클라우드기초

최종석(jschoi@ssu.ac.kr)

C2 Open PaaS 정의

paas를 구현하는 쿠버네티스

==

cp, container platform 이라고 함.

01 Container Platform 이해

01. Container Platform 구축환경

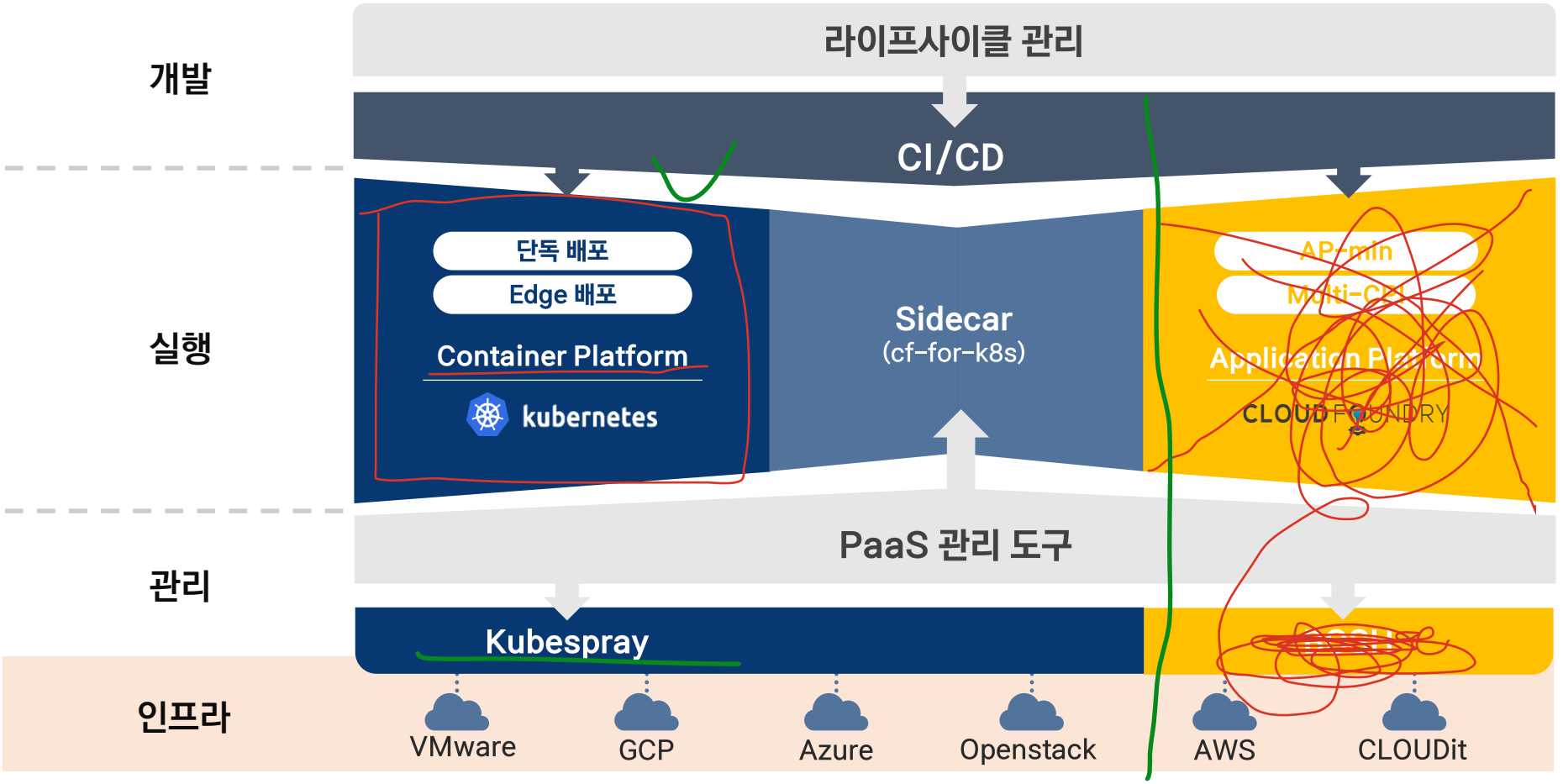
02. Container Platform 서비스 소개

03. Container Platform 발전과정

01 Container Platform 구축환경

» 개방형 클라우드 플랫폼 아키텍처

개방형 클라우드 플랫폼 핵심 엔진은 **애플리케이션 플랫폼**과 **컨테이너 플랫폼**으로 구성



출처: 개방형 클라우드 플랫폼 대표포털

01 Container Platform 구축환경

» 지원환경 (IaaS)

대부분의 Iaas에서 kubernetics가 사용가능하게끔 지원함.

다양한 인프라(IaaS)를 지원하여 간편하게 엔진(AP, CP) 구축 및 운영 가능

프라이빗

IAAS계의 최강자 중에 하나

openstack.

레드햇 오픈 스택 플랫폼이 있음
오픈 쉬프트라고

애도 유명함

가상화 쪽으로
유명함

VMware vSphere®

CSP도
이들을 사용.

퍼블릭

Azure

퍼블릭

aws

IaaS

퍼블릭

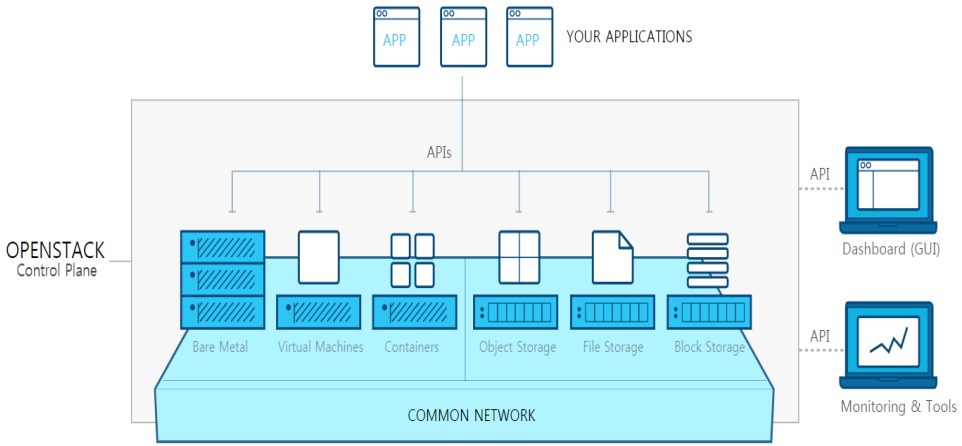
Google Cloud Platform

01 Container Platform 구축환경

» 지원환경 (IaaS)

openstack.

데이터 센터 전체에서 대규모 컴퓨팅, 스토리지 및 네트워킹 리소스 풀을 제어하는 클라우드 운영 체제



관리자가 제어할 수 있는
대시보드를 통해 관리

사용자가 웹 인터페이스를 통해
리소스 프로비저닝 가능

출처: OpenStack 정의 (www.openstack.org)

01 Container Platform 구축환경

» 지원환경 (IaaS)

서버를 가상화하는 베어메탈 하이퍼바이저

: 호스트 운영체제에서 다수의 운영체제를 실행하기 위한 논리적 플랫폼

서버를 가상화하여 IT인프라 관리에 소모되는 시간과 비용을 절감

하드웨어를 많이 사용하지 않고도 애플리케이션을 통합할 수 있음

ESXi 기반으로 개발 – 안정성과 성능, 그리고 환경 지원에 대한 업계 표준을 정립

vSphere client 지원 – 간편하게 몇 분만에 가상머신을 만들고 프로비저닝할 수 있음

씬 프로비저닝을 지원 – 스토리지 리소스를 물리적인 스토리지의 실제 용량 이상으로 초과 할당 가능하여 메모리 리소스 성능 최적화

01 Container Platform 구축환경

» 지원환경 (IaaS)

Building blocks의 형태로 서비스를 제공하는 아마존닷컴의 클라우드 컴퓨팅 서비스

서비스형 인프라(IaaS) 및 서비스형 플랫폼(PaaS)을 포함하는
가장 널리 사용되고 있는 종합적 클라우드 컴퓨팅 플랫폼

200개 이상의 서비스를 Building Block 형태로 제공

2023년 4월 기준, 세계 31개 지역에
99개 가용 영역(데이터센터)을 운영하고 있음

출처: 아마존웹서비스 홈페이지(https://aws.amazon.com/ko/what-is-aws/?nc1=f_cc)

01 Container Platform 구축환경

» 지원환경 (IaaS)

Google Cloud Platform

구글 내부와 동일한 지원 인프라스트럭처 위에서
호스팅을 제공하는 구글의 클라우드 컴퓨팅 서비스

GCP는 수십 개의 IaaS, PaaS 및 SaaS 서비스를 제공

GCP를 통해 인프라 관리, 서버 프로비저닝, 네트워크 구성으로 발생하는
간접비용을 줄일 수 있음

출처: 구글클라우드 홈페이지(https://cloud.google.com/why-google/)

01 Container Platform 구축환경

» 지원환경 (IaaS)

Azure

글로벌 네트워크에서 구축, 관리 및 사용할 수 있는
마이크로소프트의 클라우드 컴퓨팅 서비스

하이브리드 클라우드 환경을 통해 동일한 방식으로 응용 프로그램을 빌드 및 배포

클라우드 리소스를 모니터링하고 관리 및 보호하는
통합형 개발 및 관리 도구를 통해 탁월한 생산성을 확보

출처: 애저 홈페이지(https://azure.microsoft.com/ko-kr/overview/what-is-azure/)

02 Container Platform 서비스 소개

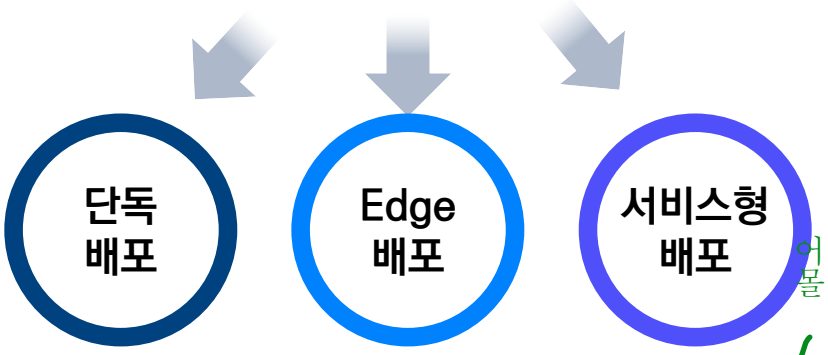
» 서비스 소개

IAas와 연결하여 사용

컨테이너 플랫폼이란

쿠버네티스를 사용할 수 있도록 배포하는

컨테이너 플랫폼



3가지 종류의 배포를 지원하는 플랫폼

쿠버네티스 기반의 단독 배포,
서비스형 배포 및 Edge 배포
기능을 구현하여
클라우드 기반 서비스 및 운영에
필요한 부가 서비스를 제공

어 ㅓ플리케이션 형과 같이 쓰는 것이므로
몰라도 됨.

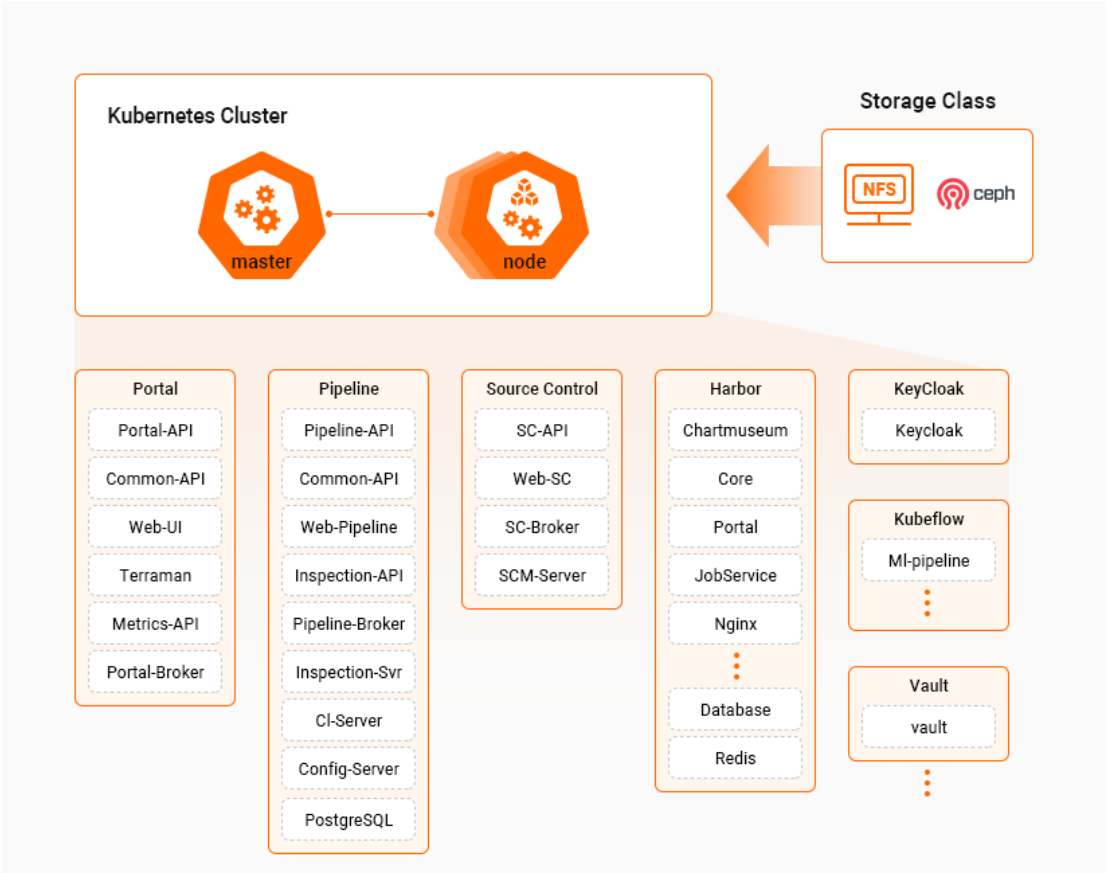
출처: 개방형 클라우드 플랫폼 대표포털

02 Container Platform 서비스 소개

» 동작원리

관련 오픈소스

컨테이너 플랫폼은 **쿠버네티스 클러스터 및 운영**에 필요한 **스토리지 서버**로 구성되며
디스크립션을 기반으로 **컨테이너화된 애플리케이션을 배포하는 방식**으로 동작



여러가지가
기본적으로
다같이 깔리는
플랫폼

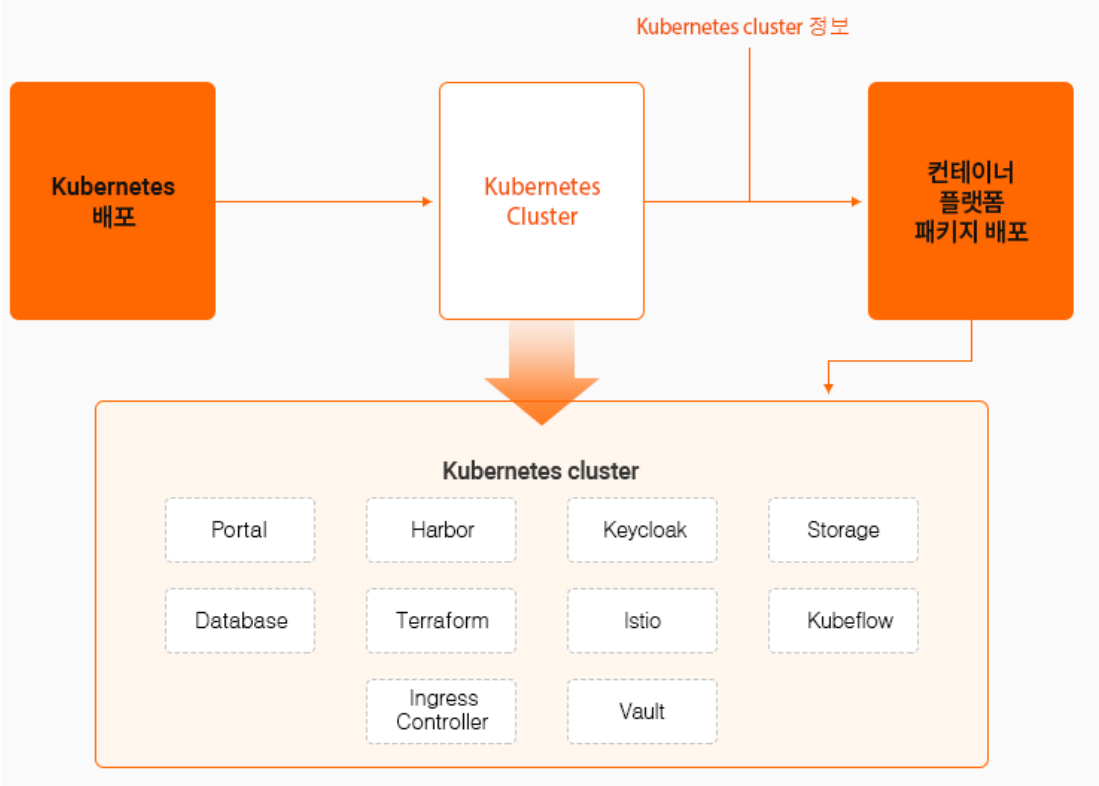
출처: 개방형 클라우드 플랫폼 대표포털

02 Container Platform 서비스 소개

» 배포방식

단독 배포

컨테이너 플랫폼을 단독으로 배포하여 독립된 쿠버네티스 환경을 제공하는 배포 방식



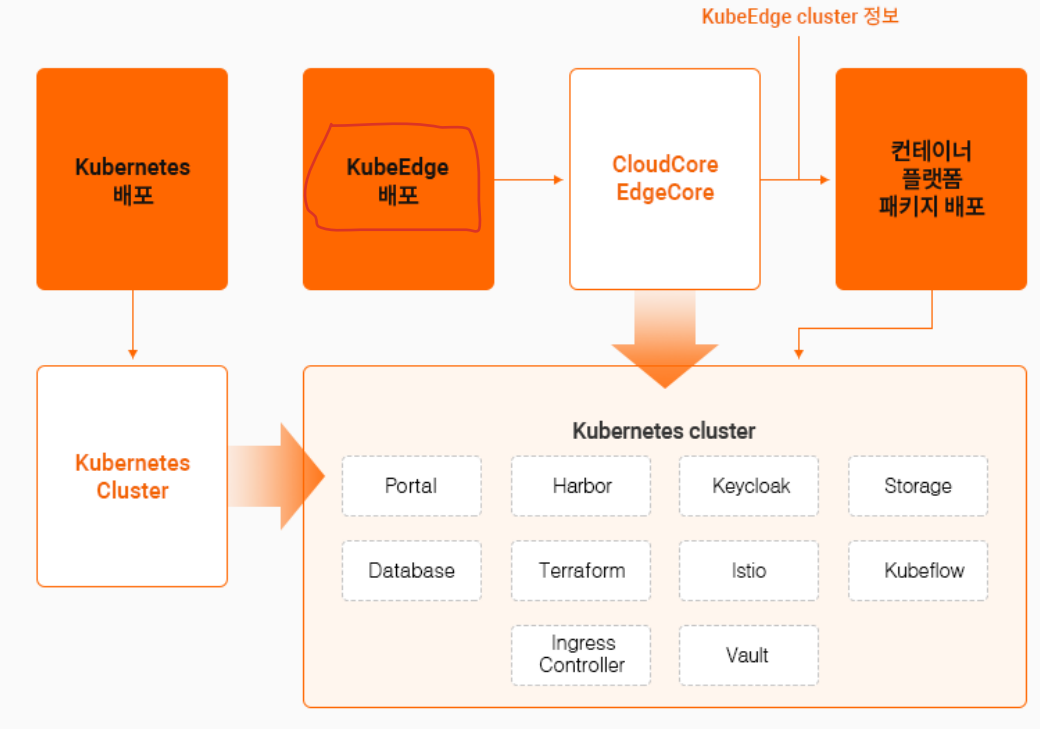
출처: 개방형 클라우드 플랫폼 대표포털

02 Container Platform 서비스 소개

» 배포방식

Edge 배포 설치 위치가 다름

- Edge Computing 기반 배포 방식으로 단독 배포와 같이 쿠버네티스 클러스터를 Edge 환경에 단독으로 배포하여 독립된 쿠버네티스를 제공하는 배포 방식



출처: 개방형 클라우드 플랫폼 대표포털

02 Container Platform 서비스 소개

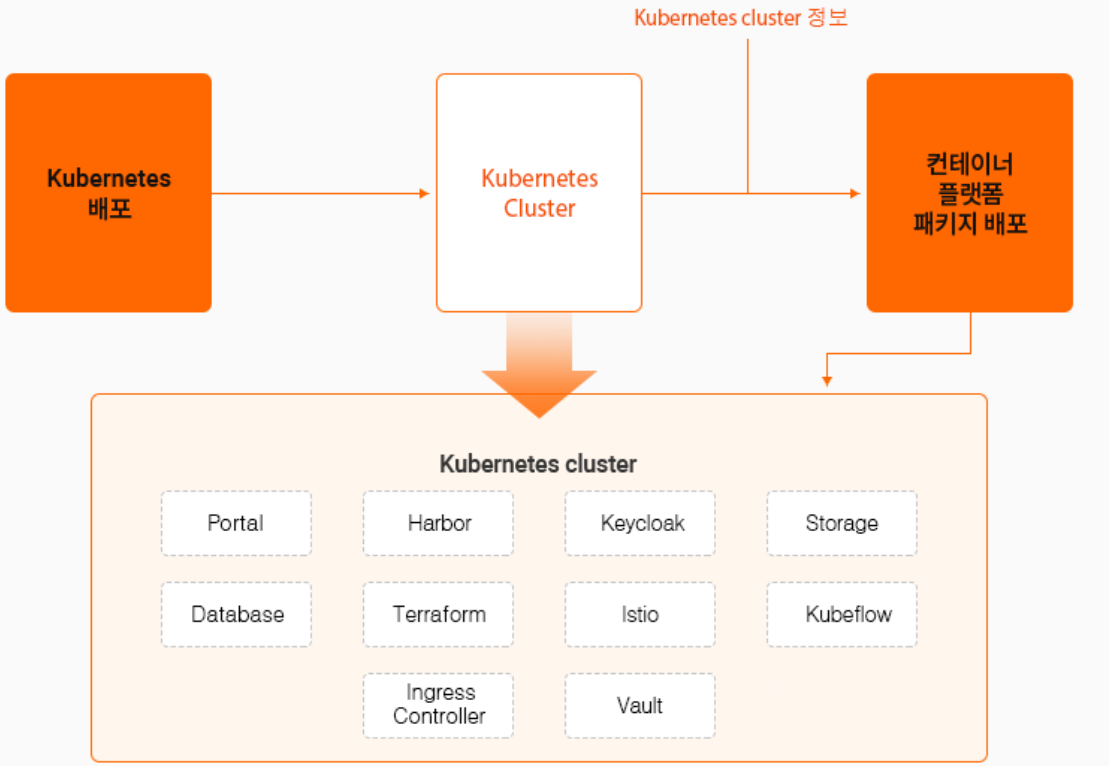
» 배포방식

요즘엔 안쓴다 그래서!

서비스형 배포

컨테이너 플랫폼을 애플리케이션 플랫폼에서 서비스 형태로 쿠버네티스 클러스터를 등록하여 사용하는 방식

cp와 ap가 같이사용되는 방식.



