정보통신산업진흥원 부설
SW공학센터

SW아키텍처 참조모델

[클라우드 시스템을 기반으로한 통합 로그/데이터 수집 참조모델]

[Version 1.0 20121230]

SW공학센터

SW공학기술팀

SW아키텍처 실무자 포럼 클라우드 분과

SEC-2012-RM003

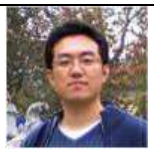
본 문서는 국내 기업의 소프트웨어 품질 및 생산성 향상을 지원하기 위하여 작성되었습니다.

본 문서는
지식경제부 산하
정보통신산업진흥원 부설
SW공학센터
SW아키텍처 실무자 포럼
에서 작성되었습니다.

SW공학센터는 SW제품 생산능력 향상, SW공학기술 산업현장 적용 등을 위해 대학 및 전문 연구기관과 기업 현장을 연결하는 중심 허브가 되어 SW개발 중소기업들에게 전문 컨설팅을 제공하고 있습니다. 이 같은 역할을 충실히 수행하기 위해 산업과 기업의 SW공학기술 관련 요구사항에 전문적이고 신속한 대응을 할 수 있는 핵심 기능 연구와 SW개발 프로젝트의 성공 여부와 문제점을 미리 예측하여 SW품질과 생산성, 제품결함 등을 총체적으로 진단 할 수 있는 SW공학 컨설팅 등도 추진하고 있습니다. 이와 더불어 SW품질과 생산성, 비용 등을 체계적으로 추적 평가할 수 있는 데이터 수집 체계도 강화하여 SW기업들이 이를 손쉽게 활용하게 함으로써 전체적인 국가 SW품질을 향상시키는 업무도 수행중입니다.

본 지침에 대한 더 많은 정보와 SW공학에 대한 추가 정보를 얻고 싶다면, SW공학센터 홈페이지(www.software.kr)를 방문하여 주십시오.

담당자 강승준 책임 [ksj@nipa.kr, 02-2132-1344]

프로필	이름	약력
	김 정	오로라플래닛 대표이사 한컴 UMS, KT SoIP 서비스 플랫폼 TA NHN NEXT 모바일전공 iOS 교수 OSXDEV 맥/iOS 개발자 커뮤니티 운영진 godrm77@gmail.com
	김 성 조	LG CNS에서 회사생활을 시작 Software Architect로 Java프로젝트 지원과 Framework을 개발 Performance Engineer로 연간 20여개의 주요 사이트 문제 해결 제니퍼소프트에서 성능모니터링 솔루션 개발 현재 LG CNS에서 클라우드컴퓨팅 환경을 위한 관리플랫폼 연구 sjokim@gmail.com
	고 준 일	비니아 대표 호서대학교 게임학과 겸임교수 삼성SDS 멀티캠퍼스 강사 goya320@gmail.com
	장 선 진	소프트웨어인라이프 대표이사 소프트웨어 마에스트로 멘토 한중일 공개소프트웨어 포럼 정회원 Lisp 한국 사용자 모임 설립자 및 회장 jangsunjin@gmail.com
	장 회 수	프리랜서, 여전히 선택과 집중에 대해 미숙한 12년차 자바 개발자. 소프트웨어 아키텍처와 통합/배포, ALM, 개발 프로세스에 관심 JBoss User Group 과 OSD(Optimal Software Dev.) , RSM(Real Software Modeling) 커뮤니티 활동 architectj@gmail.com
	한 상 욱	삼성SDS에서 Software Architect로 근무했던 Server Side Architecture와 Cloud Service 및 Big Data 전문가 현재 Splunk Korea에서 Senior Sales Engineer로 활동중. powerhan96@naver.com

〈목 차〉

<표 목차>

<그림 목차>

1. 참조 모델 개요

클라우드 컴퓨팅 분야는 구글, 아마존과 같은 PaaS, IaaS 서비스를 시작으로 SaaS, BaaS 등 다양한 형태로 진화하고 있다. 대규모의 플랫폼이나 인프라를 구성하고 확장하는 클라우드 아키텍처에서 클라우드를 구성하는 노드 수가 많아지고, 관리 포인트도 비례해서 늘어나면서 전체를 안정적으로 운영하기 위한 요구사항도 기하급수적으로 늘어나고 있다.

시스템에서 발생하는 이벤트 또는 상태 정보를 기록하고 문제를 추적하거나 진단하기 위한 절차를 로깅(logging)이라고 하고, 이런 로그 정보를 기반으로 사용자가 시스템의 상황을 인식하는 행위를 모니터링이라고 한다. 로깅 및 모니터링 솔루션은 그 종류도 다양하고 상용 뿐만 아니라 오픈소스 형태로 제공되는 제품도 많이 있지만, 자사의 시스템에 딱 맞는 솔루션의 선정 및 적용은 여전히 큰 과제로 남아 있다. 또한 클라우드 환경의 출현으로 분산 시스템이 일반화 되는 상황에서 기존의 로깅 및 모니터링 방식을 그대로 적용하는 것은 부족하기 때문에 로깅 및 모니터링에 대한 큰 그림을 이해하는 것이 중요하다. 점점 애플리케이션 데이터와 가상화 노드를 포함한 HW 장비 로그, 고객 관리까지 통합 운영을 위해서 기존의 오픈소스 기반 빅데이터 솔루션은 너무 무겁고 방대해서 레거시 시스템이 많은 기업에서 도입하기 망설여진다.

가. 클라우드 컴퓨팅의 정의

클라우드 컴퓨팅(Cloud Computing)은 인터넷을 통해 서버, 스토리지, SW 등 ICT 자원을 필요 시인터넷을 통해 서비스 형태로 이용하는 소프트웨어 유통 방식이다.

클라우드 컴퓨팅(Cloud Computing)이란 용어의 기원은 2006년 구글의 직원인 크리스토프 비시글리어(Christophe Bisciglia)가 유휴 컴퓨팅 자원에 대한 활용을 제안하면서 처음 사용된 이래 다양한 개념으로 사용했다.

클라우드 컴퓨팅의 활용 방안이 높아 정의는 다양하나, 공통적으로 이용자가 필요로 하는 IT 자원을 필요한 만큼 빌려쓰는 개념으로 인터넷을 통해 가상화된 형태로 제공받을 수 있는 것을 의미를 내포하고 있다.

표 1. 기관별 클라우드 정의

기관명	정의
NIST	이용자는 IT자원(소프트웨어, 스토리지, 서버, 네트워크)을 필요한 만큼 빌려서 사용하고, 서비스 부하에 따라서 실시간 확장성을 지원받으

	며, 사용한 만큼 비용을 지불하는 컴퓨팅
Gartner	인터넷 기술을 활용해 많은 고객들에게 수준 높은 확장성을 가진 자원들을 서비스로 제공하는 컴퓨팅의 한 형태
Forrester Research	표준화된 IT 기반 기능들이 IP로 제공되고, 언제나 접근이 허용되며, 수요 변화에 따라 가변적이며, 사용량이나 광고를 기반으로 비용을 지불하고 웹 또는 프로그램적인 인터페이스를 제공하는 형태
IBM	웹 기반 응용 소프트웨어를 활용해 대용량 데이터베이스를 인터넷 가상공간에서 분산처리하고, 이 데이터를 컴퓨터나 휴대전화, PDA 등 다양한 단말기에서 불러오거나 가공할 수 있게 하는 환경
TTA	가상화와 분산처리 기술을 기반으로 인터넷을 통해 대규모 IT자원을 임대하고, 사용한 만큼의 요금을 지불하는 컴퓨팅 환경을 의미

클라우드 컴퓨팅의 주요한 특징은 기존의 컴퓨팅 서비스에 있어 자원을 이용자가 직접 소유·관리하는 방식과 달리 이용자가 필요한 만큼의 자원을 가상화된 형태로 인터넷을 통해 제공받는 방식의 서비스로 소유와 관리가 분리된 방식이다.

나. 클라우드 컴퓨팅 기술

개인용 컴퓨터(PC)가 보급되기 시작한 90년대 초반의 컴퓨터는 문서 편집기나 게임, 또는 그래픽 작업과 같은 윅스테이션 형태를 취하거나 기업에서는 재고 관리, 문서 관리와 같은 용도로 사용되고 있었다. 90년대 말 광역 네트워크 기술의 발전과 인터넷의 보급은 기존 컴퓨터에 날개를 달아 주었다. 단순 문서 편집기가 협업 시스템으로 발전하고, 사람이 직접 이동하거나 우편으로 처리하던 업무들이 온라인화 되었으며, 마케팅 방식에도 변화가 생기고 그 효과가 입증되면서 기업들은 너도나도 전산 시스템을 도입하기 시작했다.

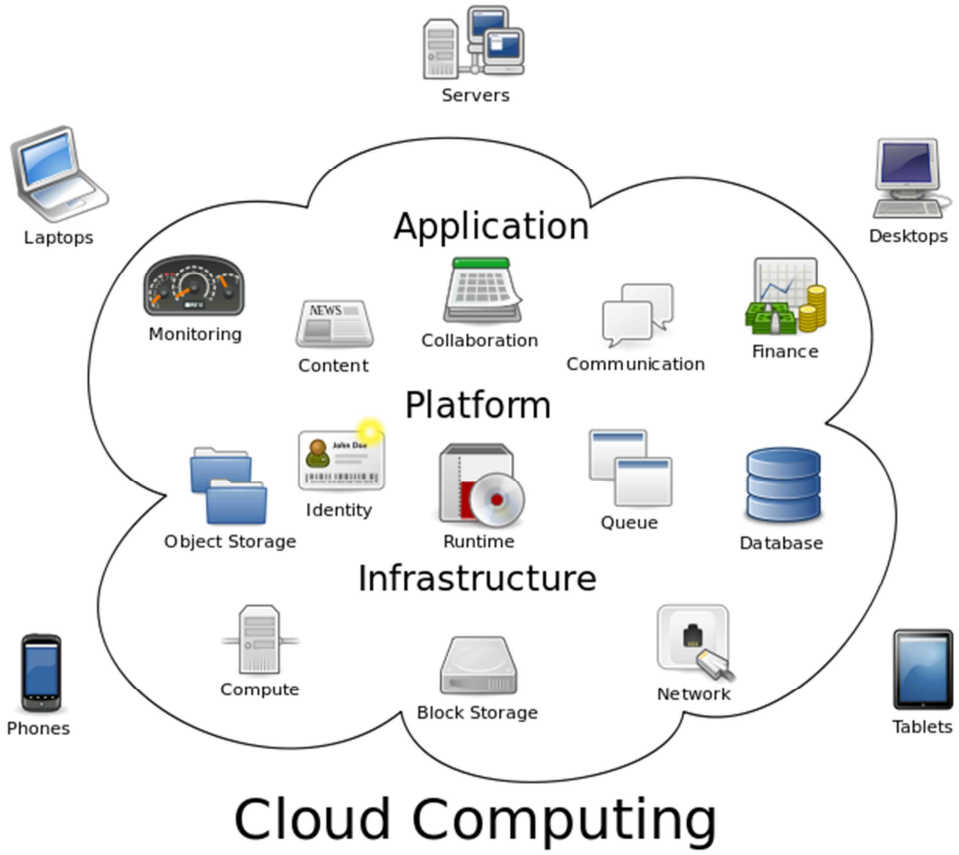


그림 1. 클라우드 컴퓨팅 개념

큰 기업들은 자사의 전산 시스템을 한데 모아 대형 전산실을 운영할 수 있었지만, 중소 규모의 기업은 비용의 문제로 자체 전산실 구축이 어려웠다. 특히 닷컴 바람과 함께 일기 시작한 벤처 붐은 전산 자원의 효율적 유지에 대한 욕구에 부채질을 해주었다. 이시기 데이터 센터라는 새로운 IT사업 모델이 나오면서 많은 기업들이 상대적으로 저렴한 비용으로 편리하게 자사의 전산 시스템을 운영할 수 있는 기반이 됐다. 1999년 데이콤의 IDC(Internet Data Center)를 시작으로 많은 통신 회사들이 데이터 센터 건립에 뛰어 들었으며, 지금은 새로운 서비스를 개발 운영하기 위해 IDC를 이용하는 것은 당연한 것이 되었다.

'클라우드' 역시 이와 유사한 변모의 과정의 일부라고 할 수 있다. 2000년대 초반에는 온라인 쇼핑몰을 시작으로 B2C형 e-Commerce 시장이 활성화됐고, 중반 이후에는 스마트폰과 무선 통신의 발전으로 '개인 단발(Personal Device)'과 모바일 서비스, 그리고 개인 컨텐츠가 급속히 증가하였다. 결국 이와 같은 IT 트랜드를 받혀줄 수 있는 어떤 새로운 개념의 인프라 출현이 어쩌면 당연한 것일지 모른다. 바

로 클라우드 말이다.

우리가 흔히 부르는 클라우드라는 용어는 '클라우드 컴퓨팅(Cloud Computing)'을 줄여서 부르는 말로 IDC가 그랬던 것처럼 특정 기술을 지칭하거나, 특정 서비스를 지칭하는 것이 아닌 수많은 IT관련 기술과 시스템 사용 형태를 통칭하여 부르는 추상화 된 표현이다. 클라우드 컴퓨팅을 정의하면 “하드웨어나 소프트웨어와 같은 모든 컴퓨팅 자원을 네트워크를 경유하는 서비스의 형태로 제공하는 것으로 복잡한 시스템 구조를 간단한 구름 기호로 추상화하여 표현하는 것”이라고 할 수 있다.

인터넷이 IT트랜드의 중심이었던 시절의 데이터 센터는 IDC였다. 클라우드가 IT트랜드의 중심인 지금의 데이터 센터는 그래서 CDC 즉, Cloud Data Center라고 부른다. IDC가 IT산업에서 당연한 것이 됐던 것처럼, 이제는 클라우드가 당연한 것이 되어가고 있다.

다. 클라우드 개념

앞에서 언급한 클라우드의 정의에서 알 수 있듯이 클라우드는 데이터 센터의 자원들 즉, 인프라에서 출발했다. 하지만, 단지 인프라 뿐만이 아니라 그 위에 올라갈 공통 플랫폼이나 애플리케이션까지도 클라우드의 범주에 포함되기도 한다. 그 정의가 다소 모호한 면이 없지 않지만 클라우드의 가장 큰 특징은 바로 모든 것을 '서비스'의 형태(XaaS, Everything as a Service)로 인식한다는 점이다. 최근의 시스템들은 그 규모가 거대해지고 각 모듈 또는 컴포넌트 간의 의존 관계가 복잡해지고 특히 외부 시스템과의 연동이 비일비재 하게 이루어지다 보니, 모든 시스템의 복잡한 인터페이스 구조를 이해하고 연동하는 것이 불편해졌다. 그래서 마치 전화가 어떻게 작동이 되는지 모르더라도 전화를 걸고 통화한 시간만큼의 가격만 지불하면 되듯이 IT 자원을 사용할 수 있으면 편할 것이다. 클라우드에서는 서버나 스토리지도 이와 같이 서비스 형태로 제공 받는다. 별도의 메일 서버를 구축하는 대신 클라우드 업체에서 제공하는 메일 서비스를 간편한 방식으로 내 시스템과 연동하여 전송량에 따라 비용을 지불할 수 있다.

즉 클라우드는 모든 IT 자원을 '서비스' 형태로 사용할 수 있는 특징이 있다.

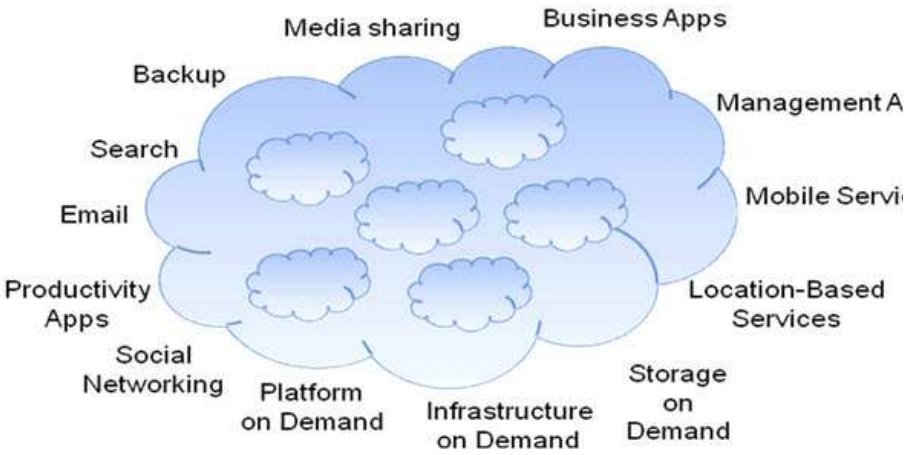


그림 2. 클라우드 기반 서비스

뒤에 나올 클라우드 분류에서는 XaaS 형태로 표현되는 여러가지 클라우드가 나온다. 지금까지 얘기한 인프라를 서비스로 제공하는 것 외에, 네트워크 도메인만 묶어 NaaS (Network as a Service)로 제공할 수도 있고 복잡한 백엔드(Backend) 서비스를 추상화하여 서비스로 제공하는 BaaS (Backend as a Service)도 있다. 그 외에도 데스크탑 환경을 사용할 수 있는 DaaS (Desktop as a Service), 스토리지를 서비스로 제공하는 STaaS (Storage as a Service) 등도 있다.

미국 국립표준기술연구소(NIST, National Institute of Standards and Technology)에서는 클라우드를 다음과 같이 5가지로 정의하고 있다.

1. 수요 기반의 셀프 서비스 (On-demand self-service) : 사용자는 자기의 필요에 따라 서버와 네트워크 스토리지와 같은 컴퓨팅 능력을 프로비저닝 할 수 있고 또 사람의 손을 빌지 않고 자동으로 제공하는 것이 가능하다.
2. 광범위한 네트워크 접근성 (Broad network access) : 클라우드의 능력은 네트워크를 통해 제공되고 표준화된 메커니즘으로 모바일 폰, 태블릿, 노트북, 웍스테이션과 같은 다양한 종류의 씬 클라이언트, 팻 클라이언트 플랫폼을 지원한다.
3. 리소스 풀링 (Resource pooling) : 클라우드 서비스 사업자는 멀티 터넌트(Multi-tenant) 모델을 기반으로 다수의 사용자에게 컴퓨팅 자원을 제공하며 이는 사용자의 요청에 따라 다양한 물리, 가상 자원을 동적으로 할당하거나 재배치 할 수 있다. 자원에는 스토리지, 프로세싱, 메모리, 네트워크 대역, 가상 머신을 비롯하여 이메일 서비스 등도 포함된다. 여러가지 자원을 선택적으로 조합하여 규모의 경제를 형성한다.
4. 신속한 탄력성 (Rapid elasticity) : 클라우드 서비스는 신속하고 탄력적으로, 때로는 자동으로 자원을 공급할 수 있어 신속하게 자원을 확장 또는 축소할 수 있다. 사용자 측면에서는 자원의 공급이 거의 무한대에 가깝게 보이며 그 양에 관계없이 언제든지 구매가 가능하다.

5. 계측형 서비스 (Measured service) : 클라우드 컴퓨팅에 사용되는 모든 자원은 계측과 통제가 가능하고 사용한 자원에 대해 공급자와 소비자 간의 투명한 보고가 가능하다. 클라우드 컴퓨팅이 제공하는 미터링(metering) 기능으로 자원을 제어하고 최적화 할 수 있다. 이는 사용한 만큼 지불하는 종량제를 의미한다. 더 많이 사용할 수록 더 많은 비용을 지불하는 것이다. 전기 회사가 전력 공급하고, 통신 회사가 음성 서비스, 데이터 서비스를 제공하는 것처럼, 네트워크 보안 서비스나 데이터 센터의 호스팅 서비스에서 각 부서의 과금까지도 사용량을 기준으로 하는 종량제 서비스가 가능하다.

이 외에도 클라우드에는 다양한 특징이 있다. 민첩성(Agility), API 중심, 확장성, 보안성, 유지 보수성, 안정성, 멀티터넌시(다중 사용자 특성), 가상화 특성, 비용 절감 특성 등이 그것이다.

지금은 너무나 당연한 것이 되어 버린 전기도 원래는 개인이나 기업이 직접 수차나 발전기를 돌려 수급하고 있었다. 구러다 표준화의 필요성과, 전기를 사용하는 기기간의 호환성, 공정하고 합리적인 가격 등의 이유로 전기를 범 국가 차원에서 기간 산업으로 지정하여 대형 발전소를 짓고 각 소비자에게 제공한 후 사용한 만큼 지불하게 만든 것이 지금의 전력 체계다. 클라우드도 이와 유사한 길을 걷고 있다.

라. 클라우드 컴퓨팅 분류

앞에서 클라우드는 모든 것을 서비스로 바라보는 특징이 있고 했다. 그래서 다양한 IT자원의 속성에 따라 클라우드 서비스도 다양하게 분류될 수 있는데, IT자원의 가장 하위부터 살펴보면 데이터 센터에 들어가는 서버와 스토리지 그리고 네트워크 등의 하드웨어가 있을 수 있다. 이런 것들을 '인프라'라 하고 그래서 클라우드 서비스에는 IaaS (Infrastructure as a Service)라고 하는 하나의 분류가 있다.

1) 클라우드 인프라 서비스 (IaaS, Infrastructure as a Service)

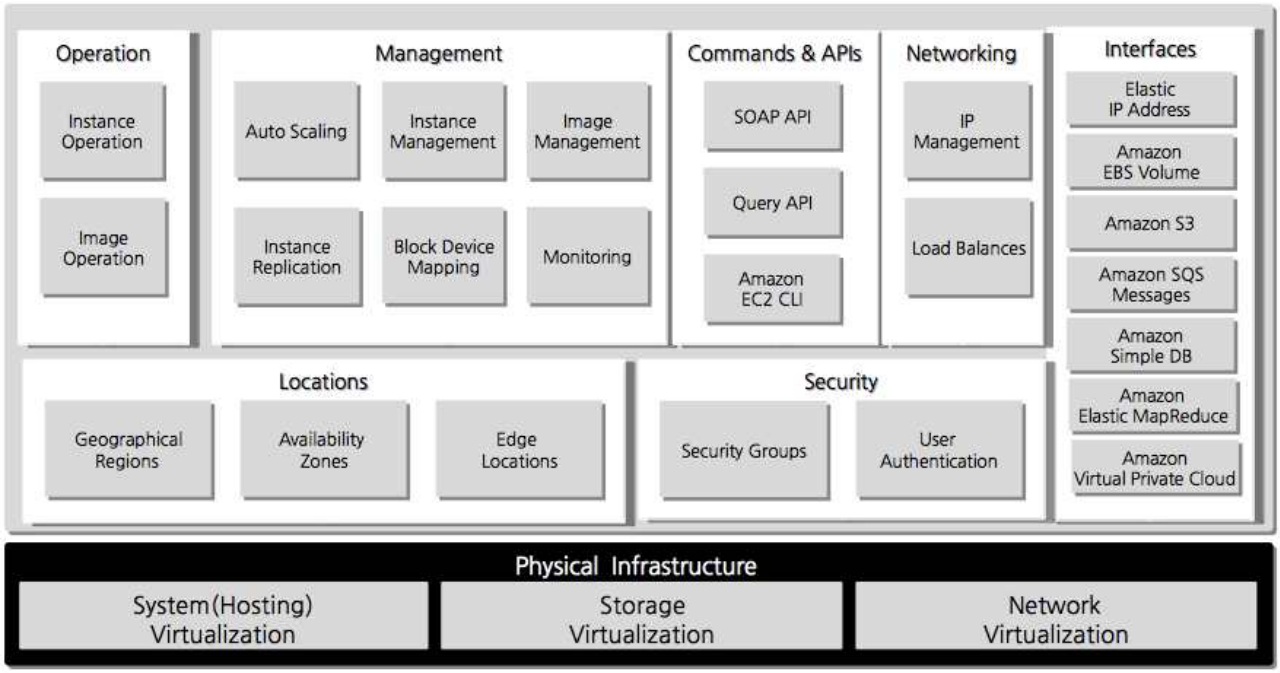


그림 3. 클라우드 인프라 서비스 (IaaS)

필요한 서버 한 대를 확보하기 위해서도 견적 > 계약 > 발주 > 배송 > 설치 > 테스트의 긴 프로세스가 있고 IDC에서 제공하는 '서버 호스팅' 서비스를 받는 경우에도 계약, 설치, 테스트의 과정을 피할 수 없는데, 최근의 클라우드 인프라는 이 모든 과정이 자동화 되어 있어 회원 가입 후 버튼만 누르면 필요한 서버를 즉시 사용할 수 있다. 뿐만 아니라 한 대의 서버를 1년, 3년, 5년 단위로 계약하여 필요에 상관없이 계속 유지하는 것이 아니라, 시간 단위로 과금하고 더 이상 필요하지 않으면 그냥 지워 버리면 끝이다. 웹 상에서 원하는 크기의 서버를 선택하면 그 즉시 사용할 수 있고 사용하다 잠시 중단시켜 두면 과금이 되지 않는다. 개발 환경이나 연구용 환경과 같이 서버 구성이 빈번히 변경되는 경우에 비용을 상당히 절감할 수 있다. 특히 최근 유행하는 빅 데이터(Big Data)기술의 경우 수백 수천 대의

서버를 제한된 시간 동안 사용하는 경우에는 기존 호스팅 방식으로는 그 비용과 구축 시간을 감당할 수 없다.

현재 IaaS 형태의 클라우드 서비스를 제공하고 있는 회사는 대표적으로 Amazon의 AWS (Amazon Web Service), Rack Space, HP Cloud 등이 있고 국내에도 KT uCloud, LG CNS Cloud, T Cloud Biz등이 있다. 10GBps 네트워크, 서비스 이중화, 고성능 SAN스토리지, SSD 스토리지, 전용 서버 Farm 등 고성능 기업형 상품을 준비하기도 하고, 개인 사용자들을 위한 1Core짜리 저렴한 상품들도 제공하고 있다. 즉, 하드웨어 인프라 역시 기업 수준에서 사용하는 개념에서 이제는 일반 사용자도 누구나 사용할 수 있는 편리한 서비스로 바뀐 것이다.

2) 클라우드 플랫폼 서비스 (PaaS, Platform as a Service)

'플랫폼'이라 하면 하드웨어 아키텍처나 소프트웨어 프레임웍과 같이 특정 애플리케이션이 동작할 수 있는 환경을 의미한다. 넓은 의미로는 인프라도 일종의 플랫폼이 될 수 있고, 각종 운영체제(OS)나 자바의 가상 머신(JVM)도 플랫폼에 포함 될 수 있다. 클라우드 환경에서의 플랫폼은 플랫폼의 기본적인 특징을 서비스 형태로 제공할 수 있는 것을 말하며 클라우드 인프라와 클라우드 서비스를 이어주는 일종의 가교 역할을 하는 중요한 요소이다. 물론, 플랫폼이 없어도 서비스하는 데는 지장이 없는 경우도 있지만 최근의 IT환경을 보면 신규 서비스의 생성 속도가 상당히 빠르고 서비스 증가 속도 만큼 유사 기능에 대한 수요도 늘어나고 있기 때문에 각 서비스에 공통적으로 들어가는 기능을 중복 개발하는 것은 낭비일 수 있다. 예를 들어, 10가지 모바일 서비스가 있고 이 모든 서비스는 서버 상태를 사용자에게 알려주는 푸시 노티피케이션(Push Notification) 기능을 사용한다고 하면, 사용 형태가 유사한 10개의 푸시 서비스를 개발해야 하는데 이는 낭비일 것이다. 이와 같은 기능을 플랫폼 서비스로 제공하면 기능의 재사용과 통제가 훨씬 쉬울 수 있다.

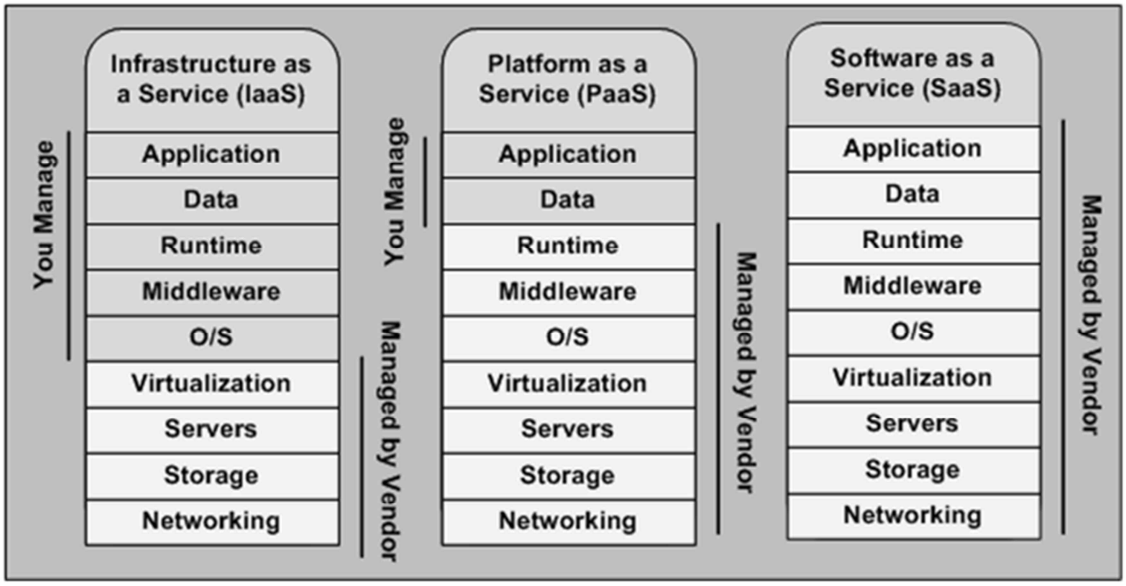


그림 4. 클라우드 플랫폼 서비스 (PaaS)

클라우드 플랫폼 서비스에는 메일 전송 서비스, 로깅 및 모니터링 서비스와 같은 기본적인 서비스에서 부터 SDK, Rest API 등 개발 관점에서의 다양한 도구와 라이브러리도 플랫폼의 범위에 포함될 수 있고, 데이터베이스 서비스, 맵리듀스 서비스, 애플리케이션 배포 서비스, 설정관리 서비스 등 고급 기능을 제공하는 플랫폼 서비스도 있다.

