나도 해보자!
클라우드 서비스와 프로젝트 이해하기

이기하
dasomell@gmail.com
2021.10.14

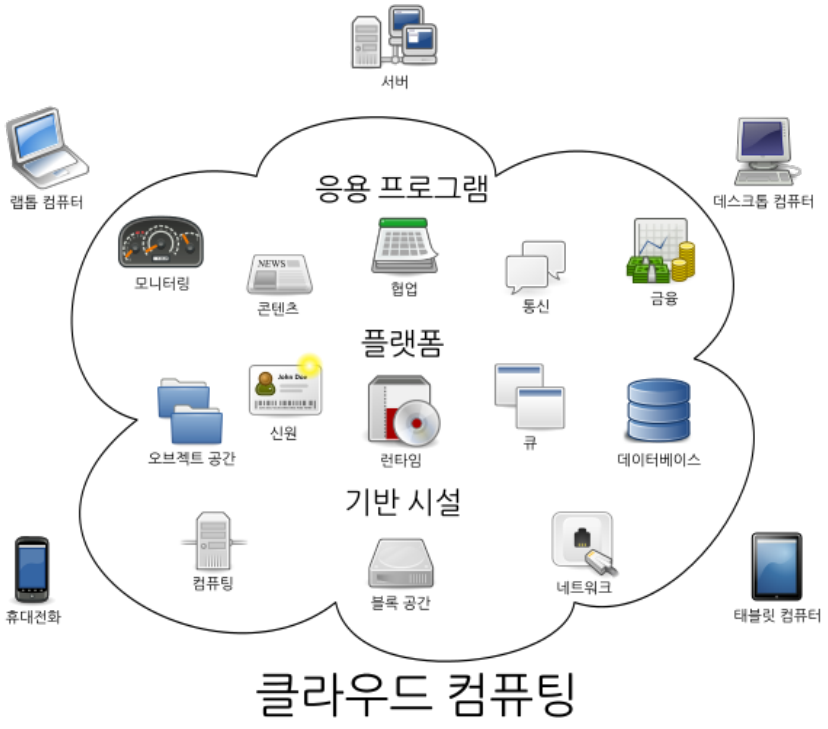
목 차

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)

2.Cloud란?

2.1 Cloud 정의

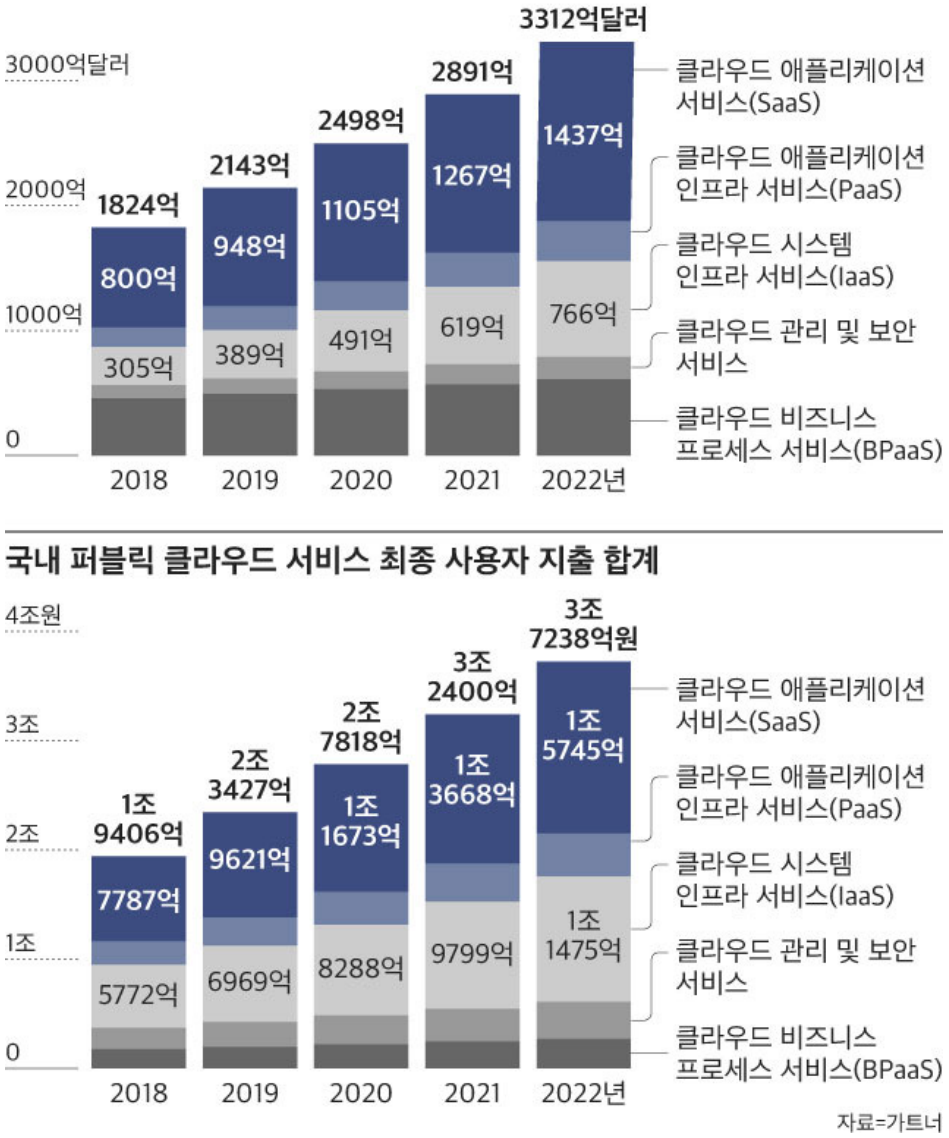


네트워크 기반의 컴퓨팅 기술

- Compute, Storage, and Networking Resources
- Delivered as a service (i.e. IaaS)
- Self service access via web interfaces and APIs
- Provision / release resources in minutes
- Pay for usage model

이미지 출처: https://ko.wikipedia.org/wiki/클라우드_컴퓨팅

2.2 Cloud 서비스 전망



이미지 출처: https://biz.chosun.com/site/data/html_dir/2019/04/03/2019040302058.html

SaaS (Software as a Service)

비즈니스 프로세스, 엔터프라이즈 애플리케이션 및 협업 도구를 포함하여 소프트웨어가 클라우드를 통해 고객에게 서비스로 제공되는 소프트웨어 배치 모델

PaaS (Platform as a Service)

클라우드 기반 환경의 애플리케이션, 최적화된 미들웨어, 개발 도구, Java 및 웹 2.0 런타임 환경 등 컴퓨팅 플랫폼을 제공

IaaS (Infrastructure as a Service)

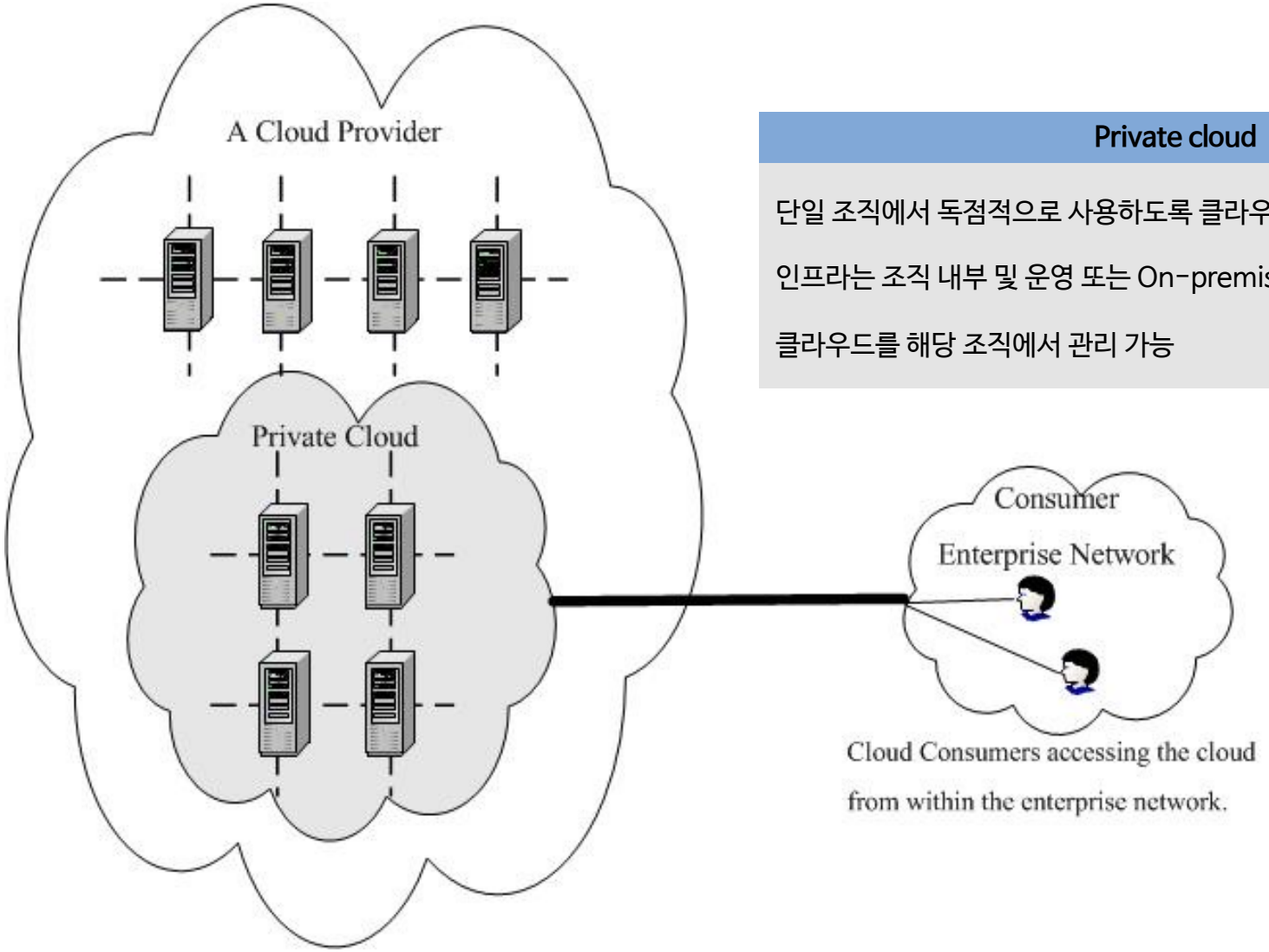
서버, 네트워킹, 데이터 센터, 스토리지 서비스 기능과 같은 컴퓨터 인프라를 아웃소싱 서비스 형태로 제공

BPaaS (Business Process as a Service)

클라우드 컴퓨팅 참조모델에 의한 전통적인 SaaS, PaaS, IaaS 외에 비즈니스 프로세스를 서비스 기능을 제공

BPM (workflows) RPA (robot)

2.3 Cloud Deployment Models



Private cloud

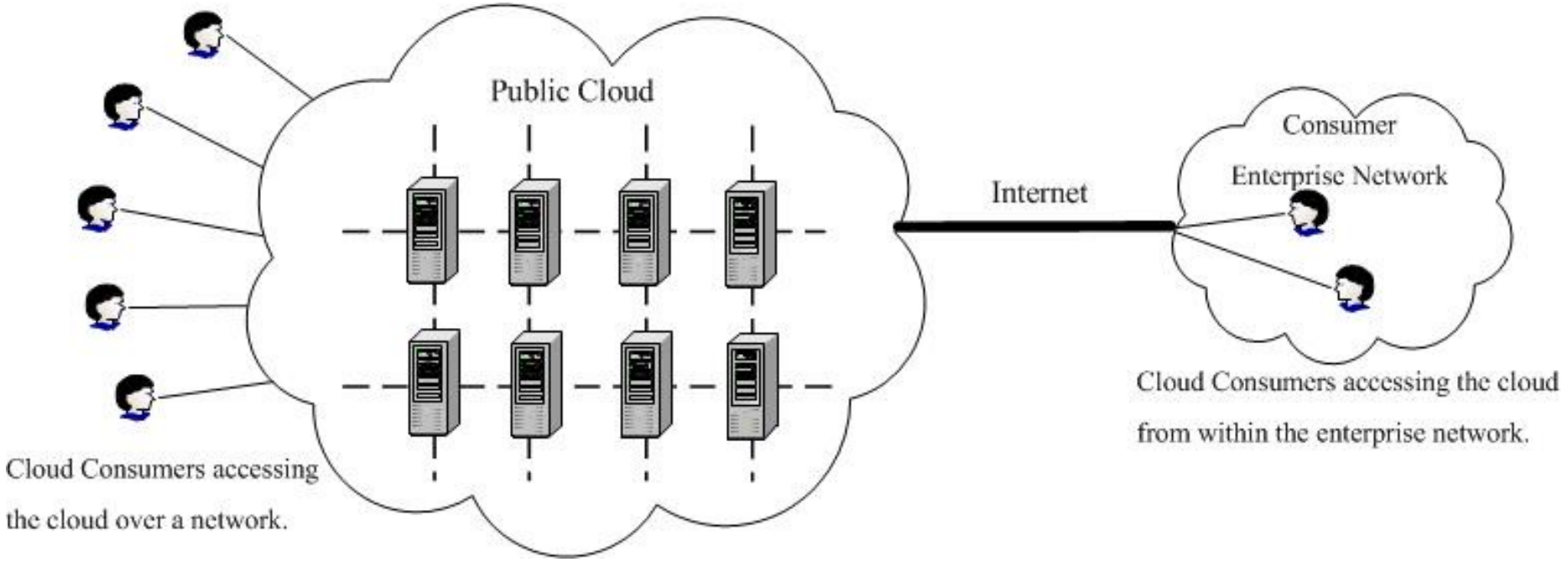
단일 조직에서 독점적으로 사용하도록 클라우드 인프라를 제공

인프라는 조직 내부 및 운영 또는 On-premise(자체 보유 전산실)