출처: 개방형 클라우드 플랫폼 대표포털

03 Container Platform 발전과정

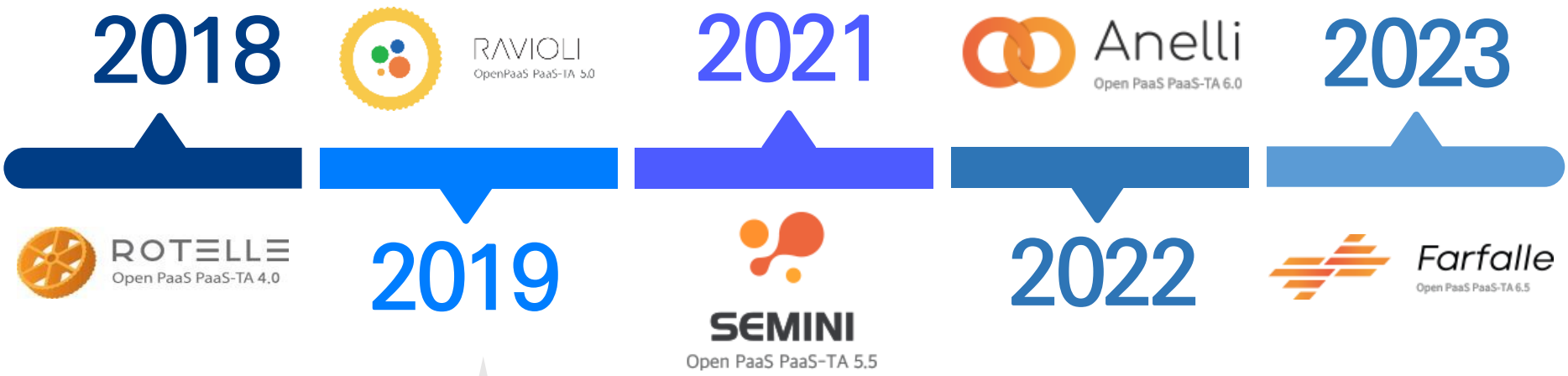
» 4.0 버전

- Bosh를 기반으로 Cloud Foundry Container Runtime을 활용하여 컨테이너 서비스 시작

출처: 개방형 클라우드 플랫폼 대표포털

03 Container Platform 발전과정

» 5.0 버전



- 컨테이너 서비스에 대한 DevOps 환경을 제공하기 위해 CI/CD 환경 개발
- 새로운 기술 도입으로 인한 배포 방식 다양화
- 기존 서비스 형태 배포 유지
- 단독 형태 배포, Edge 배포 개발

출처: 개방형 클라우드 플랫폼 대표포털

03 Container Platform 발전과정

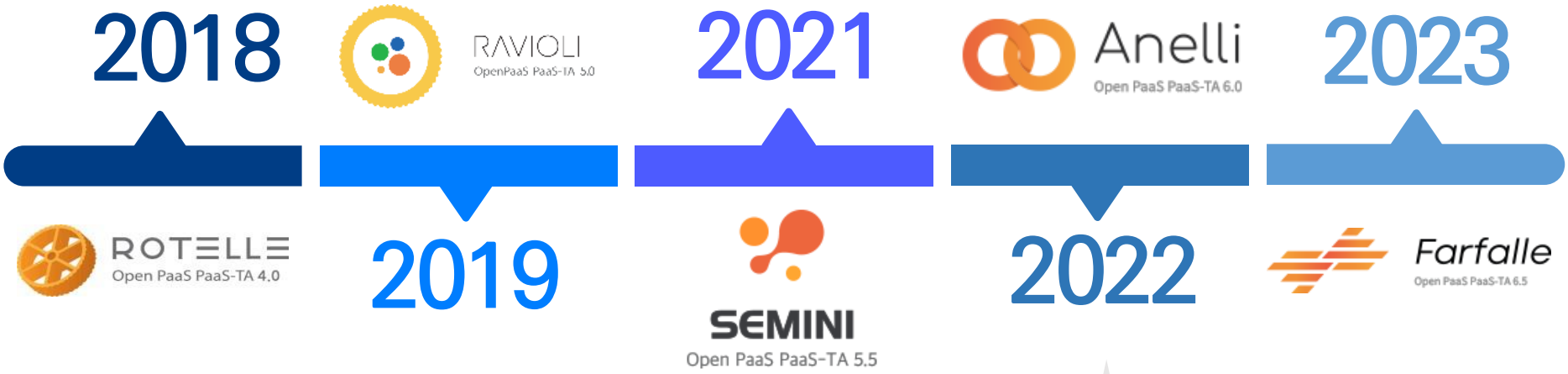
» 5.5 버전

- 컨테이너 플랫폼 가용성 확보 및 DevOps 개선
- 기존 서비스 형태 배포 유지
- 단독 형태 배포, Edge 배포 고도화

출처: 개방형 클라우드 플랫폼 대표포털

03 Container Platform 발전과정

» 6.0 버전



- 기존 단독 형태 배포, 서비스 형태 배포, Edge 샘플 제공 및 배포 고도화
- 클러스터를 위한 기본적인 스토리지 제공
- 패키지 관리자를 통한 배포 방식 채택 및 스크립트 단일화를 통한 배포 단순화
- HA(High Availability)구성 지원으로 운영 안정성 확대

출처: 개방형 클라우드 플랫폼 대표포털

03 Container Platform 발전과정

» 6.5 버전

- 포탈 글로벌 대시보드 사용 및 Metric-API를 통한 클러스터 상태 정보 수집 강화
- Terraform 도입으로 인한 멀티 클러스터 배포 및 관리 단순화
- Vault 도입으로 인한 Secret 정보 보안 강화
- Kubeflow 도입으로 인한 머신러닝 워크플로우 배포 단순화
- Ceph을 통한 클러스터 내 스토리지 Pod 구성

출처: 개방형 클라우드 플랫폼 대표포털

02 Container, Podman, K8S의 이해

== 쿠버네티스
k8s라고 불림

쿠버네티스 라이트 버전. 경량화된
==k3s라고 불림.

01. Container 이해
02. Podman 이해 배포를 도와주는 오픈 소스.
(컨테이너를 쉽게 만들고 쉽게 배포)
03. Kubernetes 이해 (1)
04. Kubernetes 이해 (2)

01 Container 이해

» 정의

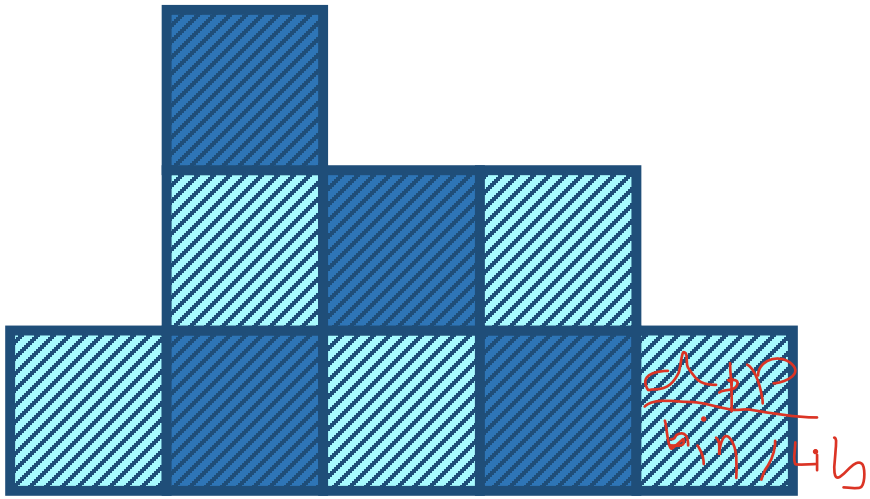
Container란

프레임워크, 커널, 어플리케이션 등등 환경

컨테이너

SW서비스를 실행하는 데 필요한
특정 버전의 프로그래밍 언어 런타임 및
라이브러리와 같은 종속 항목과
애플리케이션 코드를 함께 포함하는 경량 패키지

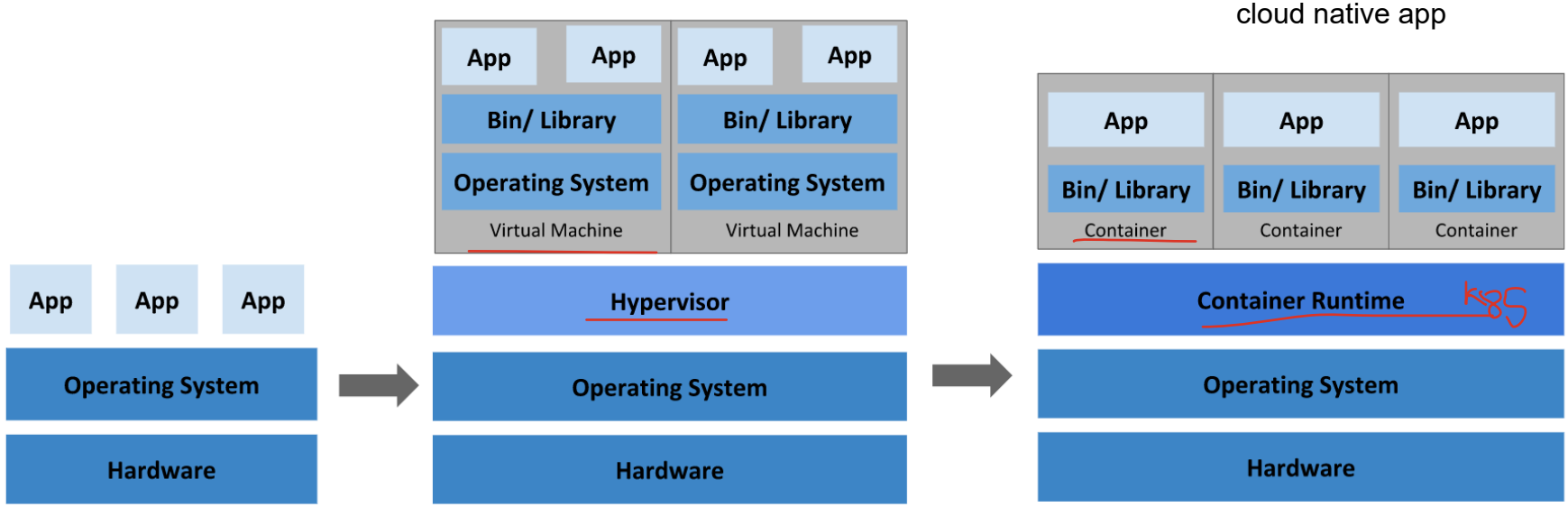
애플리케이션의 코드를 관련 구성 파일,
라이브러리 및 앱 실행에 필요한 종속성과
함께 번들로 제공하는 SW패키지



출처 : Google Cloud 홈페이지, Azure 홈페이지

01 Container 이해

» 등장배경



전통적인 배포 시대

물리서버에 애플리케이션 설치
▶ 리소스 고갈
▶ 물리서버 추가
▶ 비용 증가

가상화된 배포 시대

단일 물리서버
▶ VM 다수 실행
▶ VM 간 애플리케이션 격리
▶ 보안성 강화, 비용 감소

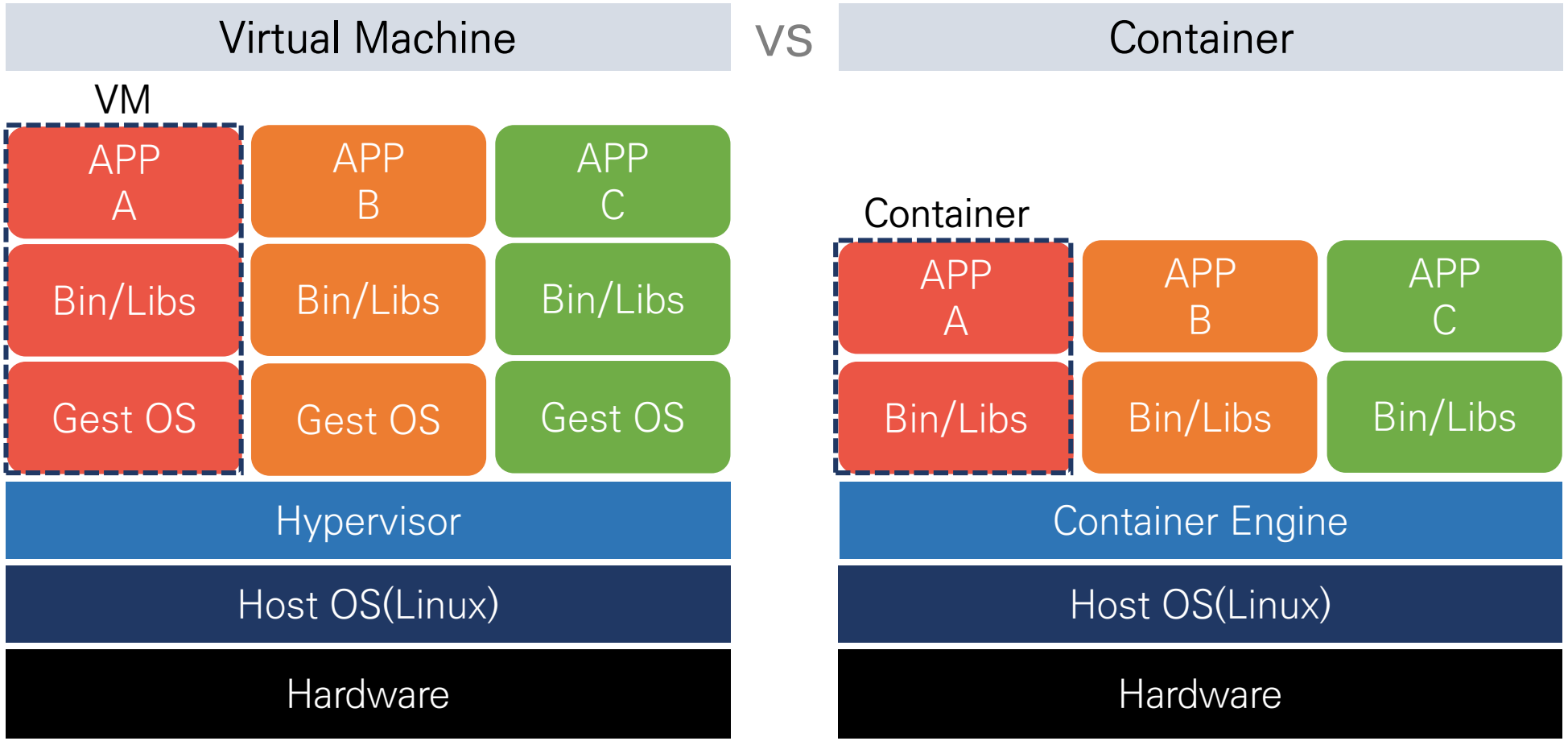
컨테이너 개발 시대

단일 물리서버
▶ 컨테이너(운영체제 공유)
▶ 애플리케이션 격리 완화
▶ 이식성 제고

출처 : 쿠버네티스 홈페이지

01 Container 이해

» Hypervisor 기술과 Container 기술



출처 : 삼성SDS 인사이트리포트, https://www.samsungsds.com/kr/insights/middleware_in_containers.html

01 Container 이해

» Container 특징

- 쉽고 효율적인 컨테이너 이미지 생성으로 **애플리케이션을 기민하게 생성, 배포할 수 있음**
- 안정적, 주기적으로 컨테이너 이미지를 빌드, 배포할 수 있기 때문에 **지속적인 개발, 통합 및 배포** 가능
- 빌드/릴리즈 시점에 애플리케이션 컨테이너 이미지를 만들기 때문에 **개발과 운영의 관심사 분리**
- 애플리케이션의 헬스와 그 밖의 시그널을 볼 수 있어 **가시성(Observability)**이 높음
- 랩탑, 클라우드에서 모두 동일하게 구동되므로 **개발, 테스팅 및 운영 환경에 걸친 일관성** 확보
- 주요 퍼블릭 클라우드와 어디에서든 구동되어 **클라우드 및 OS 배포판 간 이식성**이 높음
- 논리적 리소스를 사용하는 OS 중심으로 추상화 수준이 높아져 **애플리케이션 중심 관리 가능**
- 느슨하게 묶이고, 분산되고, 유연하며, 자유로운 **마이크로 서비스** 형태로 애플리케이션 배포, 관리
- **리소스가 격리**되어 애플리케이션 성능을 예측할 수 있음
- 고효율 고집적의 **리소스 사용량**

출처 : 쿠버네티스 홈페이지

01 Container 이해

» Container Engine & Container Runtime

Container Engine

- 컨테이너를 관리하기 위한 API나 CLI 도구를 제공하는 소프트웨어
- 컨테이너 엔진은 사용자 입력을 받고, 컨테이너 이미지를 꺼내고(pull), 컨테이너 실행 방법을 명시한 메타데이터를 만든 다음, 컨테이너 런타임에 이 정보들을 전달
- 도커 엔진(docker-ce)부터 레드햇의 파드맨(Podman), 로켓 컴퍼니의 rkt 등

Container Runtime

기반

OCI를 준수하면 작동 가능

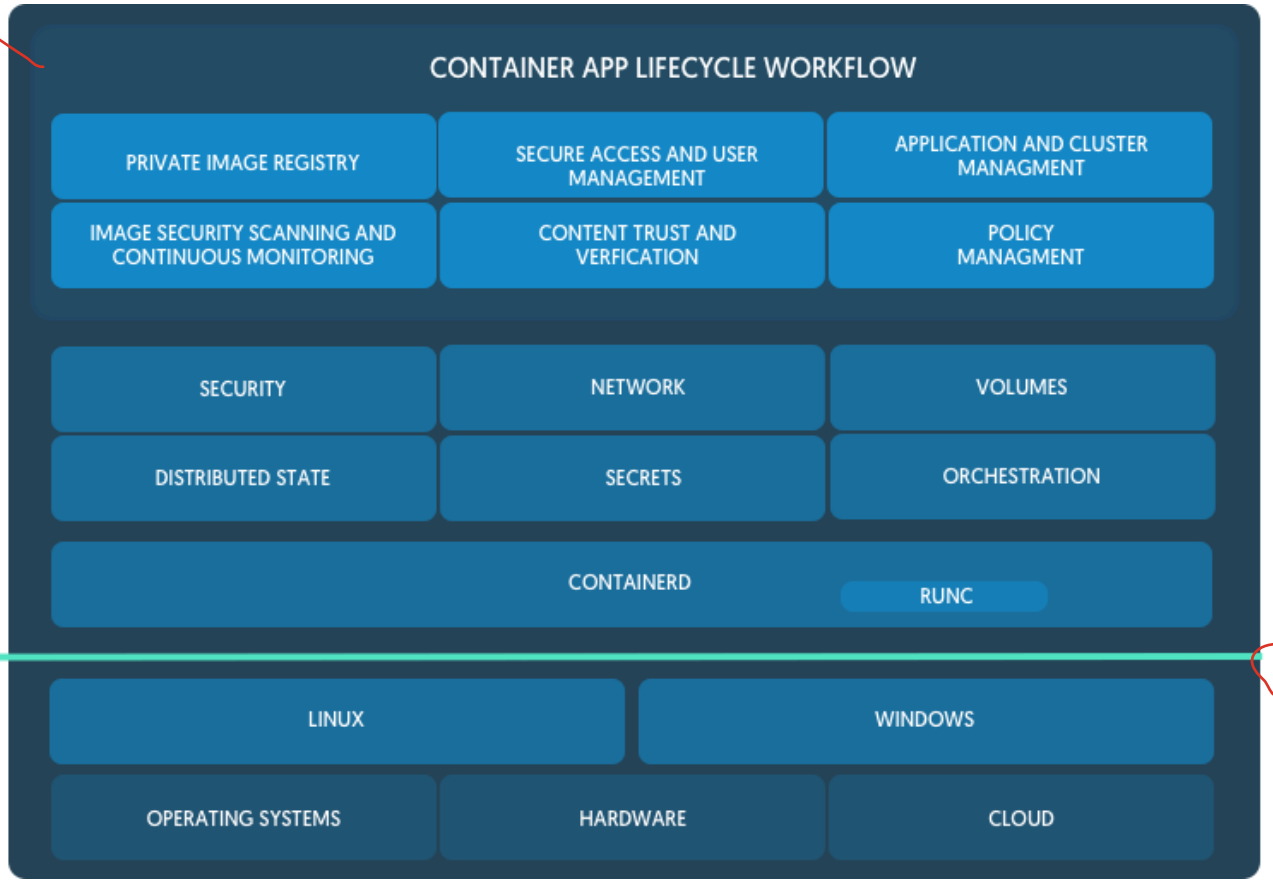
- 루트 파일시스템과 메타 데이터(spec file)를 받아 컨테이너를 실행하는 도구
- 가장 일반적으로 쓰이는 런타임은 OCI(Open Container Initiative)를 준수하는 runC
- 컨테이너-디 (containerd), 크라이-오 (cri-o) 등도 runC에 의존

출처: Redhat 블로그(https://developers.redhat.com/blog/2018/02/22/container-terminology-practical-introduction#namespace)

01 Container 이해

» Container Engine & Container Runtime

컨테이너 중심 체계



쿠버네티스가 관장

규정 기준을 세워서

출처: 도커 공식 블로그(https://www.docker.com/blog/)

02 Podman 이해 혹은 Docker

둘다 비교해보자. 한국에서는 도커가 유행. 외국은 비등 비등

» Podman이란?

**리눅스 시스템에서 컨테이너 및 컨테이너 이미지를
실행, 관리, 배포할 수 있도록 설계된 Daemonless 컨테이너 엔진**

유일한 특징
쿠버네티에서 실행되는 컨테이너 단위

podman

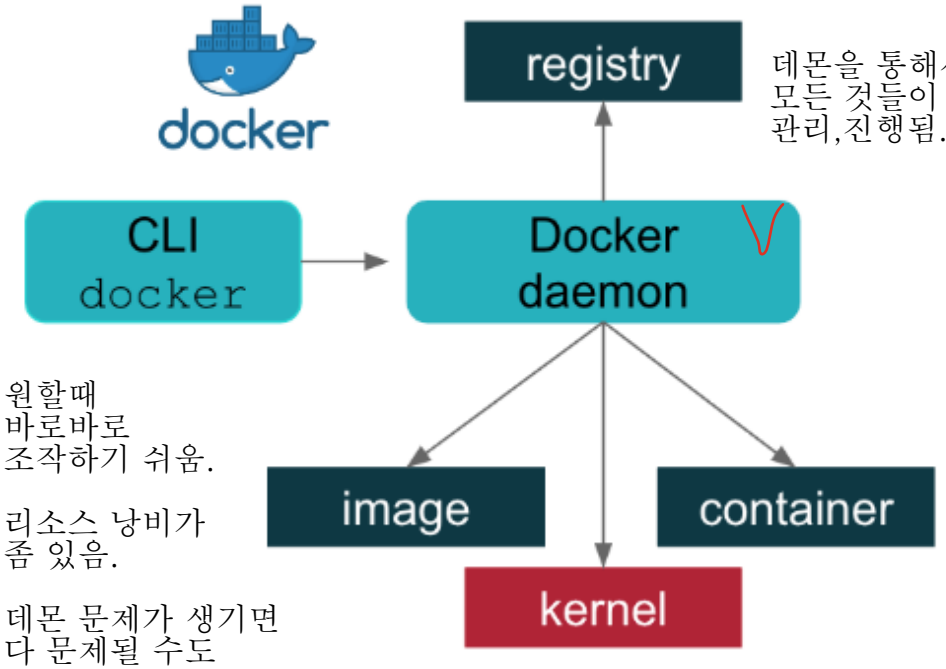
- Podman은 Pod Manager tool의 약자
- Rootless 컨테이너와 Pod를 위한 리소스를 실행하고 분리하여 더 손쉽게 액세스할 수 있는 컨테이너 환경을 만드는 동시에 보안 리스크를 줄일 수 있음
- OCI(Open Container Initiative)를 준수하며 Docker와 거의 대부분의 CLI Command가 동일
- 컨테이너를 관리하기 위해 RESTful API를 배포

* Daemon – 서비스 요청에 대해 응답하기 위해 오랫동안 (long-running) 실행중인 백그라운드 프로세스

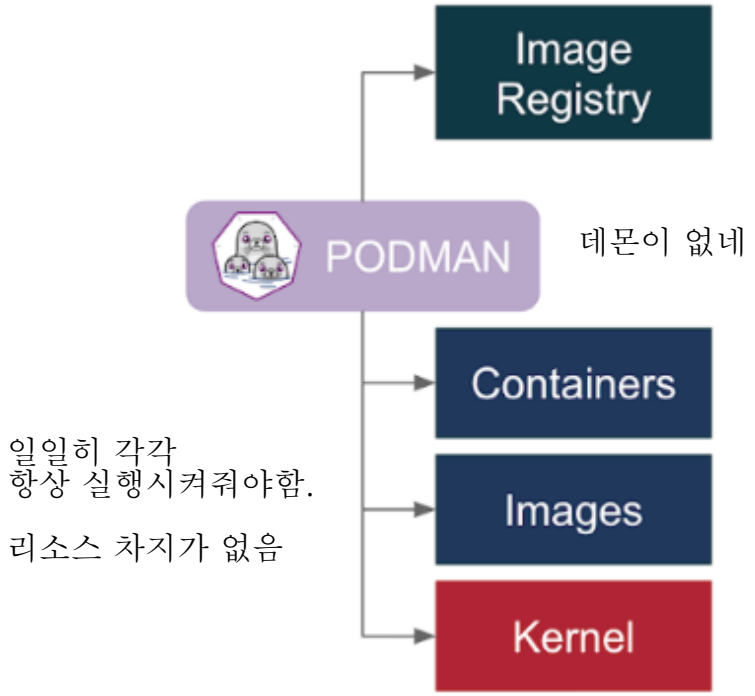
출처: 파드맨 공식 문서(https://docs.podman.io/en/latest/#)