최근에는 웹 서비스의 수 많은 소프트웨어 스택(Software Stack)을 모듈화 화여 마치 레고(Lego)블록을 조립 하듯 웹 애플리케이션을 조립하고 이를 소프트웨어 배포 단위(Java의 경우 jar, war, ear 단위)가 아닌 클라우드 인프라의 가상 서버 단위(Virtual Machine Instance)로 클라우드 인프라 위에 직접 프로비저닝(Provisioning)하여 코드 작성에서부터 운영환경 적용까지의 긴 과정을 단축 시키는 형태의 플랫폼 서비스도 출현하고 있다. (DPaaS, Development PaaS라고도 한다.)

3) 클라우드 서비스 (SaaS, Software as a Service)

클라우드 인프라와 클라우드 플랫폼은 각자의 약어를 풀어 사용한 반면 '클라우드 서비스'만 유독 '소프트웨어'를 뺀 이유는, 일반 사용자 입장에서는 인프라나 플랫폼을 알 이유가 없고, 자신들이 실생활에서 사용할 하나의 서비스라는 측면에서 '클라우드 서비스'라고 줄여부른다.

클라우드 서비스라고 하면 연상 되는 N드라이브, 다음클라우드, DropBox와 같은 서비스는 기술적으로 보면 동기화 서비스, 스토리지 서비스에 더 가까워 보이지만 이 서비스들을 클라우드라고 하는 이유는 클라우드의 기본 특징을 만족하기 때문이다. 웹, PC, 모바일 등 어디서든지 접근할 수 있는 Thin Client의 특징을 가지고 있고 동기화가 어떻게 이루어지고, 디스크에 저장되는 데이터 속성 등에 상관없이

사용자는 '필요한 만큼 사용하는 서비스'의 특징도 가지고 있고 모두다 네트워크를 통해 제공되는 서비스이기 때문이다.

쉽게 구분하기 위해 MP3플레이어를 예로 들어보자. 똑 같은 음악 파일 재생 서비스라 하더라고 만약 MP3파일을 내 PC에 다운로드 받아 플레이어를 통해 재생해주면 이 서비스는 클라우드 서비스라고 하기 어렵다. 만약 온라인 상의 내 공간에 MP3파일을 저장해 뒀다가 내가 듣고 싶을 때 스트리밍을 통해 음악을 재생해주면 그건 클라우드 서비스라고 할 수 있다. 그래서 mSpot, Spotify와 같은 서비스가 클라우드 서비스인 것이다.

IaaS, PaaS, SaaS로 구분된 클라우드 서비스의 분류는 사실 그 경계가 다소 불명확한 면이 있다. 클라우드 인프라 (IaaS)를 제공하고 있는 아마존 AWS의 상품 목록을 보면 PaaS나 SaaS의 모습을 갖는 서비스들이 매우 많다. RDS(Relational Database Service)로 불리는 아마존의 한 서비스는 데이터베이스 인프라를 서비스 형태로 제공하는 상품인데 이를 인프라 서비스로 봐야할 지 프랫폼 서비스로 봐야할지 모호할 수 있다.

4) 기타 분류

클라우드의 3가지 분류 외에도 NaaS(Network as a Service)는 네트워크를 서비스 형태로 제공한다. VPN과 같은 서비스를 서비스 형태로 제공할 수 있는데 VPN 장비를 추가하는 것이 아니라 VPN 에이전트(Agent) 모듈을 대상 서버에 설치하여 상호 간의 동일 네트워크를 형성하고 보안 통신을 가능하게 해준다. 최근 부상하고 있는 네트워크 기술인 SDN (Software Defined Network) 역시 NaaS의 한 형태로 제공될 수 있다.

그 외에 VDI (Virtual Desktop Infrastructure) 또는 DaaS(Desktop as a Service)는 개인 PC를 가상화하여 어디서 접속하더라도 내 PC환경을 그대로 사용할 수 있고 특히 기업의 보안 통제 측면에서 그 활용도가 있는 클라우드다.

마. 클라우드 서비스 유형

클라우드의 분류의 축을 90도 정도 틀어보자. 위에 소개한 클라우드는 사용자를 특정 짓지 않고 모두가 동일한 권한으로 동일한 비용을 지불하고 사용할 수 있는 클라우드로서 이를 '퍼블릭 클라우드(Public Cloud)'라고 한다. 퍼블릭 블라우드는 모두에게 동일한 기능을 제공하기 때문에 가장 보편적인 기능과 사양을 가진 상품을 제공하는 것이 일반적이다. 물론, 서비스가 활성화 되어 더 다양한 상품이 출시 될 수는 있지만 특정 사용자를 위한 맞춤형 상품을 제공하지는 않는다.

그러면 이런 퍼블릭 클라우드만 있는 것일까? 그렇지 않다. 퍼블릭 클라우드 외에도 프라이빗(Private) 클라우드가 있다. 용어 자체에서도 느낄 수 있듯이 프라이빗 클라우드는 특정 기업 또는 특정 조직

만을 위한 클라우드 서비스를 말한다. 오픈 소스를 이용하여 자체적으로 구축한 클라우드도 프라이빗 클라우드라고 하고, 클라우드 솔루션 회사에서 구축을 해주거나, 어플라이언스(Appliance) 형태로 하드웨어와 클라우드 솔루션 일체형 상품을 자체 데이터 센터에 설치한 경우도 프라이빗 클라우드라고 한다.

큰 대기업의 경우 회사 내에 수많은 시스템이 있고 다양한 종류의 인프라를 사용하면서 시스템 별로 자원 산정을 하고 있다. 프라이빗 클라우드는 기존의 기존의 인프라를 묶어 그 위에 클라우드 솔루션을 얹어 클라우드 효과를 내기도 하고, 추가 확보할 인프라를 클라우드 상품으로 대치하여 구매할 수 있다.

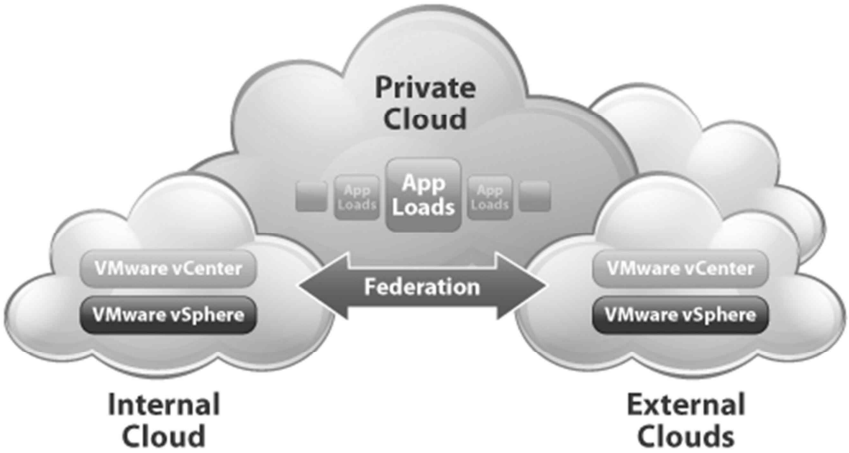


그림 5. 프라이빗 클라우드 개념도

어떤 방식이든 각 부서에서 인프라를 사용하는 측면에서는 퍼블릭 클라우드와 유사하게 필요에 따라 자원을 생성, 삭제 할 수 있음에는 동일하다. 프라이빗 클라우드를 도입하면 기업은 내부에서 필요로 하는 전체 인프라 자원에 대해서만 그 수요를 산정하면 되기 때문에 개별 관리 방식에 비해 유지 비용 절감의 효과를 얻을 수 있을 뿐만 아니라 인프라 자원의 표준화도 가능하다.

1) 퍼블릭 클라우드(Public Cloud)

클라우드 인프라가 일반 대중이나 대형 산업 그룹에게 제공이 되며 클라우드 서비스를 판매하는 기관에 의해서 소유가 된다. '퍼블릭 클라우드'란 기업의 파이어 월(fire wall)의 외측에 구축되는 형태로, 공중의 인터넷망을 통해 불특정 다수의 기업이나 개인 유저에게 제공되는 클라우드 서비스이다.

퍼블릭 클라우드의 장점은 단기간에 고기능의 서비스를 다양한 디바이스로 이용할 수 있는 것과 저비용으로 이용할 수 있어 운용 관리의 부담이 적다는 점이다. 퍼블릭 클라우드는 이용 빈도에 따라 유연

하고 신속한 컴퓨터 자원의 추가나 삭제가 가능하며, 일반적으로 이용량에 따라 과금되는 종량제 형태가 사용되고 있다.

그러나 퍼블릭 클라우드에 대하여 보완 서비스 품질, 사외 데이터보관 등의 이유로 불안을 느끼는 기업들이 많으며, 대기업의 경우에는 기존에 보유하고 있던 다양한 비즈니스 애플리케이션과 수 많은 중요 정보에 대한 책임을 질 수 없다는 점 때문에 사용하지 못하는 경우도 있다.

퍼블릭 클라우드의 특징이라고 할 수 있는 일률적인 서비스 제공이 어떤면에서는 단점이 되고 있는 것이다. 즉, '너무 일률적'이어서 기업 고유의 요구에 맞는 서비스를 기대하기 힘들고, 서비스 제공과정이 정확하게 파악되지 않아 신뢰하기 어렵다는 것이다.

2) 프라이빗 클라우드(Private Cloud)

클라우드 인프라가 특정 기관만을 위해서 운영이 되는 형태이다. 기관 자체 또는 제3자 기관에 의해 관리될 수 있고 영내 또는 외부에 존재할 수 있다.

프라이빗 클라우드는 가상화, 표준화, 자동화 등 클라우드 기술을 활용하여 고객 또는 제공 사업자 측의 데이터센터에 자사 전용의 클라우드컴퓨팅 환경을 구축하는 것으로 컴퓨터 자원을 기업이 유연하게 이용하는 시스템이다. 기업의 파이어 월(fire wall)내에 구축되는 형태의 프라이빗 클라우드는 프라이빗 SaaS, 프라이빗 클라우드 기반(PaaS, IaaS)의 층으로 분류된다.

프라이빗 클라우드를 활용하면 비용 절감, 자산 활용도 증가, 정보 가용성 향상, 신속한 신규 서비스 구축, 비즈니스 대응 능력을 비롯하여 퍼블릭 클라우드와 거의 동일한 이점을 누릴 수 있다. 또한 클라우드컴퓨팅을 고려할 때 가장 우선적으로 보는 부분인 보안 및 규정 준수에 있어서도 프라이빗 클라우드는 신뢰할 만하다.

프라이빗 클라우드는 특정 애플리케이션을 추가하여 방화벽을 통해 기업 경계를 보호하는 볼트온(BoltOn) 방식이 아니고, 애플리케이션 및 시스템에 완전히 통합시키는 빌트인(Builtin)방식 즉, 공장에서 출고 될 때부터 특정 데이터 및 기타 자산에 대해 정의하고 관리할 수 있도록 하기 때문에 보안 및 규정 준수를 강화할 수 있다. 뿐만 아니라 프라이빗 클라우드에서는 방화벽, 암호화, 암호를 비롯한 현재 보안 인프라스트럭처의 모든 구성요소가 그대로 작동되며, 필요에 따라 자산 액세스, 사용, 위치 및 관리 규칙을 가상 컨테이너에 포함할 수도 있다.

3) 하이브리드 클라우드(Hybrid Cloud)

클라우드 인프라가 2개 이상의 클라우드(프라이빗, 커뮤니티, 퍼블릭)로 구성이 된다. 프라이빗 클라우드의 강력한 보안기능과 통제력, 퍼블릭 클라우드의 비용 효율성이 결합된 방식이다.

하이브리드 클라우드 환경에서 사용자는 프라이빗 클라우드를 통해 퍼블릭 클라우드 서비스에 접근할

수 있다. 또 비즈니스 업무로 서비스 접속이 집중되는 피크 시간대에는 퍼블릭 클라우드를 적절하게 할당하고 기업 관련 보안 데이터를 다루는 업무에는 기존 사내 인프라를 활용할 수 있어 합리적인 대안이 될 것이다.

하이브리드 클라우드는 기업들이 구축해 사용하고 있는 IT 인프라를 활용 할 수 잇고 아직 보안, 가용성 등이 완벽학 검증되지 않은 퍼블릭 클라우드를 단위 업무에 먼저 적용해 점차 확대 적용하는 중간 단계 역할을 할 수 있다.

클라우드 시스템이 출현한 기간이 아직 길지 않기 때문에 클라우드에 대한 표준화가 잘 이루어져 있지 않다. AWS의 방식이 업계 표준으로 통용되고 있는데, 상황에 따라 퍼블릭 클라우드와 프라이빗 클라우드를 혼용해야 하는 경우가 있다. 가령 프라이빗 클라우드의 자원이 부족한 경우 퍼블릭 클라우드로 버스팅(Bursting)하는 경우나, B2C와 B2B가 서로 혼용되는 대형 쇼핑몰 시스템의 경우가 있을 수 있다. 이런 경우 두 클라우드를 통합하여 하이브리드 클라우드(Hybrid Cloud) 형태로 활용한다. 하이브리드 클라우드는 기존 레거시 시스템과 연동하여 모든 인프라 자원을 통합 관리하는데 사용하기도 하고, 한 회사가 여러 회사의 클라우드를 운영 대행 해주는 경우, 단일 서비스로 복수 클라우드를 제어할 수 있기 위해 하이브리드 클라우드를 도입하기도 한다.

바. 클라우드 컴퓨팅 기술 요소

앞에서 설명했던 것과 같이 클라우드에 사용되는 기술은 현존하는 모든 IT기술을 총망라한다. 다만 기존 환경에서 추가되거나 바뀐 부분에는 어떤 것이 있는지 살펴볼 필요가 있다. 인프라 측면에서 클라우드 환경에 추가된 기술로는 '가상화' 기술이 있고,

1) 가상화 기술 (Virtualization Technology)

가상화 기술 하나만 놓고 보더라도 그 범위는 매우 넓은데 컴퓨팅에서 가장 중요한 하드웨어(또는 서버)와 운영체제(OS)를 가상화 하는 것을 일컬어 '하드웨어 가상화'라고 한다. 하드웨어 가상화 기술은 컴퓨터 하드웨어의 물리적인 특징은 숨기고 대신 또 다른 추상화 된 컴퓨팅 환경을 사용자에게 제공해 주는 기술로 이미 1960년대에 그 개념이 정립되어 실험적으로 사용되다가 상대적으로 가격이 저렴한 x86 계열 CPU가 서버로 활용되기 시작하면서 그 활용도가 커졌다고 볼 수 있다.

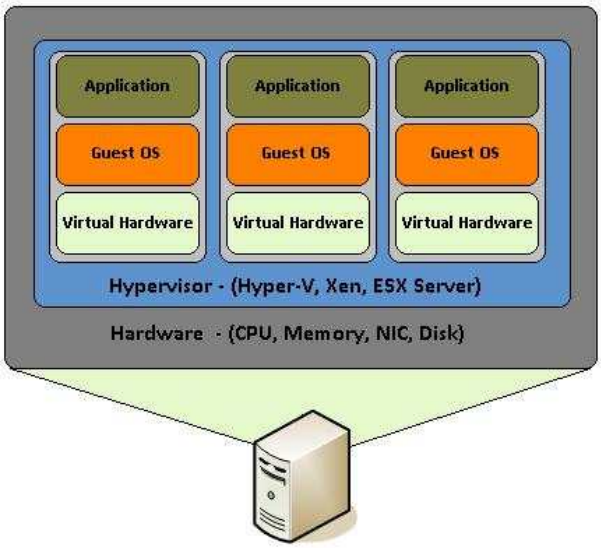


그림 6. 하드웨어 가상화

하드웨어 가상화를 통해 얻을 수 있는 이점 중 하나는 큰 서버 한 대를 여러 대의 가상 서버로 분리하여 사용할 수 있다는 것이다. 아마존 AWS가 처음 클라우드 인프라 사업을 사작하게 된 계기도 아마존의 온라인 쇼핑몰에 투자한 인프라가 놀고 있는데 착안하여 남는 자원을 재활용하여 추가적인 이익을 얻어보자는 데서 출발했다.

그림 7. 하드웨어의 남는 공간을 가상 서버로 재활용

2000년대 중반 x86 계열의 멀티 코어 CPU들은 '하드웨어 가상화'를 지원하기 시작했다. CPU의 하드웨어 가상화는 앞에서 얘기한 하드웨어 가상화와 다소 혼선이 있을 수 있는데, CPU 수준에서 가상화를 지원하기 이전에는 순수 소프트웨어 방식으로 가상화 기능을 구현하여 사용하였다. 그런데 가상화 과정에서 메모리 관리나 CPU의 자원 관리 등에는 추가적인 연산 능력을 소모하게 되는데 이런 '가상화 오버헤드'는 전체 하드웨어 성능을 100% 활용하는데 큰 장애가 되었다. x86계열 CPU가 서버용으로 많이 사용되면서 칩 제조사에서는 CPU에 가상화를 위한 몇 가지 인스트럭션을 추가하였는데 이를 계기로 x86 계열의 가상화가 성능이 제대로 발휘되기 시작했다. Intel의 VT-x, AMD의 AMD-V가 바로 그

기술이다.

소프트웨어적으로 가상화를 지원하는 것을 하이퍼바이저(Hypervisor)라고 한다. 슈퍼바이저(Supervisor)보다 더 상위에 위치하고 있다고 하여 이런 이름을 갖게 되었으며, 하이퍼바이저의 기능은 물리적인 하드웨어에 추상화 된 가상 서버를 생성, 삭제하고 가상 서버에서 필요로 하는 자원을 하드웨어에서 분할하여 제공하는 등의 자원 배분 역할을 수행한다. 다른 말로는 VMM(Virtual Machine Manager/Monitor)라고도 한다.

결국 가상화 기술에서 핵심은 하이퍼바이저라고 할 수 있다. 앞에서 얘기한 소프트웨어 방식의 하이퍼바이저를 완전 가상화(Full Virtualization) 또는 소프트웨어 가상화(Software Virtualization)라 하고 CPU의 하드웨어 가상화 기능을 이용하여 가상화 성능을 향상 시키는 하이퍼바이저를 하드웨어 지원 가상화(Hardware Assisted Virtualization)라 하며, 그 외에 OS 수준에서 가상화를 원활하게 해주는 하이퍼바이저를 부분 가상화(Para-Virtualization)라고 한다.

완전 가상화(Full Virtualization)는 가상 서버의 운영체제(게스트 OS)가 하이퍼바이저 안에 완전히 포개지는 형태를 취하기 때문에 게스트 OS 입장에서는 하드웨어의 운영체제(호스트 OS)를 일체 볼 수가 없고, 게스트 OS를 수정하지 않고 그대로 사용할 수 있는 장점이 있다. 대신 가상화에 필요한 모든 변환 또는 에뮬레이션(emulate)을 소프트웨어로 처리하기 때문에 부하가 큰 환경에서는 전체 성능이 떨어지는 문제가 있다.

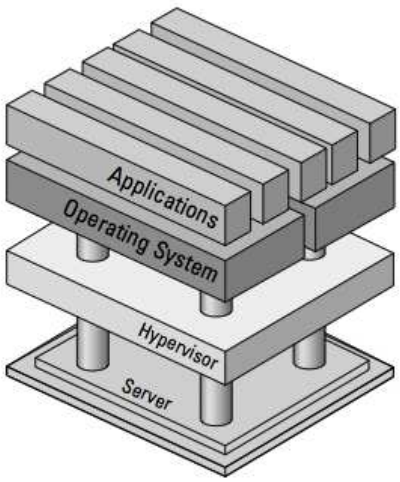


그림 8. 완전 가상화

부분 가상화(Para-Virtualization)는 완전 가상화에 비해 속도가 빠른 장점이 있다. 대신 수정된 게스트 OS를 사용해야 하는 단점이 있다. Xen 하이퍼바이저가 이에 속하며, 커널 버전 3.0 이상의 리눅스 배

포판을 게스트 OS로 사용하는 경우 OS수정 없이 Xen 하이퍼바이저에서 가상 서버를 생성할 수 있다.

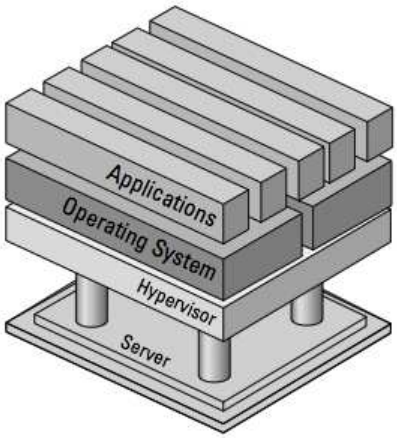


그림 9. 부분 가상화

마지막으로 **하드웨어 지원 가상화(Hardware Assisted Virtualization)는** 글자 그대로 CPU의 가상화 기능을 사용하여 완전 가상화를 실현한 것으로 최근 x86 기반의 클라우드 인프라 서비스는 모두 이 기술에 기반했다고 할 수 있다.

2) 웹 서비스

웹 서비스 기술은 우리가 일반적으로 이해하고 있는 인터넷 웹 페이지 개념 보다는 훨씬 더 큰 범위를 가지고 있다. 웹 서비스는 '네트워크를 통해 기기 간의 상호 통신할 수 있는 소프트웨어 시스템'으로 정의되는데 HTTP를 이용한 통신, Restful 서비스, XML을 통한 데이터 송수신 등이 모두 여기에 포함된다고 할 수 있다.

클라우드 환경에서 웹 서비스가 중요해진 이유는, IT 자원을 서비스 형태로 제공하기 위해 두 대상 간의 간단한 인터페이스가 필요했기 때문이다. 클라우드 인프라를 웹 브라우저에서 통제하고, 클라우드 플랫폼을 사용하기 위해 HTTP로 통신하는 API를 사용하면 별도의 추가 작업 없이 즉시 사용할 수 있다.

3) 씬 클라이언트(Thin Client)

수년 전에는 서버와 클라이언트 간의 부하 분산이 중요한 기술적 이슈가 됐던 적이 있다. 모든 연산 작업을 서버에서 처리하면 수많은 클라이언트를 허용해야 하는 서버 입장에서는 부하의 집중이 부담스럽기 때문이다. 그래서 서버에서는 최소한의 작업을 맡고 많은 부분을 클라이언트에서 처리하도록 하여 서버의 부담을 줄였던 것이다.

클라우드 환경에서는 부하를 클라이언트로 전가하지 않는다. 'Thin'이라는 단어에서도 알 수 있듯이 클라이언트는 최소한의 작업만 수행한다. 여기에는 여러가지 이유가 있는데 다양한 모바일 단말의 보급이 가장 큰 이유라고 할 수 있다. 왜냐하면 동일한 기능을 수많은 개인 기기에 맞게 개발하기가 쉽지 않기 때문이다. 그래서 클라이언트는 최소한의 작업만 수행하고 핵심 작업은 모두 서버에서 처리하되 늘어나는 서버의 부하는 필요에 따라 서버의 자원을 탄력적으로 늘렸다 줄였다 하는 방식으로 전체 시스템의 원활한 유지를 가능하게 한다. 이런 탄력적 자원 활용이 클라우드의 핵심이다.

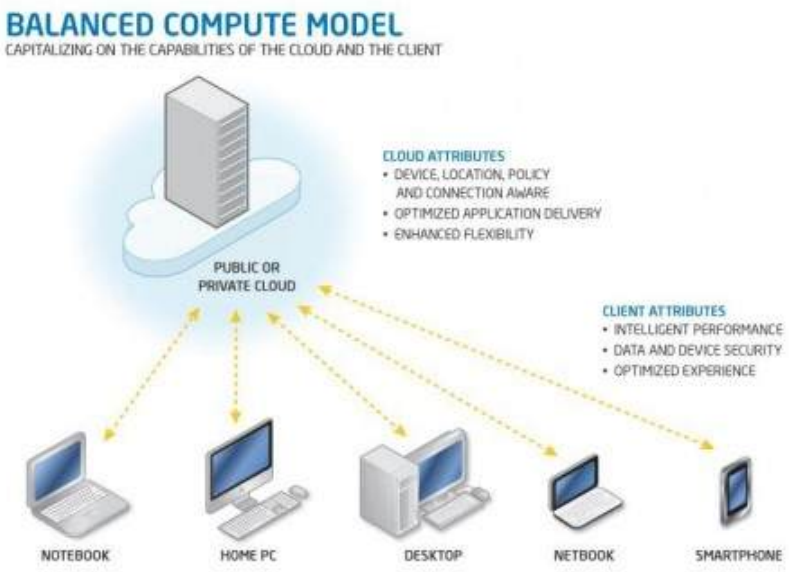


그림 10. 씬 클라이언트

Thin Client에는 크로스 플랫폼(Cross Platform)과 웹 인터페이스의 의미도 들어있다. 가상 서버를 생성하고 네트워크를 연결하고, 방화벽을 설정하는 등의 작업이 모두 웹 브라우저를 통해 제공될 수 있다. 웹 브라우저를 기본 인터페이스로 사용하면 운영 체제(OS)에도 구애 받지 않는다. 또한 HTTP 프로토콜을 사용하기 때문에 상호 통신시에 발생할 수 있는 문제도 적고 HTTP 모듈은 이식성이 매우 높기 때문에 어떤 기기에서도 사용할 수 있다.

사. 클라우드 컴퓨팅 시장

Gartner 보고서에 의하면, 2010년~2011년 ICT분야중 가장 높은 성장이 예상되는 전략기술로 클라우드컴퓨팅 분야가 선정 되었다.

그림 11. 클라우드 컴퓨팅 시장

1) 전세계 시장 현황

2010년 4월 조사된 정보통신정책연구원(KISDI)에 의하면, 전 세계 클라우드 컴퓨팅 시장 규모는 2009년 769억 달러에서 연평균 34%의 높은 성장률을 기록하며 2013년에는 2,563억 달러, 그리고 2014년에는 3,434억 달러에 이를 것으로 전망한다

미국 국방부 정보시스템 계획국(DISA: Defense Information System Agency)은 국방부와 군사령부를 비롯한 다수 국방관련 기관들이 이용할 수 있는 클라우드 컴퓨팅 인프라를 구축, 1차 테스트를 마쳤으며, EU에서 2,500만 달러의 규모로 후원하는 Reservior(Resources and Services Virtualization without Barrier) 프로 젝터에서는 IBM, Thales 등 13개 기관이 참여해 분산 클라우드 간의 상호연동 기반기술을 연구 중에 있다.

영국 CARMEN(Code analysis, repository, and modelling for e-Neuro Science)프로젝터에서는 2006~2010년에 900만 달러를 투입하여 영국 내 대학 및 업체가 연계한 e-Science 클라우드 플랫폼

제공을 추진하고 있으며, 일본은 문부과학성
(MEXT)의 지원 하에 NAREGI(National Research Grid Initiative) 프로젝트를 진행 중에 있다.

2) 국내 시장 현황

우리나라는 차세대 IT패러다임으로 클라우드 컴퓨팅이 주목받고 있는 가운데 정부도 원천기술 개발과 신규 서비스 발굴 등 본격적인 산업 육성에 나섰다. 특히, 지식경제부에서는 클라우드 컴퓨팅과 그린 컴퓨팅 분야에 대한 기술개발 내용이 집중적으로 논의되고 있으며, '독립형 콤포넌트 기반 페타급 컴퓨팅 플랫폼 기술 개발'과 '신뢰성 컴퓨팅 기반기술 개발'을 진행하고 있다.

KT는 클라우드 추진사업단을 신설('10.5)하여 데이터센터 가상화 및 사내 클라우드컴퓨팅 도입을 추진하고, KT를 클라우드화 하여 레퍼런스로 활용할 예정이다. PC뿐만 아니라 전자책, 스마트폰 등과 같은 다양한 모바일 기기 간 콘텐츠를 공유하고 통합관리.저장할 수 있는 모바일 클라우드 스토리지 서비스(u클라우드)를 6월부터 본격 시작했다.

3) 국내외 기술 개발 동향

국내외 기업의 클라우드 컴퓨팅 기술개발 동향은 다음과 같다.

표 2. 클라우드 컴퓨팅 기술개발 동향

주요기업	기술개발 동향
Google	웹 기반 서비스 시장에서 소프트웨어서비스 시장, 점차 IT 인프라 시장으로 영역을 확장하고 있으며, 2008 년 5 월에는 자사의 플랫폼 상에서 애플리케이션을 자유롭게 개발·이용할 수 있도록 해 MS 가 지배해 온 PC 플랫폼을 웹 상에서 재현하고자 하는 전략을 추구
HP	Intel, Yahoo, 싱가폴의 IDA(Infocomm Development Authority of Singapore), 미국 UIUC, 독일 KIT(Karlsruhe Institute of Technology) 등과 제휴한 Cirrus 테스트베드 프로젝트를 2008 년 7 월 발표하고, 클라우드 컴퓨팅을 위한 오픈 플랫폼 연구를 진행하고 있음
MS	클라우드 컴퓨팅과 관련하여 가장 후발주자로서 2008 년 10 월 Azure 서비스 플랫폼을 발표하고 2009 년 2 월 'Cloud computing futures'라는 연구조직을 구성하는 등 기존의 PC 기반 플랫폼 사업자에서 웹 기반 플랫폼 사업자로 발빠르게 진화하고 있음
IBM	2008 년 2 월 EU 와 공동으로 RESERVOIR research intiative 를 발표하고 'Blue cloud'라는 이름으로 클라우드 컴퓨팅 산업에 대한 계획을 발표하였으며, 클라우드 컴퓨팅 환경을 기업 고객 내부에서 구현할 수 있도록 기반 설비 구축에 치중하고 있음.
KT	가상화와 서버 통합을 통해 수천 대에 달하는 사내 IT 인프라 자원을 효율적으로 활용하고 운영비를 절감하는 한편 국가 차원의 클라우드 서비스 모델을 제공하고 있음.

