이미지 출처 : Pixel perfect - Flaticon

Docker

■ 소개

도커 != 컨테이너

컨테이너 = 서비스 "실행"에 필요한 모든 걸 포장한 것

컨테이너 런타임 = 그 컨테이너를 관리하는 도구 (CRI-O, 도커, 미란티스 등)

도커 = 컨테이너 런타임 중 하나, 근데 가장 많이 쓰임

알아두면 좋은 점

도커가 돌아갈 수 있는 환경은 **"리눅스"**

도커 위에서 돌릴 수 있는 환경도 **"리눅스"**

이미지 출처 : Pixel perfect - Flaticon

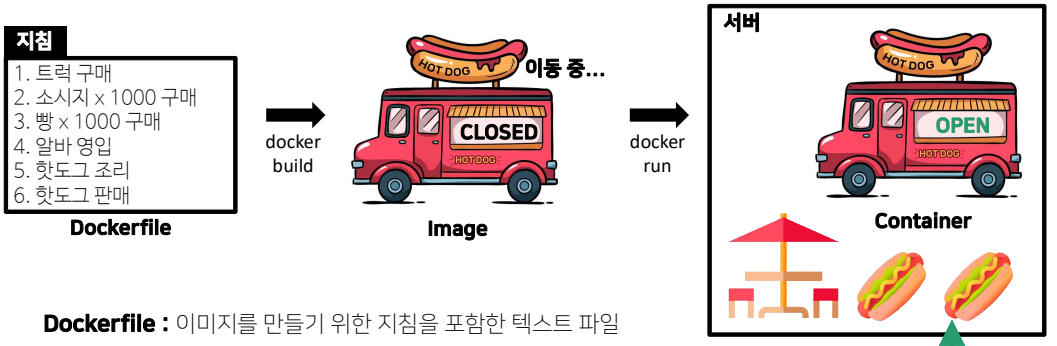
■ 컨테이너의 목적

프로세스가방에들어가신다

매번 배포할 때마다 서버 접속해서 node 깔고, nginx 깔고, npm install, npm run...
"실행(process)"을 위해 일일이 명령어 치고 구성하는게 귀찮다
"실행(process)" 구성을 "가방(image)"에 "포장"해서 들고 다니자

옮길 때(pull, push)는 닫힌 가방(image)으로 들고 다니다가
목적지 도착했을 때 열린 가방(container)으로 실행(process) 구성 꺼내 쓰자
주요 작업은 가방을 옮기다, 열다, 닫다가 전부
일일이 명령어 칠 필요 없이 손쉽게 동일한 실행(process) 가능

■ Dockerfile, 이미지, 컨테이너



Dockerfile : 이미지를 만들기 위한 지침을 포함한 텍스트 파일

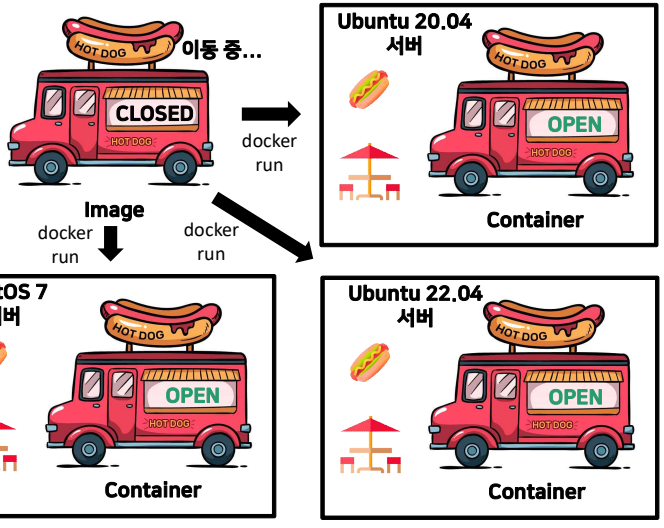
이미지 : 앱/서비스 실행에 필요한 정보를 포장한 것 (고정, 정적)

컨테이너 : 이미지를 실행한 것 = 이미지의 인스턴스 (읽기/쓰기 가능, 동적)

이미지 출처 https://kr.freepik.com/free-vector/original-pack-of-hand-drawn-food-trucks_1266833.htm