02 Podman 이해

» Docker 와 Podman



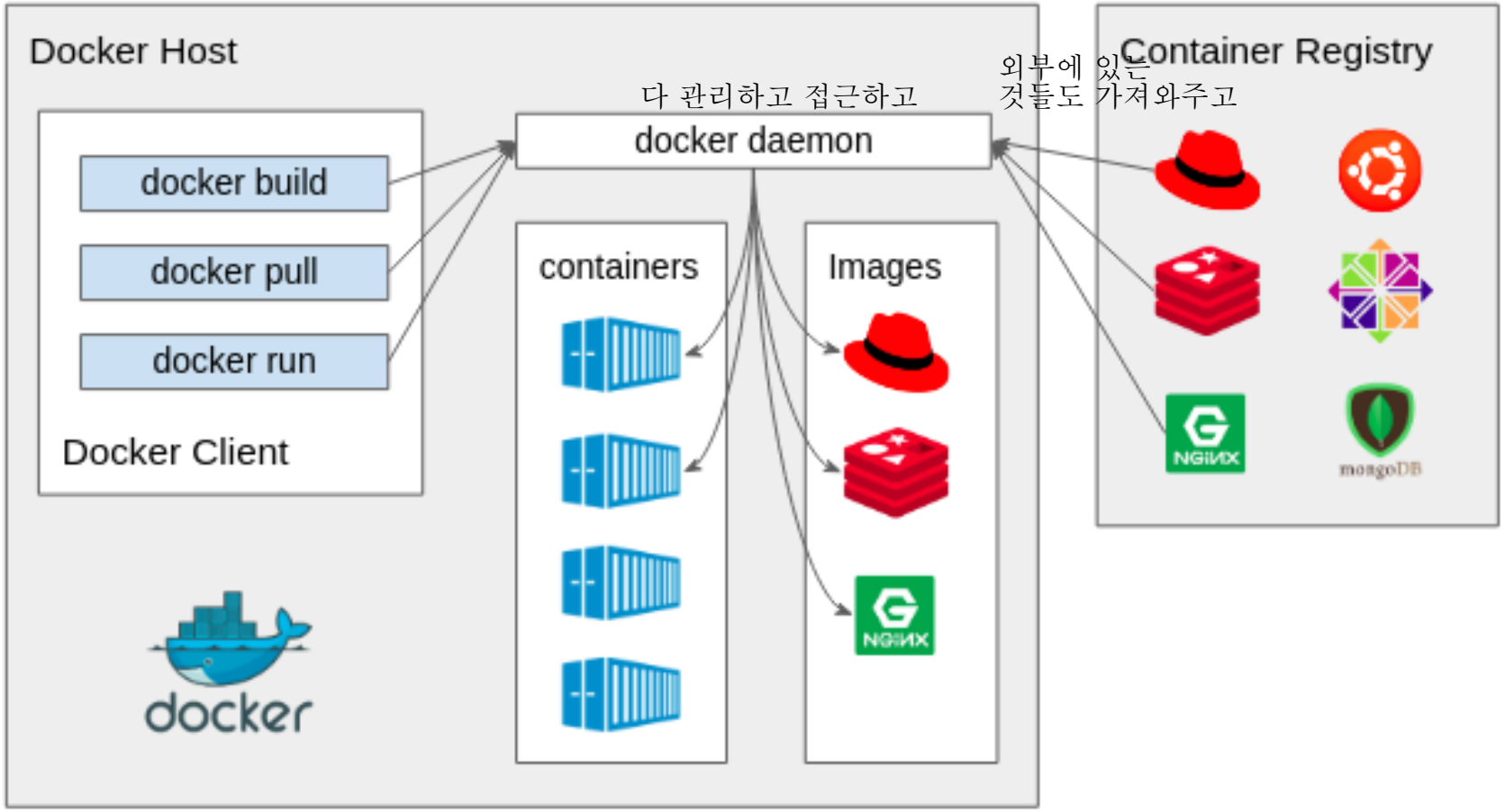
VS



출처 : Open Source Summit Japan 2019 발표자료(https://www.slideshare.net/DanielOh20/podman-buildah-and-quarkus-the-latest-in-linux-containers-technologies)

02 Podman 이해

» Podman 과 Docker의 차이



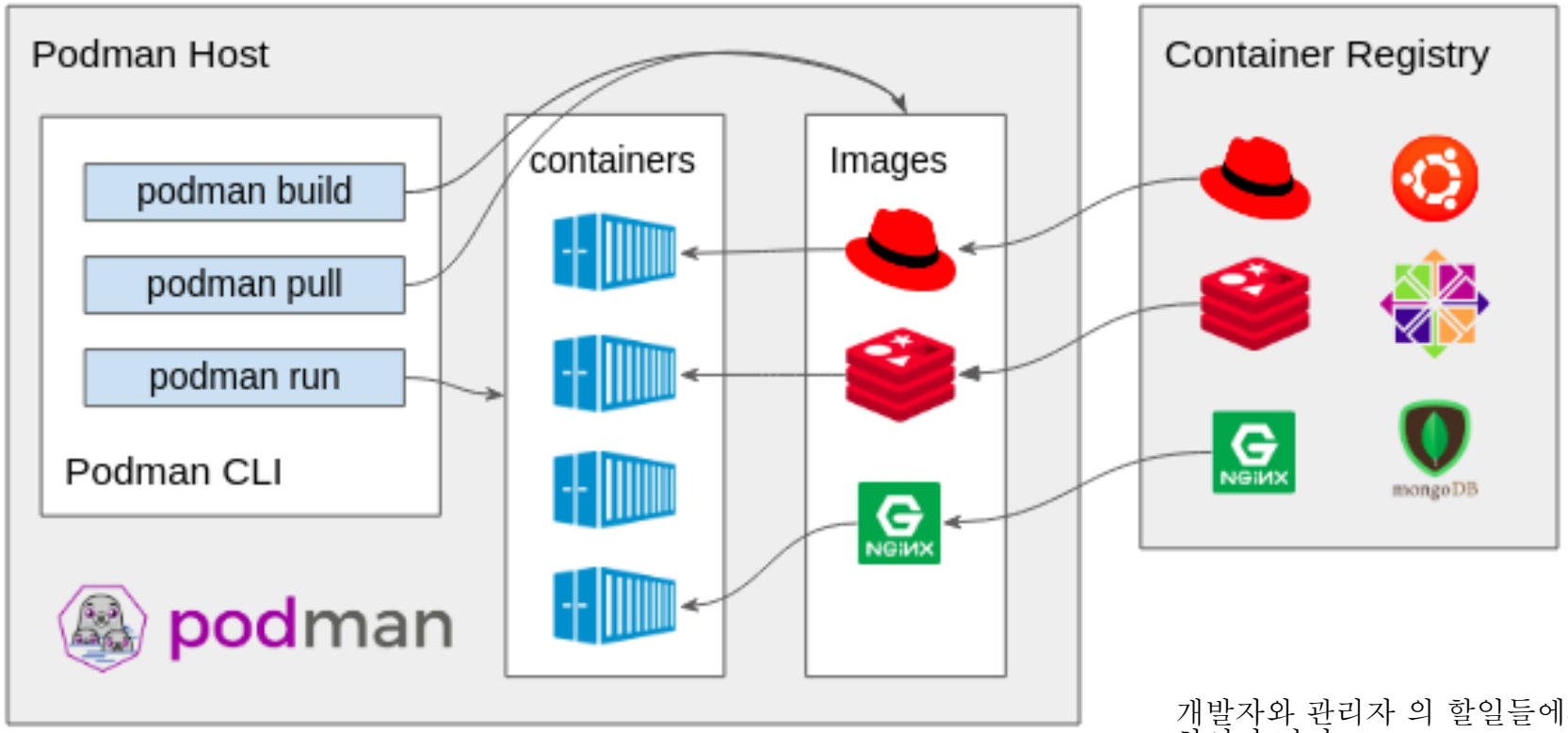
출처 : Open Source Summit Japan 2019 발표자료(https://www.slideshare.net/DanielOh20/podman-buildah-and-quarkus-the-latest-in-linux-containers-technologies)

02 Podman 이해

» Podman 과 Docker의 차이

운영에 대한 장단점이 발생함.

니가 다 알아서 해
컨테이너도 너가 알아서 해



개발자와 관리자 의 할일들에서
차이가 난다.

출처 : Open Source Summit Japan 2019 발표자료(https://www.slideshare.net/DanielOh20/podman-buildah-and-quarkus-the-latest-in-linux-containers-technologies)

03 Kubernetes 이해 (1)

» Kubernetes 개요

쿠버네티스란

kubernetes

컨테이너의 모드것들을 담당하는

컨테이너 포함 다 관리

컨테이너화된 애플리케이션을
자동으로 배포, 스케일링 및 관리해주는
오픈소스 시스템

컨테이너화된 워크로드와 서비스를
관리하기 위한
이식성이 있고, 확장가능한 오픈소스 플랫폼

PAASSSS

출처 : 쿠버네티스 홈페이지

03 Kubernetes 이해 (1)

» Kubernetes 특징

쿠버네티스 특징

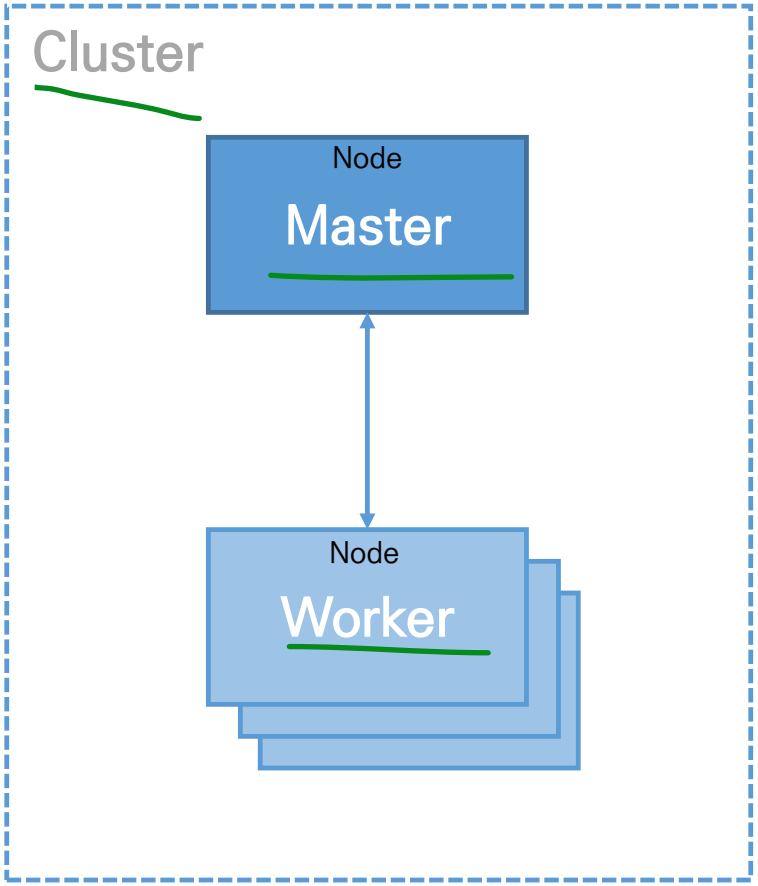
- 서비스 디스커버리와 로드 밸런싱
- 스토리지 오케스트레이션
- 자동화된 롤아웃과 롤백
- 자동화된 빈 패킹(bin packing)
- 자동화된 복구(self-healing)
- 시크릿과 구성 관리

출처 : 쿠버네티스 홈페이지

03 Kubernetes 이해 (1)

» Kubernetes Cluster 구성

클러스터 구성요소

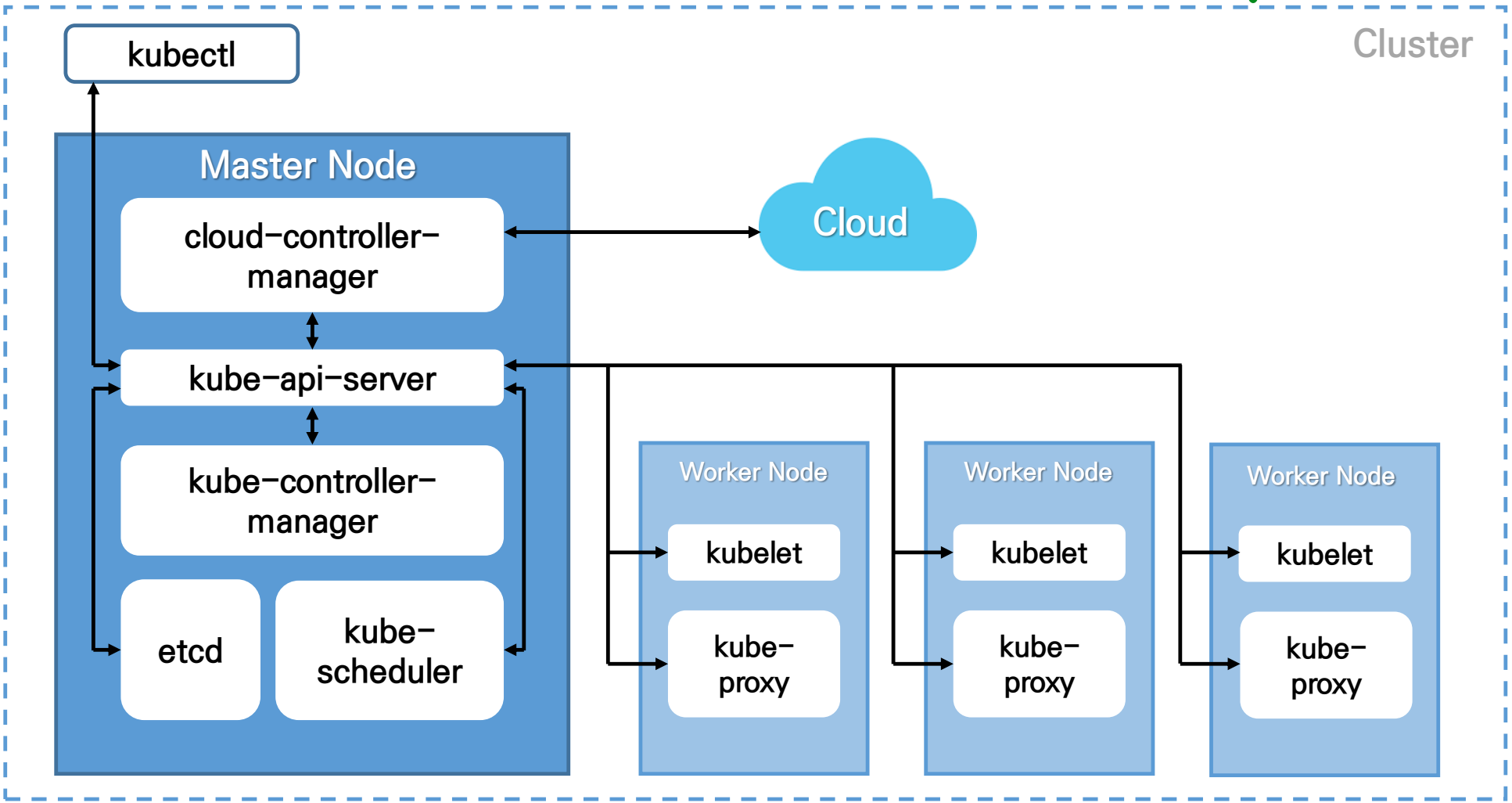


출처 : 쿠버네티스 홈페이지

- **Cluster**
  컨테이너화된 애플리케이션을 실행하기 위한 일련의 노드 머신의 집합
- **Master**
  클러스터 전체를 컨트롤하며 내부에 있는 모든 노드를 관리하는 가상 머신
- **Worker**
  마스터에 의해 명령을 받고 실제 컨테이너들이 생성되고 일을 하는 가상 머신

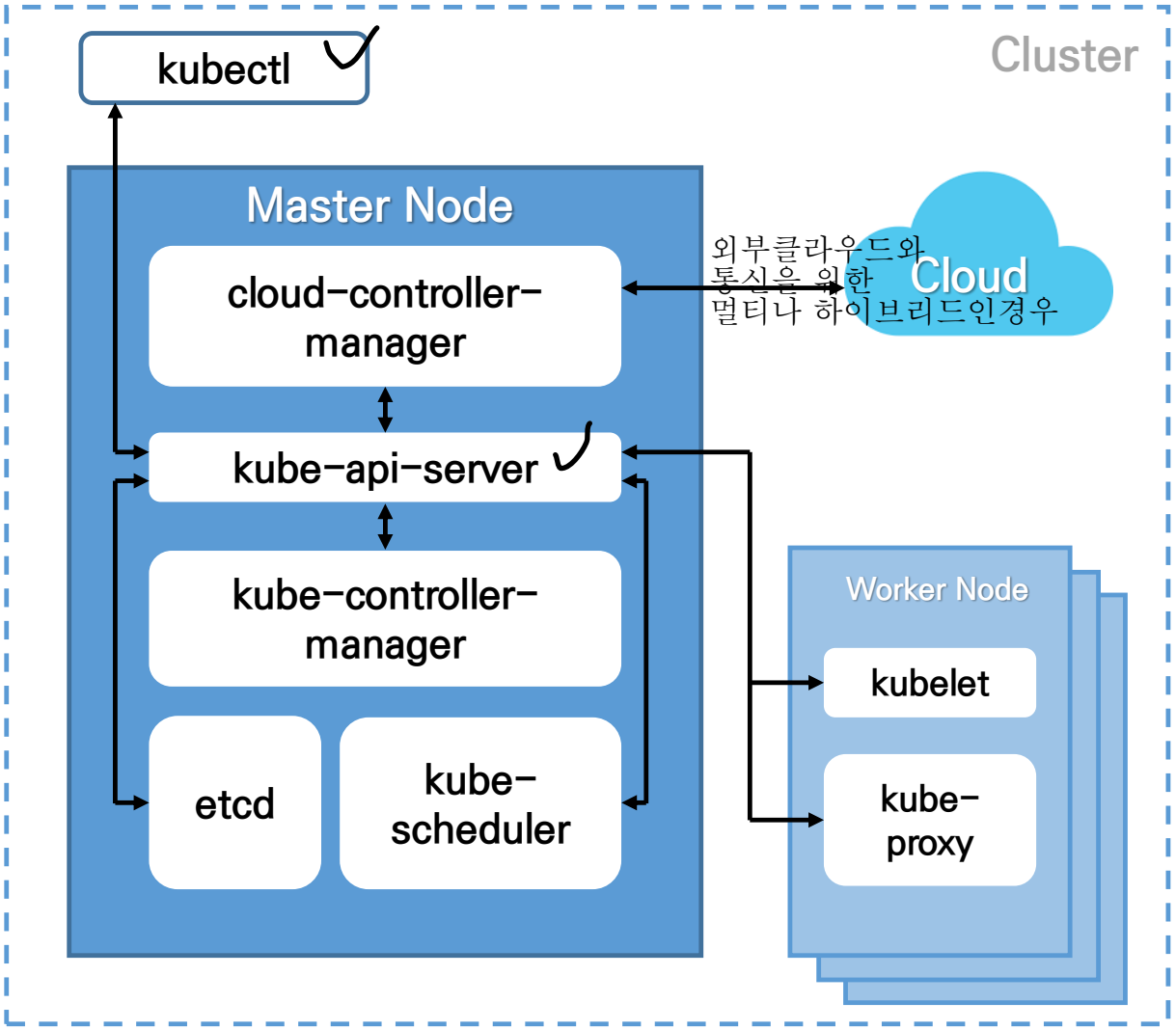
03 Kubernetes 이해 (1)

» Kubernetes Cluster 구성



03 Kubernetes 이해 (1)

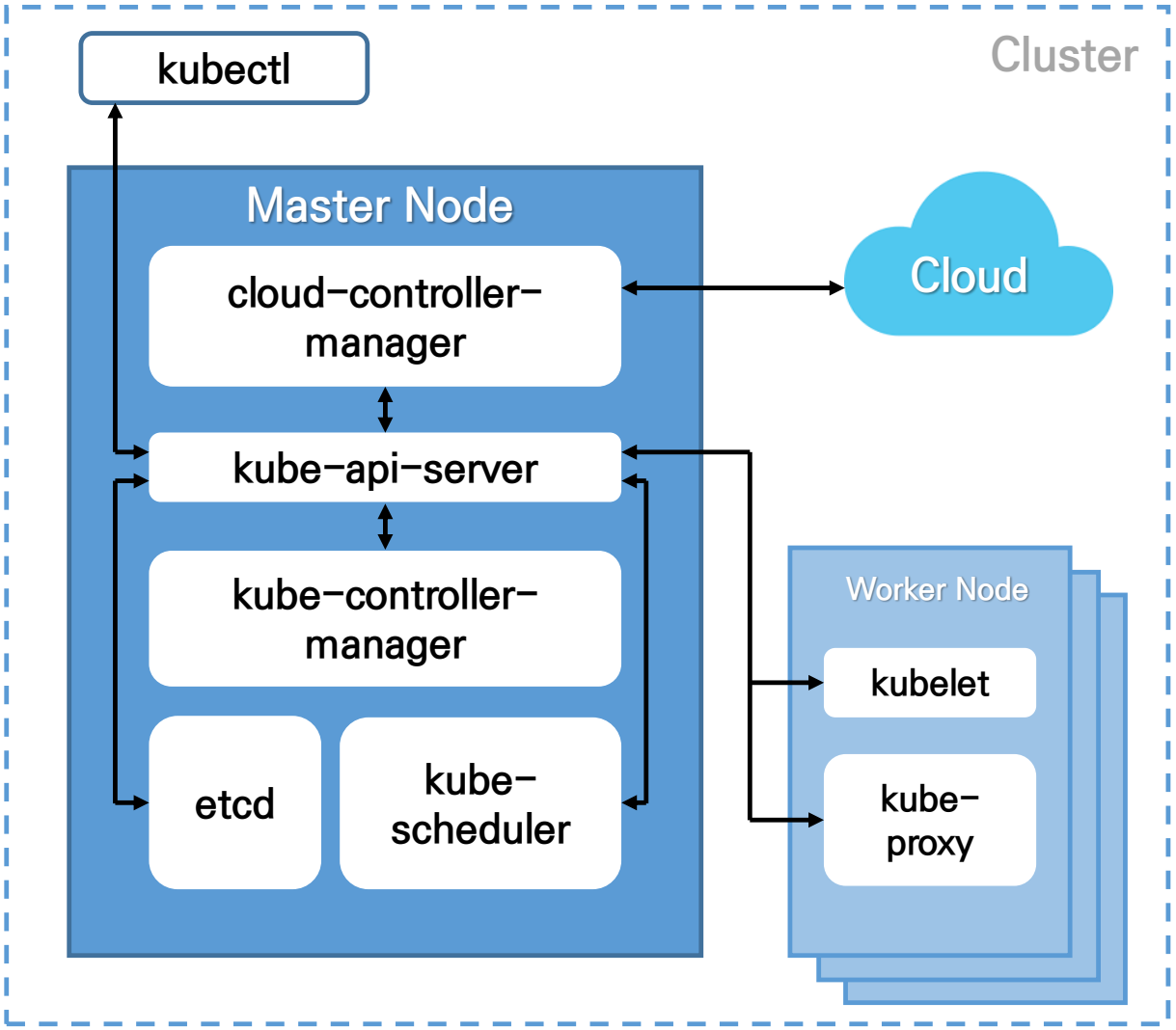
» Kubernetes Cluster 구성



- **kube-api-server**
  쿠버네티스의 모든 통신은 kube-api-server를 통해 통신
- **kube-controller-manager**
  파드들을 관리하는 각각의 컨트롤러를 제어하는 역할
- **kube-scheduler**
  리소스를 자원 할당이 가능한 노드에 할당하는 역할
  실행순서 어느 워커노드에 할당할지
- **etcd**
  클러스터 내의 모든 세부적인 데이터를 저장하는 키-값 저장소

03 Kubernetes 이해 (1)

» Kubernetes Cluster 구성



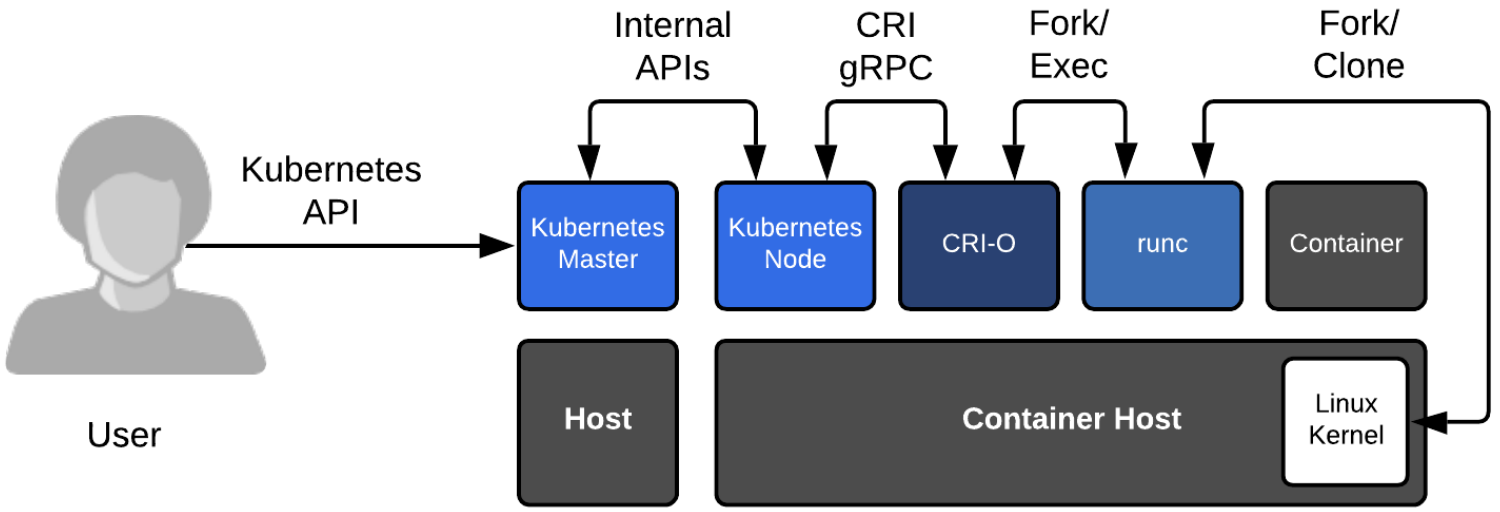
✗

- cloud-controll-manager
  컨트롤러들을 클라우드 서비스와 연결하여 관리
- kubelet
  모든 노드에서 실행되며 컨테이너 실행 및 지속적인 헬스체크를 통해 마스터의 kube-api-server와 통신을 하는 에이전트
- kube-proxy
  클러스터 내부의 가상 네트워크를 설정하여 관리

