PaaS-TA

Cloud 이해

교재

NIA 한국정보화진흥원

학습목표

- Cloud 및 PaaS-TA를 이해한다.
- PaaS-TA를 활용해 어플리케이션 개발 및 배포·운영을 할 수 있다.
- 자사 솔루션을 PaaS-TA 호환 서비스로 개발할 수 있다.

INDEX

M1. Cloud Basic

01. Cloud 이해

기본속성
Cloud 발전과정
Cloud 기반 사업 추진 기업
트렌드

02. Cloud Model 및 특징 이해

Cloud Model
IaaS
PaaS
SaaS

M2. PaaS-TA 개발 실무

01. PaaS-TA 이해

PaaS-TA 소개
PaaS-TA 아키텍처의 이해
PaaS-TA 주요 구성요소
PaaS-TA 운영 및 편의 도구
PaaS-TA 상황별 구성요소 기능

02. PaaS-TA 개발 환경의 이해

애플리케이션 환경 이해
서비스 이해
개발 환경 이해

03. PaaS-TA 개발도구 이해 및 실습

Portal
CLI
IDE
SCM
배포 Pipeline

04. PaaS-TA 개발 검증 및 문제해결

Cloud Native Application 개념
12 Factors
Micro Service Architecture

M3. PaaS-TA 배포 및 운영

01. PaaS-TA 배포 및 관리

PaaS-TA배포를 위한 기본 구성
BOSH 개념
Director 설치
BOSH를 통한 PaaS-TA 구축

02. Service Package 배포 및 관리

PaaS-TA Service Package concept
Service Broker 예제 분석
Service Broker 등록 및 활성화

03. Custom Buildpack 개발

Buildpack 개발 가이드
Creating Custom Buildpacks

M1 Cloud Basic

01. Cloud 이해

02. Cloud Model 및 특징 이해

M1 학습목표

- Cloud를 이해한다.
- Cloud Model의 개념과 특징을 이해한다.

01. Cloud 이해

M1. Cloud Basic

01. Cloud 이해

- 기본속성
- Cloud 발전과정
- Cloud 기반 사업 추진 기업
- 트렌드

02. Cloud Model 및 특징 이해

M2. PaaS-TA 개발 실무

01. PaaS-TA 이해

02. PaaS-TA 개발 환경의 이해

03. PaaS-TA 개발도구 이해 및 실습

04. PaaS-TA 개발 검증 및 문제해결

M3. PaaS-TA 배포 및 운영

01. PaaS-TA 배포 및 관리

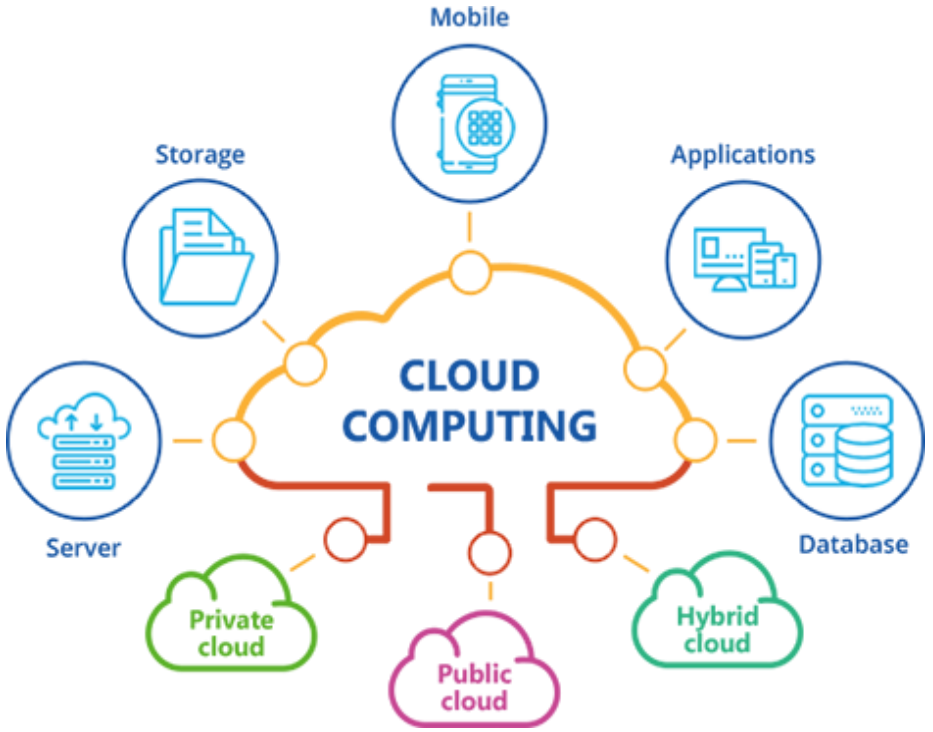
02. Service Package 배포 및 관리

03. Custom Buildpack 개발

01 기본속성

» 정의

클라우드 컴퓨팅이란



네트워크 기반의 컴퓨팅 기술

컴퓨팅 리소스를 데이터센터에 대량으로 집적시킨 후, 개별 이용자가 요구하는 만큼 가상으로 분리하여 정보통신망을 통해 제공하는 서비스

사용량에 비례하여 비용 청구

출처: NIST(미국 국립표준기술원)

01 기본속성

» 특징

주문형 셀프 서비스	사업자와 직접 상호 작용하지 않고, **사용자의 개별 관리화면을 통해 서비스를 이용**할 수 있음
광범위한 네트워크 접속	모바일 기기 등의 **다양한 디바이스를 통해 서비스에 접속**할 수 있음
리소스 공유	사업자의 컴퓨팅 리소스를 **여러 사용자가 공유하는 형태로 이용** ↳ 컴퓨터 시스템에 관한 여러 가지의 자원을 총칭하는 말
신속한 확장성	필요에 따라, 필요한 만큼 **스케일 업(처리능력을 높이는 것)과 스케일 다운(처리 능력을 낮추는 것)** 가능
측정 가능한 서비스	이용한 만큼 요금이 부가되는 **종량제**

출처: NIST(미국 국립표준기술원)

01 기본속성

» Cloud 유용성

경제성

- ✓ 사용하고자 하는 기간만 사용 가능
- ✓ SW와 데이터를 클라우드에서 통합 관리함으로써 SW 업데이트 작업 및 데이터 유지보수의 효율성을 높여 비용을 절약할 수 있음

유연성

- ✓ 컴퓨팅 리소스를 필요할 때 필요한 만큼 확장하고, 필요하지 않을 때는 축소하는 등 유연한 활용이 용이

CLOUD

가용성

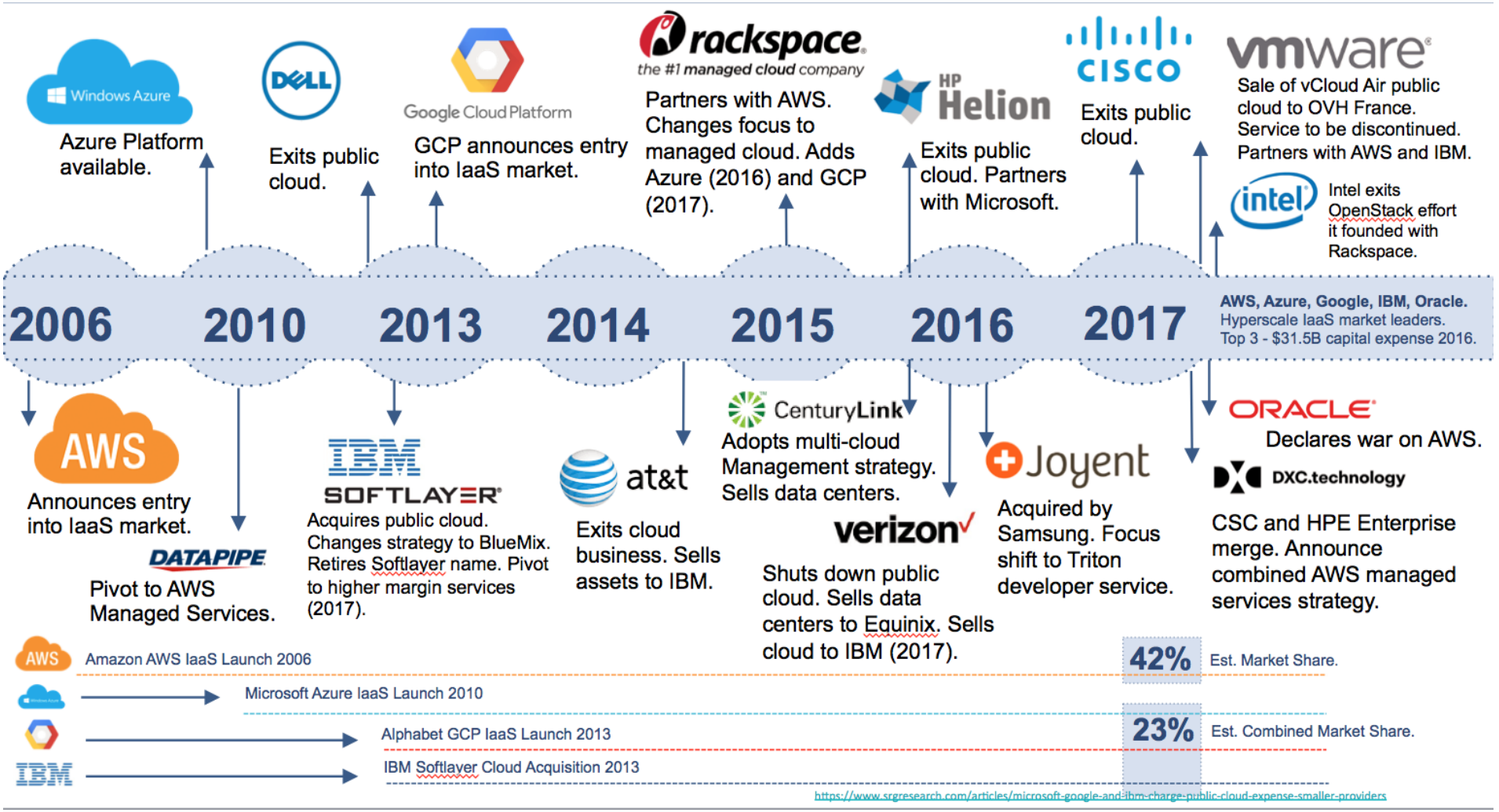
- ✓ 일부 하드웨어에 장애가 발생하더라도 서비스를 계속해서 사용할 수 있도록 구성되어 있음
- ✓ 자체 시스템을 구축할 때보다 낮은 가격에 가용성이 높은 환경을 사용할 수 있음

빠른 구축 속도

- ✓ 클라우드가 제공하는 하드웨어와 소프트웨어를 이용하여 시스템을 신속하게 구축 가능

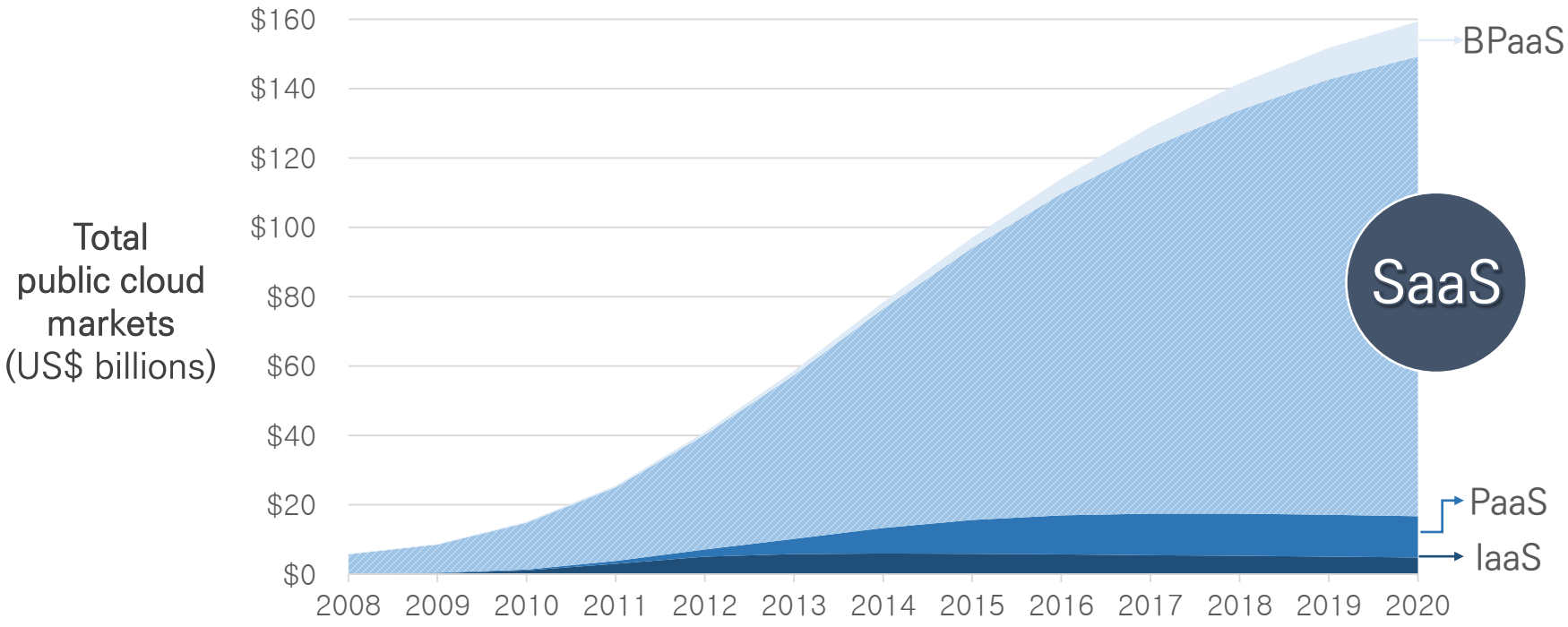
02 Cloud 발전과정

» Cloud 연혁



02 Cloud 발전과정

» Cloud 시장 규모 변화

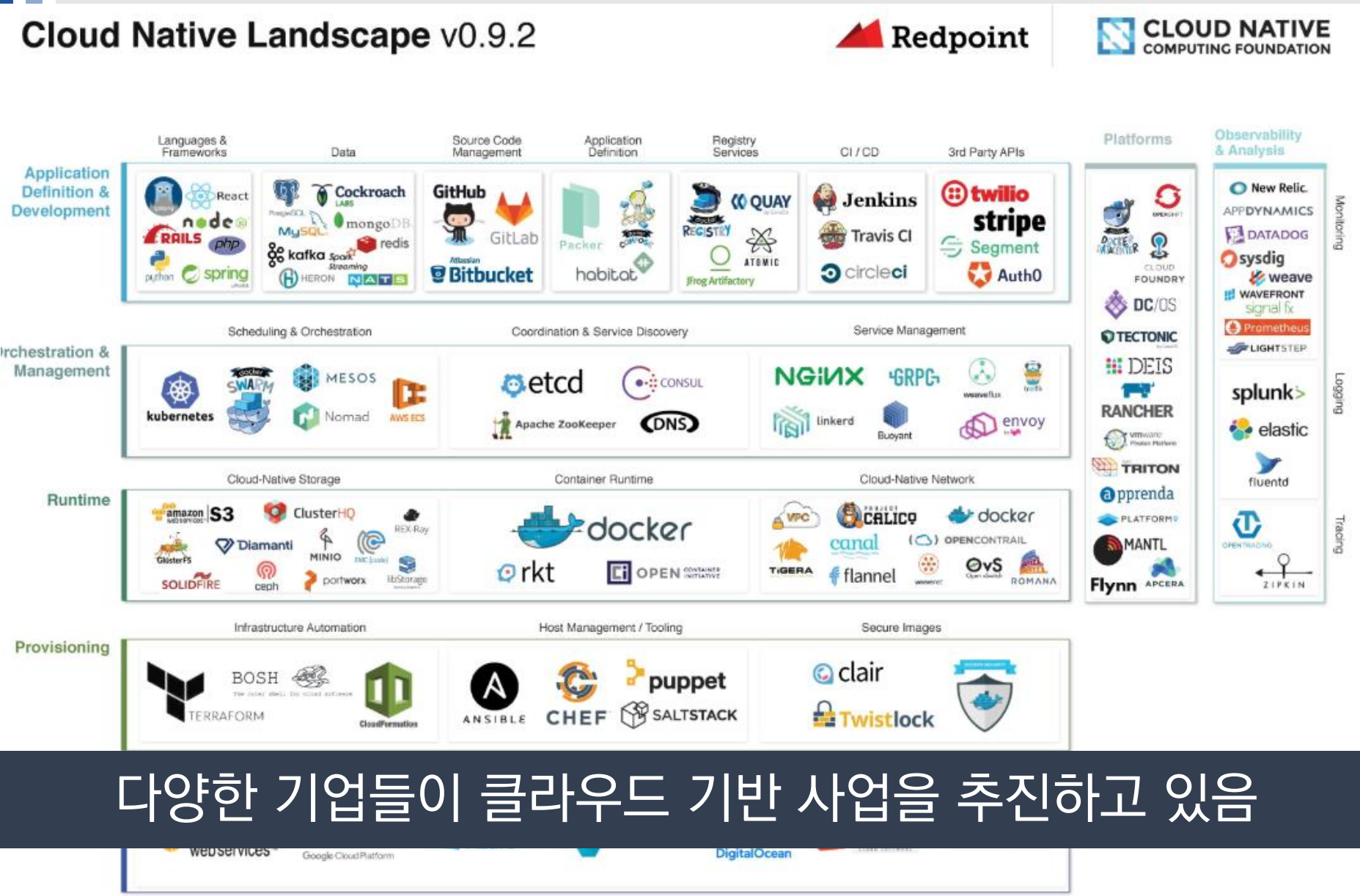


2020년 Public Cloud 시장규모 1,600억 달러 돌파 예상

요구사항의 즉각적인 서비스화를 위해 클라우드 컨테이너의 자동화 관리와 조절이 필수

출처: Forrester Research, Inc.

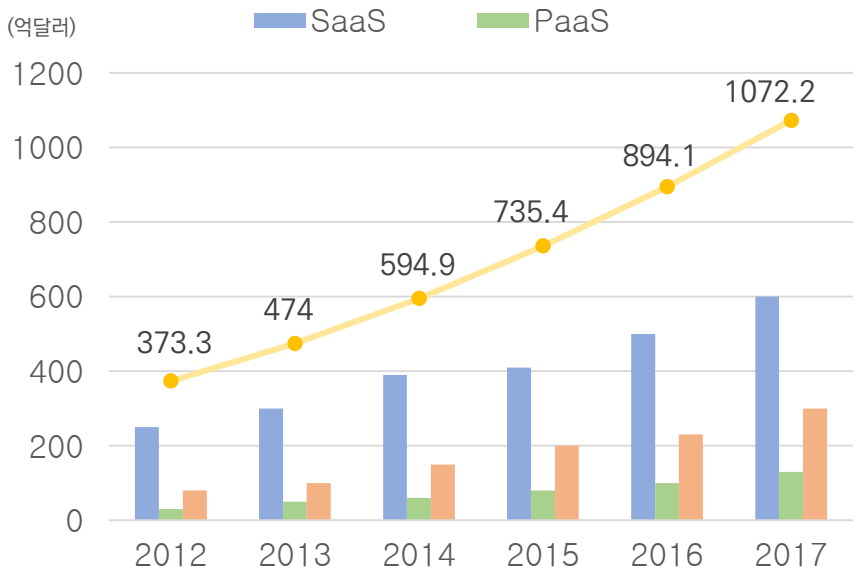
03 Cloud 기반 사업 추진 기업



04 트렌드

» Cloud 시장 동향

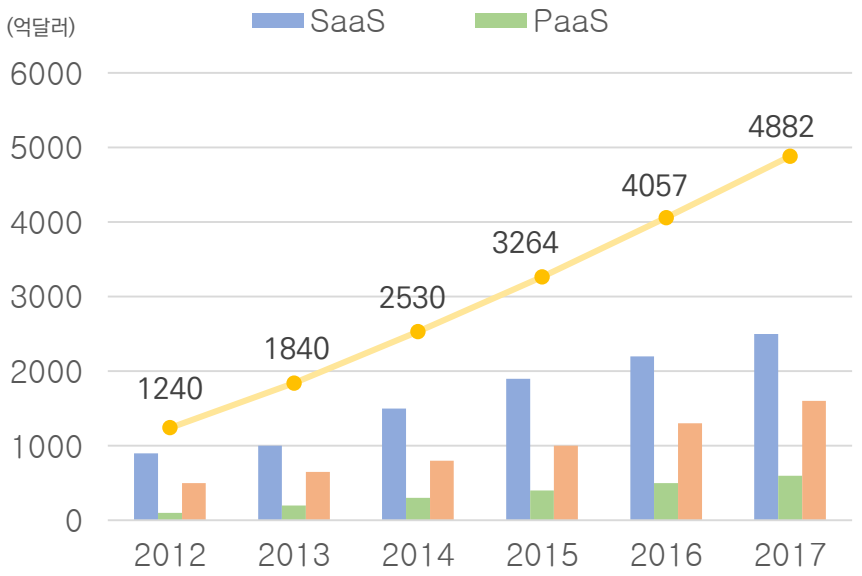
해외 클라우드 서비스 시장 전망



2017년까지 해외시장 규모 1,070억 달러

SaaS 600억 〉 IaaS 300억 〉 PaaS 170억 달러

국내 클라우드 서비스 시장 전망



2017년까지 국내시장 규모 4,900억 원

SaaS 2,600억 〉 IaaS 1,900억 〉 PaaS 400억 달러

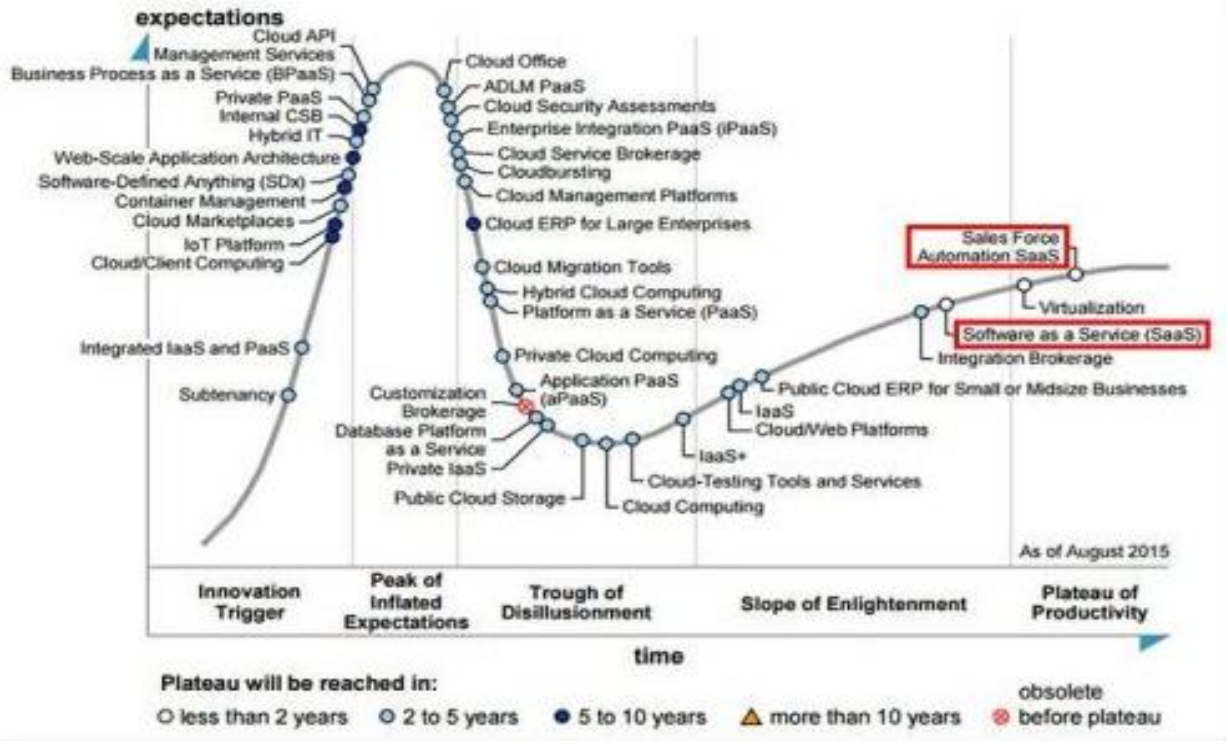
국내 PaaS 시장이 상대적으로 작아 투자를 기피하는 경향 有

출처: IDC

04 트렌드

» Cloud 기술 동향

해외 클라우드 서비스 시장 전망



PaaS 서비스 제공 기업의 급증

Amazon(AWS), Google(GCP), Microsoft(Azure), Pivotal(Cloud Foundry), IBM(Bluemix), Red Hat(Open Shift), vmware(vCloudAir) 등 70개 이상의 PaaS 기업이 탄생

PaaS 부가 기능을 통한 다양성 증대

- IBM은 왓슨을 통한 인공지능 서비스 제공
- 아마존, MS등은 IoT 기능을 플랫폼에 도입
- GE* 등은 제조와 클라우드를 접목한 PaaS 출시