클라우드를 해당 조직에서 관리 가능

이미지 출처: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-322.pdf

2.3 Cloud Deployment Models



Public cloud

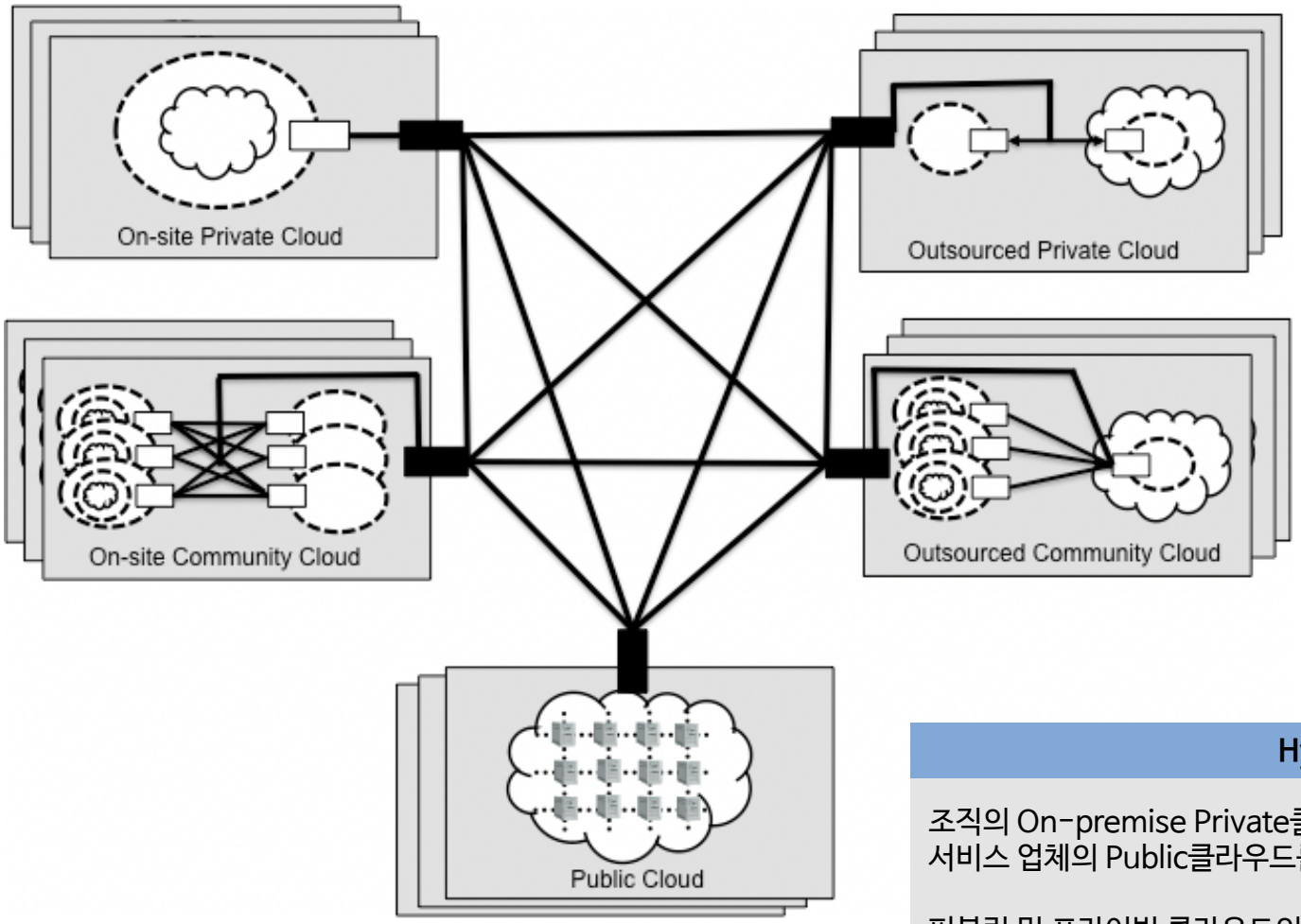
클라우드 서비스 제공 업체가 데이터 센터, 서버, 네트워킹 장비 및 스토리지와 같은 물리적 인프라를 소유
(인프라가 다른 회사와 공유될 수 있다.)

클라우드 서비스가 관리, 프로비저닝 및 유지 관리

사용자는 가상화 된 컴퓨팅, 네트워킹 및 스토리지 리소스에 서비스로 액세스합니다.

이미지 출처: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-322.pdf

2.3 Cloud Deployment Models



Hybrid cloud

조직의 On-premise Private클라우드를 사용하면서 다른 클라우드 서비스 업체의 Public클라우드를 사용하는 것.

퍼블릭 및 프라이빗 클라우드의 기능과 이점을 활용

이미지 출처: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-322.pdf

3.Cloud Architecture

3.1 Cloud 개발 모델

Cloud-Native Development

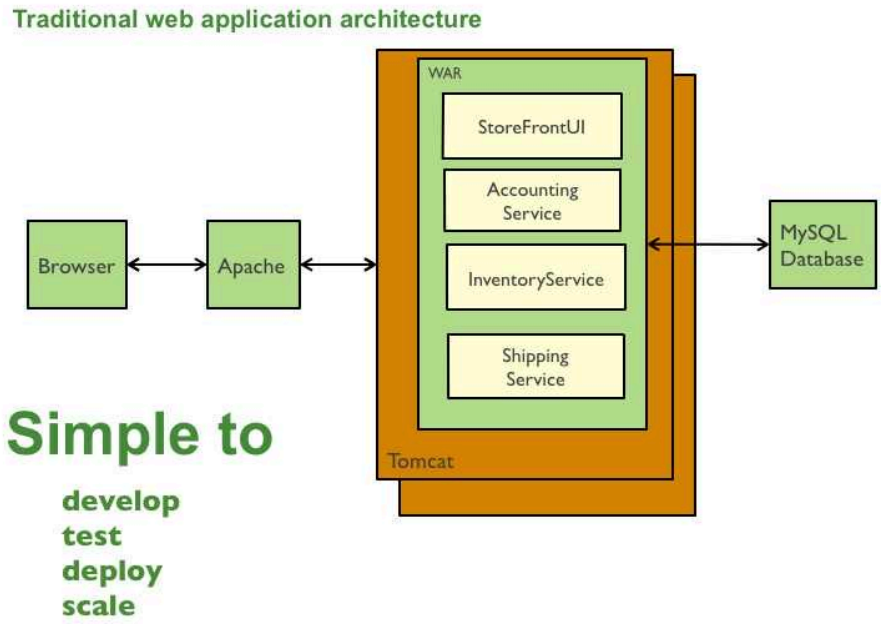
클라우드를 최대한 활용하고 활용하기 위한 목표로 설계되었습니다.
클라우드와 통신하고 애플리케이션을 배포할 때 더 나은 실행을 가능하게 하는 서비스가 필요합니다.
클라우드 네이티브는 클라우드에서 애플리케이션을 설계, 구축 및 실행하는 데 도움이 됩니다.
지속적인 통합, 컨테이너 엔진 및 오케스트레이터로 구성됩니다.
이러한 구성 요소는 더 빠른 속도, 더 많은 확장성 및 최적화된 관리 비용을 장려하기 위해 클라우드 네이티브로 고안되었습니다.
CoreDNS, Prometheus, Kubernetes 및 Fluentd와 같은 애플리케이션에는 클라우드 컴퓨팅 기본 기반에서 관리하는 자체 오픈 소스 제품 세트가 있습니다.

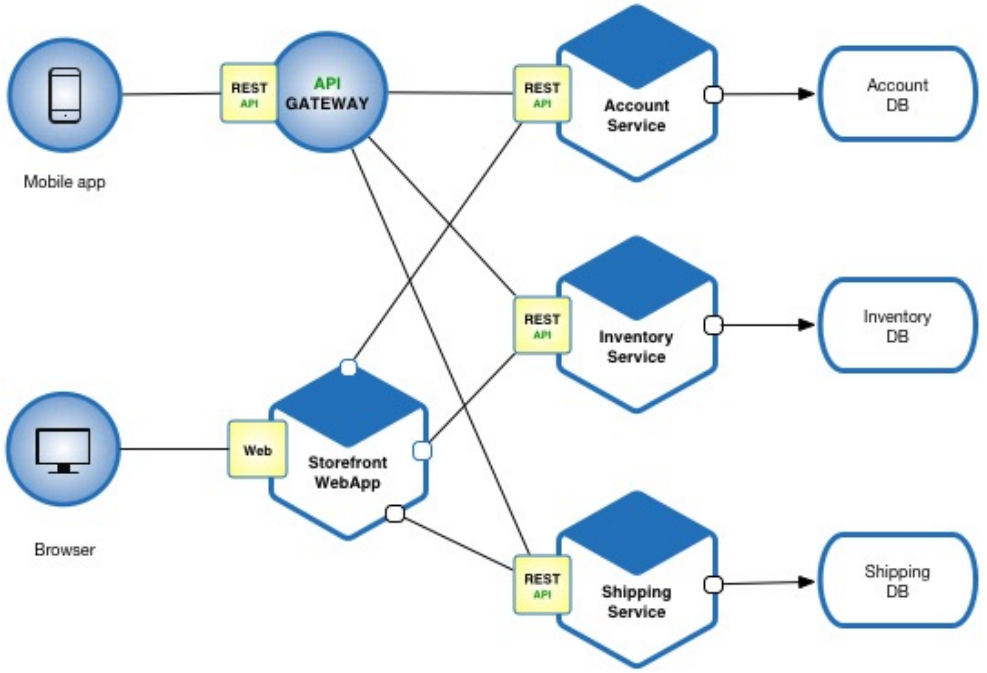
Cloud-Based Development

클라우드 플랫폼은 IT 및 소비자 만족도를 높이는 클라우드 컴퓨팅, 네트워킹, 스토리지 및 비즈니스 활용의 매력적인 조합입니다.
Microsoft는 Azure 플랫폼을 통해 리소스를 빠르게 추적 할 수 있습니다.
Google은 Google Cloud Platform을 출시하여 이러한 과제에 대비하기 시작했습니다.
클라우드는 서비스와 특징에 대해 완전한 명령을 내리므로 의도적으로 사용자가 일상적인 기능을 방해하지 못하도록 합니다.

출처: https://www2.architech.ca/difference-between-cloud-native-cloud-based-and-cloud-enabled-development/

3.2 Monolithic vs Microservice Architecture

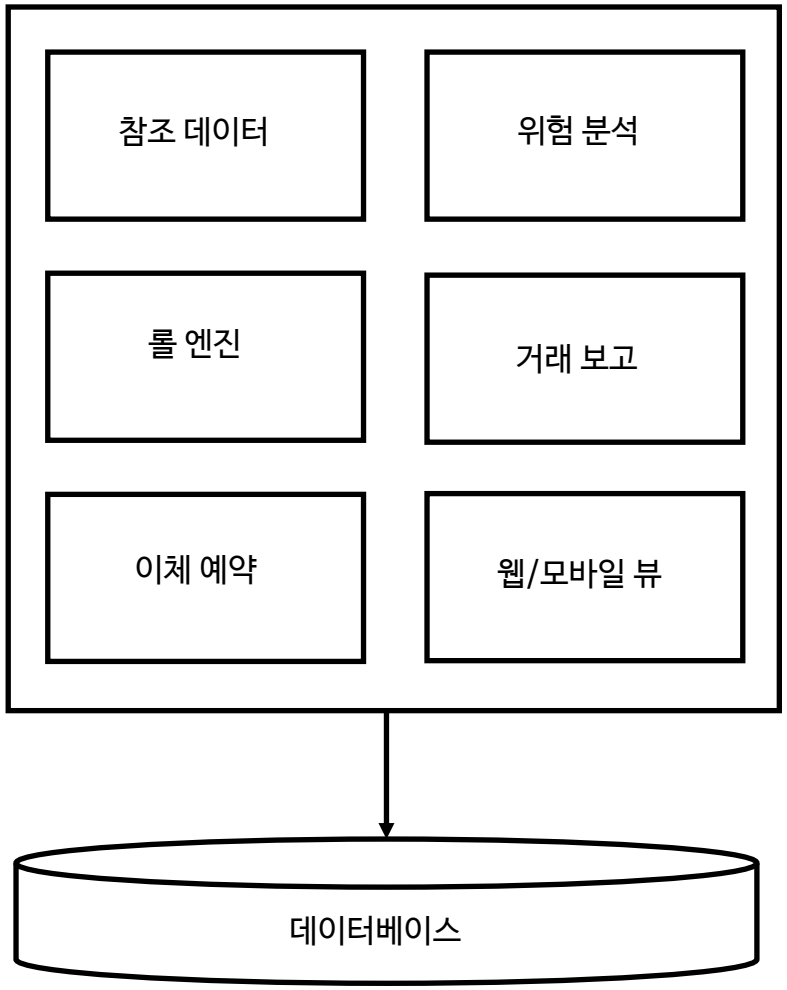




	모놀리스	마이크로 서비스
장점	• 간단한 개발 • 간단한 배포 • 간편한 확장	• 크고 복잡한 서비스를 지속적으로 배포 가능 • 개발자의 이해도 향상 • 향상된 장애 격리 • 기술 스택 선택에 대한 유연성
단점	• 개발 생산성 저하 • 지속적인 배포의 어려움 • 각 구성요소의 독립적 확장 불가 • 새로운 기술의 채택이 어려움	• 분산 시스템의 복잡성 처리 • 배포 복잡성 • 높은 메모리 소모