아. 클라우드 컴퓨팅의 파급효과

클라우드컴퓨팅이 미치는 영향은 IT서비스 환경이 복합적인 공급방식으로 변화되고 있다는 것을 전제해야 할 것이다. 변화 측면에서 볼 때, 내부 인프라가 점차 가상화되고 동적화 될 것이며 외부 호스팅 서비스 활용도가 높아 질 것이다. 물론 모든 것들이 효과적으로 상호 협력해야 한다는 전제도 포함된다.

IT 기획에는 많은 규칙과 프로세스를 가지고 있는 다면적인 특성을 가지고 있기 때문에 클라우드컴퓨팅이 내부 정보화 환경에 어떤 영향을 미치게 될 것인지를 고려해 보는 것이 좋다.

1) 아키텍처 설계 및 개발

세심하게 계획된 아키텍처식 방식의 중요성은 클라우드 옵션을 수용할 때 더욱 두드러진다. 서버, 네트워킹 및 저장소 인프라의 구조와 일괄성은 효과 적인 프라이빗 클라우드를 구축하는데 있어서 매우 중요하며 IaaS와의 상호 운용성을 위해서도 필수적이다. 온 프레미스 서비스 영역과 호스팅 서비스 영역간 간편한 통합 및 리소스 풀들 간의 응용프로그램 작업 부하 이전을 위해 서도 소프트웨어 설계화 개발을 위한 아키텍처식 방법이 필요하다. 따라서 견고한 아키텍처의 개발 전문 지식은 클라우드컴퓨팅을 성공적으로 구축하기 위한 전제조건이며 모든 요소들이 어떻게 상호 운용될 것인지에 대해 정확하게 파악하지 않은 채 클라우드로 이동할 경우 엄청난 결과가 초래될 수 있다.

2) 보안, ID 및 액세스 관리

이 부분의 기본 원칙은 크게 달라지지 않지 않았지만 응용프로그램의 범위와 복잡성은 확대 되었다. 단기간 내에 거의 모든 클라우드 서비스 공급업체들이 디렉터리 동기화를 통해 정책 통합을 자동화하는 매커니즘을 구현할 것이라는 기대는 비현실적이다. 최종 사용자들이 일방적으로 클라우드 서비스에 가입할 수 있다는 간편성을 고려해 볼 때, 이들이 사용하는 모든 서비스를 통제할 수 있을 것으로 기대되지 않는다. 따라서 호스팅 서비스 관련 위험을 평가하고 이러한 서비스를 마이그래이션하는데 필요한 것이 무엇인지 파악한후 적절한 정책을 세우고 필요한 통합을 구현할 수 있는 스킬과 경험이 필요할 것이다.

3) 저장소 및 정보 관리

저장소 관리는 종종 등한시 되고 하는 부분이지만 클라우드에서는 간과되어선 안 되는 부분이다. 인프라가 점차 가상화되고 동적화 되며 저장소 요소를 포함하게 됨에 따라 계층화된 저장소 아키텍처(IaaS 영역으로 확대될 수도 있는)를 설계, 구현 및 관리할 수 있는 스킬과 전문성이 훨씬 중요해졌다. 그리고 데이터가 여러 내/외부 클라우드로 이동함에 따라 규정 준수와 무결성 보장을 위해 콘텐츠 관리 및

정보 관리 스킬이 매우 중요하다. 호스팅 환경에서 유지 관리되는 데이터에 대한 백업 복원, 보관 및 검색과 같이 보안, 액세스, 그리고 다시 저장소로 되돌려 보내는 과정이 끊임없이 반복되어야 한다.

4) 시스템 모니터링 및 관리

동적 가상화 서버, 저장소 및 네트워킹 환경에 맞게 모니터링과 관리 프로세스를 강화해야 하는 것 외에도 증가하는 호스팅 서비스로 이해 부가적인 문제가 야기될 수 있다. IT 기획 담당자는 공급업체 환경에 대한 가시성이 종종 제한되곤 하겠지만 공금업체와 SLA(Service Level Agreement)를 정하고 이를 토대로 성능을 모니터링하는 업무에 능숙해져야 할 것이다. 호스팅 서비스와 온 프레미스 시스템 간, 그리고 호스팅 서비스들 간 종속성을 이해하고 이를 적절하게 매핑해야 하므로 종단간 모니터링과 관리 및 문제해결의 복잡 성이 가중된다.

5) 최종 사용자 지원

데스크톱 및 온 프레미스 비즈니스 응용프로그램 기능성과 관련된 최종 사용자 지원 측면 역시 크게 달라지지 않는다. 하지만, 종속성이 일시적인 경우에 동적 환경의 문제 해결 업무를 다른 그룹에 이관해야 하는 경우가 늘어나게 된다. 이 경우 내부 헬프데스크와 호스팅 서비스 공급업체 간 지원을 통합하는 것이 관건이며 사용자 혼란과 불만족을 방지하기 위해선 이 문제를 반드시 해결해야 한다.

위와 같이 클라우드컴퓨팅이 미치는 영향은 위에 설명된 내용이 전부가 아니며, 본 표준 지침에서 설명하고자 하는 것이 클라우드 구축의 모든 범위와 내용을 포함하는 것은 아닙니다. 다양한 측면에서 변화에 대응하며 보다 실용적이고 효과가 있는 시스템 구성이 되었으면 한다. 특히 공개SW기반으로 추진하는 것이 얼마나 국내 산업 및 인력양성에 도움이 되고, 예산을 절약할 수 있는 방법인지를 공유 했으면 한다.

자. 클라우드 로깅 아키텍처 사례 분석

앞에서 살펴본 바와 같이 클라우드 서비스는 그 분류에 따라 서로 다른 다양한 기술을 사용하고 있다. 따라서 클라우드는 단순히 '클라우드 아키텍처'와 같은 제목으로 하나의 표준화된 아키텍처를 만들 수는 없다.

대신 클라우드를 구성하는 모든 요소 아래에서 공통적으로 필요로 하는 기술에 대해 표준 아키텍처를 설계해 두면 클라우드 환경을 이용하여 새로운 서비스를 개발하는 회사나 클라우드 자체를 구축하려는 기업 입장에서 바닥부터 시작하지 않고 표준 아키텍처를 참조하여 나름데로 생각하는 가치를 만들어 낼 수 있을 것이다.

본 문서에서는 클라우드의 공통 요소 중에 '로깅 및 모니터링'을 세부 주제로 하여 표준화된 아키텍처를 제공할 계획이다. 클라우드 데이터 센터에서부터 클라우드 인프라, 플랫폼 그리고 다양한 클라우드 서비스, 모바일 서비스에 이르기까지 로깅과 모니터링은 빠질 수 없는 공통 컴포넌트에 해당한다. 클라우드는 거대하며, 과거에 정적이었던 부분이 더 동적으로 바뀌었으며, 클라우드 자체는 간단할지 모르지만 그 내부는 훨씬 더 복잡해졌다. 기존의 로깅 및 모니터링 기술보다 한 수준 진화한 로깅 및 모니터링이 필요하다는 것은 구지 강조하지 않아도 느낄 수 있을 것이다.

앞에서도 얘기했듯이 로깅 프레임웍을 상용 또는 오픈소스로 출시한 사례는 상당히 많다. 기존의 고정 인프라를 대상으로 만들어졌던 솔루션도 점점 클라우드 환경에 맞게 새로운 버전이 출시되고 있고, 각사의 로깅 기술을 활용하여 새로운 클라우드향 제품을 만들어내기도 한다. 최근 오픈소스의 지위 상승과 더불어 빅데이터 또는 분산 로깅 분야에서 제 역할을 톡톡히 하고 있는 오픈 소스를 분석해 본다.

1) Scribe

Scribe는 페이스북에서 만들어서 사용하는 오픈 소스 로그 관리 시스템이다. 로그를 수신하는 서버와 수신 서버에 로그를 전송하기 위한 클라이언트 인터페이스만 제공한다. 따라서 서버의 로그 저장소와 시스템 구성도 직접 해야한다. 그 중에서도 클라이언트 모듈은 모든 시스템에 맞도록 구현하기 어려워서 별도로 제공하지 않고, 인터페이스만 제공하기 때문에 클라이언트가 동작하는 환경에 맞춰서 직접 개발해야한다.

기존 시스템들 중에서 로컬에 로그 파일을 저장하는 시스템의 경우에는 Scribe 서버를 로컬에 설치하고, 서버로 로그 파일을 전송하는 클라이언트 모듈을 개발하는 과정이 필요하다. 클라이언트 모듈에서 로컬에 생성된 로그 파일을 서버로 전송되고, 서버는 다시 Scribe 중앙 서버로 전송해서 저장한다. 이

때 중앙 서버는 일반적으로 Hadoop 이거나 NAS 같은 전용 스토리지로 구성한다.

2) Chunkwa

Chunkwa는 Hadoop서브 오픈소스 프로젝트로 클라우드 서버의 로그 파일을 Hadoop 파일 시스템(HDFS)로 저장하고 MapReduce를 이용해서 분석하는 솔루션이다.

일반적인 로그 수집도 가능하며 로그 관리를 위해 설계된 솔루션이지만, Hadoop 서브 프로젝트라서 Hadoop 클러스터 자체의 로그나 클러스터 서버 상태 정보를 수집 관리하는 기능도 포함되어 있다. 특히 Hadoop 클러스터의 경우에는 여러 대의 서버에 로그가 분산 저장되어 한꺼번에 보기 불편한 데 이럴 때 유용하게 쓸 수 있는 기능이다. HDFS를 이용해서 수집된 로그를 저장하기 때문에 안정적인 수집과 준 실시간 분석도 가능하다.

하지만 scribe와 달리 Chunkwa는 Hadoop에 의존적이기 때문에 Hadoop이 없는 환경에는 적용하기 어려운 단점이 있다. 그리고 로그 자체를 저장할 때도 바이너리 파일로 저장하기 때문에 별도의 전용 뷰어가 있어야 하는 불편함이 있어서 기존 시스템과 연계성이 조금 떨어진다.

3) Flume

Flume은 분산 환경에서 안정적이고, 신뢰할 수 있는 높은 가용성의 로그 수집 및 통합 솔루션으로 대표적인 오픈 소스 제단인 아파치(Apache)에서 개발되고 있는 프로젝트다. 대량의 로그 데이터를 다양한 로그 소스로부터 중앙으로 수집해주는 역할을 주로 수행하며 1.0이전 버전을 리펙토링하여 그 불안전성을 대폭 개선한 Flume NG버전을 출시하여 실제 환경에도 적용할 수 있는 수준으로 만들었다. Flume은 간단하면서도 매우 유연한 아키텍처를 구현하기 위해 파일이 아닌 스트리밍 방식의 데이터 흐름을 이용한다. 또 다양한 신뢰성 향상 메커니즘이 적용되어 있고 분산 구조를 통한 다양한 페일오버(fail over) 및 복구 메커니즘에 포함되어 있다.

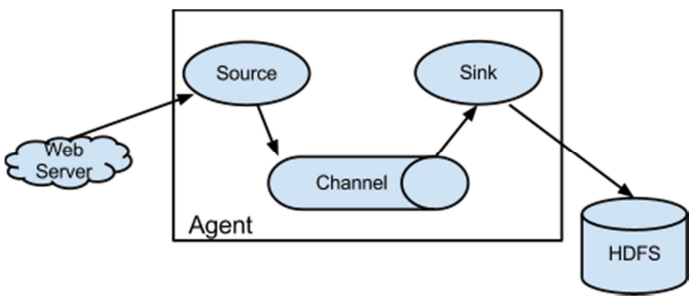


그림 12. Flume 구조

Flume의 데이터 수집 모델은 '흐름(flow)'다. 웹 서버나 각종 장비에서 발생하는 문자열 형태의 메트릭(metric) 정보가 흐르는 동안 이를 잡아서 소스로 전달해준다. 즉, 데이터 소스의 종류 하나하나가 흐름이 되는 것이다. 따라서 흐름의 특성에 따라 압축을 하거나, 배치 처리를 하거나, 안전성 보장 장치를 추가할 수 있다. 위 그림에서 웹 서버의 로그 이벤트는 소스(Source)로 수집 되고 이때 Flume 설정에 따라 필터링(filter)된어 싱크(Sink)로 전달된다. Flume 에이전트는 다수의 Flume 노드(Node)로 구성이 될 수 있는데, 소스에서 수집 된 데이터가 필터링을 거처 싱크(Sink)로 들어가고 이 데이터는 다시 다른 노드의 소스(Source)로 들어가 다른 또 형태의 처리를 할 수 있는 방식으로 마치 파이프라인 구조처럼 노드간 중첩이 가능하다. 최종적으로는 로그 데이터가 수집 되고 정제되어 HDFS로 들어가게 되는데 이 최 말단 싱크에는 HDFS, HBase등이 올 수 있다.

Flume은 기본적으로 스테이트리스(stateless)로 동작하는 분산 시스템 구조를 갖고 있다. 따라서 장애 상황에 대한 손실 비용이 적고 성능이 우수하다. 당연히 횡확장성이 보장이 된다. 로그를 수집해야 하는 대상이 많아지면 단순히 노드를 추가하기만 하면 전체 수집 능력이 증가하고, 여러 노드 중 하나가 죽으면 다른 노드가 그 역할을 자동으로 인계 받는다.

Flume의 설정은 매우 단순하다. 분산 환경에서 동적으로 변하는 인프라에 대응하기 위해 반드시 필요한 조치다. 비교적 단순한 설정으로 Flume의 로그 수집, 정제 역할을 수행할 수 있으며, 시스템 다운 없이 런타임에 설정을 바꿔도 바로 인식될 수 있다.

표 3. Flume 콤포넌트

Flume 콤포넌트	역할
에이전트(Agent)	Flume이 동작하는 JVM으로 서버당 하나를 설치하면 다수의 소스와 싱크를 생성할 수 있다.
클라이언트(Client)	로그 이벤트를 만들어내는 원천을 말한다. (웹 서버 등)
소스(Source)	클라이언트에서 생성된 로그 데이터가 수집 되는 곳으로 플러그 인을 지원하기 때문에 다양한 소스를 적용할 수 있다.
싱크(Sink)	체널을 통해 데이터를 수집하는 역할을 수행하며 별도의 스레드로 동작하여 다른 작업의 간섭을 받지 않는다.
체널(Channel)	소스와 싱크 사이의 큐(Queue) 역할을 수행하며 둘 간을 연결하고 트랜젝션 유지를 담당한다. 플러그 인 인터페이스를 가지고 있어 사용자 자체적으로 안전성 보장 메터니즘을 추가할 수도 있다.
이벤트(Event)	로그 데이터가 Flume에서 활용되기 위한 단위. 바디(Body)와 메타데이터(metadata)로 구성되며 로그 데이터는 바디에 Key-value형태의 추가 정보는 메타데이터에 저장한다.

차. 클라우드 참조 모델

지금까지 클라우드의 일반적인 정의와 분류, 그리고 클라우드 환경에서 필요한 공통 아키텍처에는 어떤 것이 있는지 살펴봤고, 그 중에서 로그 및 모니터링의 중요성에 따라 유사한 오픈 소스 솔루션에 대해서도 간단히 살펴봤다.

클라우드 그룹의 참조 모델은 모든 아키텍처를 다루는 거나 모든 아키텍처를 대신하는 오픈 소스 프로젝트를 만드는 것에 목적이 있는 것이 아니다. 국내 클라우드 환경에서 우선적으로 필요하고 Hadoop이나 다른 솔루션과 공존할 수 있으며, 오픈 소스 프로젝트로 발전 가능성을 가진 아키텍처를 찾는 것을 목표로 했다.

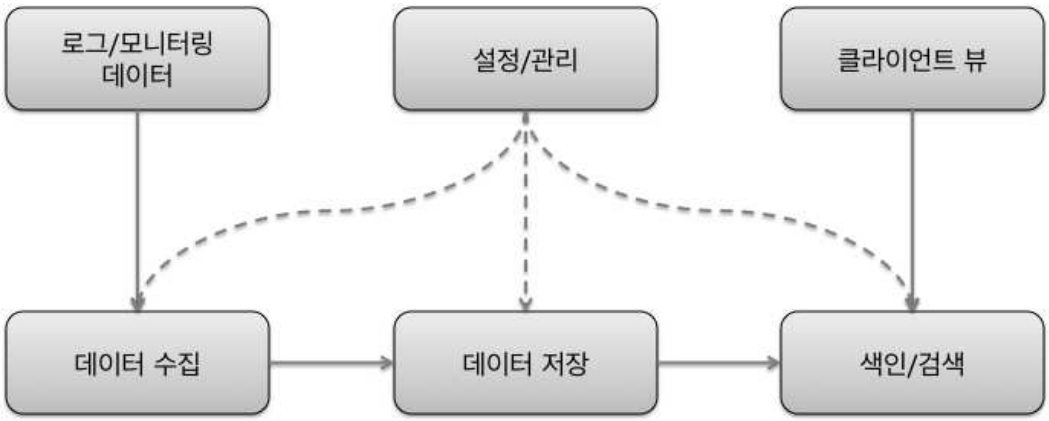


그림 13. 참조 모델 기본 개념도

클라우드 환경에 특화된 로깅 및 모니터링의 요구 사항을 분석하고, 각 요구 사항에 대한 품질 속성을 정의한 후 기본적인 로깅 및 모니터링 아키텍처를 다양한 뷰로 설계해본다. 우선 기본적인 로깅 및 모니터링 아키텍처를 제시하고, 각 구성 요소의 특징과 역할, 모듈 구성과 데이터 구조, 네트워크 측면의 뷰(View) 정의, 다양한 품질 속성에 대한 설명과 함께 다수의 활용 시나리오를 형상화 함으로써 클라우드 멀티 노드와 레거시 시스템을 연동하는 새로운 로깅 및 모니터링 솔루션의 개발 또는 최적 솔루션의 선정 등에 활용될 수 있도록 표준 아키텍처를 제공하는 것을 그 목적으로 한다.

2. 참조모델 요구사항 분석

가. 요구사항 분석 및 설계

1) 참조모델 설계 절차

본 참조 모델 설계 절차는 다음과 같이 4단계 - 수집, 분석, 설계, 검증으로 구성된다.

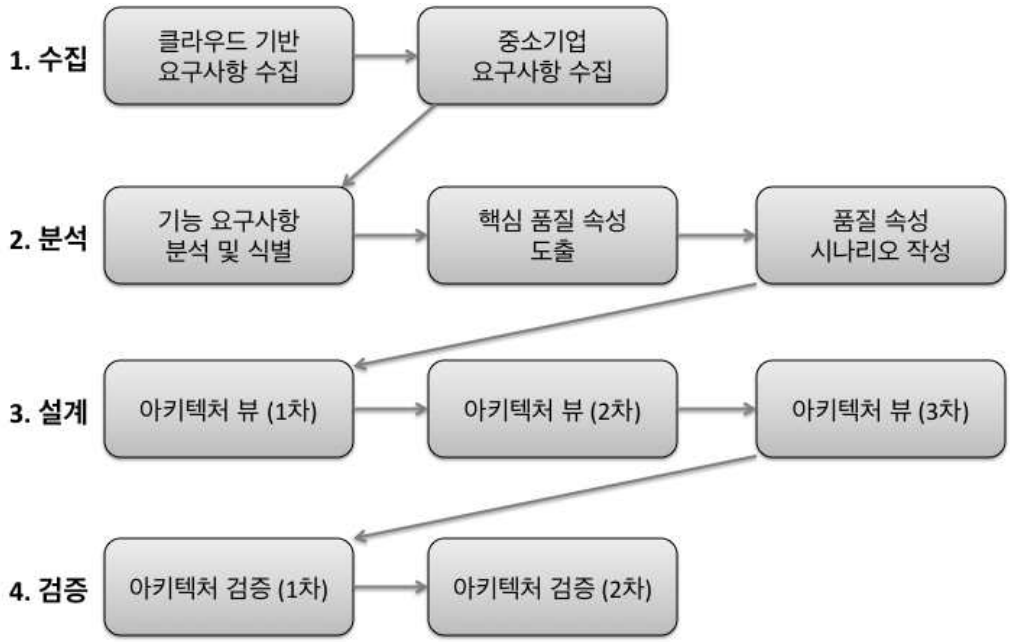


그림 14. **참조모델 설계 절차**

2) 담당자별 역할

표 4. 포럼 담당자

담당자	역할
김정	네트워크 관점 클라우드 기술 요구사항 수집 및 전체 통합
장회수	자바 기술과 실무 아키텍트 관점 클라우드 기술 요구 사항 수집과 분석
한상욱	PaaS, IaaS 관점 클라우드 기술 요구 사항 수집과 분석
고준일	모바일 관점 클라우드 기술 요구 사항 수집
김성조	클라우드 운용 관점에서 요구사항 수집과 분석

장선진	클라우드 서비스와 오픈 소스 프로젝트 관점에서 요구사항 수집과 분석

3) 입력/출력물 정의

i) 입력물

- 각 분야별 클라우드 기술 - 기능 요구 사항
- 클라우드 기반 서비스 아키텍처 모델
- 클라우드 사례별 상용 아키텍처 모델
- 오픈소스 프로젝트 동향
- 비즈니스 품질 목표
- 시스템 요구 사항
- 상위레벨 기능 요구사항
- 시스템 제약사항
- 개발 환경 및 기술 환경

ii) 출력물

- 클라우드 기반 기술 - 요구사항 분석서
- 클라우드 기반 상용 아키텍처 분석서
- 개발 대상 시스템 설명 자료
- 중요 품질 속정 리스트
- 품질속성 시나리오
- 아키텍처 뷰
- 참조 모델 오픈소스

4) 절차별 수행 방법

참조 모델 아키텍트 포럼은 총 10회에 걸쳐 이루어졌다. 그 중 수집 단계 2회, 분석 단계 3회, 설계 단계 3회, 검증 단계 3회로 나누어 진행했다.

i) 수집 단계

- 클라우드 기술 적용 사례 수집
- 중소기업 도입 가능한 클라우드 기반 기술 요구사항 수집

- 클라이드 서비스 업체에서 필요한 공통 기술 수집

ii) 분석 단계

- 비즈니스 품질 목표 이해
- 시스템 요구사항 분석
- 상위레벨 기능 요구사항 분석
- 시스템 제약사항 분석
- 개발 환경 및 기술 환경 이해
- 핵심 품질 요소 도출
- 품질 속성 시나리오 작성

iii) 설계 단계

- 아키텍처 드라이버 선정
- 아키텍처 뷰 1차 작성
- 아키텍처 뷰 2차 작성
- 아키텍처 뷰 3차 작성

iv) 검증 단계

- 아키텍처 검증

나. 비즈니스 관점의 시스템 환경

1) 배경

클라우드 환경이 일반화되고, 대부분의 시스템들이 분산 구조를 채택하면서 관리해야 하는 노드의 수가 늘어나고, 다양한 기술의 도입으로 서로 엮이는 모듈의 수가 많아졌다. 뿐만 아니라 새로운 시스템의 출현 주기가 점점 단축되고 있고 그 형태도 다양해지는 추세이기 때문에 기존의 로깅 모니터링의 방식을 그대로 적용하기 보다는 보다 효율적인 확장이 필요하다.

과거의 로깅 및 모니터링이 시스템의 안정적 운영 측면에 그 필요성이 있었다면, 향후의 로깅 및 모니터링은 새로운 가치의 창출로 연결될 필요가 있다.현재 구축된 시스템이 자동 확장이 필요한 시스템이라던지, 기존에 버리던 사용자 로그를 충분히 수집하여 새로운 비즈니스의 기회로 만들 수도 있다.

클라우드 시스템의 도입의 관점에서 보면, 기존의 고정 프레임 로깅(fixed frame logging) 방식에서 확장 프레임 로깅(flexible frame logging)으로 전환하여 플러그인을 통해 새로운 기능의 확장이 가능한 구조와 프로토콜이 필요하다.

2) 제품 포지셔닝

본 문서에서 제시하는 아키텍처는 로깅 전반에 걸친 일반화된 아키텍처이다. 본 아키텍처를 확장하여 실제 로깅 및 모니터링 솔루션을 개발할 수도 있고, 현재 구축된 시스템에서 부족한 부분을 채워 넣거나 개선하는 용도로 사용할 수 있다.

로그 에이전트에서 가시화(Visualization)까지 로깅 및 모니터링의 전체 범위 중에서 로그 수집 및 검색까지 다루고 있다. 타 시스템과의 연동을 관장하는 프로비저닝(Provisioning)과 MR(Map Reduce) 그리고 가시화 부분은 범위에서 제외한다. 특히 Flume, splunk 등 빅데이터 솔루션과 직접적으로 겹치지 않으면서 필요한 클라우드 기반 기술에 집중한다.

다. 시스템 환경 이해

1) 제품 명세 및 시스템 구조

클라우드 시스템을 구성하더라도 무관하도록, HW 제품 명세는 구체적으로 명시하지 않지만, 다음과 같은 클라우드 시스템을 기준으로 통합 로그/모니터링 시스템 참조 모델을 설계했다.

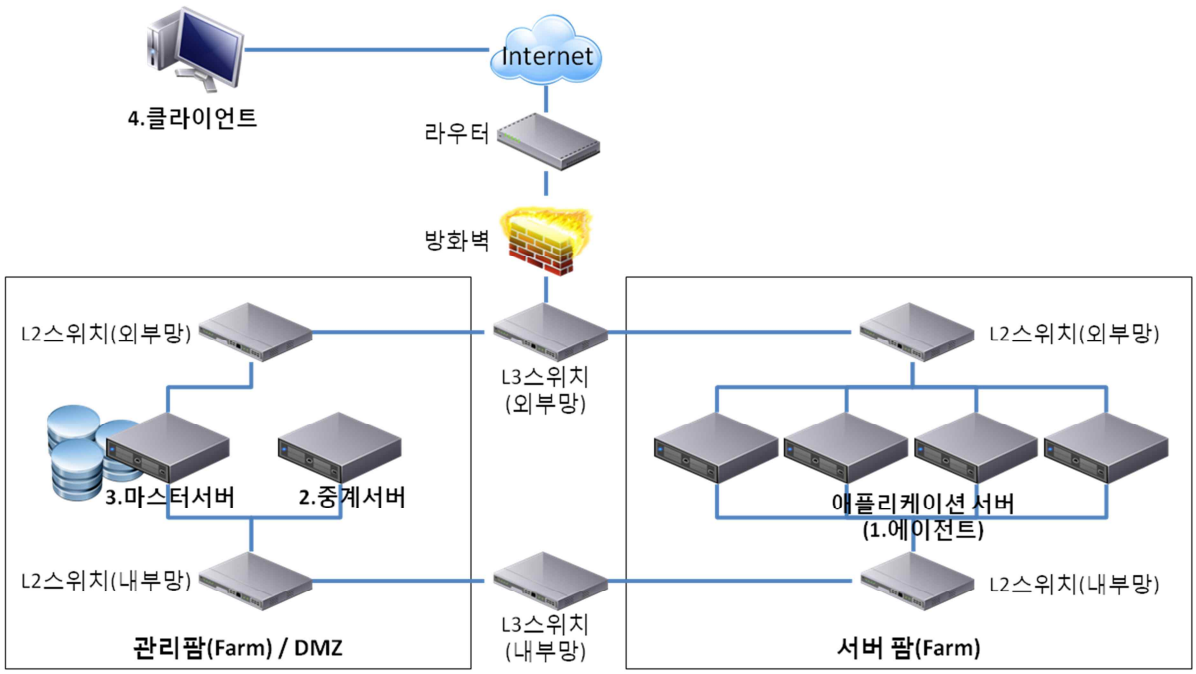


그림 15. 시스템 구조

시스템 구조의 특징은 다음과 같다.

- 클라우드 시스템의 특성상 서버팜과 관리팜으로 네트워크 구성을 구분한다.
- 관리 목적과 운영을 위해 네트워크를 이중화하여, 내부망과 외부망을 분리한다.
- 에이전트(1) : 에이전트는 애플리케이션 서버에 설치되는 작은 프로그램으로 서버 성능에 영향을 주지 않고 서버 또는 애플리케이션의 로그/모니터링 데이터를 수집하는 역할을 수행한다.
- 중계서버(2) : 로그 에이전트 또는 네트워크 장비로부터 실시간 로그 정보를 수집하는 역할을 수행하며, 대규모 인프라 환경하에서는 중계서버를 복수로 구성하여 로그 수집을 위한 성능을 향상할 수 있다.

- 마스터 서버(3) : 중계서버를 통해 수집된 비정형 로그 데이터를 필터링, 정제, 분석 및 조합하여 사용자가 원하는 가공된 데이터로 변환하여 데이터베이스 형태로 저장하며, 별도의 API를 제공하거나, 웹 UI를 통해 시스템의 현황을 보여주는 역할을 수행한다.
- 클라이언트(4) : 웹, 모바일 단말, PC용 프로그램 등 사용자 기기에 설치되어 실시간 로그 정보를 조회하거나 재집계하는 사용자 인터페이스를 지칭한다.

2) 기술적 관점의 아키텍처 환경

클라우드 환경에서 인프라 규모는 고정되지 않는다. 운영 서버의 수는 실시간으로 늘었다 줄었다 하고, 때문에 매번 설정을 하고 모니터링 대상에 추가하는 것은 불가능에 가깝다. 추가되는 자원은 자동으로 로그 수집 범위에 포함되고 또 자동으로 로그를 수집할 수 있어야 한다. 그래야 대규모 인프라 자원의 관리가 가능하며, 사람의 실수에 의한 데이터 왜곡 또는 유실을 미연에 방지할 수 있다.

뿐만 아니라 인프라의 횡(橫) 확장(scale-out)은 결국 엄청난 량의 로그 데이터를 생성하기 때문에, 보다 지능적인 필터링이 가능해야 하고, 분산 시스템 특성 상 더욱 더 복잡해지는 내부 구조로 인해 문제의 진원(震源)지를 정확히 파악하기 위해서는 각 계층 또는 모듈 간의 추적이 가능해야만 한다.

