SK 주식회사 C&C

Enterprise에서 Kubernetes기반의 Cloud를 활용한 MSA 적용 방안

양 준 기

Digital Experience를 위한 Application Requirement

- ✓ Zero downtime
- ✓ Shortened feedback cycles
- ✓ Mobile and multidevice support
- ✓ Connect devices - IoT
- ✓ Data-driven

이를 위한 인프라와 아키텍처가 필요함.

소비자의 새로운 기대

새로운 컴퓨팅 컨텍스트

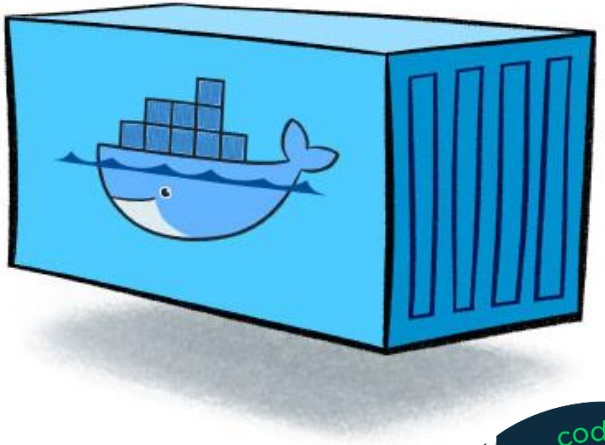
소프트웨어 구성 방식의 변화

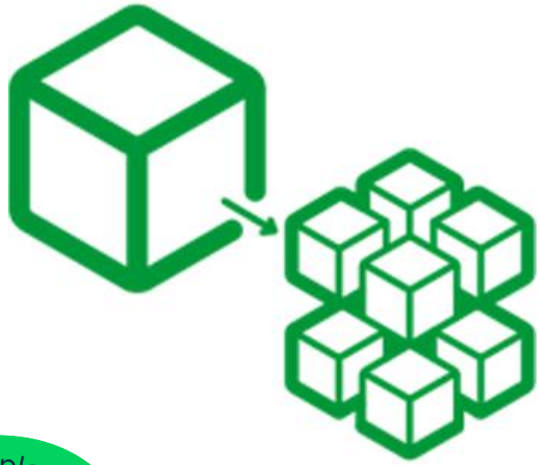
새로운 아키텍처 패턴과 운영 방식 도입

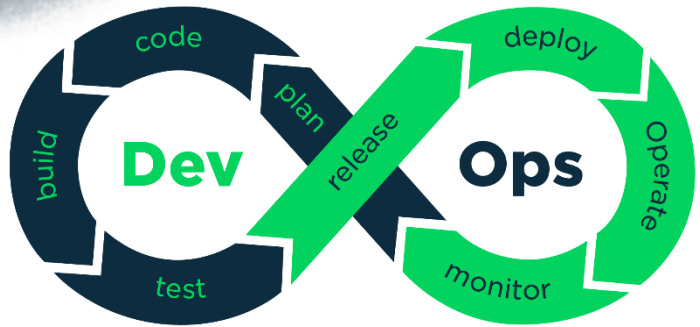
Cloud is about where we're computing.

Cloud-Native is about how.

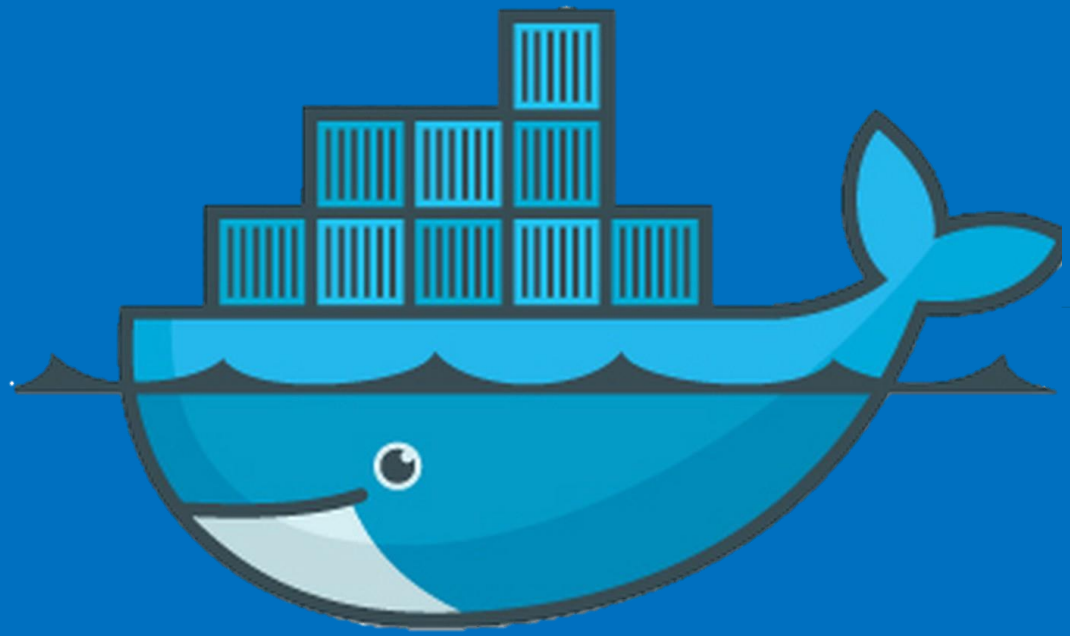
Cloud Native를 위한 기술







새로운 플랫폼의 등장 - Cloud/Containers



Cloud Computing

PaaS, CaaS

Kubernetes

어떤 클라우드를 사용할 것인가?
Cloud Deployment/Service Model - Public, Private, Hybrid

Public

- 초기 투자 및 운영비용 절감
- 신속한 개발 (Agility)

- AWS, Azure, Google, CloudZ

- 운영의 역할 (Service Provider)
- 풍부한 서비스 확보
- 다양한 Built-In Service
  (Provider Road-Map 의존적)

Private(On-Premise)

- 민감한 정보 보호
- 서비스 및 법적 규제

- VMware, OpenStack
  Azure Stack, CloudZ

- IaaS, PaaS 설치와 운영 역할
- **Public 대비 서비스 부족**
- In-house skill 확보와 향상
  ※ On-Premise 중심의
  Customer 직접 관리 및 운영

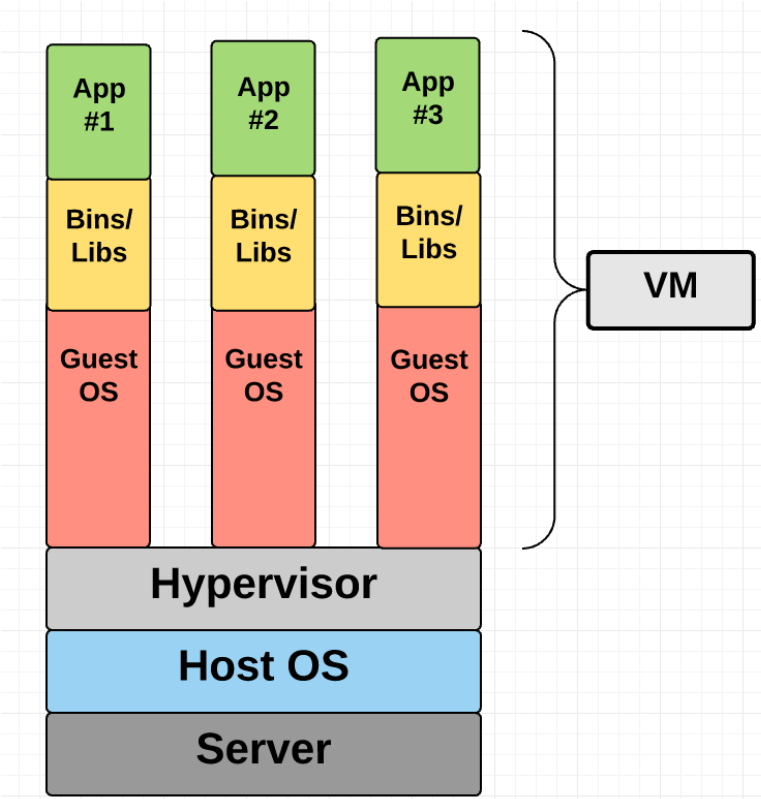
Hybrid

- 정보의 보호 및 공유
- 자원의 유연한 확장성

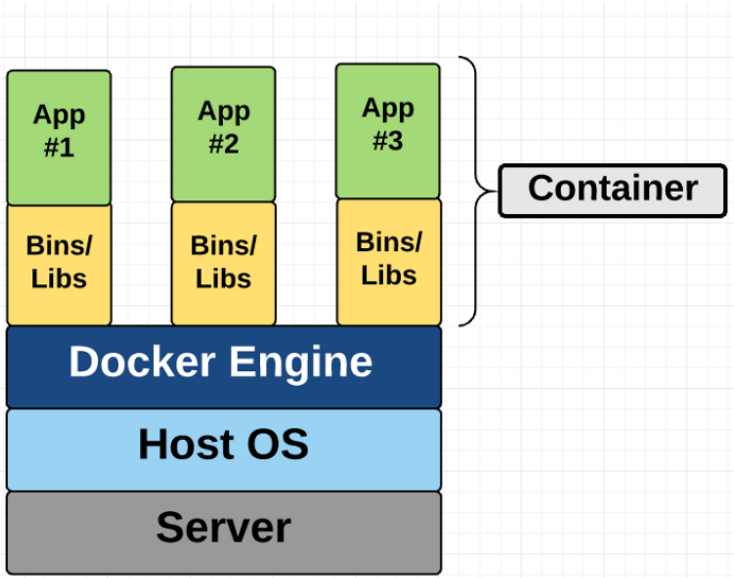
- Private 솔루션으로 활용

- Public 서비스와 Private의 자체 서비스의 상호 활용
- **Private의 구축/개발 등의 In-House Skill 여전히 존재**

Container와 VM



- 시스템 자원을 좀 더 효율적으로 이용
- 소프트웨어 딜리버리 주기를 가속화
- 어플리케이션 이동이 가능
  → MSA에 최적화



모든 것을 Container로 할 것인가?

Containerization
가능 한가?

Containerization
하기에 적합한가?

Solution Dependency

컨테이너 오케스트레이션은 필요한가?