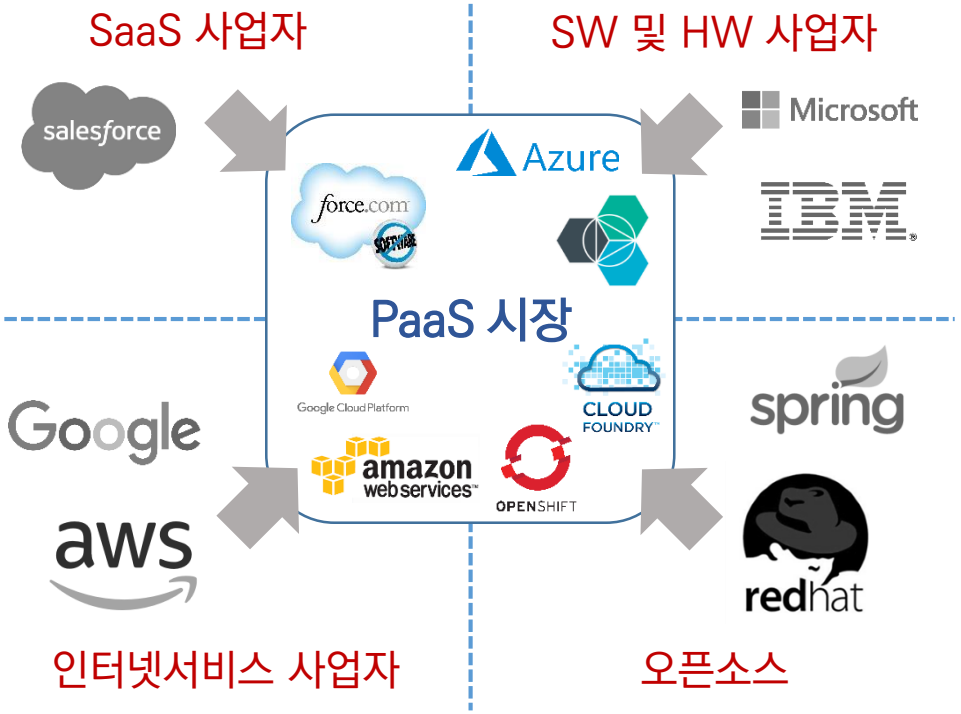
글로벌 기업들은 PaaS에 주목하고 활발한 투자와 연구 개발 진행

출처: 가트너의 하이프 사이클(2015)

04 트렌드

» Cloud 산업 동향

현재 PaaS 시장



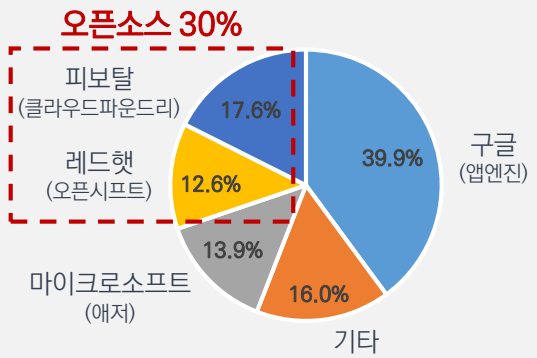
각광받는 오픈소스

① 치열한 개발 경쟁 속에서 종속성 문제 대두

- PaaS에 대한 SaaS의 종속성이 크게 증대

② 종속성 문제해결의 대안으로 오픈소스 급부상

도입 희망 클라우드 플랫폼



최근 오픈소스 기반의 개방형 클라우드 플랫폼 각광

출처: ZENOSS Survey

04 트렌드

» 2018년 클라우드 컴퓨팅 5대 트렌드

- 클라우드 서비스 솔루션의 성장
- 클라우드 스토리지 용량 증가
- IoE(Internet of Everything)의 부상
- 인터넷 품질의 개선과 5G 네트워크의 부상
- 사이버 보안 시스템에 대한 투자

출처: CIO Report

04 트렌드

» 2018년 클라우드 컴퓨팅 5대 트렌드

클라우드 서비스 솔루션의 성장

SaaS는 기업들이 **초기 클라우드 서비스를 접할 수 있는 유연하고 경제적인 경로 제시**

IaaS와 PaaS의 성장은 **퍼블릭과 프라이빗 부문**에서 사용할 수 있는
클라우드 솔루션의 수를 늘림

올해 더 많은 조직들이 클라우드가 보장하는 단순성과 고성능을 활용하게 될 것이며,
특히 **SaaS 솔루션이 압도적으로 많이 설치된 클라우드 서비스가 될 전망**

고객들이 서비스에 더 쉽게 접근하게 만들고 싶어하는 기업의 경우
SaaS, IaaS, PaaS를 통합하는 방향으로 움직일 전망

04 트렌드

» 2018년 클라우드 컴퓨팅 5대 트렌드

클라우드 스토리지 용량 증가

데이터 스토리지의 폭발적 증가에 따른 서비스 업체들의
더 큰 용량의 스토리지 장비를 갖춘 데이터센터 개설 예측

스토리지 총 용량이 작년 600엑사바이트에서 1.1 제타바이트로 약 2배 증가

미래지향적 기업들은 자사의 목표를 달성하기 위해 클라우드 공간을 적극 활용할 것

04 트렌드

» 2018년 클라우드 컴퓨팅 5대 트렌드

IoE(Internet of Everything)의 부상

IoT가 성장하는 가운데, **실시간 데이터 분석 및 클라우드 컴퓨팅의 지속적인 혁신**이 IoE를 전면에 내세울 것으로 예상

IoE는 네트워크 모든 기기들과 지능적으로 상호작용하며, **인간과 인간의 소통도 훨씬 더 쉽게 만드는 기능**을 함

고객들이 자사의 제품이나 서비스, 고객 지원 부서, 서로 간에 어떻게 연관되는지에 대해 **더 많은 통찰력 제공할 예정**

쌓인 데이터는 자동화와 스마트 로봇 사용을 통한 고객 경험 단순화 등 다양한 방법으로 사용됨

04 트렌드

» 2018년 클라우드 컴퓨팅 5대 트렌드

인터넷 품질의 개선과 5G 네트워크의 부상

많은 기업이 인터넷 품질 향상을 위해
기가바이트 **LTE 속도에서 완전한 5G 네트워크로 변화**할 것

개선된 네트워크 품질은 **응답성이 매우 뛰어나고**,
빠르게 구동되는 서비스와 앱에 대한 **소비자들의 기대치를 높일 것**

IoT와 IoE 업계도 이 분야에 속해 있는 조직들이 실시간으로 데이터를 수신하고 더욱 효율적으로 전달할 수 있게 해줌으로써, **더 빠른 네트워크 속도로부터 수혜를 입을 전망**

04 트렌드

» 2018년 클라우드 컴퓨팅 5대 트렌드

사이버 보안 시스템에 대한 투자

사이버 공격이 정교해짐에 따라, 기업들은 사이버 보안에 대한 기본적인 방어 매커니즘으로 SIEM(Security Information and Event Management)과 악성코드 탐지 시스템에 대한 투자 필요성을 느껴야 함

클라우드 서비스가 완벽한 보안 대책을 구현할 수 없는 기업을 위해 관제 보안 서비스 공급업체들을 연결해줌으로써 보안 영역에서 크게 성장 가능함

04 트렌드

» 2019년 클라우드 컴퓨팅 트렌드

SaaS 공급업체들은 엔터프라이즈 애플리케이션 포트폴리오를 강화

기업들이 애플리케이션과 워크로드, 데이터를 대대적으로 클라우드 네이티브 백본으로 마이그레이션하는 노력 가속화

퍼블릭 공급업체들이 완전 관리형 온프레미스 어플라이언스를 자신들의 하이브리드 클라우드 '진입 차선 ' 으로 삼는다

핵심 오픈소스 코드로서의 기반이 안정화되면서 쿠버네티스 도입에 속도가 붙는다.

솔루션 공급업체는 네트워크 운영 체제 내부에 쿠버네티스를 구현한다.

출처: CIO Report

04 트렌드

» 2019년 클라우드 컴퓨팅 트렌드

콘테이너가 상태 저장 및 상태 비저장 시멘틱 모두를 지원하는 수준이 높아짐

서버스 메시가 멀티클라우드의 유력한 네트워크 관리 백플레인이 된다.

클라우드-투-엣지 분산형 컴퓨팅 패브릭이 확대

클라우드 네이티브 도구에 가상화, 컨테이너화, 서버리스가 융합

컨테이너화 마이크로서비스 마켓플레이스가 확대

출처: CIO Report

MEMO

02. Cloud Model 및 특징 이해

M1. Cloud Basic

01. Cloud 이해

02. Cloud Model 및 특징 이해

- Cloud Model
- IaaS
- PaaS
- SaaS

M2. PaaS-TA 개발 실무

01. PaaS-TA 이해

02. PaaS-TA 개발 환경의 이해

03. PaaS-TA 개발도구 이해 및 실습

04. PaaS-TA 개발 검증 및 문제해결

M3. PaaS-TA 배포 및 운영

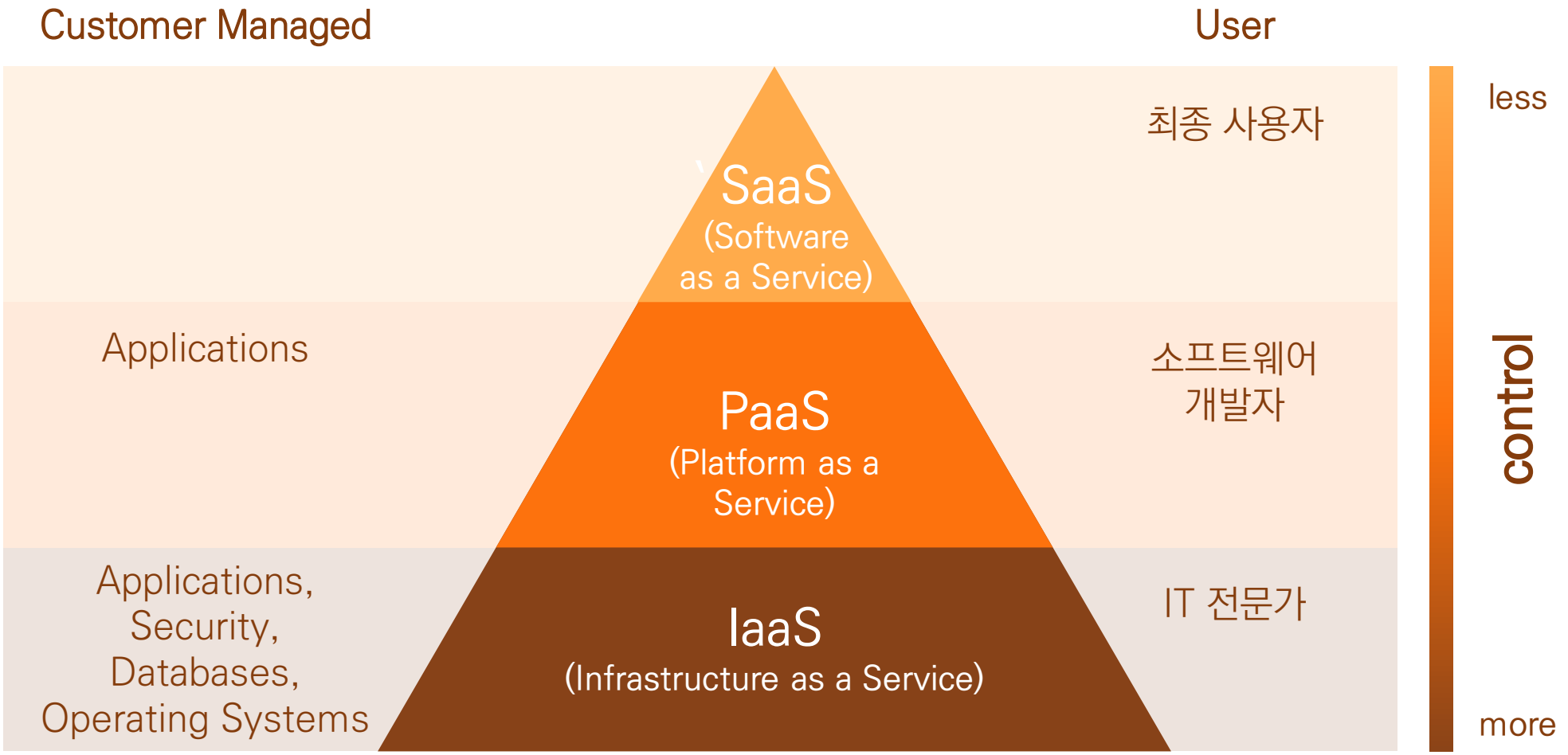
01. PaaS-TA 배포 및 관리

02. Service Package 배포 및 관리

03. Custom Buildpack 개발

01 Cloud Model

» Model별 관리 범위와 예시



01 Cloud Model

» Model별 관리 범위와 예시

설치형 S/W ------ 서비스형 S/W ------ 마켓플레이스

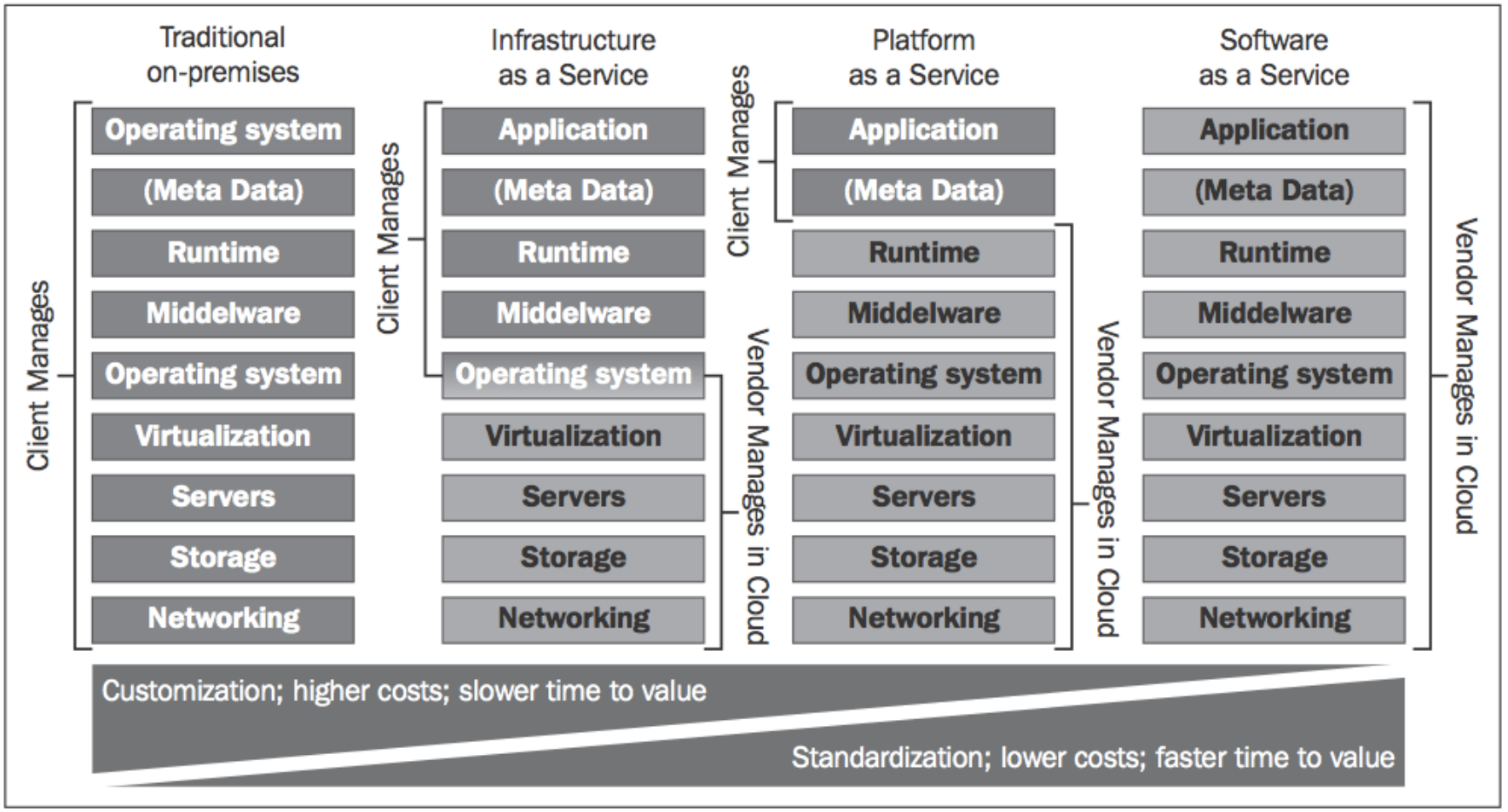
IaaS	CaaS	PaaS/aPaaS	FaaS	SaaS
Infra 관점	**컨테이너 관점**	**Runtime 관점**	**서비스 관점**	**비즈니스 과점**
openstack.	docker	CLOUDFOUNDRY	OpenWhisk	salesforce
• Virtual Machines • Disk • Networks • Firewalls	• Containers • Volumes • Ips & Ports • Load Balancers	• Web/WAS • Framework • Apps • Routes	• Actions • Triggers • Gateways • API	• Whatever • You • Want • (to pay for)

Low Level — Abstraction — High Level

Flexibility — Velocity

01 Cloud Model

» Model별 관리 범위와 예시



01 Cloud Model

» Model별 관리 범위와 예시

01 Cloud Model

» Model별 관리 범위와 예시

YOUR OWN CAR
On-premises solution

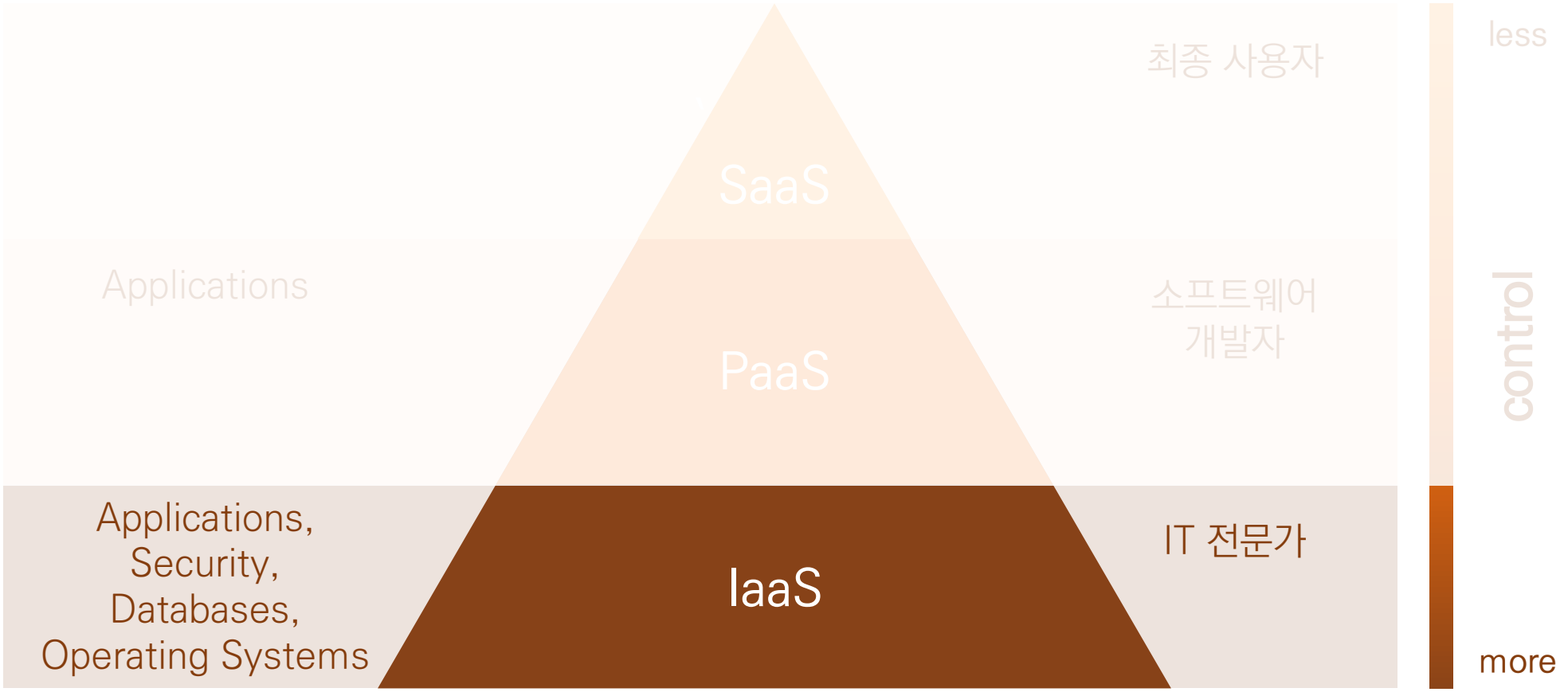
LEASED CAR
IaaS

TAXI
PaaS

BUS
SaaS

출처: RubyGarage

02 IaaS


Customer Managed
User
최종 사용자
SaaS
Applications
소프트웨어
개발자
PaaS
Applications,
Security,
Databases,
Operating Systems
IaaS
IT 전문가
less
control
more

02 IaaS

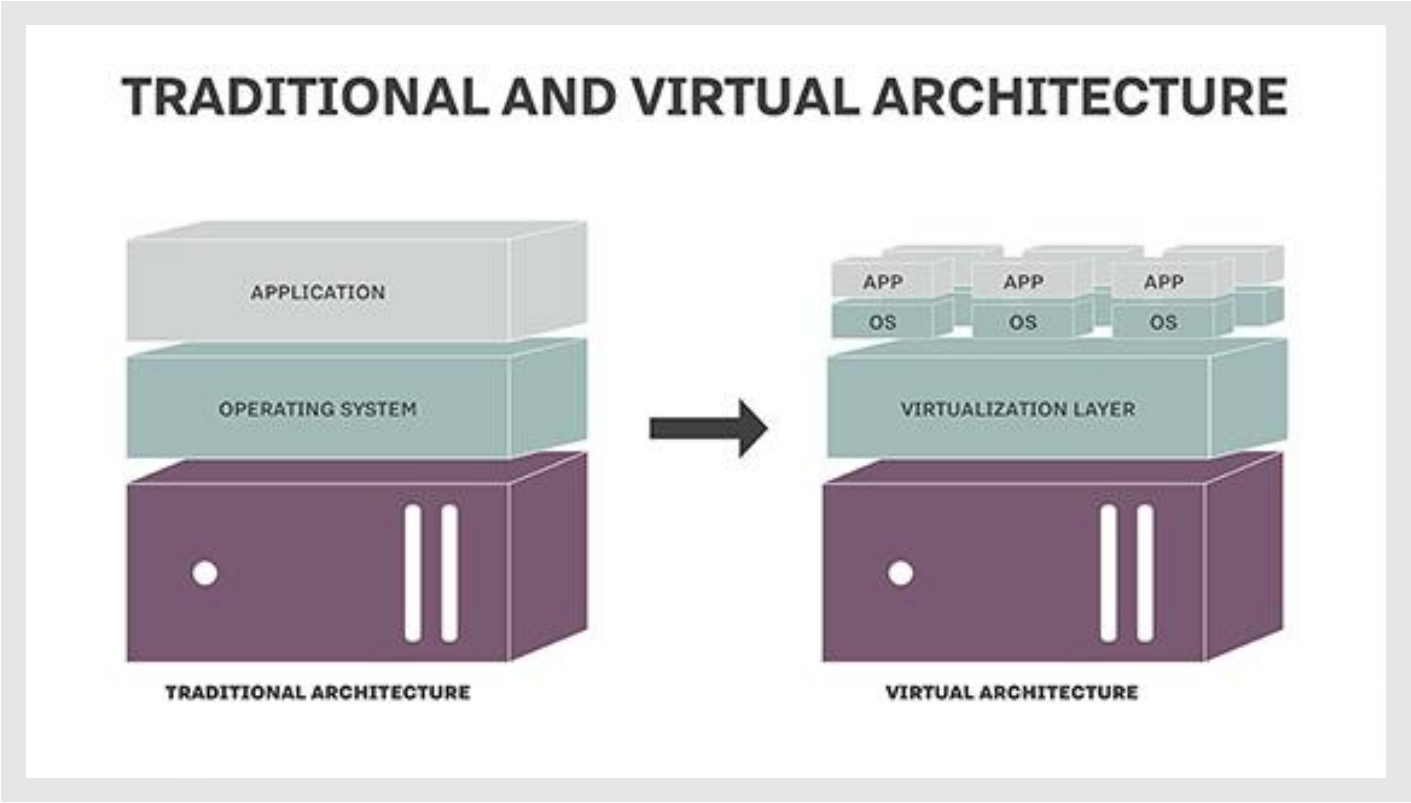
IaaS란

Infrastructure as a Service

CPU나 하드웨어 등의 컴퓨팅
리소스(자원)를 네트워크를 통해
서비스로 제공하는 모델

02 IaaS

» 특징

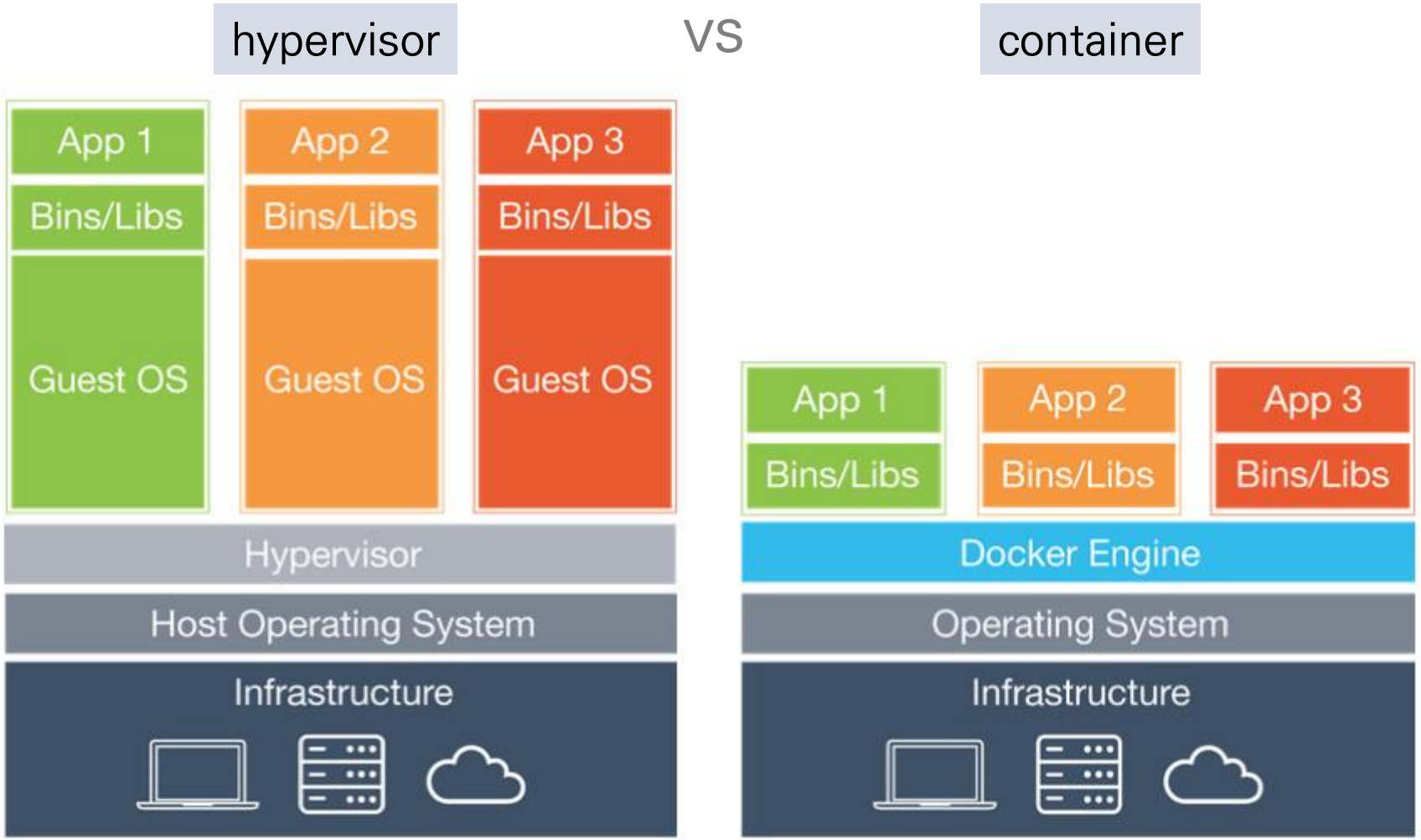


물리적 리소스를 가상화 하여 **유연한 Infrastructure**을 제공

Source : http://whatis.techtarget.com/definition/virtualization-architecture

02 IaaS

» 가상화 유형



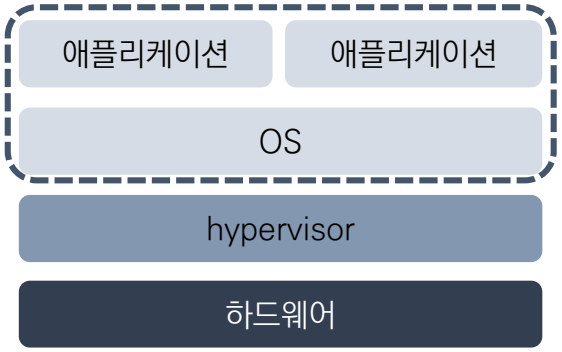
Source : https://www.blackmagicboxes.com/?page_id=64