로그 정보는 더 이상 시스템 운영을 위한 모니터링의 기초 데이터로만 활용되는 것이 아니다. 로그 정보를 가공하여 시스템 아키텍처 개선이나 비즈니스 관점에서 신규 상품 개발 등의 자료로도 충분히 활용할 수 있다. 외부 시스템과의 연동도 중요한 항목으로 이를 위해 표준화가 쉬운 구조의 프로토콜을 지원해야 한다.

본 문서는 클라우드 모든 환경을 커버하는 전체 아키텍처를 구성하는 것이 아니며, 클라우드 서비스와 클라우드 기반 기술을 활용하는 실무 환경에 적합한 “로깅 및 모니터링” 아키텍처에 집중한다.

클라우드 컴퓨팅 환경에 필요한 기술적인 아키텍처 환경은 매우 방대해졌다.

- 물리 자원을 논리적으로 관리할 수 있도록 가상화 (Resource Virtualzation)
- 자원을 이용자의 요구에 따라 제공 (On-Demand Service)
- 사용자는 사용한 만큼만 과금 (Pay-As-You-Go)
- 빠르고 쉬운 자원 제공 (Easy & Thin Provisioning)
- 스케일 인/아웃을 자동으로 관리 (Auto Scaling)
- Downtime을 최소화하는 고가용성 (High Availability)

- 플랫폼을 이루는 각각의 서비스들은 메시지 중심으로 통신 (Message Oriented Middleware)
- 사용자가 자신의 자원을 쉽게 제어 및 관리할 수 있도록 Open API 제공 (Restful Web Service)

클라우드 환경에서의 기대효과는 다음과 같다.

- 간편하고 빠른 IT 자원 운용
  - 번거로운 설치, 계약, 운용 과정 없이 서버나 스토리지를 웹서비스로 이용
  - 일시적이거나 갑작스러운 서버, 스토리지 사용이 요구되는 상황에서 클릭만으로 추가 및 제거 기능
  - 별도의 전담 인력이나 사용 교육이 필요 없이 누구나 쉽게 이용 가능
- 탄력적이고 효율적인 시스템 관리
  - 물리적 공간이나 하드웨어에 대한 제약 없이 유연한 확장 가능
  - 물리적 IT 자원 관리 인력을 핵심 업무로 재배치해 경쟁력 제고 가능
- 비용 효율성 증대
  - 사용자 기반의 유틸리티 컴퓨팅으로 원하는 때, 원하는만큼 쓰고, 사용한 만큼 요금 지불
  - 기업 IT 인프라 내 물리적 장치를 통제함으로써 설치투자비용(CapEX)과 운용비용(OpEx) 절감

위의 조건을 만족하기 위한 클라우드 기술요소를 모두 다루기엔 시간상의 한계가 있고 기술요소는 너무나 방대하다. 따라서, 클라우드 기술 요소를 다루기 보다는 기업에서 필요한 기본적인 아키텍처에 집중하고 해당 클라우드 시스템을 구현함에 있어서 필수적으로 필요한 모니터링에 관련된 로깅 아키텍처를 다룬다.

3) 기존에 존재하는 설계 자료

i) Hadoop 기반 로깅 시스템 아키텍처

Hadoop 프로젝트와 관련된 솔루션으로 로깅 시스템을 구성하는 경우 다음과 같은 구조를 가진다. 수집을 위해서 Flume, Chukwa, Scribe를 구성하고 저장은 Hadoop 구조에 HDFS 방식을 사용한다. 그리고 그 이후 색인과 분석, 운영 관리 등은 Hadoop 서브 프로젝트와 빅데이터 분야에서 사용하는 오픈 소스 프로젝트들을 사용해서 구성한다.

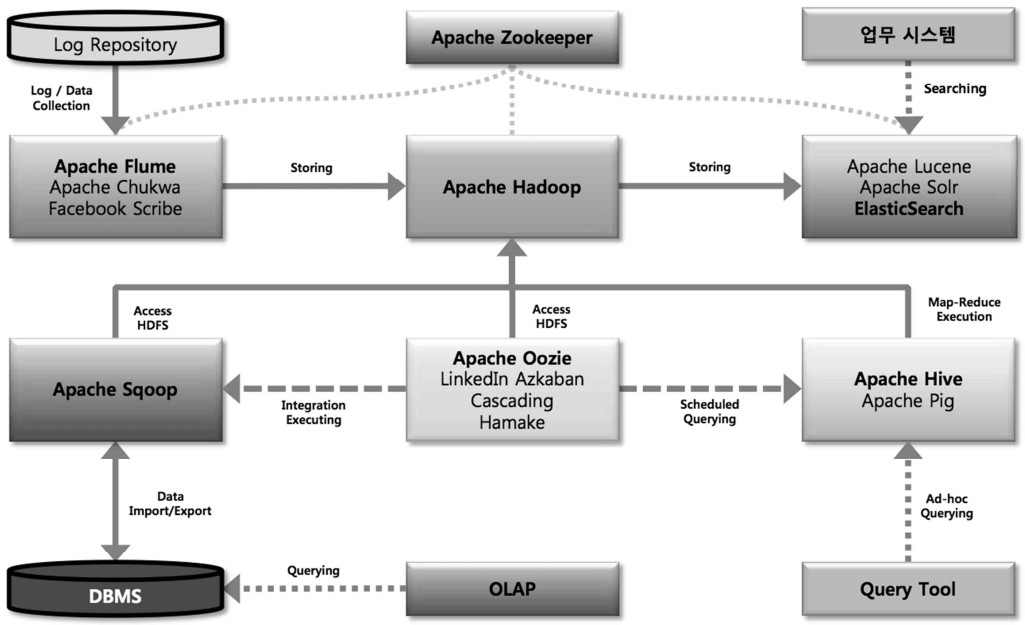


그림 16. 하둡 기반 로깅 시스템 아키텍처

이런 구성이 가능하기 때문에 클라우드와 빅데이터 도메인이 함께 성장하고 있지만, 하지만 중소기업 같은 작은 규모의 시스템에서는 쉽게 도입하기 어렵고 전체 시스템을 구성하기도 쉽지 않다.

ii) 일반 로깅 시스템 아키텍처

위의 구성과 비슷한 기능을 하는 일반적인 아키텍처를 설계하면 다음과 같다.

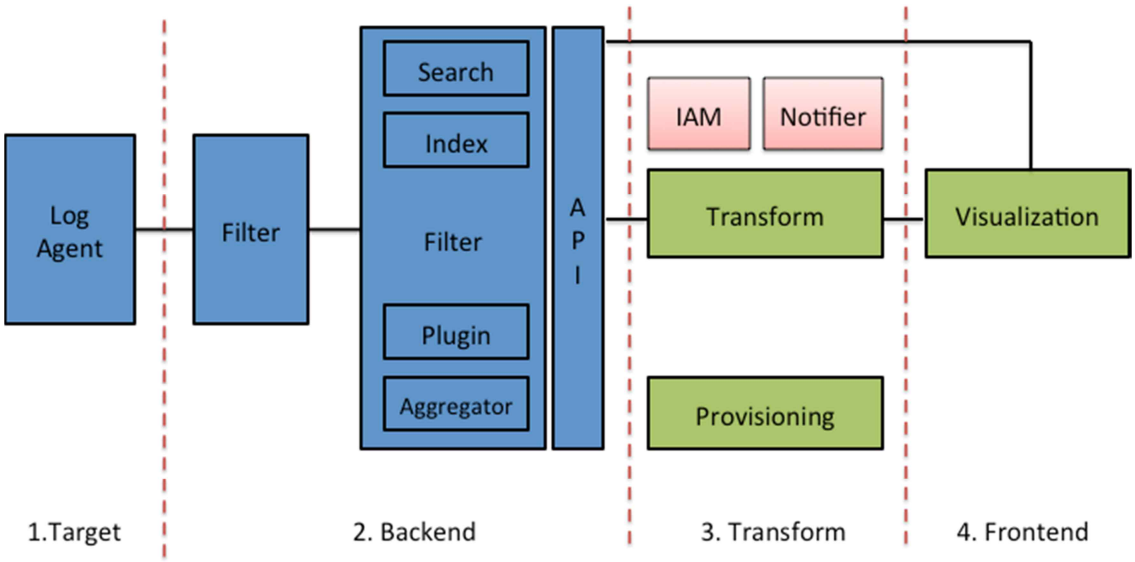


그림 17. 일반 로깅 시스템 아키텍처

위 그림에서와 같이 로깅 및 모니터링 아키텍처는 크게 4단계로 나눌 수 있다. 로그가 생성되는 단계(Target)가 첫 번째 단계다. 그리고 두 번째 단계는 생성된 로그를 수집하고, 주어진 조건에 따라 걸러내고 색인을 만들어 별도의 저장 장치에 최소한의 가공된 상태로 분산 저장한다.

세 번째 단계는 로그 데이터를 변환하고 타 시스템으로 전달하거나 상태를 알려주는 등 비즈니스 요구사항에 맞게 개발되는 단계이다. 마지막 단계는 로그 데이터를 최종적으로 사용자에게 보여주는 단계다.

본 참조 모델의 아키텍처 범위는 1, 2 단계에 해당한다.

4) 상위 레벨 기능 요구사항 목록

표 5. 상위 레벨 기능 요구사항 목록

기호	내용	중요도	구현성	합계
HFR3	가변 자원에 대한 자동 로그 수집	20	19	39
HFR1	대규모 로그의 지능적 필터링	15	14	29
HFR2	다양한 레이어 및 컴포넌트 간의 로그 연계 및 추적	11	15	26
HFR4	로그 정보 재활용을 위한 로그 표준화	10	7	17
HFR5	로그 정보의 준 실시간성	9	12	21

5) 시스템 제약사항

표 6. 제약사항 목록

기호	내용	출 처
C1	서버는 on-demand 를 만족하는 확장성을 가져야 한다.	시스템
C2	로그를 사용하는 비즈니스마다 요구사항의 상이함에 따라서 구현 방식이 달라져야 함	비즈니스
C3	기본 파일 시스템 수준 연동과 이식성을 위해 여러 개발 환경과 언어를 사용해야 함	이식성
C4	본 단계에서는 분석을 제외한 데이터 수집 단계만 다뤄야 함	비즈니스

6) 위험 요소

- 본 실험 데이터는 실제와는 거리가 있음을 인지한다.
- 로그를 통해서 얻고자 하는 것이 명확하지 않음에 따른 로그 포맷의 비 적절성
- 로그를 기록함에 따른 성능 저하 영향 최소화

7) 제품의 확장 범위

- 클라우드 기반 기술로 Scale In/Out을 고려하여 중소기업의 작업 시스템 환경에서 대형 클라우드 서비스 환경까지 확장이 가능함
- 빅 데이터 솔루션과 연계 가능
- 로그와 함께 시스템 경고 등의 모니터링 정보의 수집의 확장

라. 중요 기능 요구사항 식별

1) 시스템 상태 모니터링을 위한 근거 제공

- 시스템의 상태 측정 자료 제공
- 장애 발생이 예상되는 문제에 대한 사전 경고를 위한 근거 데이터 제공
- 로그 통계 및 지연병목 구간 모니터링을 위한 데이터 제공
- 잠재적인 서비스 장애 발생에 대한 사전 예방 방법 제공
- 서비스 장애 발생 시 그 원인을 신속히 파악할 수 있는 방법 제공
- 시스템의 장애 유무와 관계 없이 시스템 실행시는 항상 로깅할 수 있는 방법 제공
- 네트워크, 하드웨어, 시스템의 성격과 관계 없는 독립적인 로깅 모듈 제공

2) 로그 정보의 준 실시간성 수집 기능 제공

- 로그의 준 실시간 수집을 위한 가벼운 데이터 포맷을 사용
- 보다 빠른 네트워크 프로토콜을 사용
- 로그 검색 방법을 제공함으로써, 시스템 상태 및 장애발생 원인에 대한 쉽고 빠른 원인 분석 방법 제공

3) 다양한 레이어 및 컴포넌트 간의 로그 연계 및 추적

- 레이어에 관계 없는 로그 기록 기능
- 컴포넌트 간의 로그 추적성을 위한 데이터 표준 수립

4) 로그 정보 재활용을 위한 로그 표준화

- 수집 및 분석을 위한 로그 포맷 표준화
- 로그를 남기는 모듈/컴포넌트에 API 제공과는 관계 없이 실제 로그 모듈에서 포맷을 결정할 수 있는 설정값 제공
- 로거를 도입하려고 하는 시스템의 로그 표준화를 위한 Custom 기능 제공

5) 가변 자원에 대한 자동 로그 수집.

- 시스템의 확장에 따라 로그 수집 Agent 의 설정 기능
- Log Server 의 Agent 와의 통신의 자동화

6) 클라이언트 모니터링 대시보드

- 클라이언트에서 분 단위 서버 로그를 실시간으로 분석하는 기능
- 서버의 장애, 위험, 알람 통보 기능
- 다양한 차트로 실시간 서버 상태를 표현하는 기능

7) 기능요구사항 우선순위화

표 7. 중요 기능 요구사항 목록

기호	내용	중요도	구현성	합계
FR1	시스템 상태 모니터링	19	18	37
FR3	장애 추적	17	17	34
FR2	확장가능하도록 설정 제공	15	17	32
FR4	로그 연계 추적기능	13	15	28
FR6	클라이언트 모니터링 대시보드	10	9	19
FR5	로그 정보의 준 실시간성 제공	10	8	18

- 중요도가 높은 순위로 기능 우선 순위 부여
- 중요도는 중요하다고 생각되는 부분을 투표 했으며 구현성은 구현이 어려운 부분에 투표함

마. 핵심 품질속성 식별

해당 기능 요구사항 관련하여 다음과 같은 품질 요구사항을 도출할 수 있다.

1) 성능

하드웨어 및 솔루션간 데이타 교환의 성능을 보장해야 한다.

2) 사용성

서버확장에 따른 로그 컴포넌트 적용이 편리해야 한다.

3) 유연성

수집기의 시스템 확장을 고려해야 한다.

4) 통합성

이기종 시스템 간의 물리적인 통합이 용이해야 하며 이를 고려한 아키텍처여야 한다.

5) 신뢰성

최소 1/xxx 정도의 오차율로 데이타 취합이 가능해야 한다.

6) 재사용성

각 모듈들의 콤포넌트화로 커스터마징을 함에 있어 재사용성을 확보해야 한다.

7) 가용성

24*365 매시간 가용한 것을 기본 정책으로 한다.

8) 핵심 품질속성 목록

표 8. 핵심 품질속성 목록

내용	관련 항목	중요도	구현성	합계
성능	HFR5, FR1, C1	16	20	36
유연성	HFR4, FR2, C3	15	18	33
통합성	HFR4, FR2, FR6, C3	13	15	28
신뢰성	FR3,	11	10	21
가용성	HFR3, C1	8	8	16
재사용성	HFR4, FR2, C3	6	7	13
사용성	HFR3, FR6, C1	4	6	10

- 중요도가 높은 순위로 우선 순위 부여

9) 품질 속성별 세부 기능

각 품질 요구사항에 따른 기능 및 제약사항을 정리하면 다음과 같다.

i) 성능

- 인터페이스와 서버간의 통신 패킷량을 줄여야 한다.
- 컴포넌트간 로그 교환시 일어나는 데이타량을 감소시켜야 한다.
- 로그 데이타를 전달하는 데 있어서 준 실시간성의 원칙을 가져야 한다.

ii) 사용성

- 사용자는 로그 콤포넌트를 적용함에 있어서 신속하고 편리하게 사용가능해야 한다.
- 서버확장에 따른 로그 콤포넌트 적용이 편리해야 한다.
- 로그 콤포넌트를 통해 수집된 로그는 사용하기 편한 데이타 포맷이어야 한다.

iii) 유연성

- 시스템의 성격에 관계 없이 Log Component 를 적용하기 쉬워야 한다.
- 장애 및 비즈니스간의 정합성 개선에 있어 유용한 정보를 제공할 수 있는 유연한 포맷이어야 한다.
- 시스템 확장을 고려한 콤포넌트 설계를 해야 한다.
- 수집기의 시스템 확장이 고려 되어야 한다.

iv) 통합성

- 이기종 시스템 간의 물리적인 통합이 용이해야 하며 이를 고려한 아키텍처
- 이기종 시스템간의 상이한 프로토콜의 통합
- 확장되는 시스템의 로그가 문제 없이 통합되어야 한다.
- 로그 수집기에서 로그의 통합에 대한 고려가 되어야 한다.
- 로그 Agent 와 업무 시스템간의 통합은 라이브러리 추가나 삭제만큼 자연스러워야 한다.

v) 신뢰성

- 최소 1/xxx 정도의 오차율로 데이타 취합이 가능해야 한다.
- 신뢰성이 특별하게 요구되는 시스템과 약간의 데이타 유실이 관계 없는 시스템의 성격에 따라 적용 가능한 설정이 가능해야 한다.
- 각 컴포넌트 구동시 로그 수집에 대한 안정성 보장이 되어야 한다.

vi) 재사용성

- 각 모듈들의 컴포넌트화로 커스터마징을 함에 있어 재사용성이 확보 되어야 한다.
- 시스템에 적용함에 있어 플러그 인 형태로 제공되어야 한다.
- 그 외에 다른 스크립트들 역시 시스템의 환경에 대한 의존성을 최소화한다.

vii) 가용성

- 24*365 매시간 가용한 것을 기본 정책으로 한다.
- 로거의 실시간 사용이 가능해야 한다.
- 로거의 장착/미장착 이 시스템에 영향없이 자유로워야 한다.
- Logger 의 장착 여부가 Target Application 에 영향을 미치지 않고 시스템은 항상 가용해야 한다.

바. 품질속성 시나리오 목록

1) 유연성 파생 시나리오

표 9. 유연성 파생 시나리오 1

항목	내용
시나리오	클라우드 기반의 정보시스템에서 시스템 확장으로 인하여 운영중인 시스템과 로깅 서버에 시스템 신규 추가/확장으로 인한 확장이 보장되어야 한다.
자극 유발원 (Source)	정보 시스템
자극 (Stimulus)	시스템의 추가 및 클라우드 환경에서의 시스템 Scale-out 등의 확장
대상체 (Artifact)	운영중인 시스템 또는 로깅 서버
환경 (Environment)	클라우드 환경 및 운영환경
응답 (Response)	시스템 신규 추가 및 확장시 로깅 컴포넌트를 추가하여 기존 로그 수집에 통합되도록 한다.
응답 측정 (Response Measure)	로그가 제대로 되는지를 검사한다.

표 10. 유연성 파생 시나리오 2

항목	내용
시나리오	시스템 증설이 필요하다
자극 유발원 (Source)	기존 운영시스템과 신규로 추가되는 시스템
자극 (Stimulus)	Scale-out 요건으로 인한 시스템 추가 증설
대상체 (Artifact)	운영중인 시스템, 신규시스템 , 로깅 서버
환경 (Environment)	클라우드 환경 및 운영환경
응답 (Response)	시스템 신규 추가 및 확장시 기존 로그 수집에 통합
응답 측정 (Response Measure)	신규 시스템의 추가로 인한 로그 수집이 기존 로그 수집 시스템에 수집되어야 한다.

2) 성능 파생 시나리오

표 11. 성능 파생 시나리오 1

항목	내용
시나리오	클라우드 환경 및 운영환경에서 시스템 로그 이벤트 발생시 데이타 교환에 따른 패킷 최소화와 데이타 교환의 효율성을 통해서 로그 데이타 저장의 성능을 보장한다.
자극 유발원 (Source)	시스템 로그
자극 (Stimulus)	수집할 필요성이 있는 app log 및 시스템 로그
대상체 (Artifact)	로그 에이전트 및 컬렉터
환경 (Environment)	클라우드 환경 및 운영환경
응답 (Response)	시스템 로깅을 수행함에 있어서 데이타 교환에 따른 패킷을 최소화하고 성능이 보장되도록 한다.
응답 측정 (Response Measure)	로그 데이타 교환 및 저장에 따른 속도 측정

표 12. 성능 파생 시나리오 2

항목	내용
시나리오	지속적으로 확장되는 시스템의 추가에 따라 로그 메시지의 폭팔적 증가
자극 유발원 (Source)	각 시스템에서 생성하는 로그 메시지
자극 (Stimulus)	로그 메시지의 Traffic
대상체 (Artifact)	로그 에이전트 및 컬렉터
환경 (Environment)	클라우드 환경 및 운영환경
응답 (Response)	발생하는 로그에 대해서 저장을 보장하고 시스템에 영향을 미치지 않음
응답 측정 (Response Measure)	로그 데이타 교환 및 저장에 따른 속도 측정

3) 통합성 보장 파생 시나리오

표 13. 통합성 보장 파생 시나리오 1

항목	내용
시나리오	이기종 시스템에서 로그 메시지의 물리적 통합에 문제 없이 통합되기 위해서 프로토콜 통합 및 메시지 포맷 표준 준수를 통해서 로그 에이전트와 컬렉터를 통한 로그 메시지 수집하고 통합을 할 수 있어야 한다.
자극 유발원 (Source)	이기종 시스템간 물리적 통합
자극 (Stimulus)	시스템에 로그 적용
대상체 (Artifact)	운영 시스템, 로깅 에이전트, 로깅 컬렉터
환경 (Environment)	운영되는 이기종 시스템
응답 (Response)	기존 운영되던 시스템의 로그와 새롭게 추가가 되는 시스템 로그들간의 문제 없는 통합, 프로토콜 통합, 메시지 포맷 표준 준수
응답 측정 (Response Measure)	각 시스템에서의 로그가 문제 없이 통합되어야 한다.

표 14. 통합성 보장 파생 시나리오 2

항목	내용
시나리오	신규로 추가되는 시스템 노드는 플랫폼에 관계 없이 로그 수집이 통합된다.
자극 유발원 (Source)	신규로 추가되는 이기종 시스템 노드
자극 (Stimulus)	신규 추가 노드의 로그 발생
대상체 (Artifact)	운영 시스템, 로깅 에이전트, 로깅 컬렉터
환경 (Environment)	운영되고 있던 시스템노드와 신규 추가 시스템 노드
응답 (Response)	기존 운영되던 시스템의 로그와 새롭게 추가가 되는 시스템 로그들간의 문제 없는 통합
응답 측정 (Response Measure)	각 시스템에서의 로그가 문제 없이 통합되어야 한다.

사. 시나리오 우선순위화

1) 우선순위화 결과

표 15. 우선순위화된 시나리오 목록

기호	내용	중요도	구현성	합계
S3	클라우드 환경 및 운영환경에서 시스템 로그 이벤트 발생시 데이타 교환에 따른 패킷 최소화와 데이타 교환의 효율성을 통해서 로그 데이타 저장의 성능을 보장한다.	5	5	10
S1	클라우드 기반의 정보시스템에서 시스템 확장으로 인하여 운영중인 시스템과 로깅 서버에 시스템 신규 추가/확장으로 인한 확장이 유연해야 한다.	5	3	8
S2	이기종 시스템에서 로그 메시지의 물리적 통합에 문제 없이 통합되기 위해서 프로토콜 통합 및 메시지 포맷 표준 준수를 통해서 로그 에이전트와 컬렉터를 통한 로그 메시지 수집하고 통합을 할 수 있어야 한다	5	3	8

2) 상위 우선 순위 시나리오 정제

i) 핵심 시나리오 정제 목록

(a) 1순위 - 성능 정제 시나리오

표 16. 성능 정제 시나리오

항목	내용
시나리오	클라우드 환경 및 운영환경에서 시스템 로그 이벤트 발생시 데이타 교환에 따른 패킷 최소화와 데이타 교환의 효율성을 통해서 로그 데이타 저장의 성능을 보장한다.
자극 유발원 (Source)	수집할 필요성이 있는 app log 및 시스템 로그
정제 1	application 및 system 에서 업무 수행시 혹은 시스템 모니터링에 대한 이벤트 발생
자극 (Stimulus)	수집할 필요성이 있는 app log 및 시스템 로그
정제 2	이벤트가 발생하고 해당 이벤트에 대해서 로그 에이전트를 통해서 application 및 system 은 로깅을 요청한다.
대상체 (Artifact)	로그 에이전트 및 컬렉터
정제 3	로그 에이전트는 로그 컬렉터에 로그를 전달한다.
환경 (Environment)	클라우드 환경 및 운영환경.
정제 4	시스템 운영환경은 확장이 자유로운 클라우드 기반 시스템으로 가정한다.
응답 (Response)	시스템 로깅을 수행함에 있어서 데이타 교환에 따른 패킷을 최소화하고 성능이 보장되도록 한다
정제 5	데이타 교환 프로토콜에 따라 로그 메시지 로깅이 잘 되고 있는지 조회할 수 있어야 한다.
응답 측정 (Response	로그 데이타 교환 및 저장에 따른 속도 측정

Measure)	
정제 6	최종 저장되는 target 이 되는 repository 를 조회하여 시스템/app 로깅이 잘 되는지 검사한다.

(b) 2순위 - 유연성 정제 시나리오

표 17. 유연성 정제 시나리오

항목	내용
시나리오	클라우드 기반의 정보시스템에서 시스템 확장으로 인하여 운영중인 시스템과 로깅 서버에 시스템 신규 추가/확장으로 인한 확장이 유연해야 한다.
자극 유발원 (Source)	정보시스템
정제 1	운영중인 정보 시스템
자극 (Stimulus)	시스템의 추가 및 클라우드 환경에서의 시스템 Scale-out 등의 확장
정제 2	클라우드 기반의 시스템에서 시스템의 자원이 부족하여 시스템 확장이 되거나 임계점이 되어서 시스템을 축소
대상체 (Artifact)	운영중인 시스템 또는 로깅 서버
정제 3	운영중인 시스템에 영향을 주지 않고 새롭게 확장되는 시스템에 로그 에이전트 추가가 된다.
환경 (Environment)	클라우드 환경 및 운영환경
정제 4	Scale-out 이 자유로운 클라우드 기반의 운영환경
응답 (Response)	시스템 신규 추가 및 확장시 로깅 컴포넌트를 추가하여 기존 로그 수집에 통합되도록 한다.
정제 5	시스템 확장시 로그 에이전트가 제대로 동작해야 하고 로그 컬렉터는 해당 시스템의 로그를 받아서 처리되어야 한다.
응답 측정 (Response)	로그 작성 여부를 검사한다.

Measure)	
정제 6	로그 컬렉터를 통해서 타겟 repository 에 저장되었을 때 새롭게 확장된 시스템에서 전달된 로그 인지를 구분할 수 있는 식별자를 통하여 검증.

3. 설계 뷰

가. 시스템 아키텍처 드라이버 식별

표 18. 시스템 아키텍처 드라이버 목록

내용	관련 항목	중요도	구현성	합계
성능	HFR5, FR1, C1	16	20	36
유연성	HFR4, FR2, C3	15	18	33
통합성	HFR4, FR2, FR6, C3	13	15	28
신뢰성	FR3,	11	10	21
가용성	HFR3, C1	8	8	16
재사용성	HFR4, FR2, C3	6	7	13
사용성	HFR3, FR6, C1	4	6	10

나. 아키텍처 패턴

1) 후보 패턴 및 설계전술

우선 1차에서는 전체 시스템 구조를 설계하기 위한 후보 패턴을 정리

표 19. 아키텍처 후보 패턴

후보패턴	세부 설계전술	관련 품질 요구사항	관련 드라이버	트레이드오프
2-tier 패턴	수집 클라이언트 + 저장 서버 구조	유연성, 확장성	유연성, 통합성	유연성-, 통합성+
3-tier 패턴	수집 클라이언트 + 전달 + 저장 서버	유연성, 확장성	유연성, 통합성	유연성++,통합성-
	수집 클라이언트 + 전달*n + 저장*n	유연성, 확장성	유연성, 통합성	유연성++,통합성+

2) 선정 패턴 정리

i) 2-tier 패턴

표 20. 2-tier 패턴

드라이버	2-tier 패턴			
	강점	약점	트레이드오프	선정사유
유연성		확장이 어려움	-	
통합성	구조가 간단해서 구현과 시스템 통합에 편리함		+	서버와 뷰 클라이언트 구조에 적합함

ii) 3-tier 패턴

표 21. 3-tier 패턴

드라이버	3-tier 패턴			
	강점	약점	트레이드오프	선정사유
유연성	전달 계층이 있어 (비교적) 확장에 용이함		+	
통합성	네트워크 환경이나 시스템 구성이 다른 구조도 연동 가능함	관리 포인트가 늘어나며 일부 성능이 제한됨	+	에이전트와 서버 구조에 적합함

다. 아키텍처 뷰 (1차)

1) 시스템 개념도

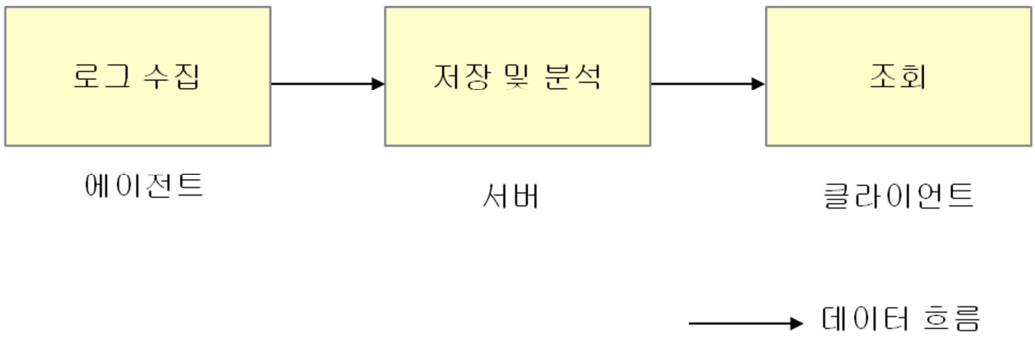


그림 18. 시스템 개념도

- 분산환경에서 로그를 수집하고 보여주는 시스템이다.
- 에이전트에서 로그를 수집한다.
- 서버는 로그는 분석하고 저장한다.
- 클라이언트는 로그를 화면에 보여준다.

