

# 2021년 광주대학교 특강

OSS를 활용한 Cloud Native 개발

---



이기하

dasomell@gmail.com

2021.10

# 목 차

---

1. 발표자 소개
2. Cloud Native 개요
3. Cloud Native 적용 프로세스
4. Cloud Native 구축 수행
  1. 요구사항 분석 및 도출
  2. 현황 분석
  3. 기술 범위 산정
  4. 작업 범위 및 프로젝트 계획 수립
5. Cloud Native 아키텍처 설계
6. 개발환경 구축
7. POC
8. 테스트/운영 환경 구축
9. 개발 시나리오 수행
10. 보안/성능/단위/통합 테스트
11. 모니터링 및 분석

# 1. 발표자 소개

## ❑ 現한화시스템 ICT부문(2021~)

- HKS(Hanwha Kubernetes Service) Platform 개발 리딩

## ❑ 前SK주식회사 C&C(2012 ~ 2021)

- Cloud 프로젝트 다수 구축(2017 ~ 2020)
  - 사내 강의 다수
  - 사내 개발자 대회 다수 분야 3등(2018)
- ## ❑ 〈나도 해보자! 시리즈〉 오픈커뮤니티 세미나 발표
- 나도 해보자! 표준프레임워크 개발환경 구축
  - 나도 해보자! Cloud Project with Kubernetes 등

## ❑ 오픈플랫폼(PaaS) 전문가과정 강의(2016)

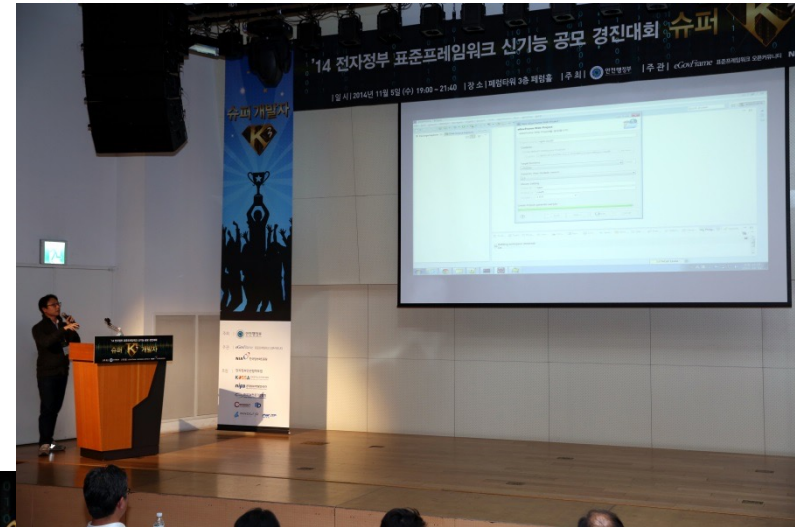
## ❑ 슈퍼개발자K 시즌3 동상 수상(2014)

## ❑ 現오픈커뮤니티 리더(2015 ~)

## ❑ 前T-Hub(SK그룹 기술커뮤니티)

- DevOps Master(2020~2021)

표준프레임워크 오픈커뮤니티  
**eGovFrame**



## 2. Cloud Native 개요

### ❑ Cloud Native 로의 환경 변화

- 클라우드는 이제 컴퓨팅 파워를 중심으로 제공하던 클라우드 컴퓨팅 시대에서, 모든 것을 클라우드에서 제공할 수 있는 XaaS 클라우드 서비스 시대로 변화
- 국내에서도 클라우드 시장을 둘러싸고 IaaS(서비스형 인프라), PaaS(서비스형 플랫폼), SaaS(서비스형 소프트웨어) 기업의 경쟁 행보가 빠르게 전개
- 경쟁행보는 기업들의 디지털 트랜스포메이션을 촉발시키는 밑거름 역할
- SW 유통의 혁명을 가져온 컨테이너 기술이 등장한 이후, 소프트웨어 개발에 있어서 컨테이너에 기반한 클라우드 네이티브 어플리케이션은 선택이 아닌 필수
- 비교적 보수적인 금융권에서도 고객에게 보다 나은 서비스를 제공하기 위해 이러한 기술 트렌드를 적극 수용하여 경쟁사에 대응

## 2. Cloud Native 개요

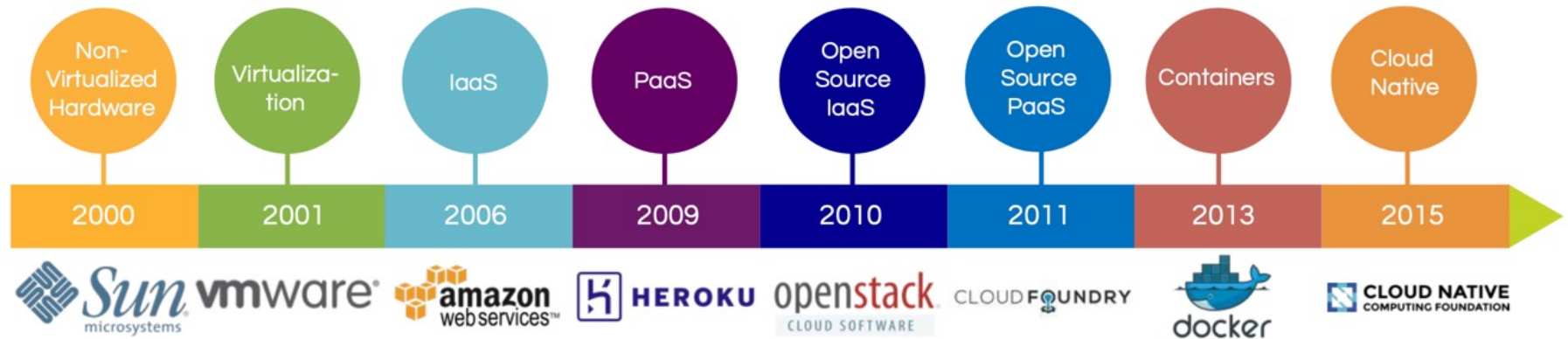
### ❑ Cloud Native 정의 및 목적

- Cloud Native는 클라우드 환경에서 그 이점을 최대한으로 활용하여 소프트웨어를 개발하고 운영하는 방식
- MSA, 애자일, DevOps, 쿠버네티스 같은 기술, 문화, 아키텍처를 도입하여 생산성, 확장성, 가용성, 경제성 등에서 큰 효과를 누릴 수 있음
- CNCF(Cloud Native Computing Foundation)재단은 클라우드 네이티브에 대해 다음과 같이 정의
  - 클라우드 네이티브 기술은 조직이 Private, Public 그리고 Hybrid-Multi 클라우드와 같은 현대적이고 동적인 환경에서 확장 가능한 애플리케이션을 개발하고 실행 가능
  - 컨테이너, 서비스 메쉬, 마이크로서비스, 불변(Immutable) 인프라, 그리고 선언형(Declarative) API가 이러한 접근 방식의 예시
  - 이 기술은 회복성, 관리 편의성, 가시성을 갖춘 느슨하게 결합된 시스템을 가능하게 한다. 견고한 자동화 기능을 함께 사용하면, 엔지니어는 영향이 큰 변경을 최소한의 노력으로 자주, 예측 가능하게 수행 가능
  - Cloud Native Computing Foundation은 벤더 중립적인 오픈 소스 프로젝트 생태계를 육성하고 유지함으로써 해당 패러다임 채택을 촉진한다. 우리 재단은 최신 기술 수준의 패턴을 대중화하여 이런 혁신을 누구나 접근 가능

## 2. Cloud Native 개요

### □ Cloud Native 발전동향

- 2015년 처음 Cloud Native라는 용어를 사용한 리눅스는 CNCF(Cloud Native Computing Foundation)재단을 만들었고, 이 곳에서 700개가 넘는 클라우드 공급자와 기술 기업들이 참여하여 운영



- Cloud Native 아키텍처는 기업의 디지털 전환에 중요한 도구로 활용되며 이를 기반으로 인공지능 및 머신러닝에 관련된 다양한 기술도 꾸준히 개발
- IT, 인터넷 기업뿐만 아니라 수많은 유통, 이커머스 기업도 클라우드 네이티브 기술을 채택
- 시장조사기관 가트너에 따르면 2022년까지 전 세계 조직 중 75% 이상이 클라우드 네이티브 컨테이너를 도입할 것으로 전망되며, 이는 2021년 현재 30% 미만인 수준 대비 크게 증가하는 수치

## 2. Cloud Native 개요

### ❑ Cloud Native 구성요소

- 비즈니스 시스템은 비즈니스 속도와 성장을 가속화하는 전략으로 발전하고 있고 아이디어를 즉시 출시하는 것이 중요
- Cloud Native는 속도와 민첩성에 직접 작용
- 이를 이루는 요인은 핵심요소는 보통 '컨테이너', '마이크로서비스', 'CI/CD', 'DevOps' 4가지를 이야기하는데 각각의 특성과 장점들이 있고, 함께 구성되고 연계될 때 시너지 효과