이미지 출처: https://microservices.io/patterns/monolithic.html

3.3 Monolithic Architecture



모놀리스 아키텍처(거래 애플리케이션)

장점

- 모든 기능은 하나의 소프트웨어 애플리케이션으로 컴파일 되고 배포
- 소규모로 시작
- 전체 애플리케이션은 응집력이 높기 때문에 초기 단계에서 빠르게 개발
- 전체 애플리케이션은 하나의 시스템이므로 배포하고 테스트 하기 쉬움

단점

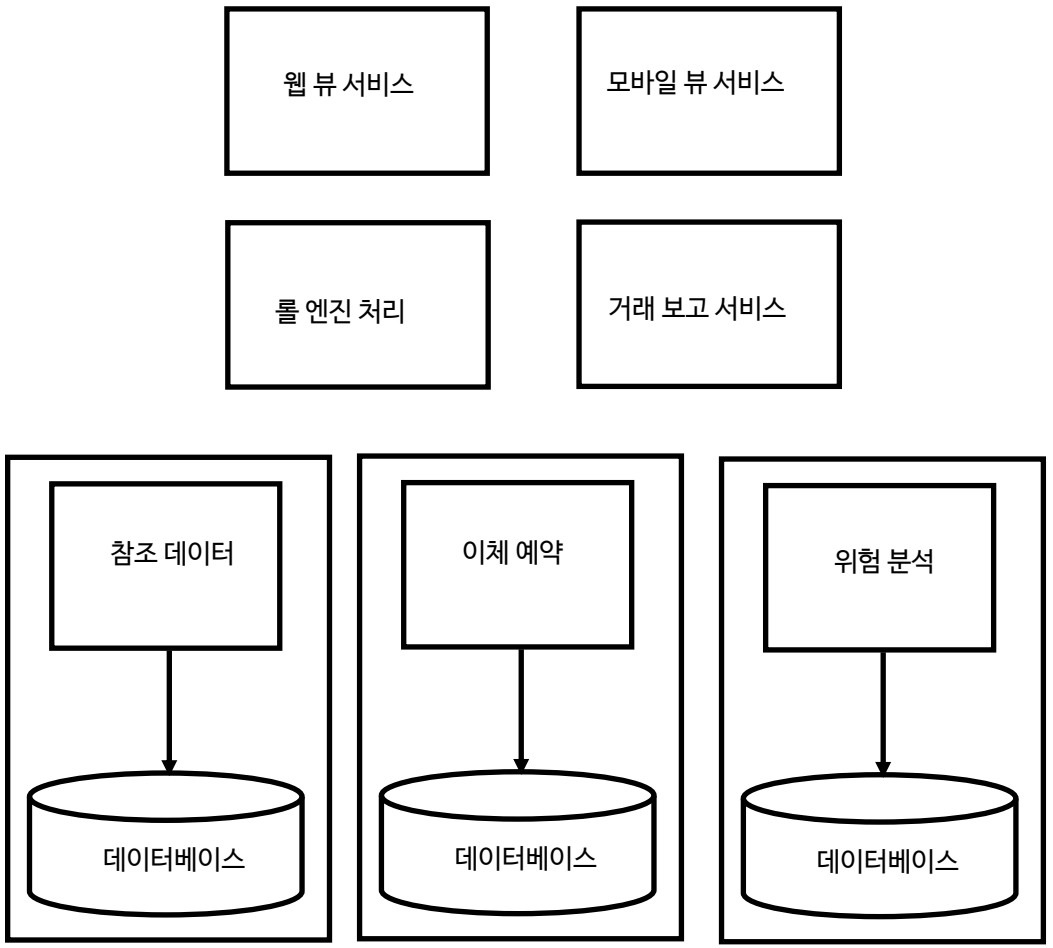
- 애플리케이션이 유기적으로 발전함에 따라 유지 관리, 운영이 어려움
- 시간이 지남에 따라 여러 개발팀에서 유지관리
- 각 팀은 애플리케이션의 각 하위 시스템을 담당
- 하위 시스템은 고도로 결합, 일정기간동안 새로운 기능을 사용할 수 있도록 개발 팀 간에 상호 의존성 존재

하나의 시스템에 따른 문제점

- 빠른 기능 롤아웃
- 더 이상 사용되지 않는 스택
- 가파른 학습 곡선
- 스케일링 문제

출처: Istio로 시작하는 서비스 메시(출판사 에이콘)

3.4 Microservice Architecture



마이크로서비스 아키텍처(거래 애플리케이션)

민첩성

- 아키텍처가 제공하는 느슨한 결함으로 개발 가속화
- 복원력과 장애 분리를 촉진
- 특정 서비스를 독립적으로 수정 배포(서비스 품질)

혁신성

- 소규모 독립 개발 팀이 서비스 범위 내에서 완전한 소유권 획득
- 서비스별 적절한 도구와 프레임워크 선택 가능
- 조직의 기술 스택과 혁신을 향상

확장성

- 필요 서비스만 확장 가능
  (느슨한 결함으로 트랙픽 분석 가능)

유지 보수성

- 모놀리스 소프트웨어에 가파른 학습 곡선 해결
- 기술 부채 해결 가능

출처: Istio로 시작하는 서비스 메시(출판사 에이콘)

3.5 Microservice Architecture 도전 과제

❑ 신뢰할 수 있는 네트워크

- 분산 아키텍처에는 동작하는 부분이 많으며, 서비스 중 하나가 언제든지 실패할 가능성이 높음
- 서비스를 구축하는 동안 필요한 탄력성 패턴 추가(예: 시스템 이중화)

❑ 네트워크 지연 시간 없음

- 네트워크 통신은 로컬 통신보다 느림
- 부하로 서비스가 느리게 응답
- 서비스 지연 테스트는 매우 어려움(개발자의 성숙도가 중요)
- 회로 차단기 패턴 도입, 데이터 캐싱

❑ 무한한 대역폭

- 프로덕션에 배포되는 서비스 수가 기하 급수적으로 증가
- 애플리케이션에 할당량 할당 및 소비 추적을 위한 매커니즘 필요
- 서버 측 로드 밸런서에 많은 부하로 성능 저하

출처: Istio로 시작하는 서비스 메시(출판사 에이콘)

3.5 Microservice Architecture 도전 과제

❑ 안전한 네트워크

- 많은 기술적 선택으로 팀은 각각의 버그 수정/문제를 추적
- 서비스 간 통신을 제어(서비스 수준 인증으로 알 수 없는 서비스 연결을 필터링)
- 일반 텍스트 교환을 수행하는 애플리케이션은 중요한 데이터 노출(보안 프로토콜 사용)

❑ 변하지 않는 토폴로지[1)]

- 서비스간의 종속성이 낮음(신속하게 구축, 릴리즈, 배포 가능)
- 동적으로 추가된 다른 서비스 연결 필요(서비스 디스커버리 기능 추가로 전체 시스템의 탄력성 향상)

❑ 한 명의 관리자

- 소수의 사람이 분산 시스템에 완전한 운영 지식을 갖는 것은 불가능함
- 마이크로서비스는 팀을 자율적으로 만드는 것이 목표
- 각 팀은 자체 서비스를 유지 관리할 책임이 있음

토폴로지(topology)[1)]: 컴퓨터 네트워크의 요소들을 물리적으로 연결해 놓은 것, 또는 그 연결 방식을 말한다.

3.5 Microservice Architecture 도전 과제

❑ 전송 비용 없음

- 마이크로서비스에서는 종속된 서비스를 많이 호출
- 데이터 교환 비용 발생(직렬화, 역직렬화)
- SOAP/XML 〉 JSON 〉 바이너리 프로토콜

❑ 동일 네트워크

- 모든 시스템이 동일한 하드웨어 세트에서 실행되고 모든 애플리케이션이 표준 프로토콜로 통신 하는 경우 네트워크가 같다고 분류
- 애플리케이션이 배포될 때마다 동일한 양의 리소스를 얻도록 제한

❑ 인프라스트럭처

- 마이크로서비스 아키텍처는 모놀리스 아키텍처보다 훨씬 복잡
- 비용 효율적으로, 필요에 따라 인프라를 프로비저닝[2] 할 수 있는 방법을 강구해야 함

프로비저닝(provisioning) [2]: 사용자의 요구에 맞게 시스템 자원을 할당, 배치, 배포해 두었다가 필요 시 시스템을 즉시 사용할 수 있는 상태로 미리 준비해 두는 것을 말한다.

3.5 Microservice Architecture 도전 과제

❑ 모니터링과 디버깅

- 하나의 기능을 위한 많은 구성 요소가 있음
- 자동화, 효율화 되어 메트릭[3] 을 포착하는 것이 중요
- 서비스를 확장 축소 할 때 모니터링에서 서비스를 검색하고 관련 조치를 취해야 함

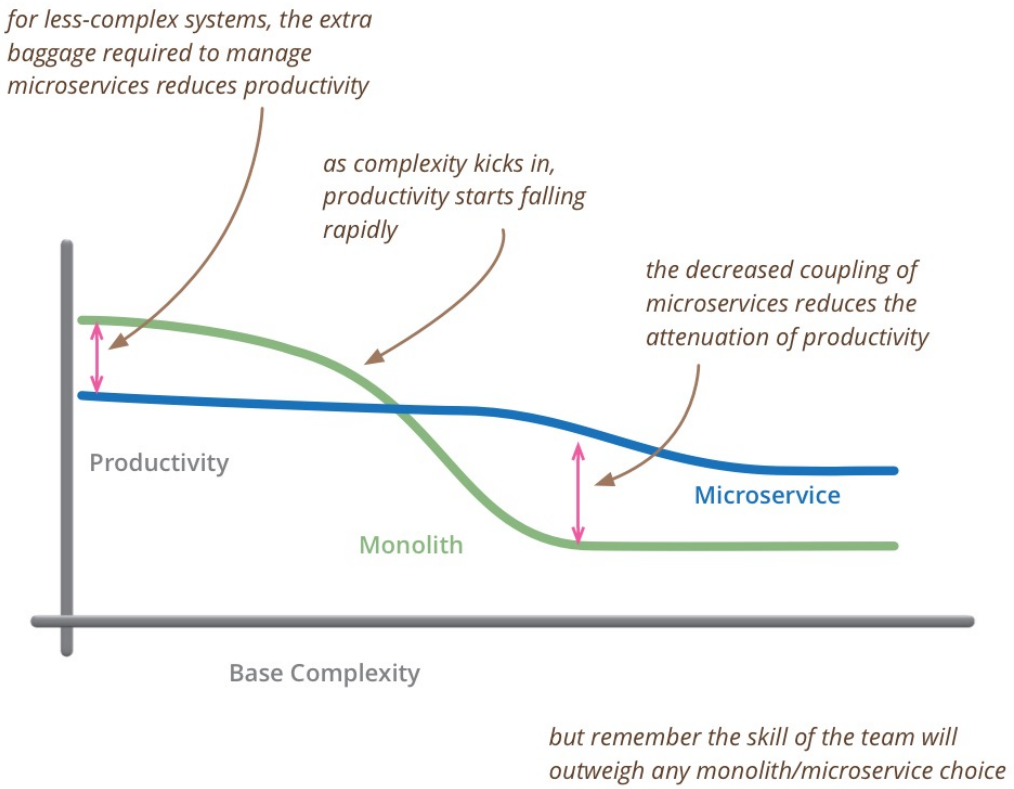
❑ 로그 처리

- 전체 시스템에서 많은 로깅을 수행하여 시스템 공간이 빨리 소진되어 로그 보관이 필요
- 여러 서비스 간의 로그를 신속하게 조회하고 연관 시킬 수 있어야 함

메트릭(metrics)[3]: 키 값 쌍으로 캡처된 단순한 숫자 측정

4.고려 사항

4.1 Microservice 생산성



마이크로 서비스 사용 여부의 핵심은 고려중인 시스템의 **복잡성**입니다.

마이크로 서비스를 사용할 때 자동화 된 배포, 모니터링, 장애 처리, 최종 일관성 및 분산 시스템이 도입하는 기타 요인에 대해 작업해야 합니다

모놀리식으로 관리하기에 특별히 복잡한 시스템을 운영할 상황이 아니면 마이크로서비스는 고려할 필요조차 없다.