02 IaaS

» 가상화 유형

hypervisor

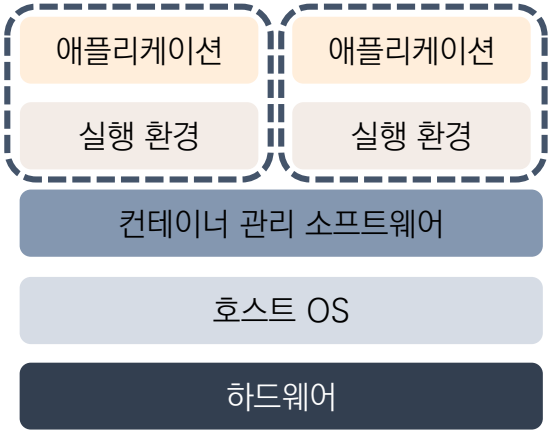
OS 환경을 통째로 가상화 함



VS

container

하나의 호스트 OS에서 멀티 OS 환경을 구현함



장점	• 가상 서버마다 OS를 선택할 수 있음 • 가상 서버들이 완전히 분리되어 있음	• 하나의 호스트 OS에서 여러 개의 OS를 동시에 이용할 수 있음 • 다른 컨테이너로의 복제성과 이식성이 뛰어남
단점	• 가상 서버마다 OS가 필요하므로 하드웨어 리소스의 소비량이 많음 • 가상 서버의 부팅에 시간이 걸림	• 운영체제의 커널을 공유하므로, 각 운영체제의 이미지는 각 운영체제에서만 실행 가능함 • 하나의 호스트 OS를 공유하기 때문에 컨테이너 하나가 사이버 공격을 받으면 다른 컨테이너가 위험에 노출될 가능성이 있음

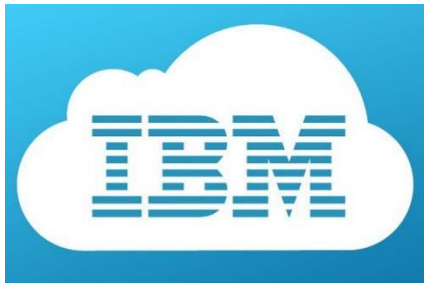
출처: 그림으로 배우는 클라우드

02 IaaS

» 제공 유형

Public IaaS provider VS Project

Microsoft Azure



Google Cloud

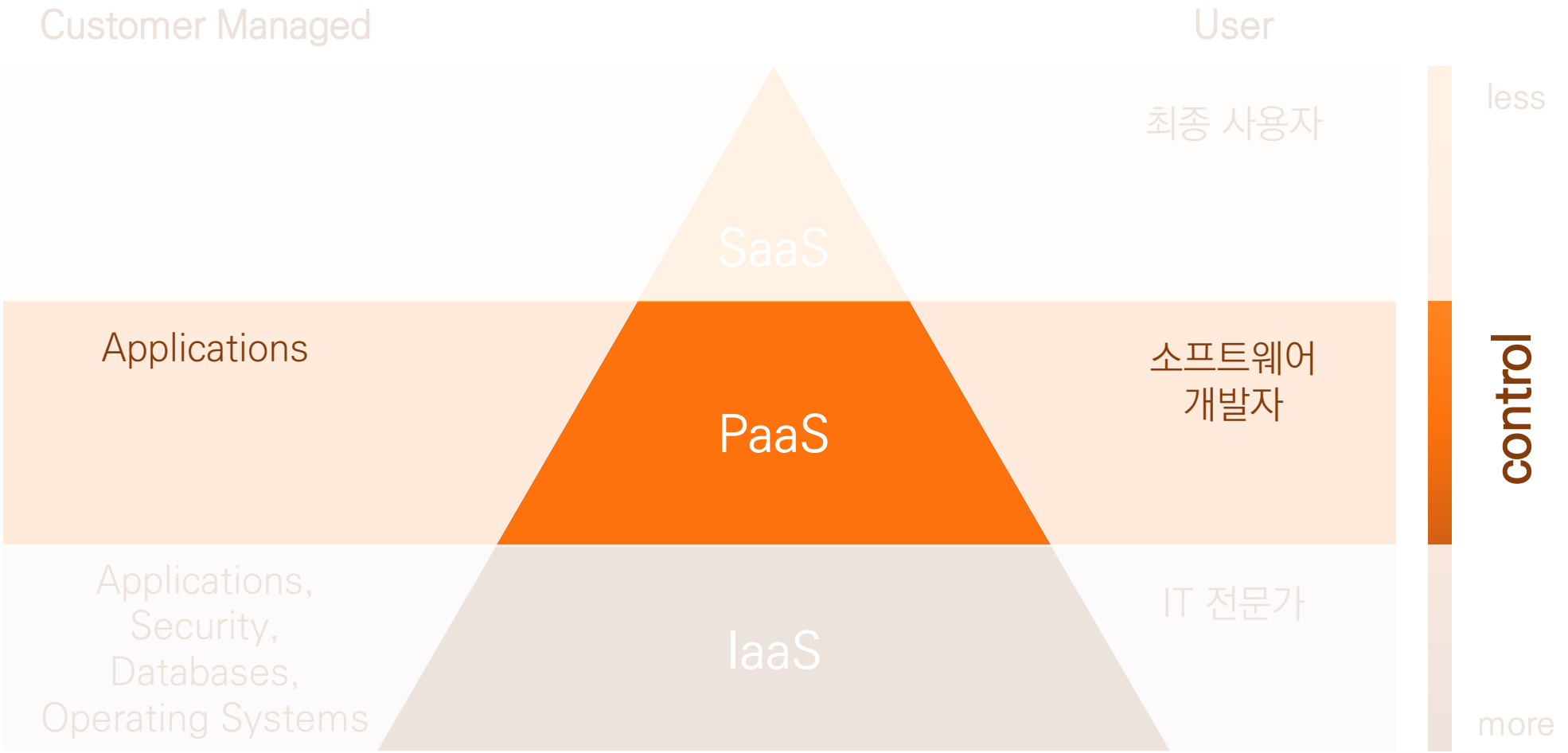
OpenNebula.org

apachecloudstack™

openstack®

EUCALYPTUS

03 PaaS


Customer Managed
User
최종 사용자
SaaS
Applications
PaaS
소프트웨어
개발자
Applications,
Security,
Databases,
Operating Systems
IaaS
IT 전문가
less
control
more

03 PaaS

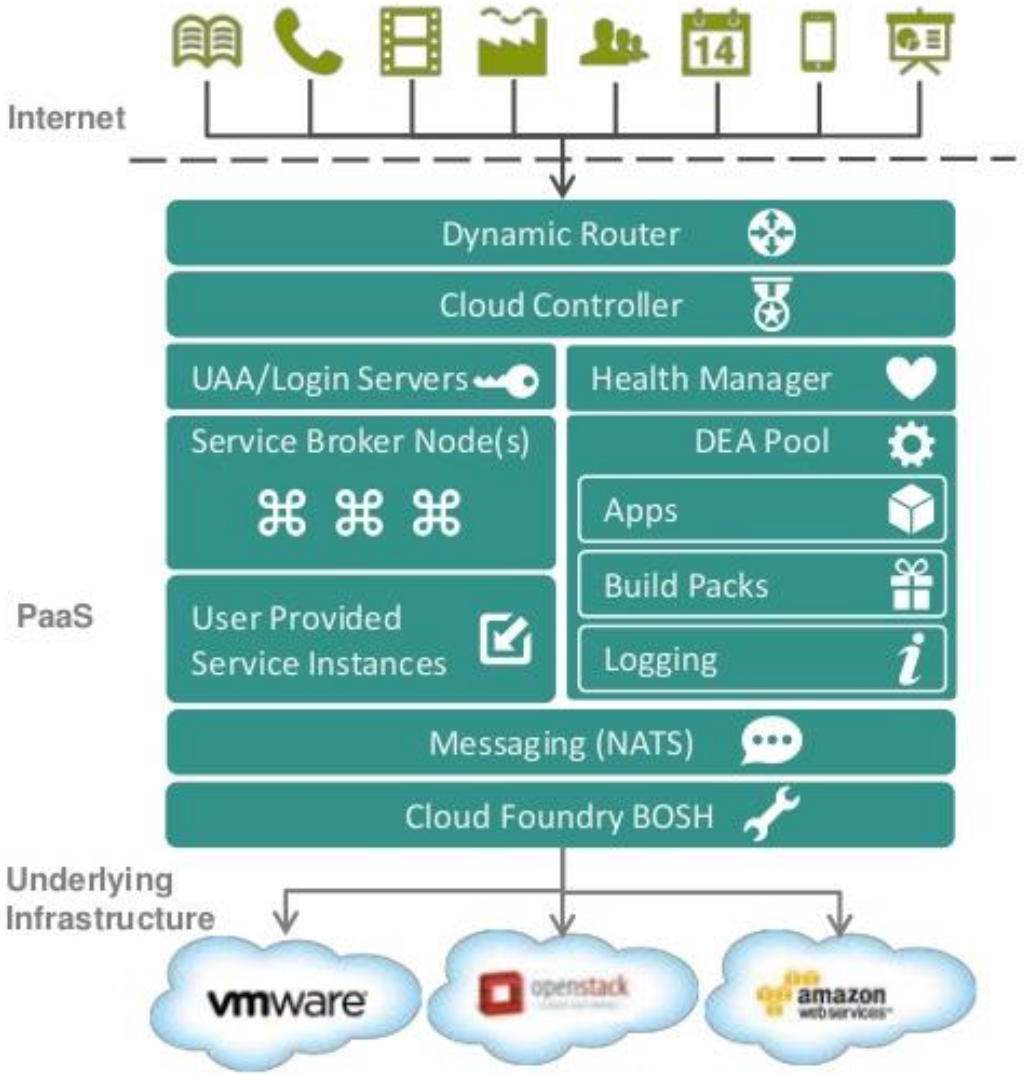
PaaS란

Platform as a Service

기업의 애플리케이션 실행 환경 및
애플리케이션 개발 환경을 서비스로써
제공하는 모델

03 PaaS

» 특징


Internet
Dynamic Router
Cloud Controller
UAA/Login Servers
Health Manager
Service Broker Node(s)
DEA Pool
Apps
Build Packs
Logging
PaaS
User Provided
Service Instances
Messaging (NATS)
Cloud Foundry BOSH
Underlying
Infrastructure
vmware
openstack
amazon

03 PaaS

» 필요성

"우리가 배운 것 중 가장 큰 한 가지는

우리가 얼마나 잘 엔지니어링 했든,

빌드를 했든, 배포를 했든 또는 교육을 잘 했다 해도

좀 더 빠르게 시장에 선보이지 않았다면

여지없이 시장은 이미 바뀌어 있을 것이라는 점*입니다.*

이것은 단지 조금 늦었다고 괜찮아 할 일이 아닌거죠."

James McGlennon
Liberty Mutual Insurance Group 최고 경영 부사장 겸 CIO

03 PaaS

» 필요성

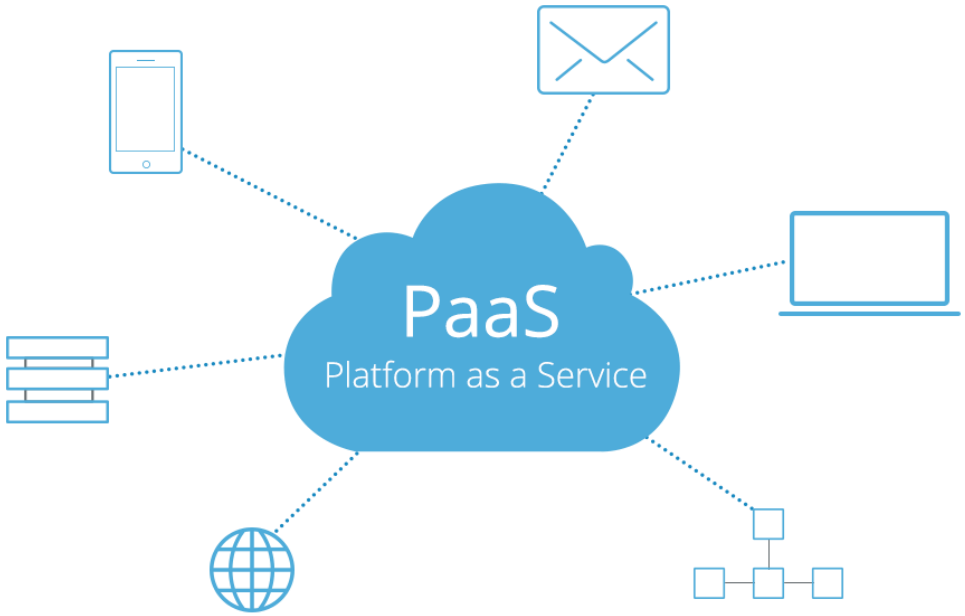
동일한 기능의 application이 넘쳐나는 현 시대

시장의 니즈에
빠르게 대응할 수 있는 능력이
곧 경쟁력입니다.

03 PaaS

» 장점

애플리케이션 실행 환경이나 데이터베이스 등이 미리 마련되어
단기간에 애플리케이션을 개발하여 서비스를 제공할 수 있음



03 PaaS

» 장점

애플리케이션 개발, 실행, 관리 할 수 있게 하는 **플랫폼을 제공**

SaaS의 개념을 개발 플랫폼에 확장한 방식

개발을 위한 플랫폼 구축 필요 없이 **웹에서 쉽게 빌려 쓸 수 있는 모델**

개발자는 개발에만 집중, **애플리케이션이 동작하는 주변 환경은 가져다 쓰는 구조**

개발에서 배포까지 라이프사이클이 짧아 **DevOps 문화를 적용하기 용이함**

03 PaaS

» 종류

PUBLIC

"서비스 제공업체가 구축한 서버, 스토리지 등의 IT 인프라를 기업들이 사용료를 내고 이용하는 방식"

HYBRIDE

"Public 클라우드와 Private 클라우드를 동시에 제공하고 양쪽 장점만 선택해 사용 가능한 클라우드 서비스"

PRIVATE

"기업 자체적으로 데이터센터 안에 클라우드 환경을 구축해 사용하는 방식"

출처: Microsoft Azure

03 PaaS

» 종류

PUBLIC

"서비스 제공업체가 구축한 서버, 스토리지 등의 IT 인프라를 기업들이 사용료를 내고 이용하는 방식"

비용 절감

하드웨어나 소프트웨어를
구매하지 않아도 되며,
사용한 서비스에 대해서만 지불

유지 관리 하지 않음

서비스 공급자가 유지 관리

높은 안정성

광대한 서버 네트워크를 통해
실패를 방지함

무제한에 가까운 확장성

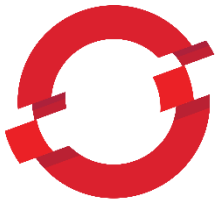
비즈니스 요구 사항을
만족시키도록
주문형 리소스 사용가능

03 PaaS

» 종류

HYBRIDE

"Public 클라우드와 Private 클라우드를 동시에 제공하고 양쪽 장점만 선택해 사용 가능한 클라우드 서비스"



OPENSHIFT

제어

조직이 중요한 데이터의 사설 **인프라를 유지 관리** 할 수 있음

유연성

필요할 때 공용 클라우드에서 **추가 리소스를 활용**할 수 있음

비용 효율성

공용 클라우드에 맞게 **규모 조정 가능**하여 **필요할 때만 추가 컴퓨팅 기능에 대해 지불**

용이성

워크로드를 점진적으로 마이그레이션 할 수 있으므로 **부담없이 클라우드 전환 가능**

03 PaaS

» 종류

PRIVATE

"기업 자체적으로 데이터센터 안에 클라우드 환경을 구축해 사용하는 방식"

DEIS

AppScale SYSTEMS

apprenda

Cloudify

유연성 향상

특정 비즈니스 요구 사항을 만족시키기 위해 **클라우드 환경을 사용자 지정할 수 있음**

보안 강화

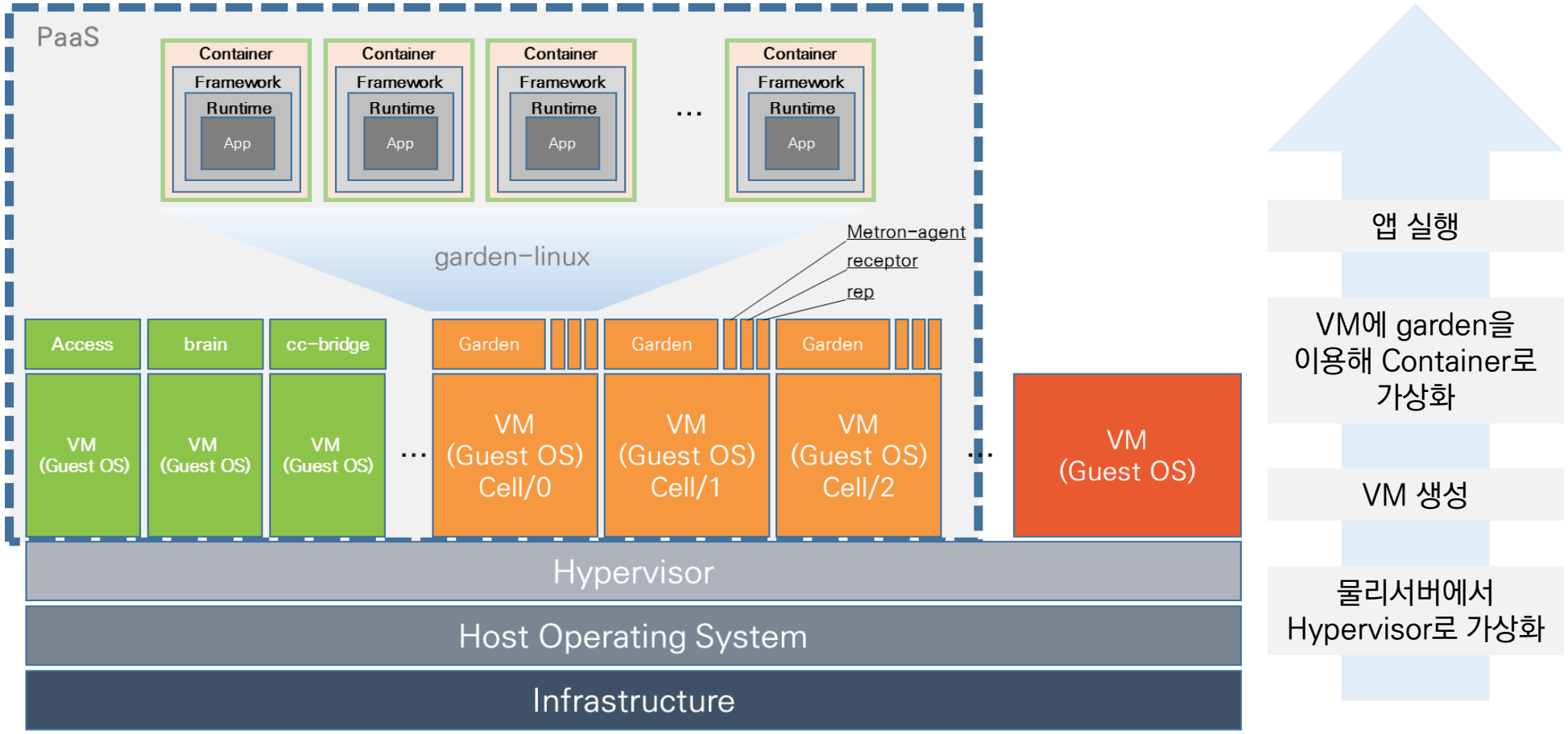
다른 사용자와 리소스를 공유하지 않으므로, **제한과 보안 수준을 강화할 수 있음**