### Four key principles of cloud-native development



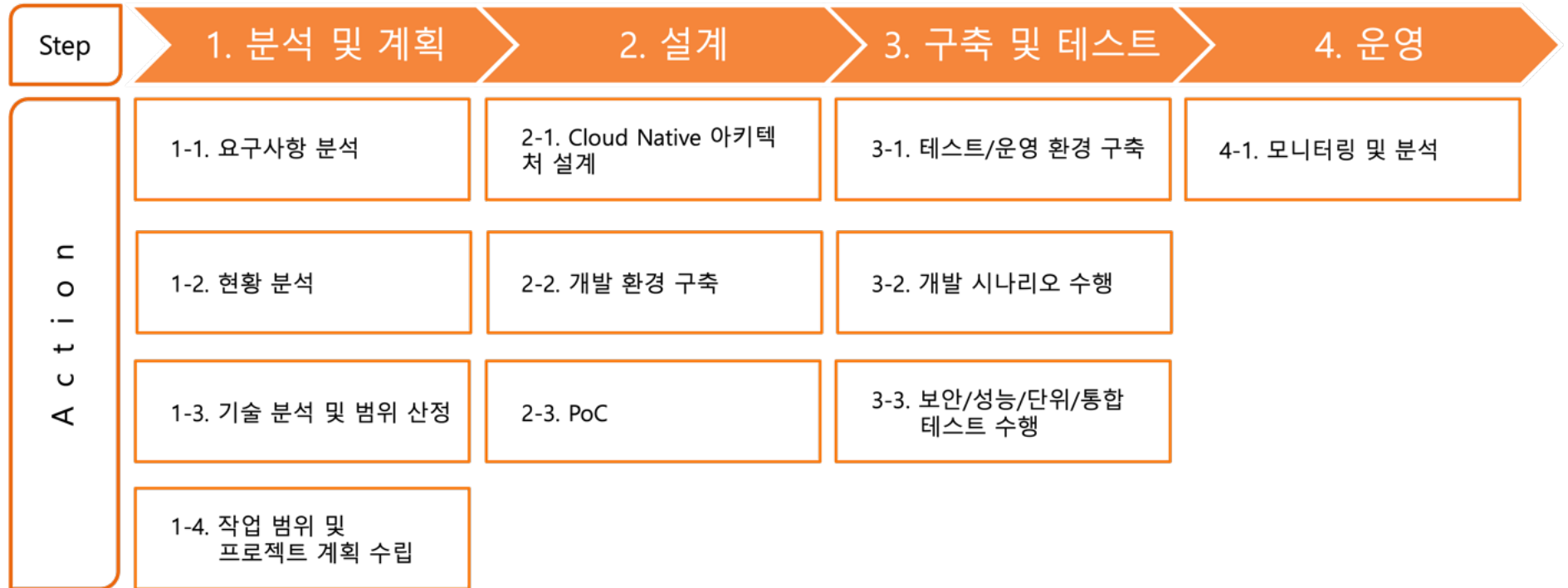
## 2. Cloud Native 개요

- Cloud Native를 구현하는데 핵심은 컨테이너화
- 컨테이너화를 통해 애플리케이션의 소프트웨어 코드와 이를 구동하는데 필요한 모든 구성 파일, 라이브러리, 디펜던시를 플랫폼 혹은 서버의 클라우드 네트워크에서 원활하게 이동하고 하나의 ‘파일’로 만드는 것
- 애플리케이션 개발 및 디플로이먼트 과정의 간소화도 가능
- 컨테이너화는 또한 ‘마이크로서비스’의 성장으로 연결
- 컨테이너 혹은 소프트웨어 패키지 내에서 개발되는 마이크로서비스는 애플리케이션의 특정 부분이 독립적으로 작동할 수 있도록 지원
- 이를 통해 클라우드를 기반으로 구동되는 애플리케이션은 더욱 기민해지고 고장을 쉽게 분리할 수 있으며 유지보수의 용이성도 한층 높아짐



### 3. Cloud Native 적용 프로세스

#### □ Cloud Native 적용 프로세스



### 3. Cloud Native 적용 프로세스

#### □ Cloud Native 적용 프로세스(계속)

##### - 분석 및 계획

단계	내용
요구사항 분석 및 도출	적용대상 시스템의 목적과 적용 범위 및 계획 등에 대한 고객 요구사항을 도출하여 분석
현황 분석	대상 시스템에 대한 상세정보, 환경, 시장현황 분석
기술 분석 및 범위 산정	요구사항 및 현황 분석 정보를 기반으로 적용 기술과 전략을 도출
작업 범위 및 프로젝트 계획 수립	도출된 기술과 분석된 현황정보를 기반으로 작업 범위와 일정에 따른 계획 수립

##### - 설계

단계	내용
Cloud Native 아키텍처 설계	전략을 기반으로 향후 개발되고 운영될 시스템의 아키텍처를 정의
개발 환경 구축	PoC 수행 및 서비스를 개발할 수 있는 기본 환경을 구축
PoC (Proof of Concept)	확인이 필요한 설계 및 기술에 대한 개념 검증(Proof of Concept) 및 Pilot 을 구축

### 3. Cloud Native 적용 프로세스

#### □ Cloud Native 적용 프로세스(계속)

##### - 구축 및 테스트

단계	내용
테스트/운영 환경 구축	로컬/개발환경에서 입증된 구축환경을 클라우드 테스트/운영환경에서 재구축 단, 운영의 특수한 인프라환경이 있다면 예외적으로 적용
개발 시나리오 수행	협의된 일정에 따른 Agile, DevOps 방식 어플리케이션 개발 진행
성능/보안/단위/통합 테스트	사 내외 환경과 보안을 고려한 상태에서 Cloud Native 에 최적화 된 테스트 수행

##### - 설계

단계	내용
모니터링 및 분석	기 구축 된 Cloud Native 모니터링 시스템을 통한 운영 환경 체크하고 시스템 성능 및 운영 시스템의 흐름 분석 후 리포팅

## 4.1 요구사항 분석 및 도출

### □ 정의

- 도출된 요구사항 간 상충 해결, 소프트웨어 범위 파악, 외부 환경과의 상호작용 분석
- 개발 대상에 대한 요구사항 중 명확하지 않거나 이해되지 않는 부분을 발견 및 걸러내는 과정

### □ 수행 상세

- 요구사항 분류
  - 기능/비기능 분류
  - 요구사항이 소프트웨어에 미치는 영향 범위 파악
  - 생명 주기동안 변경이 발생하는지 확인
  - 하나 이상의 상위 요구사항에서 유도 or 다른 원천으로부터 직접 발생인지 분류
- 개념 모델링 생성 및 분석
  - 요구사항을 더 쉽게 이해할 수 있도록 현실 세계 상황을 단순화, 개념적으로 표현
  - 객체 모델, 데이터 모델, 유스케이스 다이어그램, 데이터 흐름 모델, 상태 모델, 목표 기반 모델, 사용자 인터페이스 등 다양한 개념 모델 작성 가능
  - 모델링 표기는 UML 등을 사용

## 4.1 요구사항 분석 및 도출

- 요구사항 할당
  - 아키텍처 구성요소를 식별하는 활동
  - 다른 구성요소와 어떻게 상호작용하는지 분석 → 추가 요구사항 발견
- 요구사항 협상
  - 두 이해관계 사이 상충될 경우 적절한 지점 합의하기 위한 기법
  - 각각 우선순위 부여 → 중요도 파악 가능 → 문제 해결 도움
- 정형 분석
  - 형식적으로 정의된 의미를 지닌 언어로 요구사항을 표현
  - 구문과 의미를 갖는 정형화된 언어를 사용하여 수학적 기호로 표현
  - 요구사항 분석의 마지막 단계

### □ 기술 및 기법

- 요구사항 정의서 작성 지침
  - 요구사항 정의서는 향후 작업의 기준선이 되므로 최대한 구체적으로 명확하게 작성
  - 문장은 간결하고 직설적으로 표현
  - 실제 사용하는 용어로 표현

## 4.2 현황 분석

### □ 정의

- 대상 기업의 시스템이나 사용중인 Private/Public 클라우드 환경, 장비 및 어플리케이션 시스템에 대한 정보를 수집하고 분석하여, Cloud Native 어플리케이션 구축을 위한 전략 기초자료를 생성

### □ 수행 상세

- 환경 분석
  - 확보된 요구사항을 기반으로 시스템 관련 정보를 수집
  - 인프라 환경 및 인력에 대한 정보를 기반으로 실행 가능한 개발 환경을 식별

### □ 기술 및 기법

- 기술 동향 분석

항목	내용	비고
인프라 기술 동향	인프라 자동화 아키텍처 동향	시장 및 벤더 현황
	하이브리드 멀티 클라우드 동향	시장 및 벤더 현황
S/W 기술 동향	오픈소스 소프트웨어 동향	시장 및 벤더 현황
	CNCF 동향	시장 및 벤더 현황

## 4.2 현황 분석

### - 타사 구축 사례 분석

항목	사례 예시
동종업계 구축 사례	금융사 Cloud Native 애플리케이션 개발 프로젝트 사례
경쟁사 구축 사례	Private 클라우드 + 오픈소스 소프트웨어 구축 사례

### - 개발/운영 환경 및 시스템 분석

항목	내용	비고
어플리케이션 정보	어플리케이션 명칭 및 설명	
	개발환경 정보	
	어플리케이션 특징	이중화, 데이터 크기, 처리량 등
	어플리케이션의 크기와 복잡도 및 상호의존성 정도	
	사용자의 유형	예) 내부용, 외부 고객용 등
	주요 적용 기술	예) Spring, Vue.js
	인터페이스 정보	예) portal 연계, 기관 연계
	기타 필요한 정보	
인프라 환경	IaaS, PaaS 벤더명 및 제품명	AWS, Azure, GCP, HKP, OpenShift 등
	운영체제 종류 및 버전	예) CentOS 7.4.1708
	제품 구성 사양(CPU, Memory, Disk 등)	
	구성 관련 정보(이중화, 백업 등)	
소프트웨어 사용 정보	시스템 S/W 제품명 및 버전	예) Docker 10.1, Kubernetes 12.1
	사용 유형 및 의존성	
	라이선스 유형	예) MIT
기타	개발 관련 정보	
	운영 관련 정보	배포 주기등 운영 상 특징

## 4.3 기술 범위 산정

---

### □ 정의

- Cloud Native 어플리케이션을 개발하고 아키텍처를 구성하는데 필요한 기술 범위를 정한다.

### □ 수행 상세

- 분석된 요구사항 및 현황에 따라 적용 가능한 Cloud Native 기술을 선정하고 정의한다.
  - CNCF 최신 TrailMap 을 기반으로 대상을 식별
  - 분석된 OSS 현황을 기반으로 기술 선정
  - 활발한 사용성과 보안이 검증된 기술과 소프트웨어 우선 적용



## 4.3 기술 범위 산정

### □ 기술 및 기법

- CNCF TrailMap 기술 리스트

항목	기술	항목	기술
컨테이너화	Docker	분산 DB & 스토리지	Vitess
CI/CD	Argo		Rook
	Jenkins		Etcad
오케스트레이션 & 애플리케이션 등록	Kubernetes		KV
	HELM	스트리밍 & 메세징	gRPC
모니터링 & 분석	Prometheus		NATS
	Fluentd		Cloudevents
	JAEGER	컨테이너 저장소 & 런타임	Containerd
	OpenTracing		HARBOR
서비스 프록시 / 디스커버리 / 메시	Envoy		Cri-o
	CoreDNS		Nexus
	LINKERD	소프트웨어 배포	Notary
네트워킹 & 정책 & 보안	CNI		
	Open Policy Agent		
	Falco		

## 4.4 작업 범위 및 프로젝트 계획 수립

### □ 정의

- 프로젝트에서 Cloud Native개발/운영 환경 구축을 위한 작업 범위와 계획을 수립

### □ 수행 상세

- 작업 범위 설정
  - 프로젝트의 내용과 요구사항에 따라 기술 및 작업 범위를 어디까지 둘 것인지에 대한 정의
- 프로젝트 수행 전략 수립
  - Private, Public, Hybrid-Multi 클라우드 등 인프라 환경에 따라 수행 전략을 수립
  - Cloud Native, 오픈소스 및 Legacy 소프트웨어 활용 전략을 수립
- 프로젝트 작업 계획 수립
  - 인프라 및 소프트웨어 환경과 활용 경험에 따라 일정을 산정
  - Agile방법론과 DevOps 문화를 고려한 프로젝트 작업을 계획 수립