컨테이너 오케스트레이션의 기능

컨테이너 오케스트레이션은 대규모의 동적 환경에서 컨테이너의 라이프사이클과 관련된 모든 것을 관리

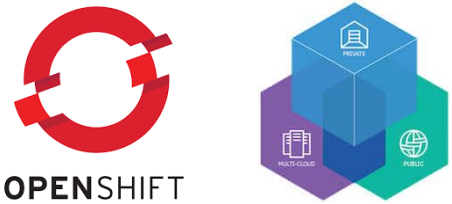
- 컨테이너의 프로비저닝 및 배포
- 컨테이너의 이중화 및 가용성
- 호스트 인프라에 애플리케이션 로드를 균등하게 분산하기 위해 컨테이너 확장 또는 제거
- 호스트에 리소스가 부족하거나 호스트가 손실된 경우 한 호스트에서 다른 호스트로 컨테이너 이동
- 컨테이너 간 리소스 할당
- 외부 시스템에 컨테이너에서 실행되는 서비스 노출
- 컨테이너 간 서비스 디스커버리 로드 밸런싱
- 컨테이너 및 호스트의 상태 모니터링
- 어플리케이션을 실행하는 컨테이너에 대한 어플리케이션 설정

어떤 Container Orchestration Tool을 선택할 것인가?

Container Orchestration Tool

Kubernetes as a Service

Enterprise Kubernetes Platform



Kubernetes 도입 시 선택 기준은 무엇인가?

Public / Private / Hybrid

기술내재화가
되어 있나?

누가 관리할 것인가?

Kubernetes로 구축 시 고려해야 할 것은 무엇인가?

- K8s 는 Container Service 를 제공할 뿐 Enterprise에서 요구하는 가용성 및 안정성 확보 필요
  → 반복적인 성능 및 가용성 테스트를 통한 안정성 확보
- Network 구성, Storage 구성, 모니터링, 로깅, 알람 구성을 위한 다양한 컴포넌트 Integration 필요
  → Know-how 적용, 관련 Tech.전문가 참여
- 운영 방식에 따른 Namespace 를 통한 자원 분리 및 Network 정책도 필요
  → 운영 및 확장을 위한 정책 및 Multi-tenancy 정책 수립
- K8s는 분기마다 업그레이드되고 있고, 필요 기술을 검증하고 반영하는 작업 필요
  → 플랫폼은 small start로 구축하고 지속적 업그레이드 경험이 중요

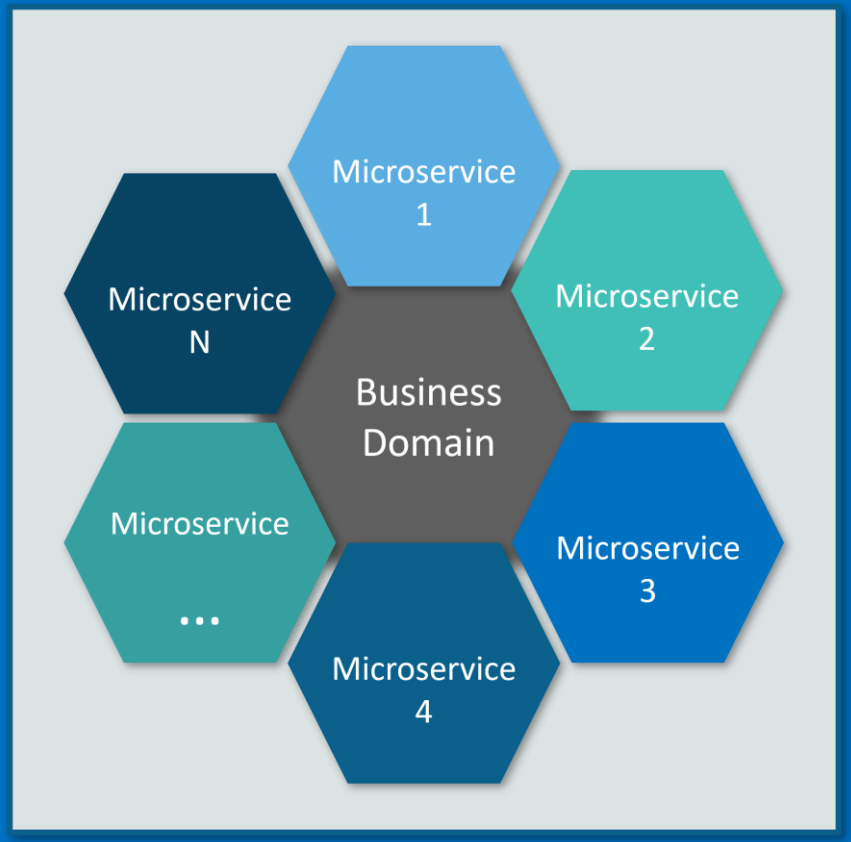
Enterprise Architecting !!

Microservices

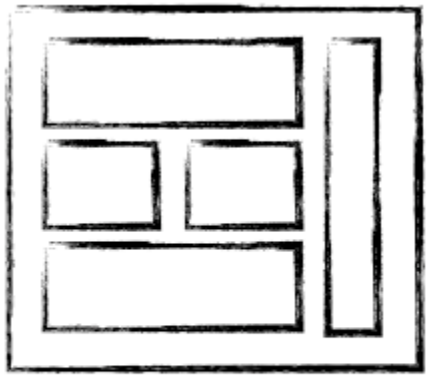
Microservices

Microservices 단위

Microservices 장점



Monolith와 Microservices



MONOLITHIC/LAYERED

MICRO SERVICES

Traditional Company와 Cloud Native Company

Microservices 란 무엇인가?
(by James Lewis and Martin Fowler)

단일 애플리케이션을 작은 서비스 suite으로 개발하는 방법으로, 각각 자체 프로세스에서 실행되고 가벼운 연결방식, 대개는 HTTP기반 API와 통신한다.

비즈니스 기능을 중심으로 구축되며 완전 자동화 된 배포 머신를 통해 독립적으로 배포 할 수 있다.

최소한의 중앙 집중식 관리가 있으며, 다른 프로그래밍 언어로 작성되고 다른 데이터 저장 기술을 사용할 수 있다.

https://martinfowler.com/microservices/#what

Microservices - Characteristics

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

Microservices를 하면 무엇이 좋아지나?

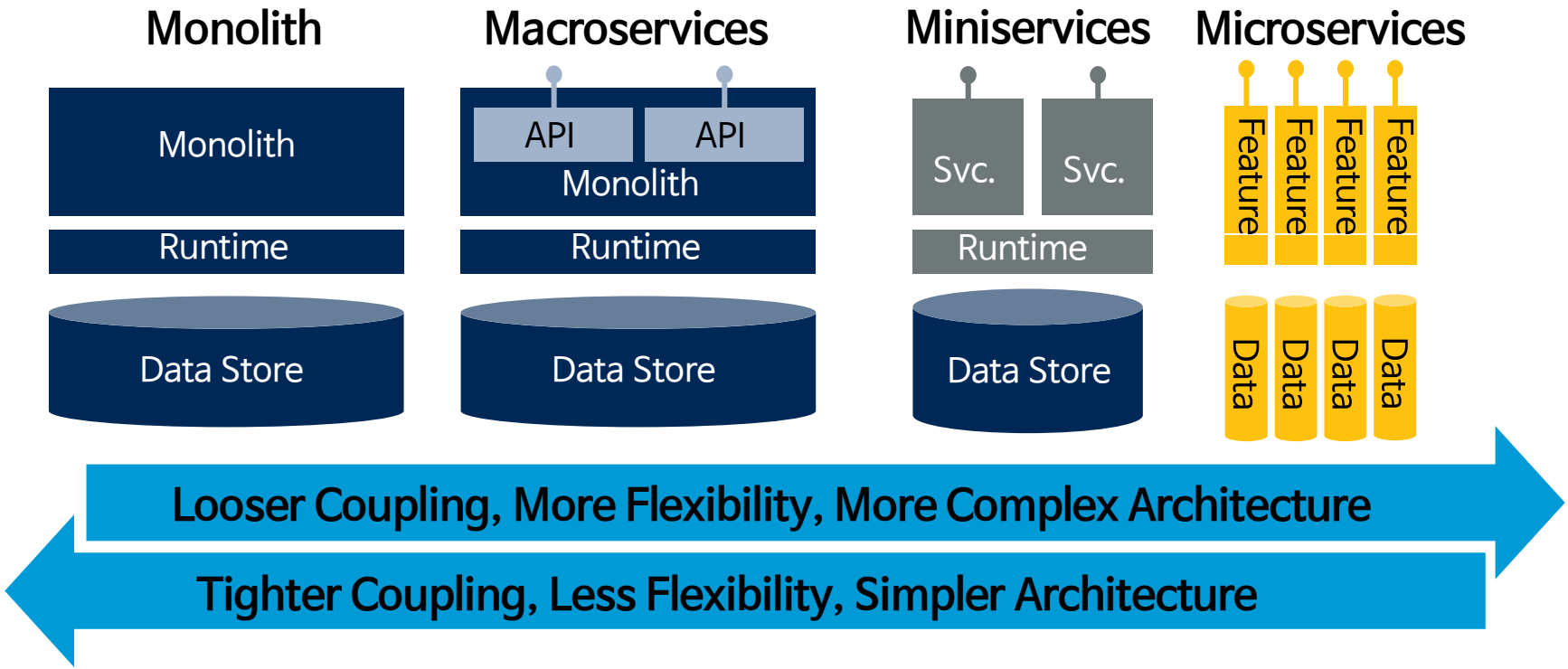
- ☑ **장애 격리와 복구** : 장애 발생된 영역으로 영향도가 제한되고 빠르게 복구됨
- ☑ **비용 효율적 증설** : 개별 서비스에 필요한 수준에 Align된 이중화/삼중화 구현
- ☑ **서비스 개선 속도 증가** : 단위 서비스별 변경 및 배포를 통한 빠른 서비스 출시 가능
- ☑ **생산성 향상** : 코드의 양이 적어지고 쉽게 이해할 수 있으며 쉽게 수정 가능
- ☑ **신기술 도입** : 원하는 최근 기술을 도입하고 시도할 수 있음.
- ☑ **Polyglot** : 새로운 서비스 개발시 언어/플랫폼/데이터베이스를 선택할 수 있음.

어떻게 어떤 단위로 나누어야 하는가?

Microservices가 어느 정도의 Service 크기인가?

Microservices를 어떻게 나누어야 하나?

Monolith와 Microservice의 단계별 Service Architecture



Gartner : Gartner Round Table Session 1 - The Path to Microservices

어떻게 서비스를 나누어야 하는가?