2) 모듈뷰

i) 분할

그림 19. 모듈뷰 - 분할

- 4개의 모듈로 구성된다.
- 중계서버는 분산환경을 위해 1차 부터 고려한다.

ii) 일반화

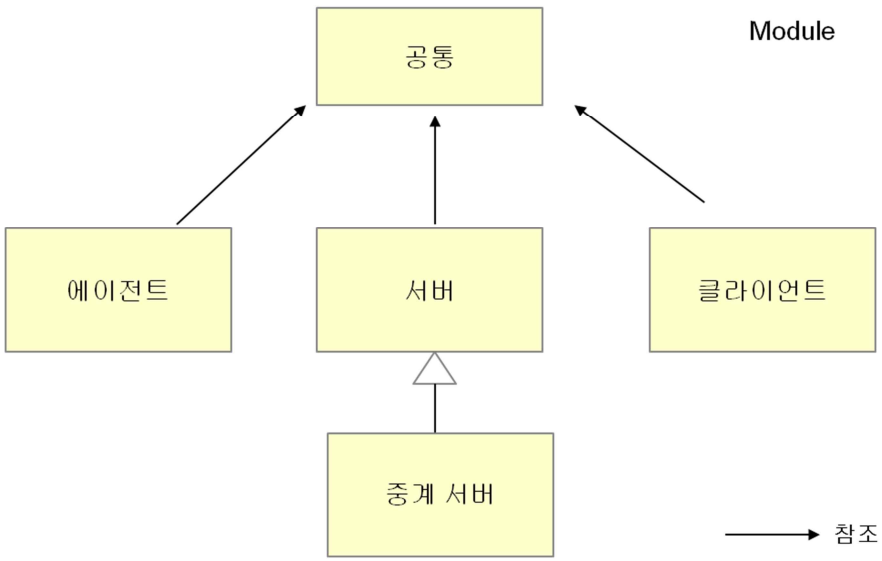


그림 20. 모듈뷰 - 일반화

- 중계 서버는 서버의 모듈들을 상속한다.

iii) 사용

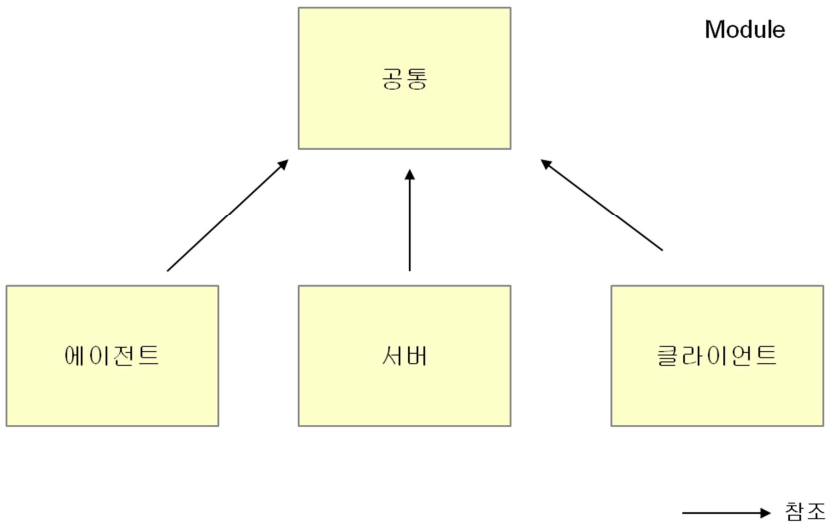


그림 21. 모듈뷰 - 사용

- 에이전트는 로그를 로딩하고 전송한다.
- 서버는 로그를 분석하여 실시간 통계를 만들고 저장한다.
- 클라이언트는 로그를 화면에 보여준다.
- 공통은 에이전트/서버/클라이언트를 위한 공통 라이브러리를 포함한다.,

3) C&C뷰

i) 2-tier C/S 구조

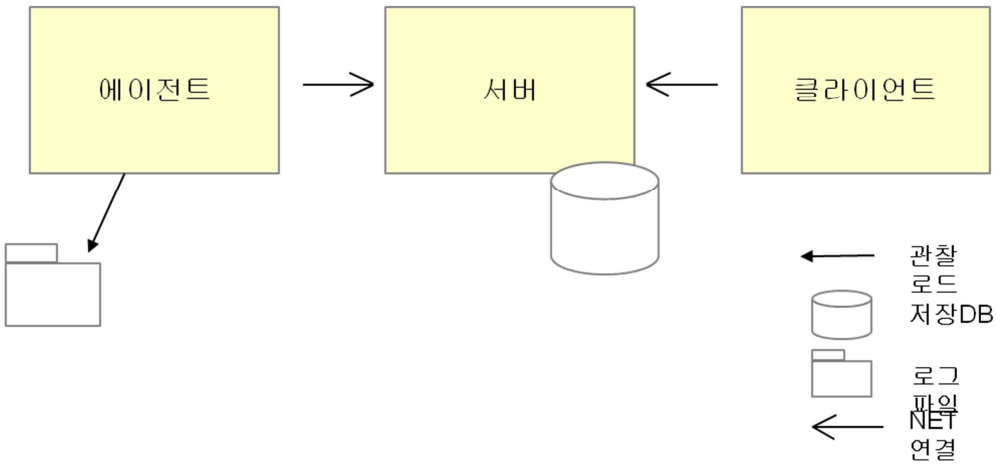


그림 22. 2-tier C/S **구조**

- 에이전트는 로그파일을 관찰한다.
- 에이전트는 로그를 로딩하고 네트워크에 연결된 서버로 전송한다.
- 서버는 로그를 분석하고 적재한다.
- 클라이언트는 저장된 로그를 보여준다.

ii) 3-tier 확장 구조

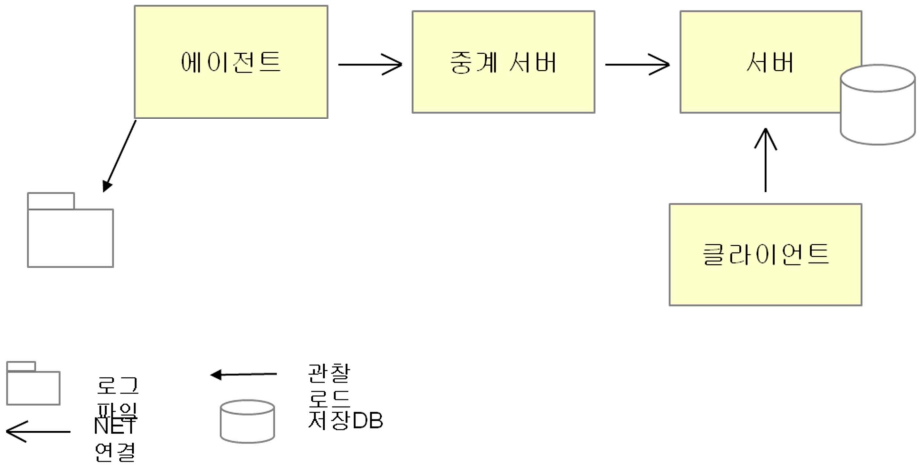


그림 23. 3-tier 구조

- 에이전트는 로그파일을 관찰한다.
- 에이전트는 로딩되 로그를 네트웍을 통해 서버에 전송한다.
- 서버는 로그를 분석하고 적재한다.
- 클라이언트는 저장된 로그를 보여준다.
- 중계 서버는 데이터를 서버에 중계한다.

iii) 3-tier Scale 확장 구조

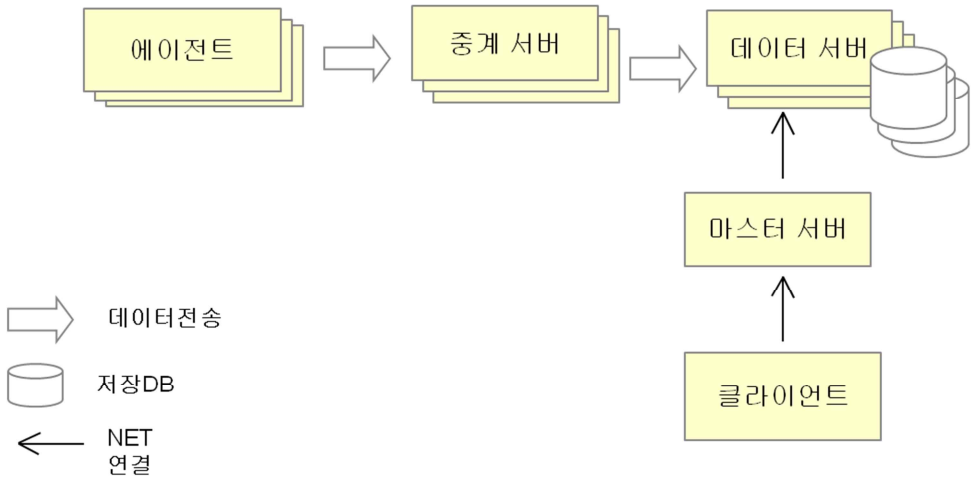


그림 24. 3-tier Scale 확장 구조

- 시스템의 규모가 커지면 서버는 마스터 서버와 데이터 서버로 분리될 수 있다.
- 데이터 서버는 중계 서버로 부터 데이터를 받거나 에이전트로 부터 데이터를 받아 저장한다.
- 마스터 서버는 여러개의 데이터 서버에서 동시에 데이터를 조회하여 클라이언트에게 보여준다.
- 클라이언트는 마스터 서버만을 바라본다.

4) 할당 뷰

i) 배치

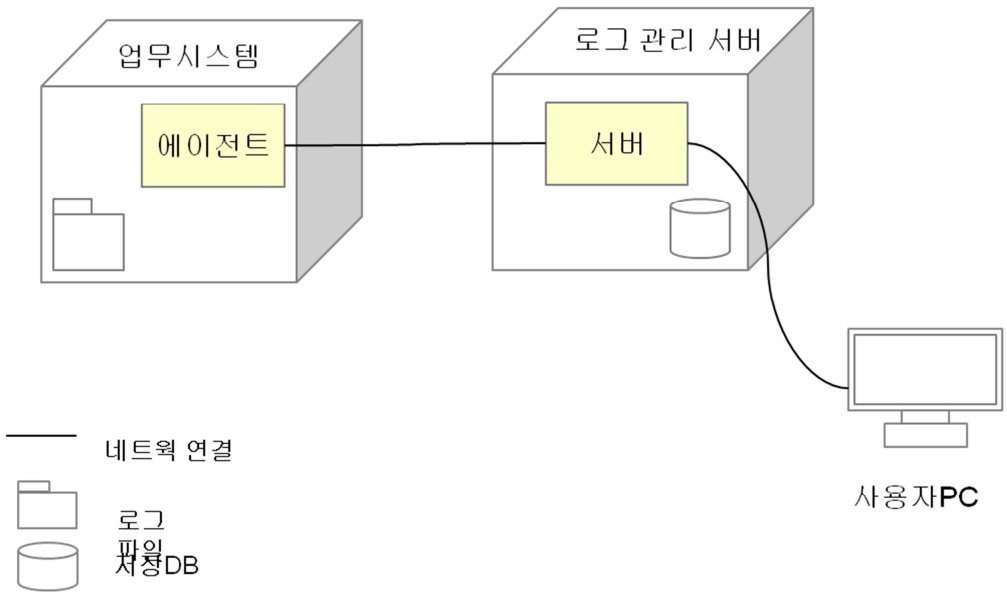


그림 25. 할당뷰 - 배치

- 에이전트는 로그 파일이 있는 업무시스템에 설치한다.
- 로그 관리 서버는 별도의 H/W를 사용한다.
- 저장 DB는 로그 관리서버에 서버(프로세스)와 같이 위치한다.
- 사용자는 PC에서 클라이언트 화면으로 데이터를 조회한다.

ii) 구현

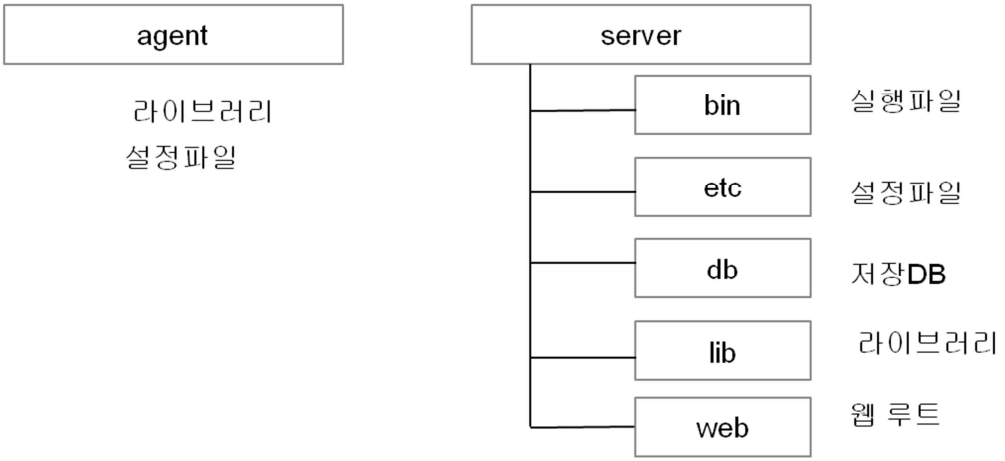


그림 26. 할당뷰 - 구현

- 에이전트는 하나의 디렉토리에 라이브러리와 설정을 모두 설치한다.
- 서버는 bin, etc, db, lib, web의 5개 디렉토리로 분할하여 설치한다.

라. 아키텍처 뷰 (2차)

2차 설계에서는 시스템 각 구성 요소별 아키텍처를 설계하는 것을 목표로 함

1) 아키텍처 드라이버 식별

i) 에이전트 아키텍처 드라이버

표 22. 에이전트 아키텍처 드라이버 목록

내용	관련 시스템 항목	중요도	구현성	합계
성능	HFR5, FR1, C1	14	8	22
통합성	HFR4, FR2, C3	16	9	25
사용성	HFR3, C1	9	8	17

ii) 서버 아키텍처 드라이버

표 23. 서버 아키텍처 드라이버 목록

내용	관련 항목	중요도	구현성	합계
성능	HFR5, FR1, C1	17	10	27
신뢰성	FR3	16	14	30
가용성	HFR3, C1	14	10	24
재사용성	HFR4, FR2, C3	8	8	16

iii) 네트워크 아키텍처 드라이버

표 24. 네트워크 아키텍처 드라이버 목록

내용	관련 항목	중요도	구현성	합계
성능	HFR5, FR1, C1	16	10	26
유연성	HFR4, FR2, C3	13	15	28
통합성	HFR4, FR2, FR6, C3	9	9	18

iv) 클라이언트 아키텍처 드라이버

표 25. 클라이언트 아키텍처 드라이버 목록

내용	관련 항목	중요도	구현성	합계
통합성	HFR4, FR2, FR6, C3	9	8	17
가용성	HFR3, FR6, C1	16	12	28
재사용성	HFR4, FR2, FR6, C3	7	9	16
사용성	HFR3, FR6, C1	20	15	35

2) 아키텍처 패턴

i) 에이전트 후보 패턴 및 설계전술

표 26. 에이전트 아키텍처 후보 패턴

후보패턴	세부 설계전술	관련 품질 요구사항	관련 드라이버	트레이드오프
Polling 패턴	로그 파일을 주기적 감시	성능, 신뢰성, 사용성, 유연성	성능, 통합성	성능-, 통합성++
구독/발행 패턴	로그 변화 감지와 전송	유연성, 사용성	성능,사용성	성능+, 사용성+
플러그인 패턴	에인전트를 특정 모듈에 삽입하는 구조	사용성, 신뢰성	유연성,사용성	성능-, 사용성++

ii) 서버 후보 패턴 및 설계전술

표 27. 서버 아키텍처 후보 패턴

후보패턴	세부 설계전술	관련 품질 요구사항	관련 드라이버	트레이드오프
비동기 패턴	비동기 방식 소켓 처리 방식	성능, 신뢰성	성능, 통합성	성능++, 통합성-
아답터 패턴	서버 로직과 아답터를 연결하는 방식	성능, 유연성	성능, 유연성	성능-, 유연성++
멀티쓰레딩 큐 패턴	기능 처리별 쓰레드와 큐 구성	성능, 가용성	성능, 가용성	성능+, 가용성+
NoSQL DB	로그 저장과 인덱싱 처리	성능, 신뢰성	성능, 가용성	성능+, 가용성-
File DB	로그 저장과 태깅, 인덱싱 처리	성능, 신뢰성	성능, 가용성	성능++, 가용성+

iii) 네트워크 후보 패턴 및 설계전술

표 28. 네트워크 아키텍처 후보 패턴

후보패턴	세부 설계전술	관련 품질 요구사항	관련 드라이버	트레이드오프
멀티 프로토콜	UDP/TCP/HTTP 방식을 모두 지원	성능, 유연성	성능, 통합성	성능+, 통합성+
단일 프로토콜	UDP 만 지원	성능, 유연성	성능, 통합성	성능++, 통합성+
	TCP 만 지원	성능, 유연성	성능, 통합성	성능+, 통합성-
고수준 프로토콜	Apache Thrift 지원	성능, 유연성	성능, 통합성	성능-, 통합성+

iv) 클라이언트 후보 패턴 및 설계전술

표 29. 클라이언트 아키텍처 후보 패턴

후보패턴	세부 설계전술	관련 품질 요구사항	관련 드라이버	트레이드오프
2-tier 클라언트서버 패턴	HTTP 웹 브라우저/서버	사용성, 통합성	사용성, 통합성	사용성+, 통합성+
MVC 패턴	차트와 그래프 처리하는 뷰 분리	재사용성, 가용성	사용성, 재사용성	사용성+, 재사용성+

v) 선정 패턴 정리

표 30. 에이전트 Polling 패턴

드라이버	에이전트 – Polling 패턴			
	강점	약점	트레이드오프	선정사유
성능	주기적으로 처리	데이터 처리가 비교적 늦음.	-	
통합성	안정적이고 연동처리 및 구현이 쉬움		++	에이전트 로그 처리 구조에 적합함

표 31. 서버 비동기 패턴

드라이버	서버 – 비동기 패턴			
	강점	약점	트레이드오프	선정사유
성능	시스템 자원 활용이 높고 효율이 좋음	구현이 복잡하고 어려움	++	효율적인 서버 소켓 처리 구조에 적합함
통합성		타모듈 연동에 따라 복잡도 증가함	-	

표 32. 서버 아답터 패턴

드라이버	서버 – 아답터 패턴			
	강점	약점	트레이드오프	선정사유
성능		아답터 자체가 오버헤드가 됨	-	
유연성	스크립트 언어로 서버 로직을 변경할 수 있음		++	서버 로직의 유연성을 위한 구조로 적합함

표 33. 서버 멀티쓰레딩 패턴

드라이버	서버 – 멀티쓰레딩 패턴			
	강점	약점	트레이드오프	선정사유
성능	성능 향상	복잡도 증가	+	
가용성	효율성이 높아서 가용성도 좋아짐		+	서버 데이터 처리 구조에 적합함

표 34. 서버 파일 DB

드라이버	서버 – 파일 DB			
	강점	약점	트레이드오프	선정사유
성능	파일 구조라 로직이 단순하고 빠름		++	
가용성	에러율이 낮음	파일이 많아질	+	서버 데이터

		경우 노드가 심해질 수 있음		저장소 구조에 적합함

표 35. 네트워크 멀티프로토콜 패턴

드라이버	네트워크 - 멀티프로토콜 패턴			
	강점	약점	트레이드오프	선정사유
성능	성능에 맞는 프로토콜 사용	복잡도 증가함	+	
통합성	환경에 맞춰 사용가능한 프로토콜 선택 가능	성능 저하가 될 수 있음	+	안정적이고 효율적인 네트워크 전송을 위한 구조에 적합함

표 36. 클라이언트 2-tier 패턴

드라이버	클라이언트 - 2-tier 패턴			
	강점	약점	트레이드오프	선정사유
사용성	클라이언트 구조가 단순하고 사용이 간편함	구조적 단순함으로 서버에 종속적임	+	웹 방식으로 간단하게 조회하는 클라이언트 구조에 적합함
통합성		서버 확장에 제약이 있음	+	

표 37. 클라이언트 MVC 패턴

드라이버	클라이언트 - MVC 패턴			
	강점	약점	트레이드오프	선정사유
사용성	뷰를 분리해서 처리 구조가 간편함	컨트롤러가 방대해질 수 있음	+	클라이언트 로직 구조에 적합함
재사용성	뷰와 모델을 재사용할 수 있음		+	

3) 시스템 개념도

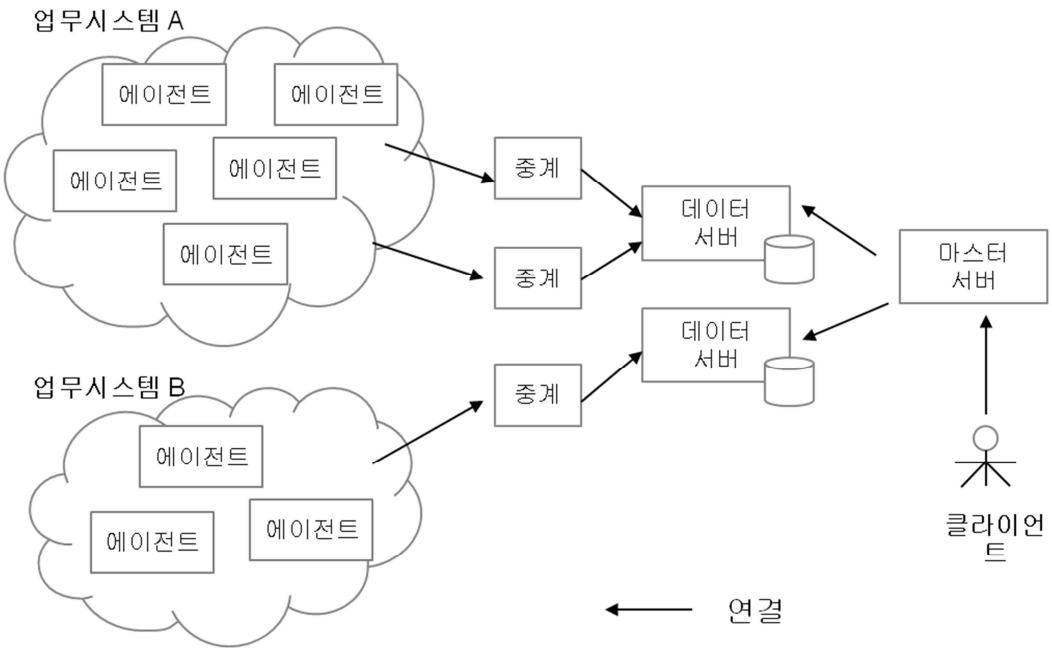


그림 27. 시스템 개념도

- 에이전트는 로그를 데이터 서버에 바로 전송하거나 중계서버를 거쳐 전송할 수도 있다
- 중계서버는 여러개가 사용될수 있다.
- 데이터서버는 데이터를 저장하고 가공한다.
- 마스터서버는 클라이언트의 요청을 받아 여러 데이터서버로부터 데이터를 조회한다.

4) 모듈 뷰

i) 분할

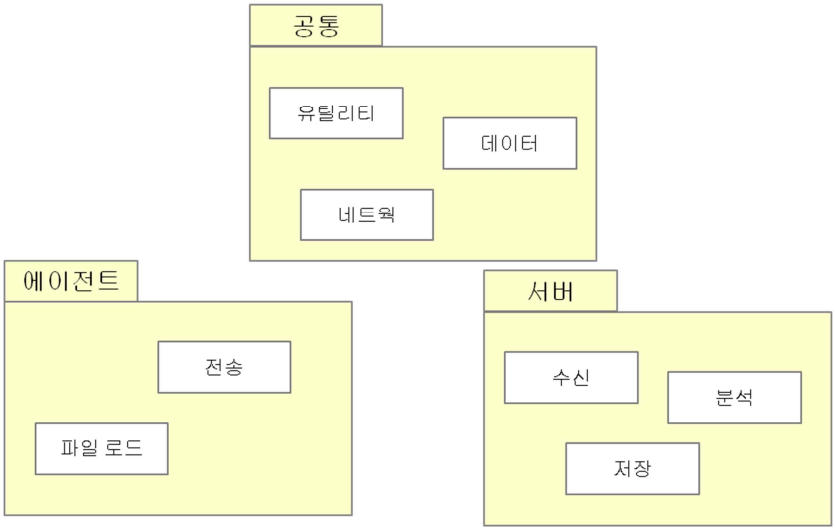


그림 28. 모듈뷰 - 분할

- 공통
  - 에이전트/서버에서 모두 사용되어야 하는 로직을 포함한다.
  - 시스템에서 사용하는 공통 라이브러리(유틸리티)를 포함한다.
  - 데이터 전송시 필요한 공통 클래스를 포함한다.
  - 네트웍 처리로직을 포함한다.
- 에이전트
  - 파일을 로딩하는 로직을 포함한다.
  - 로딩된 데이터를 전송하는 로직을 포함한다.
- 서버
  - 에이전트가 보낸 로그를 수신하기 위한 로직을 포함한다.
  - 조회를 위해 데이터 통계를 만들거나 분석하는 로직을 포함한다.
  - 로그 데이터를 로드 DB에 저장하는 로직을 포함한다.

ii) 일반화

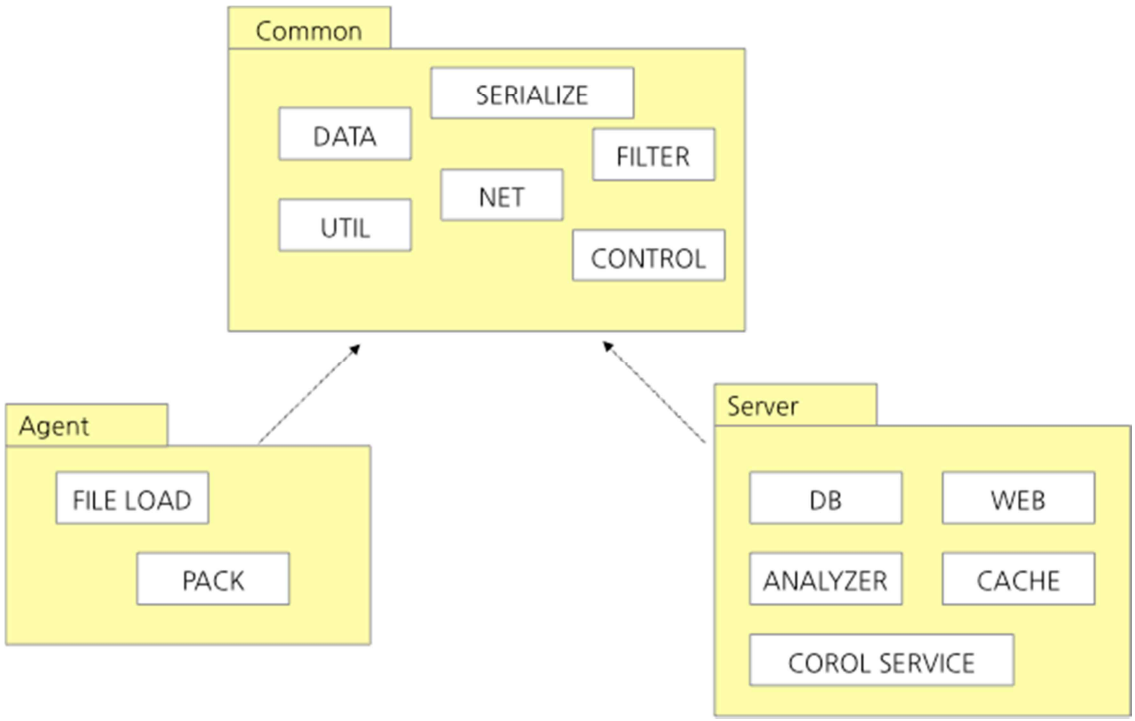


그림 29. 모듈뷰 – 일반화

(a) 일반 (Common) 모듈

- 에이전트와 서버의 공통 기능을 일반화한 모듈
- 데이터 처리 블록, 직렬화 블록, 네트워크 송수신 블록, 데이터 필터 블록, 큰트롤 블록, 유틸리티 블록 등을 포함한다

(b) 에이전트(Agent) 모듈

- 파일 수집해서 데이터 처리 블록과 연계한다.
- 버킷 데이터 생성하고 처리하는 블록이 필요하다.

(c) 서버(Server) 모듈

- 데이터, 웹서비스, 로그 데이터 분석과 캐시, 설정 블록 등을 포함한다.

- 중계/마스터 서버 역할을 동시에 포함할 수 있다.

5) 핵심 기능별 아키텍처

i) 공통 유틸리티 모듈

모든 모듈에서 같이 사용하는 공통 유틸리티 기능에 대한 구조는 다음과 같다.