## 4.4 작업 범위 및 프로젝트 계획 수립

### □ 기술 및 기법

- 전략 수립 - 클라우드 환경에 따른 작업 범위

작업 / 환경	Private	Public	Hybrid-Multi
인프라 프로비저닝	IaaS, IaC 적용	IaaS, IaC 적용	IaaS, IaC 적용
컨테이너 오케스트레이션	Kubernetes 구축	관리형 Kubernetes 설치	K8S Core(+관리형 K8S)
DevOps Tool Chain	OSS DevOps Tool Chain 구축	클라우드 벤더 DevOps 솔루션 설치	OSS DevOps Tool Chain (+ 벤더 DevOps 솔루션)
CI/CD Pipeline	OSS CI/CD Pipeline 구축	클라우드 벤더 CI/CD 솔루션 설치	OSS CI/CD Pipeline (+ 벤더 CI/CD 솔루션)

## 4.4 작업 범위 및 프로젝트 계획 수립

### □ 기술 및 기법(계속)

#### - 전략 수립 - 기술 및 소프트웨어

구분	내용	비고
CNCF	분석된 요구사항과 현황에 따라 적용 가능한 CNCF Trail Map 단계적 적용	Ex) 모니터링: Prometheus + Grafana
기타 OSS	요구사항, Cloud Native 환경에 대한 적합성, 라이선스, 유지보수성 검토 후 적용	Ex) 관계형 데이터 베이스: MariaDB
Legacy	기존 소프트웨어 이식, 이관 및 대체 검토	

#### - Agile개발방법론 관리와 도구

관리도구	내용	비고
Redmine	이슈 등록/관리, 일정관리, 제한된 애자일 특화 기능 지원	OSS
Wiki사이트	산출물 관리를 위한 다양한 OSS 가 존재	OSS 사용 가능
Jira	이슈 관리, 애자일 기반 프로젝트를 위한 거의 모든 기능 지원	상업용 도구
Confluence	JIRA와 연동하여 프로젝트 관리 지원 및 산출물 관리 지원	상업용 도구

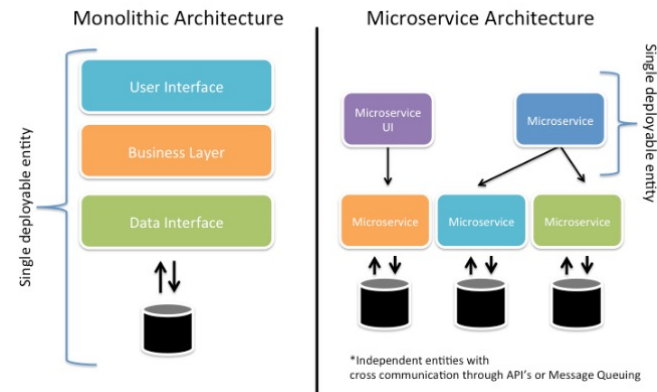
## 5. Cloud Native 아키텍처 설계

### □ 정의

- 클라우드의 이점을 최대한 활용할 수 있는 애플리케이션 개발과 운영 환경에 대한 구조를 설계
- Private/Public/Multi/Hybrid 등 각각 클라우드 환경마다 생산성, 적합성, 확장성, 회복성을 고려한 최적의 아키텍처를 설계
- Cloud Native를 위해 컨테이너, MSA, DevOps, CI/CD 등의 주요 기술요소들을 필요시 적극 활용

### □ 수행 상세

- 마이크로서비스 아키텍처 설계
  - 마이크로 서비스 애플리케이션은 도메인 단위로 쪼개진 여러 개의 서비스를 조합하여 구현 방식
  - 각 서비스마다 개별 DB를 가지며 개발, 테스트, 빌드, 배포까지 독립적으로 수행 가능해야 함



## 5. Cloud Native 아키텍처 설계

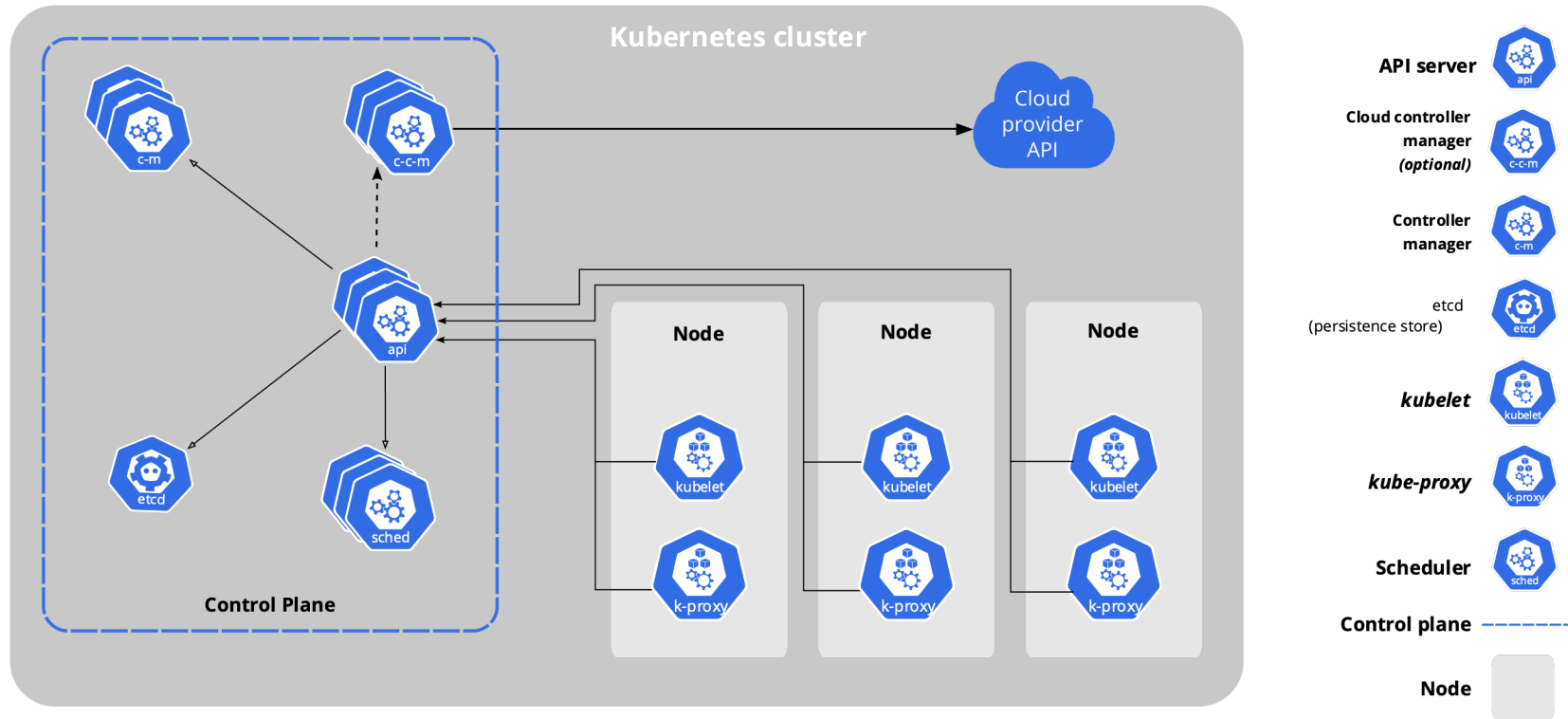
### - 마이크로서비스 아키텍처 설계 방법

설계 구분	내용
서비스 단위 구분	<p>도메인 주도 설계(Domain Driven Design) 로 서비스 단위를 경량화하여 구분한다.</p> <ul style="list-style-type: none"> <li>● 도메인 모델은 UML로 전체적인 흐름을 표현한 후 서브 도메인을 분리하고 바운디드 컨텍스트와 컨텍스트 맵을 통해 모델 관계를 정의</li> <li>● 업무 전문가와 개발자가 함께 서비스 분리 전략을 수립한다. 해당 서비스 기업의 조직구조와 업무단위를 반드시 반영</li> </ul>
서비스 통합 관리	<p>업무 단위로 분리된 서비스들을 관리하기 위해 서비스 메시(Service Mesh) 를 적용해야 한다. 사이드카 프록시 패턴의 Istio 또는 쿠버네티스 서비스를 적용할 수 있다. 서비스 메시는 아래의 기능들을 담당한다.</p> <ul style="list-style-type: none"> <li>● Configuration: 서비스의 재빌드·재부팅 없이 설정사항을 반영(Kubernetes Configmap)</li> <li>● Service Discovery: 서비스 검색 및 등록(Kubernetes Service, Istio)</li> <li>● Load Balancing: 서비스 간 부하 분산(Kubernetes Service, Istio)</li> <li>● API Gateway: 클라이언트 접근 요청을 일원화(Kubernetes Ingress)</li> <li>● Service Security: 마이크로서비스 보안을 위한 심층 방어메커니즘 적용(Spring Cloud Security)</li> <li>● Logging: 서비스별 로그의 중앙집중화(Loki, ELK Stack)</li> <li>● Monitoring: 서비스별 메트릭 정보의 중앙집중화(Prometheus)</li> <li>● Distributed Tracing: 마이크로서비스 간의 호출 추적(Zipkin)</li> </ul>
데이터 트랜잭션	<p>데이터 일관성을 위해 아래와 같은 분산 아키텍처에 적합한 트랜잭션 관리를 적용한다.</p> <ul style="list-style-type: none"> <li>● CDC(Change Data Capture): 트랜잭션 변경사항을 로그에 남기고 실시간 비동기로 타겟 DB에 반영한다. 장애가 발생해도 복구 시점까지 반복 수행</li> <li>● SAGA 패턴: 트랜잭션이 종료될 때 완료 이벤트를 수신하고 다음 로컬 트랜잭션을 실행. 실패가 발생하면 로그를 남기고 취소상태에서 보상 트랜잭션을 실행한다. Long-Lived 트랜잭션에 적합</li> </ul>

## 5. Cloud Native 아키텍처 설계

### □ 컨테이너 개발/운영 환경 설계

- 마이크로서비스 애플리케이션을 서비스하기 위해 컨테이너 기술을 사용하여 격리하고 독립적으로 관리
- 멀티 컨테이너 관리를 위해 컨테이너 오케스트레이션(Orchestration) 도구인 쿠버네티스를 활용
- 멀티 클러스터 환경을 관리하는 랜처(Rancher) 와 쿠버네티스 상용 솔루션인 레드햇 오픈시프트(OpenShift)를 적용 가능