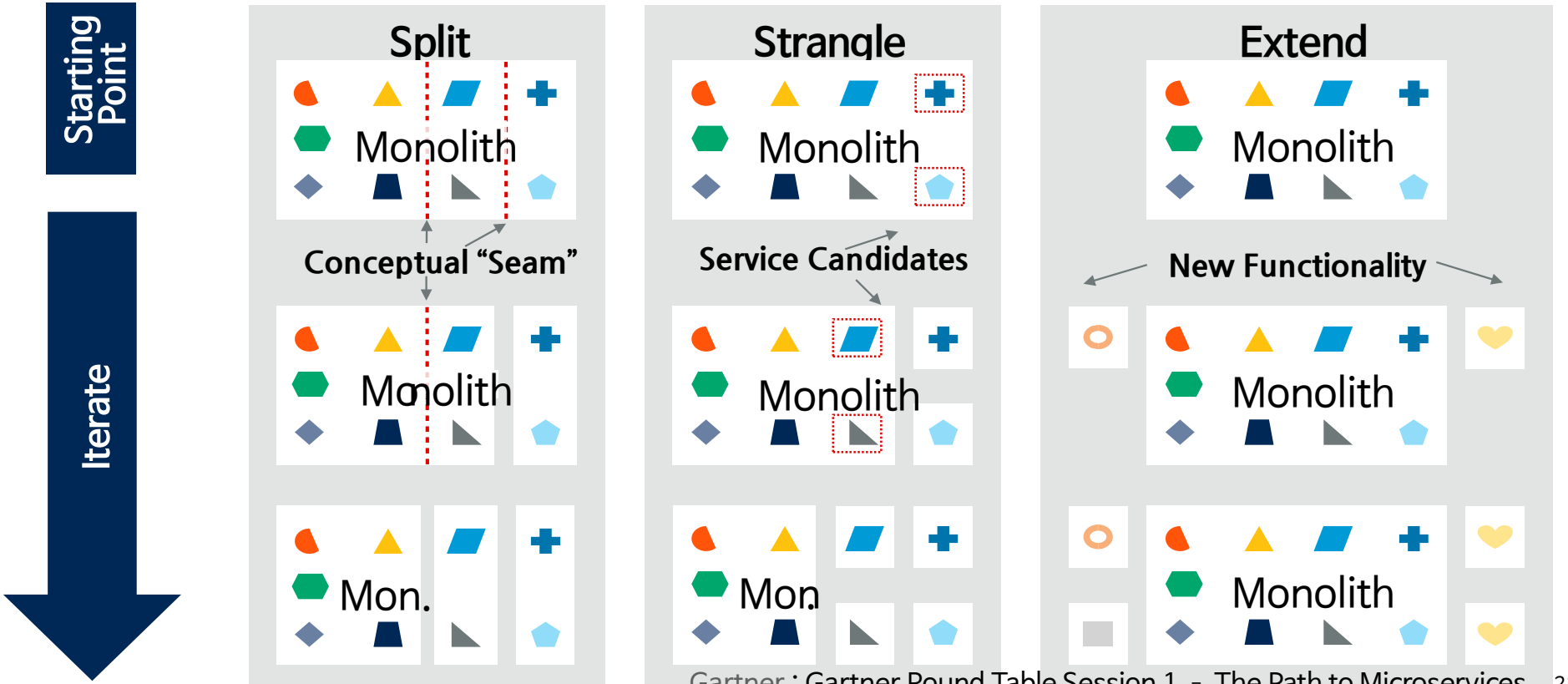
- Zhamak Dehghani : a principal technology consultant at ThoughtWorks

- 간단하고 쉽게 디커플링 할 수 있는 것으로 워밍업
  - 최소한의 준비는 해야 한다. (CI/CD Pipeline, API관리 시스템)
- 모놀리스와의 의존성 최소화
- 고정적인 기능 조기 분할
- 수직 분리 및 데이터 조기 릴리스
- 비즈니스에 중요한것과 변경이 잦은 것 분리
- 코드가 아니라 기능을 분리
- 매크로 먼저, 그 다음 마이크로
- 중단되지 않고 점진적인 단계별 마이그레이션

https://martinfowler.com/articles/break-monolith-into-microservices.html

어떻게 서비스를 나누어야 하는가?

Source : Gartner



Gartner : Gartner Round Table Session 1 - The Path to Microservices

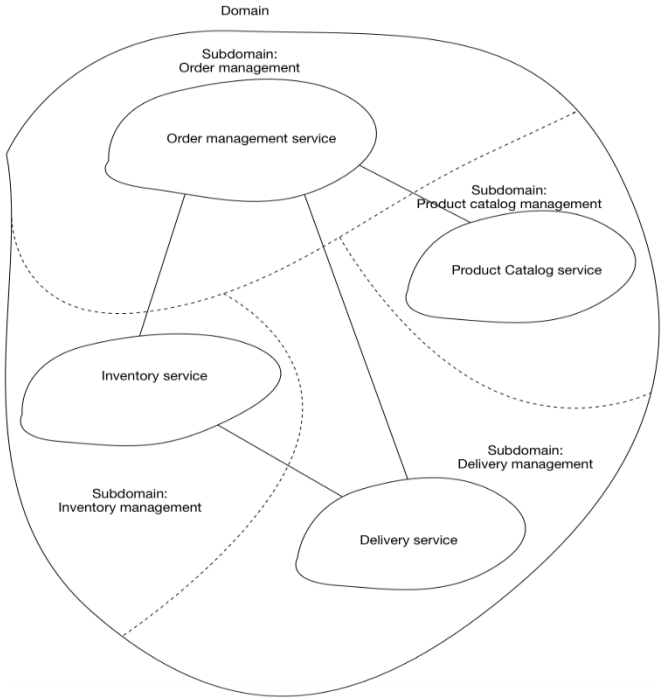
어떻게 서비스를 나누어야 하는가?
Microservices.io (1)

- DDD (Domain-Driven Design)
- 서브 도메인에 해당하는 서비스를 정의.
- DDD는 응용 프로그램의 Problem space (비즈니스)를 도메인으로 나타낸다.
- 도메인은 여러 서브 도메인으로 구성되며, 각 서브 도메인은 비즈니스의 다른 부분에 해당한다.
- 하위 도메인은 다음과 같이 분류 할 수 있다.

Core-비즈니스와 애플리케이션의 가장 중요한 부분을 위한 핵심 차별화 요소

Supporting-비즈니스와 관련이 있지만 차별화 요소는 아니다.
사내에서 구현하거나 아웃소싱 할 수 있다.

Generic-비즈니스에만 국한되지 않으며 상용 소프트웨어를 사용하여 이상적으로 구현된다.



https://microservices.io/patterns/decomposition/decompose-by-subdomain.html

어떻게 서비스를 나누어야 하는가?
Microservices.io (2)

- 비즈니스 기능에 해당하는 서비스를 정의한다.

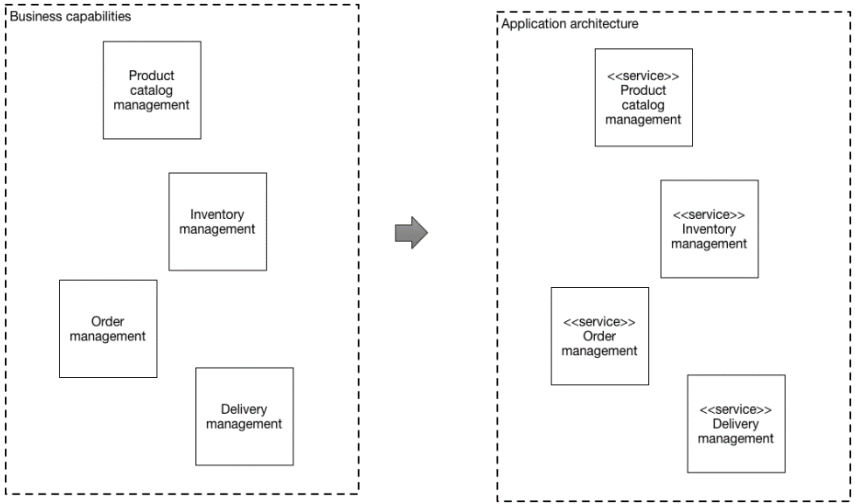
  비즈니스 기능은 비즈니스 아키텍처 모델링의 개념

  비즈니스가 가치를 창출하기 위해 하는 것.
- 비즈니스 기능은 대체로 비즈니스 개체에 해당된다.

  예를 들면,

  - 주문 관리는 주문을 담당한다
  - 고객 관리는 고객을 책임진다
- 비즈니스 기능은 종종 다단계 계층 구조로 구성.

  예를 들어 엔터프라이즈 어플리케이션에는 제품 / 서비스 개발, 제품 / 서비스 제공, 수요 생성 등과 같은 최상위 범주가 있을 수 있다.



https://microservices.io/patterns/decomposition/decompose-by-business-capability.html

어떻게 서비스를 나누어야 하는가?
SK주식회사 C&C Digital Labs for Cloud

Phase1 Microservice 후보 식별

- Task1. 도메인 전문가에 의해 독립적인 업무영역 식별
  (예. 서브시스템, 주제 영역, 트랜잭션, 메뉴 구조 등 참고)
- Task2. 독립 Lifecycle로 신규 개발하는 서비스 식별

Phase2 Biz.특성 반영 (정제)

- Task3. 변경/배포 주기가 빠른 서비스 식별
- Task4. 특정 시기에 높은 사용률로 독립적인 Scaleability가 필요한 서비스 식별
- Task5. 장애로부터 격리하여 운영할 서비스 식별

Phase3 Tech. 특성 반영 (정제)

- Task6. 외부와의 Interaction을 단순화 할 Layer 식별
- Task7. 기술을 다르게 적용해야 할 서비스 분리
  (예. 구현 언어, 플랫폼, DBMS 등)