높은 확장성

사설 클라우드에서 여전히 **공용 클라우드의 확장성과 효율성을 제공할 수 있음**

03 PaaS

» 동작 원리

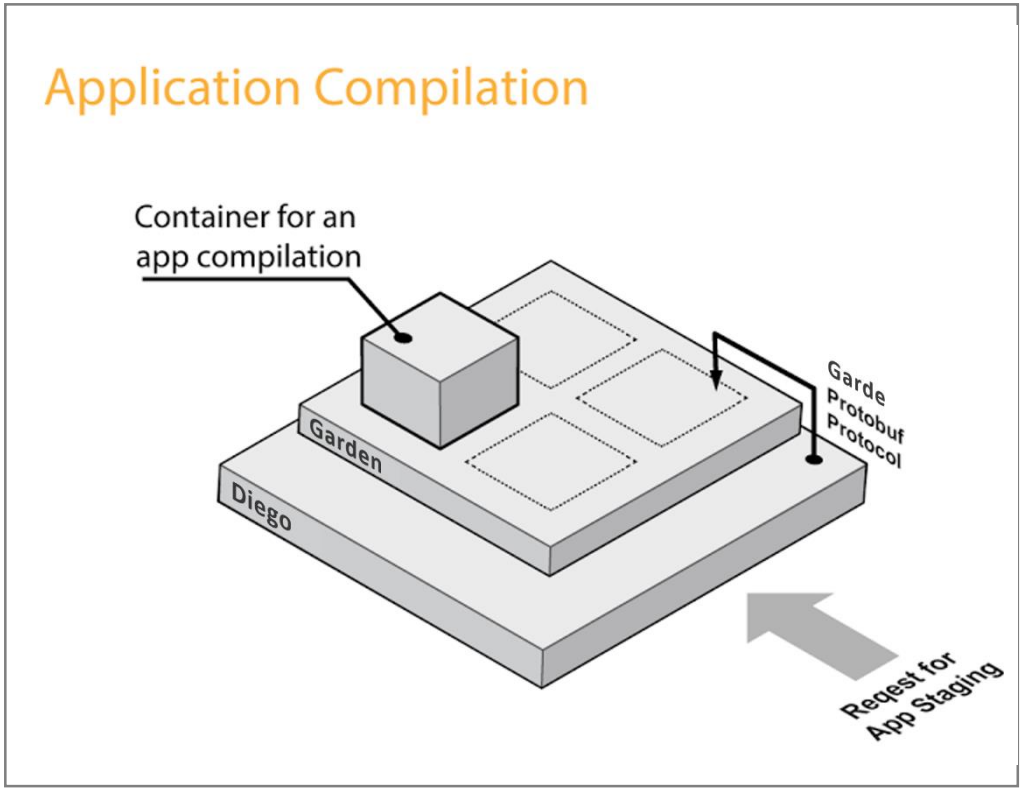


IaaS 자원 위에 **여러 인스턴스들의 유기적인 조합**으로 PaaS 동작함

03 PaaS

» 동작원리

Application이 실제 배포 시 PaaS의 구성요소 중 Diego cell에서의 모습



배포 명령 실행

↓

Diego 컨테이너 생성

↓

Application 코드 배치

↓

적합한 Buildpack 찾기

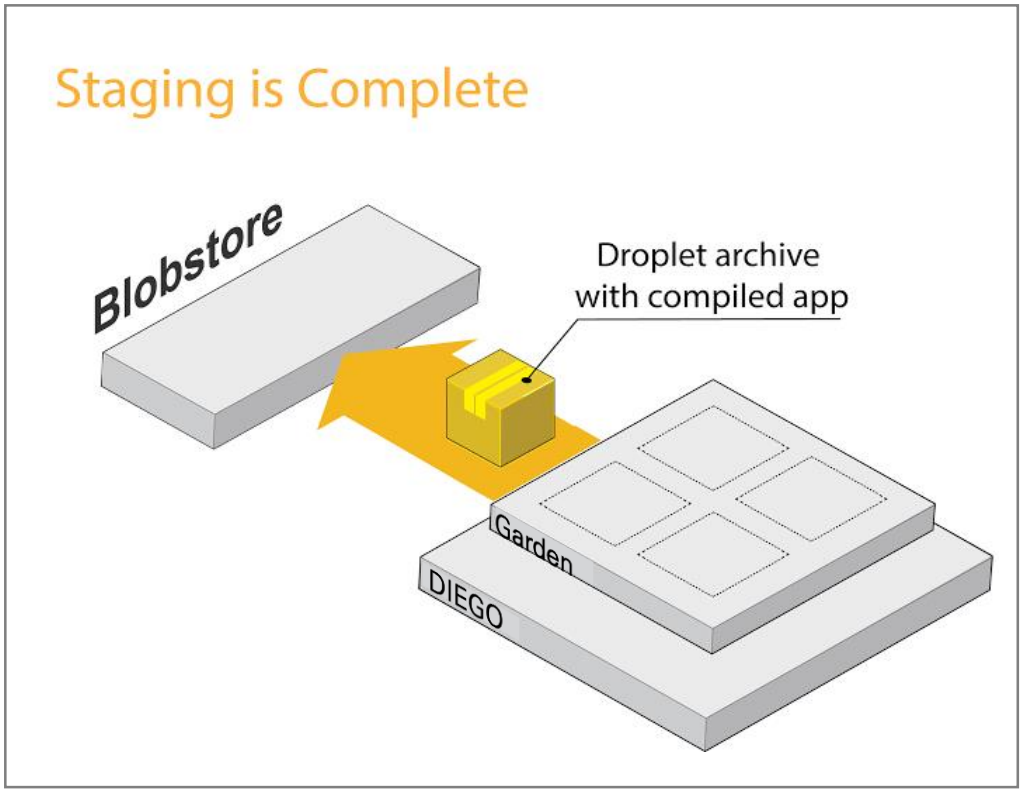
↓

컴파일 진행

03 PaaS

» 동작원리

Application이 실제 배포 시 PaaS의 구성요소 중 Diego cell에서의 모습



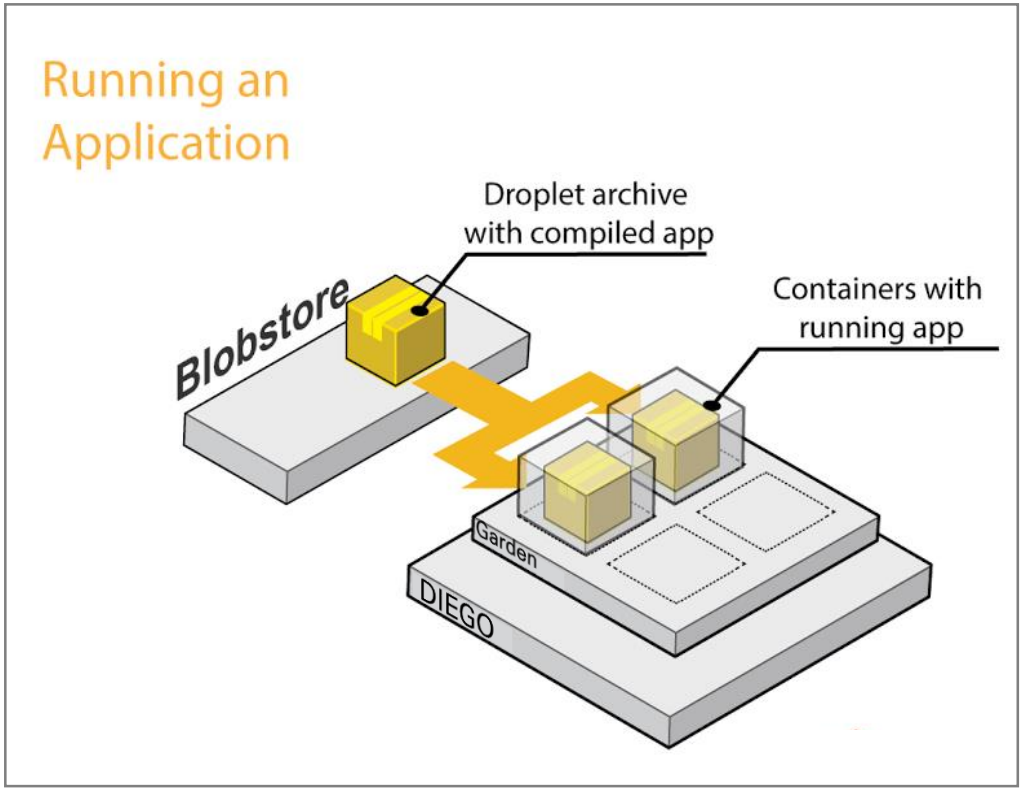
Droplet Archive
(Application이 동작할 수 있는
부가 요소를 모두 갖춘 형태)

Blobstore에 저장

03 PaaS

» 동작원리

Application이 실제 배포 시 PaaS의 구성요소 중 Diego cell에서의 모습



Application Instance가 실행될 새로운 컨테이너 생성 요청

↓

release 스크립트 실행

↓

Application 동작

03 PaaS

» 주요 용어

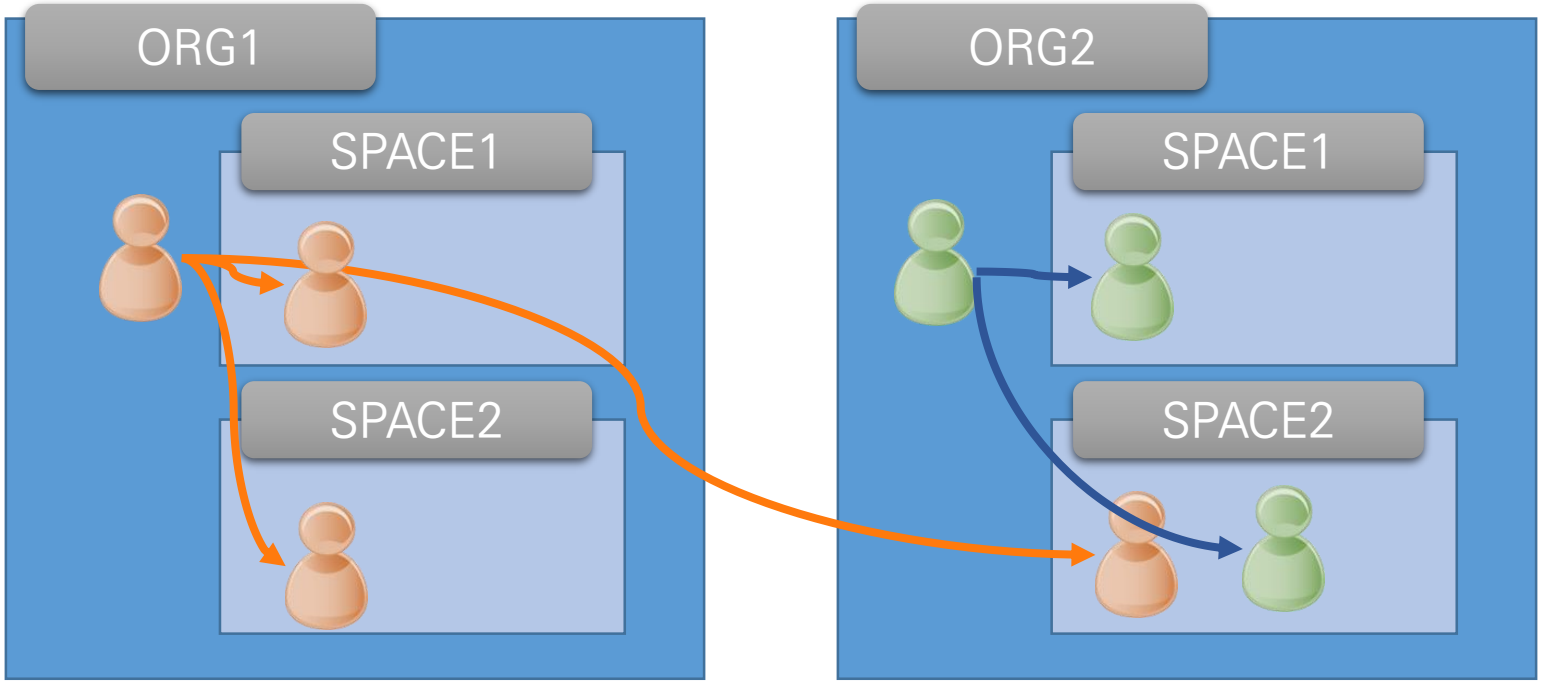
BuildPack 언어별 프로그램이 동작할 수 있도록 언어 프레임워크등의 기술 지원

Name	Supported Languages, Frame works and Technologies
Binary	n/a
Go	Go
Java	Grails, Play, Spring, or any other JVM-based language or framework
.NET Core	.NET Core
Node.js	Node or JavaScript
PHP	Cake, Symfony, Zend, Nginx or HTTPD
Python	Django or Flask
Ruby	Ruby, JRuby, Rack, Rails, or Sinatra
Staticfile	HTML, CSS, JavaScript, or Nginx
Jeus	Jeus was for tibero DB

03 PaaS

» 주요 용어

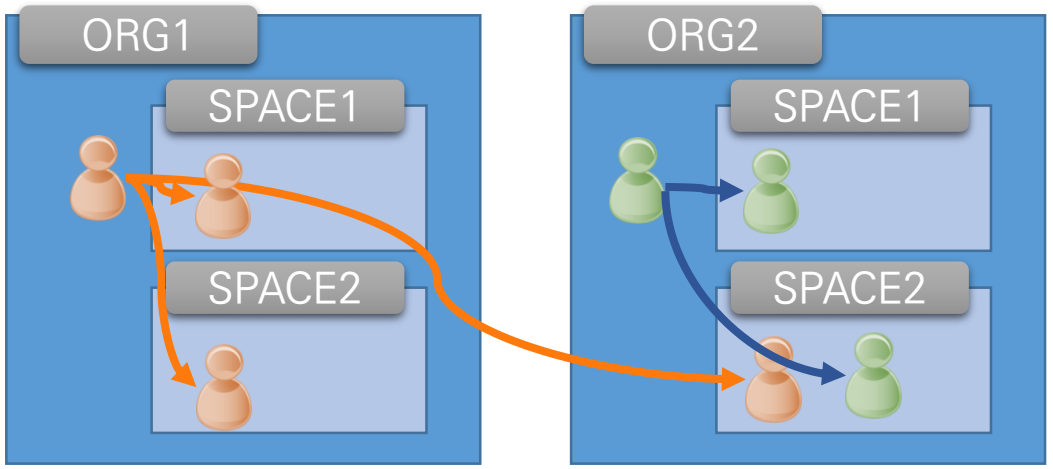
Org & Space 효과적으로 업무를 구분할 수 있는 논리적인 그룹 기능



03 PaaS

» 주요 용어

Org & Space 효과적으로 업무를 구분할 수 있는 논리적인 그룹 기능



ORG

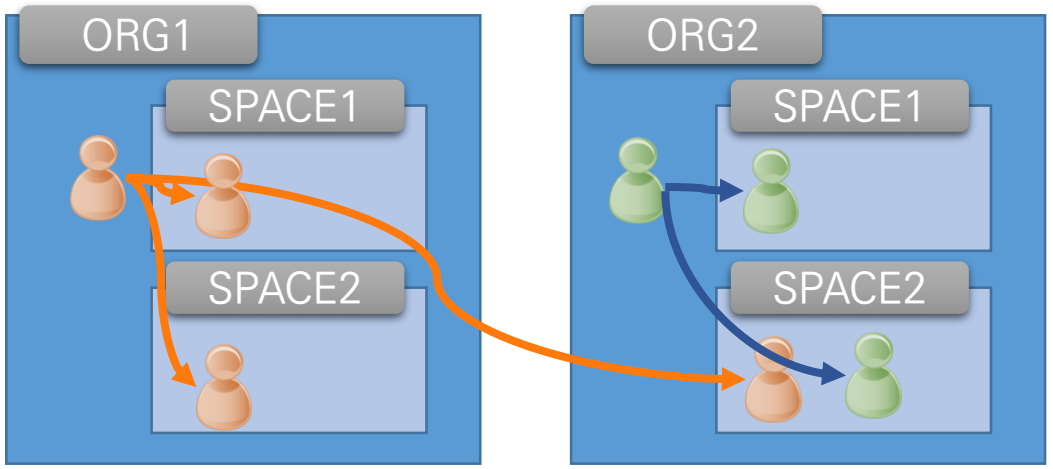
- 개인 또는 여러 공동 작업자가 소유하고 사용할 수 있는 개발 계정
- 각 계정은 ORG에 롤을 부여 받을 수 있음

Org Manager, Org Auditor,
Org Billing Manager, Org User

03 PaaS

» 주요 용어

Org & Space 효과적으로 업무를 구분할 수 있는 논리적인 그룹 기능



SPACE

- 모든 애플리케이션 및 서비스의 범위가 공간으로 지정

Space Manager,
Space Developer,
Space Auditor

Space Developer만이 애플리케이션 실행 및 서비스 연동 할 수 있음

03 PaaS

» 주요 용어

Application & Services

Application

사용자가 제작한 결과물

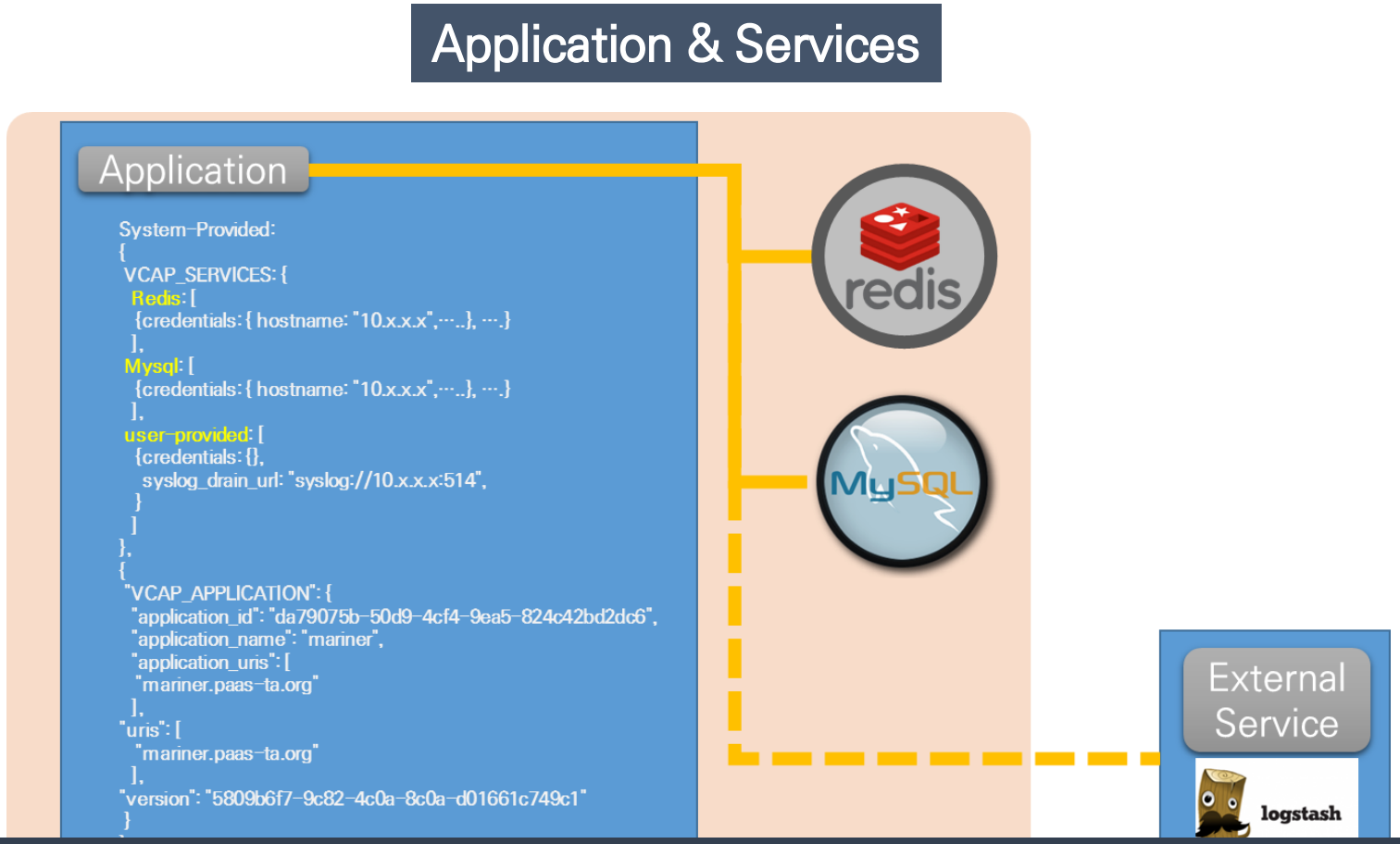
Service

Application동작에 필요한 부가 서비스

Market Place에서 준비되어있는 **서비스를 골라서 사용**

03 PaaS

» 주요 용어



연결된 서비스는 **해당 Application환경 변수**로 확인가능

03 PaaS

» 주요 용어

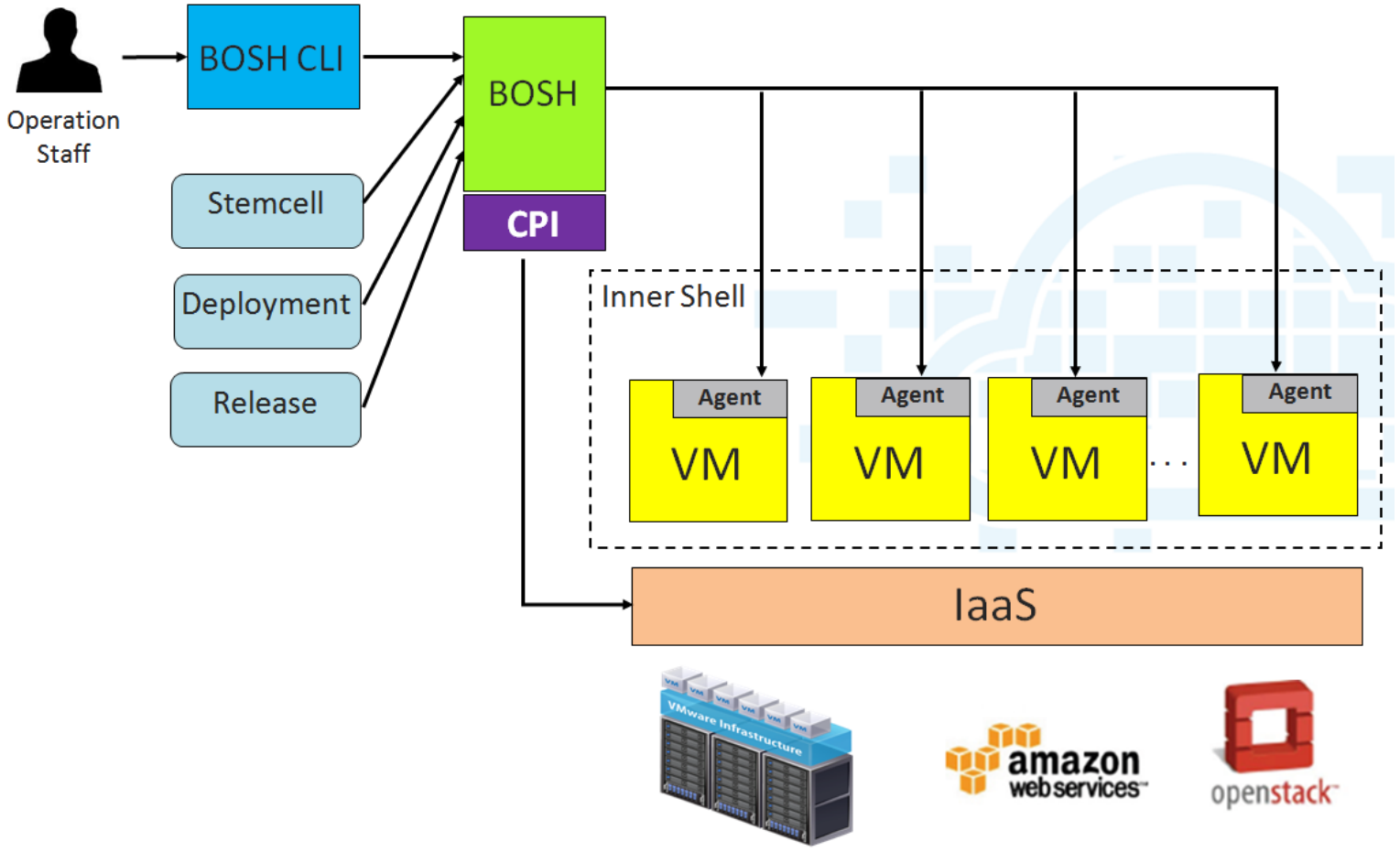
Service Packs 제공 가능한 서비스들

Service	Plan	description
glusterfs	glusterfs-5Mb, glusterfs-100Mb, glusterfs-1000Mb	데이터 저장소
rabbitMQ	standard	메시지 큐
ARCUS	arcus-100mb-plan	NoSQL 데이터베이스
redis	shared-vm, dedicated-vm	NoSQL 데이터베이스
Mongo-DB	default-plan	NoSQL 데이터베이스
AltibaseDB	altibase-plan-dedicated-vm	관계형 데이터베이스
CubridDB	utf8, euckr	관계형 데이터베이스
Mysql-DB	Mysql-Plan1-10con, Mysql-Plan2-100con	관계형 데이터베이스
Tibero	Basic	관계형 데이터베이스
pinpoint	Pinpoint_standard	애플리케이션 모니터링(APM)

03 PaaS

» 주요 용어

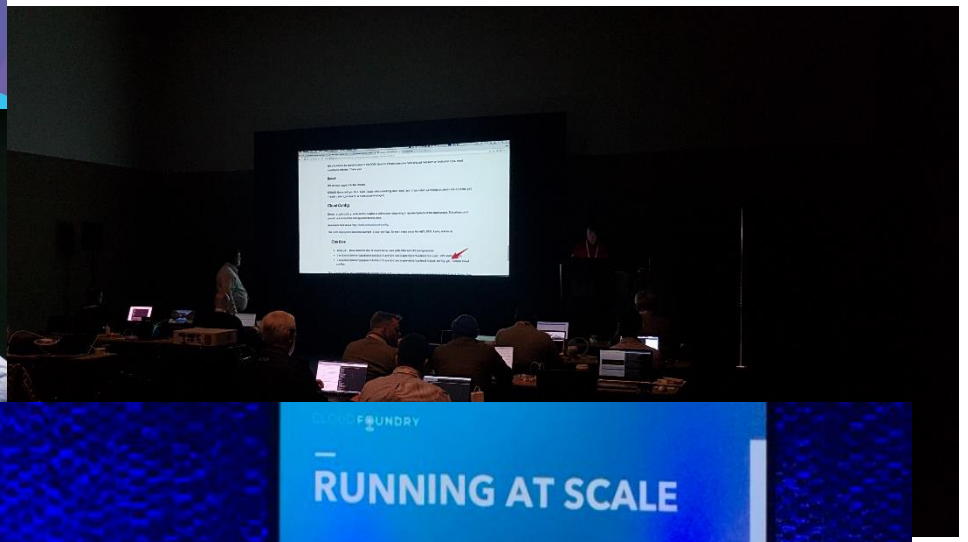
BOSH IaaS환경에서 Stemcell, Deployment, Releas를 이용하여 VM들을 관리