## 5. Cloud Native 아키텍처 설계

### - 쿠버네티스 기반 인프라 요구사항

구분	내용
정의	<ul style="list-style-type: none"><li>● 요구사항 및 작업 범위, 비용 산정을 통해 도출 된 적절한 서버 자원을 설계에 적용</li><li>● 엔터프라이즈 운영 환경은 HA 구성 필수</li></ul>
개발/테스트	<ul style="list-style-type: none"><li>● 서버<ul style="list-style-type: none"><li>○ OS: CentOS (HKP), Ubuntu, RHEL 등</li><li>○ CPU/Memory: 2Core 8Gb 이상</li></ul></li><li>● Node<ul style="list-style-type: none"><li>○ 최소 master node 1ea, worker node 1ea 이상</li><li>○ 권장 master node 2ea, worker node 2ea, edge node 2ea 이상</li></ul></li></ul>
운영	<ul style="list-style-type: none"><li>● 서버<ul style="list-style-type: none"><li>○ OS: CentOS, Ubuntu, RHEL 등</li><li>○ CPU/Memory: 4Core 16Gb 이상</li></ul></li><li>● Node<ul style="list-style-type: none"><li>○ 최소 master node 3ea, worker node 2ea, edge node 2ea 이상</li><li>○ 권장 master node 3ea, worker node 3ea, edge node 2ea 이상</li></ul></li></ul>



## 5. Cloud Native 아키텍처 설계

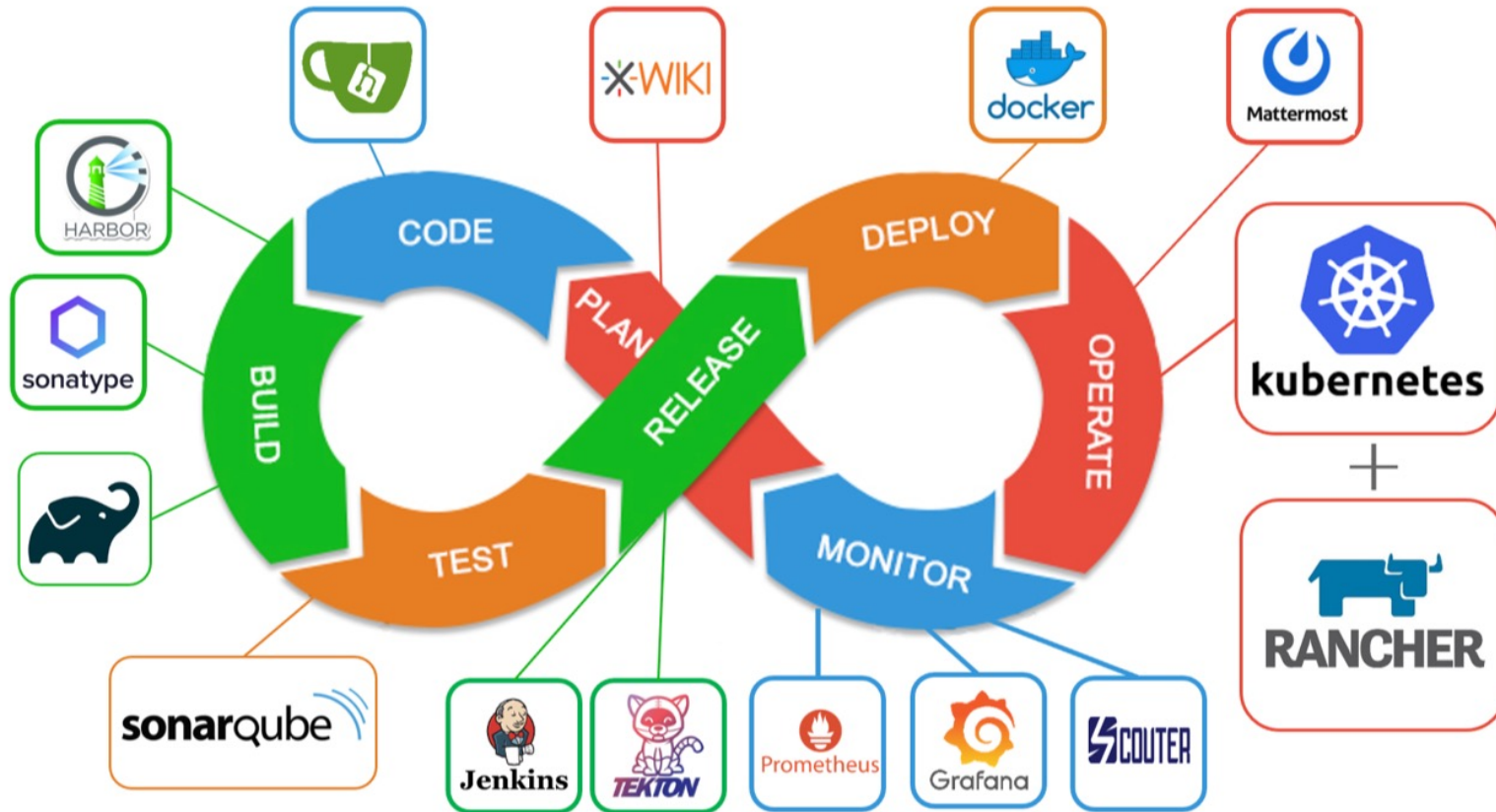
### - 쿠버네티스 설치 소프트웨어

구분	내용
정의	<ul style="list-style-type: none"><li>● 쿠버네티스 환경은 OSS를 활용한 Kubernetes 기준으로 구축</li><li>● 외부 클라우드 환경은 각 인프라/플랫폼 서비스에 종속</li></ul>
Bare Metal (Private/Public)	<p>[OSS를 활용한 Kubernetes]</p> <ul style="list-style-type: none"><li>● 컨테이너: Docker</li><li>● 오케스트레이션: Kubernetes<ul style="list-style-type: none"><li>○ 코어: kube-proxy, kube-controller-manager, kube-apiserver, kube-scheduler, etcd, core-dns, pause</li><li>○ 컨테이너 네트워크 인터페이스 (CNI): Calico</li><li>○ 게이트웨이: Ingress-NGinx Controller, MetalLB</li></ul></li><li>● 멀티 클러스터: Rancher<ul style="list-style-type: none"><li>○ 고가용성: Kube-VIP</li><li>○ 코어: K3S</li><li>○ 다운스트림: RKE (Rancher Kubernetes Engine)</li></ul></li></ul>
관리형 K8S	<ul style="list-style-type: none"><li>● EKS (Elastic Kubernetes Service)</li><li>● AKS (Azure Kubernetes Service)</li><li>● GKE (Google Kubernetes Engine)</li></ul>

## 5. Cloud Native 아키텍처 설계

### □ DevOps 체계 설계

- DevOps환경은 OSS를 활용한 Kubernetes 기준으로 구축
- 요구 사항에 따라 다른 OSS 와 상용 소프트웨어로 일부 전환 가능
- Ingress 를 활용하여 Service 도메인 적용하고, 고가용성(HA) 을 보장해야 함



## 5. Cloud Native 아키텍처 설계

- OSS를 활용한 Kubernetes DevOps Tool Chain 상세

구분	내용
XWiki	온라인 기반 Document 작성 도구
Mattermost	협업 팀 커뮤니케이션 도구
Gitea	Git 기반 소스 관리 도구
Nexus	의존성 관리 Repository
Harbor	컨테이너 이미지 Registry
Jenkins	빌드 배포 통합 관리 도구
Tekton	CI/CD Pipeline 구축 도구
SonarQube	코드품질 관리 도구
Prometheus	Metric 수집
Grafana	Metric 시각화
Scouter	애플리케이션 성능 모니터링 도구

## 5. Cloud Native 아키텍처 설계

### ❑ CI/CD 자동화 설계

- CI/CD환경은 OSS 기준으로 구축
- 요구사항에 따라 다른 OSS 와 상용 소프트웨어로 일부 전환 가능

### ❑ CI/CD 설계 이념

- 지속적인 통합과 배포
  - 개발 시 공유되는 소스코드의 지속적 통합 및 충돌방지 서비스 제공
  - 자동화 테스트를 통해 빠른 배포를 제공
- 끊임 없는 개발 및 운영 환경 제공
  - 통합 빌드 및 배포 자동화를 구축
- OSS CI/CD 상세

구분	내용
Mattermost	협업 팀 커뮤니케이션 도구
Gitea	Git 기반 소스 관리 도구
Nexus	의존성 관리 Repository
Harbor	컨테이너 이미지 Registry
Jenkins	빌드 배포 통합 관리 도구
Tekton	CI/CD Pipeline 구축 도구
SonarQube	코드품질 관리 도구

## 5. Cloud Native 아키텍처 설계

### □ 기술 및 기법

- Private Cloud 환경에서의 아키텍처
  - 아키텍처 환경

구분	내용	비고
인프라	Bare Metal VM, IaaS	Ex) Private Cloud
애플리케이션	MSA (마이크로서비스 아키텍처)	Ex) Spring Cloud, Kubernetes Service
DevOps	Private에서 설치와 운영이 가능한 DevOps Tool Chain	Ex) OSS DevOps Tool Chain
CI/CD	Private에서 설치와 운영이 가능한 CI/CD Pipeline	Ex) OSS CI/CD Pipeline

## 5. Cloud Native 아키텍처 설계

- Public Cloud 환경에서의 아키텍처
  - 아키텍처 환경

구분	내용	비고
인프라	EKS (Elastic Kubernetes Service) AKS (Azure Kubernetes Service) GKE (Google Kubernetes Engine)	Amazon Web Service Microsoft Azure Google Cloud Platform
애플리케이션	MSA (마이크로서비스 아키텍처)	Ex) Spring Cloud, Kubernetes Service, Istio
DevOps	AWS DevOps	제공서비스: CloudWatch, X-Ray 등
	Azure DevOps	제공서비스: Boards, Monitor 등
	GCP DevOps	제공서비스: Cloud Monitoring, Cloud Logging 등
CI/CD	AWS CI/CD솔루션	제공서비스: CodeCommit, TaskCat, CodeBuild, CodeDeploy, CodePipeline 등
	Azure CI/CD Pipeline 서비스	제공서비스: Repos, Pipelines, Test Plans, Artifacts 등
	GCP CI/CD 서비스	제공서비스: Cloud Build, Artifact Registry, Cloud Source Repositories 등