Phase4 조직 특성 반영

- Task8. 공유/협업이 원활한 조직을 고려한 서비스 식별

Microservices 도입 시 고려해야 할 것들

Performance	• 마이크로서비스는 서비스간의 호출이 통신을 통해 이루어지기 때문에 네트워크 전송 오버헤드가 발생 • 성능에 민감한 서비스인지 확인 필요
Transaction Processing	• 여러개의 API를 하나로 묶는 분산 트랜잭션 시나리오를 없앰 • 분산 트랜잭션이 중요한 시스템은 일부만 모놀리식으로 접근 • 트랜잭션 실패시 예외처리 로직을 어플리케이션에서 처리
Data Duplication	• 마이크로서비스는 서비스별로 적절한 데이터 스토리지 솔루션을 사용하여 구성 될 수 있기 때문에 이에 따른 데이터 중복이 발생할 수 있음 • 각 서비스별 적당한 데이터 모델링을 설계해야 함 • 데이터 중복을 허용할 수 있어야 함
Operation Overhead	• 마이크로서비스는 서비스 개수가 증가함에 따라서 많은 양의 배포 및 릴리즈 작업이 수반됨 • 각 서비스들이 서로 다른 기술을 사용할 수 있기에 필요한 기술도 증가하게 됨 • 배포 및 릴리즈에 수반되는 모든 작업들을 철저하게 자동화 해야 함
Team Management	• 마이크로서비스는 도메인 단위의 서비스가 독립적으로 개발/배포되기 때문에 팀 운영에 있어서도 독립적으로 운영될 수 있어야 함 • 팀 자체적으로 기획/개발/운영하며 스스로 서비스를 발전 시키는 하나의 회사 개념 • 이에 따라 높은 수준의 팀 성숙도가 요구됨

Microservices 도입 시 어려웠던 것들

OSS Eco
빠른 진화 속도에 적응

K8s Eco-System 의 Version-up 이 매우 빠르고, 다양한 OSS 등장

- 프로젝트 기간 內 Multi Cluster 구현을 위한 K8s Feature Porting
- Netflix OSS 에서 Service Mesh 로의 전환 (e.g. Circuit Breaker)
- Knative, Istio, ArgoCD 등 Cloud-Native Pattern 진화

Enterprise Solution
MSA 준비 미흡

Message Queue, DBMS, Enterprise Portal 등 M/W 의 Cloud-Native 준비부족

- PaaS 에서 서비스할 M/W 가 도입될 때마다 방화벽 처리 및 관리정책 정의 필요
- 수 백 개의 부서, 수 만 명의 구성원이 사용하는 Enterprise Portal 환경 고려 필요
- Cloud-Native 전환 없이 Dockerization 후 Container 에만 탑재하여 서비스

MSA 기반
App. 개발전략 부재

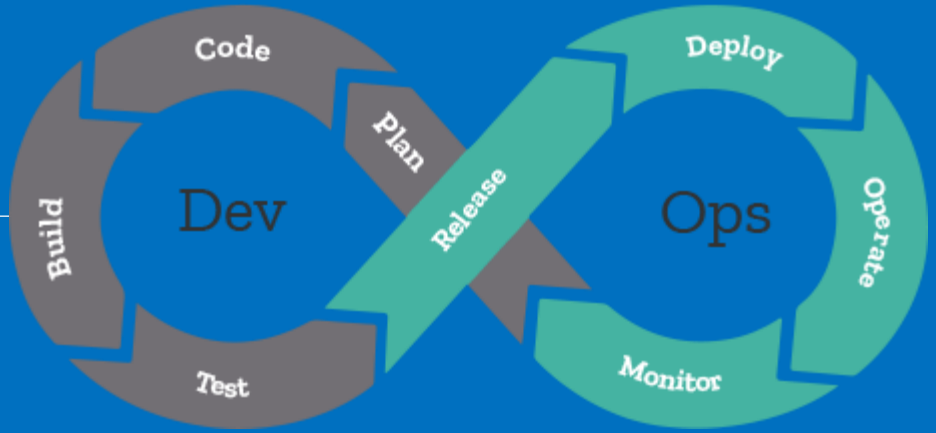
비즈니스 및 IT 현황을 고려한 운영 효율과 지속성이 확보된 개발전략 중요

- 업무의 속성과 관계에 따라 유연한 Data 통합 또는 분산 필요
- 신규개발 및 Legacy Upgrade 를 선제 수행
- Data Schema, Value Object 변경 없이 기존의 개발 방식을 고수

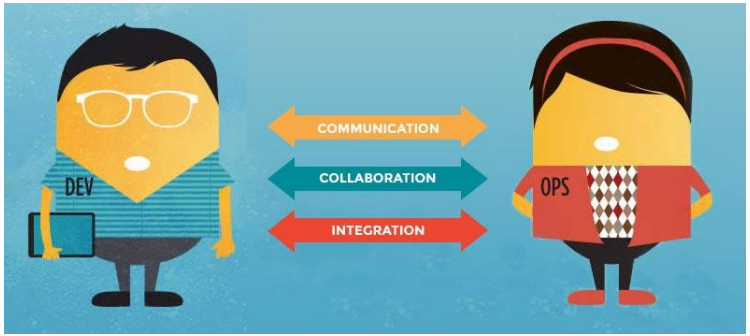
DevOps

DevOps

Continuous Delivery



DevOps는 무엇인가?



Development Operations

소프트웨어의 개발(Development)과 운영(Operations)의 합성어로서,

소프트웨어 개발자와 IT 전문가 간의 **소통, 협업 및 통합을 강조하는 개발 환경이나 문화.**

소프트웨어 개발조직과 운영조직간의 상호 의존적 대응이며

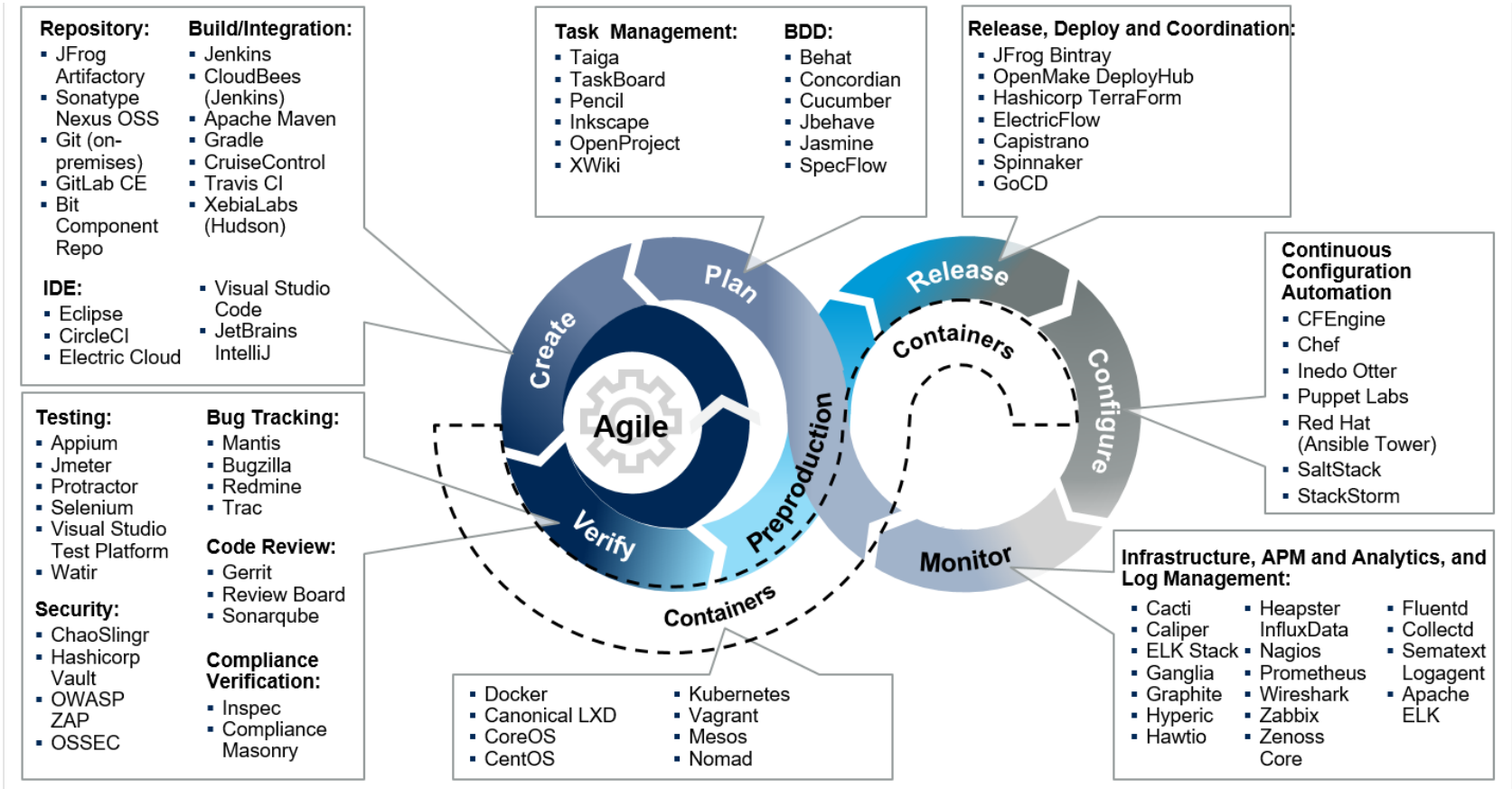
조직이 **소프트웨어 제품과 서비스를 빠른 시간에 개발 및 배포하는 것을 목적**으로 한다.

DevOps는 왜 하려고 하는가?

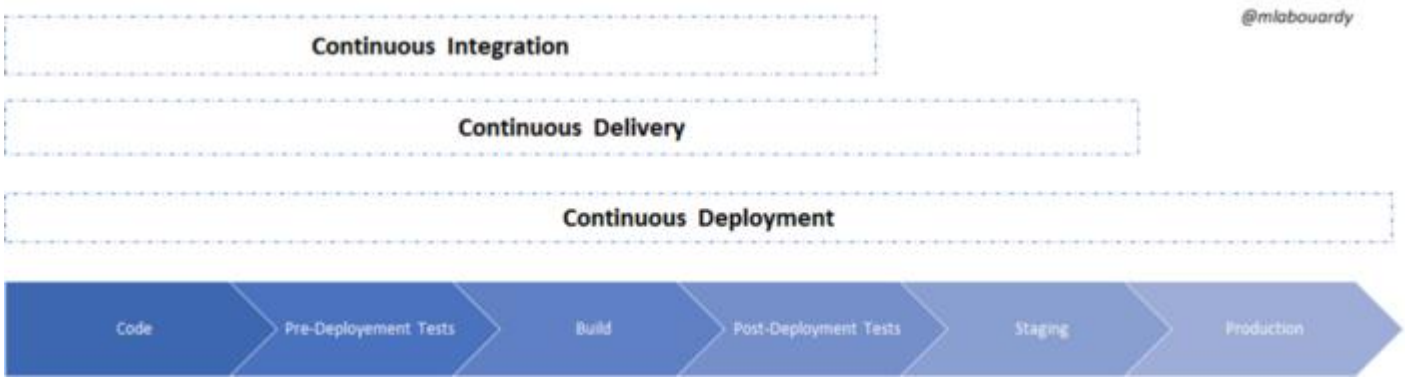
It aims at establishing a **culture**
and **environment** where building, testing,
and releasing software can happen **rapidly**,
frequently, and more **reliably.**

소프트웨어 구축, 테스트, 릴리스가 **신속**하고 자주, 그리고 보다 **안정적**으로 이루어질 수 있는 **문화**와 **환경**을 구축하는 것을 목표로 한다.

