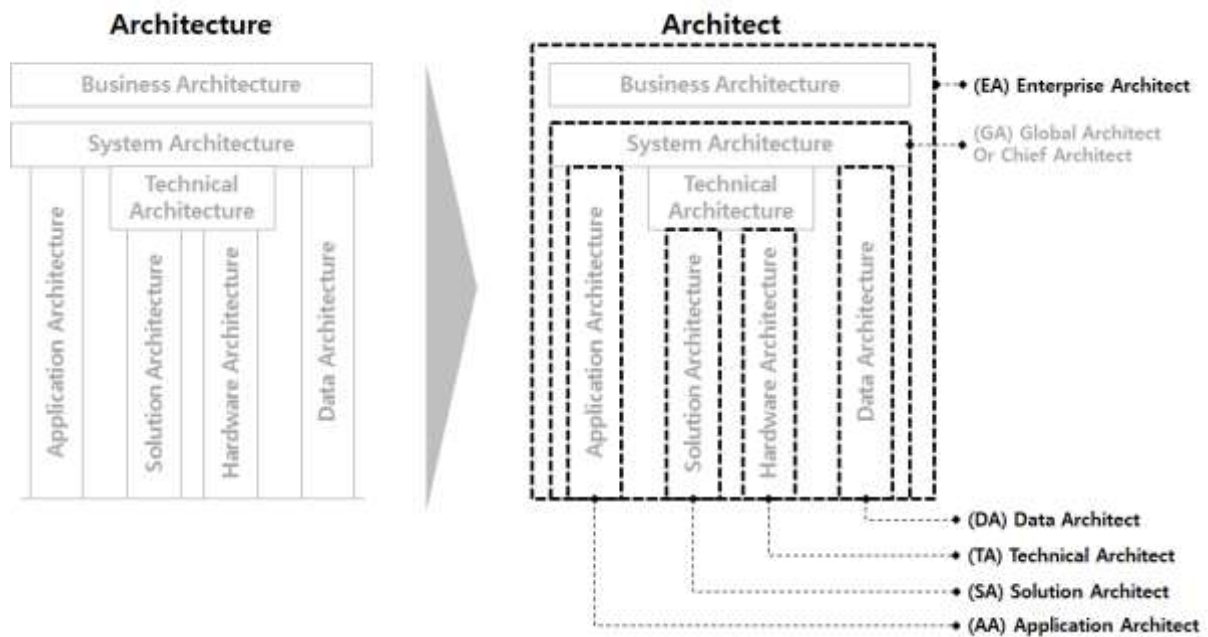
<p><b>Option A. : {title}</b></p>  <p>OPTION A에 대한 아키텍처 구조도</p> <div style="border: 1px solid black; padding: 5px; text-align: center;">아키텍처 구조 설명</div>	<p><b>Description &amp; Motivation</b></p> <p><b>Assumption</b></p> <p><b>Consideration</b></p> <p><b>Option A</b> Pros : Cons :</p> <p><b>Option B</b> Pros : Cons :</p> <p><b>Decision</b></p> <p><b>Implication</b></p>
<p><b>Option B. : {title}</b></p>  <p>OPTION B에 대한 아키텍처 구조도</p> <div style="border: 1px solid black; padding: 5px; text-align: center;">아키텍처 구조 설명</div>	

AD Template에 들어가는 각 항목에 대한 내용은 다음과 같다.

- **Description & Motivation** 해당 문제에 대한 간략한 정의와 의사 결정이 필요한 이유
- **Assumption** 본 아키텍처에 대한 가정 사항
- **Consideration** 본 아키텍처에 대한 주요 고려 사항
- **Option별 장단점** 옵션별 장단점을 기술 및 비즈니스 관점에서 정의
- **Decision** 아키텍처 옵션에 대한 선택이 종료된 경우, 어느 옵션으로 선택했는지를 명시
- **Implication** 본 의사 결정으로 미치는 영향 (기술 및 비즈니스-기간 연장, 솔루션 구매, 매출 감소 등)

## 아키텍트의 종류와 역할

이러한 아키텍처를 설계 하는 사람은 아키텍트(Architect)라고 한다. 이 아키텍트는 아키텍처 설계 프로세스에서 정의한 각 아키텍처에 대해서 아키텍처를 설계하는 역할들이 정의된다. 계층 구조를 제외하면 아키텍처는 5가지로 분리된다. Business Architecture, Application Architecture, Solution Architecture, Data Architecture로 분리되며, 아키텍트 역시 이 5개 분야에 걸쳐서 총 5개의 역할로 정의된다.



**Enterprise Architect (EA)** Business Architecture를 포함한 전체 아키텍처 설계에 대한 책임을 진다. 비즈니스 이해를 바탕으로 전체 아키텍처에 대한 큰 설계를 담당하며, 비즈니스에 대한 이해를 바탕으로 장기적인 IT 전략 수립을 담당한다. EA의 특징중의 하나는, EA의 경우 단일 프로젝트 뿐만 아니라, 해당 회사의 앞으로의 비즈니스 전략에 맞춰서 향후 모든 프로젝트에 대한 아키텍처에 대한 책임을 진다. 또한 SA,AA,TA,DA에 대한 통제 권한을 가지고 아키텍처 팀을 운용한다. 가끔 CIO와 혼동하는 경우가 있는데, CIO는 회사 내부의 IT 전략을 수립하고, IT 포트폴리오를 정의하고 수행한다는 관점에서 EA와 유사한 면이 있으나, 기술적인 면에서는 CIO와 EA의 부분이 겹칠 수 있으나 경영적인 측면에서는 다르다. CIO는 결정적으로 예산 집행권과 인사권을 가지고 있으며 설계 보다는 경영과 관리에 목적을 두는 반면, EA는 아키텍처 설계를 그 목적으로 한다.

**Solution Architect (SA)** 특정 솔루션에 대한 아키텍처를 설계한다. SA의 경우 프로젝트 내에 개발팀이 있을때, 해당 솔루션을 사용하는 모든 팀에 대한 아키텍처 설계를 담당한다.

**Technical Architect (TA)** 프로젝트 전체팀에 대한 하드웨어 및 네트워크 아키텍처를 설계한다.

**Application Architect (AA)** 애플리케이션에 대한 표준 가이드 및 아키텍처 구조를 담당한다. 팀 규모에 따라 대규모 팀인 경우 각 개발팀마다 AA를 배치하고, 소규모팀인 경우에는 프로젝트 전체팀에 대해서 애플리케이션 아키텍처 설계를 담당한다.

**Data Architect (DA)** 프로젝트 전체팀에 대해서 데이터 아키텍처 설계를 담당한다.

**Global Architect (GA) [Optional]** 일반적인 프로젝트 팀 구조에서는 잘 존재하지 않는 역할이다. EA의 경우 사내에서 진행중인 모든 프로젝트에 대해서 관여해야 하고, 비즈니스 전략 측면에서 접근을 하다보니 경영진과의 커뮤니케이션이나 의사 결정 과정에 참여가 많아지기 때문에 실제 아키텍처 설계 과정에 디테일하게 참여하기가 어렵고, 때로는 기술의 이해수준이 아주 디테일 하지 않은 경우가 있기 때문에, GA라는 역할을 뒤서 SA,TA,DA,AA에 대한 통제 권한 부여하고 기술 중심의 System Architecture의 설계 하도록 한다. 프로젝트 팀의 PM/PL의 관계를 EA/GA 관계로

보면 된다. EA는 비즈니스를 포함한 외부 대응이나 큰 그림에 신경 쓰고, GA는 기술 위주의 아키텍처 설계에 집중한다.

---

i

ii

iii