0925 Cloud

— 배운 내용 : On-Premise, Transparency(투명성, 알지 못해도 사용 가능한 특징), 그리드 컴퓨팅, 확장성(Scale Up, Scale Out), Vendor lock in, 클라우드의 보안, 클라우드의 종류 구분(IaaS, PaaS, SaaS), 멀티 클라우드, Public, Private, Hybrid Cloud, 가상화, Cloud Native, DevOps, CD, Microservices, Container, IT 서비스 개발 및 방식, Cloud Native 적용

- 중요 내용 : Transparency, 확장성(Scale UP 과 Scale Out), Vendor Lock In 특성, 클라우드의 종류 구분(IaaS, PaaS, SaaS), Hybrid cloud, 가상화, Cloud Native, DevOps, CD, Microservices, Container, IT 서비스 개발 및 방식, Cloud Native 적용

⇒ 클라우드의 보안 거버넌스(governance)

- 기업의 경영진이 클라우드를 이용할 때의 위험을 주체적이고 적절하게 관리하기 위한 구조를 구축하고 운영 하는것

⇒ 멀티 클라우드

- 클라우드 하나에 서비스를 전부 이용하는게 아니라 동일한 서비스를 여러 클라우드에 사용하는게 멀티 클라우드
- 보통 트리거 방식으로 사용해서 기존의 클라우드를 사용할 수 없게 되었을 때 다른 클라우드의 서비스를 사용하는 방식

⇒ Open Source 의 장점

- 빠른 업데이트 - 가장 큰 장점
- 비용 문제

⇒ SI(System Integrity, 시스템 통합)

**** Cloud**

1. On-Premise

⇒ 기업이 자체 데이터 및 솔루션을 저장하기 위해서 데이터 센터(전산실)를 구축해서 IT 서비스를 수행하는 방식

⇒ 하드웨어를 포함한 모든 자원에 대한 초기 투자 비용과 탄력적이지 않은 제한된 용량으로 인해 지속적인 관리 비용이 증가하는 단점이 있지만 기업에 내재된 서비스를 통해 품질 및 보안에 대한 신뢰도는 높음

- 초기 투자 비용은 On-Premise 가 높지만 오랫동안 사용한다면 이 방식이 이득

⇒ On-Premise 는 설계를 할 때 자원 사용량을 최대로 해서 설계

2. Cloud Computing

⇒ 컴퓨터와 소프트웨어를 자신이 소유하는 것이 아니라 네트워크를 통해 클라우드 사업자의 컴퓨터와 소프트웨어를 서비스로서 사용하는 것

⇒ IT 자산을 소유하는 것이 아니라 서비스로 이용하는 모델

⇒ 사용자는 네트워크와 인터넷 그리고 자원이 어디에 있는지 몰라도 사용할 수 있음

- Transparency(투명성) - 사용자는 어디에 있는지, 무엇인지 알지 못해도 사용할 수 있는 특성

1) 클라우드가 등장하기까지의 흐름

⇒ 메인 프레임

- 서버는 단일 서버

⇒ 클라이언트 서버 - 서버 컴퓨팅

- 서버가 여러 개 일 수 있음
- 여러 클라이언트의 요청을 middle ware(혹은 load balancer) 가 여러 서버에게 나눠주는 방식

⇒ 네트워크 컴퓨팅 - 그리드 컴퓨팅

- 작업을 나눠서 여러 processor 를 가지고 처리하는 방식

⇒ 클라우드

2) 클라우드가 보급된 배경

⇒ 저렴해진 자원

⇒ 네트워크의 발달

3) 정의

⇒ NIST(미국 국립 표준 기술 연구소)