## 5. Cloud Native 아키텍처 설계

- 3대 Cloud 벤더의 Cloud Native 서비스

구분	AWS	Azure	GCP
Server	Elastic Compute Cloud (EC2)	Virtual Machines	Compute Engine
Serverless	Lambda	Functions	Cloud Functions
Container	EKS, ECS	AKS	GKE
PaaS	Elastic Beanstalk	App Service Service Fabric Cloud Services	App Engine
API	API Gateway	API Management	Cloud Endpoints

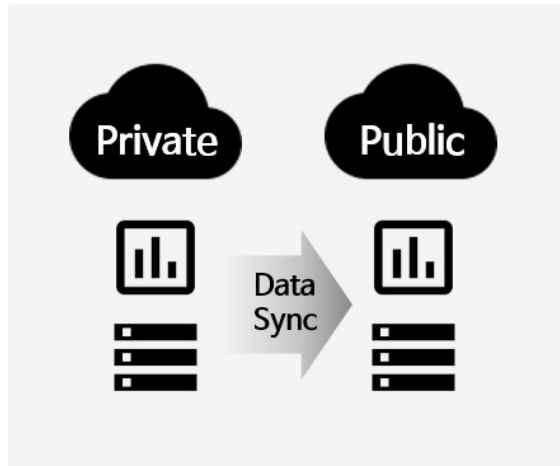
- Hybrid-Multi Cloud 환경에서의 아키텍처

- 아키텍처 환경

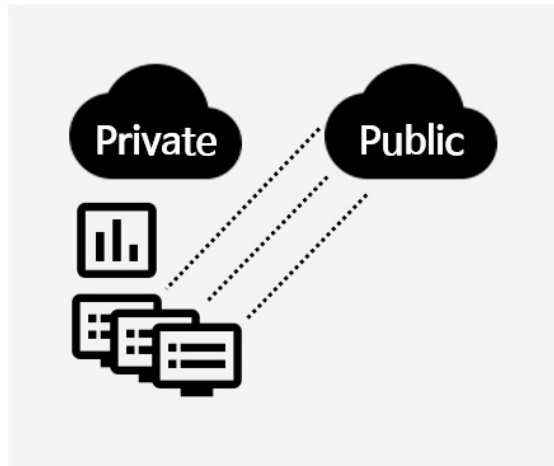
구분	내용	비고
인프라	Private Cloud + Public Cloud	Ex) Private Cloud + AWS
애플리케이션	MSA (마이크로서비스 아키텍처)	Ex) Spring Cloud, Kubernetes Service, Istio
DevOps	Private + Public 환경에서 설치와 운영이 가능한 PaaS 또는 Cloud Native DevOps Tool Chain	Ex) OSS DevOps Tool Chain, OpenShift PaaS
CI/CD	컨테이너 기반의 CI/CD Pipeline	Ex) OSS CI/CD Pipeline

## 5. Cloud Native 아키텍처 설계

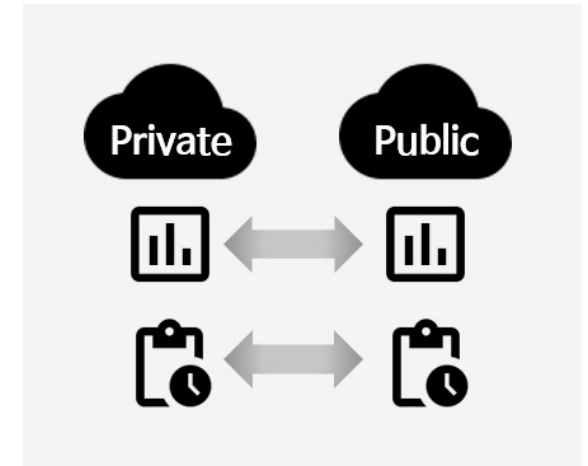
### - 활용 가능 시스템 구성



재난 복구



클라우드 버스팅



백업 및 아카이빙



## 6. 개발환경 구축

### □ 기술 및 기법

- 개발 환경 설계 시 고려사항
  - 권한 및 인증이 아키텍처 요건에 충족하는지 확인
  - Private 환경에서 소프트웨어 동작 이상유무 점검
- 아키텍처 플랫폼 템플릿

구분	설명	활용
Helm	Kubernetes 패키지 관리 도구	여러 환경에 배포하는 경우 Chart 로 패키징하여 배포 및 업데이트 수행
Ansible	Cloud 환경 설정 관리 도구	Code 형태로 구성을 관리하여 누락되지 않게 동일한 환경을 구축
Kubespray	Ansible 을 활용한 Kubernetes 자동 환경 구성 도구	고가용성 Kubernetes 클러스터 구축 자동화
Terraform	코드를 통한 인프라 구성 도구	스케줄러로 리소스 요청하여 인프라 구축
Script	직접 작성하고 실행하는 커스텀 스크립트	모듈화 하기 어려운 설정을 스크립트로 작성하여 수동 반영

## 6. 개발환경 구축

### □ 정의

- 애플리케이션 개발과 PoC 수행을 위한 기본 환경을 구축한다.

### □ 수행 상세

- 개발 환경 설계
  - 인프라 자원은 아키텍처 요건에서 PoC와 Pilot 작업이 가능한 정도로 설계
    1. 개발에 필요한 적정 VM 자원의 개수와 스펙을 비용 측면을 고려하여 산정
    2. 하이브리드-멀티 클라우드의 경우 목적에 따라 구성 규모와 여부를 결정
  - 소프트웨어 구성은 아키텍처 요건에 맞게 최대한 반영하여 설계
    1. 운영 환경까지 고려하여 DevOps Tool Chain 을 모두 반영
    2. 외부 시스템과 연동 시 이를 고려한 환경 설계

### □ 개발 환경 구축

- VM, 웹 애플리케이션 구성
  - Private Cloud: VM 생성 후 [Helm|Kubespray|Ansible|Terraform|Script] 템플릿을 통해 구성
  - Public Cloud: IaaS, PaaS, FaaS 지원 및 Private (e.g. HKP) 환경 설치 포함
- DevOps Tool Chain 및 CI/CD Automation 구축
  - OSS DevOps Tool Chain 환경 구성
  - 환경에 따라 Public 벤더 DevOps 서비스로 구성

## 7. PoC

### □ 정의

- 개발 환경에서 Cloud Native 아키텍처 설계에 대한 개념 검증(Proof of Concept) 수행을 진행
- 앞서 도출된 기술과 업무 범위에 대해 확인하는 과정이 포함되어야 함

### □ 수행 상세

- Cloud Native 아키텍처 설계에 따른 과제별 검증
  - Private Cloud: OSS Kubernetes 설치 후 DevOps Tool Chain 및 CI/CD 전체 사이클 테스트 진행
  - Public Cloud: 개발 환경 Provisioning 이후 벤더 서비스와 기타 서비스 간의 호환성 점검
  - Hybrid-Multi Cloud: 활용 목적에 따른 버스팅/복구/백업 점검
  - MSA: 일부 서비스 구현으로 DDD 모델링과 서비스메시 검증
- 보완점 및 해결방안 도출
  - 과제별 검증을 통해 리스크 확인
  - 리스크에 대한 해결방안 도출 및 즉시 적용
  - 대규모 수정이 필요할 경우, PoC 결과를 참고하여 아키텍처 설계와 미리 도출 된 기술/작업 범위를 보완
  - 변경점이 있다면 해당 부분 적용하여 자원규모와 비용을 재산정

## 7. PoC

---

### □ 기술 및 기법

- PoC 진행 시 고려사항
  - 구축 여부와 별개로 품질이나 대상범위에 이상이 없는지 확인
  - Private 및 특정 환경에서 수행 시 내부 보안을 준수했는지 점검
  - DevOps 팀 운영 및 개발 환경 확인

## 8. 테스트/운영 환경 구축

### □ 정의

- 애플리케이션 서비스 운영 환경을 구축한다. 설계된 Cloud Native 아키텍처의 요건을 모두 수용하여 구축
- 요구사항에 따라 또는 필요시, 서비스에 대한 검증이 충분히 가능한 테스트 환경을 별도로 구축

### □ 수행 상세

- 운영 환경 설계
  - 인프라 자원은 PoC 결과 반영된 최종 아키텍처 요건에 맞게 설계
    1. PoC 결과를 적용한 실제 서비스 운영에 필요한 적정 VM 자원의 개수 산정
    2. 하이브리드-멀티 클라우드의 경우 목적에 따라 구성 규모와 여부를 결정
  - 소프트웨어 구성은 PoC 결과 반영된 최종 아키텍처 요건에 맞게 설계
    1. DevOps Tool Chain 과 CI/CD Pipeline 구성
    2. 외부 시스템과 연동 시 이를 고려한 환경 설계
- 운영 환경 구축
  - VM, 웹 애플리케이션 구성
    1. Private Cloud: VM 생성 후 [Helm|Kubespray|Ansible|Terraform|Script] 템플릿을 통해 구성
    2. Public Cloud: IaaS, PaaS, FaaS 지원 및 Private 환경 설치 포함
  - DevOps Tool Chain 및 CI/CD Automation 구축
    1. OSS DevOps Tool Chain 환경 구성
    2. 환경에 따라 Public 벤더 DevOps 서비스로 구성

## 8. 테스트/운영 환경 구축

### □ 기술 및 기법

- 운영 환경 설계 시 고려사항
  - 타 시스템 연동 시 운영 환경에 따른 보안/인증 등 추가작업 여부 확인
  - OSS 또는 사용 솔루션에 대한 운영 서비스 관련 조항 재확인
- 운영 환경 구축 시나리오

작업	내용	비고
VM 및 네트워크 구성	인프라 구성 작업 서비스 및 도구로 설치 및 구성	n분 내 설치
소프트웨어 구성	패키징 설치 도구 또는 서비스로 즉시 설치	n분 내 설치
표준 환경 구성	특정 환경에 따른 보안 패치 또는 스크립트 수행	
환경 테스트	설치 및 설정 환경 전체 사이클에 대한 테스트 수행	
결과서 작성	작업 절차 및 보완점, 트러블슈팅 작성	

## 9. 개발 시나리오 수행

### □ 정의

- Cloud 환경에서 이를 최대한으로 활용한 Cloud Native 애플리케이션을 개발
- DevOps 팀이 Tool Chain 과 Agile 방법론을 활용하여 마이크로서비스를 구축하는 과정을 수행