DevOps Toolchain - Fully



DevOps Toolchain - Pipeline

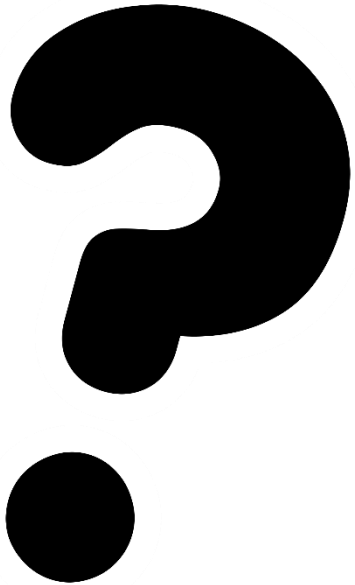


- Continuous Integration : 개발환경에서 코드를 테스트, 빌드, 통합하는 것을 자동화하는 것
- Continuous Delivery : CI pipeline의 확장. 개발자들이 애플리케이션에 적용한 변경 사항이 버그 테스트를 거쳐 리포지토리(예: GitHub 또는 컨테이너 레지스트리)에 자동으로 업로드되는 것
- Continuous Deployment : 개발자의 변경 사항을 리포지토리에서 고객이 사용 가능한 프로덕션 환경까지 자동으로 릴리스하는 것.

DevOps를 어떻게 적용할 것인가?

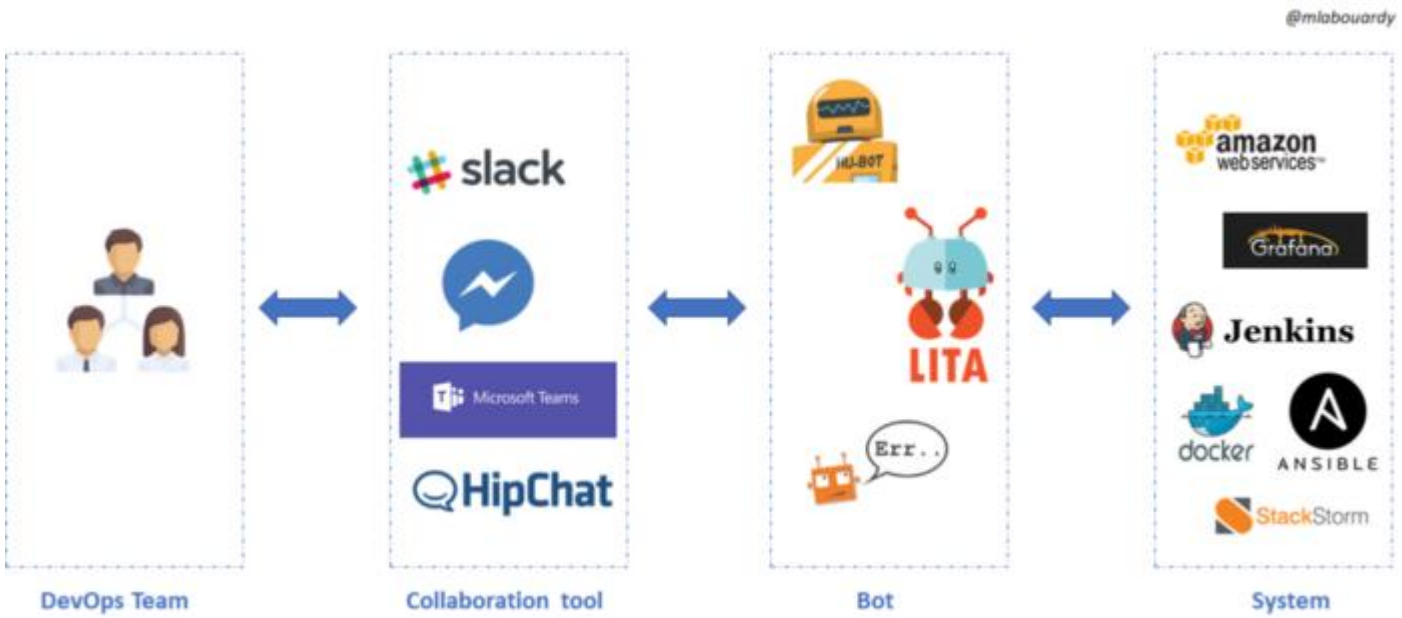
DevOps를 위한 문화는?

DevOps를 위한 환경은?



DevOps Trend - ChatOps

채팅을 통한 빠른 소통 및 해결



https://chatbotsmagazine.com/how-chatops-can-help-you-devops-better-5-minutes-read-507438c156bf

DevOps Trend - VoiceOps

Virtual Assistant for IT Ops Environment

https://thenewstack.io/voiceops-virtual-assistants-ops-environments/

DevOps 도입 시 어려웠던 것들

공유/협업을 위한 효과적인 Tool ?
기존의 Tool은?

SW Quality를 위해
얼마나 노력해야 하나?

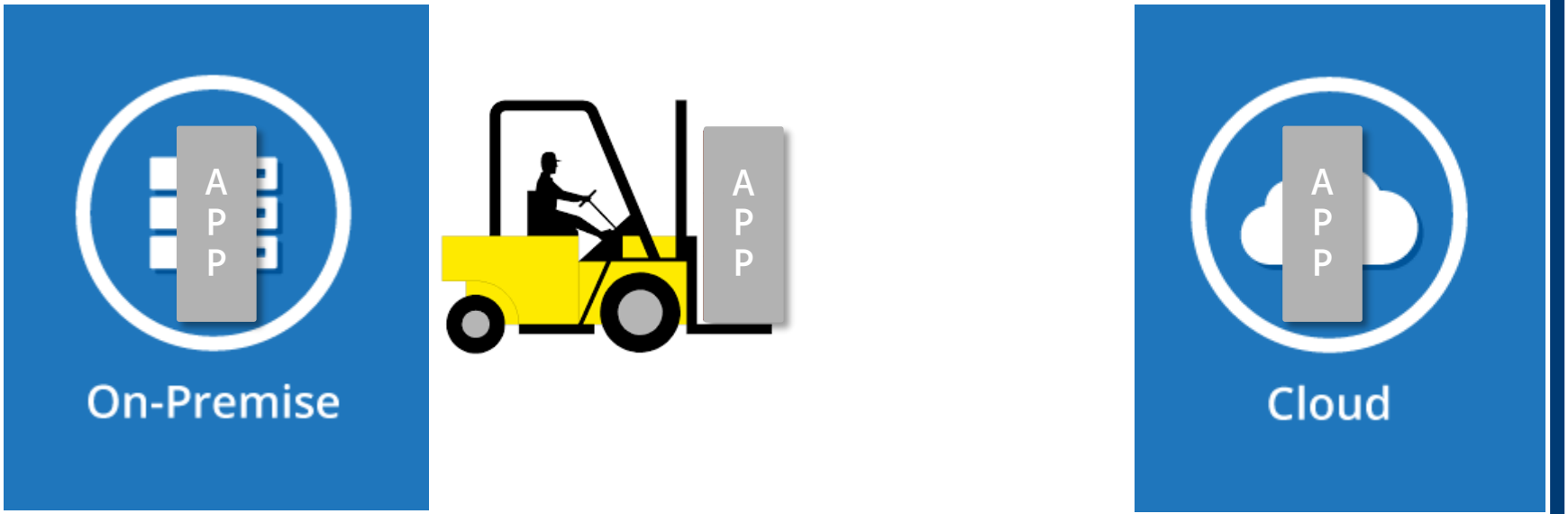
Team Member는?
현업? 운영자?

Cloud로 어떻게 Migration할 것인가?

CLOUD
MIGRATIONS

Cloud Migration

Fork and Lift



Cloud Native 의 성숙도 모델
(Source : Pivotal)

Cloud Native	• 마이크로 서비스 구조 사용과 원칙 준수 • API기반의 소프트웨어 아키텍처
Cloud Resilient	• 장애를 고려한 IT디자인 • 개별 애플리케이션의 장애가 전체 서비스에 영향을 주지 않는 디자인 • 적극적인 장애 테스트 - 커버리지 90% 이상을 추구 • 모니터링, 메트릭들을 플랫폼 차원에서 중앙화된 형태로 지원 • 퍼블릭/프라이빗 클라우드를 아우르는 스케일 전략
Cloud Friendly	• 마이크로 서비스의 12원칙 고려 • 수평적 확장이 가능한 구조 • 플랫폼 차원에서 HA 구조를 지원 - 애플리케이션 차원 HA
Cloud Ready	• App에서 파일 시스템 제거, 혹은 오브젝트 스토리지 도입 • 독립 실행형 애플리케이션 준비 • 플랫폼이 관리하는 backing 서비스를 사용