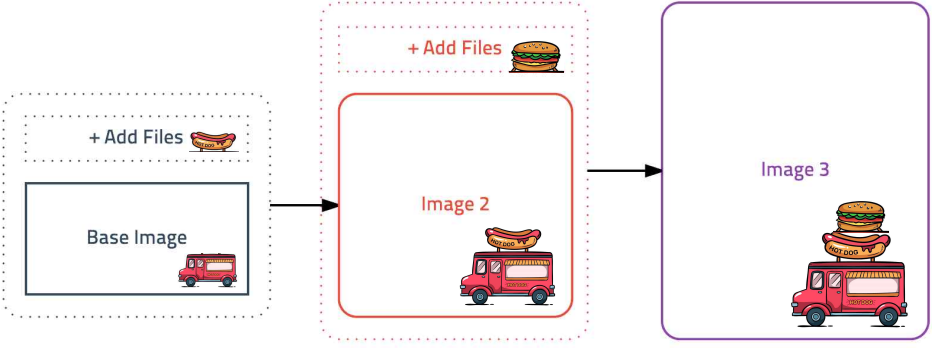
■ Build Once, Run Any Where

한 번 이미지를 만들면(build),
다양한 OS 환경에서
동일한 실행의 컨테이너 동작(run)



■ 이미지는 진화한다

이미지 출처 https://kr.freepik.com/free-vector/original-pack-of-hand-drawn-food-trucks_1266833.htm
이미지 출처 https://subicura.com/assets/article_images/2017-02-10-docker-guide-for-beginners-create-image-and-deploy/create-image.png

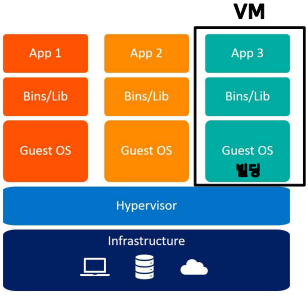


이미지는 레이어로 구성된다
이미지를 변경한다 = 변경사항이 담긴 새 레이어를 추가한다

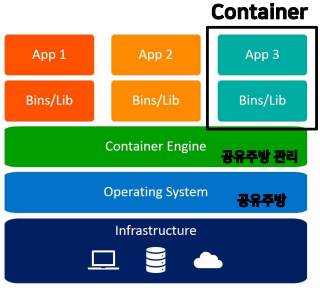
VM vs 컨테이너

이미지 출처 : Pixel perfect - Flaticon

이미지 출처 : https://images.contentstack.io/v3/assets/blt300387d93dabf50e/bltb6200bc085503718/5e1f209a63d1b6503160c6d5/containers-vs-virtual-machines.jpg



VMs (Virtual Machines)



Containers

문제

파스타를 먹고 싶다

해결방법

1. 빌딩을 짓는다 (어...?)
2. 완성된 빌딩을 방문한다
3. 파스타 재료를 준비한다
4. 파스타를 요리해서 먹는다

해결방법

1. 공유 주방을 방문한다
2. 파스타 재료를 준비한다
3. 파스타를 요리해서 먹는다

시간 덜 걸리고 가벼움

이미지 출처 https://kr.freepik.com/free-vector/original-pack-of-hand-drawn-food-trucks_1266833.htm

■ 컨테이너의 장점

가볍다

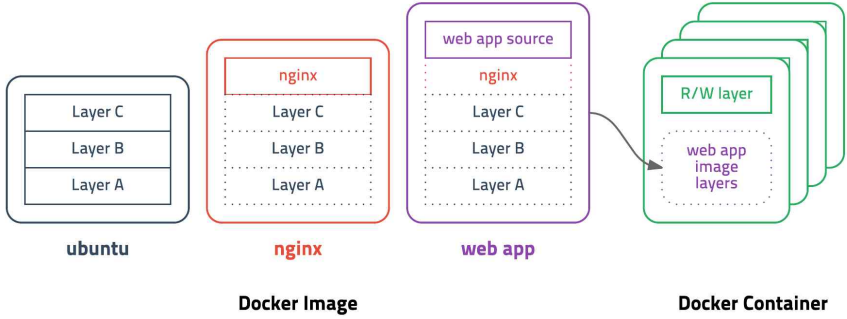
옮기기에 용이하다
(가벼우니까)

실행 시작/종료가 빠르다
(가벼우니까)

다양한 OS 환경에서 일관성 있게 실행된다
(공통적이고 필수적인 것만 가볍게 챙겼으니까)