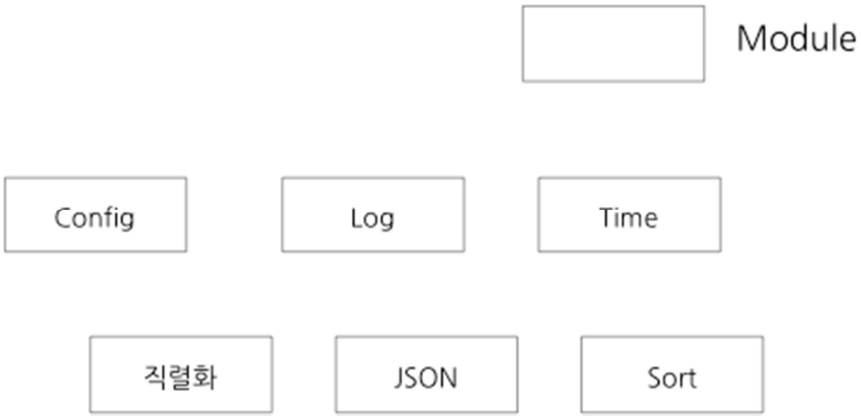


그림 30. 공통 유틸리티 모듈

- Config : 에이전트 동작을 위해 필요한 설정정보를 관리한다.
- Log : 에이전트 수행을 디버깅하기 위한 로그
- Time : 시간을 파싱하거나 포맷팅(아파치 common 구조)
- 직렬화 : 데이터를 바이트배열로 변환하거나 객체로 변환
- JSON : 데이터를 JSON문자열로 변환하거나 역변환
- Sort : 데이터를 정렬(Arrays 구조 참고)

ii) 로그 자료 데이터 구조

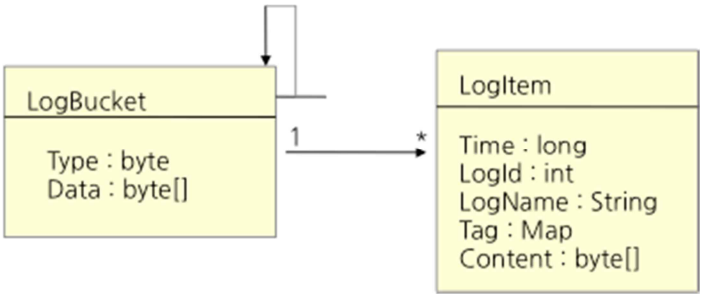


그림 31. 로그 자료 데이터 구조

- LogBucket은 내부에 LogBucket을 갖을 수 있다.
- LogBucket의 Type은 Data의 byte[]가 어떤 데이터인지를 명시한다.
- Type이 “0” 인경우 Data는 여러개의 LogItem임을 의미한다.

iii) 로그/모니터링 데이터 흐름

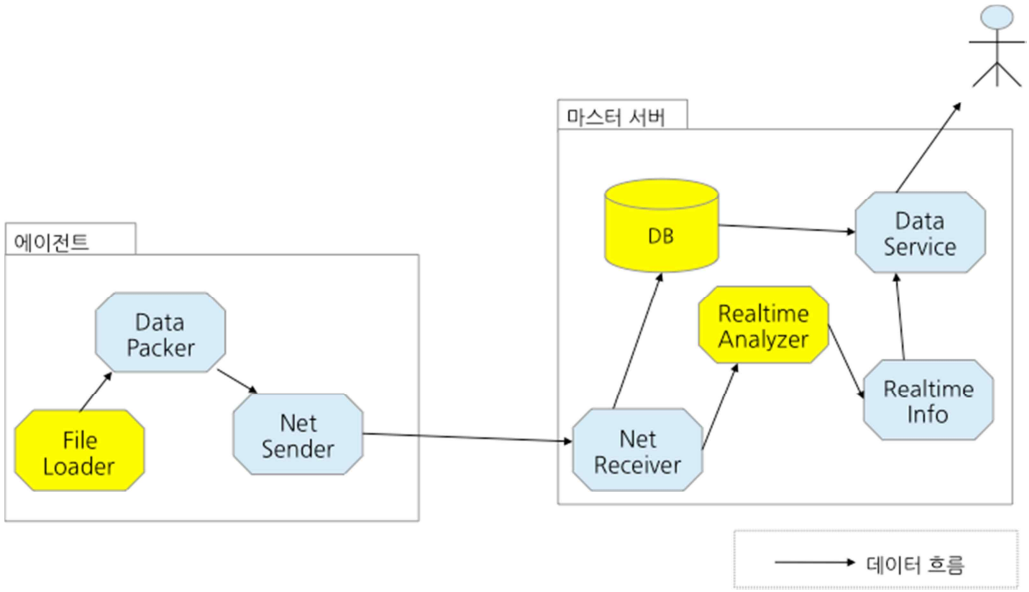


그림 32. 로그/모니터링 데이터 흐름

- 에이전트에서 서버까지 데이터 흐름은 모두 단방향으로 진행한다.
- 전송 프로토콜의 유연성을 고려해서 UDP 만으로 전송한다.
- 전송 프로토콜의 안정성을 고려해서 TCP 방식도 지원한다.
- 사용자는 마스터 서버의 데이터 서비스를 통해 실시간 현황 및 DB 데이터를 조회한다.

6) 에이전트 기능별 아키텍처

i) 모듈 구성

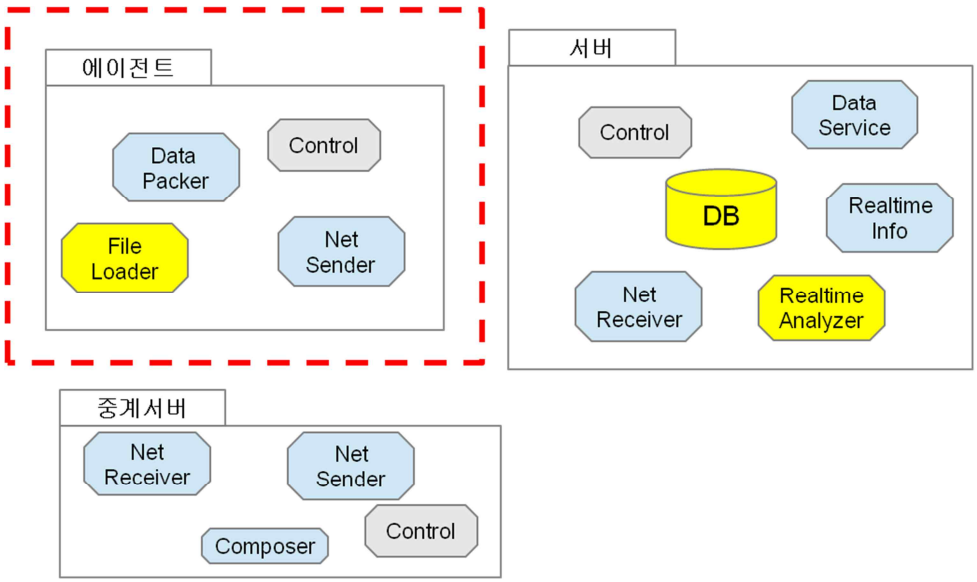


그림 33. 에이전트 모듈 구성

- File Reader : 로그 파일 읽어서 메모리로 불러오는 처리
- Line Parser : 로그 한 줄 단위 처리
- Composer : 데이터 패킷 생성 처리
- Net Sender : 전송 처리

ii) 파일로드 기능

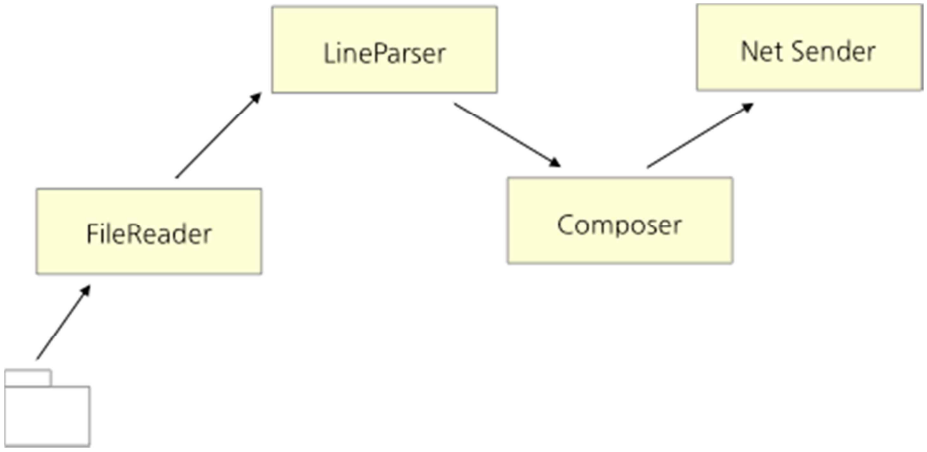


그림 34. 에이전트 파일로드 기능

- FileReader는 파일을 블럭단위로 로딩한다.
- LineParser는 파일을 라인단위로 파싱하고 시간 정보를 처리하여 LogItem으로 변환한다.
- LineParser는 멀티라인에 대한 조합기능을 갖는다.
- Composer는 파서가 전송하는 LogItem을 묶어 압축하고나 암호화한다.
- Net Sender는 UDP혹은 TCP를 통해 로그를 서버에 전송한다.

iii) 라인파서 기능

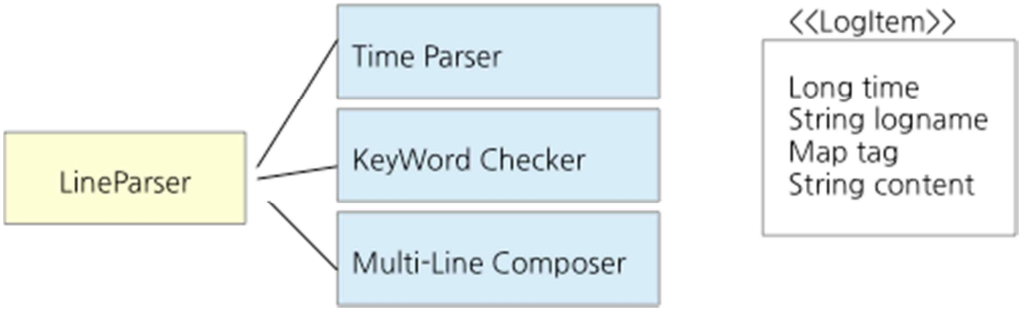


그림 35. 에이전트 라인파서 기능

- LineParser는 한 줄씩 읽어서 시간정보, 키워드를 추출하여 태깅하고 여러줄로 구성된 로그를 감지하여 조합한다.
- Time Parser는 한줄 로그를 읽어 시간을 파싱하고 LogItem.time에 셋한다.
- KeyWord Checker는 로그내용에서 지정한 키워드가 있는지 확인하여 LogItem.tag에 추가한다.
- tag에는 여러 개를 추가할 수 있다.
- Multi-Line Composer는 원래 여러줄로 구성된 로그를 감지하여 하나의 LogItem.content에 셋한다.
- LogItem은 시간과 내용을 갖는 한줄의 로그이다.

iv) 전송 기능

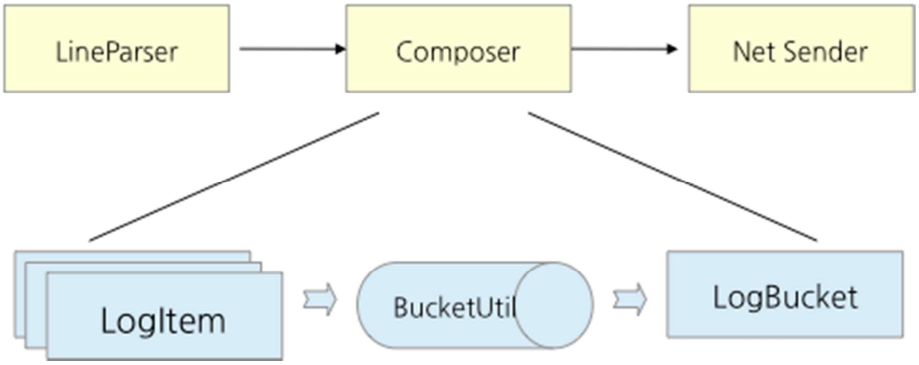


그림 36. 에이전트 전송 기능

- Composer는 입력된 LogItem들을 묶어서 하나의 Bucket으로 리턴한다.
- NetSender는 LogBucket는 byte[]로 변환하여 전송한다.
- NetSender는 UDP 혹은 TCP를 사용한다.

7) 서버 기능별 아키텍처

i) 서버 모듈 구성

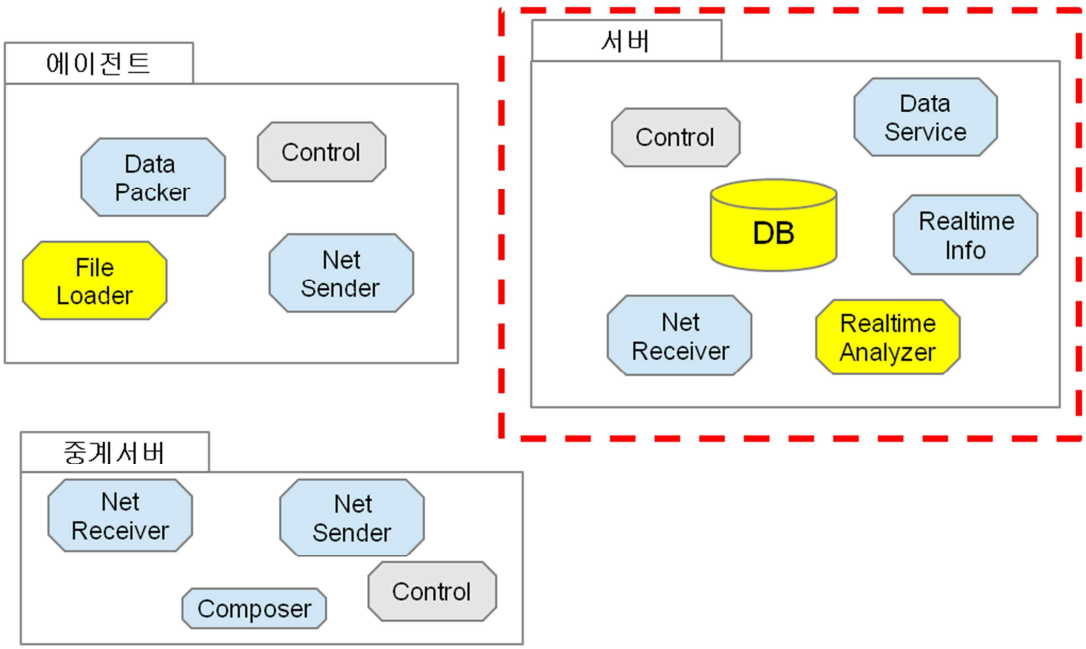


그림 37. **서버 모듈 구성**

- 서버는 데이터를 받아 저장함과 동시에 실시간으로 분석한다.
- 분석된 데이터는 처리 현황이나 경보 정보 형태로 Realtime Info에 보관한다.
- 서버의 데이터서비스 블록은 웹 서비스 형태로 제공되며, 실시간 현황 혹은 데이터 조회를 할 수 있다.

ii) 중계서버 모듈 구성

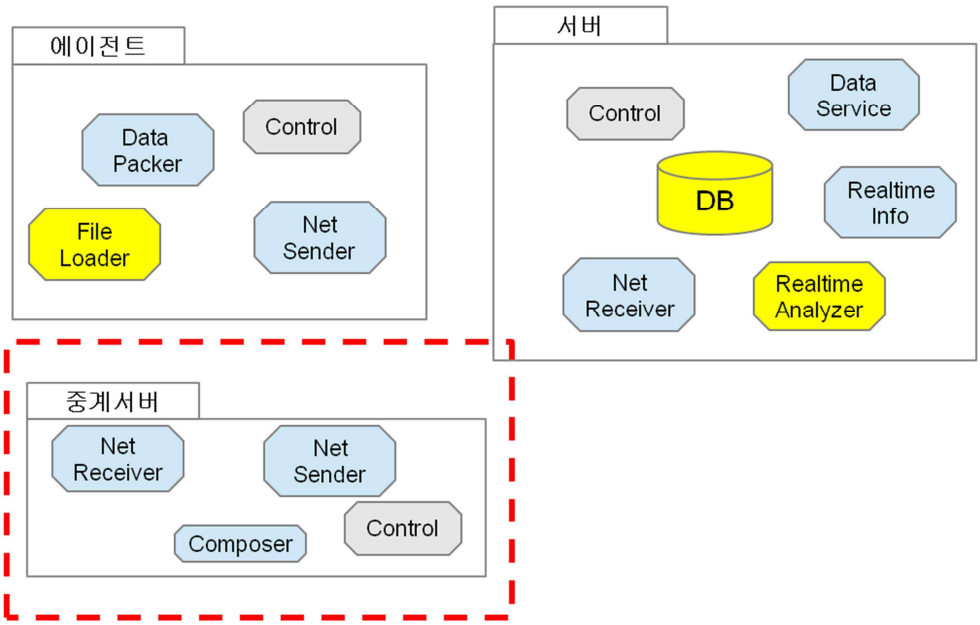


그림 38. 중계서버 모듈 구성

- 중계서버는 수신된 데이터를 다시 정해진 마스터 서버로 재전송한다.
- 중계서버/제어서버는 동일한 모듈이 설정에 따라 다르게 동작한다.
- 중계서버에는 LogItem을 묶어 압축하고나 암호화하는 Composer가 포함된다.

iii) 저장 기능

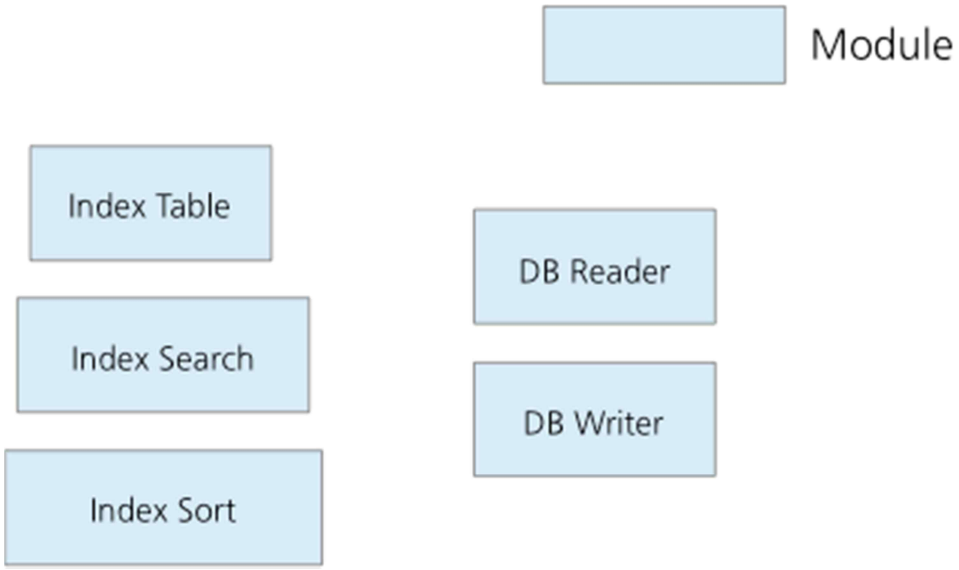


그림 39. 서버 저장 기능

- Index : 로그 데이터는 시계열로 적재한다. 시계열 인덱스를 관리하기 위해 3가지 모듈이 필요하다

  (1) Index Table : 로그를 쓰거나 읽기를 수행한다.

  (2) Index Search : 지정한 시간의 인덱스를 찾아낸다. 실제 읽는것은 IndexTable에서 처리한다.

  (3) Index Sort : 시간 순서가 바뀐데이터가 입력되면 인덱스 소팅을 수행한다.

- DB Writer : 데이터를 쓰는 모듈
- DB Reader : 데이터를 읽는 모듈

iv) 분석 기능

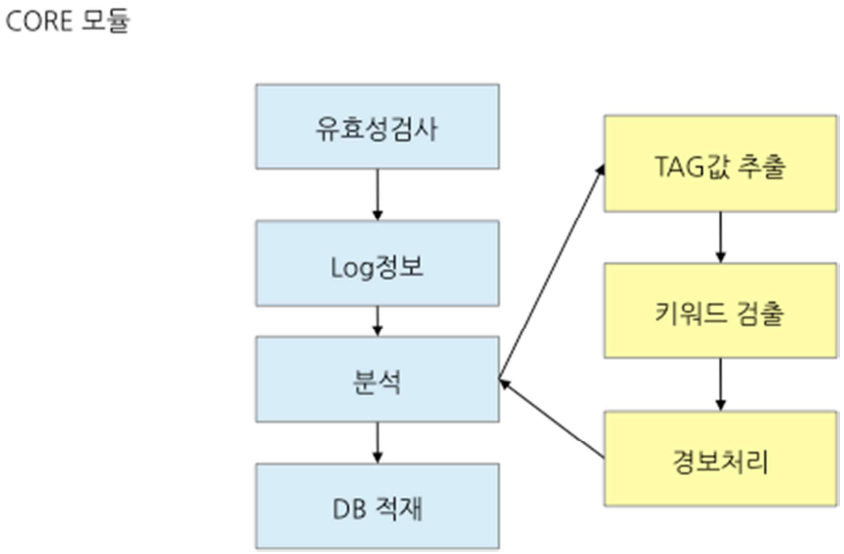


그림 40. 서버 분석 기능

- 유효성 검사 : 정상적인 로그인지 확인한다 특히 ID정보나 데이터가 없는 로그 정보인지 확인한다.
- LOG정보 : 로그시간과 이름을 검사한다. 시간정보가 없는 경우 서버 시간을 할당한다. 로그 이름은 이전에 전송된 이름정보를 할당한다.
- TAG값추출 : 주요 성능 및 관리 TAG값을 별도의 모듈에 케쉬한다.
- 키워드 검출 : 어댑터를 통해 서버쪽에서 분석해야할 주요키워드를 확인하고 태깅한다.
- 경보처리 : 에이전트로 부터 전송된 경고 메세지를 전송한다.
- DB적재 : 로그를 저장 모듈에 전달한다.

v) 데이터 수신 기능

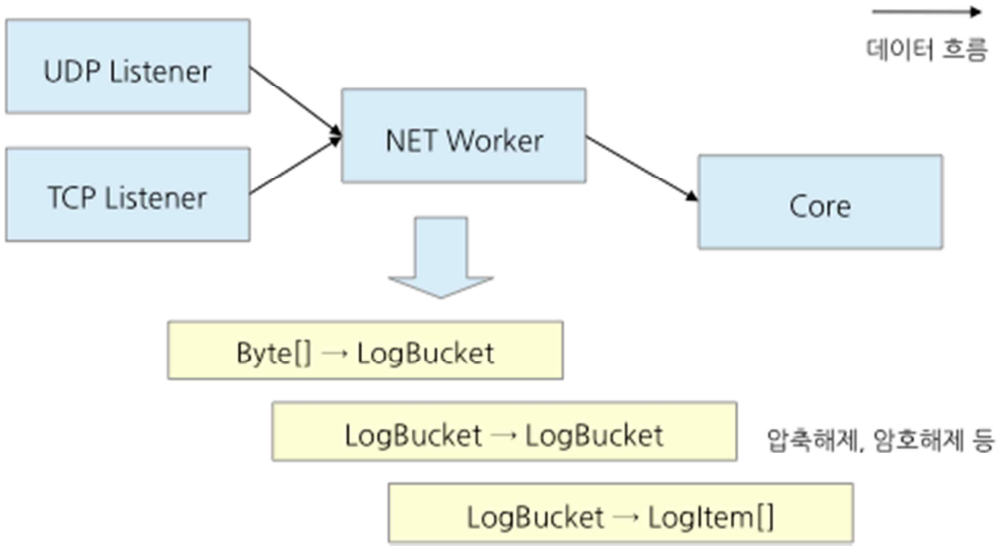


그림 41. 서버 데이터 수신 기능

- UDP Listener: UDP로 로그 데이터를 받는다
- TCP Listener : TCP로 로그를 받는다.
- Net Worker : UDP/TCP로 받은 byte[]를 LogBucket을 바꾸고 압축을 풀거나 암호를 해제한다. 최종으로 LogItem[]를 만들어서 Core모듈에 전송한다.

vi) 조회 기능

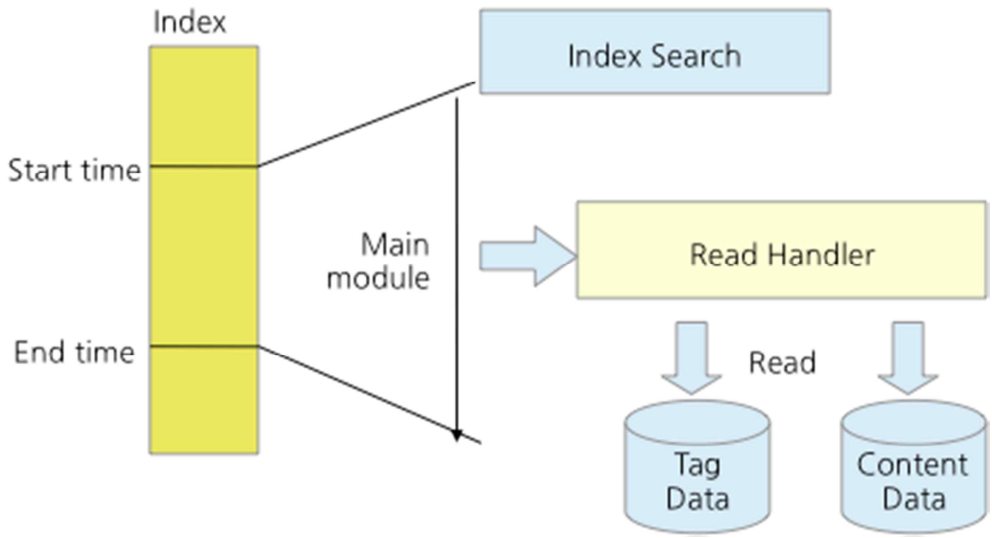


그림 42. 서버 조회 기능

- Index Search를 통해 시작 위치를 검색한다.
- Main모듈은 시작에서 부터 한건씩읽어 ReadHandler에 전달한다.
- ReadHandler는 Tag/Content 위치정보를 이용해 실제 Tag/Content Data를 읽어드린다.
- 데이터를 조회할때는 어댑터(Read Handler) 방식으로 로딩한다.
- ReadHandler는 클라이언트가 작성한 어댑터이다.

vii) 통계 생성 기능

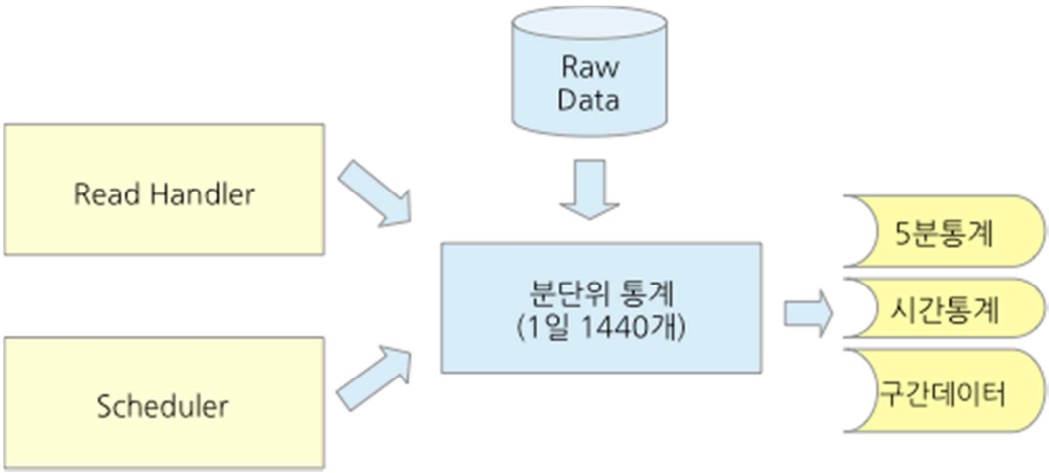


그림 43. 서버 통계 생성 기능

- 사용자가 데이터를 읽을때 혹은 지정한 시간에 스케줄러가 동작함으로 통계정보가 생성된다.
- 통계정보는 분단위 데이터이며 테그 혹은 로그에거 키워드 갯수등이 포함된다.
- SUM/COUNT/AVG/MAX/LAST등의 연산에 의해서 만들어진다.
- 분단위 통계만을 생성하며 하루 단위로 생성하여 저장한다.
- 사용할때는 뷰의 목적에 따라 1분 단위 데이터를 5분 혹은 1시간으로 변형하여 사용하며 일부 시간대의 정보만을 사용할 수도 있다.

viii) 커스터마이징/어댑터 구조

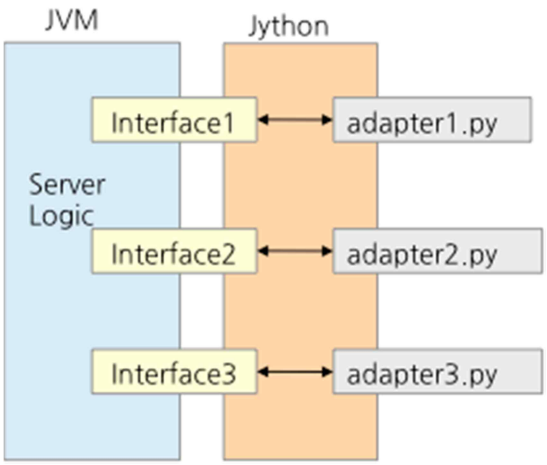


그림 44. 서버 아답터 구조

- 유연성이 극도로 필요한 서버로직에서 스크립트 엔진을 활용하여 어댑터를 구현한다.
- Jython 엔진은 python 프로그램에서 자바 클래스를 사용할 수 있으며, 인터프리터로 동작하기 때문에 성능보다는 유연성이 강조되는 부분에서 활용한다.

ix) 파일DB 구조

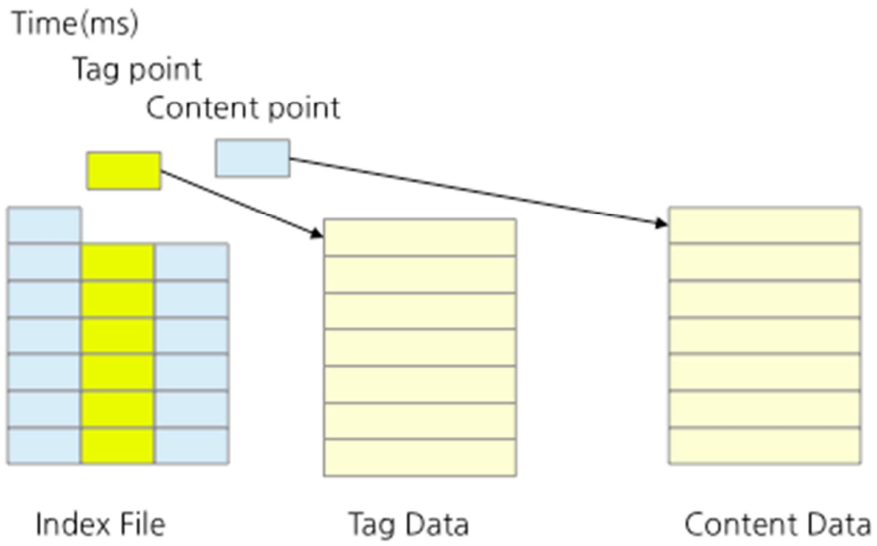


그림 45. 서버 파일 DB 구조

- 인덱스 파일을 Tag Data파일과 Content Data파일의 위치값을 가지고 있다
- 인덱스의 시간값은 순차임이 보장되어야 한다.
- 인덱스 파일 검색시 시간 값에 대한 Binary검색을 사용한다.