6 Strategies for Migrating Application to the Cloud
(Source : Amazon AWS)

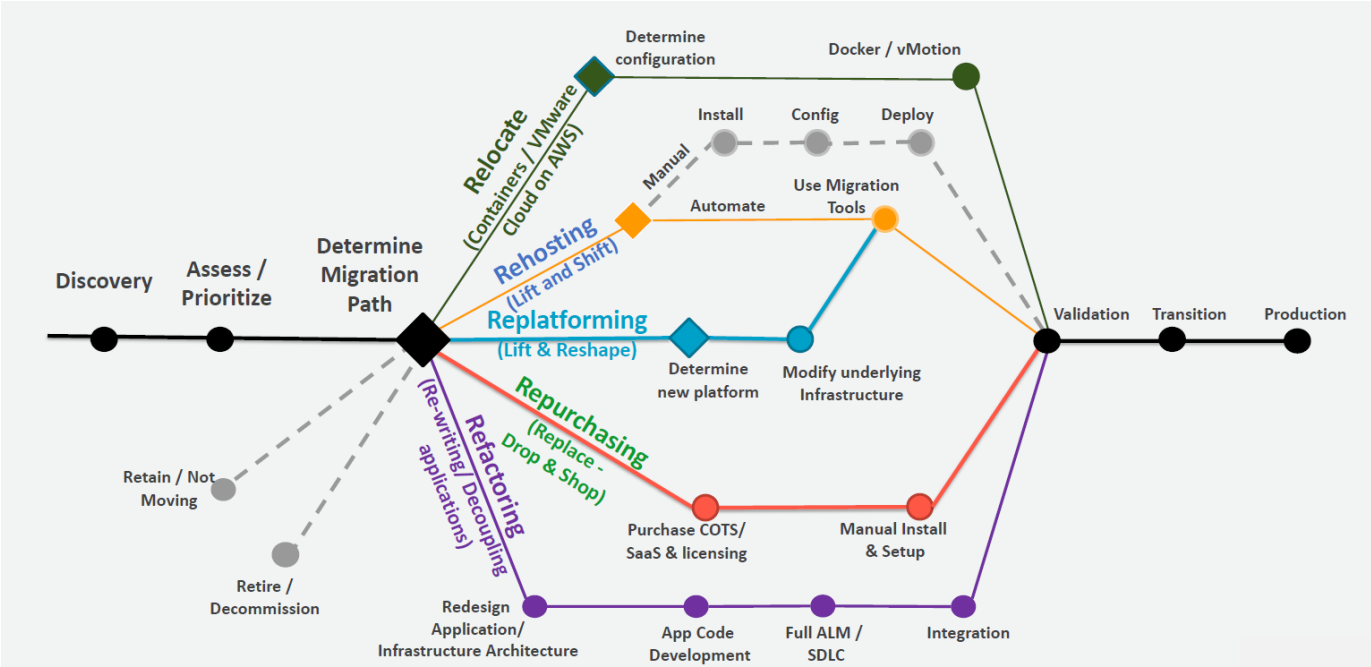
Discover/
Assess/
Prioritize
Applications

1. Rehosting : lift-and-shift
2. Replatforming : lift-tinker-and-shift
3. Repurchasing : Moving to a different product
4. Refactoring/Re-architeting
5. Retire - Get rid of
6. Retain - revisit or do nothing

+ 7. Relocate - Containers

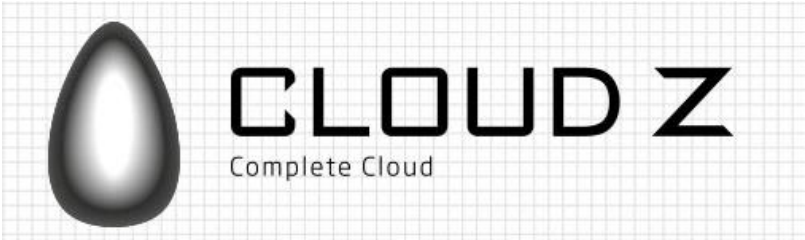
https://aws.amazon.com/ko/blogs/enterprise-strategy/6-strategies-for-migrating-applications-to-the-cloud/

Cloud Native or Lift-and-Shift?
(Source : Amazon AWS)



https://aws.amazon.com/ko/blogs/enterprise-strategy/cloud-native-or-lift-and-shift/

SK 주식회사 C&C Offerings



Cloud Consulting
SI, Delivery, OS 경험 기반의
Cloud 컨설팅

Managed Service
End-To End Monitoring
기반의 통합 관리 서비스

Migration Service
검증된 표준 프로세스 기반
마이그레이션 서비스

Digital 사업수행 방법론

CNAPS 2.0

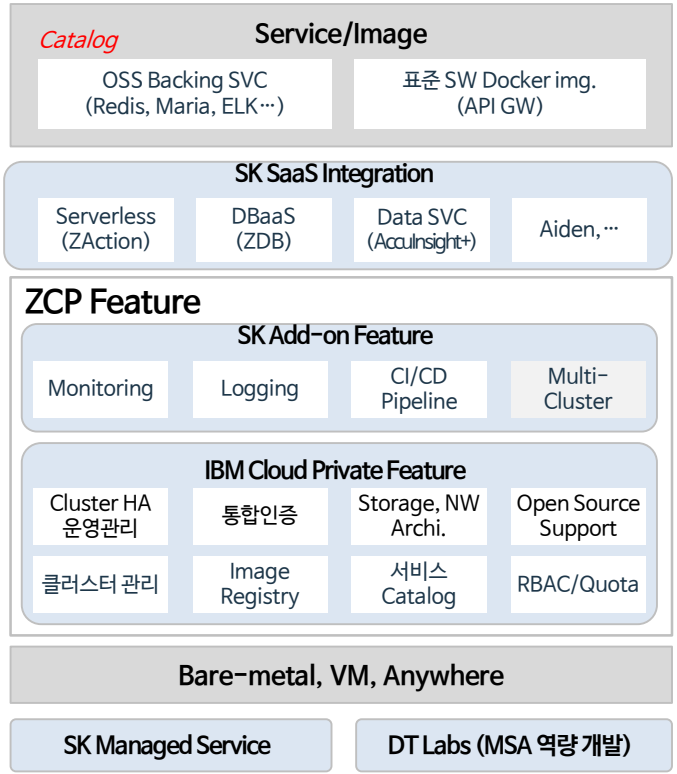
Cloud Native Application Development in PaaS

실무형 트레이닝

Digital Labs for X

Cloud / Blockchain / Data/AI / UX

CloudZ Container Platform (ZCP) 소개



- ✓ Platform Customizing
  - 고객사 별 Monitoring 및 logging Customizing
  - CICD Toolchain Customizing 및 Pipeline Workflow 설계
- ✓ 특화 Service Catalog 설계, Docker Container화 대상/방안 설계

- ✓ **SK SaaS 서비스 Integration** [1)]
  . Serverless , Data분석 Service, DBaaS, AI/Vision API
- ✓ **SK Add-on Feature**
  . Monitoring, Logging, CI/CD Pipeline
  . Hybrid Support, Multi-Cluster Mgmt (‘19.2Q)

- ✓ Kubernetes feature (IBM Cloud Private)
  . K8S Orchestration, Image Repository, Service Catalog, OPS
  . RBAC & Quota 관리, Docker Image 관리

- ✓ Platform Infra 가용성 구축 및 성능 최적화
  . Worker Node vs Container 분산 및 Failover 자동화 구성
- ✓ SK Managed Service, MSA 역량 개발 지원
  . Operation 및 L1/L2 Support, MSA 아키텍처 가이드/교육

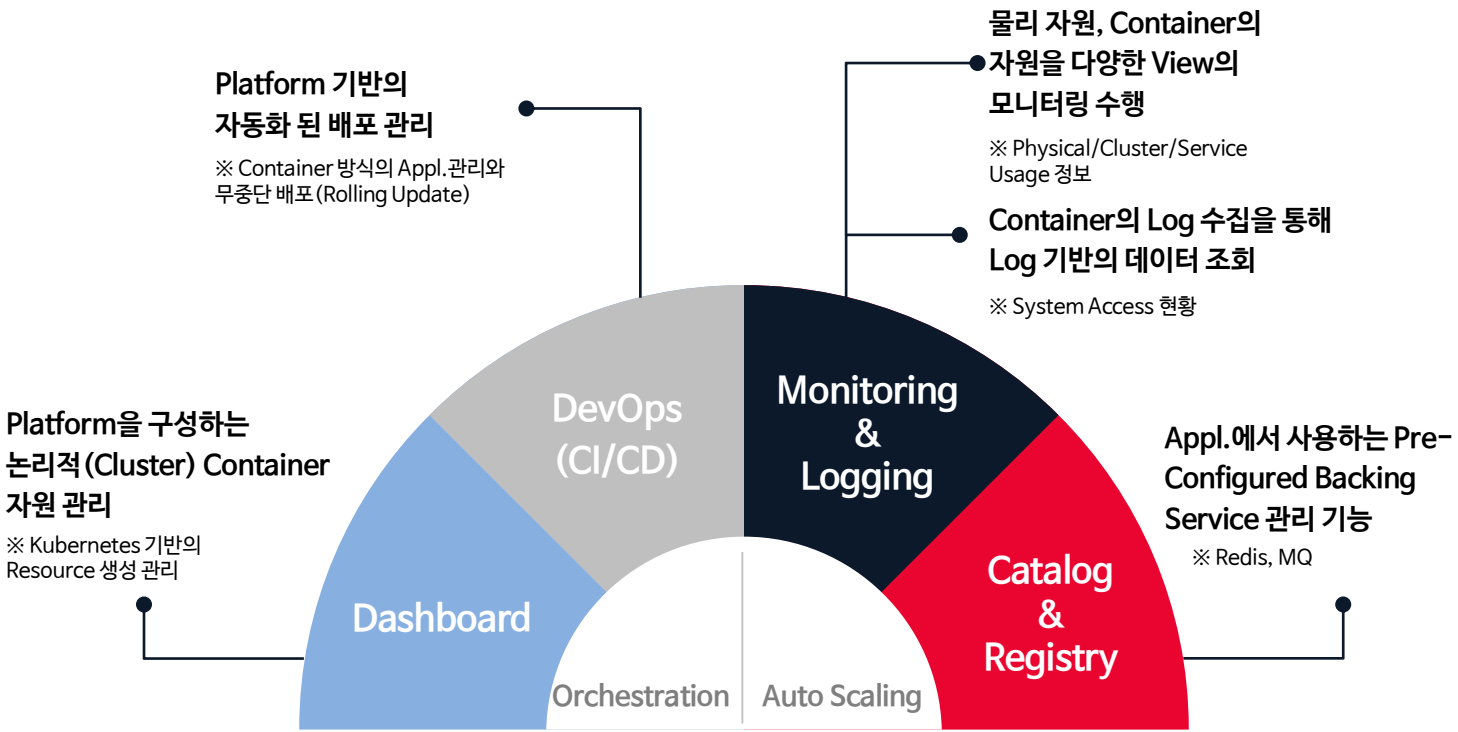
[1)] : SaaS Service 는 별도 협의 및 구매

ZCP 종류 - Public/Local

Public Cloud와 Private Cloud를 지원하는 Cloud Z CP - Public과 Cloud Z CP - Local이 있으며, 두 모델 모두 Managed Service를 지원

기반 서비스	Cloud Z CP Local	Cloud Z CP Public
설치 장소	고객 데이터센터 (On-Premise 환경)	SKC&C CloudZ Public
특징	• 고객 Data Center 및 Colocation 환경에 최적화하여 설치 구성 • On-Premise 환경의 아키텍처 수행과 설치형 플랫폼으로 제공 • 고객 비즈니스 보안적 요건 및 Enterprise 용 플랫폼 구축	• Cloud 상의 Public 서비스로 사용 • Kubernetes를 Managed Service로 제공 – Platform 운영을 고려하지 않고 활용 • Platform 사용 규모 중심의 과금 방식 • 쉽게 설치 및 삭제