이미지 출처 https://subicura.com/assets/article_images/2017-01-19-docker-guide-for-beginners-1/image-layer.png

■ 컨테이너 활용 예시

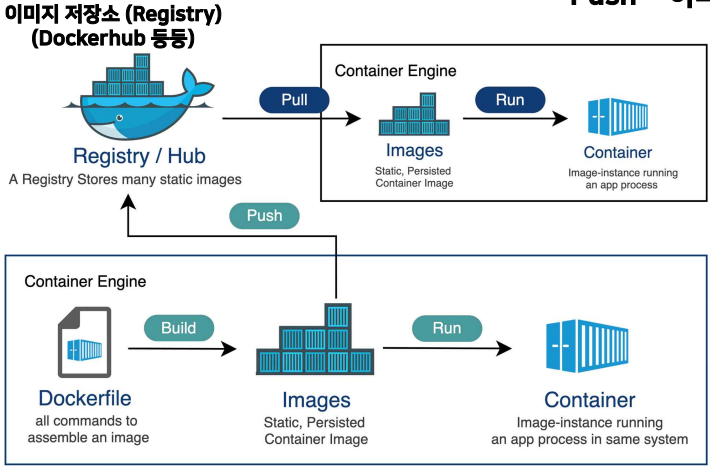


ubuntu 베이스 이미지에 필요한 레이어를 추가한다
최종 이미지(web app)에 읽기/쓰기 가능한 레이어를 추가하면 그것이 곧 컨테이너다

이미지 출처 https://blogs.sap.com/wp-content/uploads/2022/09/1-83.png

■ 이미지를 공유하자 (Registry)

Pull = 이미지 다운로드
Push = 이미지 업로드



이미지 출처 : https://kubernetes.io/images/nav_logo.svg

Kubernetes

이미지 출처 : https://kubernetes.io/images/nav_logo.svg

■ 쿠버네티스의 개념

컨테이너 오케스트레이션

"대규모 운영 환경"에 적합한 도구

여러 대의 서버(=클러스터)에서
다수의 컨테이너를 실행할 때
컨테이너를 **자동**으로 **관리**해주는 소프트웨어
(물론 한 대의 서버에서 쿠버네티스 운영도 가능)

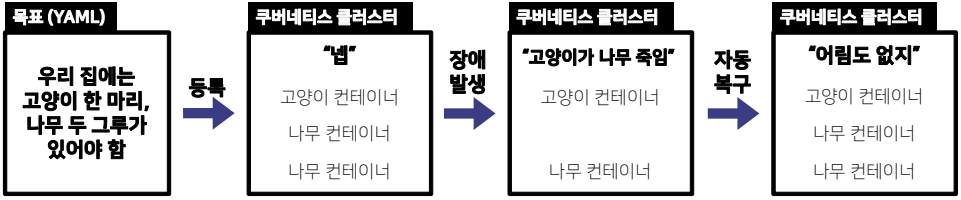
이미지 출처 : https://kubernetes.io/images/nav_logo.svg

■ 컨테이너 오케스트레이션

클러스터 = 고성능 컴퓨팅을 위해 여러 대의 서버(컴퓨터)를 합친 것
노드 = 클러스터를 구성하는 각각의 서버(컴퓨터)

쿠버네티스 클러스터 = 컨트롤플레인 노드x1 + 워커 노드xN
컨트롤플레인 노드 = 전체적인 제어 담당, 워커 노드 관리
워커 노드 = 컨테이너 실행 담당

사용자는 쿠버네티스 클러스터에 **"목표 상태"(Desired State)**를 YAML파일로 등록
쿠버네티스 클러스터는 등록된 **"목표 상태"**에 맞게 쿠버네티스 리소스를 **자동으로 생성 및 유지**