03 PaaS

» 관련 활동

CLOUD FOUNDRY

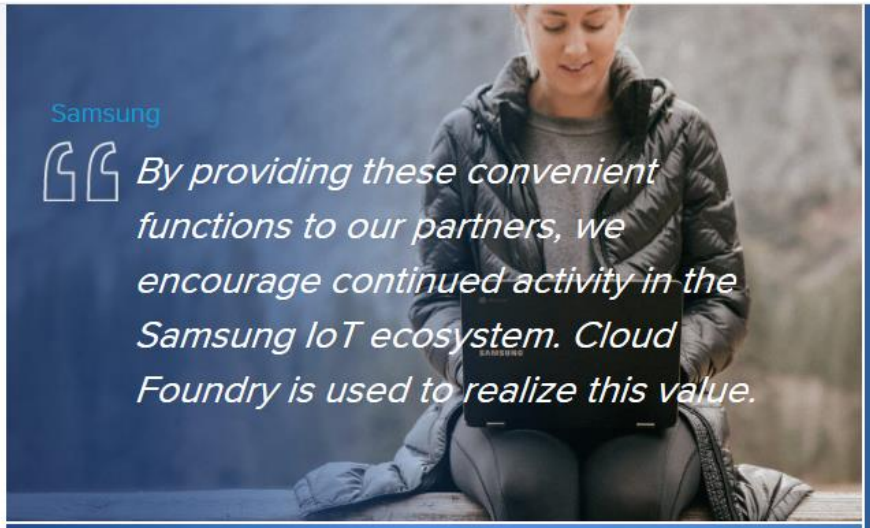


기반이 되는 opensource community의 활발한 활동

03 PaaS

» World wide use cases

CLOUD FOUNDRY Discover Use Ecosystem Community Events About

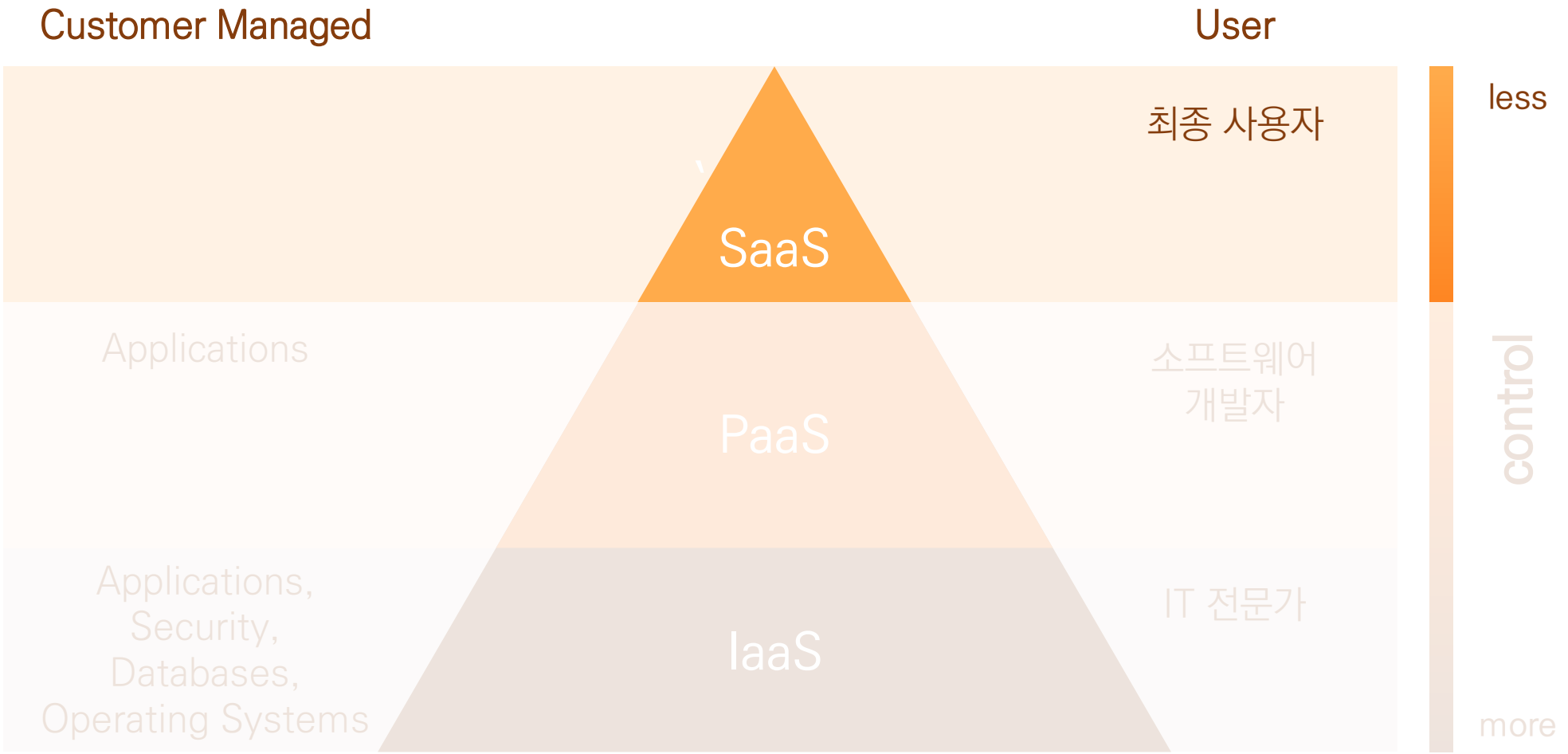


03 PaaS

» World wide use cases

주최	내용
SAMSUNG 삼성	IoT 서비스와 장치 간에 자동화 개발을 용이하게 하는 개발 도구
네덜란드 정부	대 시민 인프라 서비스 개발 도구
미국 교통부	교통 인프라 서비스 개발 도구
GE	산업인터넷용 애플리케이션을 개발 지원 소프트웨어 세트 predix
BOSCH BOSCH	IoT 애플리케이션 개발 도구 Bosch IoT Suite
verizon Verizon	네트워크 애플리케이션 개발 도구
HONDA Honda	IBM Bluemix를 통한 커넥티드 카 서비스
PaaS-TA OpenPaaS 한국	클라우드 파운드리를 통한 IT 서비스 제공 표준화(PaaS-TA)

04 SaaS


Customer Managed
User
최종 사용자
SaaS
Applications
소프트웨어
개발자
PaaS
Applications,
Security,
Databases,
Operating Systems
IT 전문가
IaaS
less
control
more

04 SaaS

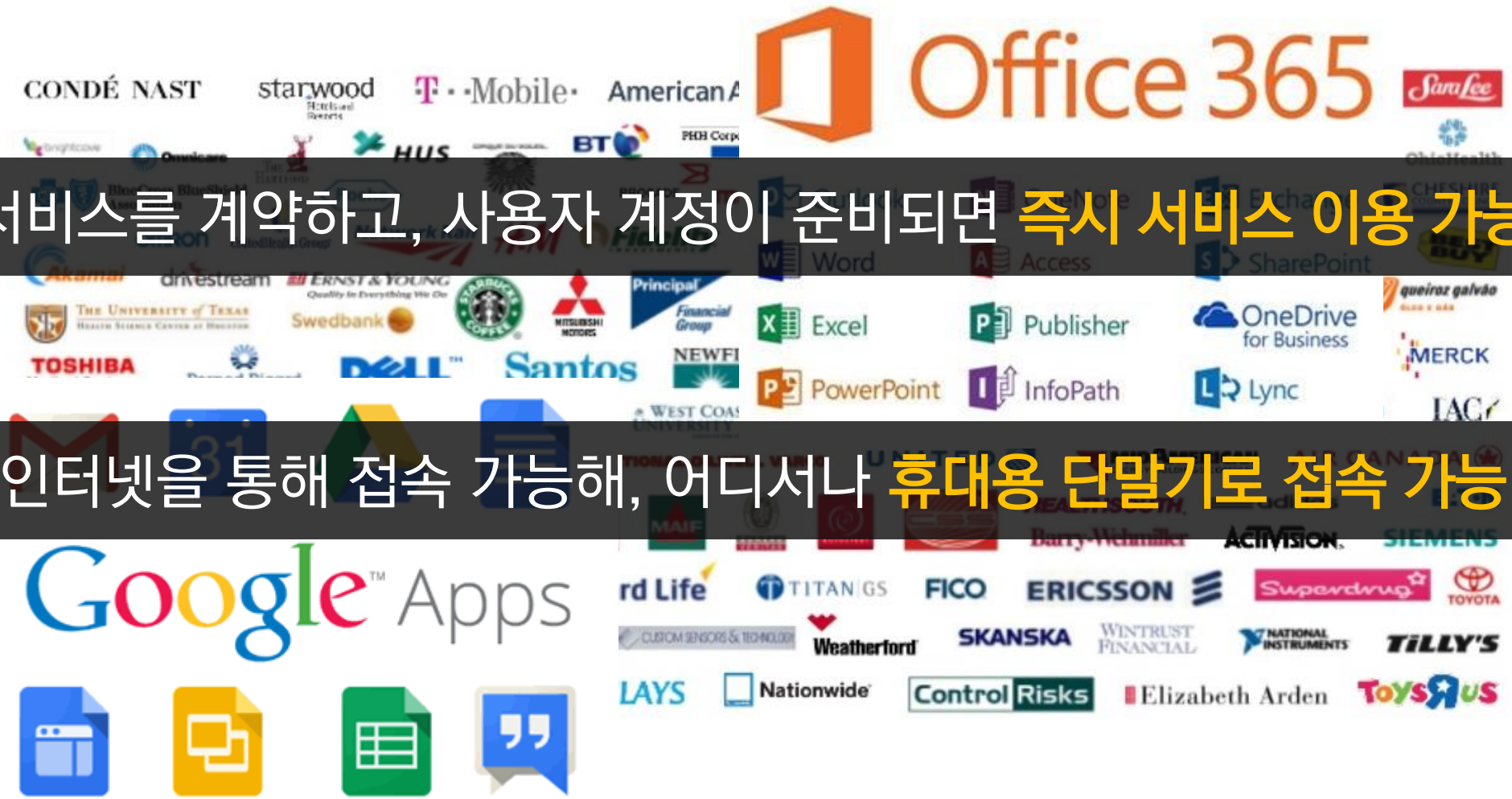
SaaS란

Software as a Service

주로 업무에서 사용하는 소프트웨어의 기능을 인터넷 등의 네트워크를 통해 필요한 만큼 서비스로 이용할 수 있도록 제공하는 형태

04 SaaS

» 특징

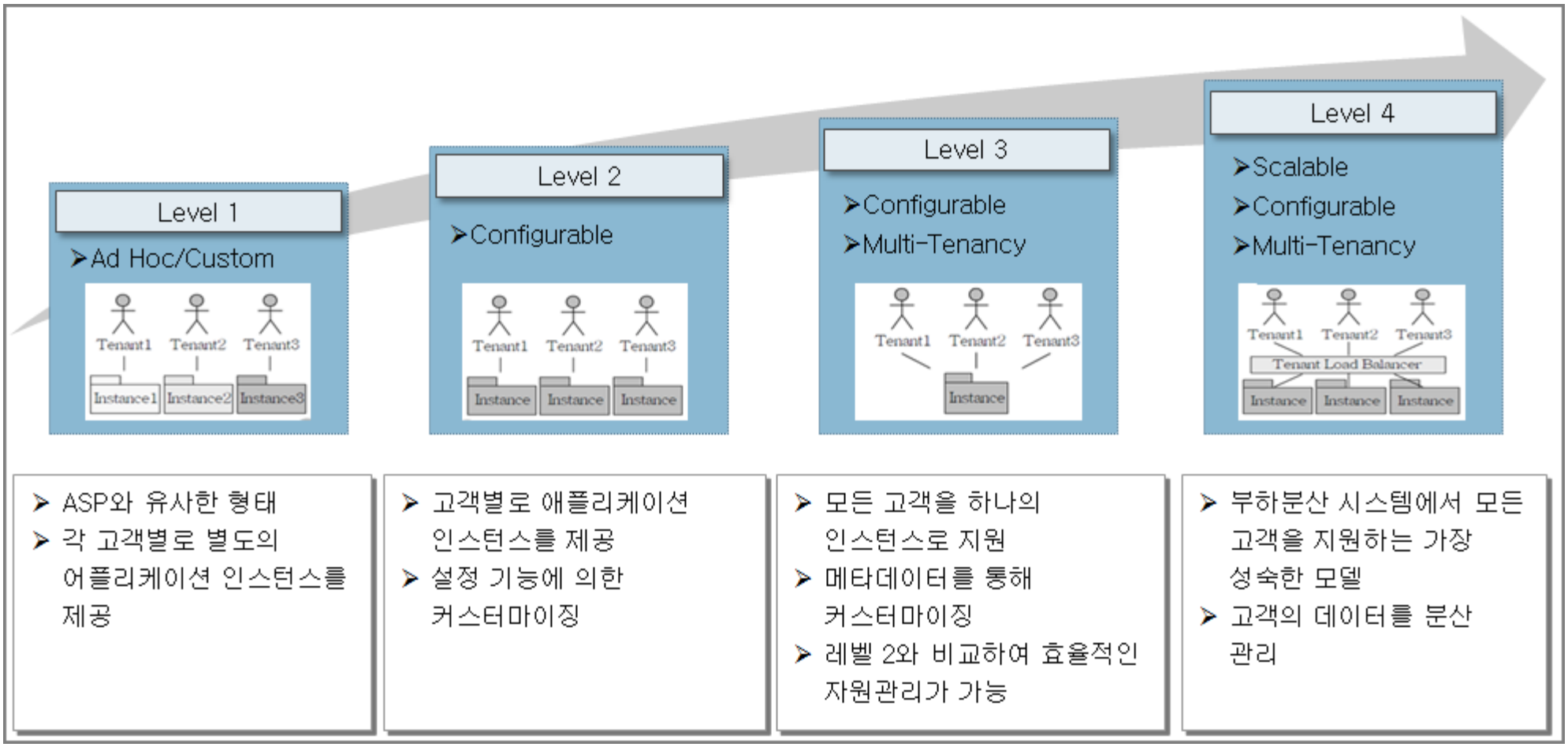


서비스를 계약하고, 사용자 계정이 준비되면 **즉시 서비스 이용 가능**

인터넷을 통해 접속 가능해, 어디서나 **휴대용 단말기로 접속 가능**

04 SaaS

» 특징



- ASP와 유사한 형태
- 각 고객별로 별도의 어플리케이션 인스턴스를 제공

- 고객별로 애플리케이션 인스턴스를 제공
- 설정 기능에 의한 커스터마이징

- 모든 고객을 하나의 인스턴스로 지원
- 메타데이터를 통해 커스터마이징
- 레벨 2와 비교하여 효율적인 자원관리가 가능

- 부하분산 시스템에서 모든 고객을 지원하는 가장 성숙한 모델
- 고객의 데이터를 분산 관리

성숙된 SaaS는 Multi Tenant, Configuration, Scalability 충족

04 SaaS

» 정의 (IaaS, PaaS 환경 위에서 Software 기능을 서비스로 제공함)

SaaS 개념

- SaaS는 “주문형(on-demand) SW” 또는 “서비스형 SW”라고도 하며, SW 및 관련 데이터를 중앙에서 관리하고 인터넷을 통해 접속하는 사용자에게 SW를 이용 할수 있도록 하는 모델로서, 사용자들은 필요한 기능을 필요한 만큼만 이용하고 요금을 지불하는 형태

SaaS 유형

- 오피스·ERP·CRM 등 전통적인 SW 영역에서 발전하고 있는 기업용 기반 SaaS
- 제조·의료·금융 등 개별 산업의 생산성 향상과 부가가치 창출을 위한 산업융합용 특화SaaS
- 문서저장·협업 등 개인의 편리한 생활을 정보생활용 SaaS

SaaS 기술적 특징

환경 설정 (Configuration)

- 소스코드 레벨의 수정 없이 사용자의 요구사항을 수용 (사용자가 필요한 기능은 설정을 통해 지원)

다중 사용자 지원 (Multi-Tenancy)

- 하나의 어플리케이션을 다수의 사용자(Tenant)가 공유하여 사용하는 다중 소유 아키텍처 지원

확장성 (Scalability)

- 가용성 및 성능 지원을 위해 다수의 인스턴스를 생성하고 사용자의 데이터를 분산 관리하며, 가상화·분산병렬 처리 등을 통해 확장성 있는 서비스를 제공

04 SaaS

» 특징 (사용자가 독립적으로 이용할 수 있는 응용 SW•데이터를 제공하는 클라우드 서비스)

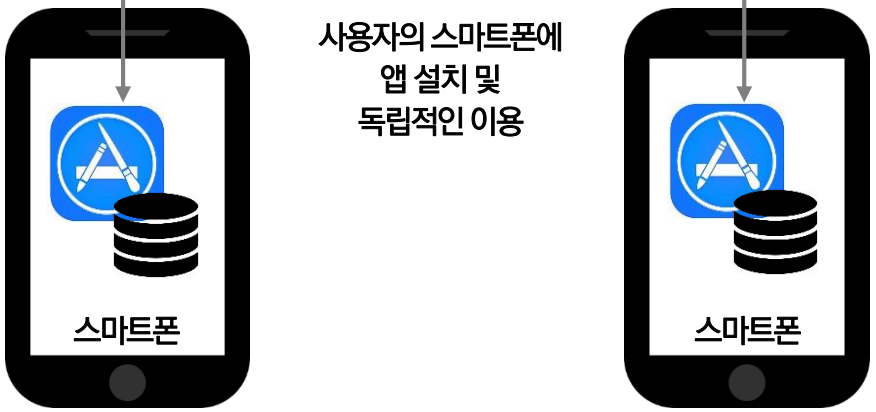
SaaS의 주요 특징

- **테넌트** — 사용자 별로 격리된 환경인 테넌트 제공
  - **Application & Data** — 사용자가 독점적으로 이용하는 어플리케이션과 데이터 제공
  - **보안** — 테넌트 별 보안 정책 적용
  - **가용성** — 테넌트 별 가용성 보장
  - **확장성** — 사용자의 요구에 따라 용량 확대 및 축소
  - **과금** — 사용량 측정 및 과금

04 SaaS

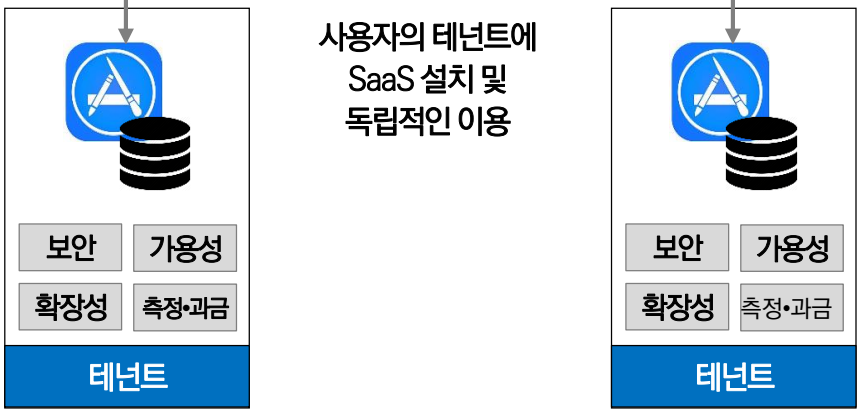
» 특징

스마트폰 앱



VS

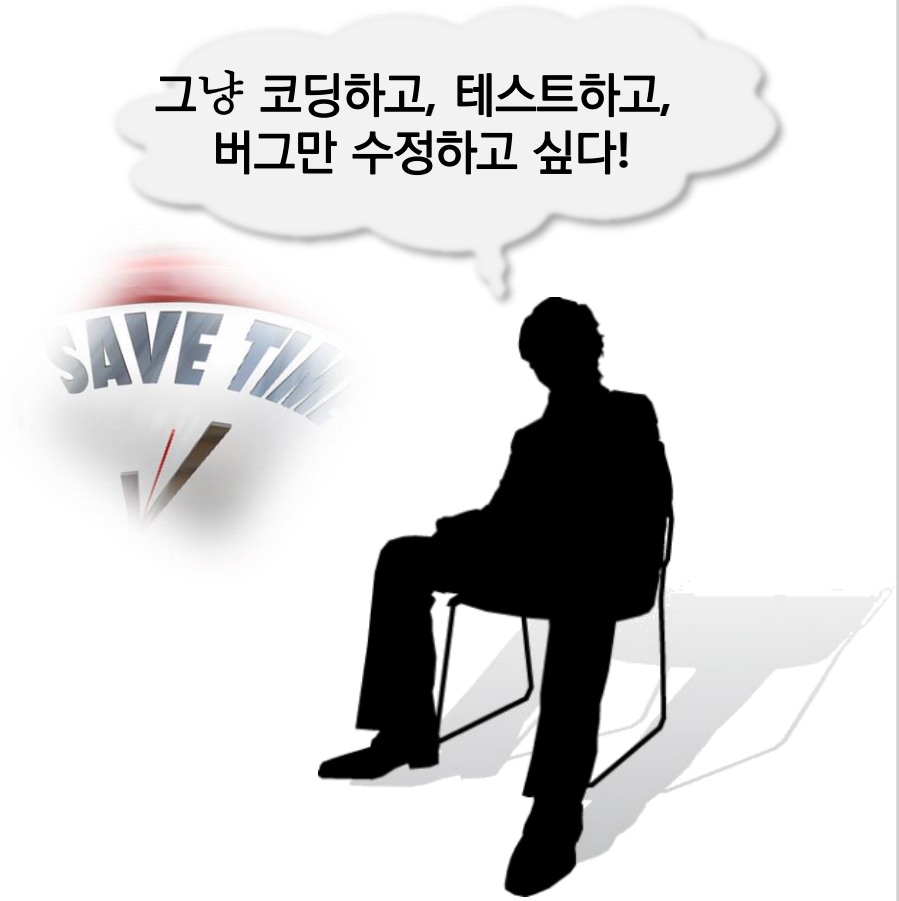
클라우드 SaaS



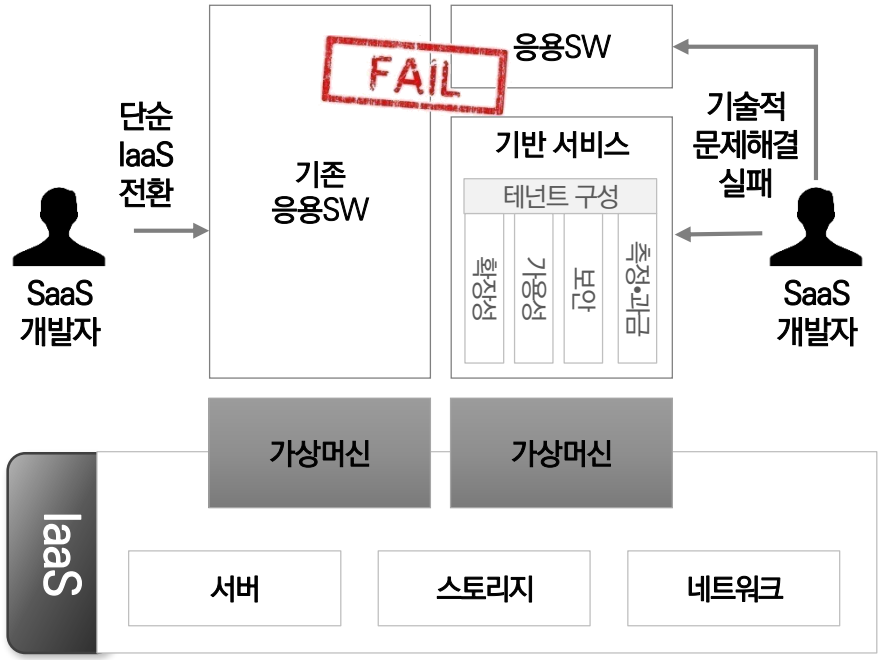
04 SaaS

» 특징

서비스 개발자의 고민



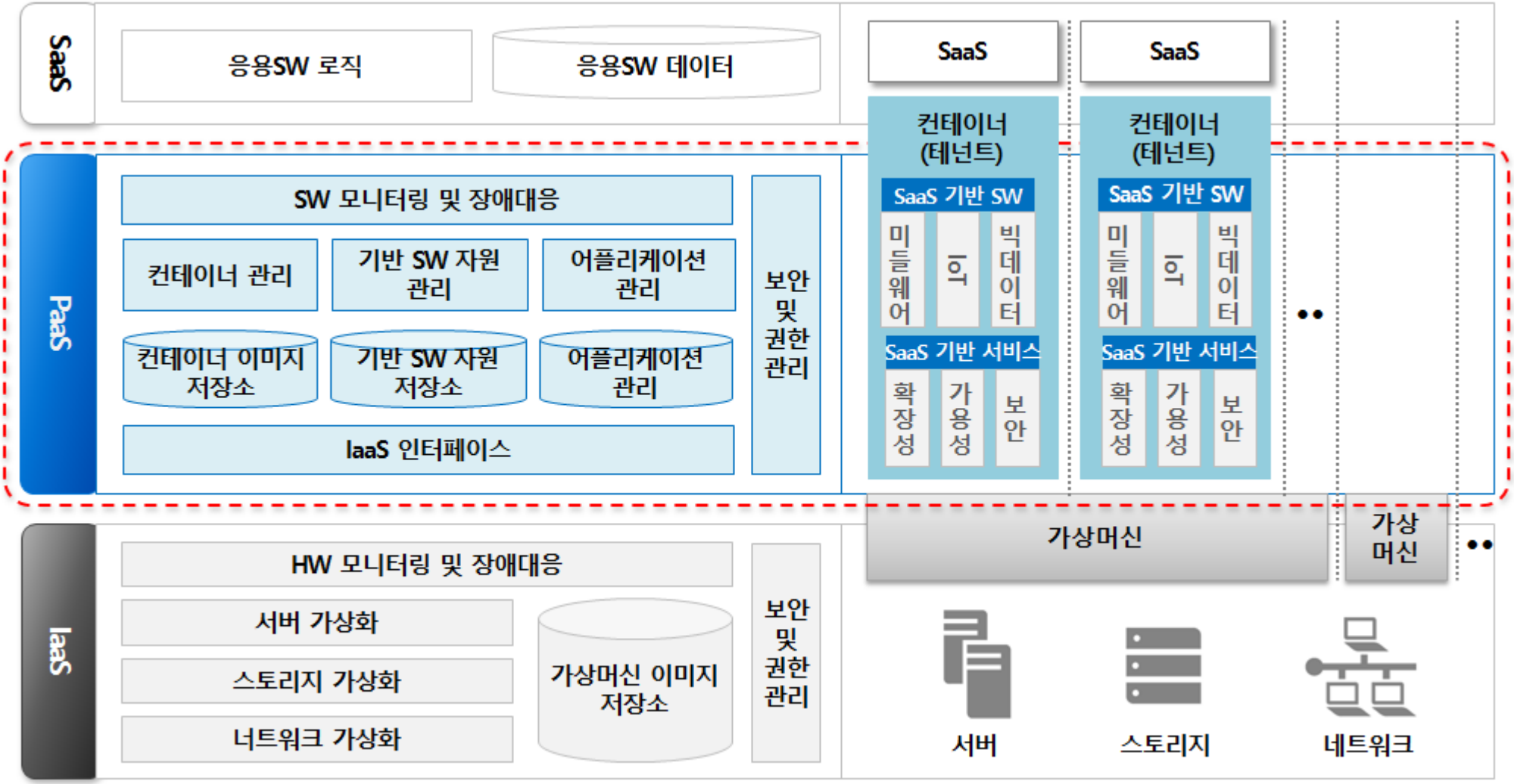
현황 및 문제점



- IaaS를 기반으로 SaaS를 개발하면, 대부분 기존 응용SW를 IaaS 환경으로 설치하는 것으로 그치거나, 기술적인 문제를 고민하지 않아 실패로 끝남