- 공유 구성이 가능한 컴퓨팅 리소스의

4) 특징

⇒ 주문형 셀프 서비스 : 사업자와 직접 상호작용하지 않고 사용자의 개별 관리 화면을 통해 서비스를 이용할 수 있음

⇒ 광범위한 네트워크 접속 : 모바일 기기 등의 다양한 디바이스를 통해 서비스에 접속할 수 있음

⇒ 리소스 공유 - 데이터의 공유와 하드웨어, 소프트웨어를 공유

⇒ 신속한 확장성

- Scale Up : 처리 능력을 높이는 것
- Scale Down : 처리 능력을 낮추는 것

⇒ 측정 가능한 서비스 : 이용한만큼 요금을 지불

5) 장점

=> 경제성

=> 자동화 : 업데이트, 보안 패치, 백업 등

=> 이동성 : 웹을 이용할 수 있는 모든 장치에서 사용이 가능

=> 리소스 공유

=> 가용성 : AWS 의 경우 99.99% 이상의 가용성을 제공함

=> 빠른 구축 속도

⇒ 확장성

- Scale Up : 하드웨어의 성능을 업그레이드 하는 것
- Scale Out : 동일한 컴퓨터를 추가(컴퓨터 갯수 증가)하는 것

⇒ 중복성 또는 탄력성 : Business Continuity Management(BCM)

- AWS 의 글로벌 인프라

6) 단점

⇒ 인터넷 엑세스 - 인터넷이 되지 않는 환경에서 사용 불가능

⇒ Security - 보안

⇒ Privacy

⇒ Vendor Lock In

- vendor(제공자)가 제공한 상황, 환경 등을 사용자가 함부로 바꿀 수 없음
- 원하는 서비스를 마음대로 바꿔서 사용할 수 없음

7) 클라우드와 On-Premise

⇒ 비용 문제

- On-Premise : 초기 투자 비용이 많이 발생함, 추가적인 관리 비용(관리 인원이 필요)이 발생
- Cloud : 초기 투자 비용은 없음
- 보통 5년 정도 동일한 시스템을 이용한다면 그 이후에는 On-Premise 가 비용 절감 효과를 보임

⇒ 확장성 비교

- On-Premise
  - 자사 시스템의 개별 요구 사항에 따라 설계
  - 시스템의 도입과 확장에는 고도의 기술을 갖춘 인재가 필요함
- Cloud
  - 클라우드 사업자가 설계 및 구축을 하기 때문에 전문 인력이 필요하지 않음
  - 중소 기업이 대기업에 필적하는 규모의 서비스를 이용할 수 있음

⇒ 안정성과 신뢰성

- 클라우드의 리스크
  - IaaS 의 리스크로는 클라우드 사업자의 하드웨어 장애로 인한 데이터의 손실이나 서비스 중단 등의 문제가 발생할 수 있음
  - 가상 서버를 백업 하거나 쉽게 가상 서버 환경을 구축하기 위한 탬플릿을 마련하는 것이 대책
- 클라우드의 보안 거버넌스(governance)
  - 기업 사용자는 자사가 보유한 정보의 관리와 처리를 클라우드 사업자에게 맡겨버리기 때문에 보안 등의 리스크를 통제할 수 없다는 문제가 발생
  - 클라우드의 보안 거버넌스(governance - 기업의 경영진이 클라우드를 이용할 때의 위험을 주체적이고 적절하게 관리하기 위한 구조를 구축하고 운영 하는 것)의 관점에서 바라보면 클라우드 서비시의 incident 와 서비스 복구와 같은

사항들은 제어하기 어렵고 클라우드 사업자의 갑작스런 파산이나 서비스 중단과 같은 서비스의 연속성에 대한 리스크가 존재함

- 이런 이유들로 인해 대부분은 알려진 클라우드를 사용하고 멀티 클라우드 형태를 취하게 됨
  - 하나의 클라우드를 사용하거나 서비스를 다른 클라우드로 분리하는게 아니라 동일한 서비스를 여러 개의 클라우드에 상요하는게 멀티 클라우드
- 사업자 및 이용자 측에 잠재된 보안 위험 요소를 확인해두고 클라우드 사업자와 이용자 사이의 책임 분계선 등 이용자가 관리할 수 있는 범위를 파악한 후 보안 대책과 백업을 마련할 필요가 있음

	SaaS	PaaS	IaaS
사용자 관리	사용자 책임	사용자 책임	사용자 책임
애플리케이션 문제	사업자 책임	사용자 책임	사용자 책임
개발 도구에 문제	사업자 책임	사용자 책임	사용자 책임
미들웨어에 문제	사업자 책임	사업자 책임	사용자 책임
OS 에 문제	사업자 책임	사업자 책임	사용자 책임
하드 웨어에 문제	사업자 책임	사업자 책임	사업자 책임