– 마틴 파울러 –

이미지 출처: https://martinfowler.com/bliki/MicroservicePremium.html

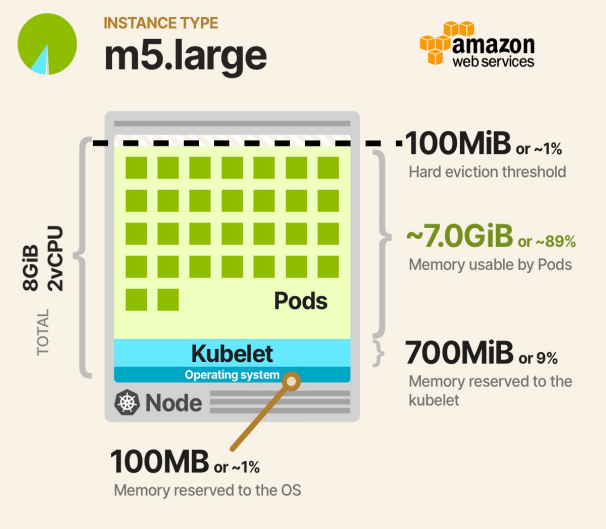
4.2 Microservice 성숙도 모델

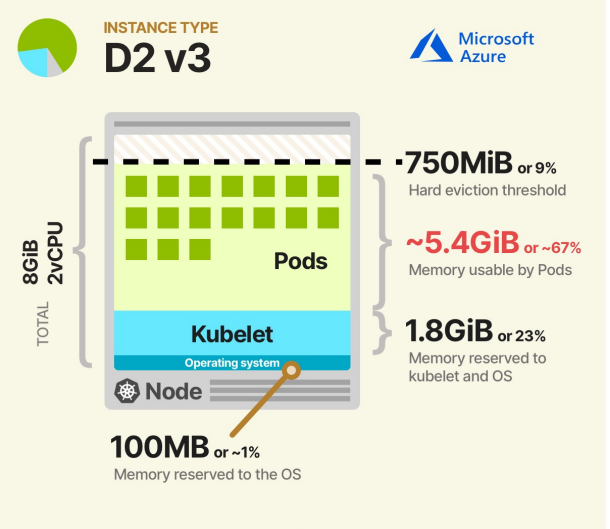
	Level0 Traditional	Level1 Basic	Level2 Intermediate	Level3 Advanced
Application	Monolithic	Service Oriented Integrations	Service Oriented Applications	API Centric
Database	One Size Fit All Enterprise DB	Enterprise DB + No SQLs and Light databases	Polyglot, DBaaS	Matured Data Lake /Near Realtime Analytics
Infrastructure	Physical Machines	Virtualization	Cloud	Containers
Monitoring	Infrastructure	App & Infra Monitoring	APMs	APM & Central Log Management
Process	Waterfall	Agile and CI	CI & CD	DevOps

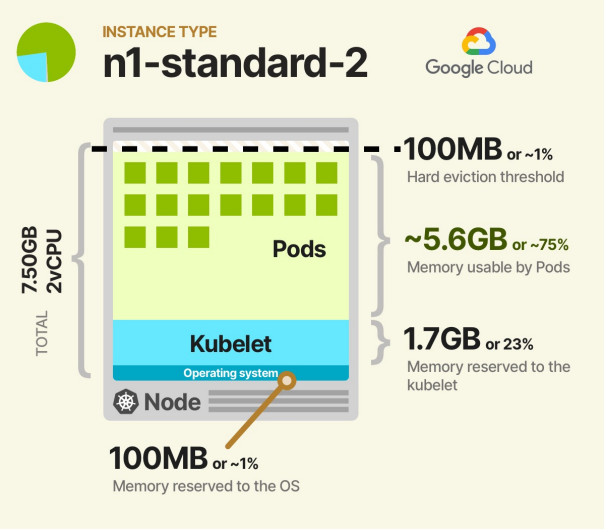
출처: A Microservices Maturity Model, from “Spring 5.0 Microservices,” 2nd Edition”by Rajesh R V (O’Reilly)

4.3 Kubernetes 기반 메모리 산정 시 고려사항

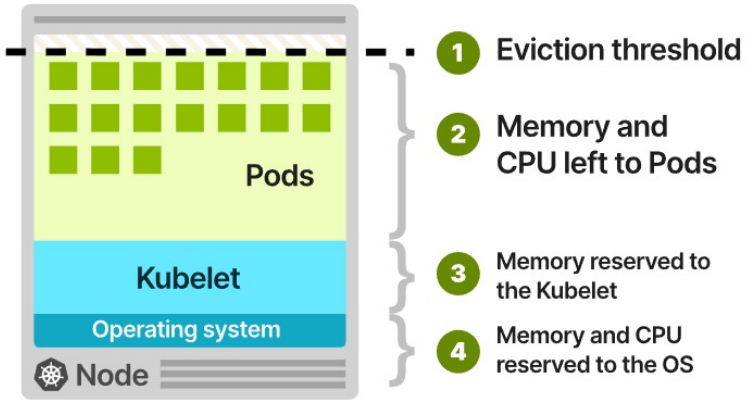
❑ Kubernetes 노드의 할당 가능한 메모리 및 CPU







❑ 클러스터 노드에서 리소스가 할당되는 방법



1. OOM 방지를 위해서 예약된 리소스
2. 파드에서 사용할수 있는 리소스
3. Kubetlet, 컨테이너 런타임 등과 같은 Kubernetes 에이전트를 실행하는 데 필요한 리소스
4. SSH, system 등과 같은 운영 체제 및 시스템 데몬을 실행하는 데 필요한 리소스

이미지 출처: https://learnk8s.io/allocatable-resources

4.4 Cloud Native Stack

Code	• Back-end (Go, Java(Spring boot, jpa), meteor, Node.js, Python, php, Scala) • Front-end (React, Vue.js, Angular, jQuery, Xplatform, nexacro)
Tooling & Management	• Chef • Puppet • Ansible • SaltStack • Deis (Engine Yard) • Glider Labs • CircleCI • TravisCI • Bouyant.io • WeaveWorks • SysDig • Panamax (CenturyLink) • CloudNative • Wercker • Shippable • Brooklyn(Apache) • Giant Swarm • DCHQ.io • Nirmata • Cloud66 • StackEngine • Convox.io • Magnetic.io • Dozensmore…
Orchestration : Scheduling & Cluster Management	• Kubernetes (Google/CoreOS) • Mesos, Marathon(Mesosphere) • Swarm, Machine, Compose (Docker) • Fleet(CoreOS) • Serf, Terraform, Atlas (Hashicorp) • Helios (Spotify) • Project Titan (Netflix) • Chronos (AirBnB) • Aurora (Apache) • Cloudify (Gigaspaces) • Magnum+Heat (Open Stack)
Service Discovery	• Consul (Hashicorp) • etcd (CoreOS) • Eurika (Netflix) • Zookeeper (Apache) • SmartStack (AirBnB) • Mesos-DNS (Mesosphere)
Container Engine	• libcontainer (Docker) • runC (Open Container Foundation) • appC (CoreOS) • Ubuntu LXD (Canonical) • Drawbridge (Microsoft) • LXC/libvirt (Red Hat)
Minimal OS	• CoreOS (CoreOS) • Project Atomic (Red Hat) • Photon (Vmware) • RancherOS (Rancher) • Snappy Ubuntu Core (Canonical) • Windows Nano Server (Microsoft)

출처: https://www.slideshare.net/SeongBokLee/cloud-native-application-67952084

4.5 Legacy → Cloud 전환

- **VM(Virtual Machine)[4] 으로 전환**
  - Java EE(EJB) 기반에서 WAS 변경
  - EJB기반은 WAS에 종속적인 기능들을 다수 포함
- **JAVA 버전 변경 시**
  - 기존 솔루션이 변경하려는 JAVA 버전 지원여부
- **솔루션의 라이선스 정책 확인**
  - Core당 비용 책정 시 라이선스 비용 과다 청구(cloud에서는 core를 share하여 사용)
  - Cloud용 라이선스 정책이 존재하여 비용이 올라갈 가능성 존재
- **시스템의 형상관리**
- **운영자의 시스템 아키텍처 이해도**
- **첨부파일, DB 용량**
  - 시스템 open을 위한 Cut-over[5] 시간 판단에 주요 근거

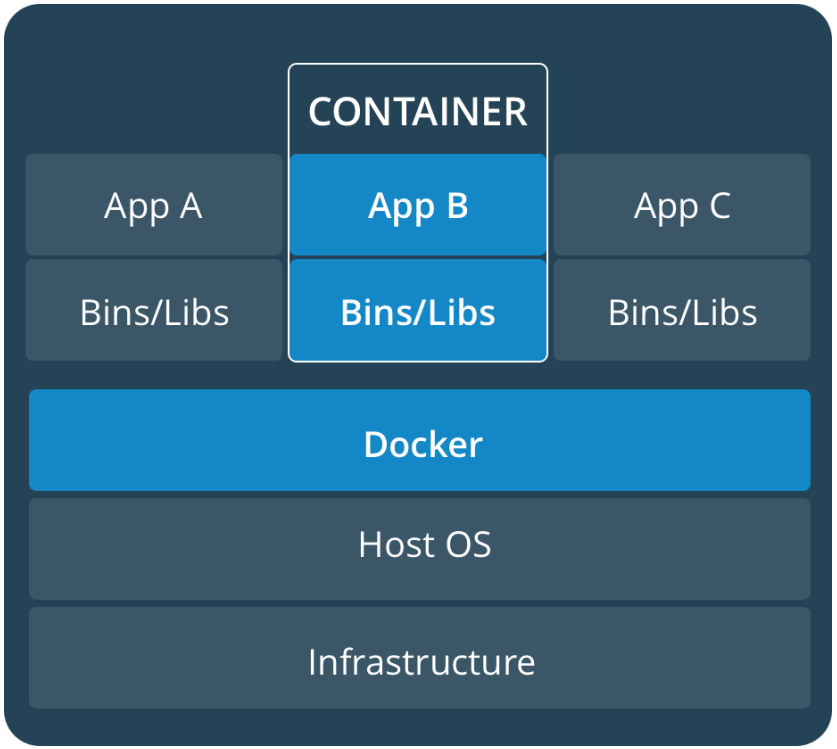
가상 머신(Virtual Machine, VM) 4): 물리적 하드웨어 시스템(오프프레미스 또는 온프레미스에 위치)에 구축되어 자체 CPU, 메모리, 네트워크 인터페이스 및 스토리지를 갖추고 가상 컴퓨터 시스템으로 작동하는 가상 환경

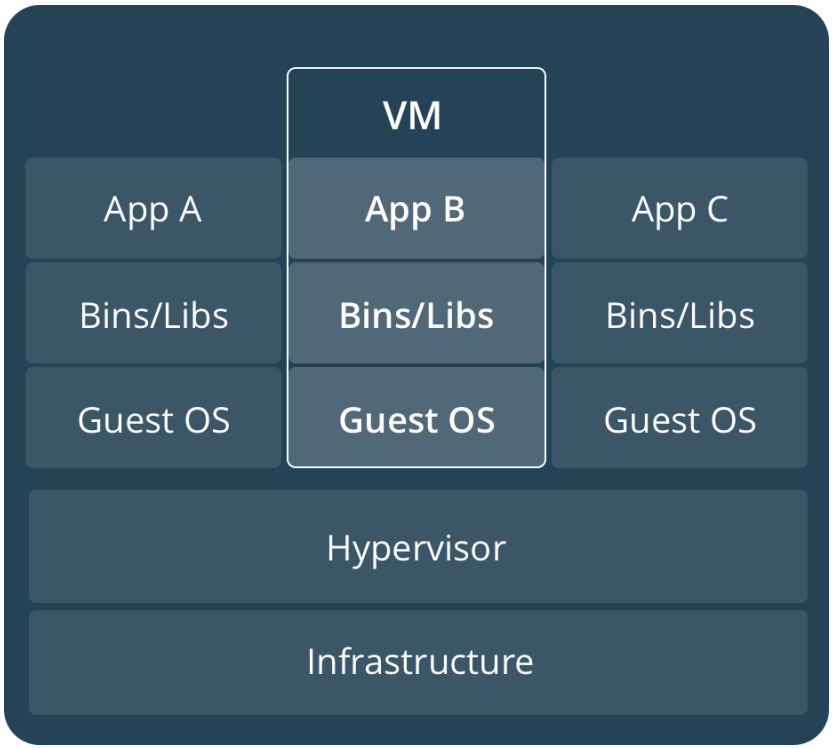
Cut-over5): 기존에 운영되던 정보시스템을 완전히 중단시키고, 새로 구축된 정보시스템을 본격 오픈하는 것을 의미

5. 표준프레임워크를 활용한 Kubernetes

❑ Docker 정의

- 개발자와 관리자가 컨테이너를 사용하여 애플리케이션을 개발, 배포 및 실행하기 위한 플랫폼
- 리눅스 컨테이너를 사용하여 응용 프로그램을 배포하는 것을 컨테이너화라고 함
- 컨테이너는 새로운 것은 아니지만, 애플리케이션을 쉽게 배치하기 위해 사용
- Containers and Virtual Machines





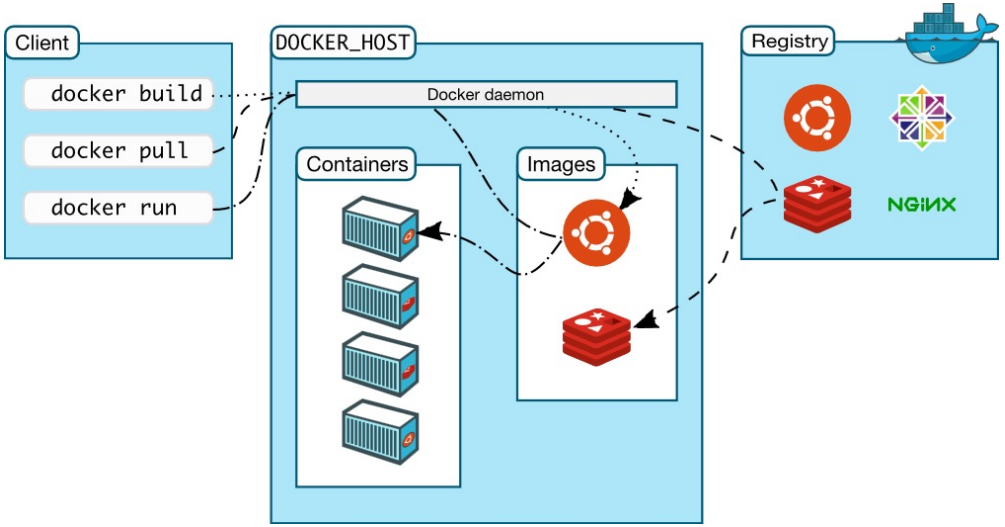
이미지 출처: https://docs.docker.com/get-started/