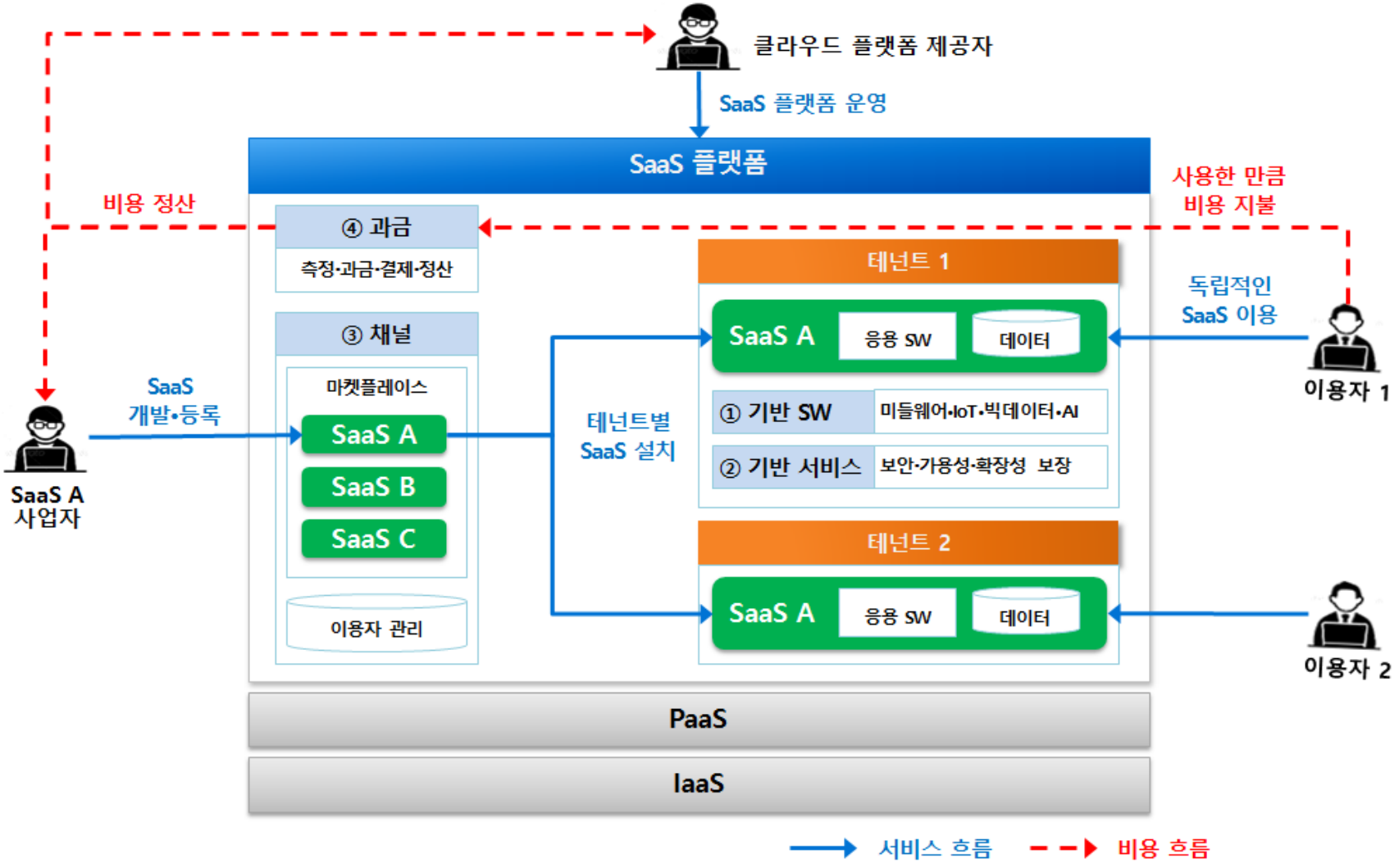
04 SaaS

» 특징 (PaaS는 컨테이너를 기반으로 한 미들웨어 성격의 클라우드 서비스로 SaaS의 개발과 실행을 지원함)



04 SaaS

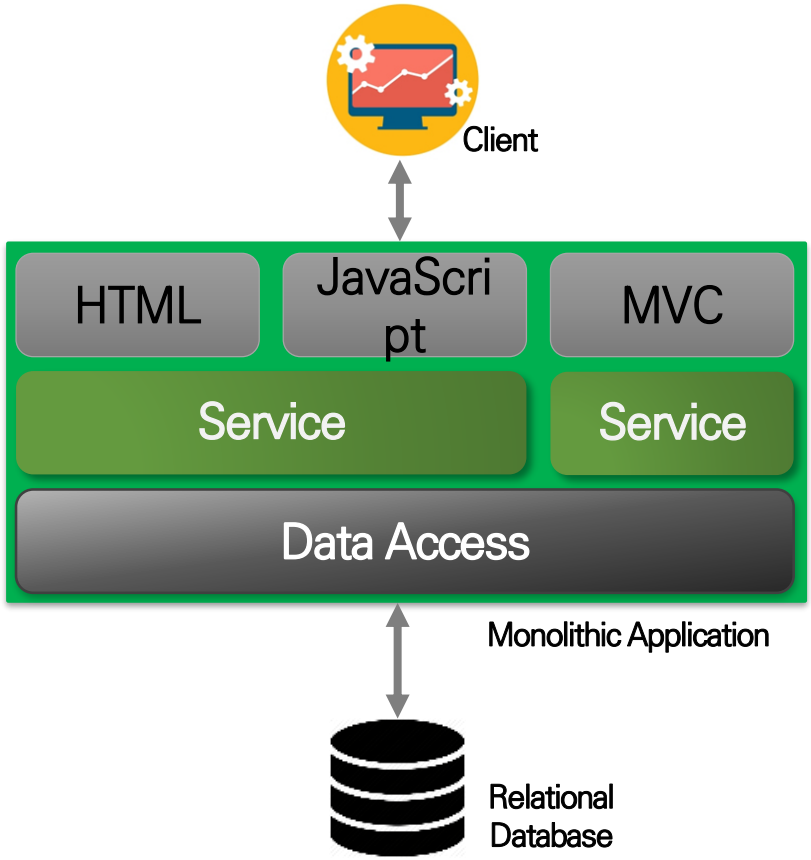
» 특징 (PaaS는 컨테이너를 기반으로 한 미들웨어 성격의 클라우드 서비스로 SaaS의 개발과 실행을 지원함)



04 SaaS

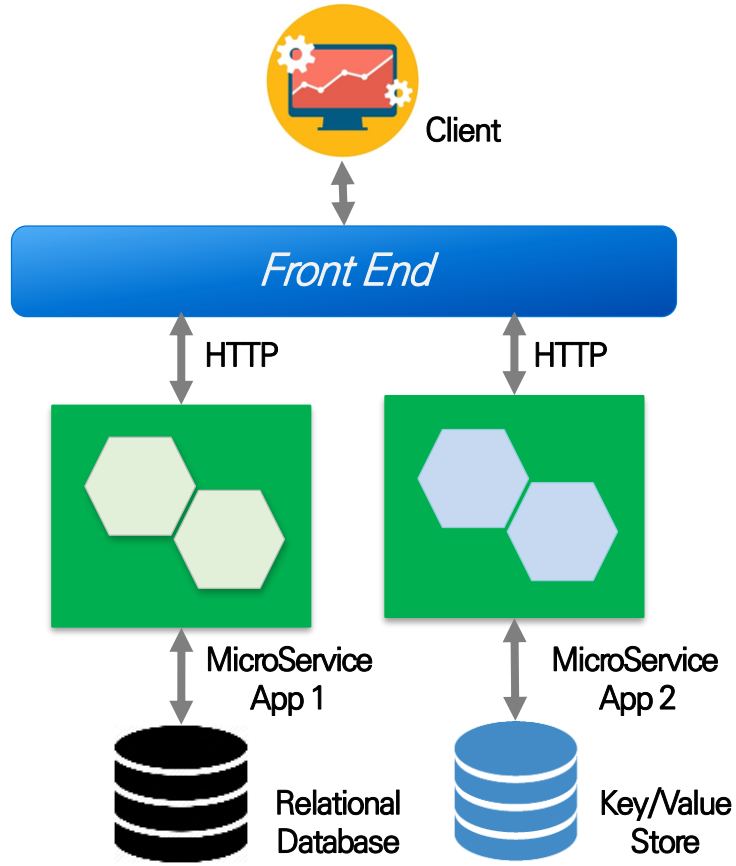
» 특징

전통적인 3-tier Application 구성 예



[Single Process]

Microservices 구성 예



[Multiple Processes]

05 Cloud Native Application

» 개념

Native application (desk-top)

- **특정 Platform이나 Device에서 사용되도록 개발된 Application**
- 해당 플랫폼에 일반적으로 설치된 기타 소프트웨어 및 운영 체제와 상호 작용하고 이를 활용할 수 있음
- 특정 장비를 위한 하드웨어 및 소프트웨어를 사용할 수 있으며, 이는 네이티브 어플리케이션이 GPS나 카메라같은 최신 기술을 모바일 디바이스에서 이용할 수 있다는 것을 의미함
- 이는 native apps의 장점이 Web apps나 mobile cloud apps 보다 우위에 있다고 해석할 수 있음

Web application

- **표준 Web 기술을 사용하여 Platform이나 Device에 상관 없이 사용되도록 개발된 Application**
- Client-server 소프트웨어 어플리케이션
- 프로그램은 원격 서버에 저장되고 인터넷을 통해 웹 브라우저에서 실행
- 예) webmail, online retail sales, online auctions, wikis, instant messaging services 등

Cloud (native) application

- **Desktop application + Web application으로 클라우드 환경에서 실행되는 어플리케이션 프로그램**
- desktop app과 같이 응답 속도가 빠르고 오프라인에서도 작업 가능
- web apps처럼 특정 기기(device)에 종속될 필요가 없고 온라인으로 쉽게 업데이트 가능
- 사용자가 통제 가능하지만, 사용자 컴퓨터나 저장장치의 저장공간을 사용할 수 없음 .
- 잘 작성된 cloud apps은 desktop apps의 호환성과 Web apps의 휴대성을 모두 제공함

05 Cloud Native Application

» 개념 (Cloud 환경에 최적화 되어 서비스 되도록 개발된 어플리케이션)

Cloud Native Application의 핵심은

‘서비스’

- 어플리케이션을 여러 개의 서로 독립적인 기능을 하는 서비스로 구분
- 서비스들을 어떻게 구성하고 어떻게 연결하고 어떻게 관리하느냐가 관건
- ‘서비스’들을 묶어서 하나의 통합된 ‘(비즈니스) 서비스’를 할 수 있도록 하기 위한 다양한 기능과 기술 필요

05 Cloud Native Application

» 개념 (Cloud 환경에 최적화 되어 서비스 되도록 개발된 어플리케이션)

cloud-native application 은 최소의 상태(stateful)

컴포넌트들이 격리된 상태의 (마이크로)서비스로 구성되며,

각각의 서비스는 분산되고, 탄력적이며 수평적 확장성 있는

시스템으로 구성됨. 또한, 어플리케이션과 각각의 독립적인

배포 단위는 클라우드 중심의 디자인 패턴들과 셀프서비스

가능한 탄력적인 플랫폼에서 운영되도록 설계되어 있음.

출처 〈 "A Cloud-native Application Reference Model for Enterprise Architects", ClouNS〉

05 Cloud Native Application

» 주요 특징 비교

구분	전통적인 어플리케이션 아키텍처의 특징	클라우드 네이티브 어플리케이션 아키텍처의 특징
확장성	Scale Up	Scale Out
구조	단일구조 & 계층형(Monolithic & Layered)	분산 & Microservices
상태관리	Stateful – steady	Stateless – fluid and ephemeral
확장성	인프라 종속적 & 고정 인프라/ 고정된 용량	인프라와 무관 & 유연한 인프라 / 유연한 용량
종속성	대기시간 불허용, 단단한 결합	대기시간 허용, 느슨한 결합
관계형	Consolidated /clustered DB/Rich / Chatty Client	Sharded / replicated/ distributed DB / Mobile/thin Client
라이센스	상업 라이센스	오픈소스 / 지원 라이센스
가용성	인프라 지원 가용성 - 인프라 중복	앱 지원 가용성 – 복원력
복구/관리	수동 빌드/배치, 수동 오류 복구	자동 복구
백업/이중화	Active/Passive/DR	Active/Active

05 Cloud Native Application

» 주요 특징 비교

특징	필요성	방법론
• 서비스 • 오류/에러 처리 • 수평적 확장성 • 비동기 처리 • 무상태 모델 • 인력 개입 최소화 및 지속적인 배포 (CI/CD)	• 빠른 배포 (delivery) • 안전성 (fault tolerance, design for failure) • 확장성 – Scalability • 고객 다양성	• 마이크로서비스 지향 아키텍처 (하나의 서비스는 한가지 일에 집중 – Bounded Context) • API 기반 협업 및 통합 • 12-factor 기반의 방법론 • 멀티테넌트 DB 아키텍처 사용 • 셀프서비스 Agile/Scrum 방법론 사용

06 Microservices

» 문제점 (통짜 구조)

- **규모가 큰 애플리케이션에서는 불리**
  - ✓ 빌드, 배포, 서버 기동 시 시간이 오래 걸림
  - ✓ 한 두 사람의 실수는 전체 시스템의 빌드 실패를 유발→ 프로젝트가 커질수록 협업하기 어려움
  - ✓ 컴포넌트들이 서로 로컬 콜 (call-by-reference)기반으로 강하게 결합(tightly coupled) → 특정 컴포넌트나 모듈에서의 성능 문제나 장애가 다른 컴포넌트에까지 영향
  - ✓ 코드가 너무 커져서 유지보수 어려움
  - ✓ 기존 로직/데이터/인터페이스 등의 변경에 따른 영향을 파악하기 어려움

- **배포가 잦은 시스템에 불리**
  - ✓ 사소한 컴포넌트의 수정인데도 전체 어플리케이션을 재컴파일하여 배포를 하고, QA를 거쳐야 함

06 Microservices

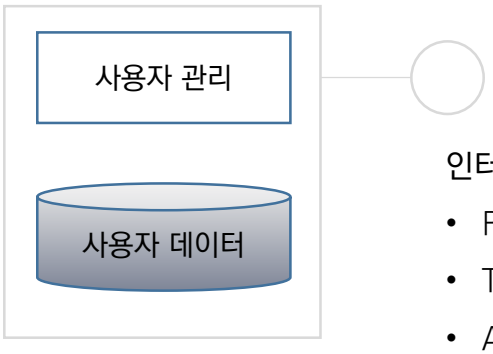
» 문제점 (통짜 구조)

- **새로운 기술에 대한 도입 어려움**
  - ✓ 컴포넌트 별로, 기능/비기능적 특성에 맞춰서 다른 기술을 도입하고자 할 때 유연하지 않음
  - 예) 파일 업로드/다운 로드와 같이 IO 작업이 많은 컴포넌트의 경우 node.js를 사용하는 것이 좋을 수 있으나, 어플리케이션이 자바로 개발되면 다른 기술 추가가 매우 어려움
  - ✓ On-Premise Cloud, CI와 연계된 배포 자동화(Jarvis), 향후 Docker와 같은 Container 기술과 연계 어려움
- **경직성**
  - ✓ 시스템을 분리, DB의 분리 어려움
  - ✓ 시스템간 연계의 증대에 대한 유연한 대응이 어려움
  - ✓ 새로운 버전이나 기술의 도입 어려움 또는 불가능
  - ✓ 조직의 비대화, 코드의 비대화 → 변화에 대한 저항 또는 장벽으로 작용
  - ✓ 한 어플리케이션에서 개발한 기능은 타 어플리케이션에서 사용하기 어려움

06 Microservices

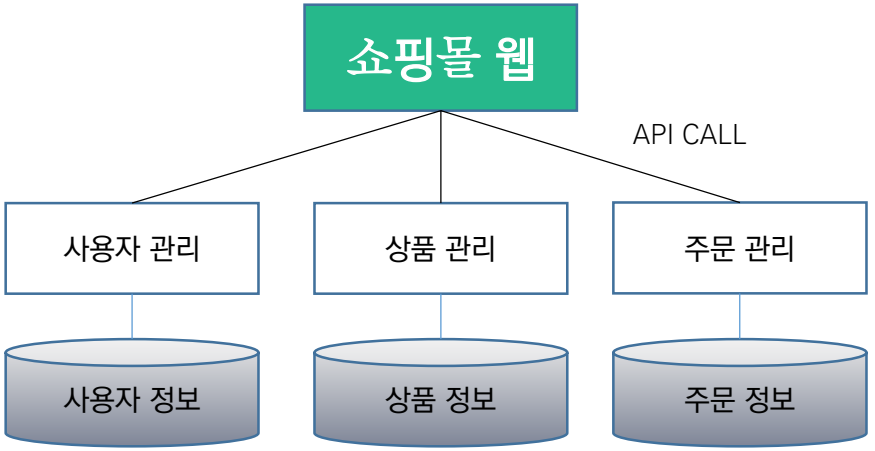
» 개요

- 업무상의 기능 또는 역할을 하나의 기능 묶음으로 개발된 컴포넌트 → 한 가지의 역할만 수행
- REST API 등을 통하여 서비스들의 기능을 제공하고 사용
- 데이터를 공유하지 않고 서비스별로 독립적으로 가공하고 저장함



인터페이스

- REST
- Thrift, Protocol buffer
- AMQ



하나의 기능을 구현 하는데 , 여러 개의 서비스를 조합하여 기능을 제공 예)
주문 하기 : 사용자 정보 조회 , 상품 정보 조회 , 신규 주문 생성

06 Microservices

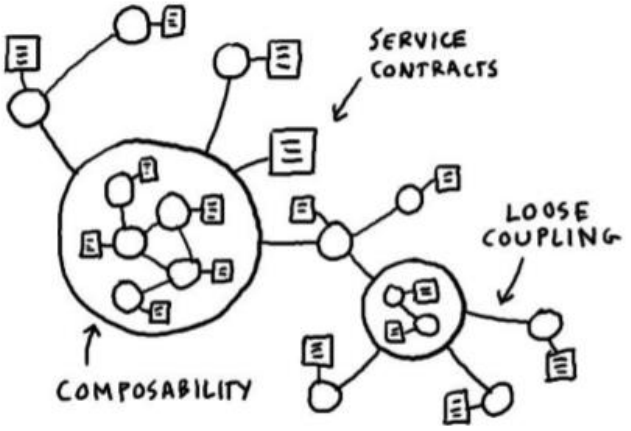
» 개요

*"마이크로서비스 아키텍처 스타일이란 단일 어플리케이션을
자체 프로세스로 실행되고 경량 매커니즘(주로 HTTP 리소스 API)으로 통신하는
작은 서비스들의 모음으로 개발하는 방식이다"*

– 마틴 파울러(Martin Fowler)

비즈니스 시스템 (어플리케이션) 을 개발 / 배포 / 운영할 때 ,

- **ONE THING**
  한 가지 기능(비즈니스 관련 기능/역할)을 수행하는데 초점을 맞춘 서비스를
- **SMALL**
  독립적이고 배포가능한 **가장 작은 단위의 서비스**(= atom)로 분리하고
- **API**
  API를 통해 다른 서비스와 연계하며
- **AUTONOMOUS**
  각각 자율적으로 개발, 운영. 즉, 독립적인 팀이 각 서비스(=atom)의 개발과 운영을 담당



06 Microservices

» 특징

- **작게 쪼개진, 범위가 명확한 다수의 작은 서비스들**
  - 단일 책임 원칙(Single Responsibility Principle)
  - 도메인 주도 개발(Domain Driven Development)
  - 제한된 컨텍스트(Bounded Context)
  - 독립적으로 관리됨(Independently Managed)
- **각 서비스의 명확한 소유권**
  - 일반적으로 “DevOps” 모델 필요/채택

Attribution: Adrian Cockroft, Martin Fowler …

06 Microservices

» 특징 (프로그램 언어, 프레임워크, 미들웨어 등에 초점을 맞춘 개념이 아님)

1. Componentization via Services (부품화 된 서비스) 한 가지 기능만 수행
2. Organized around Business Capabilities (비즈니스 기능/역할에 따른 분할)
3. Products not Projects (프로젝트가 아닌 개별 제품)
4. Smart endpoints and dumb pipes (단순한 어플리케이션간 연동과 파이프처리 – 유닉스의 철학)
5. Decentralized Governance (통제의 분권화)
6. Decentralized Data Management (데이터 관리의 분권화 – Polyglot Persistence)
7. Infrastructure Automation (자동화된 환경구축 – DevOps)
8. Design for failure (장애에 대비한 설계)
9. Evolutionary Design (변화에 대응하는 설계)

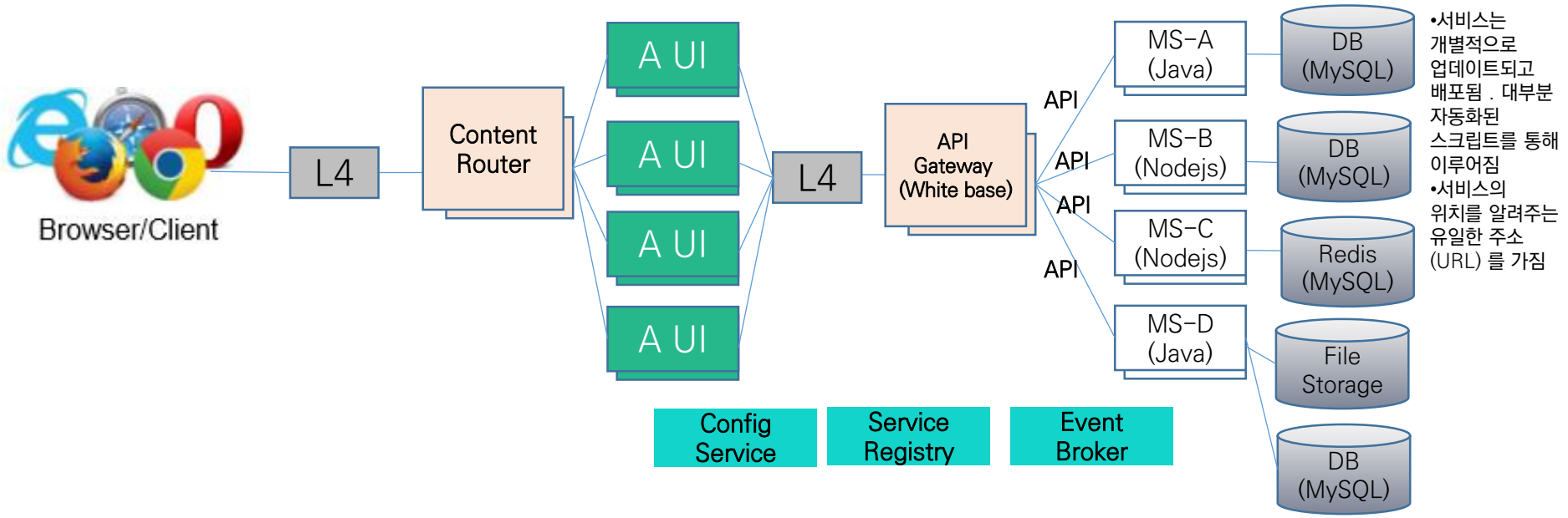
〈 출처 : http://martinfowler.com/articles/microservices.html〉

- 지금 개발하고 있는 것이 마이크로서비스 모델인지 아닌지 결정하는 단 하나의 심플한 규칙을 적용하고 싶다면, 서비스가 다른 서비스 및 서비스 버스(EBS)에 영향을 주지 않고 독립적으로 업데이트되고 배치될 수 있는지 보면 된다.
- 잘 개발된 마이크로서비스는 명확한 인터페이스와 프로토콜을 사용하며, 비즈니스를 독립적인 미들웨어로 압축 요약한다.