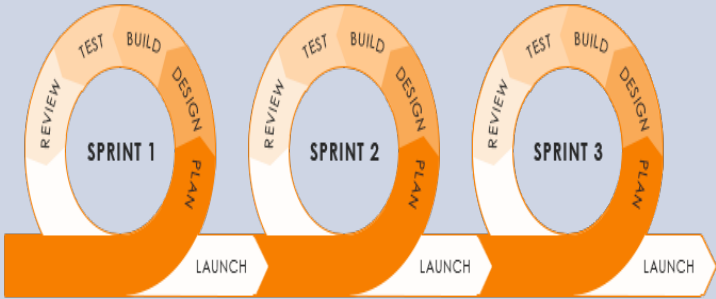
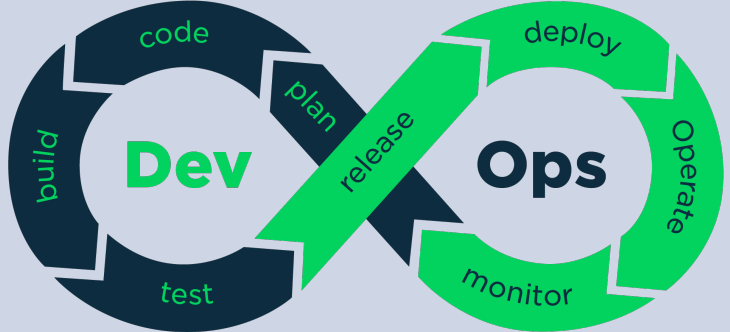
### □ 수행 상세

- DevOps 팀 운영
  - 독립적인 팀의 마이크로서비스 애플리케이션 개발
  - 각 마이크로서비스 애플리케이션 팀의 CI/CD Automation 수행
  - WIKI, 업무관리도구, 실시간 커뮤니케이션 도구 등을 활용
  - 관제 시스템을 통해 오류 트래킹 및 수정
  - 인프라 변경 시 프로비저닝 수행
- Agile 방법론 수행
  - Scrum, Kanban, XP 등의 애자일 개발 프레임워크 선택
  - WIKI, 업무관리도구, 채팅을 통한 커뮤니케이션과 문서 관리 활용
  - 스토리, 스프린트, 역할 설정 및 실행 반복 수행
  - 변화 대응 및 고객과의 협업이 계획보다 우선

## 9. 개발 시나리오 수행

### □ 기술 및 기법

- Agile 과 DevOps 적용 시 고려사항
  - Agile 과 DevOps 를 이분법적으로 나누려고 노력하지 않아도 되며, 함께 유기적으로 융합하여 사용하는 것이 효율적
  - Agile 은 고객가치, 자율성, 변화 적응과 같이 문화와 방향에 초점이 맞춰지고, DevOps는 보다 도구를 적극적으로 활용하여 개발/운영에 대한 프로세스를 변화시켜 즉각적인 결과 확인 가능. 때문에 프로젝트가 진행될수록 Agile 보다 DevOps 의 절차적인 부분만 강조되기 쉬운 데 DevOps의 내면에는 반드시 Agile 문화가 바탕에 있어야하며 구성원 모두의 생각과 행동이 변화에 빠르게 대응하고MVP(Minimum Viable Product) 개발을 목표로 IT Agility를 달성해야 함
- Agile 과 DevOps 특징

특징	Agile	DevOps
정의 / 목표	<ul style="list-style-type: none"> <li>고객의 가치와 변화 대응 최우선</li> <li>프로젝트 문화와 환경에 대한 관리/운영에 집중</li> <li>빠른 실패확인과 릴리즈를 반복 수행하는 MVP 모델을 지향</li> </ul> <p><b>AGILE METHODOLOGY</b></p> 	<ul style="list-style-type: none"> <li>개발자와 운영자의 커뮤니케이션과 조화(Seamless)가 주목적으로 개발/운영 환경에 대한 통합, 긴밀한 협력에 집중</li> <li>프로젝트의 모든 지식/노하우가 운영에 장착되도록 프로젝트 팀 전체가 책임을 가지고 실행</li> </ul> 



## 9. 개발 시나리오 수행

### - Agile 과 DevOps 특징 (계속)

특징	Agile	DevOps
프로세스	<ul style="list-style-type: none"> <li>소규모 단위 개발, 빠른 릴리즈</li> <li>점진적 완성을 목표로 하는 프로덕트 개발 일정으로 설계-개발-테스트-배포-피드백 단계를 반복 수행</li> </ul>	<ul style="list-style-type: none"> <li>지속적 코드 통합, 배포 자동화</li> <li>설계-개발-빌드-테스트-배포-운영-모니터링 단계를 흐름으로 끊임없이 수행</li> </ul>
도구	JIRA, Confluence, Redmine, Trello, Wiki	JIRA, Confluence, Redmine, Wiki, Slack, Mattermost, Gitlab, Jenkins, Kubernetes 등
이행	프레임워크: Scrum, Kanban, XP 등	모델: CALMS, 3Ways, 5Plays 등

### - Agile - Scrum

구분	내용
정의 / 특징	<ul style="list-style-type: none"> <li>1~4주 정도의 짧은 주기로 작은 목표를 세워 점진적으로 제품을 개발</li> <li>5~9명으로 구성되는 소규모 팀이 업무 주기(스프린트)를 반복하며 과제를 완수</li> <li>매일 15분 정도의 Scrum 미팅</li> <li>계획, 프로세스 보다는 열린 마음으로 협력과 변화 적응에 집중</li> </ul>

## 9. 개발 시나리오 수행

### - Agile - Scrum(계속)

구분	내용
5가지 핵심 가치	 <p>Scrum의 5가지 가치</p> <p>(출처: <a href="https://www.scrum.org/resources/what-is-scrum">https://www.scrum.org/resources/what-is-scrum</a>)</p> <ol style="list-style-type: none"> <li>① 용기: 옳은 방향으로 일을 처리하고 어려운 문제를 해결하려는 용기</li> <li>② 집중: 모든 노력과 기술을 스프린트와 팀 목표 달성을 위해 집중</li> <li>③ 약속: 팀과 개인의 목표달성을 위해 헌신을 약속</li> <li>④ 존중: 개개인이 능력을 갖춘 독립적인 존재임을 인정하고 존중</li> <li>⑤ 개방: 업무의 모든 내용을 공개하고 자신에게 불리해도 숨기지 않는 투명성</li> </ol>
구성원 역할	<ul style="list-style-type: none"> <li>● 제품책임자(Product Owner): 비즈니스 목표를 달성하기 위해 백로그와 제품을 관리</li> <li>● 스크럼마스터(Scrum Master): PO와 Scrum팀이 가치와 원칙으로 제품을 만들고, 변화에 적응하며 신속하게 작업을 수행하는지 관리. 매일 Scrum 미팅을 진행하고 상황을 모니터링</li> <li>● 스크럼 팀(Scrum Team): 최선의 노력으로 변화에 민감하게 반응하며 제품을 개발하여 고객만족을 실현</li> </ul>

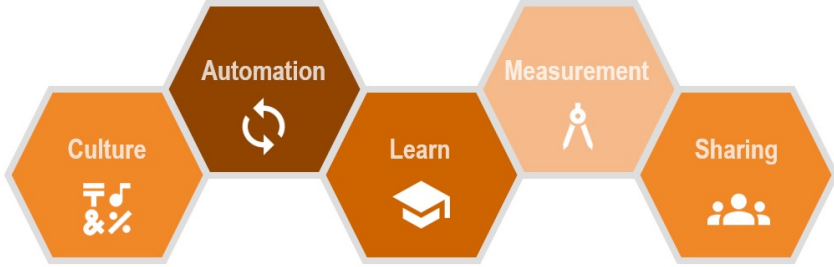
## 9. 개발 시나리오 수행

### - Agile - Scrum(계속)

구분	내용
용어	<ul style="list-style-type: none"> <li>● 사용자 스토리 (User Story): 사용자가 사용하는 관점에서 어떤 가치를 제공할 것인가를 정의</li> <li>● 스프린트 (Sprint): 계획-개발-리뷰-피드백 등의 업무주기에 대한 1사이클로 보통 1~4주의 기간</li> <li>● 제품 백로그 (Product Backlog): 사용자 스토리의 집합</li> <li>● 스프린트 백로그 (Spring Backlog): 스프린트 목표에 대한 작업 목록</li> <li>● 칸반 (Kanban) 작업 목록을 상태와 일정에 따른 흐름을 보여주는 게시판</li> <li>● 스프린트 리뷰 (Sprint Review): 스프린트 종료일에 Scrum 팀 개발자가 제품을 시연</li> <li>● 스프린트 회고 (Sprint Retrospective): 스프린트 종료일에 지난 일정을 돌아보고 보완점을 도출하여 제품 품질을 향상하는 과정</li> <li>● 잠재적 출시 가능 제품 (Potentially Shippable Product Increment), 최소 실행가능 제품 (Minimum Viable Product): 스프린트의 결과물로써, 목표한 제품을 만들어 배포할 수 있는 최소한의 제품</li> </ul>
실행	<ol style="list-style-type: none"> <li>① 제품 백로그 작성: PO가 사용자 스토리를 기반으로 제품 백로그를 작성한다</li> <li>② 스프린트 계획 회의: 모든 구성원이 참여하여 제품 백로그를 기반으로 스프린트 목표와 백로그를 계획한다</li> <li>③ 스프린트 백로그 작성: 스프린트 목표를 달성 가능하도록 Scrum 팀이 나누어 담당하고 칸반보드를 활용하여 작성한다</li> <li>④ 데일리 Scrum 미팅: 짧은 시간안에 끝내야 하며 주로 이슈와 진척도를 체크한다</li> <li>⑤ 제품 개발: MVP를 목표로 개발과 배포를 수행한다</li> <li>⑥ 스프린트 리뷰: Scrum 팀이 만든 MVP를 고객과 이해관계자들에게 시연하고 피드백을 받는다</li> <li>⑦ 스프린트 회고: Scrum 마스터의 관리하에 개선점을 도출하여 다음 스프린트 반영하도록 한다</li> <li>⑧ 다음 스프린트: PO가 제품 출시를 최종 허가할 때까지 다음 스프린트를 진행한다</li> </ol>

## 9. 개발 시나리오 수행

### - DevOps - CALMS

구분	내용
정의/ 특징	<ul style="list-style-type: none"> <li>문화, 자동화, 효율, 측정, 공유 5원칙을 기반으로 각 영역별 실행 수준을 진단하고 성숙도 레벨을 판단</li> <li>DevOps 내재화 평가를 위한 모델</li> </ul>
5가지 핵심 원칙	<p style="text-align: center;"><b>The CALMS Framework for Devops</b></p>  <p style="text-align: center;">CALMS 요소</p> <ol style="list-style-type: none"> <li>① 문화(Culture): 변경사항을 공유하여 협업 및 의사소통을 유도하고 공동책임 문화를 기반으로 한다</li> <li>② 자동화(Automation): 개발 프로세스에서 반복/수동 작업을 최소화하고 자동화를 적용한다</li> <li>③ 효율(Lean): 제품 생산에 낭비되는 모든 요소를 관리하고 배제시켜 더욱 빠른 생산 주기를 제공한다</li> <li>④ 측정(Measurement): 모든 요소와 항목을 측정하고 그 데이터를 활용하여 개발/운영 과정을 개선한다</li> <li>⑤ 공유(Sharing): 모든 성공과 실패의 경험을 투명하게 공유한다</li> </ol>