❑ IMAGES

- 도킹된 컨테이너를 생성하기 위한 지침이 포함된 읽기 전용 템플릿
- 종종 추가적인 사용자 지정과 함께 다른 이미지를 기반으로 함
- 다른 가상화 기술에 비해 이미지를 매우 가볍고 작고 빠르게 만드는 요소 중 하나임

❑ CONTAINERS

- 실행 가능한 이미지의 인스턴스
- DockerAPI또는 CLI를 사용하여 컨테이너를 생성, 시작, 중지, 이동 또는 삭제
- 하나 이상의 네트워크에 연결하거나, 컨테이너에 스토리지를 연결하거나, 현재 상태에 따라 새 이미지를 만들 수도 있음



이미지 출처: https://docs.docker.com/engine/docker-overview/

❑ DOCKERFILE

- 이미지를 생성하기 위한 스크립트

```
FROM openjdk:8-jre-alpine
LABEL maintainer=〈dasomell@gmail.com〉
RUN ln -snf /usr/share/zoneinfo/Asia/Seoul /etc/localtime && echo Asia/Seoul 〉 /etc/timezone

COPY ./build/libs/sample-boot2.war app.war
ENTRYPOINT ["java","-server","-D64","-Djava.security.egd=file:./dev/urandom","-jar","/app.war"]
EXPOSE 8080
```

Docker를 구동하기 위한 base image

Time Zone 설정

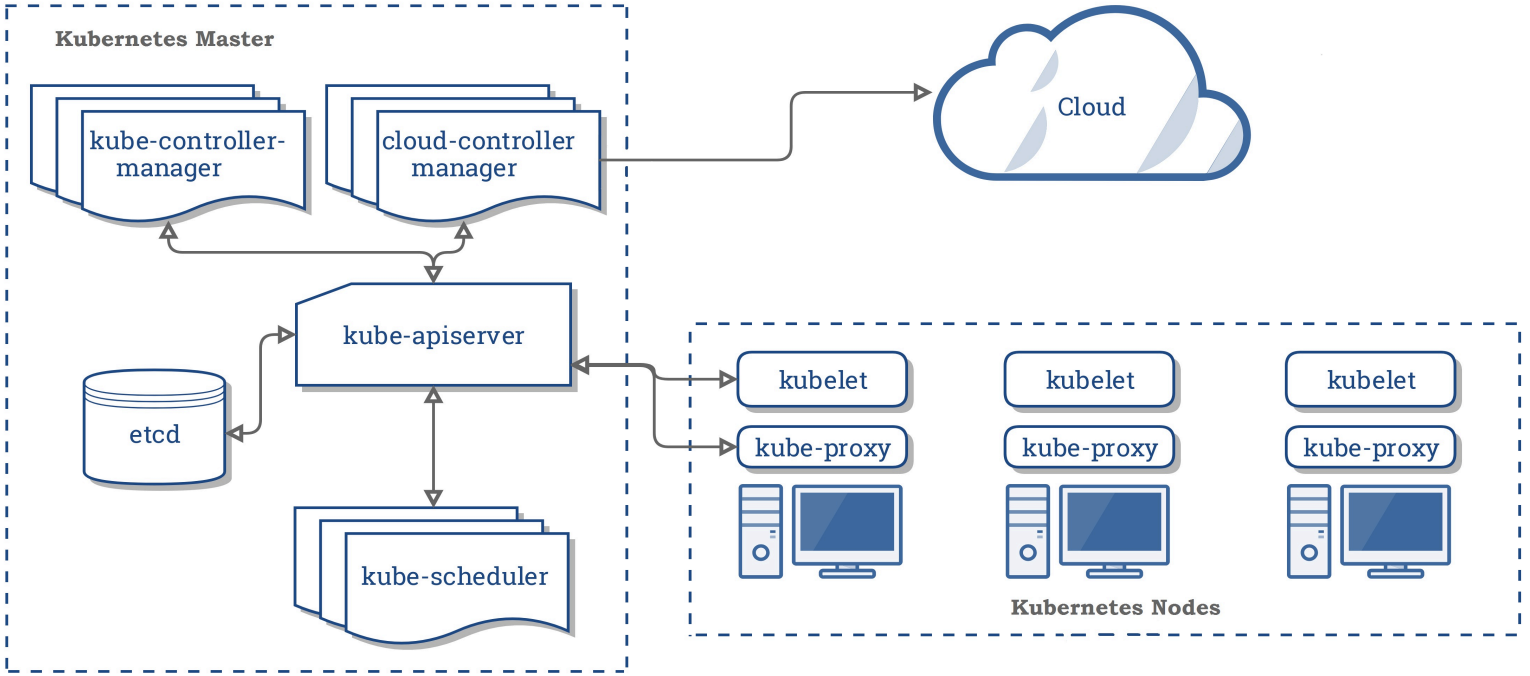
파일 복사

해당 포트 노출

컨테이너가 시작되었을 때 스크립트 혹은 명령을 실행

❑ Kubernets 정의

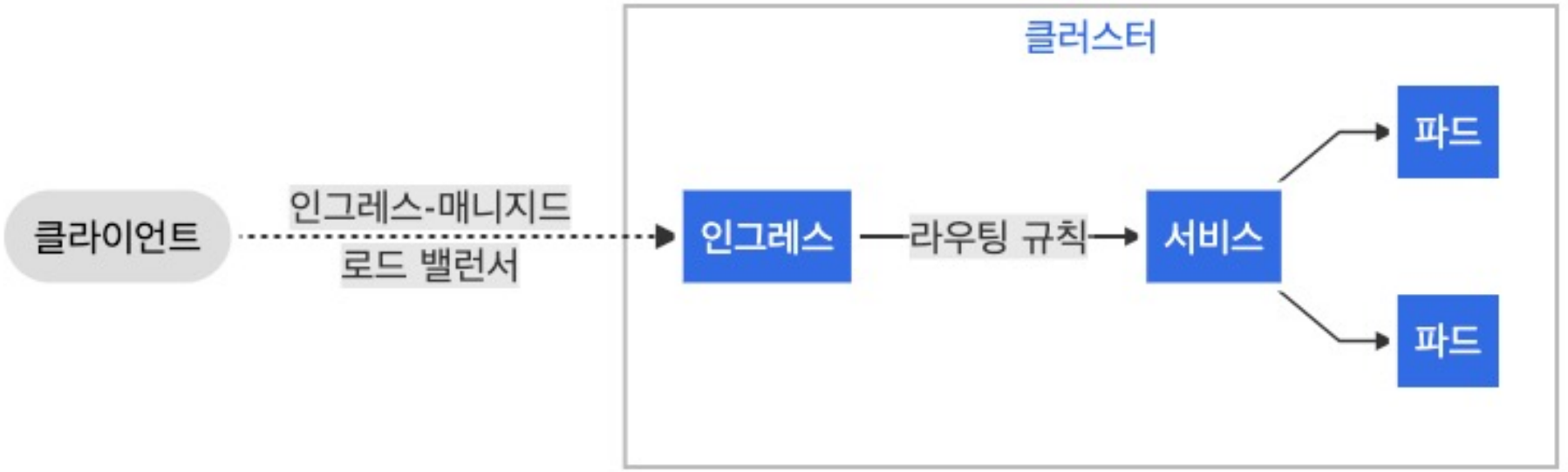
- 쿠버네티스(K8s)는 디플로이 자동화, 스케일링, 컨테이너화된 애플리케이션의 관리를 위한 오픈 소스 시스템으로서 원래 구글에 의해 설계되었고 현재 리눅스 재단에 의해 관리되고 있다.
- 목적은 여러 클러스터의 호스트 간에 애플리케이션 컨테이너의 배치, 스케일링, 운영을 자동화하기 위한 플랫폼을 제공하기 위함이다.
- 도커를 포함하여 일련의 컨테이너 도구들과 함께 동작한다. (위키백과)



이미지 출처: https://kubernetes.io/ko/docs/concepts/overview/components/

❑ Ingress

- 인그레스는 클러스터 외부에서 클러스터 내부 서비스로 HTTP와 HTTPS 경로를 노출한다.
  트래픽 라우팅은 인그레스 리소스에 정의된 규칙에 의해 컨트롤된다.
- 인그레스는 외부에서 서비스로 접속이 가능한 URL, 로드 밸런스 트래픽, SSL / TLS 종료 그리고 이름 기반의
  가상 호스팅을 제공하도록 구성할 수 있다.
  인그레스 컨트롤러는 일반적으로 로드 밸런서를 사용해서 인그레스를 수행할 책임이 있으며, 트래픽을 처리하는데 도움이
  되도록 에지 라우터 또는 추가 프런트 엔드를 구성할 수도 있다.



이미지 출처: https://kubernetes.io/ko/docs/concepts/services-networking/ingress/

5.3 Cloud 환경구성

❑ Ingress

Ingress 오브젝트 명시

Ingress 이름

이 오브젝트를 생성하기 위해 사용하고 있는 쿠버네티스 API

이름 기반의 가상 호스팅
이름 기반의 가상 호스트는 동일한 IP 주소에서 여러 호스트 이름으로 HTTP 트래픽을 라우팅하는 것을 지원한다.

팬아웃(fanout)
팬아웃 구성은 HTTP URI에서 요청된 것을 기반으로 단일 IP 주소에서 1개 이상의 서비스로 트래픽을 라우팅

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: opdc-ingress
spec:
  rules:
  - host: opdc.io
    http:
      paths:
        - path: /war
          backend:
            serviceName: egov-sample
            servicePort: 8080
        - path: /maven
          backend:
            serviceName: sample-boot
            servicePort: 8080
  - host: opdc2.io
    http:
      paths:
        - path: /gradle
          backend:
            serviceName: sample-boot2
            servicePort: 8080
```

출처: https://kubernetes.io/ko/docs/home/

❑ Service

- Kubernetes의 서비스는 포드와 유사한 REST 객체입니다.
  모든 REST 오브젝트와 마찬가지로 서비스 정의를 API 서버에 POST하여 새 인스턴스를 작성할 수 있습니다.

```
apiVersion: v1
kind: Service
metadata:
  name: egov-sample
spec:
  selector:
    app: egov-sample
  ports:
  - port: 8080
    targetPort: 8080
```

Service에서 사용하는 포트

Service에 연결될 Pod가 사용하는 포트

출처: https://kubernetes.io/ko/docs/home/

❑ Deployment

- 디플로이먼트 는 파드와 레플리카셋에 대한 선언적 업데이트를 제공한다.

- 디플로이먼트에서 의도하는 상태를 설명하고, 디플로이먼트 컨트롤러(Controller) 는 현재 상태에서 의도하는 상태로 비율을 조정하며 변경한다.
  새 레플리카셋을 생성하는 디플로이먼트를 정의하거나 기존 디플로이먼트를 제거하고, 모든 리소스를 새 디플로이먼트에 적용할 수 있다.

Pod 복제 개수(레플리카 파드)

Deployment history(기본 10개)

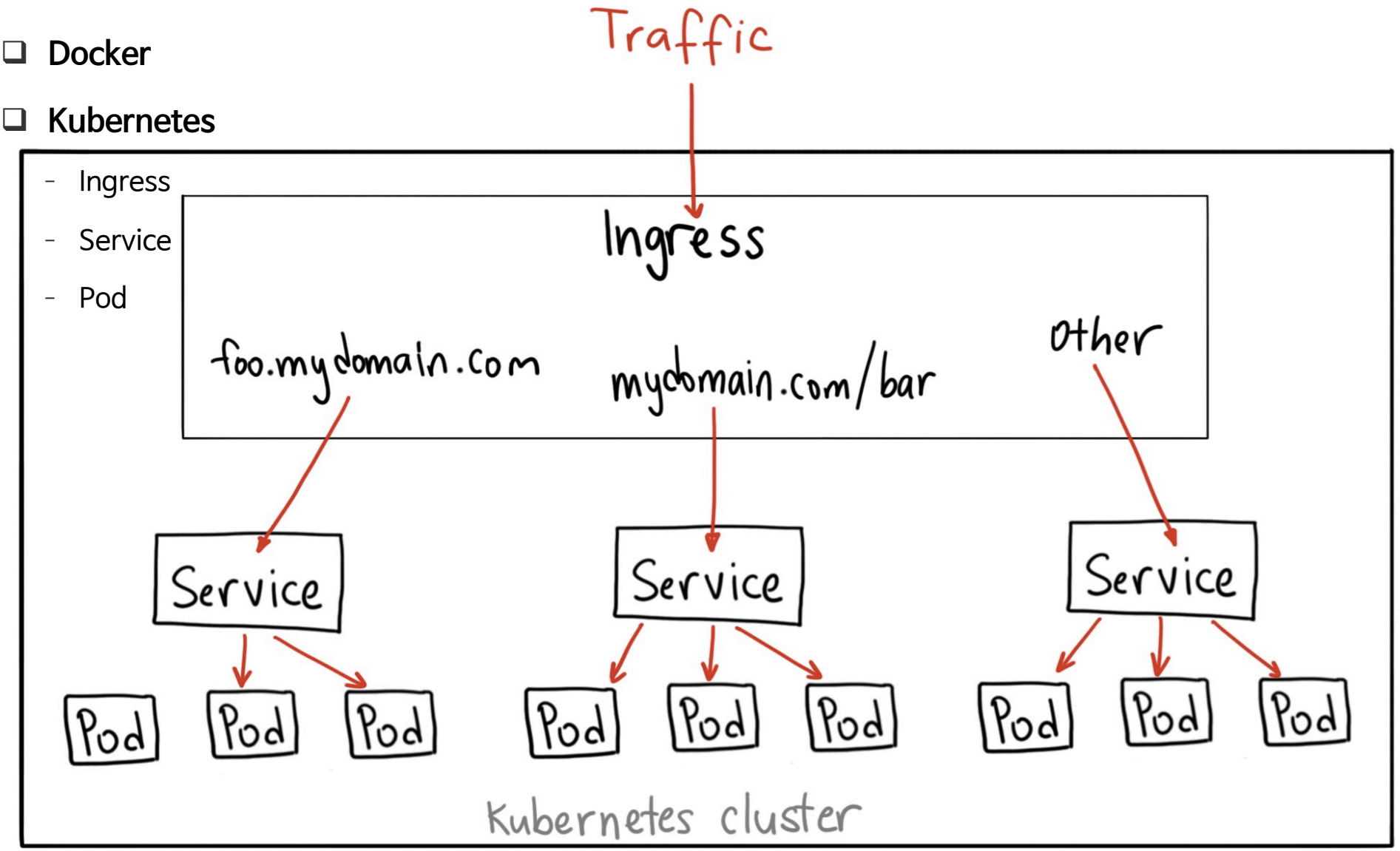
배포 이미지(Docker)

IfNotPresent 또는 Never: 로컬 이미지
Always가 기본값

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: egov-sample
  labels:
    app: egov-sample
spec:
  replicas: 1
  revisionHistoryLimit: 1
  selector:
    matchLabels:
      app: egov-sample
  template:
    metadata:
      labels:
        app: egov-sample
    spec:
      containers:
        - name: egov-sample
          image: localhost:5000/egov-sample:1.0.0
          ports:
            - containerPort: 8080
          imagePullPolicy: Always
          resources:
            requests:
              cpu: 0.5
              memory: 0.5Gi
            limits:
              cpu: 0.5
              memory: 0.5Gi
```