8) 클라우드 서비스의 종류

- IaaS, PaaS, SaaS
- I 는 Infrastructure, P 는 Platform, S 는 Service

	IaaS	PaaS	SaaS
직접 구축해야 하는 것			
애플리케이션	애플리케이션	애플리케이션	제공
데이터	데이터	데이터	제공
런타임(실행 환경)	런타임	제공	재공
미들 웨어	미들 웨어	제공	제공
OS	OS	제공	제공

가상화	제공	제공	제공
서버	제공	제공	제공
스토리지	제공	제공	제공

⇒ IaaS(Infrastructure as a Service)

- 사용자에게 인프라 자원을 사용할 수 있도록 해주는 클라우드 서비스
- 컴퓨터만 빌려주는 개념
- 가장 대표적인 서비스가 AWS 의 EC2(Elastic Compute Cloud)
- 모든 CSP 사업자는 이 서비스를 퍼블릭으로 제공함

⇒ PaaS(Platform as a Service)

- 서비스 개발자가 애플리케이션 개발, 실행, 관리 등을 할 수 있도록 안정적인 플랫폼 또는 프레임 워크를 제공하는 클라우드 서비스 방식
- 개발 환경을 제공하는 개념
- 원하는 소프트웨어를 직접 개발해서 사용하기 때문에 SaaS 보다 자유도가 높지만 다른 환경으로의 마이그레이션이 어려울 수 있음
- GCP(Google Cloud Platform), ServiceNow(아마존), Oracle DBMS on Cloud(OCI), 세일스포스의 Force.com, 사이보우즈의 Kintone, Cloud Foundary 등
    - 개발 도구 - SDK, 프레임워크(Ruby on Rails, Spring, Node.js, Eclipse 등) 제공
    - 핵심 기능 - 프로그래밍 언어(Ruby, Java, Python 등), Application Server(Apache, Tomcat 등), 데이터 베이스(MySQL, PostgreSQL, MongoDB, Amazon RDS, Oracle, MS SQL server 등), 메시지 미들웨어(RabbitMQ, Amazon SQS, Kafka 등), 애플리케이션 통합 소프트웨어, 비지니스 프로세스 관리, 데이터 통합, 관리 파일 전송, 보안, 테스트 환경 등을 제공함
- Chat GPT 의 code interpreter 도 여기에 해당

⇒ SaaS(Service as a Service)

- 사용자가 필요로하는 소프트웨어를 제공하는 서비스

- 하나의 서버를 여러 기업에서 공유하는 것을 전제로 한 멀티 터넌트 서비스를 제공하지만 데이터는 기업 사용자 별로 분리되도록 설계해서 보안성을 확보함
- 전자 메일이나 그룹웨어, CRM 서비스 등을 예시로 들 수 있음
- SaaS 의 대표적인 서비스가 Google Apps, Salesforce, MS Office 365, Naver Office 등

⇒ DaaS(Desktop as a Service)

- 데스크탑을 빌려주는 서비스
- SaaS 와 IaaS 의 결합
    - 원하는 소프트웨어를 사용할 수 없는 SaaS 와 어느 정도의 기본 제공을 받지 못하는 IaaS 의 단점을 보완하는 방식

9) 클라우드 이용 모델

⇒ Public Cloud

- 클라우드 서비스 공급자(AWS, GCP, Azure, Oracle, Naver 등)가 시스템을 구축하고 인터넷 망 등의 네트워크를 통해서 불특정 다수의 기업과 개인에게 서비스를 제공하는 형태
- 확장과 축소가 쉬운 구조

⇒ Private Cloud

- 폐쇄적인 구조
- 특정 기업의 특정 사용자만을 대상으로 하는 클라우드 서비스
- 자사가 보유하고 운용하는 형태(On-Premise Private Cloud 방식)로 만들 수 있고 클라우드 사업자가 장비를 보유하고 프라이빗 클라우드 서비스를 제공하는 형태(Hosted Private Cloud 형태)로 구축할 수도 있음