x) DB인덱스 정렬 방식

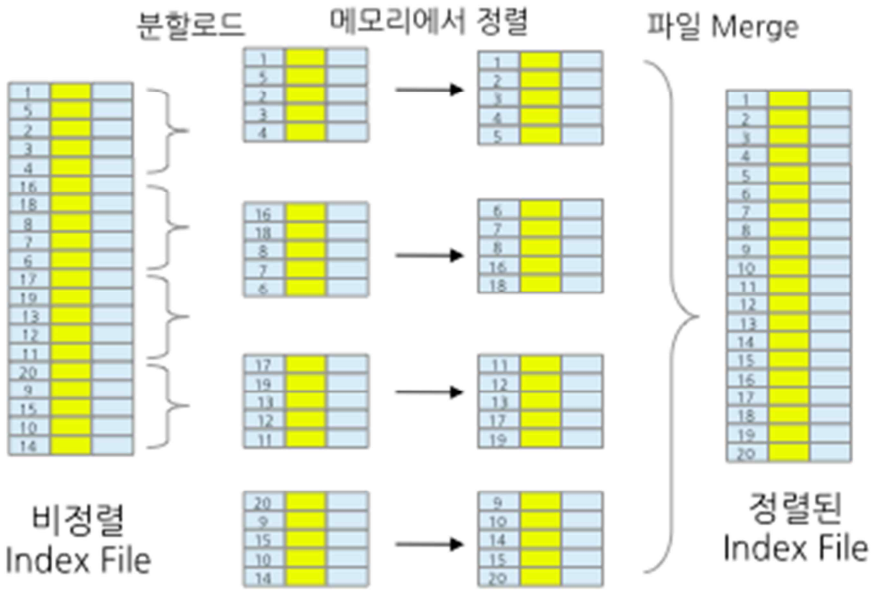


그림 46. 서버 DB 정렬 방식

- 인덱스 데이터는 시스템에 상황에 따라 정렬되지 않은 상태로 입력될 수 있다.
- 파일 정렬 방식은 메모리에서 수행하는 방식의 정렬(퀵, 버블, insertion)을 사용할 수 없다.
- 5000건씩 메모리에서 Tree소스와 파일에서 merge소스를 사용한다.
- 소스방법에 대해서는 변경 가능하도록 해서, 정렬 방식에 대한 수정이 전체에 미치는 영향을 최소화 해야 한다.

xi) 쓰레드/큐 모델

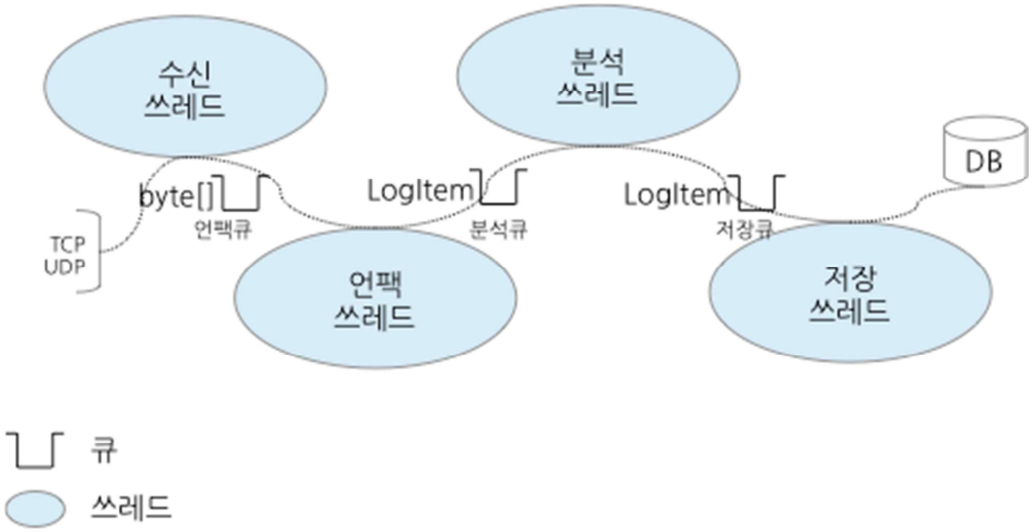


그림 47. 서버 쓰레드/큐 모델

o 데이터는 4개의 쓰레드와 3개의 큐를 거쳐 DB에 저장된다.

o 수신 쓰레드는 byte[]을 읽어 언팩큐에 추가한다.

o 언팩큐에 저장된 byte[]는 LogBucket의 직렬화된 데이터이다.

o 언팩쓰레드는 언팩큐의 byte[]을 LogBucket으로 변경하고 이것을 풀어서 LogItem[]으로 변경하고 그것을 분석큐에 입력한다.

o 분석 큐의 LogItem에는 시간정보나 logname이 채워지지 않을 수 있으며 분석쓰레드는 그것들의 값을 셋팅한다. 일부 정보를 채울 수 없는 경우에는 에러 메세지와 함께 버린다.

o 완전한 LogItem은 저장큐에 추가되고 저장 쓰레드는 DB에 저장한다.

o 4개의 메인 프로세스 스레드는 반드시 Singleton으로 수행되어야 한다. 프로세스 내에서 시간의 순서가 변경되어서는 안된다.

8) 네트워크 기능별 아키텍처

i) 네트워크 프로토콜

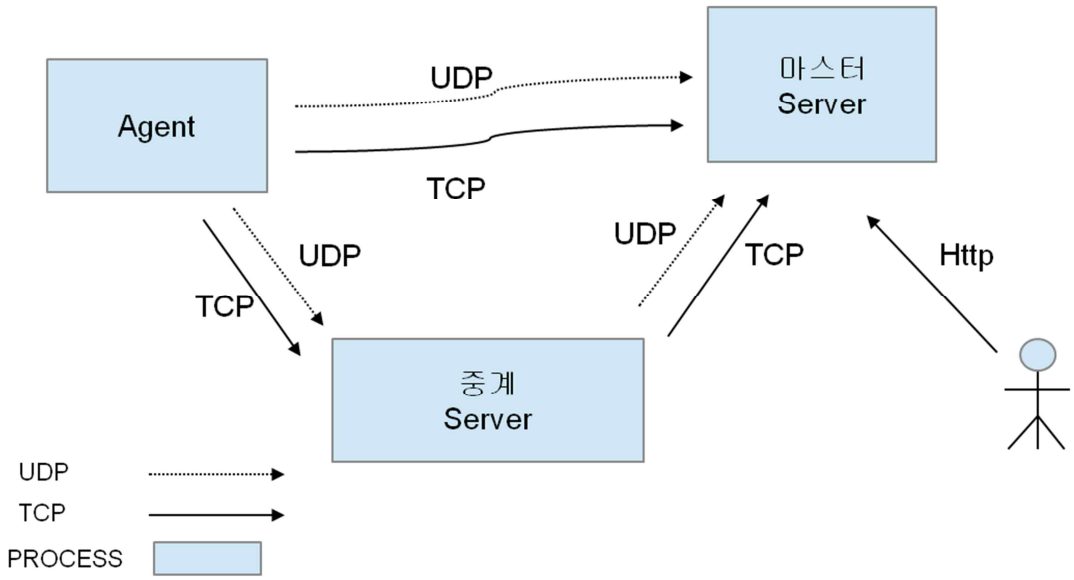


그림 48. **네트워크 프로토콜**

(a) UDP

- 데이터 전송의 기본 단위는 UDP 방식으로 처리한다
- 네트워크 관리와 데이터 송수신 효율성에서 TCP보다 좋음
- 다만 패킷 유실의 위험성이 있는 네트워크에서는 안정성을 위해서 TCP 방식도 고려

(b) TCP

- 안정적인 데이터 전송이 필요한 경우 TCP 방식으로 처리한다.

(c) HTTP

- 마스터 서버는 데이터 서비스를 HTTP 웹 서비스로 제공한다.
- 설정 서버도 최신 서버 설정을 위한 데이터 조회 HTTP API를 제공한다.

ii) 데이터 전송 계층

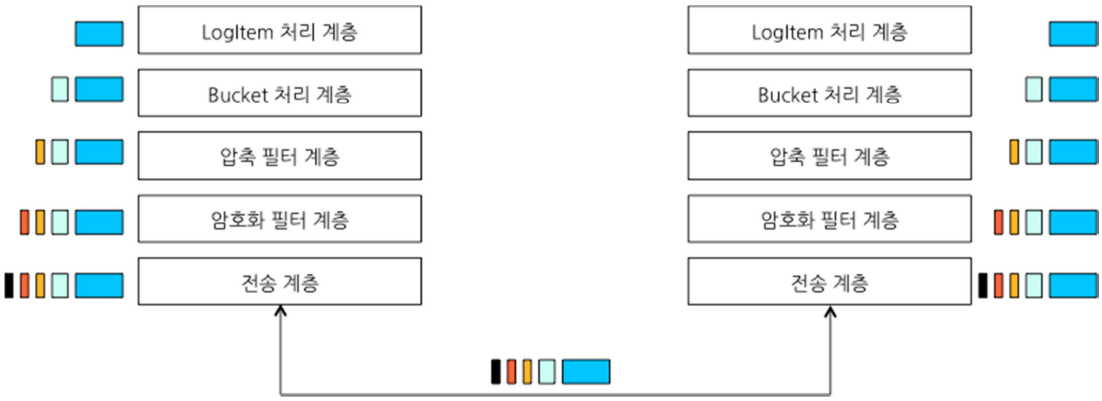


그림 49. 네트워크 데이터 전송 계층

(a) 로그 데이터(LogItem) 처리 계층

- 로그 데이터 수집하고 기본 처리 단위로 만든다.
- 만약 버킷에 여러 로그 데이터가 포함된 경우, 각 로그 데이터는 Id로 구분된다.

(b) 버킷(Bucket) 처리 계층

- 로그 데이터를 전송하기 위해 버킷 단위로 만든다.
- 여러 로그 데이터를 한 버킷에 포함할 수도 있다. (1:n 구조)

(c) 압축 필터 계층

- 전송을 위한 기본 버킷 단위 데이터를 압축해야 할 경우 헤더에 압축 flag를 설정하고 payload는 압축한다.
- 수신부의 압축 필터 계층에서는 헤더의 압축 flag를 참조해서 압축을 복원할 수 있다.

(d) 암호화 필터 계층

- 전송을 위한 기본 버킷 단위 데이터를 암호화해야 할 경우 헤더에 암호화 flag를 설정하고 payload는 암호화한다.
- 수신부의 암호화 필터 계층에서는 헤더의 암호화 flag를 참조해서 복호화할 수 있다.

(e) 전송 계층

- 버킷 데이터를 전송 프로토콜에 맞춰서 전송한다.

9) 클라이언트 기능별 아키텍처

i) 모듈 구성

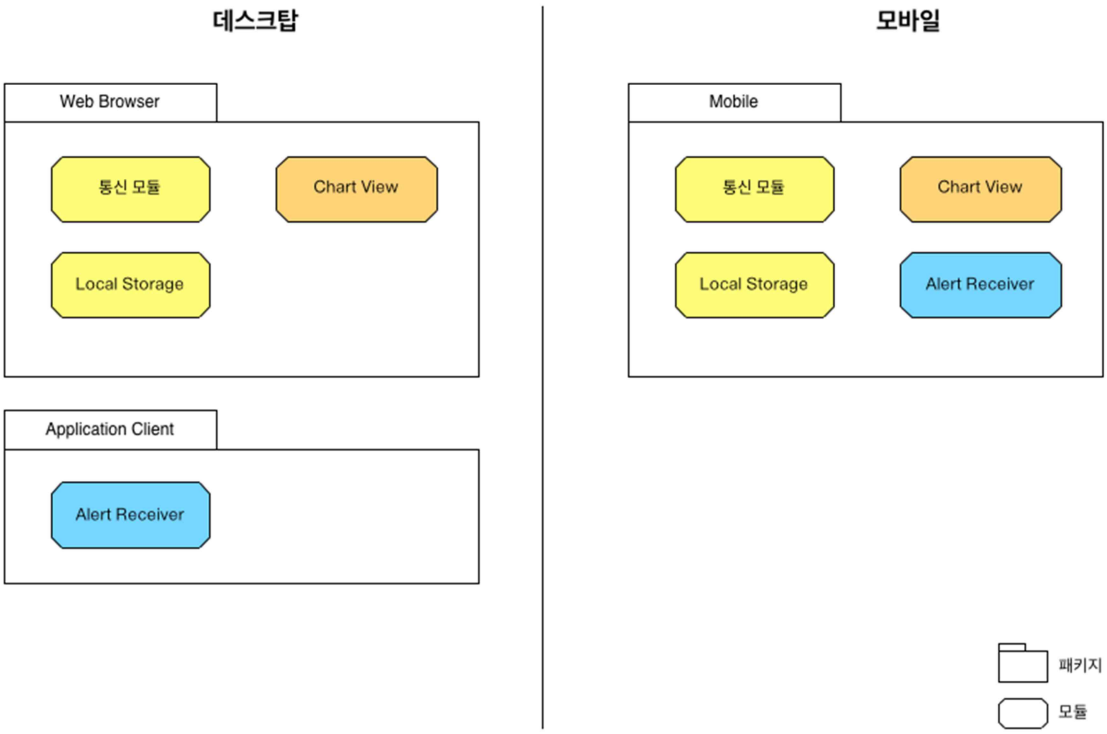


그림 50. 클라이언트 모듈 구성

- 클라이언트는 조회를 위한 Web Client와 알림 수신을 위한 Application Client로 이루어진다.
- Chart View는 수신된 데이터를 기반으로 다양한 차트를 보여준다.
- Application Client는 각 플랫폼별로 구현되는 Demon 형태의 애플리케이션이다.

ii) 통신 기능

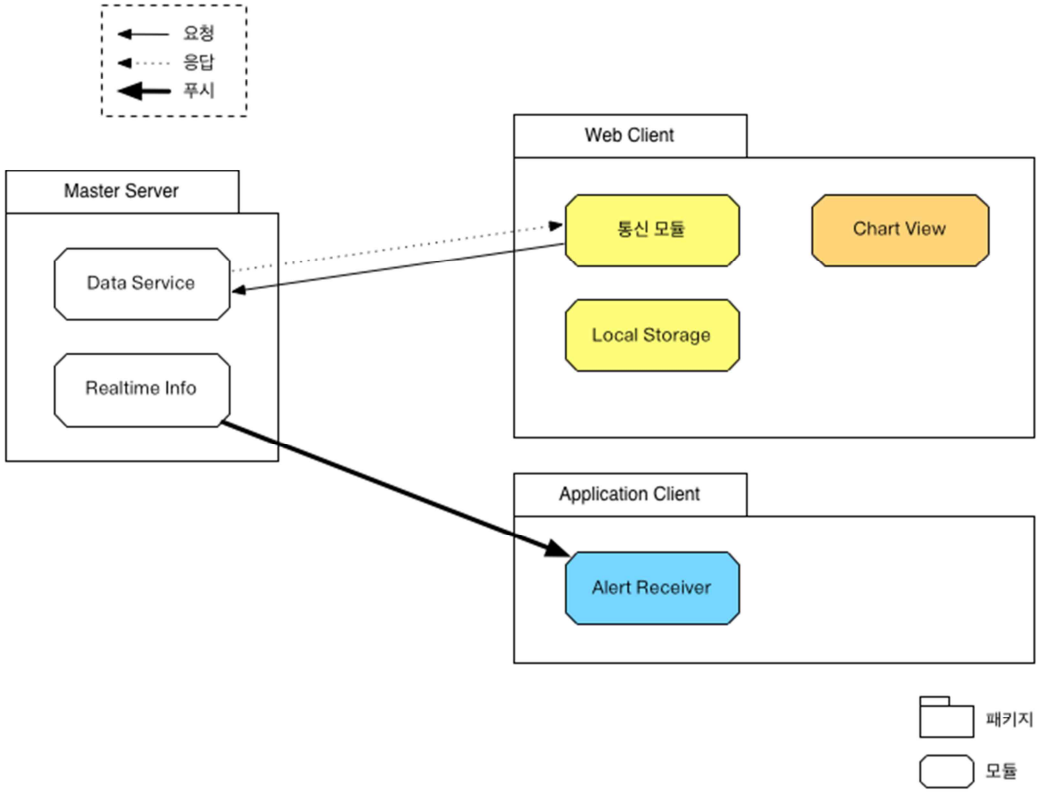


그림 51. 클라이언트 통신 기능

- Web Client는 사용자의 요청이 있을 경우 Master Server에 데이터를 요청한다.
- Web Client는 비동기적으로 Master Server에 데이터를 요청한다.
- Application Client는 Master Server의 Realtime Info 데이터를 푸시형태로 수신한다.

iii) 데이터 흐름

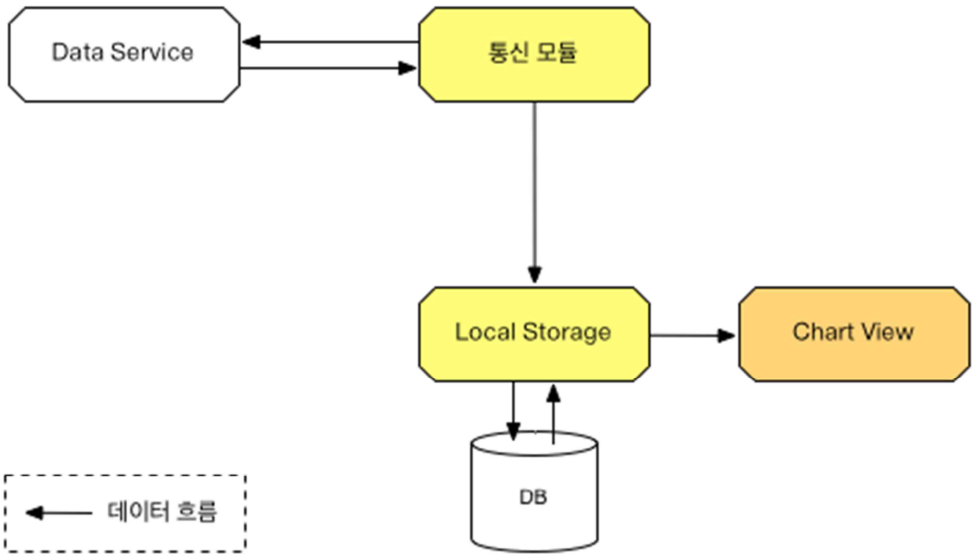


그림 52. 클라이언트 데이터 흐름

- Client의 통신 모듈은 Master Server와의 통신으로 통계 정보를 가져온다.
- 통신 모듈은 분 단위로 Master Server에 재요청을 한다.
- 통신 모듈은 비동기적으로 Master Server의 데이터를 요청과 수신을 반복한다.
- 수집된 데이터는 Local Storage를 통해 DB에 적재된다.

마. 아키텍처 뷰 (3차)

3차 설계에서는 2차 아키텍처를 보완하고 전송 데이터 구조, 데이터 저장소 구조, 예외 사항 등에 대한 기능별/모듈별 상세 아키텍처를 정하는 것을 목표로 했다.

1) 핵심 기능별 아키텍처

i) 데이터 직렬화 구조

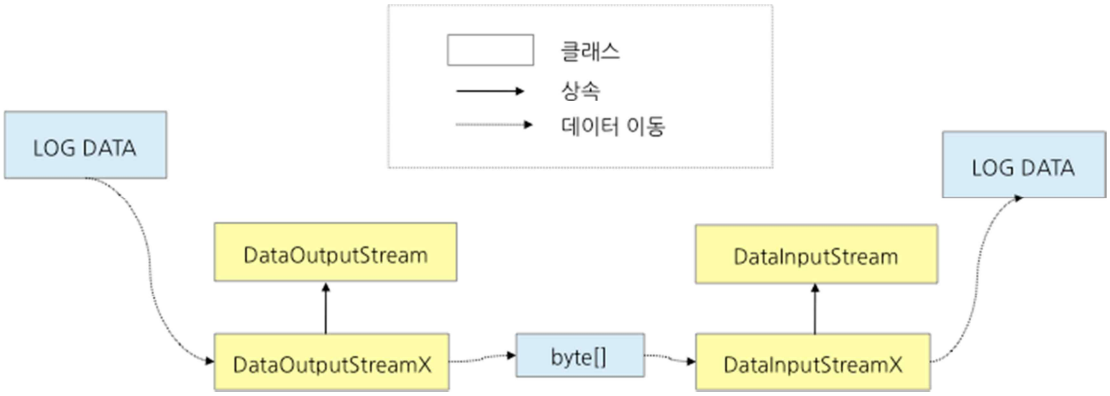


그림 53. 데이터 직렬화 구조

(a) 직렬화 처리

- DataOutputStream과 DataInputStream 으로 구성
- 로그 데이터를 자바 방식을 기준으로 직렬화하여 바이트 스트림으로 만들어서 사용함
- 언어와 시스템 환경에 독립적인 구조 지원

ii) 로그 데이터 구조

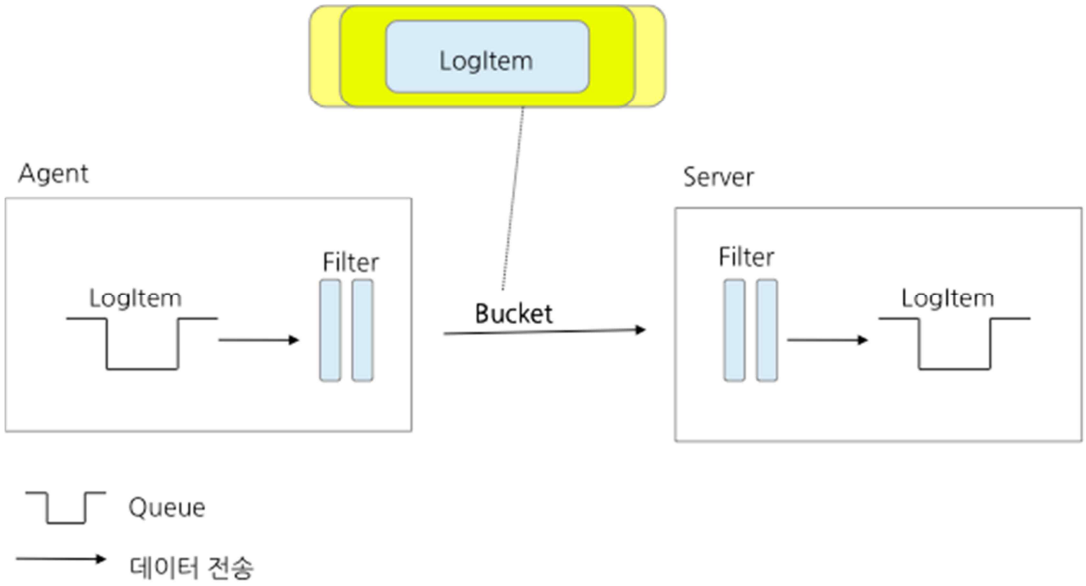


그림 54. 로그 데이터 구조

(a) 버킷(Bucket)

- 버킷은 전송을 위한 기본 단위
- 하나 이상의 로그 아이템이나 버킷을 포함할 수 있다.
- 데이터 전송하기 전에 압축, 암호화 필터 등을 처리

(b) 로그 아이템 (LogItem)

- 기본 데이터 수집 단위
- 한 줄의 로그 데이터 혹은 모니터링 데이터

2) 에이전트 기능별 아키텍처

i) 로그파일과 에이전트 수행단위

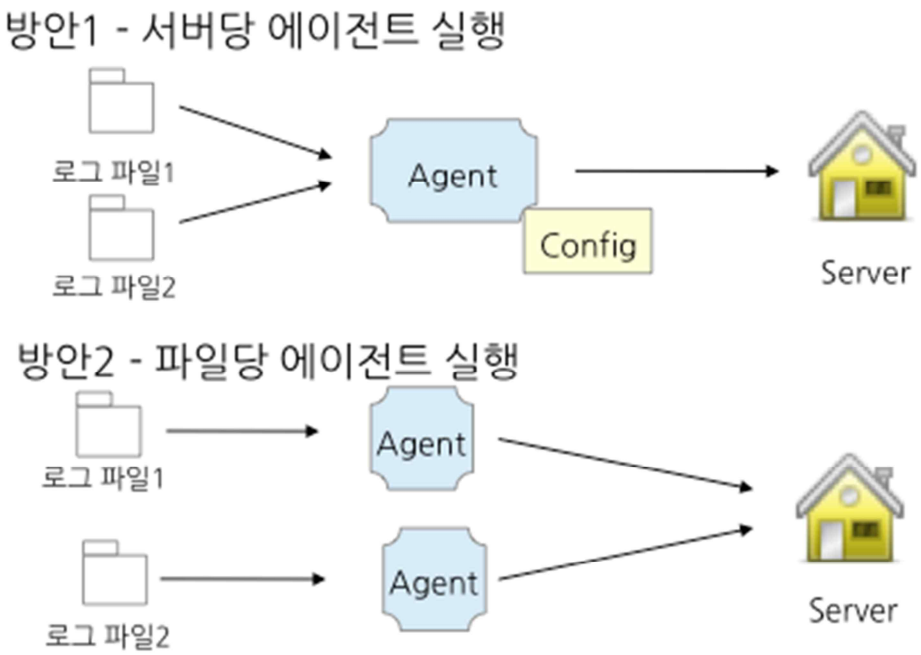


그림 55. 에이전트 수행 단위

(a) 방안1

- 대상 머신당 하나의 에이전트를 실행한다.
- 어떤 로그를 수집할지에 대한 처리는 설정파일에 가지고 있어야 한다.
- 서버에서 설정을 통제하는 방안이 필수가 된다.

(b) 방안 2

- 에이전트는 로그파일 하나만 바라보기 때문에 로직이 단순하다.
- 에이전트를 재기동해도 수집에 미치는 영향이 파일 하나로 한정된다.
- 로그마다 다른 포멧 다른 적재 룰을 가지고 있을 수 있는데 대응이 단순하고 쉽다.

ii) 멀티 플랫폼 직렬화

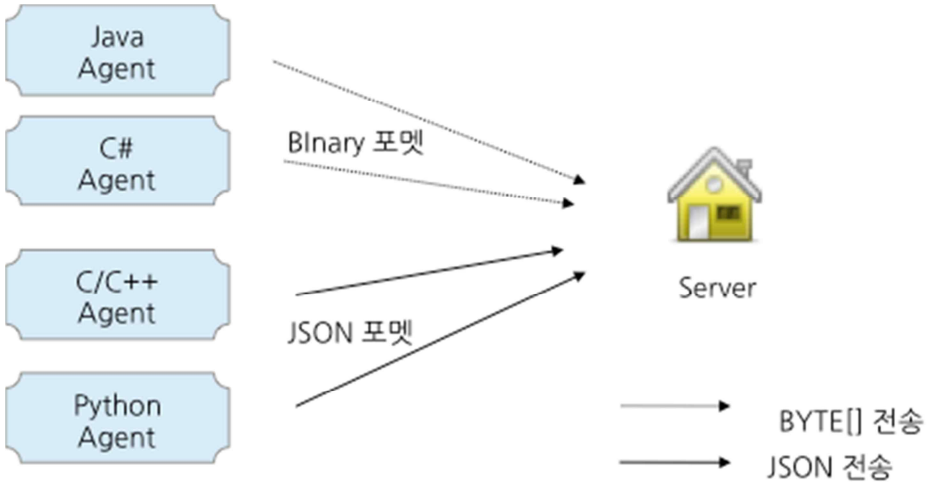


그림 56. 멀티 플랫폼 직렬화

- Java/C#에이전트는 LogBucket의 직렬화를 그대로 사용한다.
- C/C++, Python은 Java형식의 직렬화 기능을 적용하기 어려울 경우, JSON포멧을 활용하여 데이터를 직렬화한다.

iii) 전송 준단에 대한 예외 처리

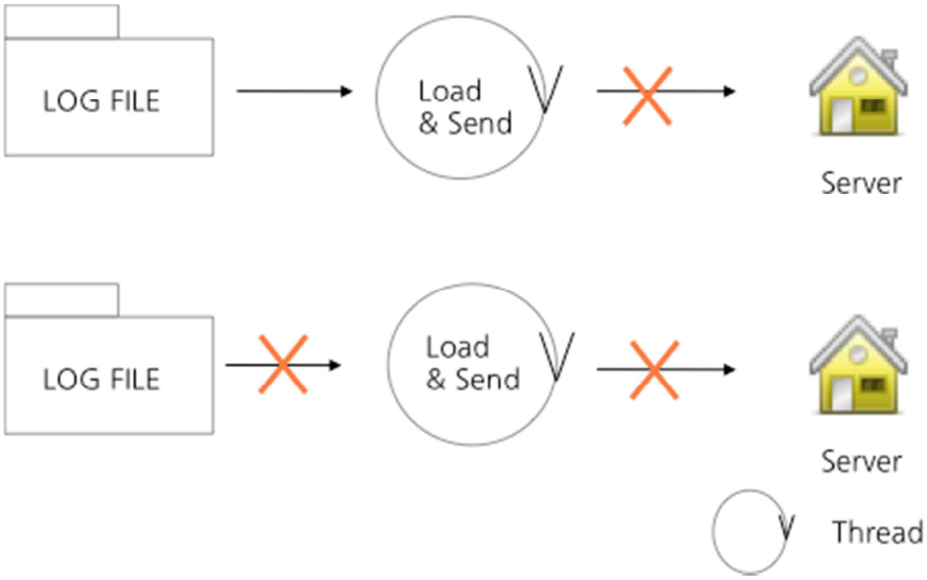


그림 57. 전송 중단 예외처리

- 로그를 읽는 쓰레드와 보내는 쓰레드를 별도로 두면 서버가 비정상적인 상태가 되었을 때 재 전송 처리를 해야 한다.
- 이 문제를 해결하고 로직을 단순화하기 위해 로그를 읽고 보내는 처리를 단일 쓰레드로 한다.
- 보내는 통신에 문제가 발생하면 쓰레드는 파일 읽기 또한 중단하여 서버가 정상화 되었을때 로그를 전송한다.

iv) Agent Time와 Server Time

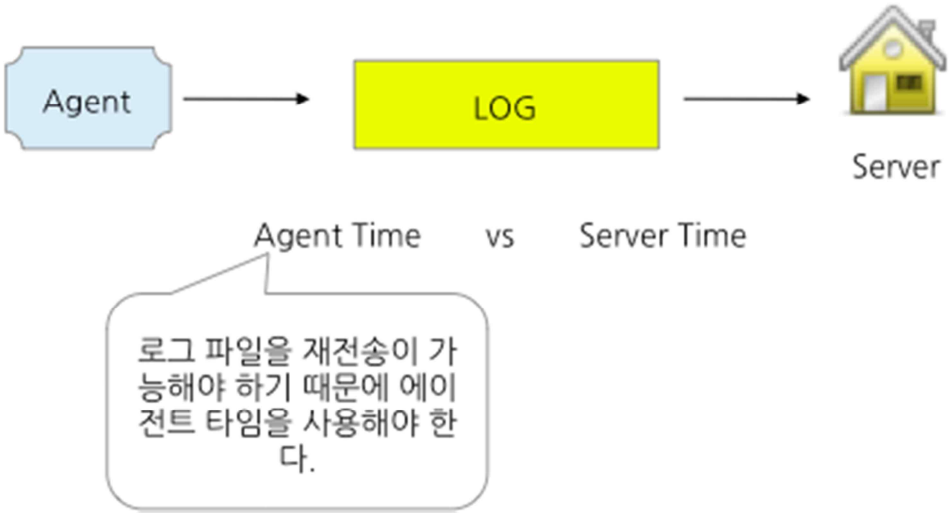


그림 58. 에이전트 시각과 서버 시각

(a) Agent Time 장단점

- 재전송 가능
- 실시간 처리 어려움
- 여러 에이전트의 로그를 하나의 파일로 통합할 수 없음. (별개 보관)
- 에이전트에서 시간을 지정할 수 없는 로그에 대해서는 서버 시간 사용가능

(b) Server Time 장단점

- 실시간 처리 가능
- 여러 에이전트 로그를 한 파일로 묶을 수 있음
- 재전송이 불가능함

v) 로그 이름과 로그 아이디

그림 59. 로그 이름과 로그 아이디

o 한 줄의 로그를 서버로 전송할때 에이전트는 로그의 이름을 부여해야한다. 하지만 로그 이름을 이해가능하게 사용하려면 길이가 길어지고 모든 로그에 이름을 부여해야하기 때문에 데이터 량이 증가하는 문제가 발생한다.

o 이것을 해결하기 위해서는 LOG ID와 LOG NAME을 혼용해야 한다. 그런데 이때 ID는 서버가 부여할 수도 있고 에이전트에서 임의로 지정할 수 있다.

o 에이전트가 지정하는 방식이 좋으나 유일해야 하는 문제를 해결하기 위해서는 LOGNAME과 ID를 일정한 룰로 맵핑해서 만들어야 한다.

o crc32를 사용하여 이 문제를 해결한다. LOG NAME을 유일하게 부여하고 이 이름에 대한 CRC32(4byte) 값을 ID로 사용하여 데이터량을 줄일수 있다.