5.4 Cloud 테스트 환경

- ❑ Docker
- ❑ Kubernetes
  - Ingress
  - Service
  - Pod



이미지 출처: https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0

5.4 Cloud 테스트 환경

❑ Docker 설치

- https://www.docker.com/products/docker-desktop
- https://docs.docker.com/docker-for-windows/install/
- https://docs.docker.com/docker-for-mac/install/
- 정상 설치 확인
  - docker version

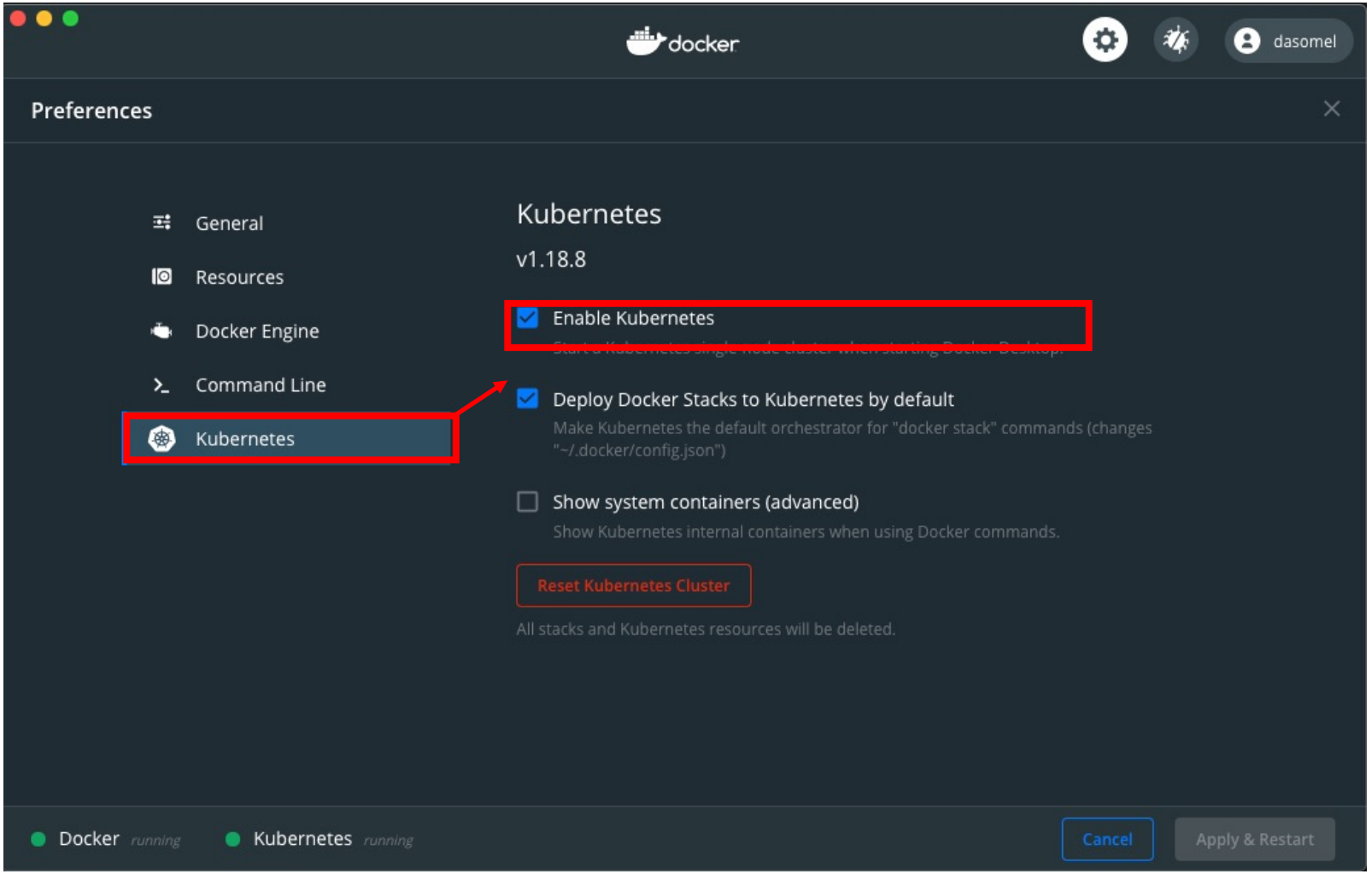
❑ Windows 10에 Linux용 Windows 하위 시스템 설치 가이드

- WSL2 (Windows Subsystem for Linux 2)
- 윈도우 사용자 필요시 참조
- https://docs.microsoft.com/ko-kr/windows/wsl/install-win10

5.4 Cloud 테스트 환경

❑ Kubernetes 설치

- docker-desktop 설치 후 Preferences → Kubernetes → Enable Kubernetes → Apply & Restart



5.4 Cloud 테스트 환경

❑ Infrastructure as Code

- 코드로서의 인프라스트럭처는 물리적 하드웨어 구성이나 인터페이스 구성 도구가 아닌 기계가 읽을 수 있는 정의 파일들을 통한 컴퓨터 데이터 센터의 관리 및 프로비저닝 과정이다(출처: 위키백과)

❑ Helm

- 정의: Kubernetes를 위한 패키지 매니저
- 설치 참고: https://helm.sh/ko/docs/intro/install/

❑ Dokcer Private Regstry 설치 및 배포상태 확인

- docker pull registry:latest
- docker run -d -p 5000:5000 --restart=always --name registry registry:2
- curl -X GET http://localhost:5000/v2/_catalog
- curl -X GET http://localhost:5000/v2/"repositories"/tags/list

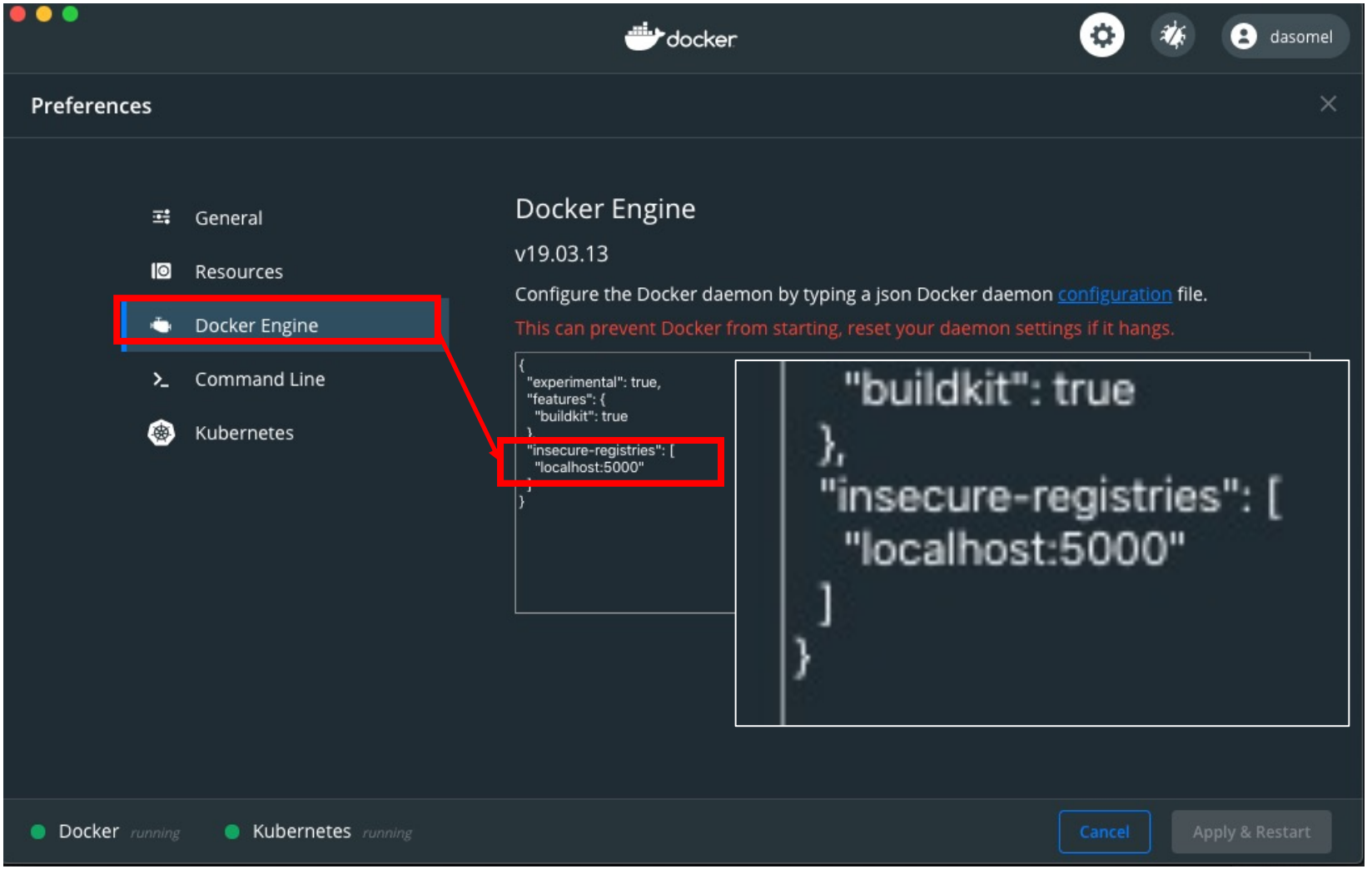
❑ Ingress Controller 설치

- helm repo add bitnami https://charts.bitnami.com/bitnami
- helm install ingress bitnami/nginx-ingress-controller

5.4 Cloud 테스트 환경

❑ Docker 환경설정 추가

- "insecure-registries": ["localhost:5000"]



5.4 Cloud 테스트 환경

❑ Windows 유틸리티

❑ cmder

- https://cmder.net

❑ chocolatey

install	shell
cmd.exe	@"%SystemRoot%₩System32₩WindowsPowerShell₩v1.0₩powershell.exe" -NoProfile -InputFormat None -ExecutionPolicy Bypass -Command "iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))" && SET "PATH=%PATH%;%ALLUSERSPROFILE%₩chocolatey₩bin"
powershell.exe	Set-ExecutionPolicy Bypass -Scope Process -Force; iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))