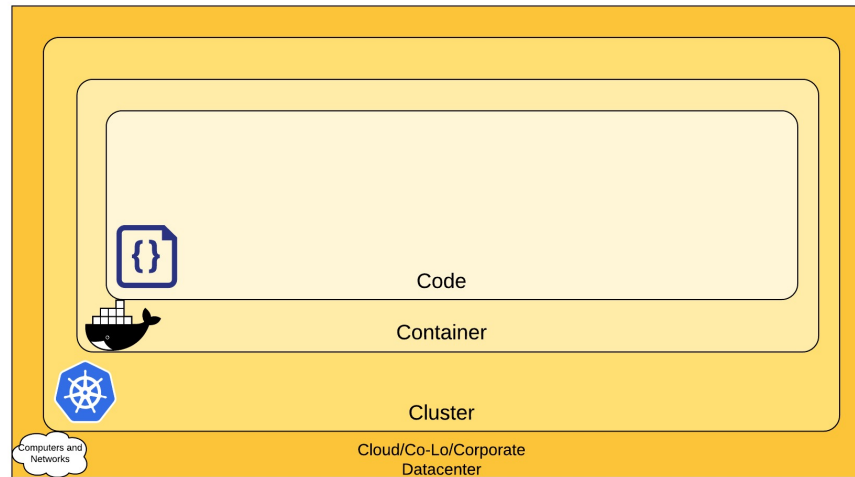
## 10. 보안/성능/단위/통합 테스트

### □ 정의

- Cloud Native 애플리케이션을 구축하는 주요 요소인 DevOps 와 CI/CD Automation 을 활용하면 테스트의 과정은 개발과 분리되지 않음
- 각 마이크로서비스 애플리케이션은 개발-테스트-빌드-배포를 pipeline 으로 관리하여 품질을 빠른 주기로 검증
- 모든 서비스에 대한 통합 테스트만 별도로 수행하며 이 주기도 너무 길지 않게 해야함

### □ 수행 상세

- 보안 테스트
  - Cloud Native 보안은 4C(Cloud, Cluster, Container, Code) 라는 4계층으로 생각할 수 있음



## 10. 보안/성능/단위/통합 테스트

### - 클라우드

- 클라우드 위에 쿠버네티스 클러스터를 구성하는 신뢰 컴퓨팅 기반(trusted computing base)은 Private/Public 클라우드 공급자가 해당 환경에서 워크로드를 안전하게 실행하기 위한 보안 권장 사항을 제시한다. 이를 기반으로 테스트 케이스를 작성하여 테스트
- 쿠버네티스 클러스터 환경에 대한 인프라 보안사항을 체크

제어영역	내용
API 서버 네트워크 접근	<ul style="list-style-type: none"> <li>쿠버네티스 컨트롤 플레인에 대한 모든 접근은 인터넷에서 공개적으로 미허용</li> <li>클러스터 관리에 필요한 IP 주소 집합으로 제한된 네트워크 접근 제어 목록에 의해 제어</li> </ul>
노드 네트워크 접근	<ul style="list-style-type: none"> <li>지정된 포트의 컨트롤 플레인에서 만 (네트워크 접근 제어 목록을 통한) 연결을 허용하고 NodePort와 Load Balancer 유형의 쿠버네티스 서비스에 대한 연결을 허용하도록 노드를 구성</li> <li>노드가 공용 인터넷에 비노출</li> </ul>
클라우드 공급자API 쿠버네티스 접근	<ul style="list-style-type: none"> <li>각 클라우드 공급자는 쿠버네티스 컨트롤 플레인 및 노드에 서로 다른 권한 집합을 부여</li> <li>관리하는 리소스에 대해 최소 권한의 원칙을 따르는 클라우드 공급자의 접근 권한을 클러스터에 구성</li> </ul>
Etcd 접근	<ul style="list-style-type: none"> <li>etcd(쿠버네티스의 데이터 저장소)에 대한 접근은 컨트롤 플레인으로만 제한</li> </ul>
Etcd 암호화	<ul style="list-style-type: none"> <li>가능한 한 모든 드라이브를 암호화</li> <li>etcd는 전체 클러스터(시크릿 포함)의 상태를 유지하고 있기에 특히 디스크는 암호화 필수</li> </ul>

## 10. 보안/성능/단위/통합 테스트

- 클러스터
  - 설정 가능한 컴포넌트 보안 체크
  - 클러스터 내 애플리케이션 보안 체크
    1. 쿠버네티스 API 접근 관련
    2. 인증
    3. 애플리케이션 시크릿 관리
    4. 파드 보안 관련
    5. 서비스 품질
    6. 네트워크 정책
    7. 인그레스 TLS
- 컨테이너
  - 컨테이너 영역에서의 보안 사항을 설정하고 체크

구분	내용
취약점 스캔	● 컨테이너 알려진 취약점 검사
이미지 서명 및 시행	● 컨테이너의 내용에 대한 신뢰 시스템을 유지하기 위한 컨테이너 이미지 서명 체크

## 10. 보안/성능/단위/통합 테스트

### - 코드

- 가장 많은 제어를 할 수 있는 공격 영역
- 코드 보안상 고려할 영역 체크 및 테스트

구분	내용
TLS 접근	<ul style="list-style-type: none"><li>● 기본적으로 전송 중인 모든 것에 대한 암호화 검증</li><li>● 서비스 간 네트워크 트래픽을 암호화할 수 있다면 인증서를 가진 서비스의 양방향 검증을 mTLS를 통해 수행</li></ul>
통신포트 범위	<ul style="list-style-type: none"><li>● 통신이나 메트릭 수집 이외의 서비스 포트가 노출되었는지 확인</li></ul>
타사 종속성	<ul style="list-style-type: none"><li>● 타 라이브러리 정기적 스캔으로 알려진 취약점 확인</li><li>● 각 언어에선 CI 자동수행 도구 지원</li></ul>
정적 코드 분석	<ul style="list-style-type: none"><li>● 대부분 언어에서 시큐어 코딩에 대한 검사도구와 가이드를 제공</li></ul>
동적 탐지 공격	<ul style="list-style-type: none"><li>● SQL인젝션, CSRF, XSS 탐지 및 테스트</li></ul>



## 10. 보안/성능/단위/통합 테스트

### □ 성능 테스트

- 목적
  - 클러스터 구성 후 SLA, SLI, SLO 정의
  - 시스템 디자인과 시스템 시나리오에 대한 일치 점검
  - 고가용성, 확장성, 안정성 에 대한 근거 제공
- 성능 점검 요소 및 방법

구분	내용
DNS	<ul style="list-style-type: none"><li>● 애플리케이션 응답시간에 매우 중요한 요소</li><li>● 리소스 제한/미제한 시 성능 측정</li><li>● CPU, Memory, I/O 에 대한 SLI, SLO 설정 및 QPS테스트 실행</li></ul>
API-Server	<ul style="list-style-type: none"><li>● Kube API-Server는 오케스트레이션에 대한 작업의 시작</li><li>● Control, Worker를 구분해서 테스트</li><li>● 테스트 시나리오 및 SLI, SLO 설정</li><li>● 성능 테스트를 위한 Access 보안을 임의로 해제 가능</li></ul>
기타 리소스	<ul style="list-style-type: none"><li>● 애플리케이션 및 특정 네트워크에 대한 성능점검을 위한 테스트 시행</li><li>● CPU, Memory, I/O 등 기본적인 성능 감시를 중점적으로 시행</li><li>● Public 클라우드의 경우 개별적으로 지원하는 모니터링 시스템을 활용 (ex. AWS CloudWatch)</li><li>● 파드, kubelet, container 서비스 종료 점검</li><li>● 노드 taint, 파드 오토스케일 점검</li></ul>

## 10. 보안/성능/단위/통합 테스트

### □ 단위 테스트

- 모듈 테스트
  - 특정 단위의 함수 및 모듈이 정상 동작하는지 확인
  - 테스트 자동화를 통해 주기적으로 자주 실행
  - 함수의 내부구조를 볼 수 있는 화이트박스 테스트 실행
- 서비스 테스트
  - 각 마이크로 서비스를 단위로 테스트 실행
  - 개별 서비스 테스트를 통해 전체 서비스의 문제를 보다 빠르게 해결하기 위한 시나리오를 설정하고 API 대상으로 테스트 수행

### □ 통합 테스트

- 통합 서비스 테스트
  - 서비스 테스트 이후 전체 마이크로 서비스를 통합하는 과정에서 발생하는 오류를 체크하는 테스트
  - 서비스 간의 연계, 데이터 공유 등 상호작용 위주의 테스트 실행
  - 너무 많은 서비스 간의 종합적인 테스트는 오류를 발견하더라도 원인 분석과 조치가 어려우니 스프린트 이후 자동화된 테스트 수행으로 점진적인 문제해결이 기본
- 사용자 테스트 (End to End, UI 테스트)
  - 통합 서비스 테스트 이후 사용자 요구사항까지 체크하는 테스트
  - 실제 사용자 환경을 재현한 시나리오 기반의 테스트가 기본
  - 유저 화면을 통해 시스템을 직접 조작하는 것이 가장 신뢰성을 보장

## 10. 보안/성능/단위/통합 테스트

### □ 기술 및 기법

#### – SLI (Service Level Indicator: 서비스 수준 척도)

- 서비스를 판단할 수 있는 기준을 정량적으로 측정한 값
- 추적 가능한 모든 지표를 SLI 로 다룰 필요는 없으며 몇가지 기준으로 나누어 분류

구분	내용
사용자 대면 시스템	요청에 대한 올바른 리턴, 응답 속도 그리고 요청에 대한 시간 당 처리량이 가장 중요
저장소 시스템	Read/Write 속도와 처리시간, 접근성과 보안이 중요
빅데이터 시스템	대용량 데이터 처리량 그리고 유입부터 완료까지 걸리는 시간이 중요

#### – SLO (Service Level Objectives: 서비스 수준 목표)

- SLI 에 의해 측정된 서비스 수준의 목표 값
- $SLO == (SLI \leq \text{최소값} \leq SLI \leq \text{최대값})$
- 단순, 비 완벽, 적은 수, 점진적 목표값 을 기준으로 수준 설정
- Ex) SLI: API응답시간, SLO: 일일 API 응답시간은 99.9% 이상에서 100ms 이하