3) 서버 기능별 아키텍처

i) 서버 내부 구조

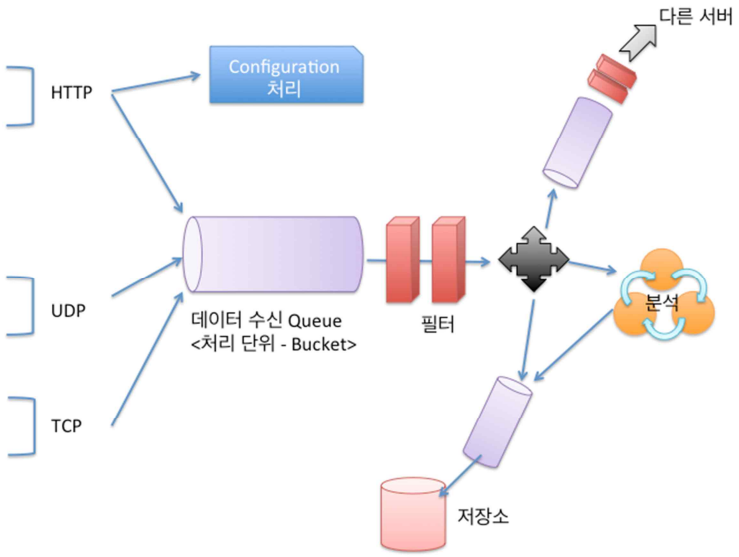


그림 60. 서버 내부 구조

(a) 데이터 수신 큐

- 패킷 데이터를 잃어버리지 않고 수신하기 위한 처리 큐
- 프로토콜 독립적으로 버킷 단위로 처리

(b) 필터

- 버킷 구조의 필터 플래그별 처리 필터
- 암호화, 압축 필터 등

ii) 데이터 저장소 구조

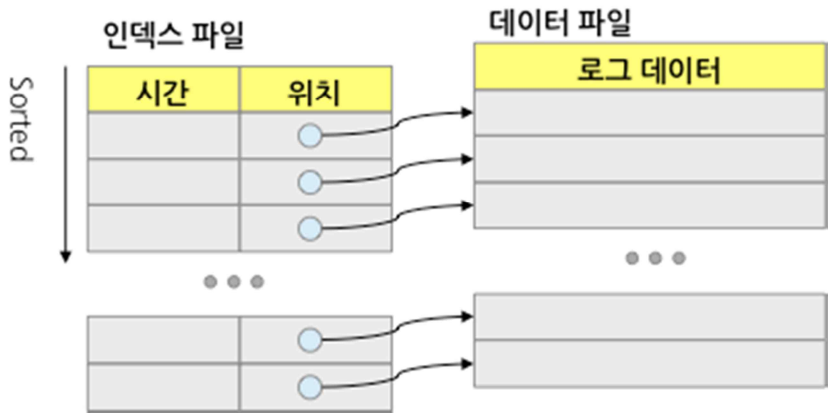


그림 61. 데이터 저장소 구조

(a) 저장소 구조

- 로그 데이터 구분을 위해서 Log Id와 같은 구조의 디렉터리를 사용해서 저장한다.
  (예) /NIPA/websys/framework/error
- 아이디는 관리 편의상 다음과 같은 제약사항을 갖는다.
  a) 영어/숫자만 사용
  b) 특수문자 사용불가
  c) 대소문자를 구분 (소문자 권고)
- 한 디렉터리에는 인덱스와 데이터 파일 두 개만 존재한다.

(b) 인덱스 파일

- 시계열 인덱스 파일 구조를 사용한다.
- 시간순으로 데이터가 전달되는 구조를 가진다.
- 시간 8바이트, 위치 8바이트 크기
- 인덱스 파일이 깨지더라도 데이터 파일로 복구가 가능함

(c) 데이터 파일

- 로그 데이터는 가변 길이
- 시간 정보와 로그 데이터가 기본 구조

iii) 용량 계획

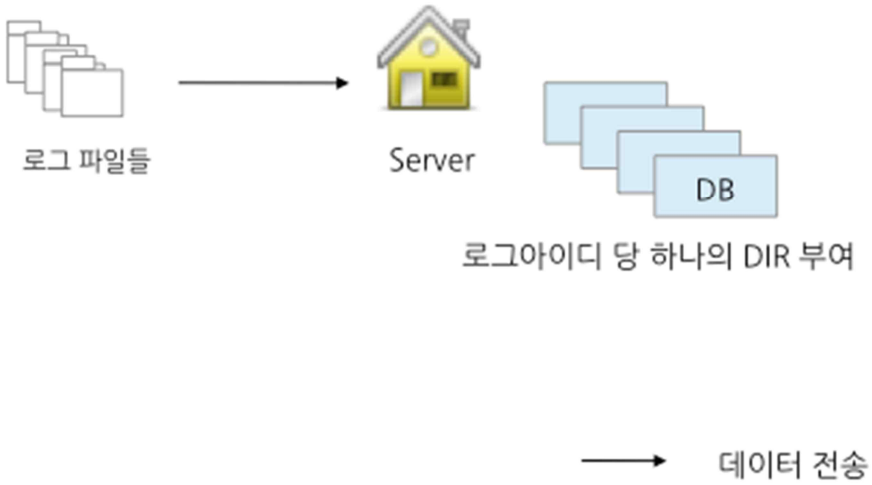


그림 62. 용량 계획

- 로그파일 각각에는 고유의 이름과 아이디를 부여한다.
- 서버는 로그 이름별로 별도의 디렉토리를 생성하고 로그파일을 적재한다.

iv) Cascade 설정 관리

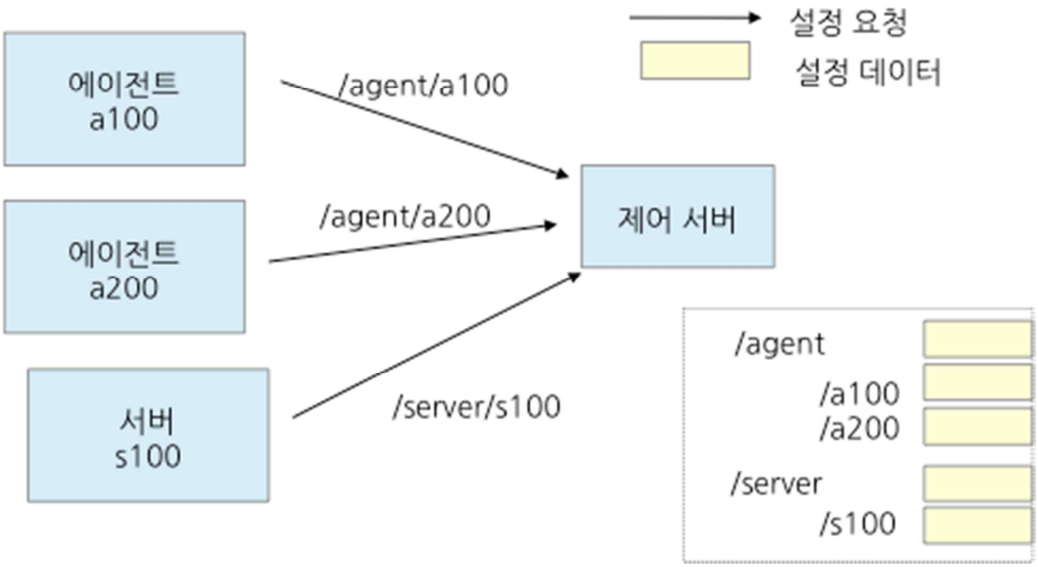


그림 63. Cascade 설정 관리

- 제어서버는 각 단계별 설정정보를 보관한다.
- 만약 에이전트가 /agent/a100의 설정을 요청하면 제어서버는 /agent의 설정과 /agent/a100의 설정을 합하여 리턴한다.
- Cascade형태의 설정정보 서버는 설정을 공통화하여 관리할수 있는 개념이다.

4) 클라이언트 기능별 아키텍처

i) 로컬 저장소 활용

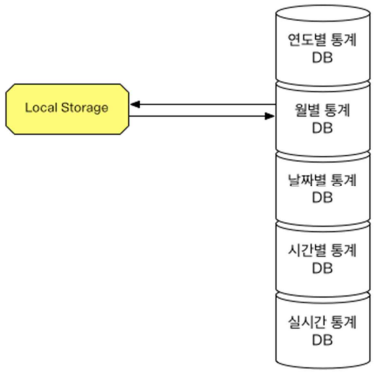


그림 64. 클라이언트 로컬 저장소

- Local Storage는 사용자가 요청한 데이터를 서버로부터 수신하여 DB에 보관한다.
- Local Storage는 반복되는 사용자의 요청을 처리하기 위한 목적을 가진다.
- 실시간 통계를 제외한 현재 해당되는 구간의 통계정보는 수집하지 않는다.

(a) 실시간 통계 DB

- 실시간 통계 DB는 실시간 로그 정보를 그래프로 표시하기 위한 목적으로 수집된다.
- 사용자는 1시간 이내에 수집된 정보를 전체적으로 확인하거나 특정 구간의 정보를 확인하는 용도로 사용된다.
- 실시간 통계 DB는 분 단위로 수집되며 1시간이 지난 데이터는 삭제한다.

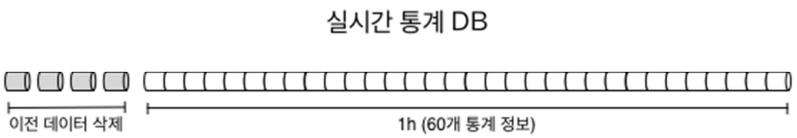


그림 65. 클라이언트 실시간 통계

(b) 시간별 통계, 날짜별 통계, 월별 통계, 연도별 통계

o 시간별, 날짜별, 월별, 연도별 통계를 DB에 수집하는 목적은 사용자가 구간 정보를 요청할 경우 서버와의 통신을 최소화하기 위한 목적을 가진다.

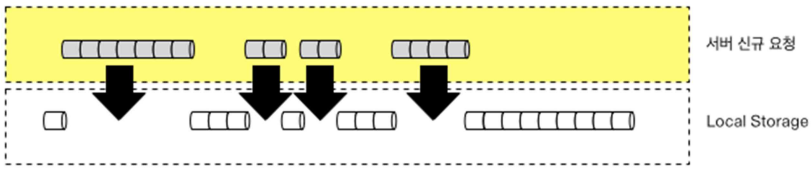


그림 66. 시간대별 통계 저장 구조

ii) 차트 기능

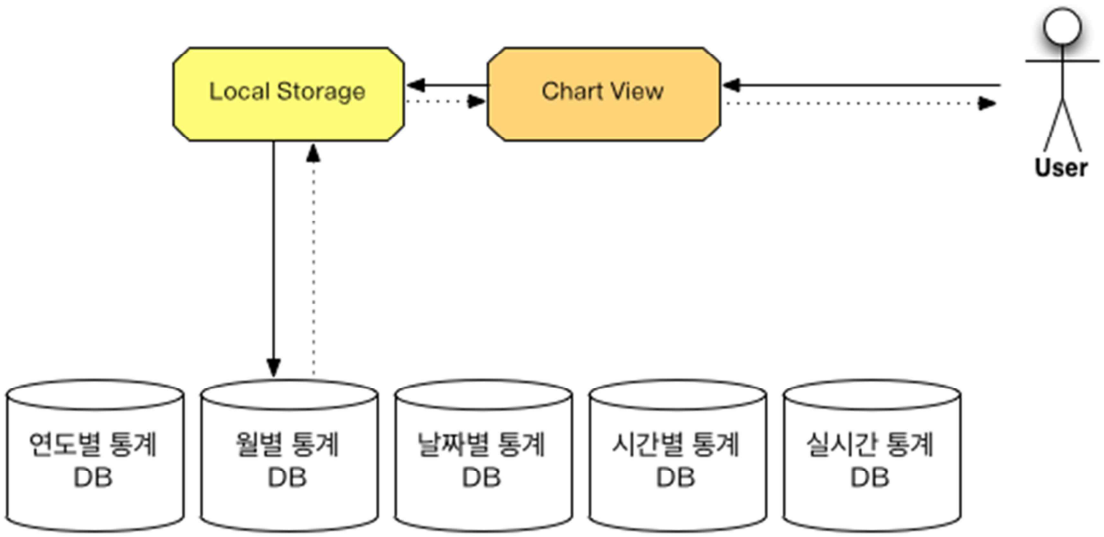


그림 67. 클라이언트 차트 기능

o Chart View는 사용자의 요청을 받았을 때, Local Storage에 데이터를 불러와 화면을 구성한다.

o 만일 사용자가 요청한 구간에 데이터가 없을경우 서버에 요청한다.

iii) 알람 수신 기능

서버에 이상이 생길 경우 장애, 알람, 상태 등을 알람을 보내기 위한 구조적 접근

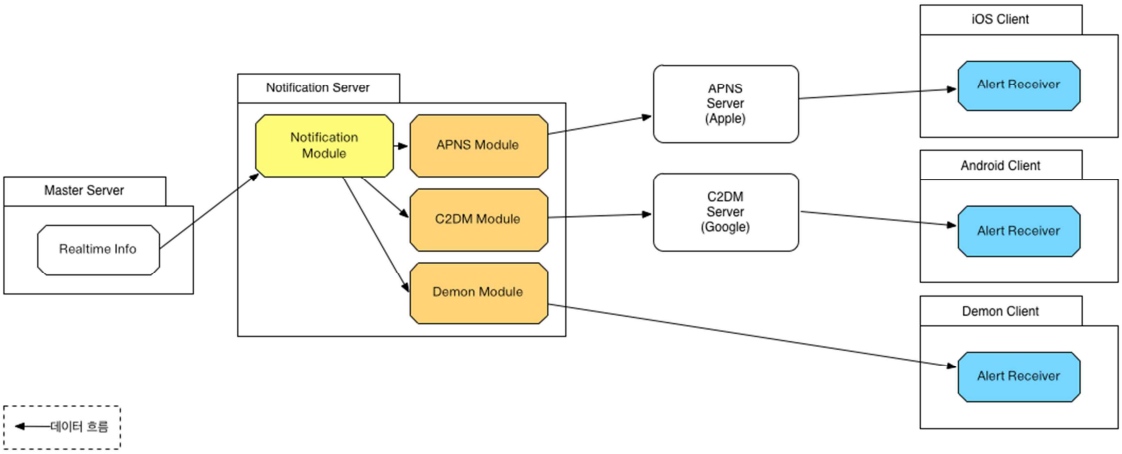


그림 68. 클라이언트 알람 수신 기능

(a) Notification Server

- Notification Server는 클라이언트에 보낼 알림 메시지를 처리하는 패키지이다.
- Notification Server는 Notification Module을 통해 Master Server의 Realtime Info에 데이터를 가져오고 관리한다.
- Notification 타입에는 APNS, C2DM, Demon 3가지로 이루어진다.
- Notification 타입은 Plug-in 형태로 추가/삭제할수 있고, 타입별 모듈의 변경이
- Notification Module에 영향을 주어서는 안된다.

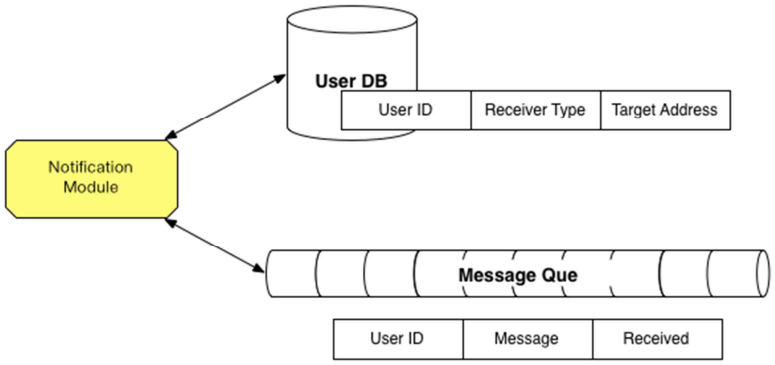


그림 69. **클라이언트 알람 모듈**

(b) Notification Module

- Notification Module은 각 사용자별로 받고자 하는 Notification Type을 관리한다.
- Notification Module은 보낼 메시지를 메시지 큐에 넣어 관리한다.

4. 아키텍처 검증

가. 아키텍처 검증

1) 품질 속성 검증 절차

i) 비즈니스/아키텍처 목표 이해

- “중소기업에서 필요한 클라우드 기반 요소기술과 참조 아키텍처 - 로그/모니터링 데이터 수집 모델”
- 아키텍처 접근법에 대한 설명과 브레인스토밍 후 유틸리티 트리 작성

ii) 참조 모델의 특징

- 모든 클라우드 요소기술을 다루기 보다는 꼭 필요한 적정기술을 찾는 데 촛점
- 프로비져닝과 분석, 비주얼라이제이션 도구는 빅 데이터 솔루션 영역
- 아키텍처 패턴은 단순한 구조에서 확장 가능하도록 하는 패턴에 중심

iii) 위험요소 도출

- 공중망으로 전송할 경우 보안문제와 멀티플랫폼 이기종 지원을 위한 구현의 복잡성
- 구성 요소 확장에 따른 스토리지 저장 공간
-

2) 유틸리티 트리

표 38. 우선순위화된 유틸리티 트리 목록

품질 속성	속성 정제	기호	시나리오	중요도	구현성	합계
유연성	운영중인 레거시 시스템 연동	A1	기존에 운영중인 레거시 시스템 로그를 수집할 수 있어야 한다	4	4	8
	신규 서버 확장	A2	신규 서버/시스템이 확장된 경우에서도 쉽게 확장되어 로그/데이터 수집이 가능해야 한다.	5	4	9

	가상화 VM 연동	A3	VM/인스턴스별 로그/데이터 수집이 가능해야 한다.	4	3	7
통합성	프로토콜 통합 및 메시지 표준	I1	물리적으로 분리된 시스템의 로그/데이터를 프로토콜 및 메시지 표준화시켜 통합이 가능해야 한다.	4	4	8
	이기종 로그 통합	I2	이기종 다른 규격의 로그/데이터도 통합이 가능해야 한다.	5	3	8
성능 (Performance)	동시 수집 용량	P1	동시에 안정적으로 최대 수집 가능한 시스템 성능을 예측하고 운용되어야 한다.	5	3	8
	데이터 패킷 용량 최소화 및 효율성	P2	데이터 패킷을 압축하여 용량을 최소화하거나 흐름 제어로 효율적인 전달이 가능해야 한다.	4	3	7

나. 아키텍처 접근 방법 분석

1) 유연성 시나리오 분석

표 39. 유연성 시나리오 분석

시나리오 번호	#1	**시나리오**	클라우드 기반의 정보시스템에서 시스템 확장으로 인하여 운영중인 시스템과 로깅 서버에 시스템 신규 추가/확장으로 인한 확장이 보장되어야 한다.
품질속성	유연성		
환경 (Environment)	클라우드 환경 및 운영환경		
자극 (Stimulus)	시스템의 추가 및 클라우드 환경에서의 시스템 Scale-out 등의 확장		
응답 (Response)	시스템 신규 추가 및 확장시 로깅 컴포넌트를 추가하여 기존 로그 수집에 통합되도록 한다.		
아키텍처 판단	민감점	절충점	위험 / 무위험

중계 서버 확장 구조	네트워크 구성에 따라서 중계 서버가 오히려 장애포인트가 될 수 있음			
마스터 서버 저장소 구조		저장소에 HDFS를 도입하려면 중복 투자가 될 수 있음	마스터 서버에 집중 저장될 경우 위험 요소 있음	
마스터 서버 아답터 패턴	파이썬 스크립트 아답터 형태가 로직 구조의 복잡성을 해결하는지 확인 필요			
종합 의견	확장성과 유연성을 비교적 고려한 구조를 갖고 있음. 클라이언트가 접근하는 특정 마스터 서버에 저장소가 집중될 위험성이 있음.			

2) 통합성 시나리오 분석

표 40. 통합성 시나리오 분석

시나리오 번호	#2	**시나리오**	이기종 시스템에서 로그 메시지의 물리적 통합에 문제 없이 통합되기 위해서 프로토콜 통합 및 메시지 포맷 표준 준수를 통해서 로그 에이전트와 컬렉터를 통한 로그 메시지 수집하고 통합을 할 수 있어야 한다.
품질속성	통합성		
환경 (Environment)	운영되는 이기종 시스템		
자극 (Stimulus)	시스템에 로그 적용		
응답 (Response)	기존 운영되던 시스템의 로그와 새롭게 추가가 되는 시스템 로그들간의 문제없는 통합, 프로토콜 통합, 메시지 포맷 표준 준수		
아키텍처 판단	민감점	절충점	위험	무위험

멀티 프로토콜 지원		네트워크 구성과 프로토콜 특성에 맞는 것을 선택과 통합 처리	UDP, TCP, Thrift 등을 단독을 쓸 경우의 문제점 있음	
마스터 서버 저장소 구조		마스터 서버에 저장소에 통합 저장과 인덱싱 처리		
마스터 서버 데이터 분석 방식		클라이언트에서 확인하기 위한 기초 데이터만 분석. 심도 분석은 빅데이터 분석 도구를 활용		
종합 의견	이기종 로그 통합을 위한 방안이 부족함 외부망을 사용할 경우, 암호화 세션을 지원할 경우 대안이 필요함 모바일 환경에서 로그 수집에 대한 고려가 부족함			

3) 성능 시나리오 분석

표 41. 성능 시나리오 분석

시나리오 번호	#3	**시나리오**	클라우드 환경 및 운영환경에서 시스템 로그 이벤트 발생시 데이타 교환에 따른 패킷 최소화와 데이타 교환의 효율성을 통해서 로그 데이타 저장의 성능을 보장한다.
품질속성	성능		
환경 (Environment)	클라우드 환경 및 운영환경		
자극 (Stimulus)	수집할 필요성이 있는 app log 및 시스템 로그		
응답 (Response)	시스템 로깅을 수행함에 있어서 데이타 교환에 따른 패킷을 최소화하고 성능이 보장되도록 한다.		

아키텍처 판단	민감점	절충점	위험	무위험
로그 아이템 데이터 구조	로그 아이템의 암호화, 압축 처리 필터는 필	사내망에서는 제외할 수 있음	개인 정보가 포함된 경우 로그 아	

	수		이템 데이터 노출 위험	
멀티 프로토콜 지원		네트워크 특성에 맞춰 UDP 혹은 TCP를 선택 활용		
마스터 서버 파일 DB 구조	수집 파일 관리 정책이 필요	DB 보다 성능이 좋은 파일 구조 선택		
종합 의견	파일 DB 구조와 성능에 대한 아키텍처는 실험적으로 증명했지만, 성능 시나리오를 확인하기 위한 지표가 부족함 네트워크 프로토콜과 직렬화, 암호화 방식 등에 대한 성능 감소를 감안해야 함			

5. 부록

가. 저장소 성능 비교

- 아키텍처 참조 모델의 저장소 파일 디비 성능 검증
- 로그 수집에 대한 안정성과 핵심 품질 요소 검증
- 동시 수집 용량과 데이터 패킷 송수신 용량 기준 측정

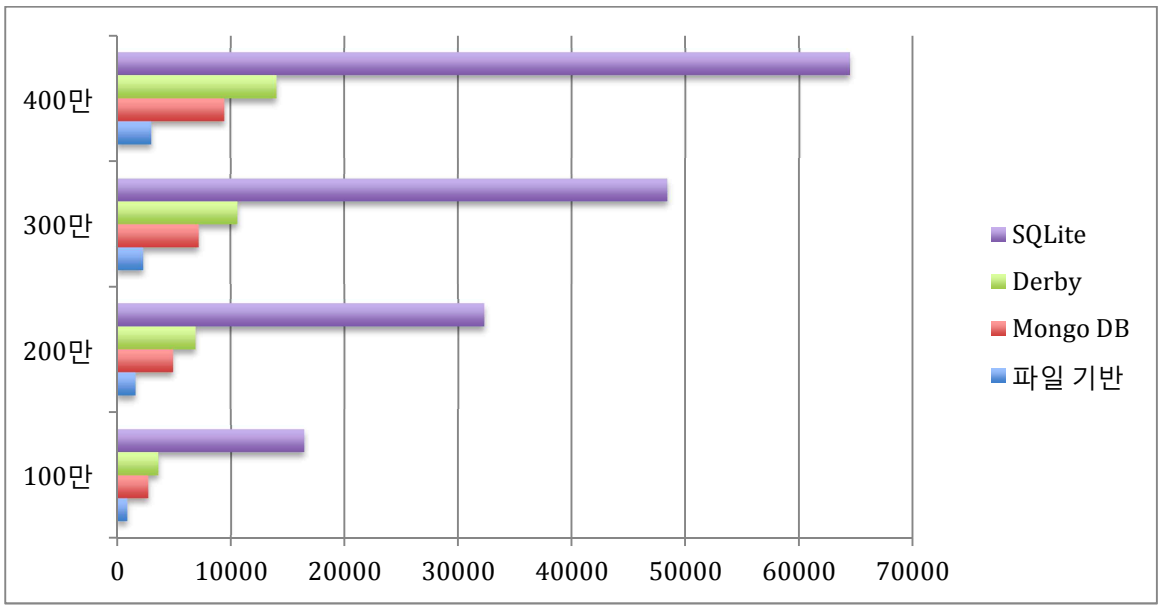


그림 70. **저장소 성능 비교**

(a) File DB

- 100,000 레코드 입력 단위 시간 측정

레코드 수	단위 시간 (ms)
0	3
100,000	910
200,000	1620
300,000	2309
400,000	3002
500,000	3697
600,000	4382
700,000	5175
800,000	5864
900,000	6559
1,000,000	7310

(b) Mongo DB

- noSQL Mongo DB로 100,000 레코드 입력 단위 시간 측정

레코드 수	단위 시간 (ms)
0	2
100,000	2717
200,000	4390
300,000	7180
400,000	9431
500,000	11655
600,000	14008
700,000	16252
800,000	18508
900,000	20932
1,000,000	23344

(c) Derby

- Derby로 100,000 레코드 입력 단위 시간 측정

레코드 수	단위 시간 (ms)
0	96
100,000	29777
200,000	54749
300,000	79646
400,000	104866
500,000	128829
600,000	153420
700,000	178328
800,000	203377
900,000	227951
1,000,000	250843

(d) MySQL

- MySQL로 100,000 레코드 입력 단위 시간 측정

레코드 수	단위 시간 (ms)
0	14
100,000	16350
200,000	32237
300,000	48377
400,000	64444
500,000	80288
600,000	96231
700,000	112074
800,000	128412
900,000	144905
1,000,000	163327