06 Microservices

» 아키텍처

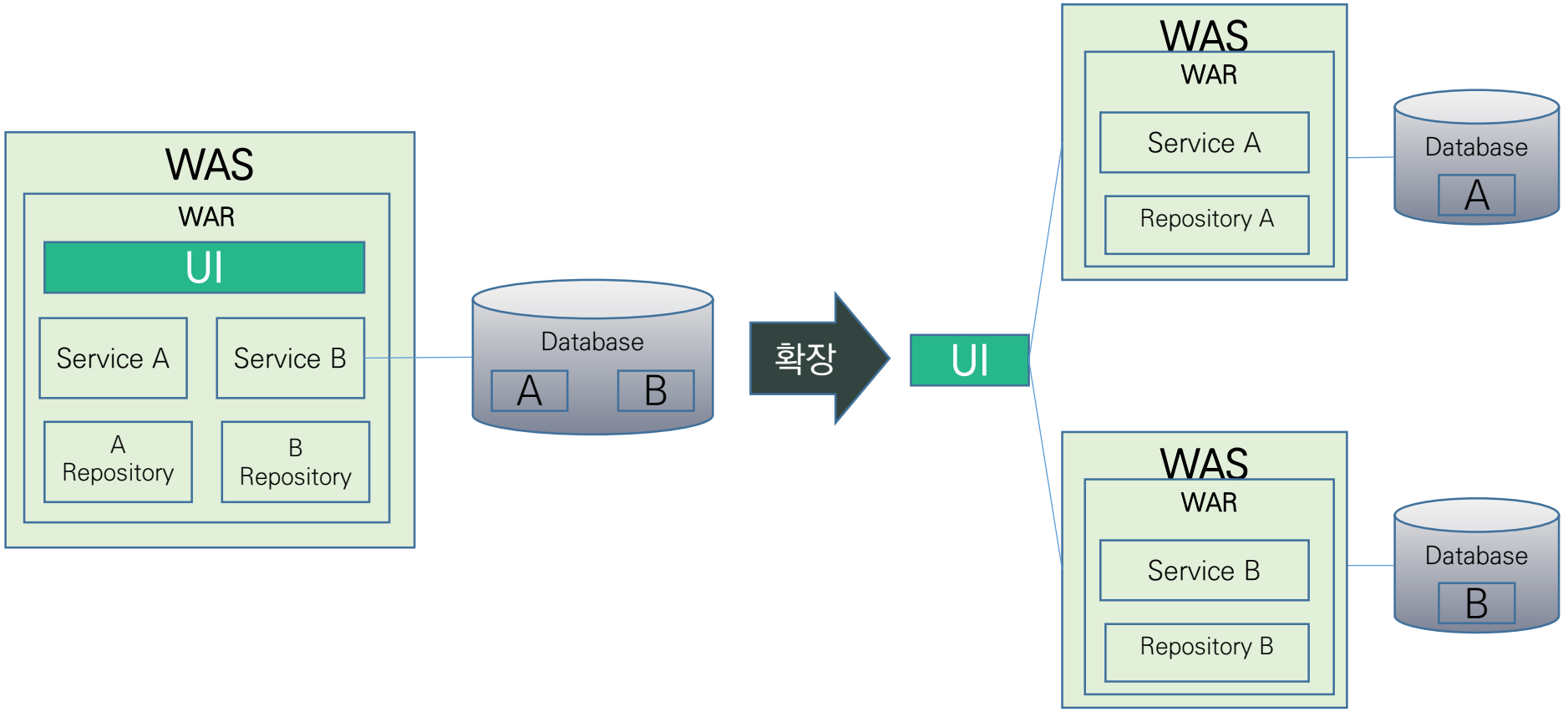
- 오로지 한 가지의 역할만 수행하는 서비스들(업무 기능, 시나리오, 특정 문제의 해결 등)이 서로 독립적이고 분권화되어 있음
- 고립된 서비스들은 표준화된API를 통해서 서로 통신/결합→ 다른 관련 서비스를 바꾸지 않고도 원하는 특정 서비스만 변경
- 구축 시 중점 사항 : 느슨하게 결합된 구성요소들, 확장성, 코드의 분리(partitioning), 업그레이드와 변경을 쉽고 빠르게 하고 유연성을 보장하는 상태
- 자가 치료 : 기계 고장으로 어플리케이션이 멈추면 자동 실행하고 이전의 상태로 복구할 수 있도록 개발
- 데이터베이스의 비정규화 : 서비스와 느슨하게 결합된 스키마 또는 각각의 microservice별로 별개의 스키마 생성



06 Microservices

» 아키텍처 모델 예시

서비스를 분할하고 서비스별로 별도의 데이터베이스 구축



06 Microservices

» 아키텍처 배경 (더 나은 소프트웨어를 더 빨리 출시 – 빠른 출시주기가 경쟁 우위의 원천)

도메인 주도 설계 (Domain Driven Design)	자동화를 통한 지속적/반복적 배포 (Continuous Delivery)	주문형 가상화 (On-demand Virtualization)
유연성, 확장성, 탄력성 (Elastic, Scalable, Resilience)	다중언어 프로그래밍 (Polyglot Programming)	인프라 자동화 (Infrastructure Automation)
애자일 방식의 개발 (Agile Development)	재사용성 (Reusability)	팀 자율성 (Self-government Team)

06 Microservices

» 아키텍처 구성하기 위한 핵심 요소

서비스

- 각 컴포넌트는 서비스라는 형태로 구현되고API를 이용하여 타 서비스와 통신
- 서비스 경계는 구문 또는 도메인(업무)의 경계를 따름

예) 사용자 관리, 상품 관리, 주문 관리와 같은 각 업무 별로 서비스를 나눠서 정의

- REST API에서 /users, /products와 같이 주요 URI도 하나의 서비스

DevOps

- DevOps는 CI에서 좀더 진화된 형태
- 개발, 테스트, 배포를 모두 자동화 시켜 개발 사이클이 끊임없이 순환되도록 함으로서 개발의 속도를 최대화 시키는 개발스타
- 배포가 서비스의 수 만큼 이루어지게 될 뿐만 아니라 테스트 또한 각각의 서비스가 연동되어 발생하는 집합체Aggregate의 수 만큼 필요하게 되므로 필연적으로 DevOps 필요

데이터 분리

- 서비스 별로 필요에 따라 별도의 데이타 베이스를 사용
- 서비스가API에서부터 데이터베이스까지 분리되는 수직 분할 원칙 (Vertical Slicing)에 따름
- 데이터베이스의 종류 자체를 다르게 하거나, 같은 데이터 베이스를 사용하더라도 스키마를 나누는 방법 사용

API Gateway

모든 api에 대한 end point를 통합하고, 몇가지 추가적인 기능을 제공하는 미들웨어

- EndPoint 통합과 토폴로지 정리
- Orchestration : 여러 개의 서비스를 묶어서 하나의 서비스 생성
- 공통 기능 처리 (Cross cutting function handling) : API 인증 (Authentication), Logging 등
- Mediation : 메시지 포맷 변환, 프로토콜 변환, 메시지 라우팅 등

06 Microservices

» 장점

- **보다 빠르고 간단한 배치와 롤백**
  - 팀별 독립적인 Delivery 속도
- **각 도메인에 적합한 프레임워크/도구/언어**
  - 컴포넌트는 Python 언어로, 카탈로그 서비스는 Java로…
- **더 큰 탄력성**
  - 오류 격리(Fault Isolation)
- **더 나은 가용성**
  - 단, 적합한 설계의 경우 ☺

06 Microservices

» 장점

- Technology Heterogeneity
  - ✓ 요구사항을 구현하기 위해 최적화된 언어와 아키텍처의 선택 : 다른 프로그래밍언어, 다른 도구를 사용하여 개발 가능
- Resilience
  - ✓ 오류 발생 시 복구될 때까지 요청 가능 서비스에서 제외 (Circuit Breaker와 로드밸런서가 담당)
- Scaling
  - ✓ 서비스들은 서로 독립적이므로 타 서비스에 영향을 주지 않고 서비스 단위로 확장 가능 →API(특히 REST API)를 통해 서비스 간 통신
  - ✓ X축 확장으로 불리는 멀티 애플리케이션(또는 서버)의 확장과 Z축 확장(Partitioning 또는 Sharding)으로 불리는 확장을 독립적으로 수행
- Ease of Deployment
  - ✓ DevOps와 결합된 각각의 마이크로서비스는 단순한 구조 → 개발속도와 개선에 높은 효용성
  - ✓ 자동화된 단위 테스트와 시나리오 테스트는 빠른 배포주기에도 불구하고 뛰어난 품질을 유지할 수 있도록 함
- Organizational Alignment
  - ✓ 각각의 마이크로서비스는 개별 팀에서 독립적으로 개발/배포가 가능.
  - ✓ 시스템의 규모가 커짐에 따라 추가로 발생하게 되는 오버헤드가 일정수준으로 관리가 가능
- Composability
  - ✓ 개별 비즈니스 요구사항에 특화된 단순한 서비스 → 개발자의 관리 범위 명확 → 소프트웨어의 복잡성을 제어(UI와 컨트롤, 도메인 로직이 별도의 마이크로서비스로 구성되어 완전히 독립적으로 개발)
  - ✓ 각 서비스는 다른 데이터 저장소를 사용할 수 있으며 서로 느슨하게 연결
- Replaceability
  - ✓ 서비스를 나누는 규칙, 즉 서비스를 모듈화하는 규칙으로 동일한 기능을 하는 서비스는 하나의 서비스로 대체 가능

06 Microservices

» 단점

- **복잡성(Complexity)**
  - ✓ Hard to develop features span multiple services
  - ✓ Greater operational complexity – more moving parts
  - ✓ Additional complexity of creating a distributed system – network latency, fault tolerance, serialization, …
  - ✓ Multiple Database & Transaction Management
  - ✓ Service interfaces and versioning
- **Complicated Test : End-to-end testing**
  - ✓ **개별 서비스에 대한 테스트는 만들기가 수월하지만 런타임 환경상에서 비동기 상호작용을 테스트하기는 까다로움**
- Require Automation for Deploy/Operation
- Devs need significant ops skills
- **중복성(Duplication)**
  - ✓ Duplication of effort across service implementations
  - ✓ **코드 중복: 여러 언어를 사용하여 개발이 진행되는 경우 코드중복은 필연 → 오버헤드, 라이브러리 호환성 등을 충분히 고려한 이후 도입**
  - ✓ **데이터 중복**: Maintaining availability and consistency with partitioned data
- Avoiding latency overhead of large numbers of small service invocations
- Designing decoupled non-transactional systems is hard
- Locating service instances

06 Microservices

» 고려사항

서비스 범위 설정 문제

- 어디서부터 어디까지를 묶어야 독립적으로 운영 가능한 서비스가 되는가?
- 동일한 문제영역을 나타내는 모델이 한 개 이상 존재할 수 있으며 이러한 문제영역을 올바르게 이해하는데 필요한 것은 실제 문제영역이 어떻게 동작하고 있는가에 대해 있는 그대로를 관찰하고 이를 바탕으로 서비스를 구성

레거시 시스템과의 공존에 대한 고려

- 마이크로서비스를 도입하더라도 (일정 기간은) 기존의 (특히 Monolith)시스템들과의 공존은 필연적으로 존재할 수밖에 없는 상황에서 기존 시스템들과 어떻게 연계할 것인가?

운영 오버헤드

- 마이크로서비스는 엄청나게 많은 양의 배포작업 : 릴리즈가 개별적으로 이루어지는 특성상 이를 별도의 운영팀에서 일괄적으로 관리하는것은 불가능하므로 배포에 수반되는 **일련의 작업들을 자동화**할 수 있도록 DevOps 도입

인터페이스 불일치:

- 어떻게 각각의 서비스의 인터페이스를 변경하는 것에 대한 영향범위를 파악할 것인가?
- 어떻게 서비스 외부로 제공하는 인터페이스가 의도하지 않은 형태로 사용되지 않도록 할 것인가?
- 어떻게 전체 시스템의 인터페이스 맵을 만들고 팀 간의 커뮤니케이션 비용을 효과적으로 제어할 수 있는가?

06 Microservices

» 모델링/구현 Tip

- API를 먼저 정의하라.
- API를 REST API Maturity Level 2 이상이 되도록 강제화하라.
- API 문서를 유지하라(예: Swagger)
- ORM을 활용하라
- DB에 너무 의존하지 마라
- 도메인 내부에서만 의미있는 값을 외부에 노출하는 것을 지양하라
- 마이크로 서비스가 별다른 설정 없이 바로 기동가능하게 하라

  (예: Java의 경우, Spring Boot + Embedded WAS 활용)

07 API Gateway

» API 정의

Application Programming Interface (API) accessibility to software that enables machines to interact with cloud software in the same way the user interface facilitates interaction between humans and computers. Cloud computing systems typically use REST-based APIs.

- 소프트웨어가 서로 의사소통을 하는 규약
- 특정한 task 가 수행되는 방법을 표현
- 일반적 의미로는 운영체제, 어플리케이션, 라이브러리 등 다양한 수준의 인터페이스를 총칭
- REST-API와 같은 표준화된 방식의 API를 사용하여 각 서비스/어플리케이션 간의 통신
- 표준화된 통신 방식을 사용하기 때문에 각 서비스의 구현은 Polyglot Programming과 같이 다양한 방향으로 구현 가능

07 API Gateway

» 정의

MSA(Micro-Services Architecture)와 함께 근래에 떠오르는 기술

- *API 게이트웨이는 API 서버 앞단에서 모든 API 서버들의 엔드 포인트를 단일화하여 묶어주고 API에 대한 인증과 인가 기능에서 부터 메세지에 따라서 여러 서버로 라우팅 하는 고급기능까지 많은 기능을 담당*
- ESB와의 관계
  - SOA : ESB = MSA : API 게이트웨이
  - ESB : API Gateway로 발전 : ESB의 대부분의 컨셉을 많이 승계
  - ESB가 SOAP/XML 웹서비스 기반의 많은 기능을 가지는 구조였다면, API 게이트 웨이는 JSON/REST 기반에 최소한의 기능을 처리하는 경량화 서비스
  - ESB의 실패와, MSA, REST 구현 사례를 통해서 필요에 의해서 탄생한 솔루션 실용성 강화
- API 게이트웨이는 여러 개의 엔드포인트를 설정하고, 각 엔드포인트 마다 메세지 흐름을 워크플로우 엔진 설정을 통해서 API 에 대한 Mediation, Aggregation 설정을 할 수 있는 미들웨어

Key Point

- *API Gateway를 도입하기 위해서는 적절한 Gateway 아키텍처를 설계 필요*

07 API Gateway

» 주요 기능

인증/인가에 관련된 기능

- API 토큰 발급
- 엔드 포인트별 API 호출 인증
- 엔드 포인트별 API 요청 인가

API 라우팅

- 백엔드 API 서버로의 로드 밸런싱
- 서비스와 클라이언트 별 엔드 포인트 제공
- 메시지 또는 헤더기반 라우팅

공통 로직 처리

- 인증(Authentication)
- 로깅(logging) 등

메디에이션 기능 (Mediation)

- 메시지 포맷 변환
- 프로토콜 변환
- Message Exchange Pattern (MEP)
- Aggregation

Logging / 미터링

- API 호출 로깅
- API 미터링 & 차징
- 사용 고객의 등급에 따른 서비스 레벨 조정

07 12 Factors

» 개발 원칙

- 버전 관리되는 하나의 코드베이스와 다양한 배포
- 명시적으로 선언되고 분리된 종속성
- 환경(environment)에 저장된 설정
- 백엔드 서비스를 연결된 리소스로 취급
- 철저하게 분리된 빌드와 실행 단계
- 애플리케이션을 하나 혹은 여러개의 무상태(stateless) 프로세스로 실행
- 포트 바인딩을 사용해서 서비스를 공개함
- 프로세스 모델을 사용한 확장
- 빠른 시작과 그레이스풀 셧다운(graceful shutdown)을 통한 안정성 극대화
- development, staging, production 환경을 최대한 비슷하게 유지
- 로그를 이벤트 스트림으로 취급
- admin/maintenance 작업을 일회성 프로세스로 실행

핵심정리

- Cloud는 리소스를 합리적으로 사용할 수 있는 **세계적인 추세**며, 그 **시장규모도 매년 크게 성장**하고 있음
- 동일한 기능의 Application이 넘쳐나는 시대에 시장의 **요구를 조금 더 빨리 충족시키는 것이 경쟁력**
- 클라우드 모델 중 **PaaS는 빠른 개발, 빠른 배포에 용이한 아키텍처**를 가지고 있음

MEMO

MEMO

M2 PaaS-TA 개발 실무

01. PaaS-TA 이해

02. PaaS-TA 개발 환경의 이해

03. PaaS-TA 개발 도구 이해 및 실습

04. PaaS-TA 개발 검증 및 문제해결

M2 학습목표

- PaaS-TA 동작 방식을 이해한다.
- Cloud Native Application의 필요성과 구현 방법을 이해한다.
- PaaS-TA ServicePack의 동작 방식과 broker 구현 방법을 이해한다.
- PaaS Platform인 PaaS-TA를 사용할 수 있다.(Portal, CLI, IDE)

01. PaaS-TA 이해

M1. Cloud Basic

01. Cloud 이해

02. Cloud Model 및 특징 이해

M2. PaaS-TA 개발 실무

01. PaaS-TA 이해

- PaaS-TA 소개 ✓
- PaaS-TA 아키텍처의 이해
- PaaS-TA 주요 구성요소
- PaaS-TA 운영 및 편의 도구
- PaaS-TA 상황별 구성요소 기능

02. PaaS-TA 개발 환경의 이해

03. PaaS-TA 개발도구 이해 및 실습

04. PaaS-TA 개발 검증 및 문제해결

M3. PaaS-TA 배포 및 운영

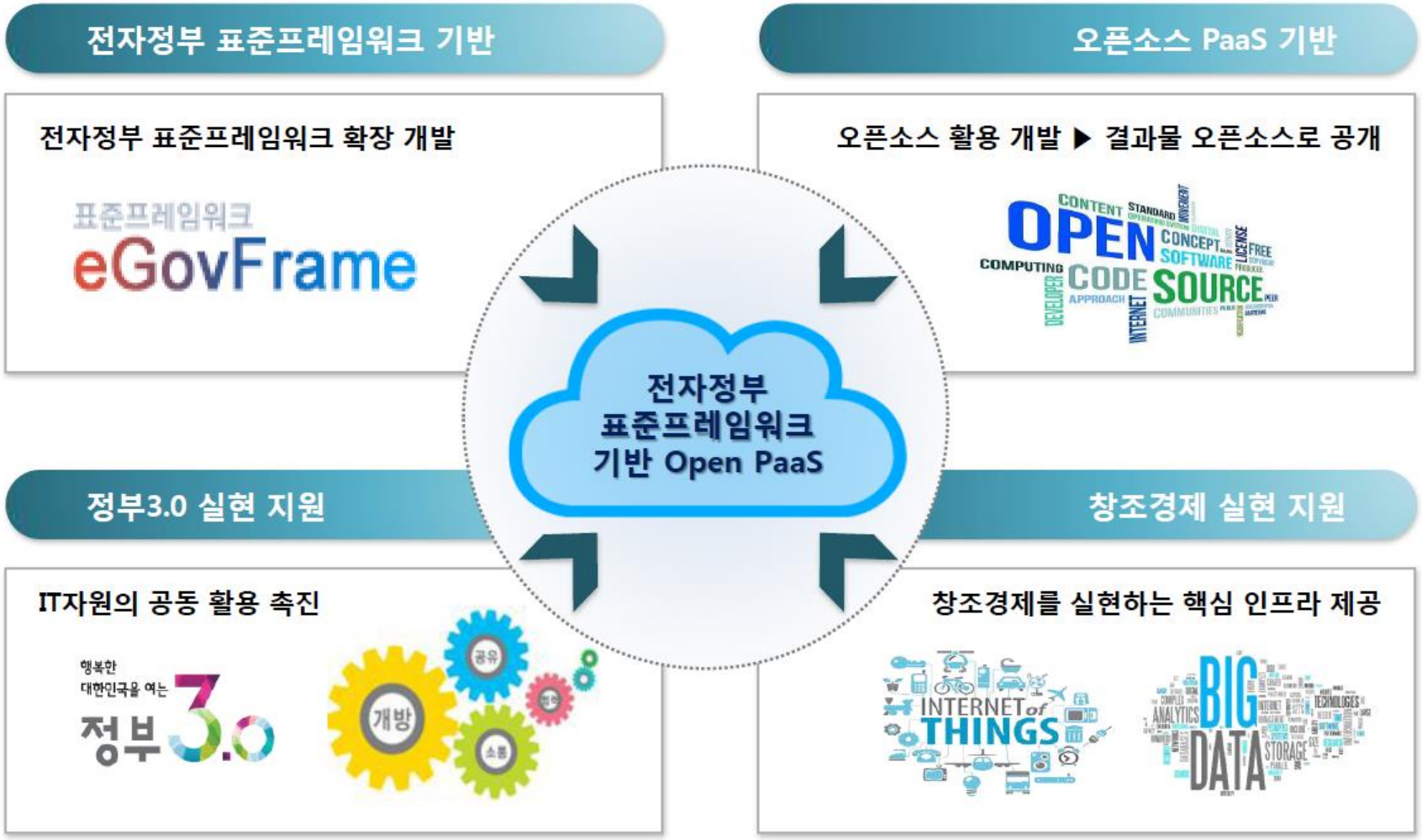
01. PaaS-TA 배포 및 관리

02. Service Package 배포 및 관리

03. Custom Buildpack 개발

01 PaaS-TA 소개

» PaaS-TA 개발 배경



출처: (NIA) 개방형 클라우드 플랫폼 PaaS-TA 소개 및 발전방향

01 PaaS-TA 소개

» PaaS-TA 정의

PaaS-TA
OpenPaaS

> *"PaaS-TA는 **클라우드 인프라 환경을 제어**하면서*
> ***애플리케이션을 쉽게 개발하고 안정적으로 운영**할 수 있도록*
> ***관리 · 지원하는 클라우드 플랫폼**"*

01 PaaS-TA 소개

» 단계 별 개발 진행

1단계 개발 (2017)

개발/운영/관리환경 고도화

2단계 개발 (2018)

이종 클라우드 지원 및 관리기술 개발

3단계 개발 (2019)

응용 마켓플레이스 구현 및 개발자 지원환경 개발

01 PaaS-TA 소개

» 단계 별 개발 진행

1단계 개발
(2017)

개발/운영/관리환경
고도화

- IaaS, PaaS, SaaS 통합 모니터링 기능 확장 개발
- Application Gateway 기능 개발
- 신규 클라우드 IaaS 지원 검증 및 설치 자동화 확장 개발
- 이종 클라우드 기반의 개방형 PaaS 플랫폼 지원 개발
- 플랫폼 운영자 및 개발자 포탈 기능 확장 개발

01 PaaS-TA 소개

» 단계 별 개발 진행

2단계 개발 (2018)

이종 클라우드 지원 및 관리기술 개발

- IaaS, PaaS, SaaS 통합 모니터링 기능 개발
- 모니터링 기반 오토스케일링 기능 개발
- 포탈 통합 계정 인증/권한 관리 기능 개발
- 응용의 개발/테스트/배포 개발도구 개발
- 신규 클라우드 IaaS지원 검증 및 설치 자동화 개발
- 이종 클라우드 기반의 개방형 PaaS 플랫폼 지원 검증

01 PaaS-TA 소개

» 단계 별 개발 진행

3단계 개발 (2019)

응용 마켓플레이스 구현 및 개발자 지원환경 개발

- IaaS, PaaS, SaaS 통합 모니터링 기능 확장 개발
- 응용의 전 주기 라이프 사이클 관리 기능 개발
- PaaS상에 동작하는 응용을 위한 마켓플레이스 기능 개발
- 응용의 사용량 측정 기능 고도화
- 신규 클라우드 IaaS 지원 검증 및 설치 자동화 확장 개발
- On-Demand 서비스 설치 개발
- 이종 클라우드 기반의 개방형 PaaS플랫폼 지원 기능 확장 개발

01 PaaS-TA 소개

» PaaS-TA 버전

PaaS-TA 1.0

Open PaaS 핵심기능 제공

- 오픈소스 PaaS 검증·안정화
- 다양한 IaaS 지원
- 설치 자동화
- 표준프레임워크 및 국산 SW 탑재

PaaS-TA 2.0

운영 및 관리 도구 강화

- 사용자 포탈, 운영자 포탈
- 모니터링 시스템
- 설치 자동화 고도화
- 미터링

PaaS-TA 3.0

개발/운영/관리 환경 고도화

- 배포 파이프라인 서비스 도구
- 형상 관리 서비스 도구
- 서비스 모니터링 도구 업그레이드
- 사용자/운영자 포털 업그레이드
- 설치 자동화 Google GCP 추가
- IaaS 관리 대시보드