이미지 출처 : https://kubernetes.io/images/nav_logo.svg

■ 리소스

파드 = 쿠버네티스 클러스터에서 컨테이너를 실행하는 가장 기본적인 컴퓨팅 단위

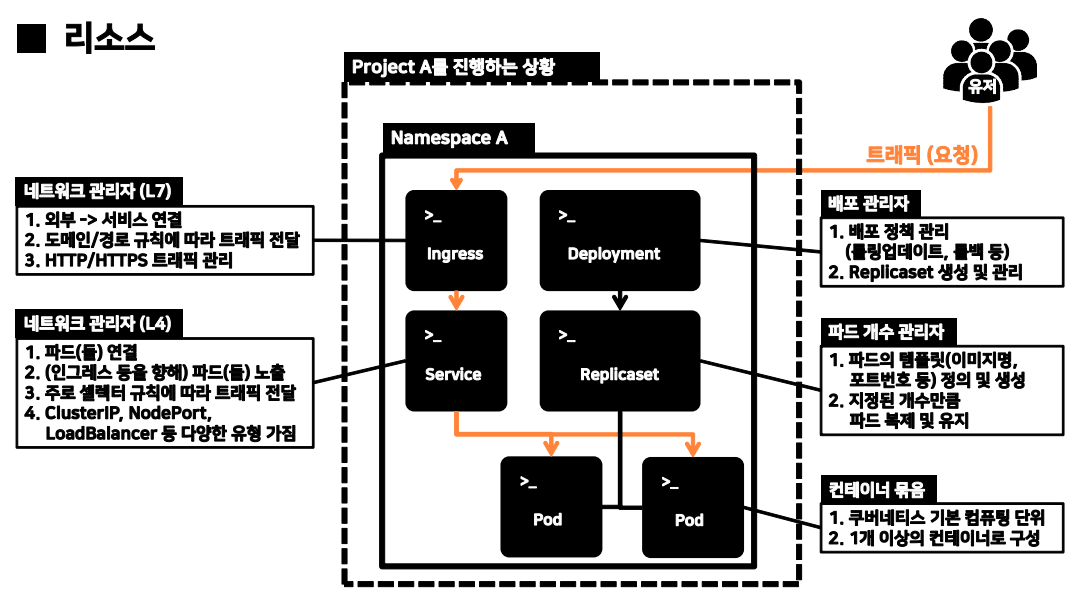
1개의 파드에 1개의 컨테이너만 실행될 수도 있고
1개의 파드에 다수의 컨테이너가 실행될 수도 있다 (= 멀티 컨테이너 파드)

"파드"만으로는 컨테이너 관리를 전부 처리하기엔 어려운 점이 많다
컨테이너를 실행하는 **"파드"** 외에도 컨테이너 관리에 도움을 주는

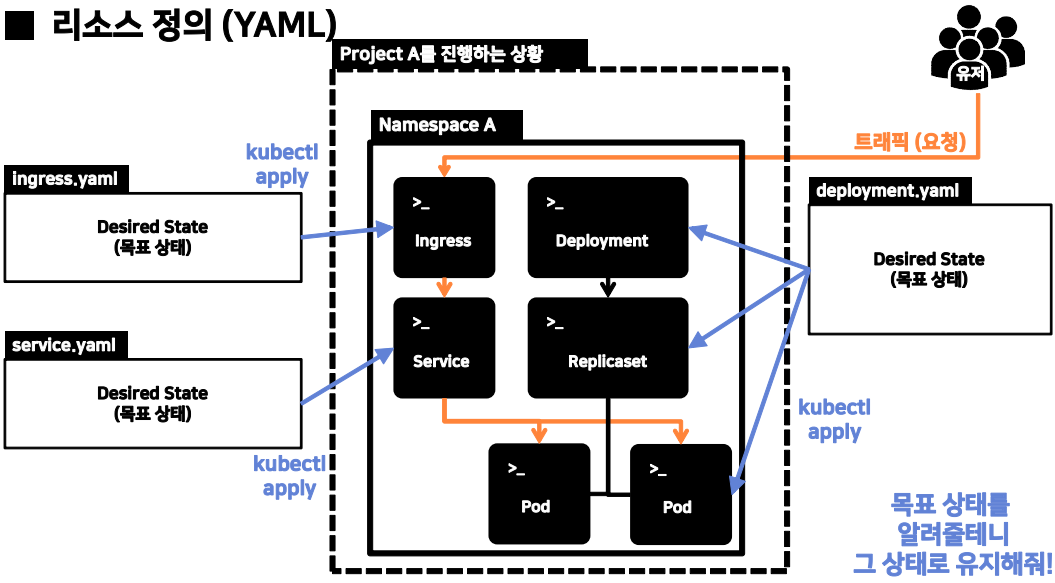
"디플로이먼트" (배포 관리)
"서비스" (L4 네트워크 관리)
"인그레스" (L7 네트워크 관리)
등등이 존재 하며