03 Kubernetes 이해 (1)

» Container Runtime Interface(CRI)

컨테이너 런타임 인터페이스는 쿠버네티스에서
다양한 컨테이너 런타임을 사용할 수 있게 해주는 API



How containers run in a Kubernetes cluster

출처 : Red Hat 블로그(https://cloud.redhat.com/blog/kubernetes-is-removing-docker-support-kubernetes-is-not-removing-docker-support)

03 Kubernetes 이해 (1)

» Kubernetes 배포 도구

쿠버네티스 == 하나의 소프트웨어. 오케스트레이션 SW.
쿠버네티스들을 편리하게 설치하는 도구들
쿠버네티스 설치에는 많고 다양한 환경설정들을 해야하는데 이것들을 쉽게 하게 해주는 것들

일반적인 서버 클러스터 환경에서도
쿠버네티스를 쉽게 설치할 수 있게 해주는 관리 툴로서
클러스터를 빠르고 일관되게 설정할 수 있음

자동화된 프로비저닝 시스템으로
AWS에 쉽고 빠르게 쿠버네티스 클러스터를 설치 가능

Ansible을 통해 쿠버네티스 클러스터를 유연하고 쉽게
배포 및 관리할 수 있는 강력한 오픈 소스 툴

* GCE, Azure, OpenStack, AWS, vSphere, Equinix Metal(전 Packet),
Oracle Cloud 등 여러 플랫폼에서 쿠버네티스 클러스터 설치 가능

출처 : 쿠버네티스 홈페이지

03 Kubernetes 이해 (1)

이번주는 용어 위주로

» Namespace

마스터 노드 : 관리하는 노드

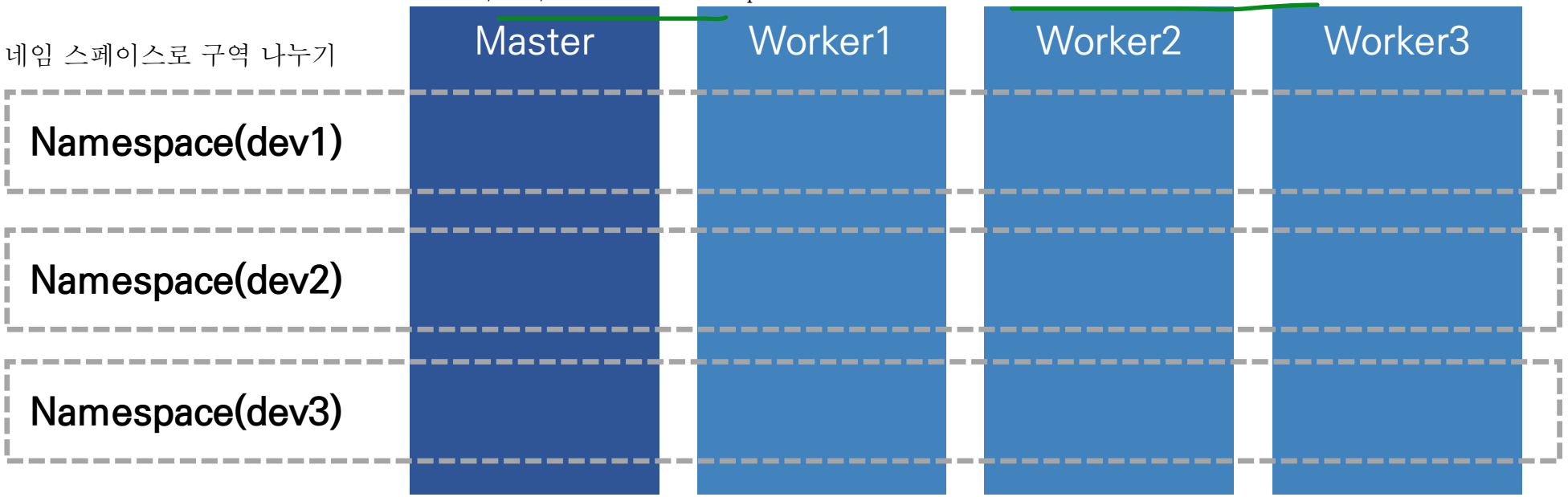
워커 노드 : 실제 실행하는 노드

네임스페이스는 클러스터를 논리적으로 분리하여 사용하는 것을 의미

쿠버네티스 내부는 2가지 노드로 구성됨. 마스터 노드==control plane

워커 노드==data plane

네임 스페이스로 구역 나누기



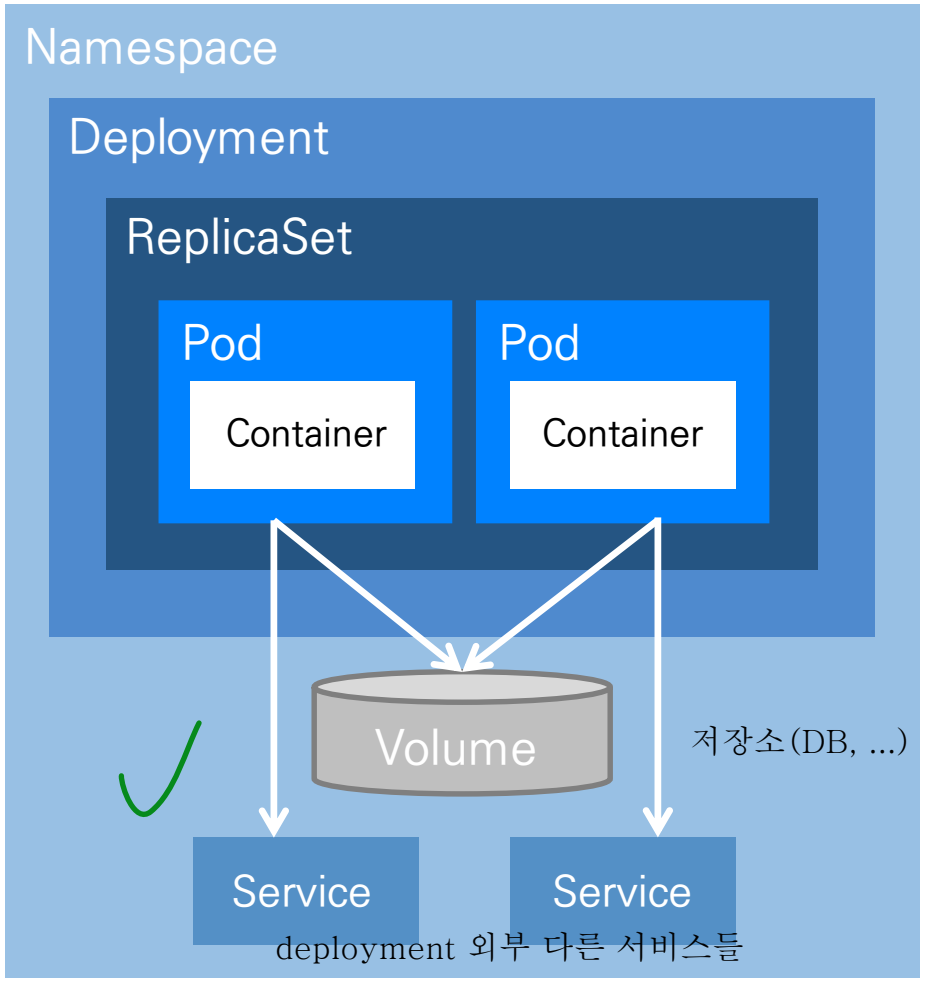
- 네임스페이스는 물리적인 마스터/워커 노드를 논리적인 단위로 분리한 리소스이며 각 사용자는 권한이 부여된 네임스페이스만 접근 가능

  예) dev1 네임스페이스에 대한 권한을 받은 사용자는 dev1만 접근 가능

03 Kubernetes 이해 (1)

쿠버네티스가 실행되고 관리하는 영역

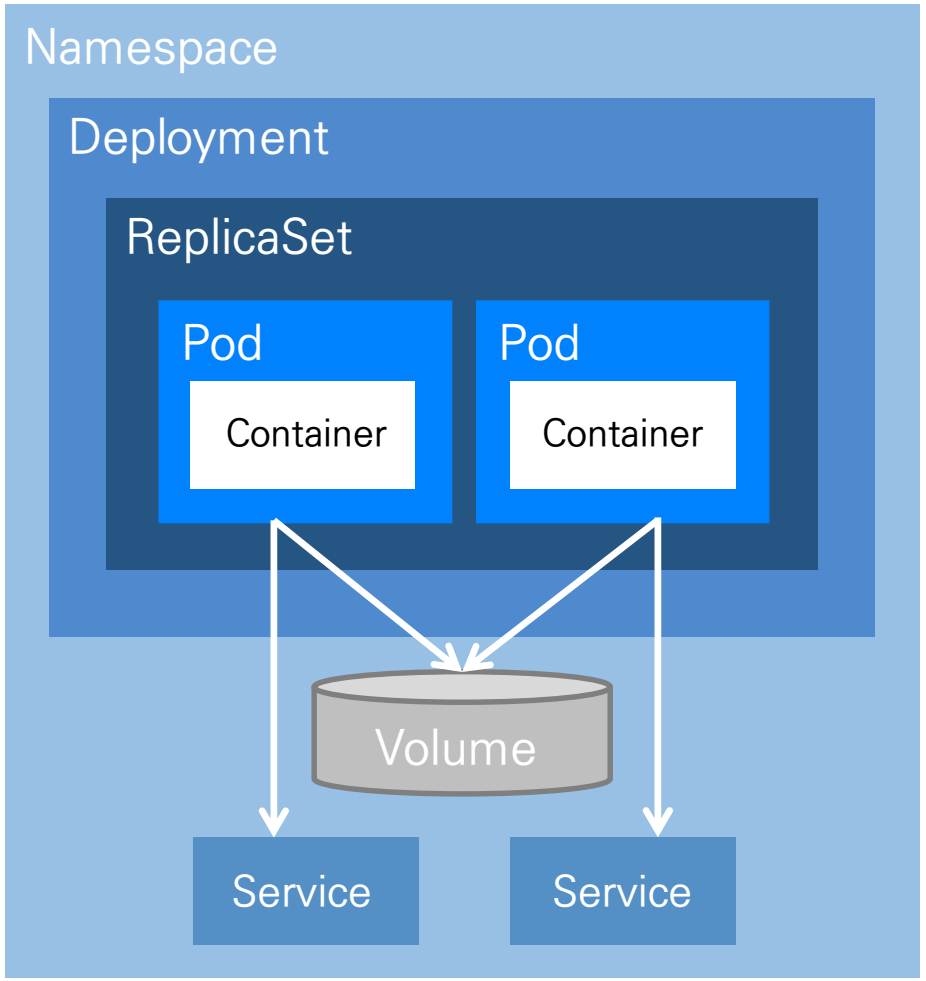
» Resource



- **Namespace** deployment가 들어가 있는
  동일한 물리 클러스터 하나를 여러 개의 논리적인 단위로 나누는 리소스
- **Deployment** 전체 MSA를 관리하는. MSA의 전체 세트
  레플리카셋을 관리하면서 실행시켜야 할 파드의 배포 및 관리를 하는 리소스
- **ReplicaSet** 파드 여러개를 관리하는 애
  파드의 개수를 유지하고 관리하는 리소스
- **Pod**
  실제로 컨테이너가 파드에서 실행되며 컨테이너를 돌리는 최소한의 단위
  도커나 파드맨이 실행시킨 단위
  배포하고 실행하면 기본적으로 3개 만들어짐.
  복제되는게 기본임. 파드마다 내부적으로 IP가 다 따로 있다

03 Kubernetes 이해 (1)

» Resource



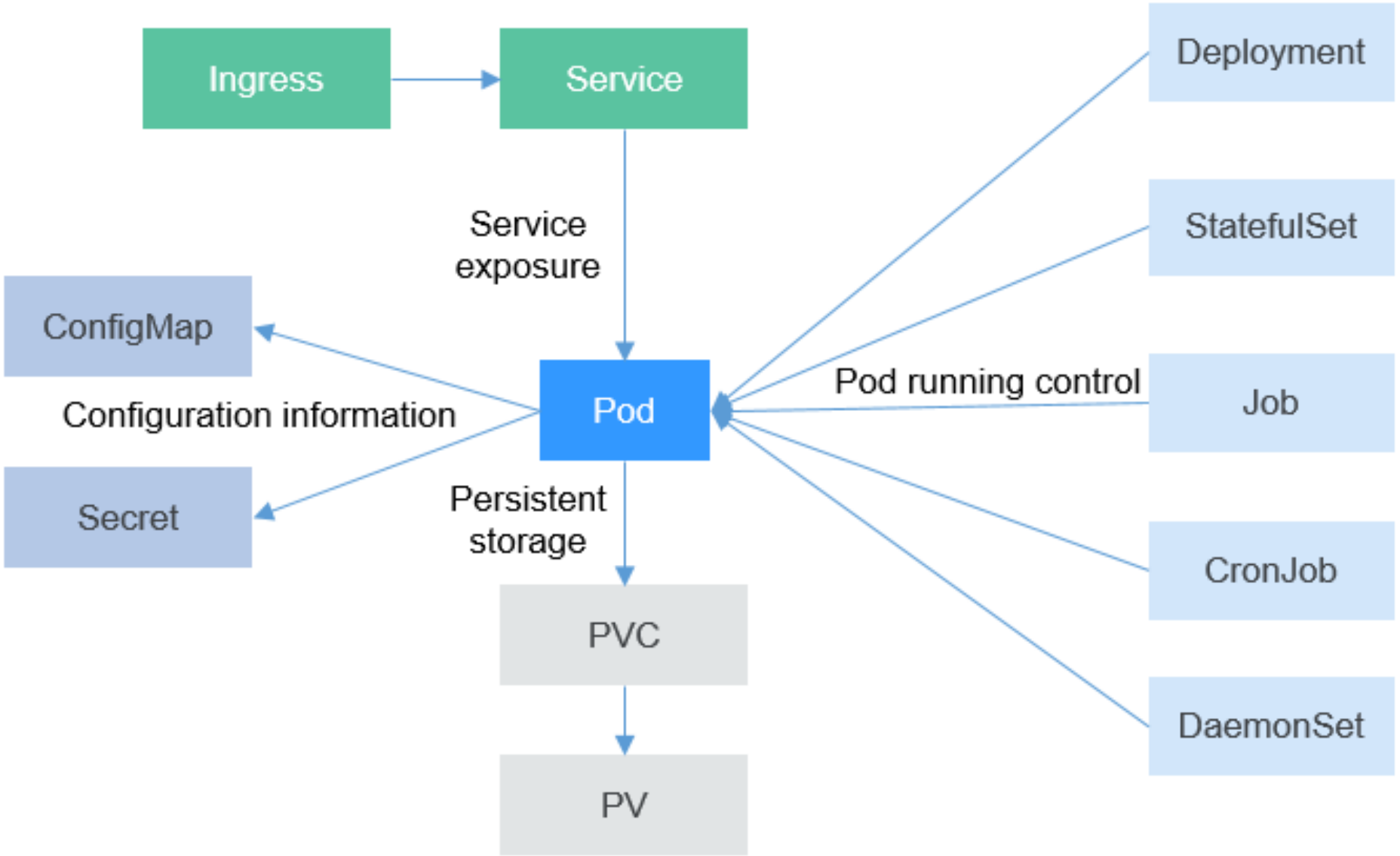
- **Service**
  클러스터 외부에서 클러스터 내부 파드에 접근 할 수 있도록 IP를 제공하는 리소스

- **Volume**
  파드의 일부분으로 동일한 파드 내의 컨테이너끼리는 볼륨 공유가 가능하며 컨테이너의 데이터를 보관하여 유지시키는 역할

mid-term

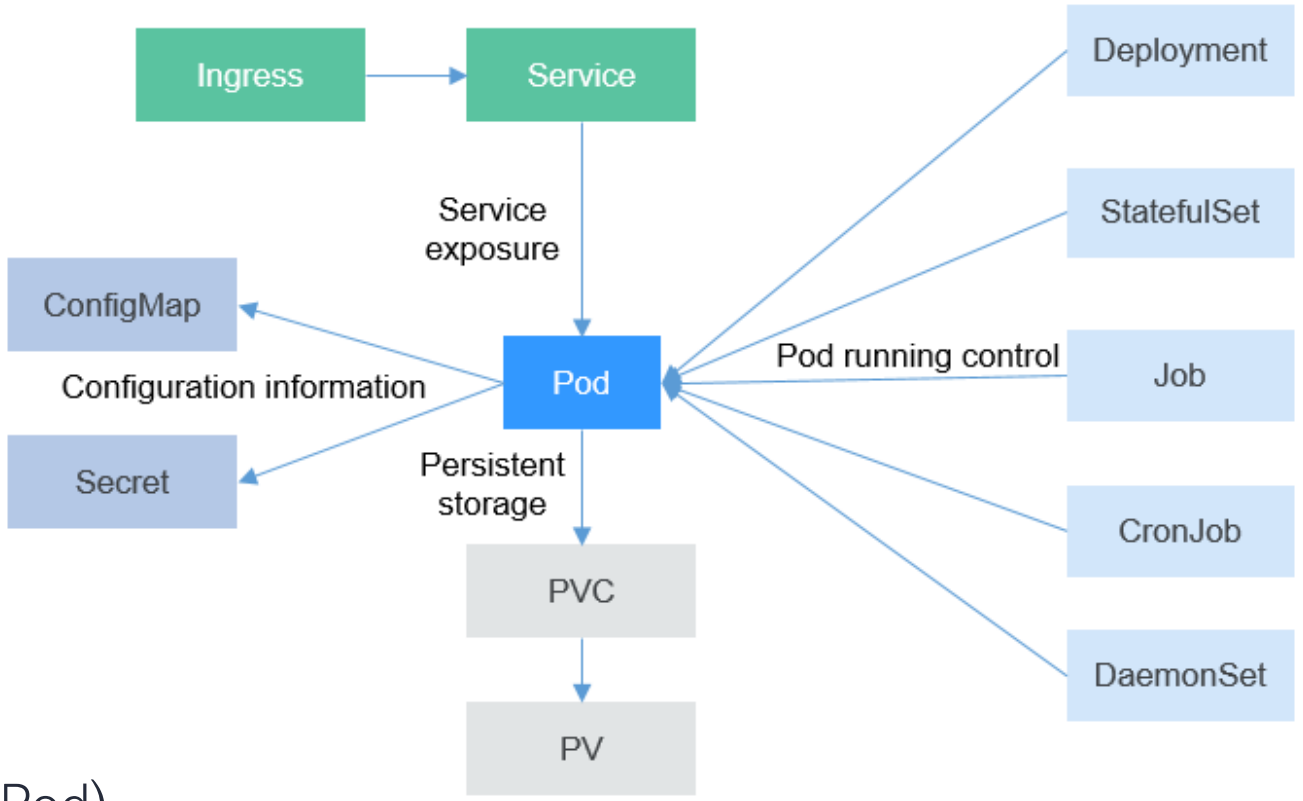
03 Kubernetes 이해 (1)

» Resource



03 Kubernetes 이해 (1)

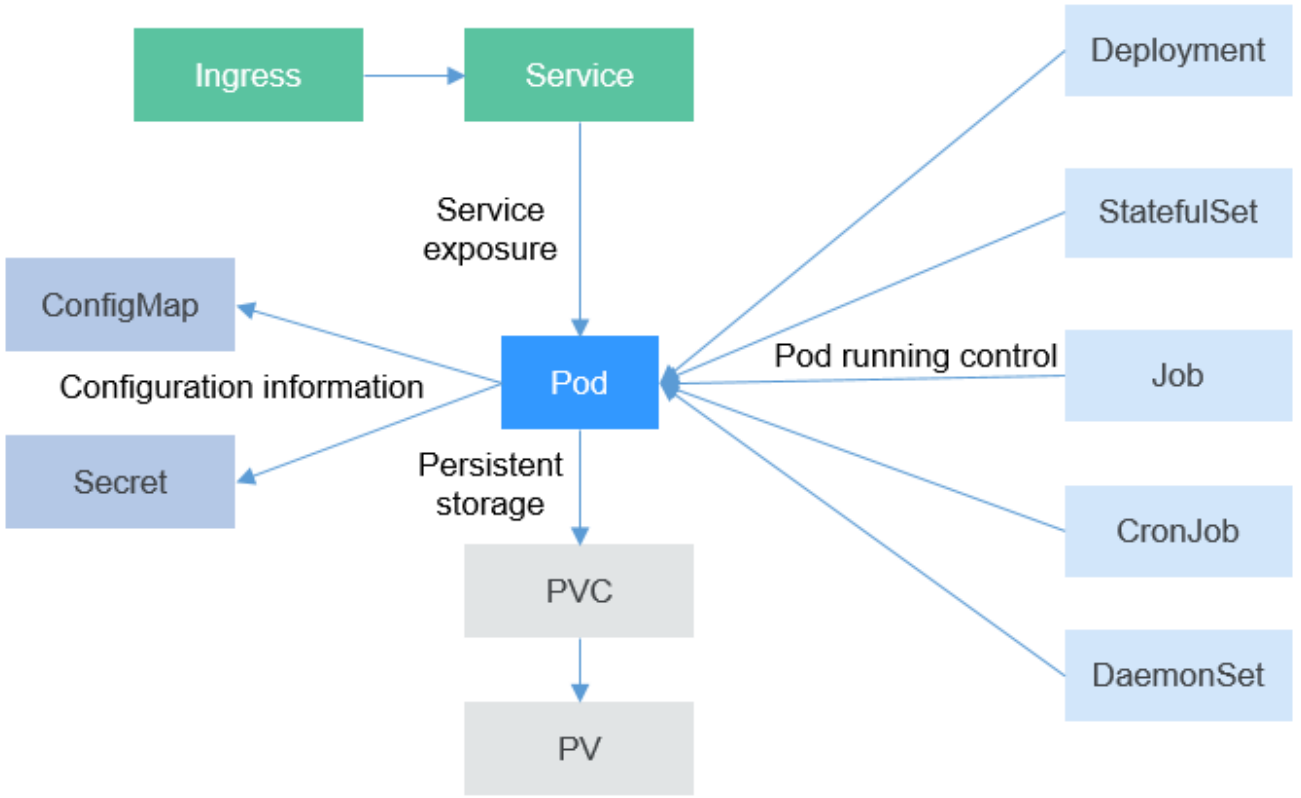
» Resource



- 파드(Pod)
  - 쿠버네티스에서 생성하거나 배포하는 가장 작고 간단한 단위
  - 하나 이상의 컨테이너, 스토리지 리소스, 고유한 네트워크 IP 주소 및 컨테이너 실행 방식을 관리하는 옵션을 캡슐화
- 캡슐화(Encapsulation)
  - 서로 연관있는 속성과 기능들을 하나의 캡슐(capsule)로 만들어 데이터를 외부로부터 보호하는 것