ZCP - 주요 Features



ZCP - 주요 특징 (1)

Docker Container 기반 (Public ↔ Private)

- ✓ Virtualization의 Overhead를 배제하고 성능/집접도 향상
- ✓ 환경 변화와 무관하게 이동성이 자유로움 (Anywhere)

Compatible Platform (Kubernetes)

- ✓ AWS, Azure, Google, Openshift 의 Standard Kubernetes
- ✓ Appl.의 코드 변화 없이 쉽게 Migration 가능

※ Vendor Lock-in 무관

OSS (Open Source Software) Platform

- ✓ OSS 기반으로 구성된 Platform으로 확장이 용이
- ✓ In-House Skill로 기능에 대한 확장

Eco-system 기반의 확장

- ✓ Kubernetes Community Plugin & Tools
- ✓ Package Mgmt (Helm), Prometheus, Open Tracking

ZCP - 주요 특징 (2)

ZCP Local (Enterprise 기업 최적화)		ZCP Public (설치/구성 및 운영 효율)	
엄격한 보안 규제	고객 데이터 및 소스 보안	관리 비용	플랫폼 운영 비용 증가
신기술 도입	Cloud-Native 로의 기술 전환의 어려움	기술의 복잡도	Container, K8S 등 복잡도 증가
벤더 종속	구축시 특정 기술에 Lock-in	다양한 환경 필요	비즈니스 혁신을 위한 빠른 환경 마련
관리의 어려움	고객 직접 관리 or 관리형 서비스	Hybrid/Multi 요구	Public 과 On-Premise 병행 Workload

고객 환경에 설치

- 고객 환경에서 최적화된 플랫폼으로 구성
- 필요시 Managed Service 제공
- 엔터프라이즈 워크로드 기반의 보안

클라우드-네이티브 앱을 위한 런타임 및 서비스

- 클라우드 네이티브 앱에 필요한 컴퓨팅 환경 (Container, Cloud Foundry) 과 다양한 서비스 제공
- DT Labs를 통한 교육

Hybrid 기반 아키텍처 구현

- ZCP Public/Local간의 상호 운영성 기반의 아키텍처 구현
- Consistency Experience 제공

오픈소스 기반 벤더락인 탈피

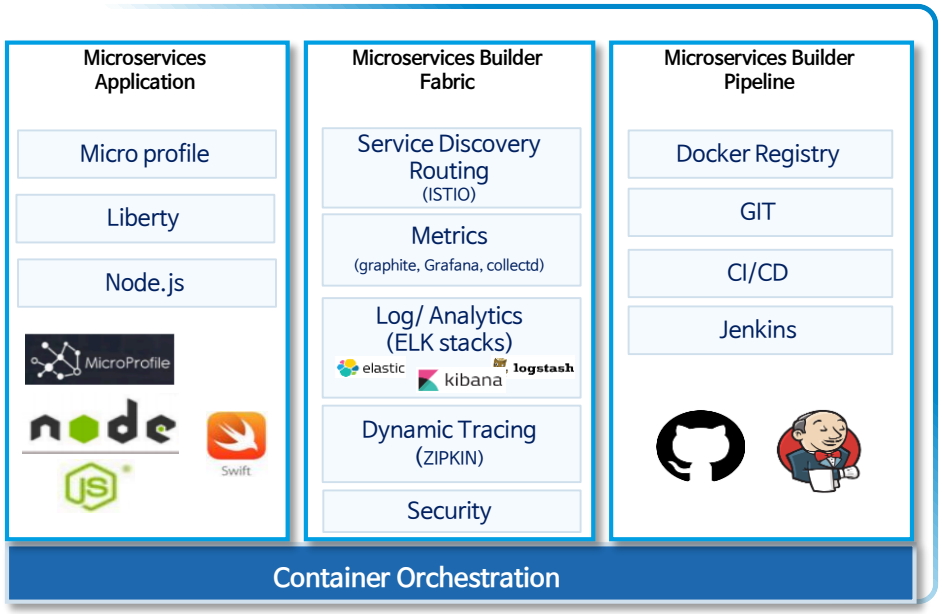
- 오픈소스 컨테이너 및 컨테이너 오케스트레이션 (Kubernetes / Docker) 기반 플랫폼
- 오픈소스 서비스 제공 및 사용자 서비스 추가 가능

유연한 환경 (검증, 테스트 등)

- ZCP Public 통한 다양한 환경 경험
- 비용, 운영비 절감의 효과

ZCP - DevOps

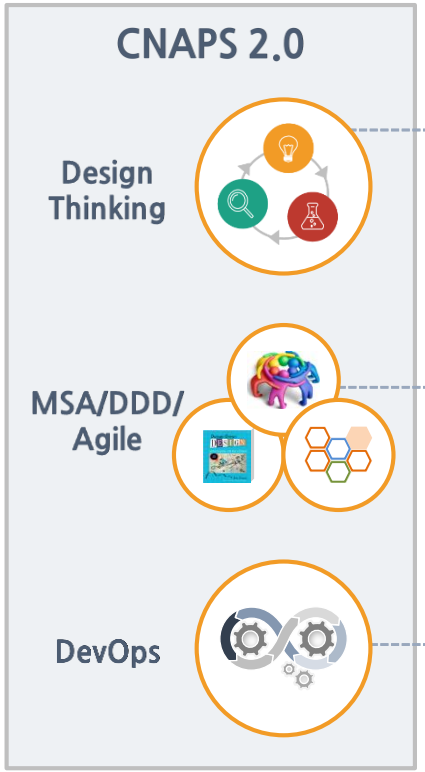
Microservices Builder 를 제공하여, 마이크로서비스 어플리케이션을 위한 실행 환경 및 개발 도구, CI/CD Pipeline 구성을 지원합니다.



- ✓ Microservices 를 개발 및 운영을 위한 통합 솔루션
- ✓ 실행환경+ 개발 도구 + Open Source + DevOps + Fabric을 제공
- ✓ Pre-built 된 통합 환경 환경으로 Microservices 개발 환경을 빠르게 구축
- ✓ Git Hub / Git Hub Enterprise연계
- ✓ Jenkins를 통한 파이프라인 구성
- ✓ Build Green Deploy 지원

CNAPS 소개

빠르고 유연하게 DT向 최적화된 어플리케이션을 만들기 위한 새로운 개발방식



핵심요소 01

사용자 중심의 제품/서비스 컨셉 개발

- 고객의 핵심문제 해결을 위한 Biz. 혁신 기회 발굴 과 Right Solution 도출
- Prototyping과 Feedback 수렴으로 빠른 제품·서비스 컨셉 개발

핵심요소 02

Microservice의 쉬운 설계와 점진적 구현

- 빠르게 변경·배포 · 대체 · 확장 가능한 작고 독립적 Appl. 개발
- 기술중심이 아닌 Biz. 중심 언어 사용하여 복잡도 감소, 이해도 향상
- 짧은 주기의 출시 가능한 수준의 릴리즈를 반복/점진적으로 완성

핵심요소 03

개발/운영 협업으로 더 빠르고 안정적 서비스 출시

- 사례를 통해 획득한 Asset 기반의 DevOps 적용
- 빠른 서비스 출시가 가능하도록 자동화 Toolchain을 개발부터 운영까지 연동

CNAPS - 공정 영역별 주요 특징

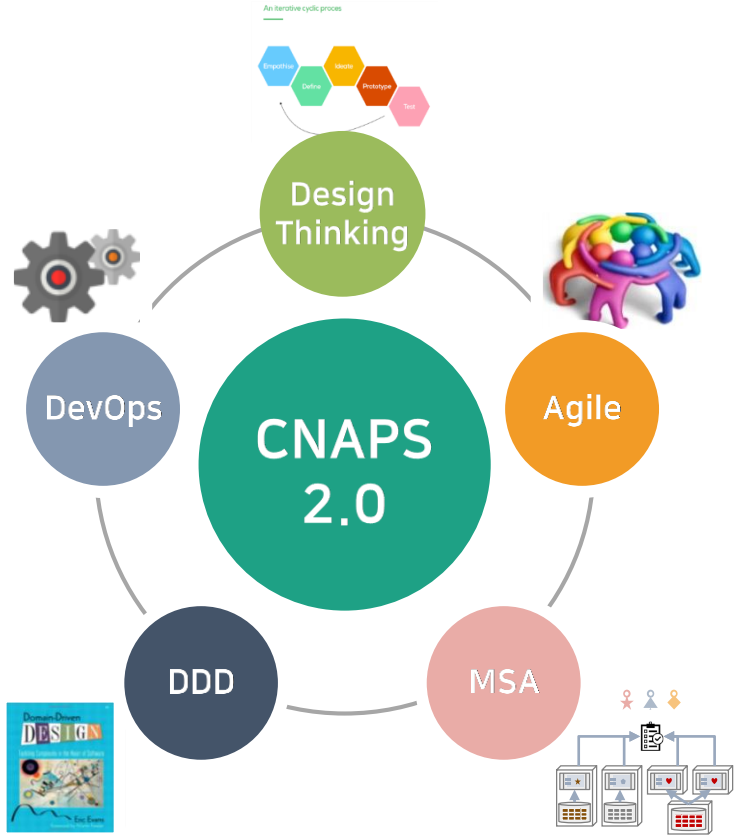
특징 01
Design Thinking
사용자 중심의 빠른 제품/서비스 컨셉 개발

특징 02
Agile Delivery
짧은 주기의 출시 가능한 수준의 제품을 반복/점진적으로 완성

특징 03
Microservice Architecture
빠르고,쉽게 변경/대체 가능한 서비스 기반 아키텍처

특징 04
Domain Driven Design
Biz. 도메인 중심 설계, Biz. 기능과 구현기술 분리 강조

특징 05
DevOps
개발/운영 협업을 통한 빠른 서비스 출시

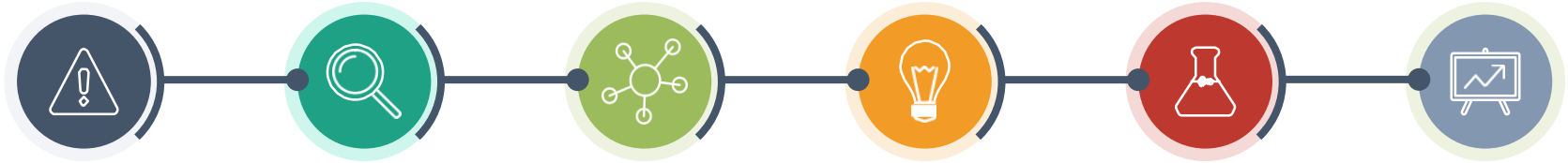


Design Thinking (서비스 기획)
- 사용자 중심의 빠른 제품/서비스 컨셉 개발 -

1.Initiation
디자인 문제 및 목표 정의
Hopes & Gripes
To-know List

3.Interpretation
핵심 사용자 문제 정의
Journey Map(As-Is)
POV(Point of View)
HMW(How Might We)