쿠버네티스에서 각자의 역할이 있는 파드와 이들을 묶어서 **"쿠버네티스 리소스"**라고 부른다

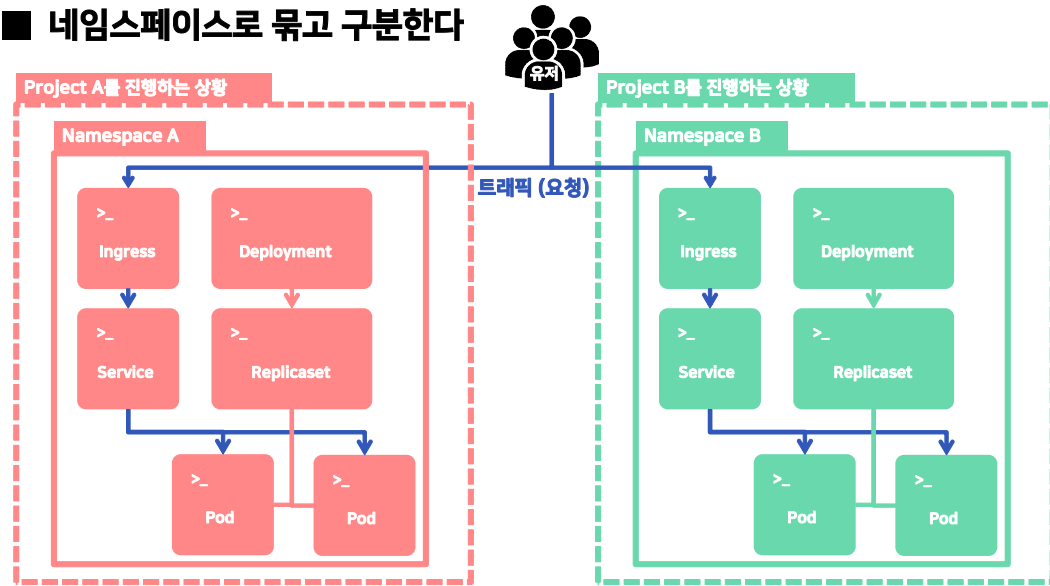
■ 리소스


Project A를 진행하는 상황
Namespace A
유저
트래픽 (요청)
Ingress
Deployment
Service
Replicaset
Pod
Pod
네트워크 관리자 (L7)
1. 외부 -> 서비스 연결
2. 도메인/경로 규칙에 따라 트래픽 전달
3. HTTP/HTTPS 트래픽 관리
네트워크 관리자 (L4)
1. 파드(들) 연결
2. (인그레스 등을 향해) 파드(들) 노출
3. 주로 셀렉터 규칙에 따라 트래픽 전달
4. ClusterIP, NodePort,
LoadBalancer 등 다양한 유형 가짐
배포 관리자
1. 배포 정책 관리
(롤링업데이트, 롤백 등)
2. Replicaset 생성 및 관리
파드 개수 관리자
1. 파드의 템플릿(이미지명,
포트번호 등) 정의 및 생성
2. 지정된 개수만큼
파드 복제 및 유지
컨테이너 묶음
1. 쿠버네티스 기본 컴퓨팅 단위
2. 1개 이상의 컨테이너로 구성