03 Kubernetes 이해 (1)

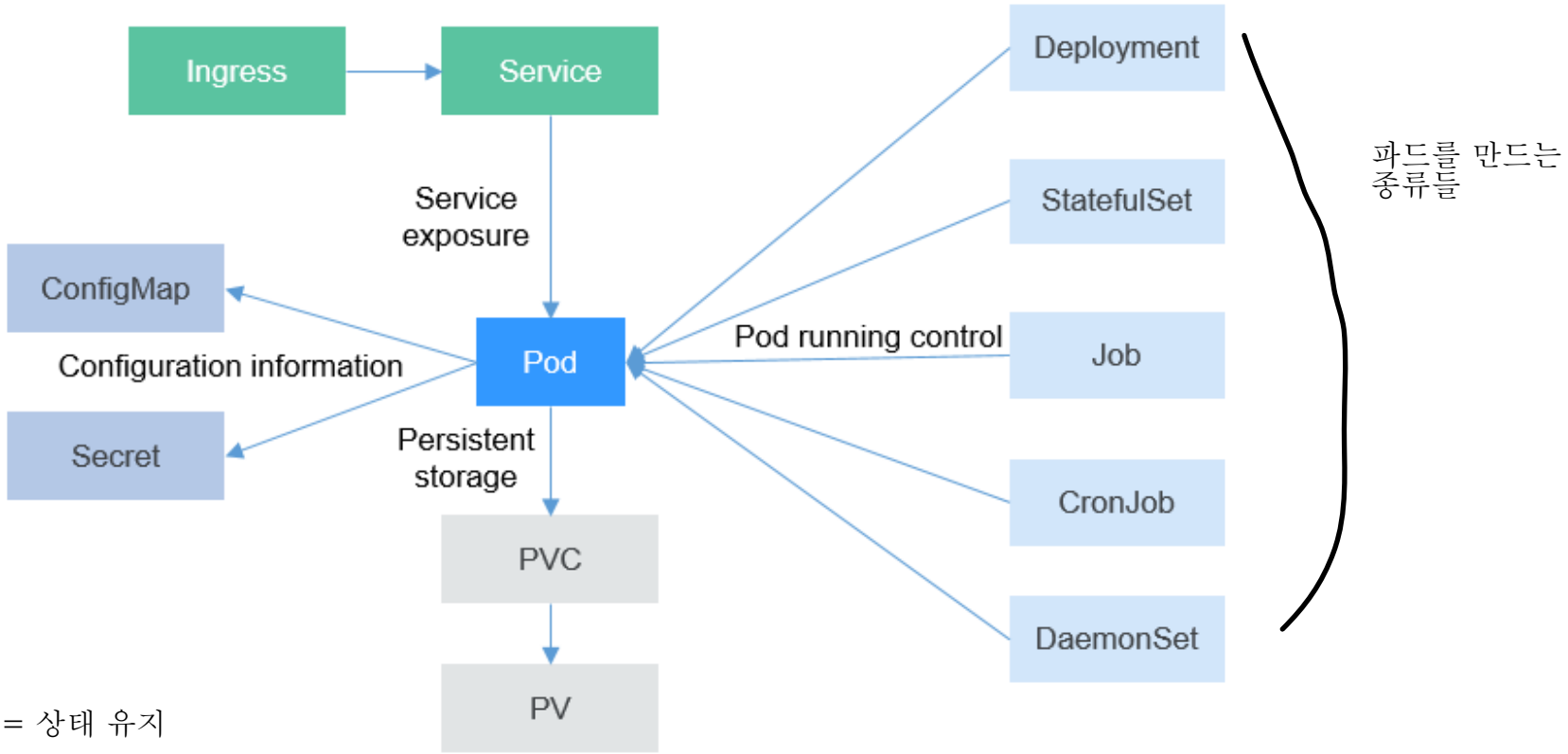
» Resource



- 디플로이먼트(Deployment)
  - 파드를 캡슐화하는 애플리케이션
  - 하나 이상의 파드를 포함할 수 있으며, 시스템은 요청에 따라 파드에 배포하는 것
  - Stateless한 어플리케이션에 적용
  - 디플로이먼트에 의해 생성된 파드는 고유한 식별자를 유지하지 않고, 파드가 재생성될 때 새로운 식별자를 생성

03 Kubernetes 이해 (1)

» Resource

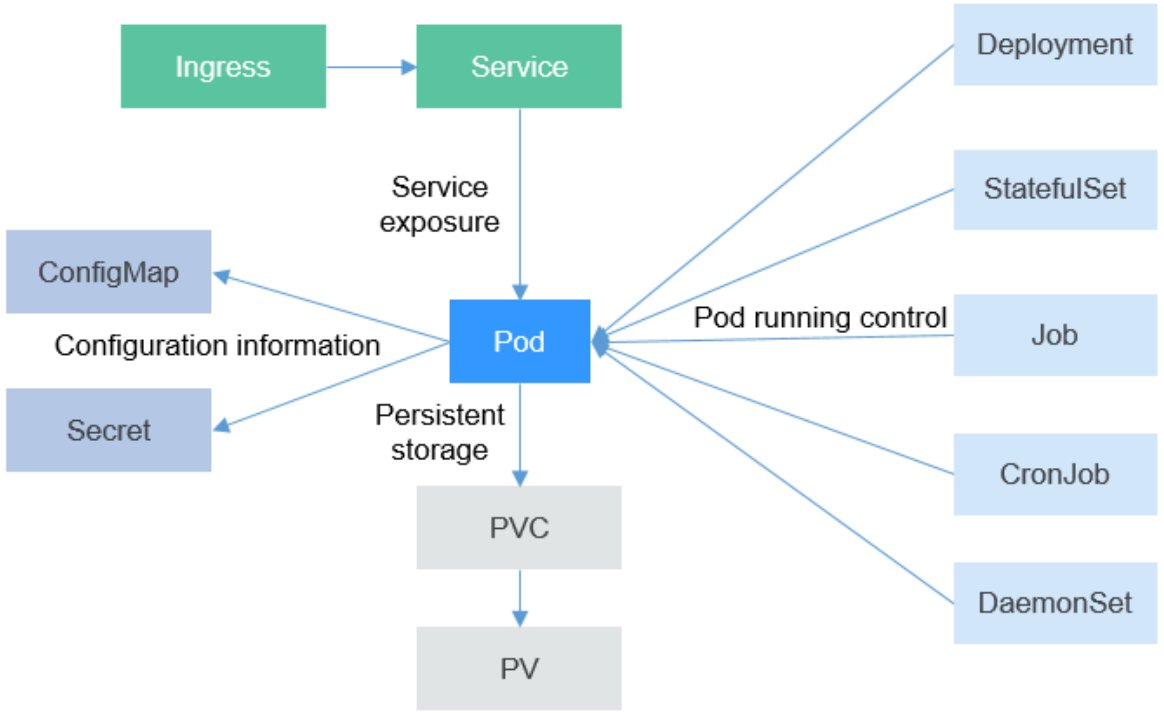


스테이트풀 == 상태 유지

- 스테이트풀셋(Statefulset)
  - 스테이트풀셋은 sateful(상태유지) 애플리케이션을 관리하는 데 사용 디플로이먼트와 마찬가지로 스테이트풀셋은 동일한 컨테이너 사양을 기반으로 파드 그룹을 관리
  - 스테이트풀셋은 주로 상태를 유지하는 애플리케이션(예: 데이터베이스)에 사용
  - 안정적이고 예측 가능한 식별자(예: pod-0, pod-1 등)를 가지고 식별자는 재배포되어도 변하지 않음

03 Kubernetes 이해 (1)

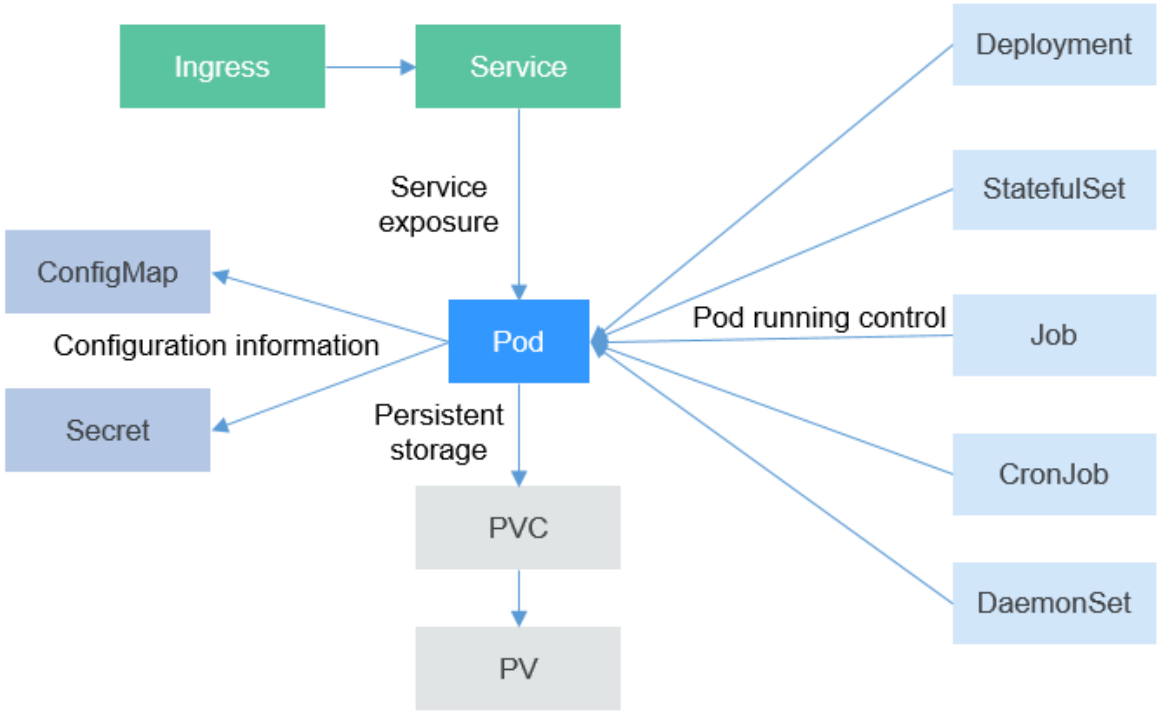
» Resource



- Job
  - 일회성 작업을 관리하기 위해 사용되는 리소스
  - 일반적으로 Job은 한 번 실행되고 특정 작업을 완료한 후 종료되는 파드를 생성하고 관리
  - Job은 데이터 처리, 배치 계산, 백업 작업 등 일시적이거나 일회성 처리가 필요한 작업을 실행하는 데 적합
  - 성공적으로 작업을 완료할 때까지 파드의 실행을 관리하여 파드가 실패할 경우, Job은 설정에 따라 파드를 재시작하고 작업이 성공적으로 완료될 때까지 계속 시도할 수 있음

03 Kubernetes 이해 (1)

» Resource

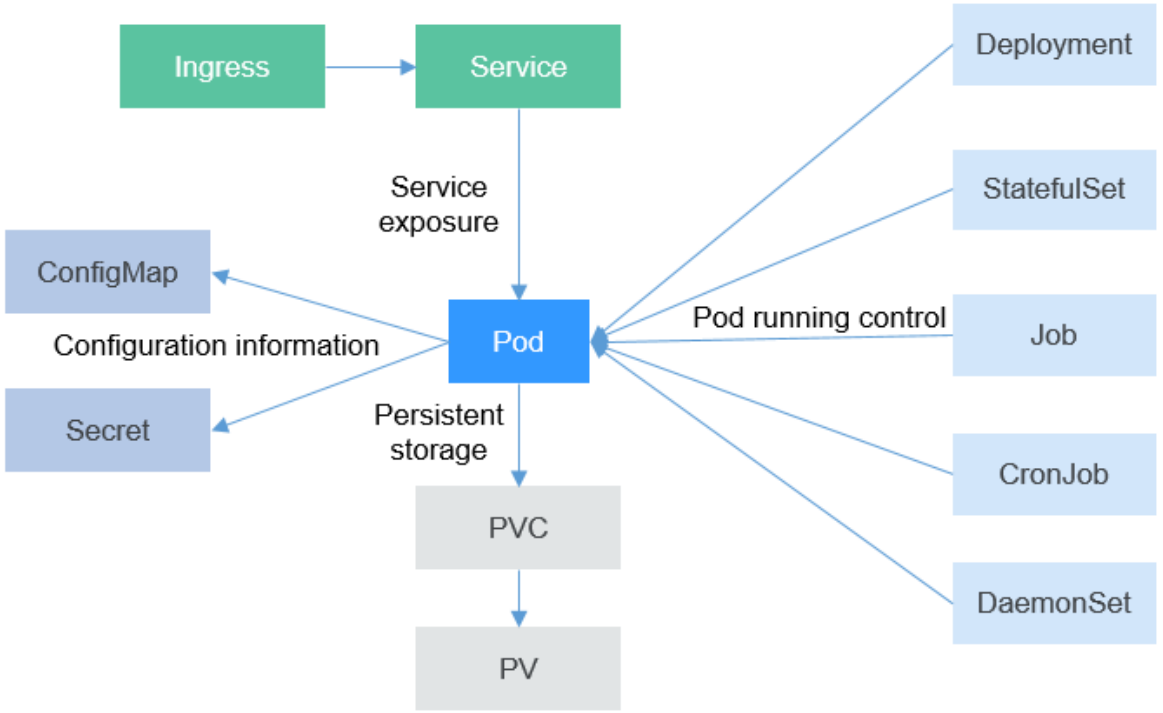



```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: example-cronjob
spec:
  schedule: "0 0 * * *"  # 매일 자정 실행
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: example
            image: example/image
            command: ["do", "something"]
          restartPolicy: OnFailure
```

- Cron Job
  - 주기적이고 반복적인 작업을 자동으로 실행하기 위한 리소스
  - 크론잡은 표준 크론 문법을 사용하여 작업 실행 간격을 설정할 수 있음
    – 예를 들어, 매일 자정, 매주 월요일 오전, 매월 첫째 날 등 사용자가 필요한 시간에 맞춰 설정

03 Kubernetes 이해 (1)

» Resource

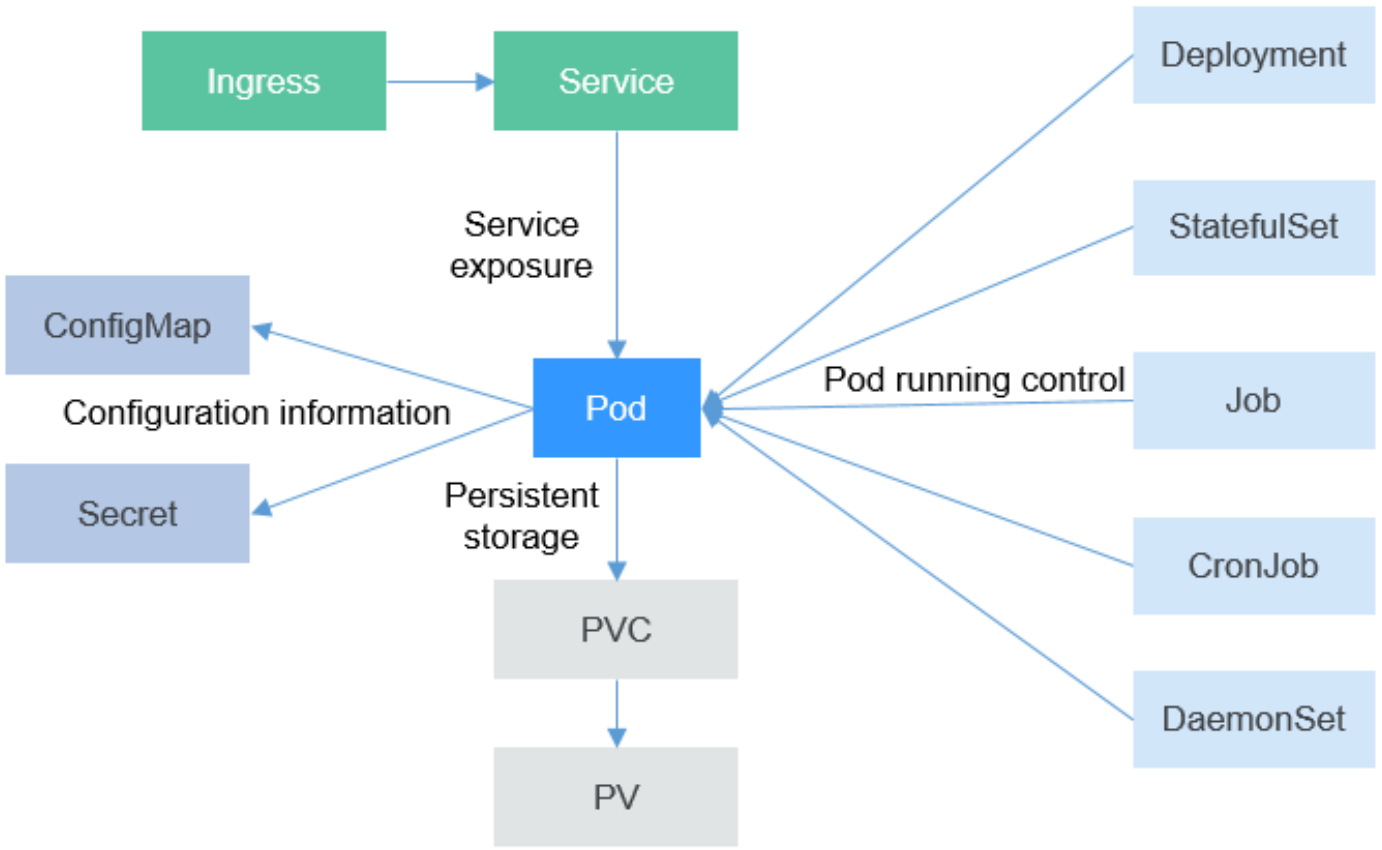



```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: example-daemonset
spec:
  selector:
    matchLabels:
      name: example
  template:
    metadata:
      labels:
        name: example
    spec:
      containers:
      - name: example-container
        image: example/image
        ports:
        - containerPort: 80
```

- DaemonSet
  - 클러스터 내의 모든 노드(또는 일부 특정 노드)에 파드를 자동으로 배포하는 데 사용되는 리소스
  - 각 노드에 대해 정확히 하나의 파드 복사본이 실행되도록 보장하는 것이 DaemonSet의 주 목적
  - (ex : 로그수집 : Fluentd, Logstash, 모니터링 : Prometheus Node, Exporter 등)

03 Kubernetes 이해 (1)

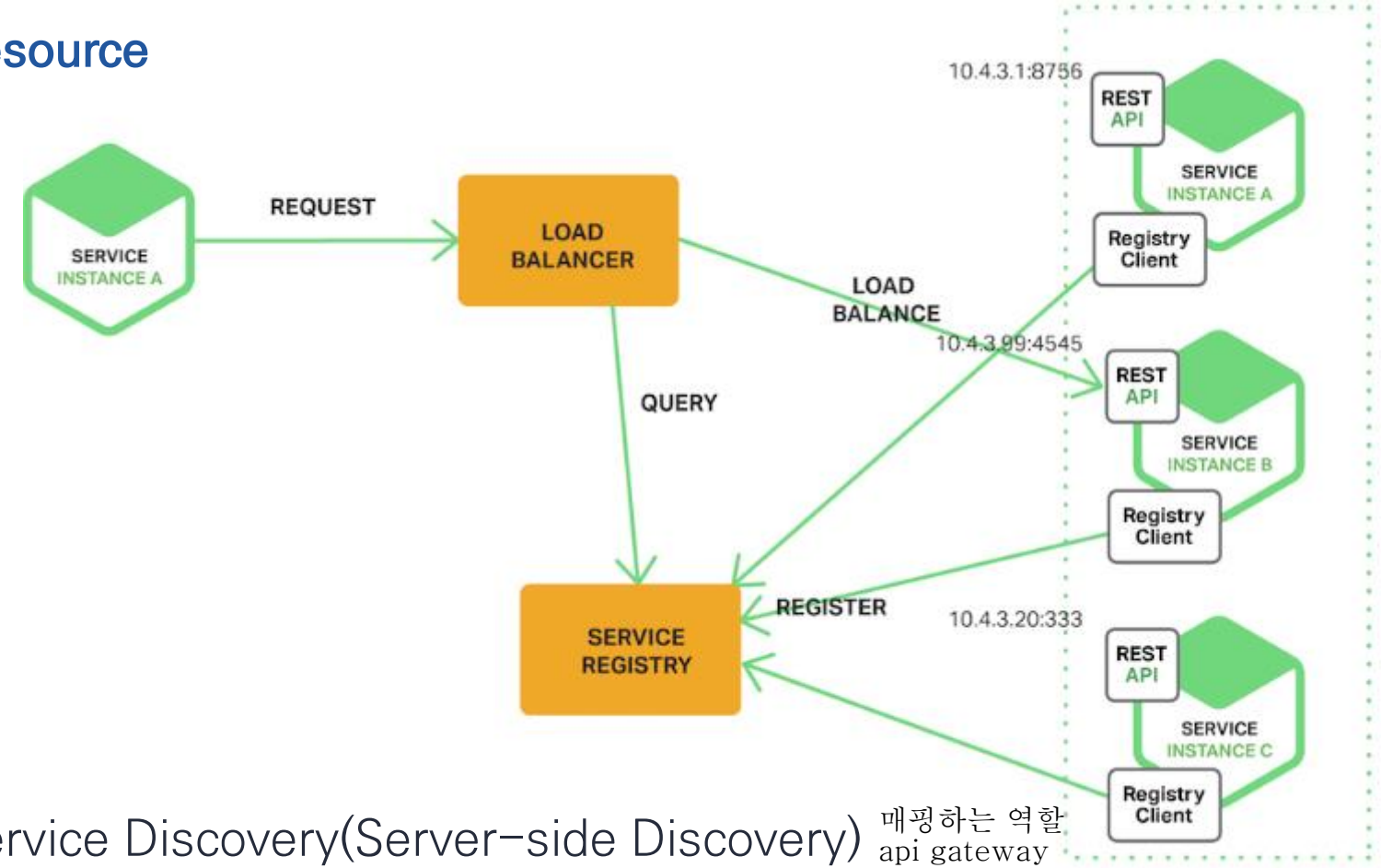
» Resource



- Service
  - 파드 그룹에 대한 안정적인 접근 주소를 제공
  - 서비스를 사용하면 특정 파드 그룹을 지속적으로 접근할 수 있는 고정된 방법을 마련
  - 로드 밸런싱과 서비스 디스커버리 역할도 수행하여, 여러 파드로 들어오는 요청을 균등하게 분배하고, 쿠버네티스 클러스터 내부에서 해당 서비스를 찾을 수 있게 해줌

03 Kubernetes 이해 (1)

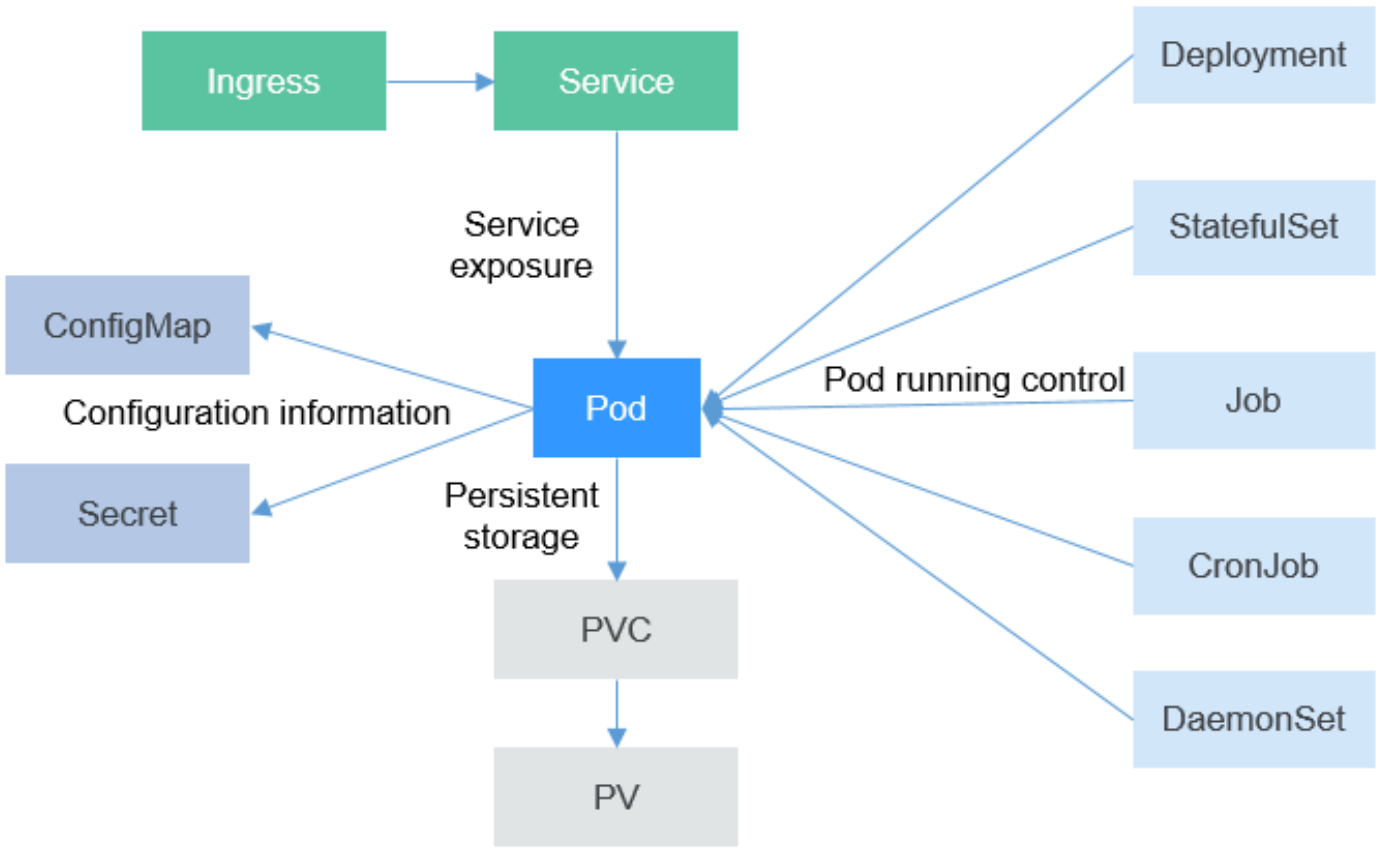
» Resource



- Service Discovery(Server-side Discovery) 매핑하는 역할 api gateway
  - 분산 네트워크 환경에서 서비스(실행 중인 인스턴스)들을 자동으로 검색하고 서로 연결할 수 있도록 하는 프로세스
  - 대규모 마이크로서비스 아키텍처에서 중요한 역할을 하며, 쿠버네티스와 같은 오케스트레이션 시스템에서 핵심 기능