#### – 화이트박스 테스트

- 모듈의 원시 코드를 오픈한 상태에서 코드의 모든 경로를 테스트하는 방법
- 설계에 초점을 둔 구조적 테스트이며, 초기 레벨에서 수행
- 기초 경로 검사, 제어 구조 검사 등의 방법이 존재

# 11. 모니터링 및 분석

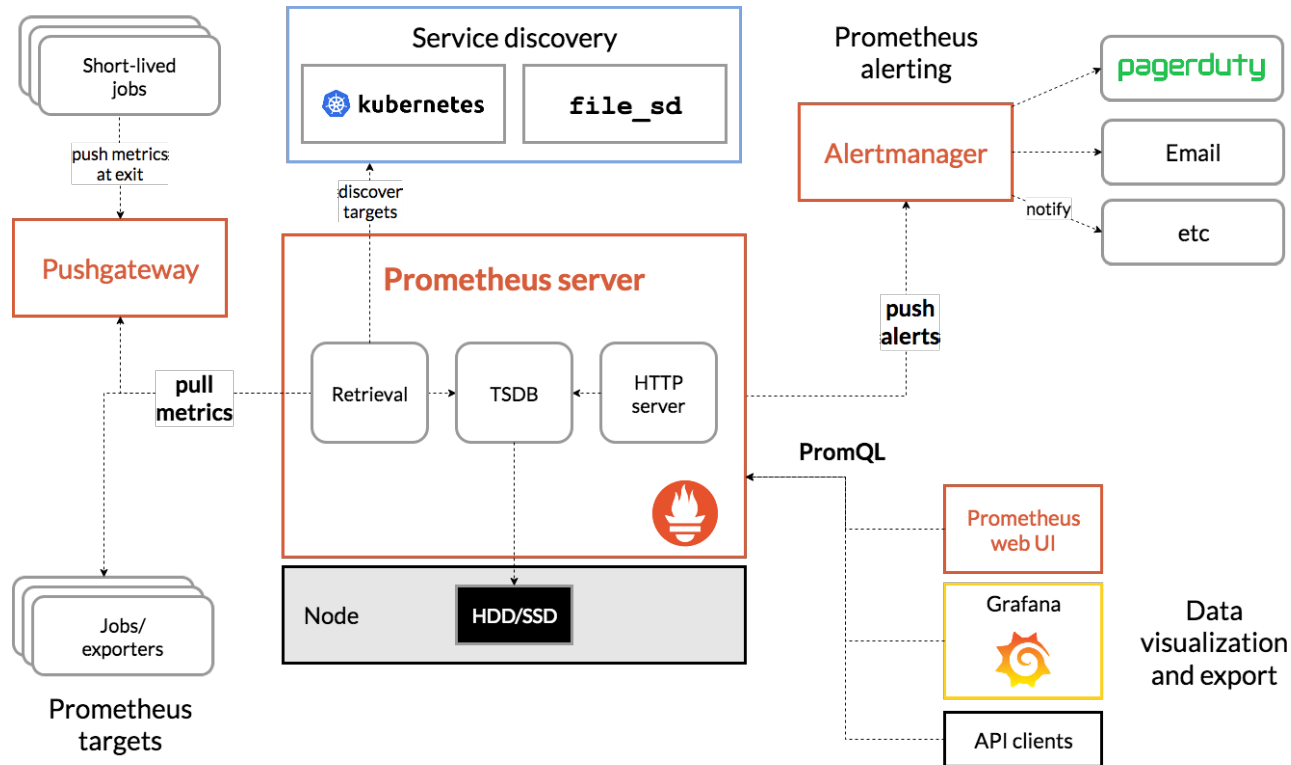
## □ 정의

- Cloud Native 환경에서의 인프라와 애플리케이션의 상태를 모니터
- Cloud Native 환경에선 자원(애플리케이션 혹은 컨테이너)이 매우 동적이며 라이프 사이클이 짧으므로 그 특성을 고려하여 모니터링을 진행
- USE패턴은 인프라 컴포넌트에 초점을 맞추고, RED패턴은 애플리케이션의 최종 UX를 모니터링하는 데 중점
- Cloud Native 모니터링 도구를 사용하여, Cloud 환경의 가용성/성능/보안을 사전에 모니터링할 수 있고, 문제가 최종 사용자 환경에 영향을 주기 전에 찾아서 해결 가능

# 11. 모니터링 및 분석

## □ 수행 상세

- Cloud Native 모니터링에 특화되어 있는 프로메테우스를 사용하여 모니터링
- 프로메테우스는 서비스 검색(Service Discovery)을 통해 데이터 수집 대상을 발견
- 수집된 데이터가 저장되면 PromQL을 이용해 대시보드에서 활용되거나 알림매니저를 통해 알림이 발송되고 알림은 다시 무선 호출, 이메일 같은 통보로 변경



## 11. 모니터링 및 분석

- 클라이언트 라이브러리 (Client Library)
  - 애플리케이션 내부에 제어하려는 코드에 인라인(Inline)으로 원하는 계측 기능을 추가 가능. 직접 계측(Direct Instrumentation)
  - 스레드 안정성, 부기(BookKeeping), HTTP 요청에 대한 응답으로 프로메테우스 텍스트 게시(text exposition) 형식 생성 같은 모든 핵심 세부사항을 처리
- 익스포터 (Exporter)
  - 가져오고 싶은 메트릭을 애플리케이션 바로 옆에 배치하는 소프트웨어
  - 프로메테우스로부터 요청을 받아서, 애플리케이션에서 요청된 데이터를 수집하고, 데이터를 올바른 형식으로 변환한 다음, 마지막으로 프로메테우스에 대한 응답으로 데이터를 반환
  - 애플리케이션의 메트릭 인터페이스를 프로메테우스의 게시 형식(Exposition Format)으로 변환하는 일대일 방식의 작은 proxy
- 서비스 검색 (Service Discovery)
  - 동적 환경에서 애플리케이션과 익스포터 목록은 금방 유효기간이 만료되기 때문에 목록을 한 번만 제공해서는 안 된다. 이 때문에 서비스 검색이 필요
  - 모든 조직은 각기 다르게 움직이기 때문에, 레이블 재지정(Relabeling)을 사용해 모니터링 대상과 그들의 레이블을 매핑하는 메타데이터 구성 방식을 서비스 검색에 허용

# 11. 모니터링 및 분석

- 데이터 수집
  - 프로메테우스는 수집scrape이라 불리는 HTTP 요청을 보내 메트릭을 가져오는 작업을 수행
  - 수집에 대한 응답은 구문 분석 후 저장 장치에 저장
  - 풀 기반(Pull-based)으로 데이터를 수집
- 저장 장치
  - 프로메테우스는 사용자 정의 데이터베이스에 로컬로 데이터를 저장
  - 분산 시스템은 신뢰성을 확보하기가 다소 어렵기 때문에, 프로메테우스는 클러스터링 미구성
- 대시보드
  - 수식 브라우저(Expression Browser)를 제공
  - 그라파나(Grafana)의 사용을 권장
- 기록 규칙 및 알림
  - 수천 대의 머신에서 들어오는 메트릭을 매번 집계한다면 지연이 발생 가능
  - 기록 규칙(Recording Rules)을 통해 PromQL 표현식을 정기적으로 수행하고, 그 결과를 저장 엔진에 저장 가능
- 알림 규칙은 PromQL 표현식을 정기적으로 수행하고, 그 결과를 알림. 알림은 알림매니저(Alertmanager)로 전송

## 11. 모니터링 및 분석

### - 알림 관리

- 알림매니저(Alertmanager)는 프로메테우스 서버에서 알림을 받아 통보로 변환
- 통보(Notification)는 이메일, 메신저 등의 서비스로 전송
- 연관된 알림은 하나의 통보로 집계될 수 있으며, 페이저 스톰(Pager Storm)을 줄이기 위해 조절
- 여러 팀에 서로 다른 라팅 설정과 통보 결과를 구성할 수 있으며, 알림 해제 가능
- 알림과 알림의 임계치는 알림매니저가 아닌 프로메테우스에서 구성

### - 장기 저장소

- 프로메테우스는 여러 머신에 데이터를 저장하는 클러스터링 저장 솔루션을 미제공
- 하지만 다른 시스템이 이러한 역할을 가져가 수행할 수 있게 원격 읽기 및 쓰기 API를 제공



# 11. 모니터링 및 분석

## □ 기술 및 기법

### - 모니터링 기술

- 블랙박스 모니터링
  1. 애플리케이션 외부에 초점
  2. 일반적으로 CPU, 메모리, 스토리지 등의 시스템 컴포넌트를 모니터링
  3. 인프라 수준의 모니터링에 유용
- 화이트박스 모니터링
  1. 애플리케이션 상태에 초점
  2. 전체 HTTP 요청, 500 오류 수, 요청 레이턴시 등을 모니터링
  3. 애플리케이션 수준의 모니터링에 유용

# 11. 모니터링 및 분석

## □ 모니터링 패턴

- USE 패턴
  - USE 패턴은 "모든 리소스에 대해 사용률, 포화도, 오류율을 확인"
  - USE 패턴을 사용하면 시스템의 리소스 제약과 오류율을 빠르게 파악 가능
  - 애플리케이션 수준의 모니터링엔 한계가 있어 인프라 모니터링에만 활용
  - U(Utilization): 사용률
  - S(Saturation): 포화도
  - E(Error): 오류율

## □ RED 패턴

- 사상은 구글의 네 가지 황금 신호(four golden signals)에서 영감
  - 레이턴시(Latency): 요청을 처리하는 데 걸리는 시간
  - 트래픽(Traffic): 시스템에 존재하는 요청의 양
  - 오류(Errors): 요청 실패율
  - 포화(Saturation): 서비스의 사용률

### ❑ Cloud Native Landscape

- <https://github.com/cncf/landscape>

### ❑ Cloud Native Technical Document

- <https://www.cncf.io/services-for-projects/#technical-documentation>

### ❑ Kubernetes Architecture

- <https://kubernetes.io/ko/docs/home>

### ❑ Cloud Native Architecture

- <https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native>

### ❑ Azure DevOps

- <https://azure.microsoft.com/ko-kr/services/devops>

### ❑ AWS DevOps

- <https://aws.amazon.com/ko/devops>

### ❑ GCP DevOps

- <https://cloud.google.com/devops>

### ❑ Scrum

- <https://www.scrum.org>

### ❑ CALMS for DevOps

- <https://www.devopsgroup.com/insights/resources/diagrams/all/calms-model-of-devops>

### ❑ Cloud Native Security

- <https://kubernetes.io/docs/concepts/security/overview>

**감사합니다**