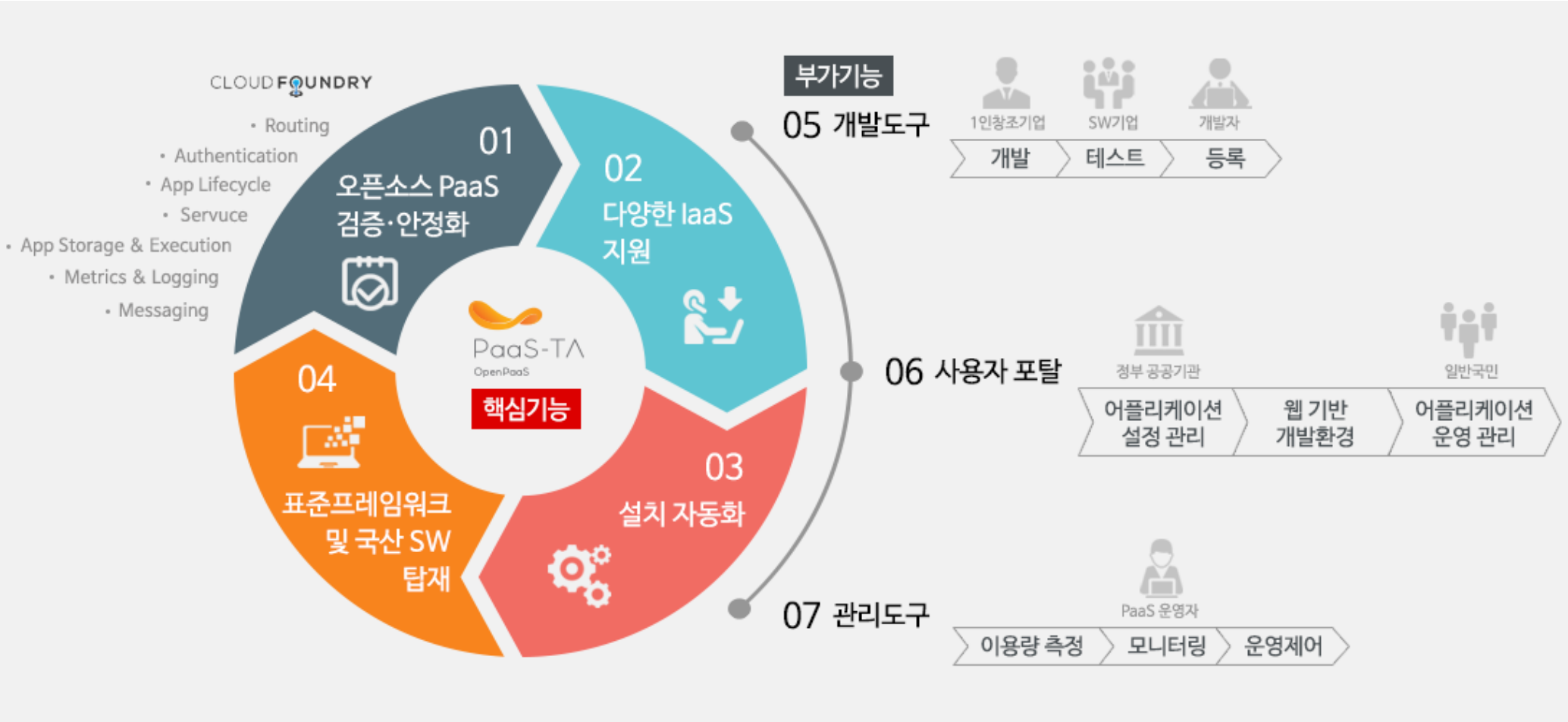
PaaS-TA 4.0

이종 클라우드 지원 및 관리 기술 개발

- IaaS, PaaS, 통합 모니터링
- SaaS 모니터링
- 응용의 전주기 라이프 사이클 관리 개발
- Logging as Service 개발
- Kubuemetes을 활용한 컨테이너 제어 및 관리 검증
- 애플리케이션 운영지원 확장
- 신규 클라우드 IaaS 지원 검증 및 그에 따른 설치 자동화 개선
- 이종 클라우드 기반의 개방형 PaaS 플랫폼 지원 개발

01 PaaS-TA 소개

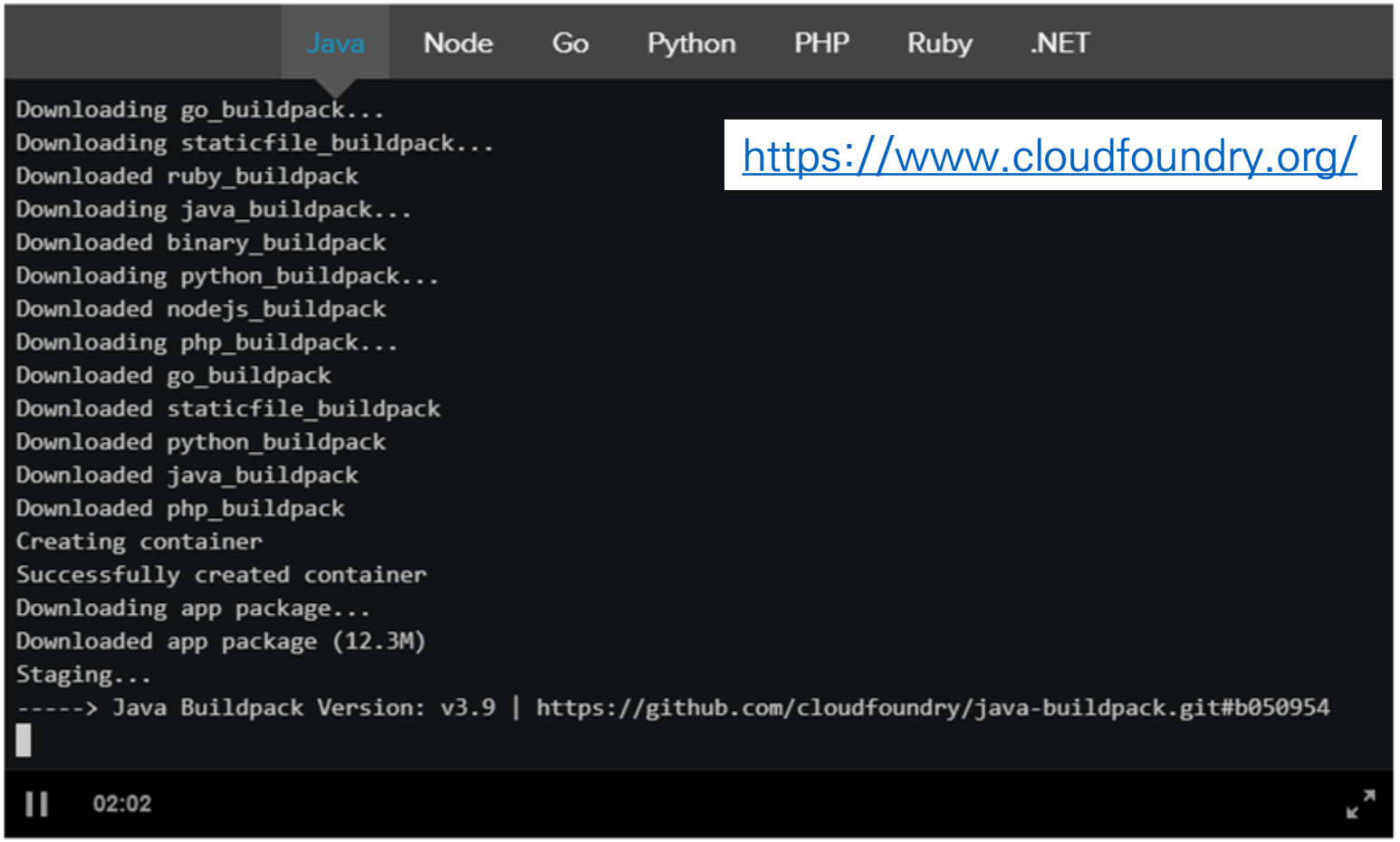
» 핵심기능



01 PaaS-TA 소개

» 핵심기능

배포 파이프라인을 이용한 개발내용 배포 라이프사이클 예시



01 PaaS-TA 소개

» Cloud Foundry와 PaaS-TA 비교

기능		Cloud Foundry	PaaS-TA 3.0
서비스 지원		글로벌 오픈소스 6종	국산SW포함 18종 (Cloud, Tibero, Altibase, Arcus 등)
런타임 환경		글로벌 오픈소스 9종	전자정부 표준프레임워크 포함 11종
GUI 도구	설치 자동화	-	4종 지원 (AWS, Openstack, Vsphere, GCP)
	IaaS 관리 대시보드	-	2종 지원 (AWS, Openstack)
	사용자 포탈	-	지원
	관리자 포탈	-	지원
모니터링	가상머신	지원(Memory)	6종 지표 지원 (CPU, Memory, Disk, Disk IO, Network IO, Process)
	컨테이너	-	5종 지표 지원 (CPU, Memory, Disk, Disk IO, Network IO)
	애플리케이션	-	지원
	서비스	-	지원
	알람	-	지원
미터링		-	지원
개발 지원 도구		-	배포 파이프라인, Git/SVN 형상관리, 품질관리

사용자 편의성과 **관리 효율성 향상**을 위한 서비스 및 도구 확대

01 PaaS-TA 소개

» 특징

글로벌하게 검증된 오픈소스를 활용하여 개발됨

전체가 오픈 라이선스인 아파치 라이선스로 개방

01 PaaS-TA 소개

» 장점

1 기존 IaaS 제공 기업은 PaaS-TA를 활용함으로써 경쟁력 강화 가능

다양한 인프라를 지원하므로 플랫폼 별 중복 개발없이 빠르게 개발 운영 가능

2 7종 이상(Spring, 전자정부 프레임워크 등)의 다양한 IaaS 지원

Openstack, AWS, Vagrant, VMWare, Cloudstack, GCP, Azure 등

01 PaaS-TA 소개

» 장점

3 풍부한 개발/운영환경 제공

- 8종(Java, NodeJS, PHP, Ruby, Golang 등) 개발 언어
- 10종(Spring, 전자정부 프레임워크 등) 프레임 워크
- 국산 SW(Tibero, Cubrid, Altibase, Arcus 등) 서비스 지원 및 IaaS, PaaS, SaaS

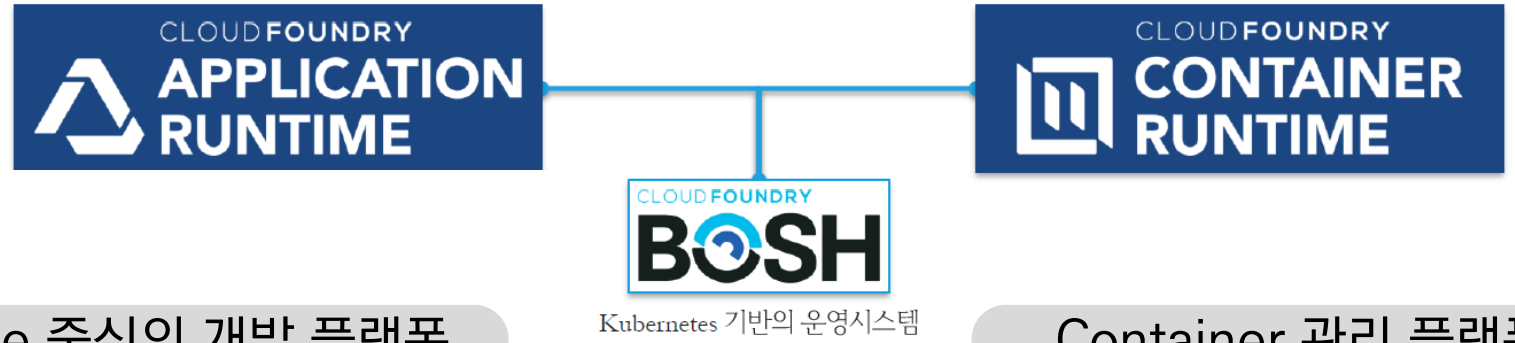
4 통합관제 지원

- 단계별 상세 가이드 문서 제공:
  플랫폼 설치, 서비스 설치, 플랫폼 확장, 애플리케이션 개발 등 23종 한글문서 제공 및 버전별 지속 업데이트
- 국내 클라우드 산업 활성화: 안정적인 기술지원과 개발자 육성지원

01 PaaS-TA 소개

» 장점

5 Application Runtime기반인 PaaS-TA에서 Container Runtime을 활용한 시너지 효과



Code 중심의 개발 플랫폼

- 개발 언어/Framework에 관계없이 원하는 Cloud에 Code를 실행
- Application Lifecycle 관리
- Application 자동실행
- Open Service Broker API를 통한 손쉬운 SaaS서비스와 연동

Container 관리 플랫폼

- Kubernetes 인증된 Container의 쉬운 관리, 유지보수 가능
- Cloud Foundry BOSH를 통한 Lifecycle 관리
- 기 개발된 Application의 Container화 및 Cloud 상에서 운영 및 재 사용이 용이

01 PaaS-TA 소개

» PaaS-TA 개발 생산성 극대화

전통적인 환경에서 개발자 업무요건	
01	하드웨어 구입요청
02	하드웨어 취득
03	하드웨어 공간 확보 및 정돈
04	OS 설치
05	OS패치 및 패키지 설치
06	계정 생성
07	프레임워크 디플로이
08	미들웨어 디플로이
09	테스팅 툴 디플로이
10	테스팅 툴 테스트
11	코딩
12	멀티태넌시 처리
13	서버 환경 설정
14	생산완료
15	런칭
16	서버 추가구매 요청
17	승인 응답 대기
18	새로운 서버 디플로이 etc

IaaS에서 개발자 업무요건	
01	가상머신 요청
02	프레임워크 디플로이
03	미들웨어 디플로이
04	테스팅 툴 디플로이
05	테스팅 툴 테스트
06	코딩
07	멀티태넌시 처리
08	가상머신 서버 환경 설정
09	생산완료
10	런칭
11	가상 머신 서버 추가구매 요청
12	승인 응답 대기
13	새로운 VM서버 디플로이 etc

PaaS에서 개발자 업무요건	
01	개발자원 신청 할당
02	코딩
03	테스트
04	런칭

01 PaaS-TA 소개

» PaaS-TA 구성 요소

01 PaaS-TA 소개

» PaaS-TA 구성 요소

- 인프라 제어 및 관리 서비스
- 실행환경
- 서비스환경
- 개발환경
- 운영환경

인프라 제어 및 관리

Large Scale의 분산된 **서비스들의 라이프사이클을 관리**하고, **릴리즈 및 Deployment 서비스를 관리**하기 위한 **통합 서비스**

다양한 **클라우드 인프라와의 연동** 기능을 제공

01 PaaS-TA 소개

» PaaS-TA 구성 요소

인프라 제어 및 관리 서비스

실행환경

서비스환경

개발환경

운영환경

- **애플리케이션의 개발 및 배포, 실행/운영 관리를 위한 서비스** 제공
- **다양한 언어팩을 지원**하여 **애플리케이션 실행을 위한 컨테이너를 제공**
- **서비스 환경과 연계하여 PaaS 내부 및 외부 서비스를 사용**할 수 있도록 지원
- PaaS 기능 사용을 위한 계정 등록 및 접근 인증, 인증 후 사용할 수 있는 API·제어가 가능한 리소스에 대한 제어를 수행하는 권한 관리, 애플리케이션에 접근하기 위한 **접근관리 서비스 제공**

01 PaaS-TA 소개

» PaaS-TA 구성 요소

- 인프라 제어 및 관리 서비스
- 실행환경
- **서비스환경**
- 개발환경
- 운영환경

- 응용 애플리케이션 실행 시 RDBMS, NoSQL, 메시징 서비스 등 **다양한 대외 백엔드 플랫폼 서비스와 연계를 위한 대외 서비스 브로커를 제공**
- 내외의 API 관리 및 연계를 위한 **API 관리 서비스를 제공**

01 PaaS-TA 소개

» PaaS-TA 구성 요소

- 인프라 제어 및 관리 서비스
- 실행환경
- 서비스환경
- **개발환경**
- 운영환경

✔ PaaS 플랫폼을 활용하여 **응용애플리케이션을 개발 및 배포, 운영**하기 위한 **셀프서비스 포털과 개발도구를 제공**

01 PaaS-TA 소개

» PaaS-TA 구성 요소

인프라 제어 및 관리 서비스

실행환경

서비스환경

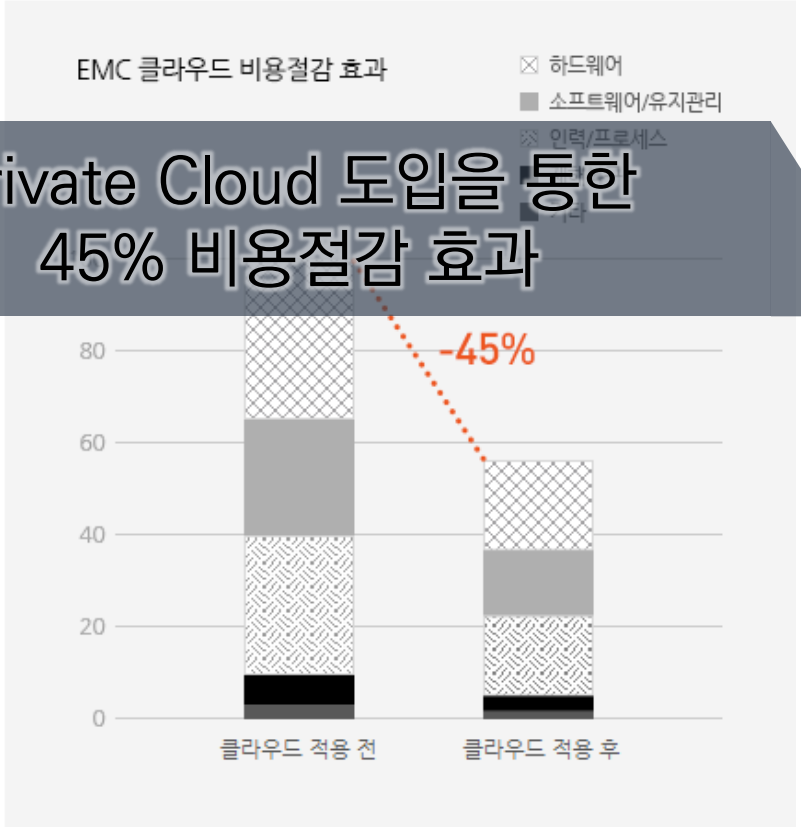
개발환경

운영환경

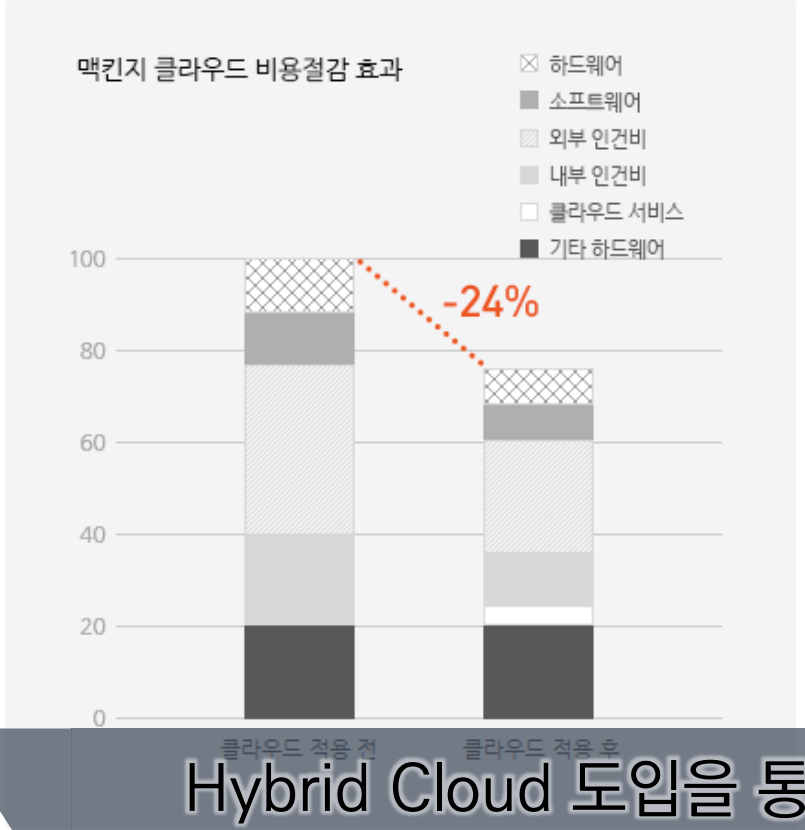
- **인프라 제어 및 연동**과 **애플리케이션 플랫폼을 위한 관리 서비스**
- 관리자를 위한 운영 대시보드와 개별 테넌트 및 모니터링 대시보드와 로그 관리, 미터링 플러그인 서비스 제공

01 PaaS-TA 소개

» PaaS-TA 기대효과



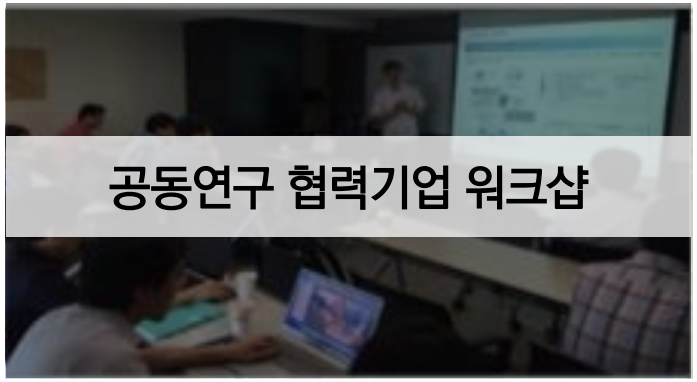
Private Cloud 도입을 통한
45% 비용절감 효과

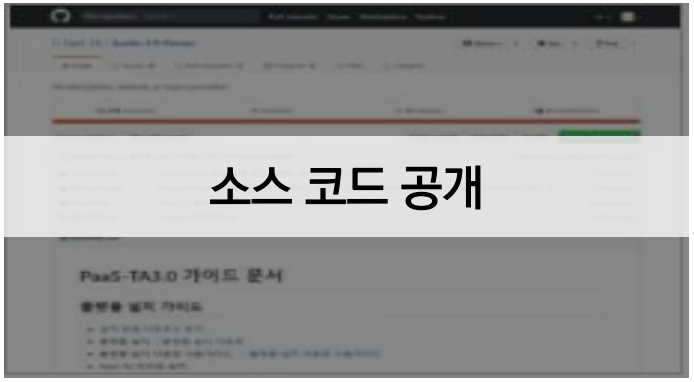


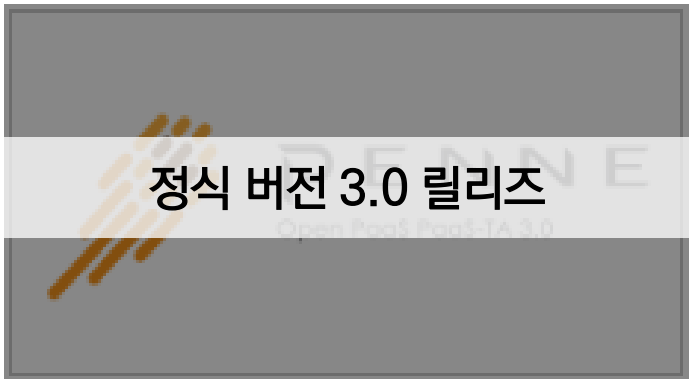
Hybrid Cloud 도입을 통한
24% 비용절감 효과

01 PaaS-TA 소개

» 활동현황

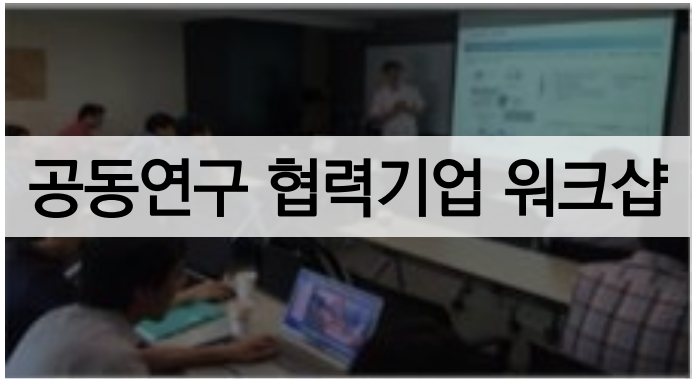






01 PaaS-TA 소개

» 활동현황



- CF, Apache 재단과 같이 모든 소스코드를 오픈소스로 공개 및 관리화 예정
- CF코어를 제외한 R&D 자체 개발 파트만 공개

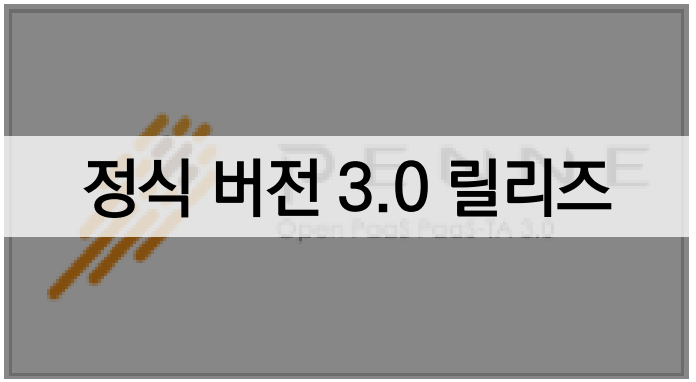
- 1차년도 개발 추진 내용 발표 및 피드백
- 각 협력기업 별 공동연구 진행 현황 공유

01 PaaS-TA 소개

» 활동현황

PUB 공개 세미나

- PaaS-TA 고도화 사업 1차년도 개발 내용 발표
- 주요 기능 시연



정식 버전 3.0 릴리즈

- Github 및 PaaS-TA 포탈에 소스코드 및 패키징 파일 등록
- 정식 버전 3.0 공개('17년 12월 14일)

01 PaaS-TA 소개

» PaaS-TA 포탈

포탈을 통한 **한글 가이드** 및 **교육, 기술 지원**

출처: paas-ta.kr

핵심정리

- PaaS-TA는 **국내 PaaS 생태계를 위해 개발**된 오픈소스
- **Cloud Foundry를 기반**으로 하고 있지만 **개발편의성을 더 개선하고 표준화**함
- **정기적인 릴리즈 발표**와 **세미나 교육지원** 등이 적극적으로 이루어지고 있음
- **국내 PaaS 생태계의 basement**가 되도록 노력 중

MEMO