03 Kubernetes 이해 (1)

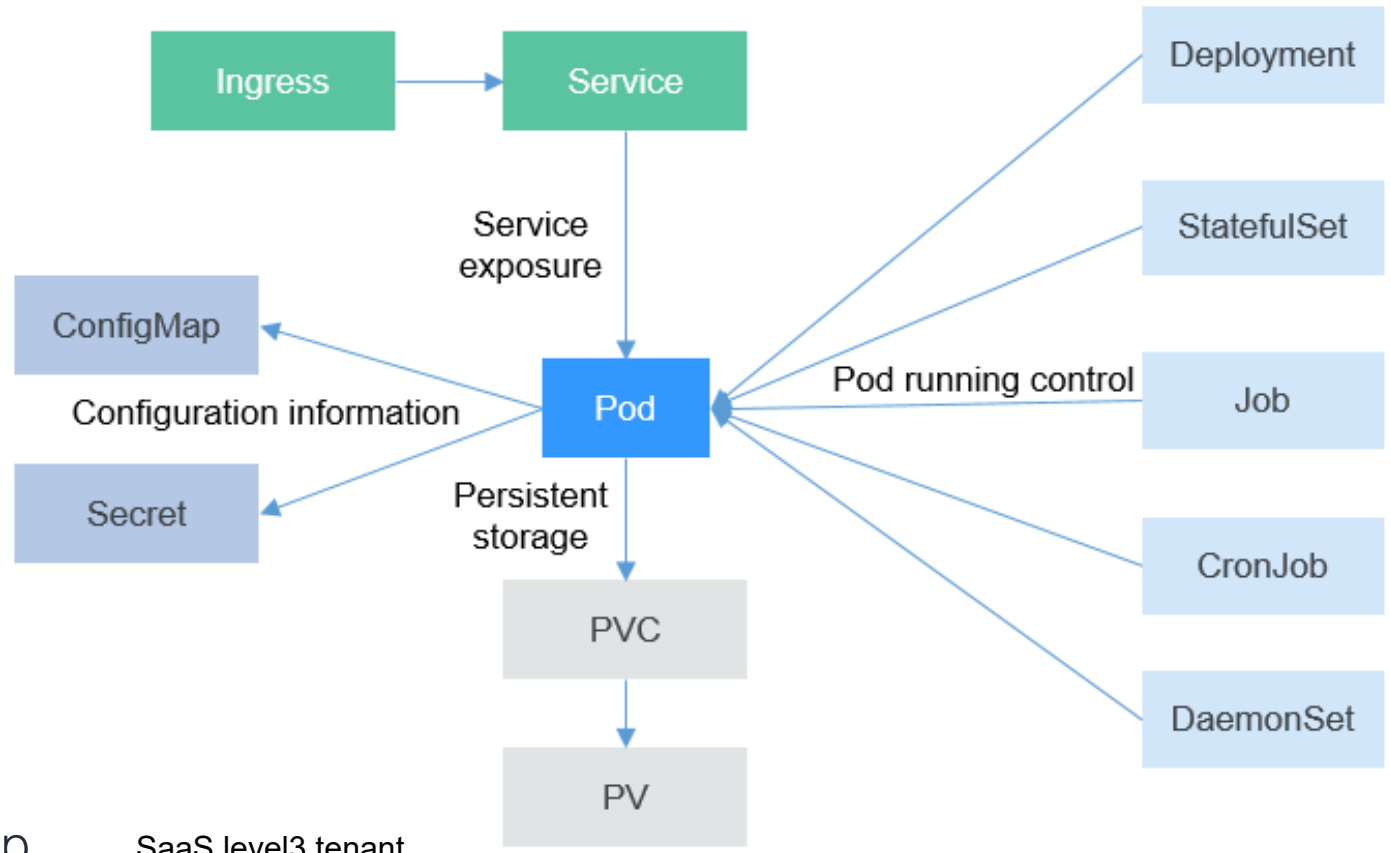
» Resource



- Ingress
  - 클러스터 외부에서 클러스터 내부로 들어오는 HTTP/HTTPS 트래픽을 관리
  - 도메인 이름을 통해 특정 웹 애플리케이션의 서비스로 트래픽을 라우팅

03 Kubernetes 이해 (1)

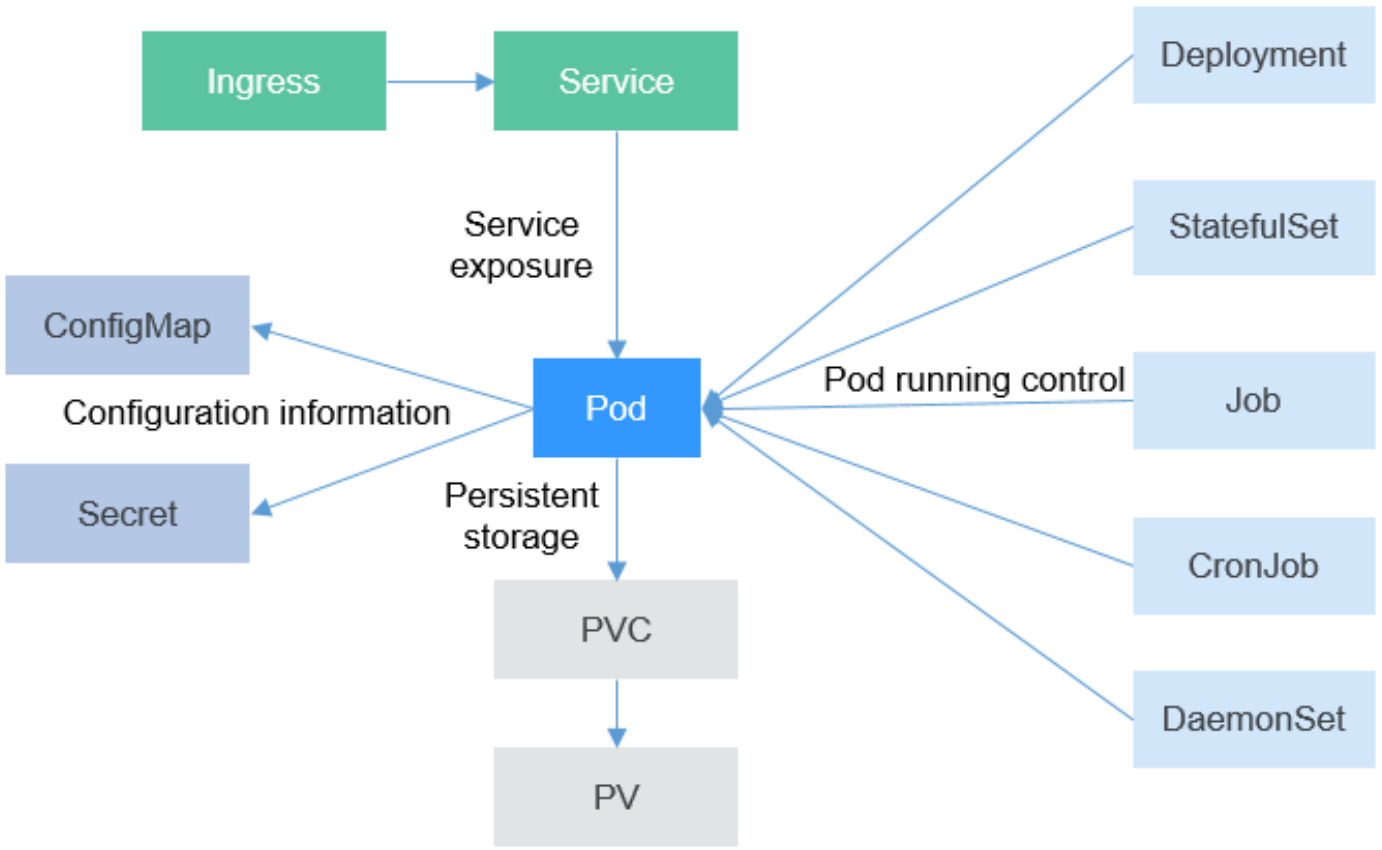
» Resource



- Configmap SaaS level3 tenant
  - 애플리케이션 코드와 설정 데이터를 분리할 수 있어, 애플리케이션의 설정을 유연하게 관리하고 환경 간 이동을 쉽게 할 수 있음

03 Kubernetes 이해 (1)

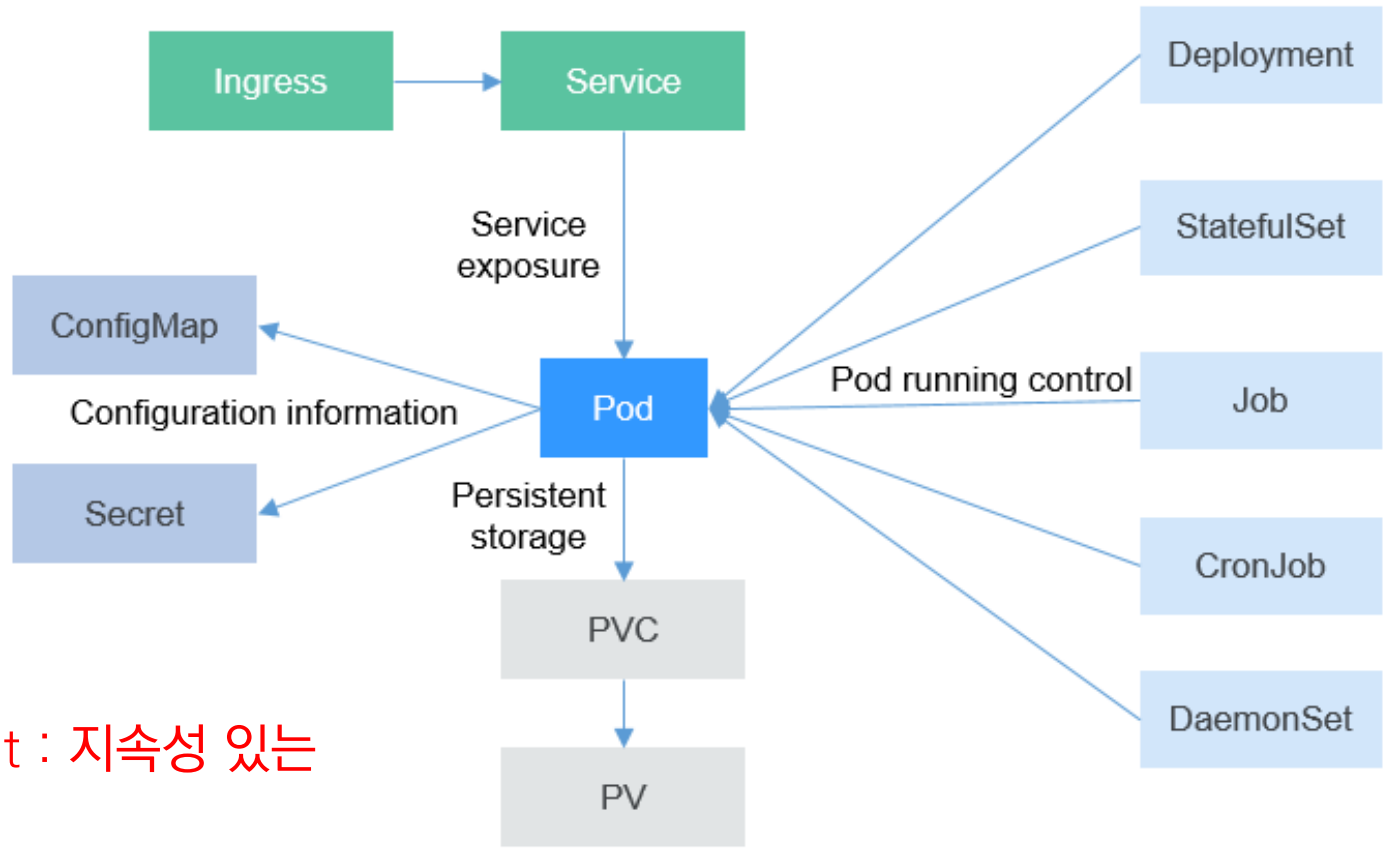
» Resource



- Secret
  - 민감한 정보를 안전하게 저장하고, 파드에서 사용할 수 있도록 제공하는 API 오브젝트 Secret을 사용하면 비밀번호, OAuth 토큰, ssh 키와 같은 민감한 데이터를 소스 코드나 애플리케이션 설정에서 분리하여 안전하게 관리할 수 있음

03 Kubernetes 이해 (1)

» Resource

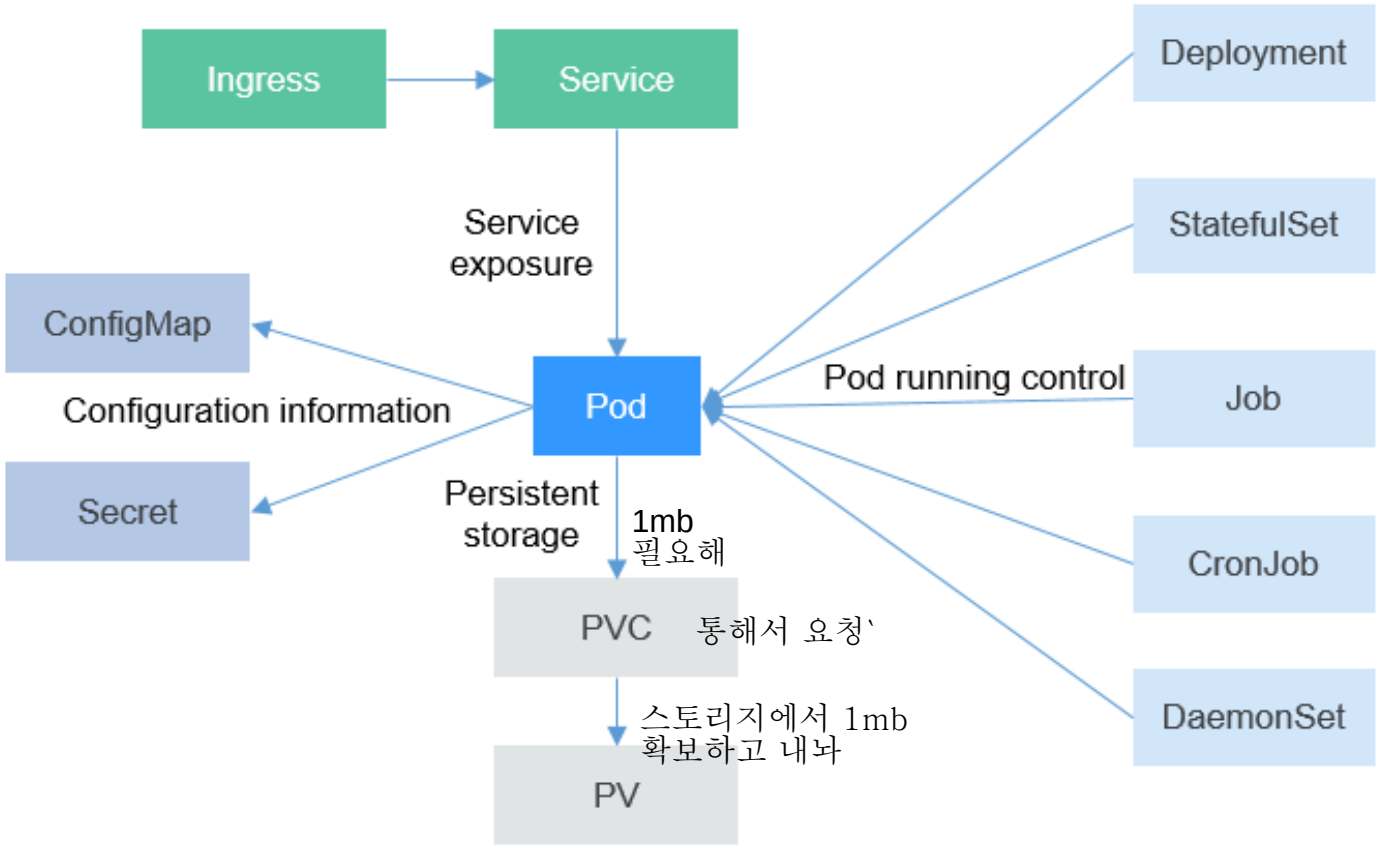


Persistent : 지속성 있는

- PVC, PV(Persistent Volume Claim)
  - 데이터의 지속적인 저장과 관리를 위한 리소스
  - 파드(Pod)의 일시적인 생명 주기와는 독립적으로 데이터를 지속적으로 저장할 수 있는 방법을 제공
  - 파드가 삭제되거나 재생성될 때도 데이터를 유지

03 Kubernetes 이해 (1)

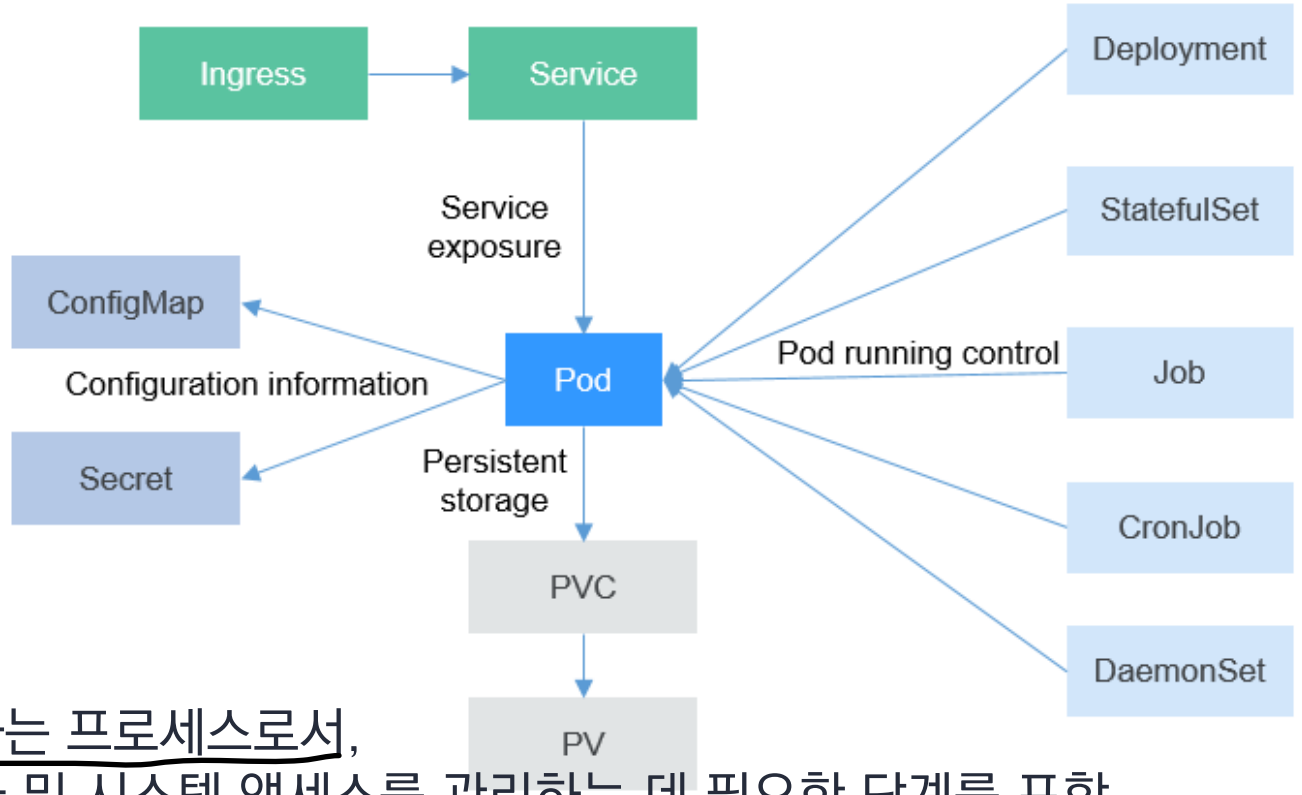
» Resource



- PVC, PV(Persistent Volume Claim)
  - PVC와 PV 사이의 관계는 "소비자(Consumer) – 제공자(Provider)" 관계와 유사
    PVC는 필요한 스토리지를 요청하고, PV는 그 요구를 충족하는 실제 리소스를 제공

03 Kubernetes 이해 (1)

» Resource

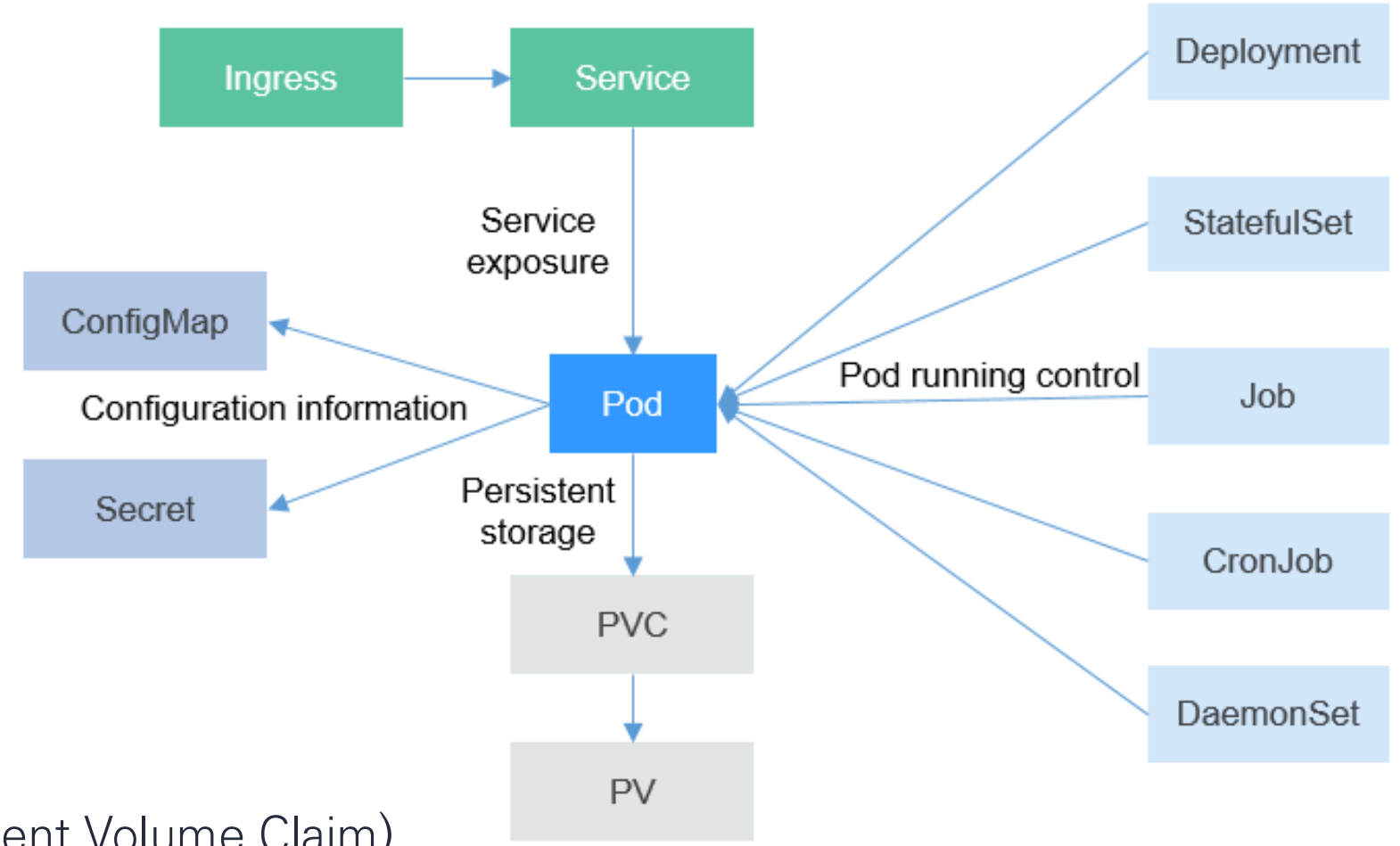


프로비저닝(Provisioning) :
IT 인프라를 생성하고 설정하는 프로세스로서,
다양한 리소스에 대한 사용자 및 시스템 액세스를 관리하는 데 필요한 단계를 포함
즉, 실제 서버의 자원을 할당하고 운영할 수 있게 해주는 방식