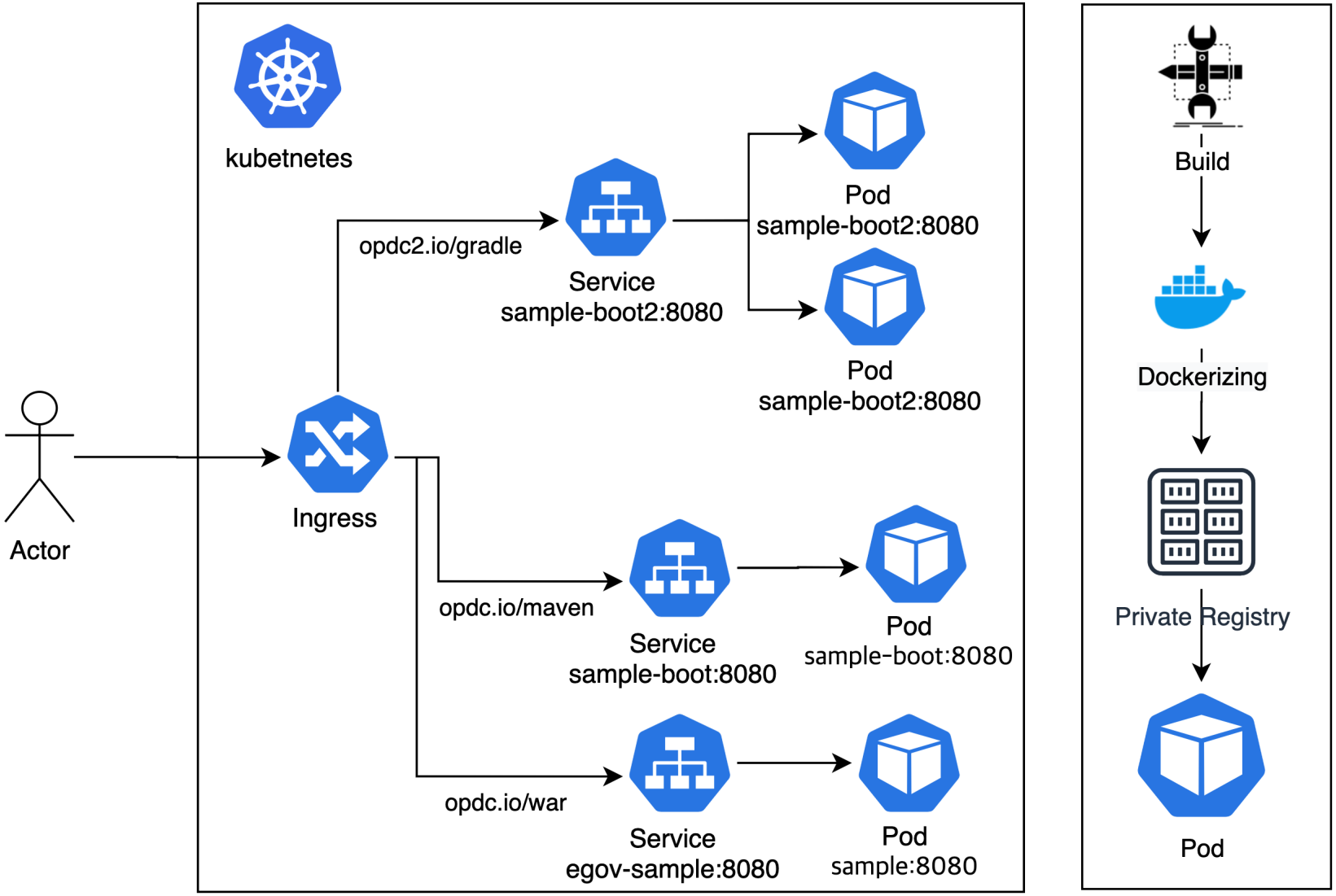
❑ Mac 유틸리티

❑ brew

- /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"

5.5 Cloud 테스트 환경 구축

❑ Architecture



5.5 Cloud 테스트 환경 구축

❑ sample 프로젝트(summary)

- mvn clean package
- docker build . -t localhost:5000/egov-sample:1.0.0
- docker push localhost:5000/egov-sample:1.0.0
- curl -X GET http://localhost:5000/v2/egov-sample/tags/list
- kubectl apply -f ./k8s/deployment.yaml
- kubectl apply -f ./k8s/service.yaml
- kubectl apply -f ./k8s/ingress.yaml
- hosts 파일 수정
  - 127.0.0.1 opdc.io
  - 127.0.0.1 opdc2.io
- http://opdc.io/war/egovSampleList.do

실습 간 한번만 실행하면 됨

5.5 Cloud 테스트 환경 구축

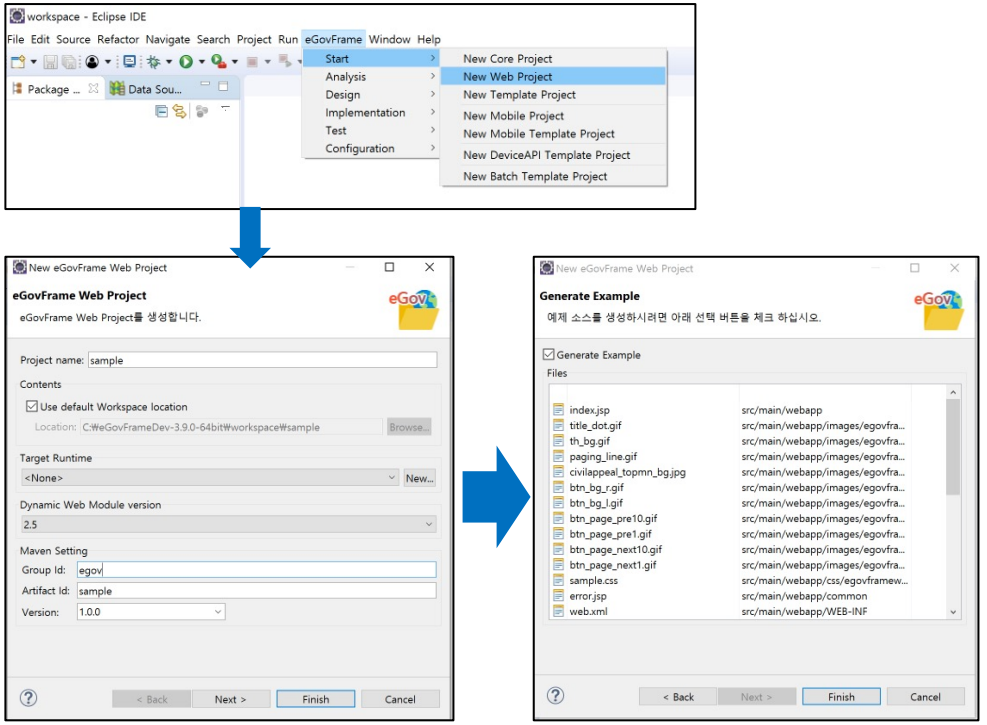
❑ sample 프로젝트(초기 설정)

- 기존 소스 수정없이 war파일을Tomcat에 배포
- 프로젝트 소스 생성

1. eGovFrame -〉 Start -〉 new Web Project
2. 프로젝트 정보 작성 -〉 Next

Project name	sample
Group Id	egov
Artifact Id	sample
Version	1.0.0

3. ‘Generate Example’ 클릭 -〉 Finish



- Maven을 활용하여 build, package(war)
  - mvn clean package
- 파일 생성(/Dockerfile)

```
FROM tomcat:9.0.39-jdk8-openjdk-buster
LABEL maintainer=〈dasomell@gmail.com〉

# WORKDIR /usr/local/tomcat
ADD ./target/sample-1.0.0.war /usr/local/tomcat/webapps/war.war

EXPOSE 8080
```

❑ sample 프로젝트(Kubernetes)

- 파일 생성(/src/k8s/deployment.yaml)

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: egov-sample
  labels:
    app: egov-sample
spec:
  replicas: 1
  revisionHistoryLimit: 1
  selector:
    matchLabels:
      app: egov-sample
  template:
    metadata:
      labels:
        app: egov-sample
    spec:
      containers:
        - name: egov-sample
          image: localhost:5000/egov-sample:1.0.0
          ports:
            - containerPort: 8080
          imagePullPolicy: Always
          resources:
            requests:
              cpu: 0.5
              memory: 0.5Gi
            limits:
              cpu: 0.5
              memory: 0.5Gi
```

- 파일 생성(/src/k8s/service.yaml)

```yaml
apiVersion: v1
kind: Service
metadata:
  name: egov-sample
spec:
  selector:
    app: egov-sample
  ports:
  - port: 8080
    targetPort: 8080
```

- 파일 생성(/src/k8s/ingress.yaml)

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: opdc-ingress
spec:
  rules:
    - host: opdc.io
      http:
        paths:
          - path: /war
            backend:
              serviceName: egov-sample
              servicePort: 8080
          - path: /maven
            backend:
              serviceName: sample-boot
              servicePort: 8080
    - host: opdc2.io
      http:
        paths:
          - path: /gradle
            backend:
              serviceName: sample-boot2
              servicePort: 8080
```

5.5 Cloud 테스트 환경 구축

❑ sample 프로젝트(실행)

- Docker Image 생성
  - docker build . -t localhost:5000/egov-sample:1.0.0
- Docker Registory 등록
  - docker push localhost:5000/egov-sample:1.0.0
- Docker Registory 확인
  - curl -X GET http://localhost:5000/v2/egov-sample/tags/list
- Deployment 생성
  - kubectl apply -f ./k8s/deployment.yaml
- Service 생성
  - kubectl apply -f ./k8s/service.yaml

실습 간 한번만 실행하면 됨

- Ingress 생성
  - kubectl apply -f ./k8s/ingress.yaml
- hosts 파일 수정
  - 127.0.0.1 opdc.io
  - 127.0.0.1 opdc2.io
- 배포 확인
  - http://opdc.io/war/egovSampleList.do

5.5 Cloud 테스트 환경 구축

❑ sample-boot 프로젝트(summary)

- boot 전환
- mvn clean package spring-boot:repackage k8s:build k8s:resource k8s:push k8s:apply
- kubectl apply -f ./k8s/ingress.yaml
- hosts 파일 수정
  - 127.0.0.1 opdc.io
  - 127.0.0.1 opdc2.io
- http://opdc.io/maven/egovSampleList.do

실습 간 한번만 실행하면 됨

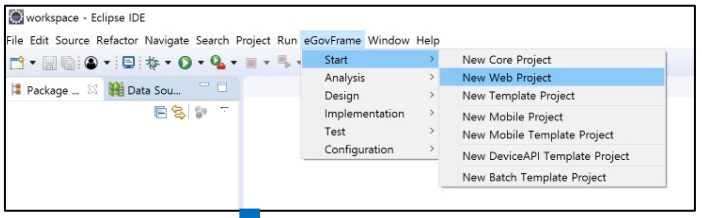
❑ sample-boot 프로젝트(초기 설정)

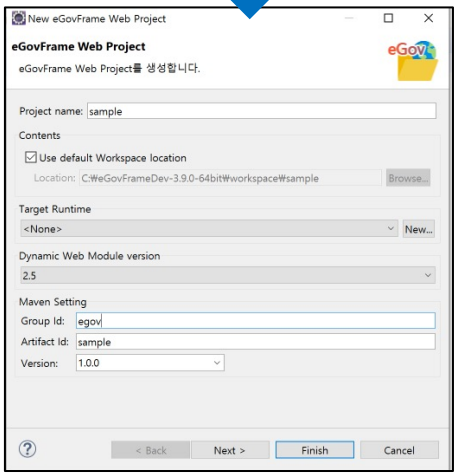
- springboot로 전환하여 embedded Tomcat kubernetes-maven-plug을 사용 Docker Daemon없이 image 생성 후 배포
- 프로젝트 소스 생성

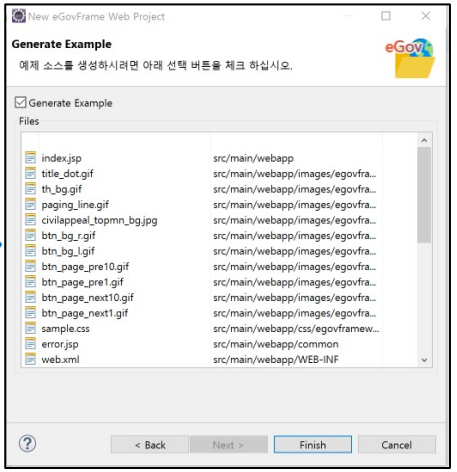
1. eGovFrame -〉 Start -〉 new Web Project
2. 프로젝트 정보 작성 -〉 Next

Project name	sample-boot
Group Id	egov
Artifact Id	sample
Version	1.0.0

3. ‘Generate Example’ 클릭 -〉 Finish







5.5 Cloud 테스트 환경 구축

❑ sample-boot 프로젝트(Boot 전환)

- pom.xml

```
<properties>
  <spring.maven.artifact.version>4.3.22.RELEASE</spring.maven.artifact.version>
  <egovframework.rte.version>3.9.0</egovframework.rte.version>
  <selenium.version>3.141.59</selenium.version>
    <jkube.build.strategy>jib</jkube.build.strategy>
    <jkube.debug.enabled>true</jkube.debug.enabled>
    <timestamp>${maven.build.timestamp}</timestamp>
    <maven.build.timestamp.format>yyyyMMddHHmmss</maven.build.timestamp.format>
</properties>

<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.5.19.RELEASE</version>
</parent>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <exclusions>
    <exclusion>
      <groupId>ch.qos.logback</groupId>
      <artifactId>logback-classic</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
</dependency>
<dependency>
  <groupId>org.glassfish</groupId>
  <artifactId>javax.el</artifactId>
</dependency>
```