5. Experimentation
프로토타이핑 및 사용자 검증
Storyboard
Rapid prototyping
Feedback Grid



2.Discovery
사용자 리서치 및 인사이트 수집
Persona
Empathy Map
Stakeholder Map

4. Ideation
아이디어 도출 및 컨셉 정의
Brainstorming
Prioritization Grid
Ideas / Concepts

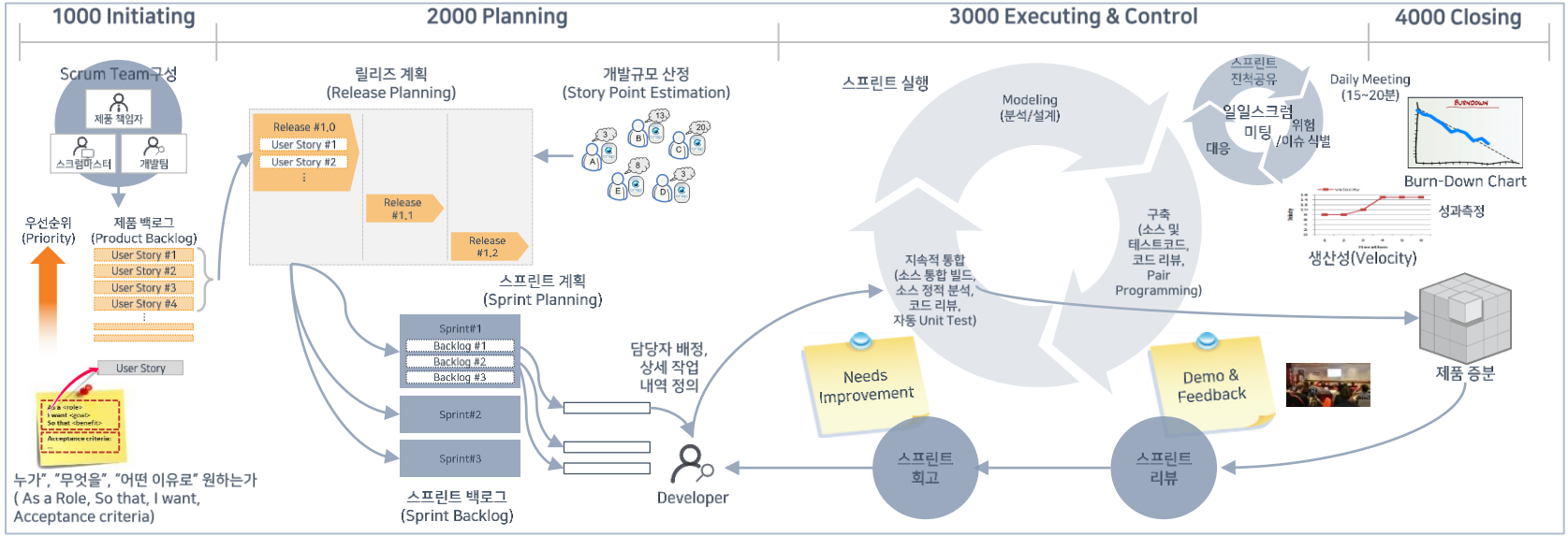
6.Evolution
측정과 평가, 실행계획 수립
Service Blueprint
Business Canvas
Value Proposition

주요 특징

- 제품/서비스 요구사항이 불명확한 상황에서 사용자에 대한 공감을 기반으로 “올바른 문제(Right Problem)” 정의
- Workshop과 Prototyping의 피드백을 신속하게 반영하여 제품/서비스 Concept을 빠르게 구체화하려는 고객에 적합
- 사전 영업단계의 DT기술을 활용한 서비스 개발과, Scope이 명확하지 않은 프로젝트에 개별 적용 가능

Agile Delivery (PJT 수행/관리)

짧은 주기의 출시 가능한 수준의 제품을 반복/점진적으로 완성



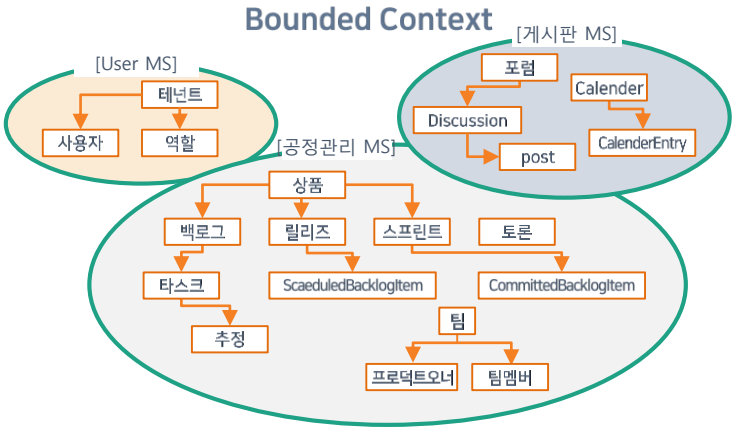
주요 특징

- 사용자 관점의 요구사항을 우선 순위에 따라 짧은 주기로 반복/점진적인 제품으로 출시 가능하도록 정립
- 표준 협업 도구(Jira, confluence) 기반으로 빠른 실행력 극대화를 위한 Practical Guide 제공
- AI, Big Data, Block-Chain 등 다양한 DT기술을 활용한 Delivery 방식에 유연하게 적용 가능

Domain Driven Design (분석/설계)

Biz도메인 중심으로 영역 분리, 시스템 내부 구조를 Biz와 기술 의존 영역으로 분할

Microservice 도출



- Biz 중심으로 독립된 영역을 식별하고, Microservice를 도출

Microservice 내부설계 및 구현

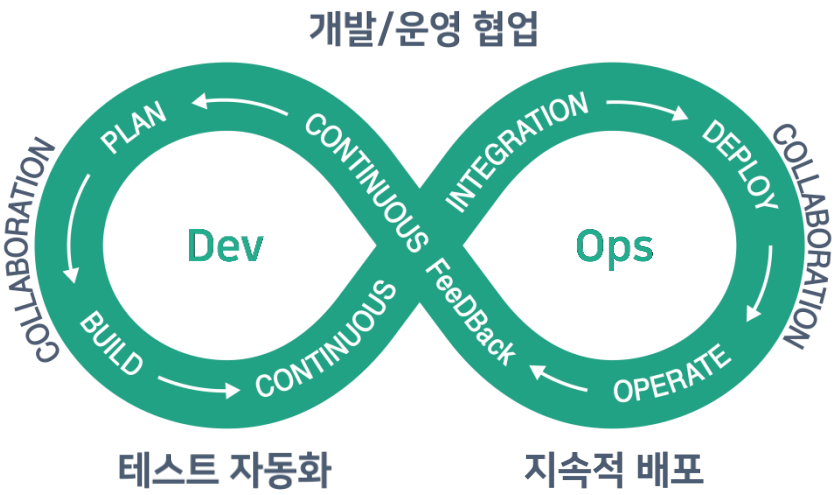
- 기술영역 과 도메인 영역을 분리하여 구성
- 기술 변경 시 기술영역만 대체/확장↑,
  Biz 변경 시 도메인 모델 만 변경, 유지보수성↑

주요 특징

- 독립된 Microservice 도출 위한 효과적 기법 제공
- 공통의 언어로 업무전문가·설계자·개발자가 Biz. 개념을 명확하게 인식/공유하고, SW 코드 형태로 반영
- 변경에 유연하고 빠르도록 기술과 도메인 영역을 분할하여 구현

DevOps

개발/운영 조직 간 협업을 통한 빠르고 안정적인 서비스 출시



01. 테스트 자동화를 통한 서비스 안정성 확보

- 회귀테스트 자동화(Unit,API,GUI)를 통한 테스트 생산성 확보
- 테스트합격 소스만 통합/빌드 하여 장애 사전 예방
- 테스트 결과 정량적 관리 하여 품질 가시성 확보

02. 지속적 배포를 통한 빠른 서비스 출시

- 변경 소스의 실시간 빠른 배포
- 배포 Pipeline 자동화 구성으로 Lead Time 최소화
- Blue-Green 배포 및 Canary 릴리스로 無 중단 서비스

03. 개발/운영을 위한 협업 기능 제공

- 운영 중 발견 개선사항을 개발 일감으로 실시간 F/B
- 개발자·운영자 간 정보의 상호 공유
- 성능 측정 및 Biz 목표 달성 모니터링

주요 특징

- 실제 적용 사례를 통해 개발된 Asset 기반의 DevOps 적용
- 빠른 서비스 출시가 가능하도록 개발부터 운영까지 모든 Pipeline이 Toolchain 으로 연동

마지막으로 생각해 볼 것…

- Cloud Native Architecture 특징과 기술을 전반적으로 이해하기 위해서는 시간이 필요한데, 기존 폐쇄된 조직에서 저변확대 하는데 까지 시간이 많이 걸림
  → Open-mind 와 배우려는 자세가 필요하고 조직의 변화관리 또한 중요

- Cloud Native를 위한 기반 환경이 뒷받침 되어야.. (빠른 배포 환경 및 Backing service)
  - ✓ 컨테이너기반, Shared Memory (Redis), MQ 등은 MSA를 위해 필수적인 Backing Service 임.

- Microservice 를 어떤 크기로 어떻게 구분할 것인가에 대한 기술적 논의는 오히려 쉬움
  - ✓ 업무를 명확히 이해하는 관련자들과 협업이 필요, 서로간 독립적이고, 종속성을 최소화 할 수 있는 단위.

Conclusion

Digital Experience 를 위한 끊임없는 노력

이를 위해 필요한 것

Infra로서 Cloud / Container의 적절한 선택과 적용

조직/문화로서 DevOps 적용 및 확산

Application은 Microservices를 목표로 단계별 전환

감 사 합 니 다