- PV(Persistent Volume)
  - Persistent Volume은 클러스터 내에서 프로비저닝된 스토리지의 일부
  - PV는 클러스터 리소스로서 관리자에 의해 생성되거나 동적 프로비저닝을 통해 자동으로 생성

03 Kubernetes 이해 (1)

» Resource

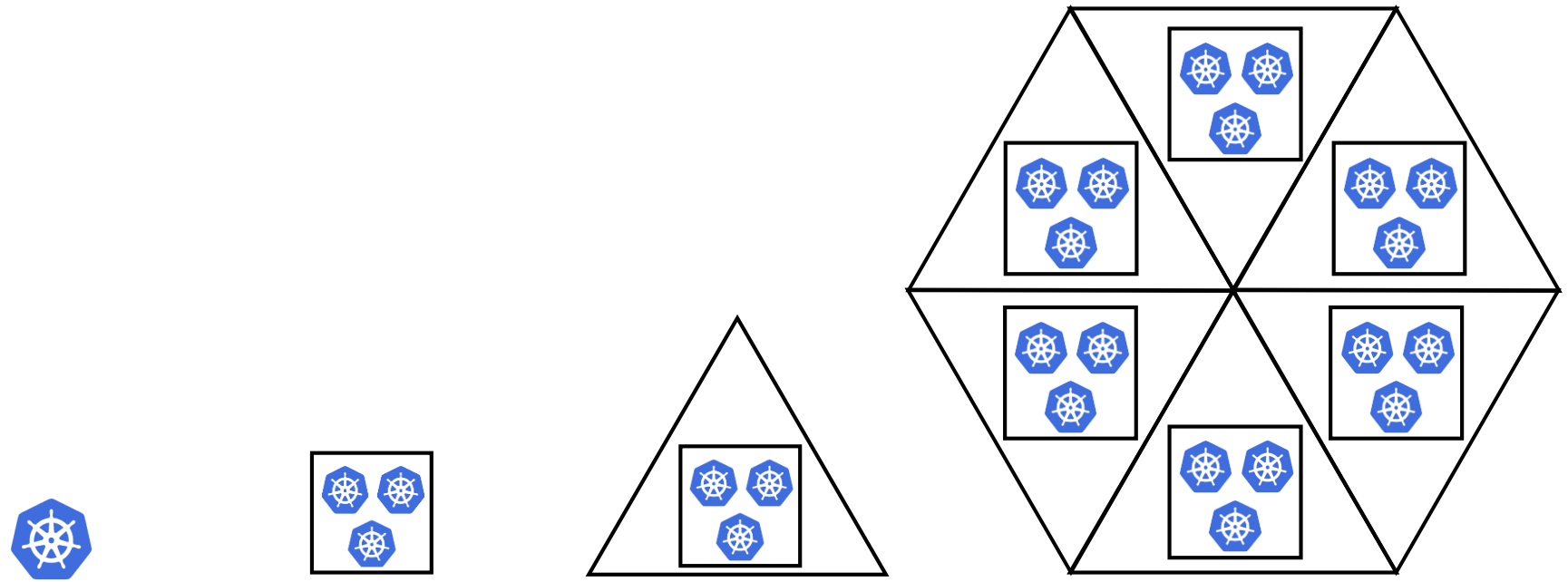


- PVC(Persistent Volume Claim)
  - 사용자가 스토리지를 요청하는 방법
  - VC는 특정 크기와 접근 모드를 명시하며, 스토리지 요구사항을 정의
  - 클러스터 관리자가 프로비저닝한 PV 중에서 PVC의 요구사항을 충족하는 것이 자동으로 바인딩

03 Kubernetes 이해 (1)

» Resource

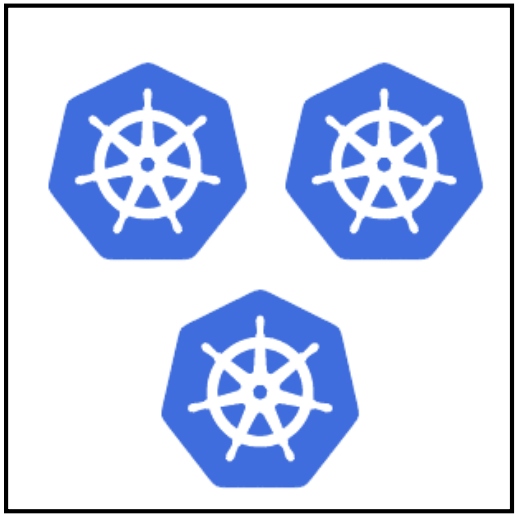
리소스 소개



03 Kubernetes 이해 (1)

» Pod 소개

파드 특징



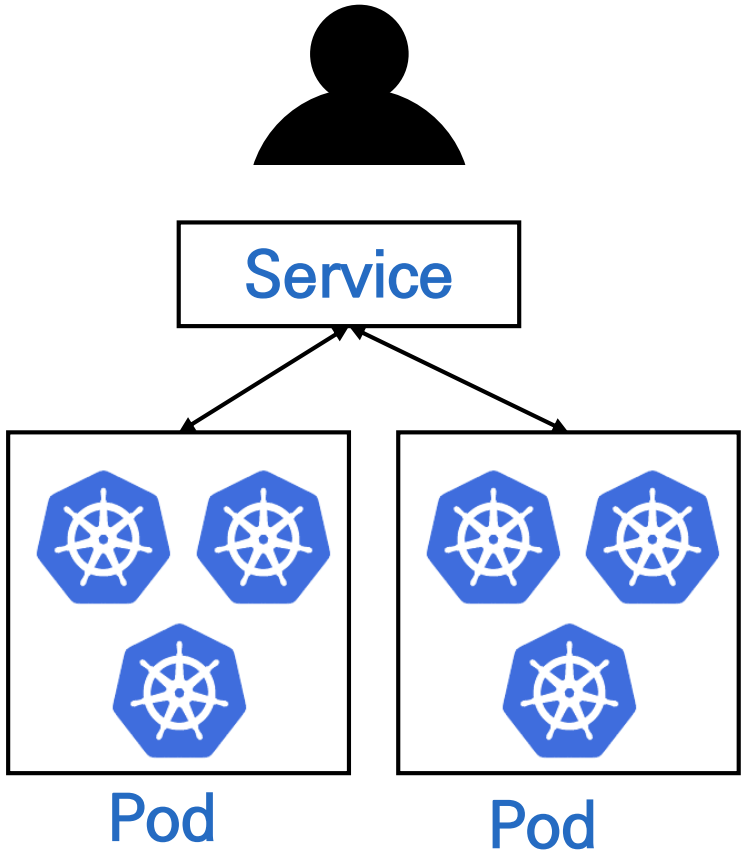
Pod

- 컨테이너가 직접 실행되는 장소
- 컨테이너를 돌리는 최소한의 단위
- 하나 이상의 컨테이너 그룹
- 컨테이너를 직접 관리하지 않고 파드 단위로 관리

03 Kubernetes 이해 (1)

» Service

서비스 특징



- 클러스터 외부에서 클러스터 내부 파드에 접근
- 고정된 IP주소 할당
- 파드 간의 로드 밸런싱 지원
- 하나 또는 여러 개의 포트 지원

03 Kubernetes 이해 (1)

» Ingress 소개

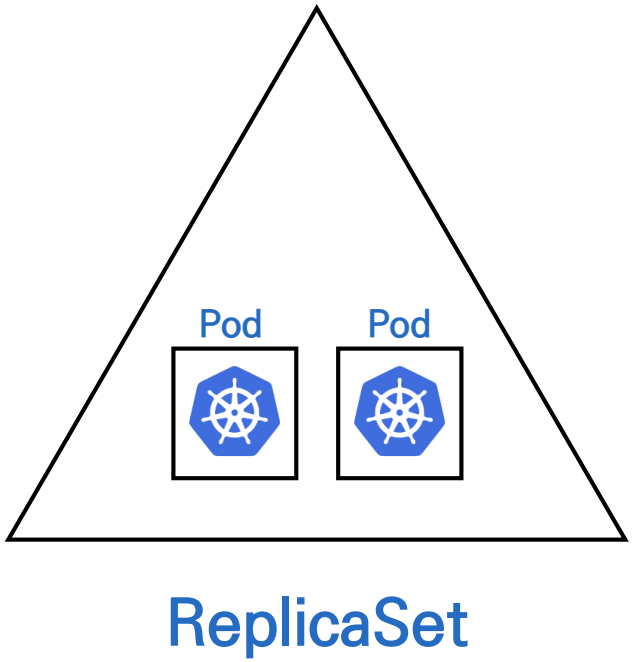
인그레스 특징

- 서버 외부로부터 서버 내부로 들어오는 네트워크 트래픽을 로드밸런싱
- L7 로드밸런싱 기능을 수행
- Service에 외부 URL을 제공
- 클러스터로 접근하는 URL 별로 다른 서비스에 트래픽을 분산
- TLS/SSL 인증서 처리
  웹으로부터의 안전한 연결을 보장
  secure socket layer
- 도메인 기반의 Virtual hosting을 지정

03 Kubernetes 이해 (1)

» ReplicaSet

레플리카셋 특징

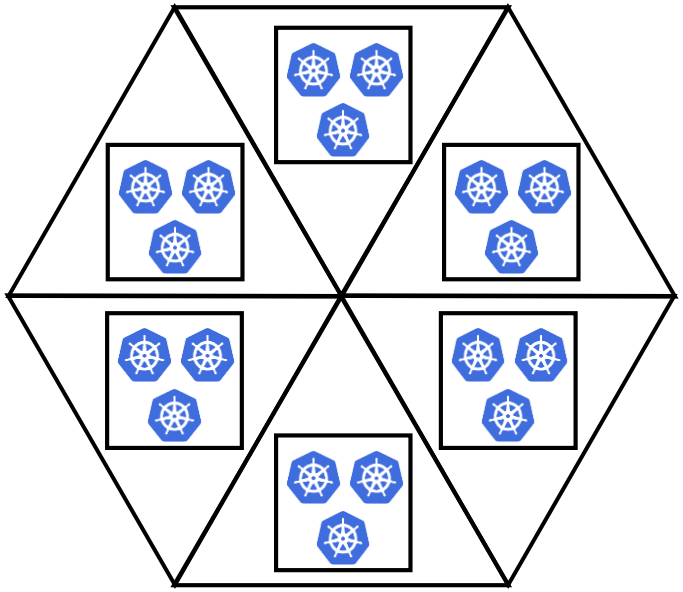


- 파드의 개수 유지 및 관리
- 실행되는 파드를 안정적으로 유지
- 파드 개수에 대한 가용성 보장

03 Kubernetes 이해 (1)

» Deployment

디플로이먼트 특징



Deployment

- 상태가 없는 앱을 배포할 때 사용되는 가장 기본적인 컨트롤러
- 레플리카셋을 관리
- 앱의 배포를 세밀하게 관리
- 파드 개수 유지
- 배포 시 롤링 업데이트

03 Kubernetes 이해 (1)

» StatefulSet 소개

스테이트풀셋 특징

- 동일한 컨테이너 스펙을 가진 파드들을 관리하며 각 파드들은 각각의 식별자를 가지기 때문에 독자성을 유지
- 파드들의 순서 및 고유성을 보장
- 식별자를 가지고 있어 재스케줄링 간에도 지속적인 유지
- 내부 파드마다 고유한 pvc를 갖도록 설정이 가능하여, 스케일 확장시 PV를 유지 가능

03 Kubernetes 이해 (1)

» Deployment vs StatefulSet

Deployment	StatefulSet
Stateless 방식의 Pod 배포 상태가 없는 애플리케이션 배포	Stateful한 방식의 Pod 관리 상태가 있는 애플리케이션 배포
Service를 통한 외부 노출	Headless Service를 통한 외부 노출
Service 요청시 랜덤한 Pod 선택	요청시 원하는 Pod 선택 가능 (단, Service 요청 불가능)
ReplicaSet을 가지고 있으며 rollback 가능	ReplicaSet이 없으며 rollback 불가능
모든 Pod가 1개의 PVC에 모두 연결	Pod 마다 각각의 고유한 PVC를 생성하여 고유한 PV를 가짐

03 Kubernetes 이해 (1)

» Job 소개

잡 특징

Job

- 여러 파드를 지정하여 지정된 파드를 성공적으로 실행하도록 하는 설정
- 배치작업, 임시작업 등 처음 한번 실행하고 종료되는 작업에 사용
- 파드의 성공적인 완료를 보장
- 단일잡, 다중잡, 병렬잡

03 Kubernetes 이해 (1)

» CronJob 소개

크론잡 특징

CronJob

- JOB을 실행하는데 스케줄러 역할
- 데이터 백업, 이메일 송신 등 정해진 시간에 맞춰 반복적으로 JOB을 실행해야 할 때 사용
- 리눅스의 crontab과 비슷

03 Kubernetes 이해 (1)

» DaemonSet 소개

데몬셋 특징

DaemonSet

- 클러스터 전체에 특정 파드를 실행할 때 사용하거나 특정 노드 또는 모든 노드에 항상 실행되어야 할 특정 파드들을 관리
- 전체 노드에서 Pod가 한 개씩 실행되도록 보장
- 로그 수집기, 모니터링 에이전트와 같이 모든 노드에 항상 실행시켜야 하는 특정 파드를 관리해야 할 때 사용

03 Kubernetes 이해 (1)

» Persistent Volume

퍼시스턴트 볼륨 특징

Persistent Volume

- 특정 파드와 상관없이 별도의 생명주기를 가지는 독립적인 볼륨
- 프로비저닝은 정적 방법, 동적 방법 두 가지 방법으로 생성
- PVC로부터 요청이 들어오면 요청에 맞는 PV 할당
- PV와 PVC 매핑은 1:1 관계로 바인딩

03 Kubernetes 이해 (1)

» Persistent Volume Claim

퍼시스턴트 볼륨 클레임 특징

Persistent Volume Claim

- 사용자가 PV에 하는 요청
- 파드와 PV를 연결
- 요청 시 스토리지 용량, 접근방법, 셀럭터 등 설정 가능
- PV와 PVC 매핑은 1:1 관계로 바인딩

03 Kubernetes 이해 (1)

» StorageClass 소개

스토리지 클래스 특징

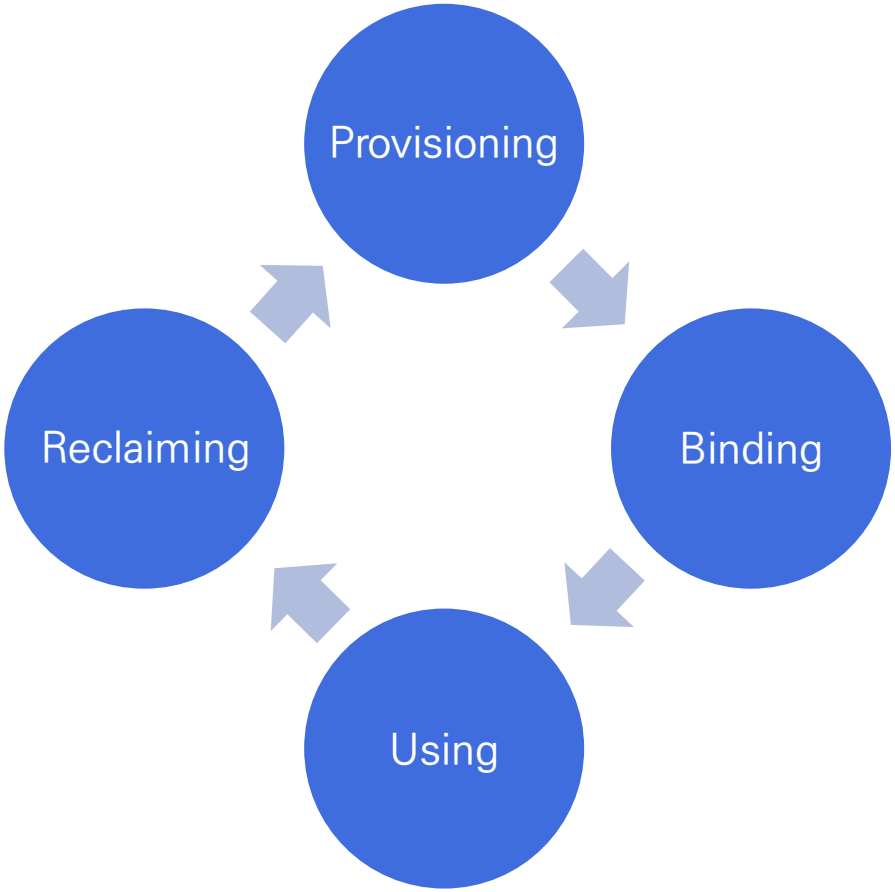
StorageClass

- PV로 확보한 스토리지 종류를 정의 하는 리소스
- PV의 동적 프로비저닝일 때 사용되며 동적 프로비저닝을 활성화 하려면 하나 이상의 스토리지 클래스가 미리 생성 되어 있어야함
- 스토리지 클래스를 미리 정의하면 그에 맞는 PV를 바로 생성할 수 있음

03 Kubernetes 이해 (1)

» PV, PVC 생명주기

생명주기

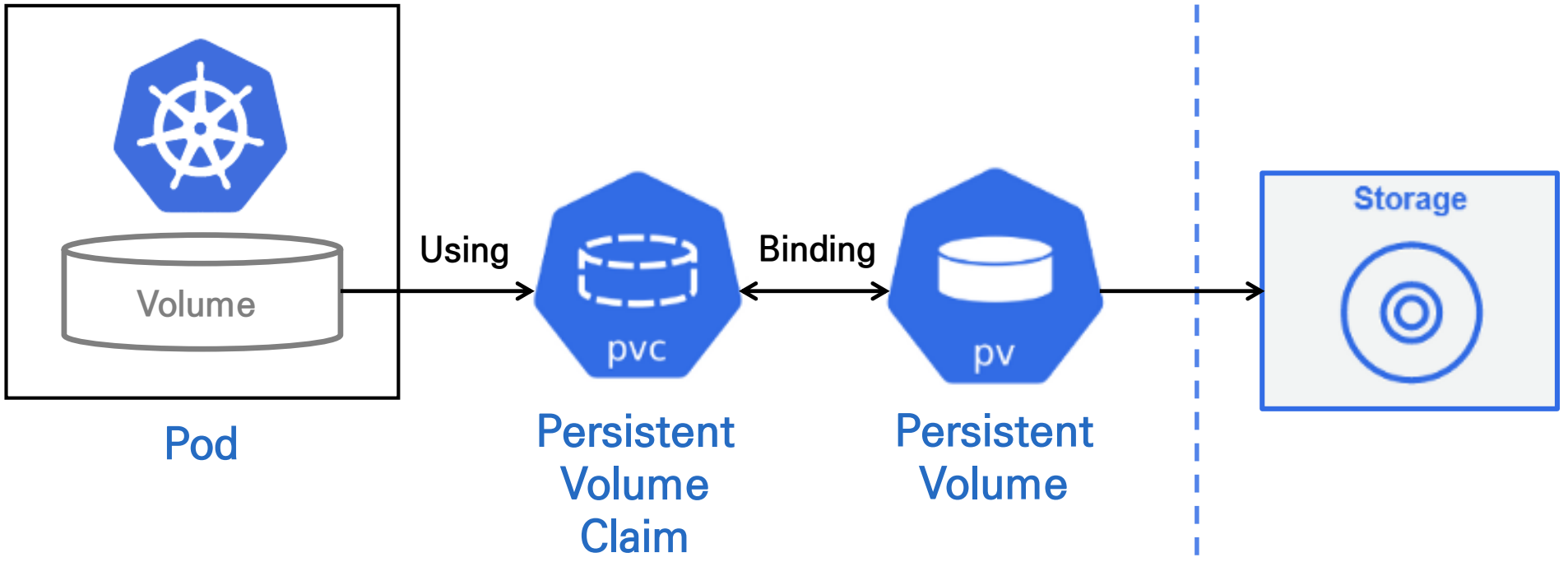


- **Provisioning**
  정적 방법과 동적 방법 두 가지가 있으며 PV를 생성하는 단계
- **Binding**
  PVC가 원하는 접근방법, 스토리지 용량 등을 요청하면 그에 맞는 PV가 연결되는 단계
- **Using**
  파드가 유지되는 동안 파드에서 설정된 PVC가 계속해서 사용되고 있는 단계
- **Reclaiming**
  사용이 끝난 PVC는 삭제되고 사용 중이던 PV는 초기화 되는 과정

03 Kubernetes 이해 (1)

» Volume 구조(정적 프로비저닝)

미리 만들었기에 그냥 쓰면 됨. 빠름.
유동적으로 변경하기가 힘듬. 유연하지 못함.
실시간 늘리거나 재구성 하기 힘듬.

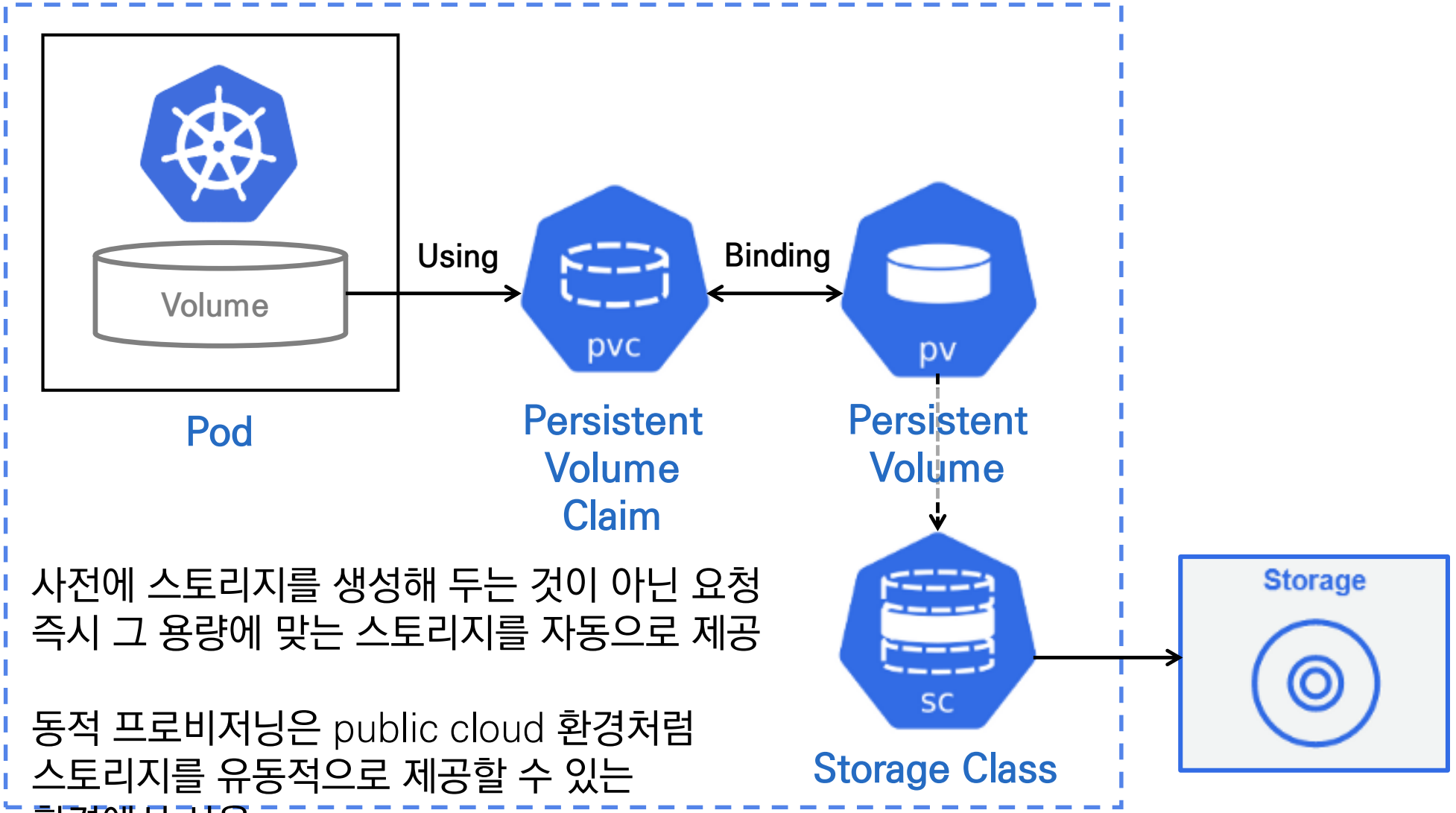


정적 프로비저닝은 클러스터 관리자가 사전에 스토리지를 준비하여 여러 pv를 사전에 제공해야하는 번거로움이 있고 또한 사용하지 않는 스토리지를 제공해야하는 자원 낭비의 단점이 존재

03 Kubernetes 이해 (1)

» Volume 구조(동적 프로비저닝)