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
  <plugin>
    <groupId>org.eclipse.jkube</groupId>
    <artifactId>kubernetes-maven-plugin</artifactId>
    <version>1.0.2</version>
    <configuration>
      <images>
        <image>
          <name>localhost:5000/egov-sampleboot:${timestamp}</name>
          <build>
            <from>fabric8/java-centos-openjdk8-jre</from>
            <maintainer>dasomell@gmail.com</maintainer>
            <assembly>
              <inline>
                <baseDirectory>/deployments</baseDirectory>
              </inline>
              <mode>dir</mode>
              <targetDir>/deployments</targetDir>
            </assembly>
            <env>
              <JAVA_LIB_DIR>/deployments</JAVA_LIB_DIR>
              <CATALINA_OPTS>-Djava.security.egd=file:/dev/urandom</CATALINA_OPTS>
            </env>
          </build>
        </image>
      </images>
    </configuration>
    <executions>
      <execution>
        <id>jkube</id>
        <goals>
          <goal>resource</goal>
          <goal>build</goal>
          <goal>push</goal>
          <goal>deploy</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
```

❑ sample-boot 프로젝트(Boot 전환)

- 파일 이동
  - src/main/webapp/WEB-INF/config/egovframework/springmvc/dispatcher-servlet.xml
    → src/main/resources/egovframework/springmvc/dispatcher-servlet.xml
  - src/main/webapp/WEB-INF/config/egovframework/validator/validator.xml, validator-rules.xml
    → src/main/resources/egovframework/validator/validator.xml, validator-rules.xml
- context-validator 수정(egovframework/spring/context-validator.xml)

```
<bean id="validatorFactory" class="org.springmodules.validation.commons.DefaultValidatorFactory">
  <property name="validationConfigLocations">
    <list>
      <value>classpath:/egovframework/validator/validator-rules.xml</value>
      <value>classpath:/egovframework/validator/validator.xml</value>
    </list>
  </property>
</bean>
```

❑ sample-boot 프로젝트(Boot 전환)

- 파일 생성(/src/main/resources/application.properties)

```
server.context-path=/maven
```

- 파일 생성(/src/main/java/egovframework/example/SampleBootApplication.java)

```
package egovframework.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.web.support.SpringBootServletInitializer;
import org.springframework.context.annotation.ImportResource;

@SpringBootApplication(scanBasePackageClasses =
SampleBootApplication.class)
@ImportResource({"classpath:/egovframework/spring/context-*.xml"
        ,"classpath:/egovframework/springmvc/dispatcher-servlet.xml"
})
public class SampleBootApplication extends SpringBootServletInitializer {
   public static void main(String[] args) throws Exception {
      SpringApplication.run(SampleBootApplication.class, args);
   }
}
```

❑ sample-boot 프로젝트(build & deploy)

- 파일 생성(/src/main/jkube/deployment.yml)

```
spec:
 template:
  spec:
   containers:
    - resources:
      limits:
       cpu: 500m
       memory: 500Mi
      requests:
       cpu: 500m
       memory: 500Mi
```

- 파일 생성(/src/main/jkube/service.yml)

```
apiVersion: v1
kind: Service
metadata:
 name: sample-boot
spec:
 selector:
  app: sample-boot
 ports:
 - port: 8080
  targetPort: 8080
```

- 파일 생성(/src/k8s/ingress.yaml)

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
 name: opdc-ingress
spec:
 rules:
  - host: opdc.io
   http:
    paths:
     - path: /war
      backend:
       serviceName: egov-sample
       servicePort: 8080
     - path: /maven
      backend:
       serviceName: sample-boot
       servicePort: 8080
  - host: opdc2.io
   http:
    paths:
     - path: /gradle
      backend:
       serviceName: sample-boot2
       servicePort: 8080
```

5.5 Cloud 테스트 환경 구축

❑ sample-boot 프로젝트(실행)

- 빌드 및 배포
  - mvn clean package spring-boot:repackage k8s:build k8s:resource k8s:push k8s:apply
  - 빌드 실패 시 eclipse에서 외부 maven 설정
- Ingress 생성
  - kubectl apply -f ./k8s/ingress.yaml
- hosts 파일 수정
  - 127.0.0.1 opdc.io
  - 127.0.0.1 opdc2.io
- 배포 확인
  - http://opdc.io/maven/egovSampleList.do

실습 간 한번만 실행하면 됨

5.5 Cloud 테스트 환경 구축

❑ sample-boot2 프로젝트(summary)

- boot 전환
- gradle 전환
- ./gradlew clean war bootRepackage
- docker build . -t localhost:5000/sample-boot2:2.0.0
- docker push localhost:5000/sample-boot2:2.0.0
- curl -X GET http://localhost:5000/v2/sample-boot2/tags/list
- kubectl apply -f ./k8s/deployment.yaml
- kubectl apply -f ./k8s/service.yaml
- kubectl apply -f ./k8s/ingress.yaml
- hosts 파일 수정
  - 127.0.0.1 opdc.io
  - 127.0.0.1 opdc2.io
- http://opdc2.io/gradle/egovSampleList.do

실습 간 한번만 실행하면 됨

5.5 Cloud 테스트 환경 구축

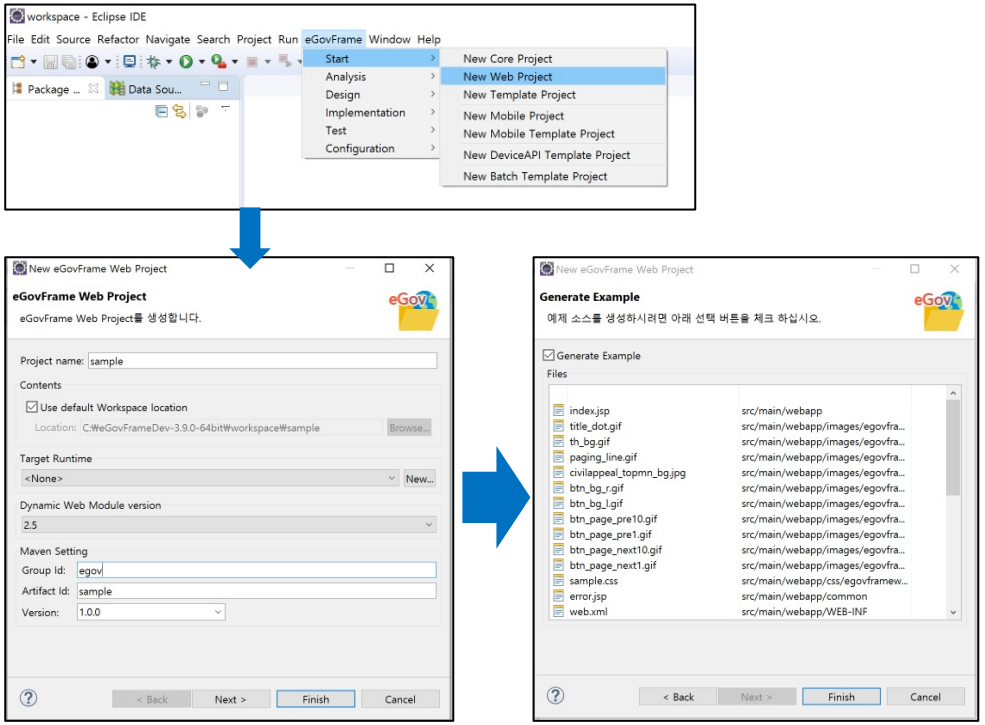
❑ sample-boot2 프로젝트(초기 설정)

- springboot, gradle로 전환하여 war 배포
- 프로젝트 소스 생성

1. eGovFrame -〉 Start -〉 new Web Project
2. 프로젝트 정보 작성 -〉 Next

Project name	sample-boot2
Group Id	egov
Artifact Id	sample
Version	1.0.0

3. ‘Generate Example’ 클릭 -〉 Finish



5.5 Cloud 테스트 환경 구축

❑ sample-boot2 프로젝트(Boot 전환)

- 파일 이동
  - src/main/webapp/WEB-INF/config/egovframework/springmvc/dispatcher-servlet.xml
    → src/main/resources/egovframework/springmvc/dispatcher-servlet.xml
- context-validator 수정(egovframework/spring/context-validator.xml)

```
<bean id="validatorFactory" class="org.springmodules.validation.commons.DefaultValidatorFactory">
  <property name="validationConfigLocations">
    <list>
      <value>classpath:/egovframework/validator/validator-rules.xml</value>
      <value>classpath:/egovframework/validator/validator.xml</value>
    </list>
  </property>
</bean>
```

- 파일 생성(/src/main/resources/application.properties)

```
server.context-path=/gradle
```

- 파일 생성(/src/main/java/egovframework/example/SampleBootApplication.java)

```
package egovframework.example;
~
@SpringBootApplication(scanBasePackageClasses = SampleBoot2Application.class)
@ImportResource({"classpath:/egovframework/spring/context-*.xml"
        ,"classpath:/egovframework/springmvc/dispatcher-servlet.xml"
})
public class SampleBoot2Application extends SpringBootServletInitializer {
  public static void main(String[] args) throws Exception {
    SpringApplication.run(SampleBoot2Application.class, args);
  }
}
```

5.5 Cloud 테스트 환경 구축

❑ sample-boot2 프로젝트(gradle 전환)

- 파일 삭제(pom.xml)
- 파일 생성(*build.gradle)*

```
plugins {
   id 'java'
   id 'war'
   id 'org.springframework.boot' version '1.5.19.RELEASE'
   id 'io.spring.dependency-management' version
'1.0.9.RELEASE'
}

sourceCompatibility = JavaVersion.VERSION_1_8
targetCompatibility = JavaVersion.VERSION_1_8

repositories {
   maven { url 'https://repo1.maven.org/maven2/' }
   maven { url 'http://maven.egovframe.kr:8080/maven/' }
   jcenter()
}

configurations {
   all*.exclude group: 'ch.qos.logback', module: 'logback-classic'
}
```

```
dependencies {
   implementation 'org.springframework.boot:spring-boot-starter-web'
   implementation 'org.springframework.boot:spring-boot-starter-jdbc'
   implementation 'org.apache.tomcat.embed:tomcat-embed-jasper'
   implementation 'org.glassfish:javax.el'

   implementation 'egovframework.rte:egovframework.rte.ptl.mvc:3.9.0'
   implementation 'egovframework.rte:egovframework.rte.psl.dataaccess:3.9.0'
   implementation 'egovframework.rte:egovframework.rte.fdl.idgnr:3.9.0'
   implementation 'egovframework.rte:egovframework.rte.fdl.property:3.9.0'
   compileOnly 'javax.servlet:servlet-api:2.5'
   implementation 'javax.servlet:jstl'
   implementation 'taglibs:standard:1.1.2'
   implementation 'org.antlr:antlr:3.5'
   implementation 'org.hsqldb:hsqldb'

}
```

- 파일 생성(*settings.gradle)*

```
rootProject.name = 'sample-boot2'
```

❑ sample-boot2 프로젝트(*build & deploy*)

- Dockerfile 작성

```
FROM openjdk:8-jre-alpine
LABEL maintainer=〈dasomell@gmail.com〉
RUN ln -snf /usr/share/zoneinfo/Asia/Seoul /etc/localtime && echo Asia/Seoul 〉 /etc/timezone

COPY ./build/libs/sample-boot2.war app.war
ENTRYPOINT ["java","-server","-D64","-Djava.security.egd=file:./dev/urandom","-jar","/app.war"]
EXPOSE 8080
```

- 파일 수정(src/main/webapp/WEB-INF/jsp/egovframework/example/sample/egovSampleList.jsp)

```
 6번 라인 : 〈%@ page import="java.net.InetAddress" %〉
92번 라인: 〈%=request.getLocalAddr()%〉
93번 라인: 〈%=request.getLocalName()%〉
```

❑ sample-boot2 프로젝트(*build & deploy*)

- 파일 생성(/src/k8s/deployment.yaml)

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-boot2
  labels:
    app: sample-boot2
spec:
  replicas: 2
  revisionHistoryLimit: 1
  selector:
    matchLabels:
      app: sample-boot2
  template:
    metadata:
      labels:
        app: sample-boot2
    spec:
      containers:
        - name: sample-boot2
          image: localhost:5000/sample-boot2:2.0.0
          ports:
            - containerPort: 8080
          imagePullPolicy: Always
          resources:
            requests:
              cpu: 0.5
              memory: 0.5Gi
            limits:
              cpu: 0.5
              memory: 0.5Gi
```

- 파일생성(/src/k8s/service.yaml)

```yaml
apiVersion: v1
kind: Service
metadata:
  name: sample-boot2
spec:
  selector:
    app: sample-boot2
  ports:
  - port: 8080
    targetPort: 8080
```

- 파일 생성(/src/k8s/ingress.yaml)

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: opdc-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
    - host: opdc.io
    - http:
        paths:
          - path: /war
            backend:
              serviceName: egov-sample
              servicePort: 8080
          - path: /maven
            backend:
              serviceName: sample-boot
              servicePort: 8080
    - host: opdc2.io
    - http:
        paths:
          - path: /gradle
            backend:
              serviceName: sample-boot2
              servicePort: 8080
```

5.6 Tool 활용

❑ k9s

- Kubernetes 클러스터와 상호작용하는 터미널 UI를 제공
  응용 프로그램을 보다 쉽게 탐색, 관찰 및 관리
  지속적으로 변화를 감시, Resource와 상호작용하는 명령 제공
  (https://github.com/derailed/k9s)

참고 문헌

❑ IBM Cloud 용어집

- https://cloud.ibm.com/docs/overview?topic=overview-glossary&locale=ko

❑ NIST(미국 국립표준기술원) Special Publication 500-322

- https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-322.pdf

❑ 01. PasS-TA 전문가 육성 교육 과정 교제 1일차.pdf

❑ Istio로 시작하는 서비스 메시(출판사 에이콘)

❑ https://martinfowler.com

❑ https://microservices.io

❑ https://www.eclipse.org/jkube/docs/kubernetes-maven-plugin/

감사합니다