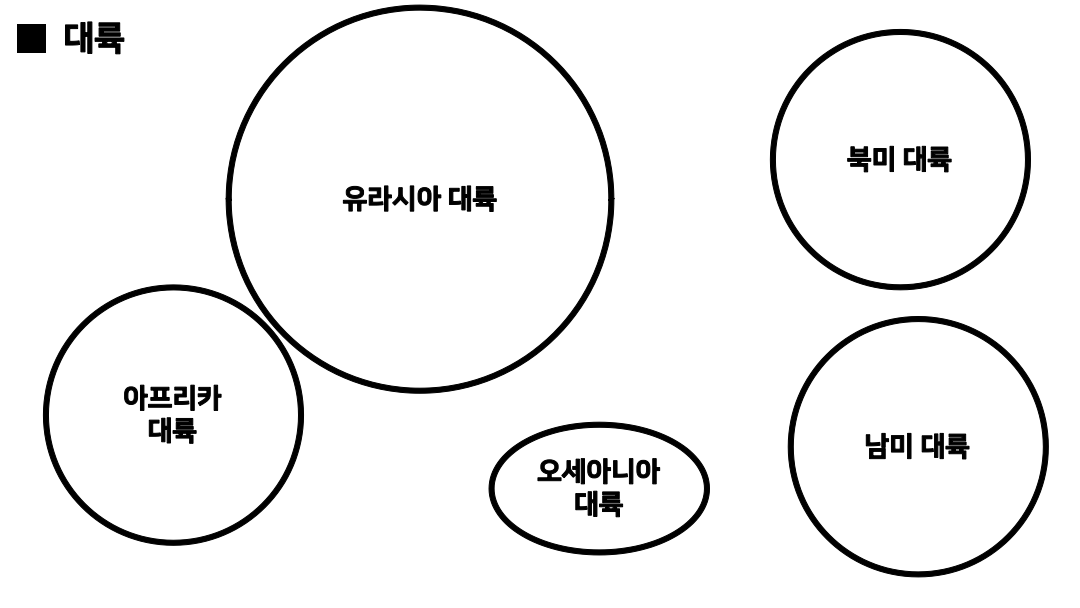
■ 리소스 정의 (YAML)

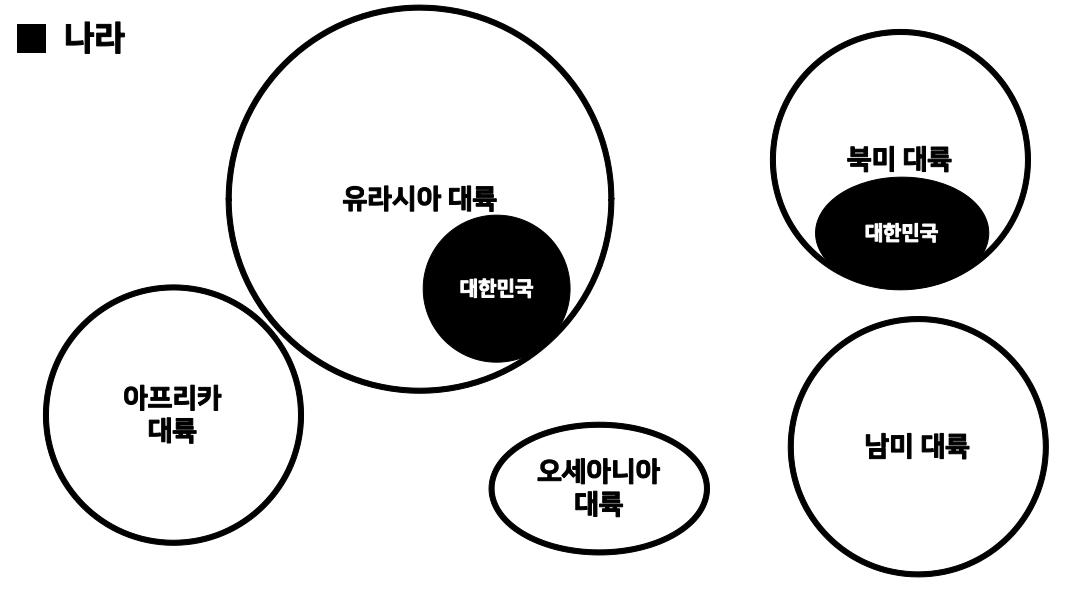

Project A를 진행하는 상황
유저
Namespace A
kubectl
apply
트래픽 (요청)
ingress.yaml
Desired State
(목표 상태)
>_
Ingress
>_
Deployment
deployment.yaml
Desired State
(목표 상태)
service.yaml
Desired State
(목표 상태)
>_
Service
>_
Replicaset
kubectl
apply
kubectl
apply
>_
Pod
>_
Pod
목표 상태를
알려줄테니
그 상태로 유지해줘!