⇒ Community Cloud

- 공통의 목적을 가진 특정 기업들이 클라우드 시스템을 구축해서 데이터 센터에서 공동 운영하는 형태

⇒ Hybrid Cloud

- Private Cloud 와 Public Cloud 를 조합하여 사용하는 방식
- 기업 내부의 중요하고 민감한 데이터에 대한 처리 작업은 통제력을 강화하기 위해서 Private Cloud 를 사용하고 일반 업무 데이터 처리와 같이 보안 요구 사항이 낮거나 workload 가 지속적으로 증가하는 작업, 애플리케이션을 여러 곳에 배포해야 하는 작업은 Public Cloud 로 구성
- Private 와 Public Cloud 중 어느 부분에 넣을지는 보안도 중요하지만 확장성도 중요한 고려 요소

10) 가상화

⇒ 클라우드를 지탱하는 2대 기술 중 하나

- 컴퓨터의 물리적인 메모리와 하드 디스크, 운영체제 등 다양한 것들을 소프트웨어로 대체하는 기술

⇒ 물리적인 컴퓨터 1개를 가지고 여러 개의 논리적인 컴퓨터를 만드는 기술

⇒ 복제가 쉽고 대수를 늘리거나 줄이는 것도 쉬워짐

11) 분산 처리와 로드 밸런서(load balancer)

⇒ 분산 처리는 여러 대에 나누어서 처리하는 기술

⇒ 여러 대에서 나눠서 처리할 수 있도록 분배하는 장치가 Load balancer

12) 이중화

⇒ 시스템이나 서버에 문제가 생겨도 계속 가동할 수 있도록 조치를 하는 것

3. Cloud Native

1) 개요

⇒ 동적인 환경(클라우드 환경)에서 확장 가능한 애플리케이션을 개발하고 실행할 수 있도록 해주는 기술

⇒ container, service mesh, microservice, immutable infra, declarative(선언형) API 등이 기술에 해당

⇒ 회복성, 관리 평의성, 가시성을 갖춘 느슨한 결합(Loosly Coupling)을 가진 시스템

⇒ 벤더 중립적인 오픈 소스 프로젝트 생태계

2) Pillas of Cloud Native

⇒ 클라우드의 핵심 4대 기술

⇒ DevOps

⇒ Microservices

⇒ Containers

⇒ Continuous Delivery

3) DevOps

⇒ IT 서비스의 설계, SW 개발, 릴리즈 및 운영에 이르기까지 전체 서비스 수명 주기에 함께 참여하는 IT 서비스 운영 및 갭라 엔지니어의 업무 방식

⇒ Plan → Code → Build → Test → Release → Deploy → Operate → Monitor

4) CD

⇒ DevOps 의 자동화 부문의 기원

⇒ 지속적 통합(Continuos Integration - CI) 과 연계

- 소스 제어에 자주 코드를 commit
- 각 코드 commit 에 자동화된 빌드를 실행
- 결과적으로 더 빠른 오류 감지가 가능해짐

⇒ 지속적 전달(Continuous Delivery)

- 최종 사용자를 위한 릴리스 코드 빌드
- 이로 인한 결과는 최종 사용자에게 소프트웨어를 더 빠르게 제공

5) Micro Services

⇒ 애플리케이션을 상호 독립적인 최소 구성 요소로 분할하는 것

- 서비스의 개념으로 보는게 아니라 독립적으로 나눌 수 있는지를 보는 것
- 마이크로 서비스의 대표적인 예시는 로그인
    - 이것 하나만으로 독립적인 서비스의 기능을 하지는 못하지만 다른 기능들과 개별적으로(독립적으로) 존재하며 사용 가능

⇒ 느슨한 결합 - 한 쪽의 변화가 다른 족에 영향을 최소로 주어야 한다는 결합

- 결합을 아예 하지 않을 수는 없음

6) Container

⇒ 실행에 필요한 모든 파일을 포함해서 전체 실행 환경에서 애플리케이션을 패키지화하고 분리하는 기술

7) IT 서비스 개발 및 구현 방식의 변화

⇒ 개발 방식

- 약 36% - DevOps/DevSecOps : 개발과 운영을 한꺼번에 고려해서 개발
- 약 32% - Agile/Scrum : 애자일 방식 중 스크럼 방식
    - 업무 별로 나누는게 아니라 팀원들이 모여서(스크럼) 서비스를 개발하는 방식
    - 서비스 단위로 구분
- 약 13% - Kanban
    - 문서 작업 등이 아니라 먼저 동작하는 애플리케이션부터 만드는 방식
- 약 10% - Waterfall
    - 고객의 요구 사항 분석, 설계, 구현, 테스트 등의 순서를 따르는 방식
    - 앞으로의 방향으로만 진행하기에 폭포
    - 중간에 잘못을 찾으면 처음 단계부터 다시 수행
- 약 5% - Water/Scrum/Fall
    - waterfall, scrum 등 여러 방식을 섞어서 사용하는 방식
- 약 4% - Lean(린)