■ 네임스페이스로 묶고 구분한다

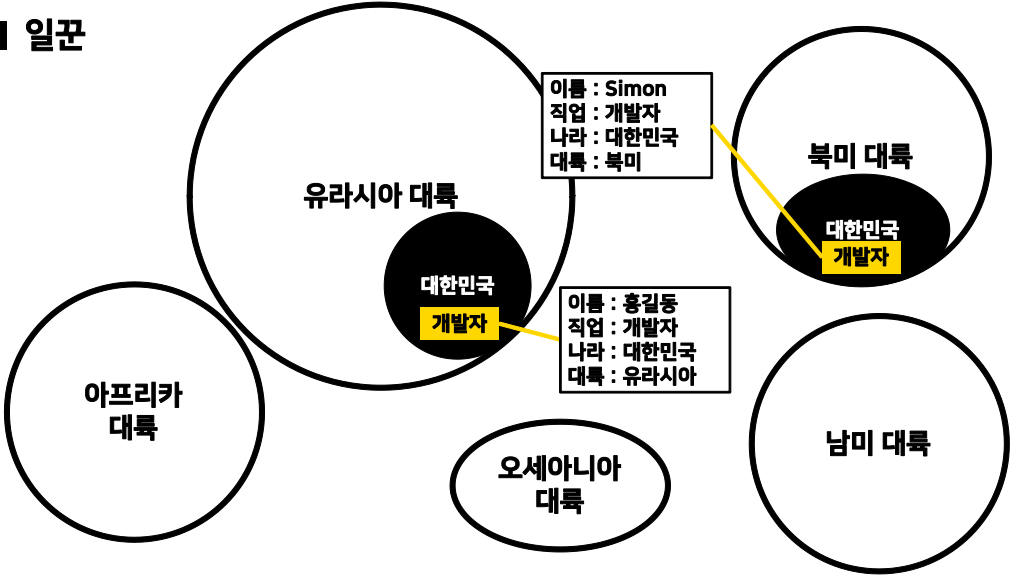


■ 대륙

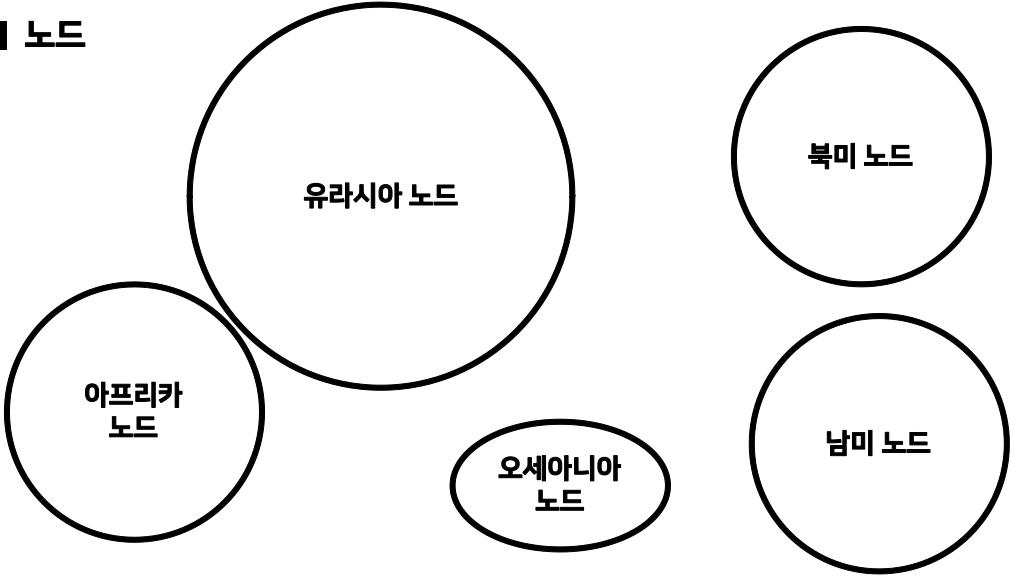



나라
유라시아 대륙
대한민국
북미 대륙
대한민국
아프리카
대륙
오세아니아
대륙
남미 대륙

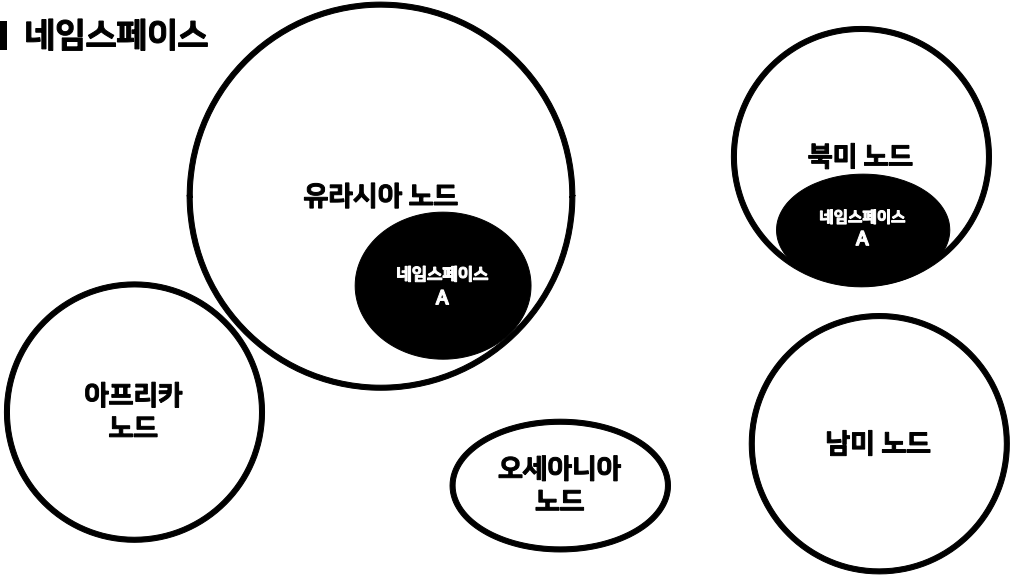
■ 일꾼


이름 : Simon
직업 : 개발자
나라 : 대한민국
대륙 : 북미
유라시아 대륙
대한민국
개발자
북미 대륙
대한민국
개발자
이름 : 홍길동
직업 : 개발자
나라 : 대한민국
대륙 : 유라시아
아프리카
대륙
오세아니아
대륙
남미 대륙

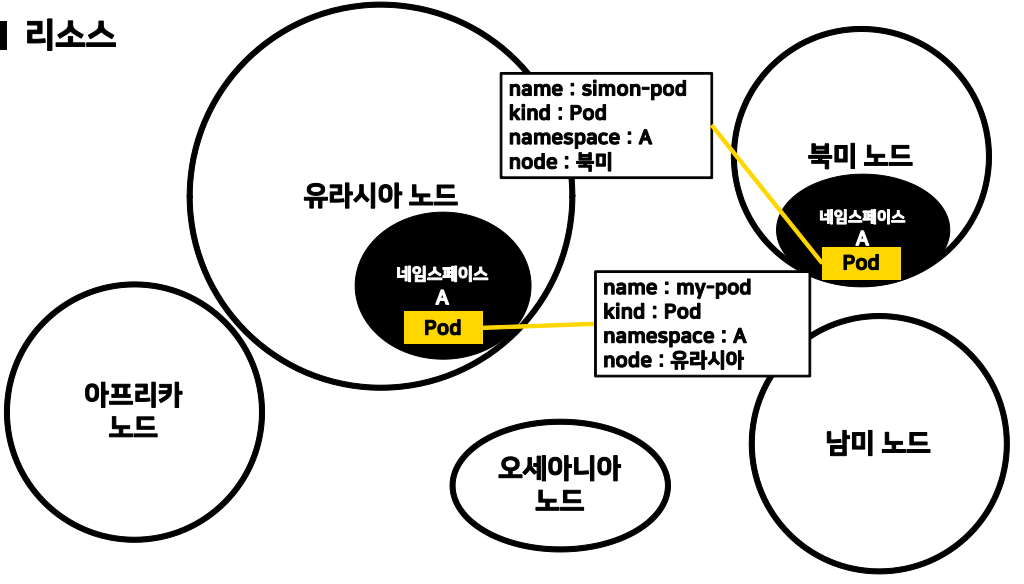
■ 노드



■ 네임스페이스



■ 리소스


name : simon-pod
kind : Pod
namespace : A
node : 북미
유라시아 노드
네임스페이스
A
Pod
북미 노드
네임스페이스
A
Pod
name : my-pod
kind : Pod
namespace : A
node : 유라시아
아프리카
노드
오세아니아
노드
남미 노드

이미지 출처 : Pixel perfect - Flaticon

■ 노드와 네임스페이스

노드 != 네임스페이스

노드 = 리소스의 물리적인 소속/위치
네임스페이스 = 리소스의 논리적인 소소/위치

노드가 같아도 **네임스페이스**가 다를 수 있고
네임스페이스가 같아도 **노드**가 다를 수 있다