⇒ 이전에는 DBA, Front end, Back end 와 같이 업무 위주로 구분했지만 최근에는 Service 단위로 구분해서 모든 팀원이 모여서 서비스를 개발

⇒ Developement Process

- waterfall → agile → devops 형태로 변하는중

⇒ Application Architecture

- Monolithic → N-tier → Microservices
- monolithic : 모든 서비스를 하나로 묶는 형태
- N-tier : 계층 별 분할

⇒ Deployment and Packaging

- Physical server → Virtual server → Containers

⇒ Application infra structure

- Data center → Hosting → Cloud

8) Cloud Native 적용이 필요한 이유

⇒ 서비스 배포 시간의 단축 : Containers + Microservices 를 통해 개발 팀과 운영 팀 사이의 의사 소통이 향상

⇒ 애플리케이션 및 서비스의 현대화

⇒신속한 신규 서비스 개발 사이클

- 풍부한 기술 생태계
  - 대규모 커뮤니티
  - kubenetes 및 CNCF slack 에 많은 개발자가 참여
  - 충성도 높은 커뮤니티 활동가들
- 오픈 소스 기반
  - 개발자에게 친숙한 개발 환경 및 운영 협업 체계
  - 풍부한 개발 인력 pool

⇒ 사업 성장을 위한 조직 문화 혁신 촉진

- 조직의 혁신을 가속화하기 위해 새로운 문화, 기술 및 프로세스를 제공
- DevOps, CI/CD, Containerization 은 서비스 개발 조직의 현대화를 촉진
- 이전보다 훨씬 빠르게 조직 문화 및 서비스 문화 변화 촉진

⇒ IT 목표

- 민첩성과 생산성
- 복원력과 확장성
- 최적화와 효율성

⇒ 사업 성과

- 시장의 성장
- 위험의 완화
- 비용 절감

9) CNCF Trail Map

⇒ CNCF : Cloud Native Computing Foundation

- 컨테이너 기반의 클라우드 네이티브 오픈소스 생태계

⇒ Product → Development → Capacity planning → Testing + Release Procedures → Postmortem / Root cause Analysis → Incident Response → Monitoring

4. DevOps

1) Agile and ITIL

⇒ Agile

- 반복적인 개발 주기와 자기 조직화된 팀을 강조하는 일련의 관행
- manifesto
  - 개인과 개인 사이의 상호작용이 프로세스 및 툴 보다 우선함
    - 좋은 도구를 사용하는 것보다 개발자들 사이의 상호작용이 더 중요하다는 의미
  - 작동하는 소프트웨어가 포괄적인 문서보다 우선함
  - 고객과의 협업이 계약 협상보다 우선함
  - 변화에 대응하는 것이 계획을 따르는 것보다 우선함

⇒ ITIL - IT Infrastructure Library

- IT 인프라 운영의 최고 사례를 모아서 발간한 것
- 2019년 ITIL ver.4 부터 Agile, Lean, DevOps 등의 방법론과의 연계성을 강화하는 Service Value System 개념으로 체계 변화

2) DevOps

⇒ IT 서비스 설계, SW 개발, 릴리즈 및 운영에 이르기까지 '전체 서비스 수명 주기에 함께 참여'하는 IT 서비스 운영 및 개발 엔지니어의 업무 방식

⇒ 애플리케이션 개발에서 프로덕션 운영에 이르는 전체 프로세스를 소유하는 방법론

⇒ 일련의 기술 구현을 넘어서 문화와 프로세스의 완전한 변화를 요구

⇒ 기술이 아닌 철학이며 변경이 아닌 변화라고 함

3) 변화

- SOA ⇒ Microservices
- Agile Team ⇒ Mode2, DevOps, Agile Mindset
- Cloud Infra Automation ⇒ CI/CD

- 이 모든 것이 Enterprise DevOps 로 변화