CSP에서 많이 제공함



사전에 스토리지를 생성해 두는 것이 아닌 요청 즉시 그 용량에 맞는 스토리지를 자동으로 제공

동적 프로비저닝은 public cloud 환경처럼 스토리지를 유동적으로 제공할 수 있는 환경에서 사용

03 Kubernetes 이해 (1)

» ConfigMap 소개

컨피그맵 특징

ConfigMap

- 컨테이너와 분리해서 환경설정 제공
- 키-값 형태의 설정 정보를 저장하는 일종의 저장소 역할
- 간단한 문자부터 큰 설정 파일까지 값으로 가질 수 있음
- 동일한 파드라도 여러 개의 컨피그맵을 사용할 수 있음

03 Kubernetes 이해 (1)

» Secret 소개

시크릿 특징

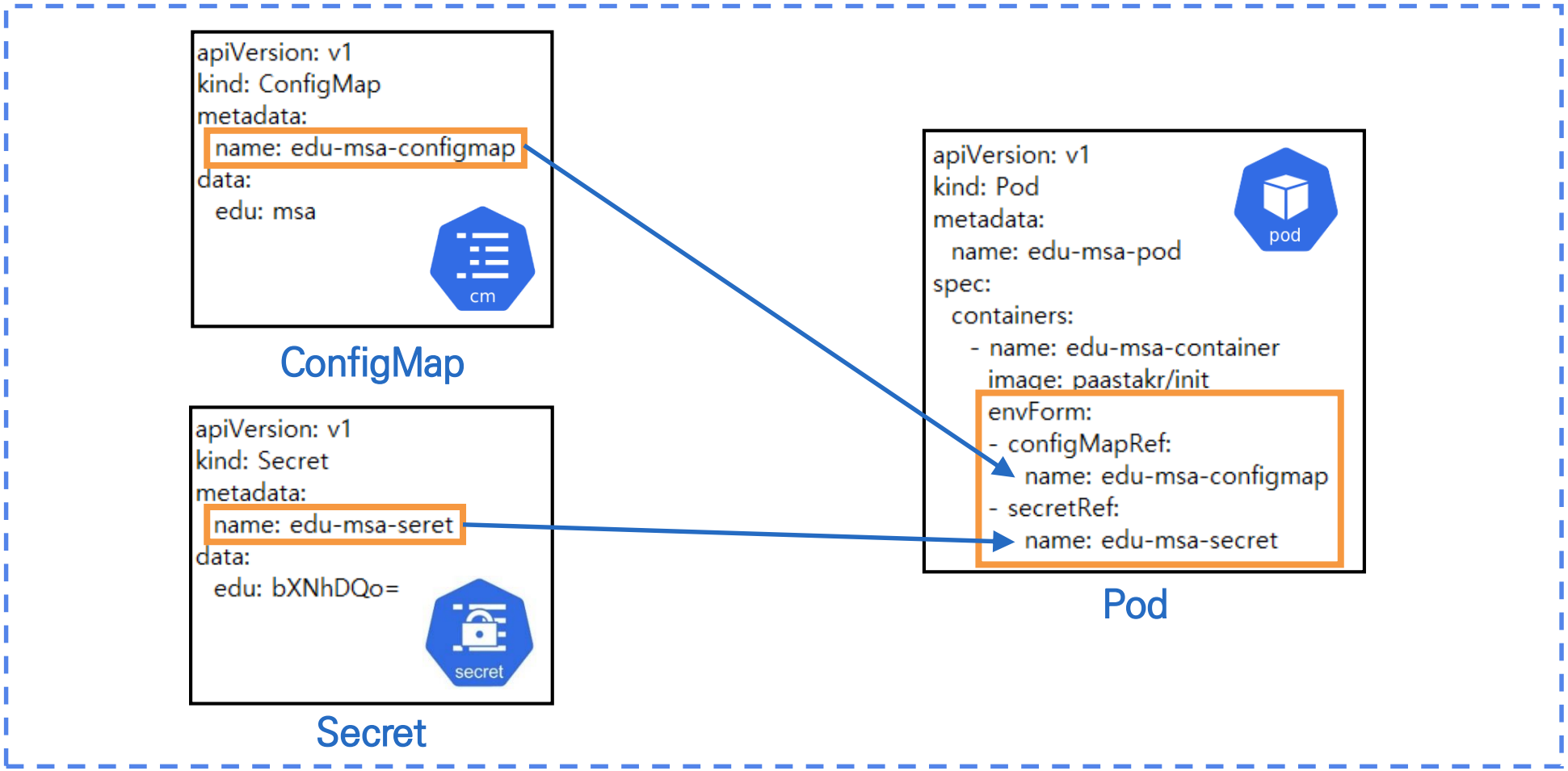
Secret

- 보안유지가 필요한 자격증명 및 개인 암호화 키 같은 정보를 저장 및 관리하는 역할
- Base64 인코딩으로 생성
- 키-값 형태의 설정 정보를 저장하는 일종의 저장소 역할
- 메모리에 저장됨
- 개별 시크릿의 최대 크기는 1MB까지 정의
- 모든 파드에는 자동으로 연결된 시크릿 볼륨이 존재

03 Kubernetes 이해 (1)

» ConfigMap, Secret 사용 방법 소개

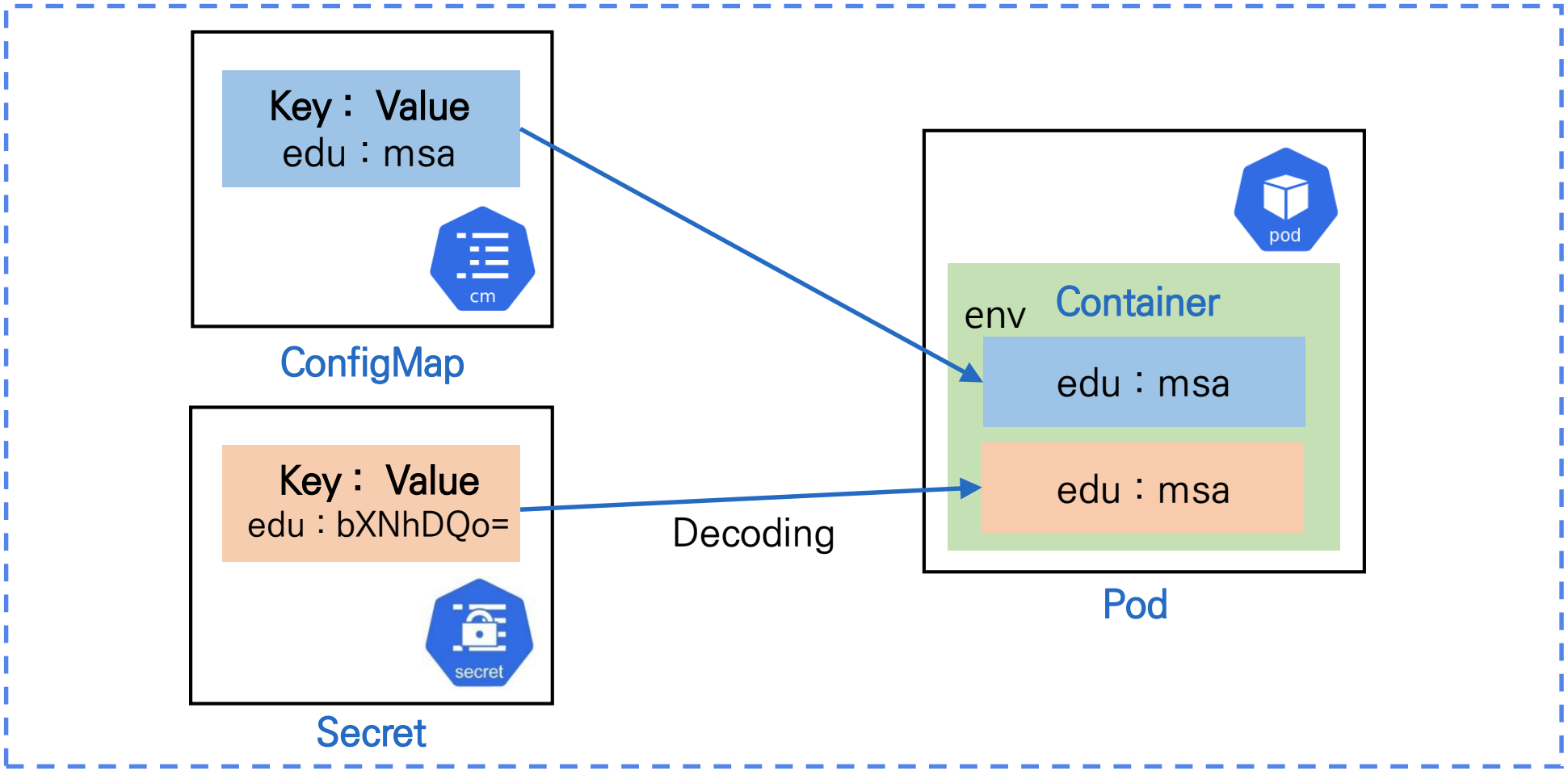
- 파드 생성시 컨피그맵과 시크릿 정보를 입력



03 Kubernetes 이해 (1)

» ConfigMap, Secret 사용 방법 소개

- 파드의 컨테이너 환경 변수에 컨피그맵의 데이터와 디코딩 된 시크릿의 데이터가 들어 감



04 Kubernetes 이해 (2)

» Kubernetes Resource 관리

쿠버네티스 리소스 관리

- 쿠버네티스 리소스는 YAML 또는 JSON으로 작성된 구성 파일과 함께 kubectl 커맨드 라인 툴을 이용하여 관리

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: paasta-pod
 labels:
   app: paasta-label
spec:
 containers:
 - name: paasta-container
   image: paasta/msa
```

YAML

```json
{
  "apiVersion": "v1",
  "kind": "Pod",
  "metadata": {
    "name": "paasta-pod",
    "labels": {
      "app": "paasta-label"
    }
  },
  "spec": {
    "containers": [
      {
        "name": "paasta-container",
        "image": "paasta/msa"
      }
    ]
  }
}
```

JSON

```
I:₩>kubectl create -f paasta.yaml
pod/paasta-pod created

I:₩>kubectl get pods
NAME         READY   STATUS         RESTARTS   AGE
paasta-pod   1/1     Running        0          7s

I:₩>kubectl delete -f paasta.yaml
pod "paasta-pod" deleted
```

kubectl 커맨드

04 Kubernetes 이해 (2)

» yaml 파일 작성 방법

디플로이먼트 예시 (1/7)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: edu-msa-board
  labels:
    app: board-msa
spec:
  replicas: 1
  selector:
    matchLabels:
      app: board-msa
  template:
    metadata:
      labels:
        app: board-msa
```

- apiVersion
  특정 리소스를 생성하기 위해 사용할 쿠버네티스 API 버전을 명세

- kind
  어떤 종류의 리소스를 생성하고자 하는지 생성할 리소스 타입을 명세

04 Kubernetes 이해 (2)

» yaml 파일 작성 방법

디플로이먼트 예시 (2/7)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: edu-msa-board
  labels:
    app: board-msa
spec:
  replicas: 1
  selector:
    matchLabels:
      app: board-msa
  template:
    metadata:
      labels:
        app: board-msa
```

- metadata
  리소스에 이름을 부여하고 리소스를 유일하게 구분 짓는 데이터
- metadata.name
  리소스의 이름을 명세
- metadata.labels
  키-값 쌍으로 구성되어 특정 쿠버네티스 리소스만 나열 또는 검색할 때 사용
- metadata.labels.app
  리소스의 레이블을 설정

04 Kubernetes 이해 (2)

» yaml 파일 작성 방법

디플로이먼트 예시 (3/7)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: edu-msa-board
  labels:
    app: board-msa
spec:
  replicas: 1
  selector:
    matchLabels:
      app: board-msa
  template:
    metadata:
      labels:
        app: board-msa
```

- spec
  생성하고자 하는 리소스에 대한 내용을 구체적으로 정의
- spec.replicas
  띄우고자 하는 파드의 개수를 설정
- spec.selector
  어떤 파드를 컨트롤하고 감시해야 하는지 정의
- spec.selector.matchLabels
  metadata.labels와 동일한 설정으로 맵핑하여 동일한 레이블을 가진 파드의 컨테이너를 카운팅하여 현재 상태를 측정

04 Kubernetes 이해 (2)

» yaml 파일 작성 방법

디플로이먼트 예시 (4/7)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: edu-msa-board
  labels:
    app: board-msa
spec:
  replicas: 1
  selector:
    matchLabels:
      app: board-msa
  template:
    metadata:
      labels:
        app: board-msa
```

- spec.template
  어떤 파드를 실행할지에 대한 정보를 설정

- spec.template.metadata
  metadata.labels와 동일한 설정으로 맵핑하여 동일한 레이블을 가진 파드를 실행

04 Kubernetes 이해 (2)

» yaml 파일 작성 방법

디플로이먼트 예시 (5/7)

```
spec:
  containers:
  - name: board-msa
    image: paastakr/edu-msa-board:latest
    imagePullPolicy: Always
    ports:
    - containerPort: 28082
  imagePullSecrets:
    - name: edu-msa-secret
  nodeSelector:
    kubernetes.io/hostname: paas-ta-worker-1
```

- spec.template.spec
  실행시킬 컨테이너에 대한 설정
- spec.template.spec.containers[]
  실행시킬 컨테이너의 이름, 이미지, 포트 번호 등을 설정
- spec.template.spec.containers[].name
  컨테이너 이름을 지정

04 Kubernetes 이해 (2)

» yaml 파일 작성 방법

디플로이먼트 예시 (6/7)

```
spec:
  containers:
  - name: board-msa
    image: paastakr/edu-msa-board:latest
    imagePullPolicy: Always
    ports:
    - containerPort: 28082
  imagePullSecrets:
    - name: edu-msa-secret
  nodeSelector:
    kubernetes.io/hostname: paas-ta-worker-1
```

- spec.template.spec.containers[].image
  컨테이너로 배포할 이미지를 지정
- spec.template.spec.containers[].imagePullPolicy
  이미지 다운로드 정책을 지정(“Always”는 이미지 여부에 상관없이 다운로드)
- spec.template.spec.containers[].ports[]
  컨테이너 포트번호를 지정

04 Kubernetes 이해 (2)

» yaml 파일 작성 방법

디플로이먼트 예시 (7/7)

```
spec:
  containers:
  - name: board-msa
    image: paastakr/edu-msa-board:latest
    imagePullPolicy: Always
    ports:
    - containerPort: 28082
  imagePullSecrets:
    - name: edu-msa-secret
  nodeSelector:
    kubernetes.io/hostname: paas-ta-worker-1
```

- spec.template.spec.imagePullSecret
  이미지 저장소에 접근하기 위한 인증정보

- spec.template.spec.nodeSelector
  파드를 배포할 특정 노드를 지정

04 Kubernetes 이해 (2)

» yaml 파일 작성 방법

서비스 예시 (1/2)

```
apiVersion: v1
kind: Service
metadata:
  name: edu-msa-board
  labels:
    app: board-msa
spec:
  ports:
  - nodePort: ${EDU_MSA_BOARD}
    port: 28082
    protocol: TCP
    targetPort: 28082
  selector:
    app: board-msa
  type: NodePort
```

- spec.ports.[].nodePort
  외부에서 접근 가능한 포트 번호를 설정
  (외부 포트)
- spec.ports.[].port
  컨테이너 포트 번호를 설정(내부 포트)
- spec.ports[].protocol
  프로토콜 방식을 설정
- spec.ports[].targetport
  접근하고자 하는 컨테이너 포트 번호를 설정

04 Kubernetes 이해 (2)

» yaml 파일 작성 방법

서비스 예시 (2/2)

```
apiVersion: v1
kind: Service
metadata:
  name: edu-msa-board
  labels:
    app: board-msa
spec:
  ports:
  - nodePort: ${EDU_MSA_BOARD}
    port: 28082
    protocol: TCP
    targetPort: 28082
  selector:
    app: board-msa
  type: NodePort
```

- spec.type
  외부에 노출하고자 하는 방식을 설정
  (“NodePort”는 포트 번호를 통해 외부에서 접근할 수 있는 방법)

04 Kubernetes 이해 (2)

» yaml 파일 작성 방법

yaml 파일 통합

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: edu-msa-board
  labels:
    app: board-msa
spec:
  replicas: 1
  selector:
    matchLabels:
      app: board-msa
  template:
    metadata:
      labels:
        app: board-msa
    spec:
      containers:
      - name: board-msa
        image: paastakr/edu-msa-board:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 28082
---
apiVersion: v1
kind: Service
metadata:
  name: edu-msa-board
  labels:
    app: board-msa
spec:
  ports:
  - nodePort: 30201
    port: 28082
    protocol: TCP
    targetPort: 28082
  selector:
    app: board-msa
  type: NodePort
```

Deployment

Service

- 서로 관련된 여러 yaml 파일을 하나의 파일로 작성하기 위해서는 중간에 “---” 로 구분 지어 작성

03 Container Platform 도구

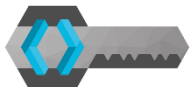
01. Terraform & Ansible
02. Vault
03. Ceph

01 Terraform & Ansible

» Container Platform 오픈소스 목록







오픈소스 명	라이선스
kubernetes	Apache 2.0
kubeEdge	Apache 2.0
harbor	Apache 2.0
mariadb	GPL v2
keycloak	Apache 2.0
terraform	MLP 2.0
ingress-nginx-controller	Apache 2.0
istio	Apache 2.0
kubeflow	Apache 2.0
vault	MLP 2.0
rook	Apache 2.0






출처: 개방형 클라우드 플랫폼 대표포털

01 Terraform & Ansible

» IaC(Infrastructure as Code) 개념

스크립트를 통해 인프라 및 구성 관리를 자동화 하는 방법론

- 수동 프로세스가 아닌 코드를 통해 인프라를 관리하고 프로비저닝
- 정보 관리를 위해 별도의 인프라 세트가 필요하지 않음(Masterless)

IaC 툴의 장점

- 물리적 환경에 대한 세부정보를 추상화 하여 중요한 코드에 집중 가능
- 비용절감
- 배포 속도 향상
- 오류 감소
- 인프라 일관성 향상
- 구성 드리프트 제거

01 Terraform & Ansible

» Terraform이란?

HashiCorp에서 운영 중인 **인프라를 손쉽게 구축하고 안전하게 변경**하고,
효율적으로 인프라의 형상을 관리할 수 있는 오픈 소스 도구

- 플랫폼에 구애 받지 않음
  → 다양한 클라우드 서비스들을 프로바이더 방식으로 제공(AWS, GCP, Azure 등)

- 변경 불가능 인프라
  → 환경에 대한 변경 사항이 적용될 때마다 현재 구성이 변경을 반영할 수 있는 새로운 구성으로 대체되고 인프라가 다시 프로비저닝됨

출처 : https://www.44bits.io/ko/keyword/terraform

01 Terraform & Ansible

» Terraform 특징

Infrastructure as Code
인프라를 코드로 정의하여 생산성과 투명성을 높일 수 있고, 정의한 코드를 쉽게 공유할 수 있어 효율적으로 협업할 수 있음

Execution Plan
변경 계획과 변경 적용을 분리하여 변경 내용을 적용할 때 발생할 수 있는 실수를 줄일 수 있음

Resource Graph
사소한 변경이 인프라 전체에 어떤 영향을 미칠지 미리 확인할 수 있으므로
종속성 그래프를 작성하여 이 그래프를 바탕으로 계획을 세우고, 이 계획을 적용했을 때 변경되는 인프라 상태를 확인할 수 있음

Change Automation
여러 장소에 같은 구성의 인프라를 구축하고 변경할 수 있도록 자동화할 수 있어 인프라를 구축하는 데 드는 시간을 절약할 수 있고, 실수를 줄일 수 있음

출처 : NHN클라우드 문서(https://docs.nhncloud.com/ko/Compute/Instance/ko/terraform-guide/)

01 Terraform & Ansible

» Ansible 이란?

Red Hat에서 개발 중인 프로비저닝, 구성 관리, 애플리케이션 배포, 오케스트레이션 등
여러 수동 IT 프로세스를 자동화하는 오픈소스 IT 자동화 툴

- 플레이북에 실행할 구성을 선언해 놓으면, 필요시마다 자동 실행 가능
  즉, 웹서버의 구성과, DB서버의 구성을 선언해 놓으면 관리자들은 필요할 때마다 그 구성대로 서버의 설정을 배포할 수 있음

출처 : https://kim-dragon.tistory.com/13

01 Terraform & Ansible

» Terraform과 Ansible 비교

유형	오케스트레이션 도구	컨피규레이션 관리 도구
구문	HCL	YAML
언어	선언적	절차적
기본접근	가변 인프라	불변 인프라
생명주기 관리	지원	미지원
기능	프로비저닝, 컨피규어링	프로비저닝, 컨피규어링
Agentless	O	O
Masterless	O	O

출처 : https://btcd.tistory.com/75

01 Terraform & Ansible

» Terraform과 Ansible 비교

유형	오케스트레이션 도구	컨피규레이션 관리 도구

오케스트레이션 도구는 복잡한 자동화와 조정 작업을 처리하는 데 초점을 맞추는 작업을 이르는 용어
여러 서버, 컨테이너, 서비스 간의 작업을 자동화하고 조율
주로 클라우드 환경에서의 리소스 관리, 애플리케이션 배포, 서비스 관리, 스케일링, 네트워킹 설정 등을 담당

서버나 다른 IT 시스템의 설정과 관리를 자동화하는 데 중점을 둔 용어
주로 서버 설정, 소프트웨어 설치, 시스템 업데이트, 보안 규칙 적용 등을 자동화하여 일관된 환경을 유지하고, 수작업으로 발생할 수 있는 오류를 최소화

출처 : https://btcd.tistory.com/75

01 Terraform & Ansible

» Terraform과 Ansible 비교

유형	오케스트레이션 도구	컨피규레이션 관리 도구

목적: 오케스트레이션 도구는 서비스와 애플리케이션 간의 상호작용과 조율을 관리하는 반면, 컨피규레이션 도구는 서버나 시스템의 설정을 일관되게 유지하고 자동화하는 데 중점

작업: 오케스트레이션 도구는 보통 "시스템 전체"에 걸친 작업을 수행하며, 여러 컴포넌트와 서비스를 통합
컨피규레이션 도구는 "단일 서버"나 "단일 시스템" 수준에서 더 세밀한 설정을 제공

출처 : https://btcd.tistory.com/75

01 Terraform & Ansible

» Terraform과 Ansible 비교

선언적(원하는 상태 설명)	절차적(어떻게 도달할지 설명)
IaC를 작성하는데 사용	**YAML 구문**을 사용하여 대상 인프라에서 수행할 절차를 정의함
선언적 형식인 **HCL**을 사용하고 코드가 작성되는 순서는 중요하지 않음	Ansible YAML 스크립트는 **절차적 언어**로 스크립트를 작성할 때 아래로 실행
코드는 **여러 파일에 분산**될 수도 있음	앤서블 스크립트는 **"Ansible Playbook"**에 저장
코드를 어떻게 작성하든 Terraform은 **종속성을 식별**하고 인프라를 프로비저닝	특정 일련의 작업을 수행해야 하는 경우 플레이북에서 동일한 작업을 정의하고, 작업은 작성된 **순서대로 수행**
기존 인프라를 작성하거나 **코드로 쉽게 변환**	루트 사용자로 지정된 가상머신에 Apache 서버를 설치하려면 **설치 작업을 정의하기 전에 사용자 생성 단계를 작성**해야 함

출처 : https://btcd.tistory.com/75

01 Terraform & Ansible

» Terraform과 Ansible 비교

변경 가능	변경 불가
원하는 **인프라** 상태를 입력으로 가져와 **프로비저닝**	기본적으로 **변경 불가능**
선언적 형식인 **HCL**을 사용하고 코드가 작성되는 순서는 중요하지 않음	최신 버전의 플레이북에 따라 구성 변경 사항을 일관되게 **유지하고 관리**
클라우드 공급자가 인프라를 재부팅 하거나 교체하지 않고 **변경 사항을 구현할 수 없는 경우** **인프라를 변경**	변경 사항은 기본 **인프라의 "교체"에** **영향을 주지 않으며,** **지정된 구성 요소의 구성만 복구하거나 수정**
이전 **인프라 구성요소를 제거하고** 최신 구성 설정이 있는 새 구성 요소로 교체	**Ansible에서 관리하는 서버는** **수행한 구성의 변경 사항을 기록**

출처 : https://btcd.tistory.com/75

01 Terraform & Ansible

» Terraform과 Ansible 비교

HashiCorp Terraform

ANSIBLE

우수한 스케줄링 기능과 함께
제공되며 사용자에게
매우 **쾌적한 환경을 제공**

보다 **안전한 보안과**
ACL기능 제공

Docker와 통합 우수

전통적인 **자동화 프레임워크**와
함께 **편안하게 조정**되기 때문에
숙성한 도구

대상 장치가
어떻게 **최종 상태**가
되었는지에 대한
명확한 증거가 없음

논리적 종속성,
오케스트레이션 서비스 및 상호
연결된 응용 프로그램과 같은
서비스에는 부적합

출처 : https://btcd.tistory.com/75

02 Vault

» Vault 란?

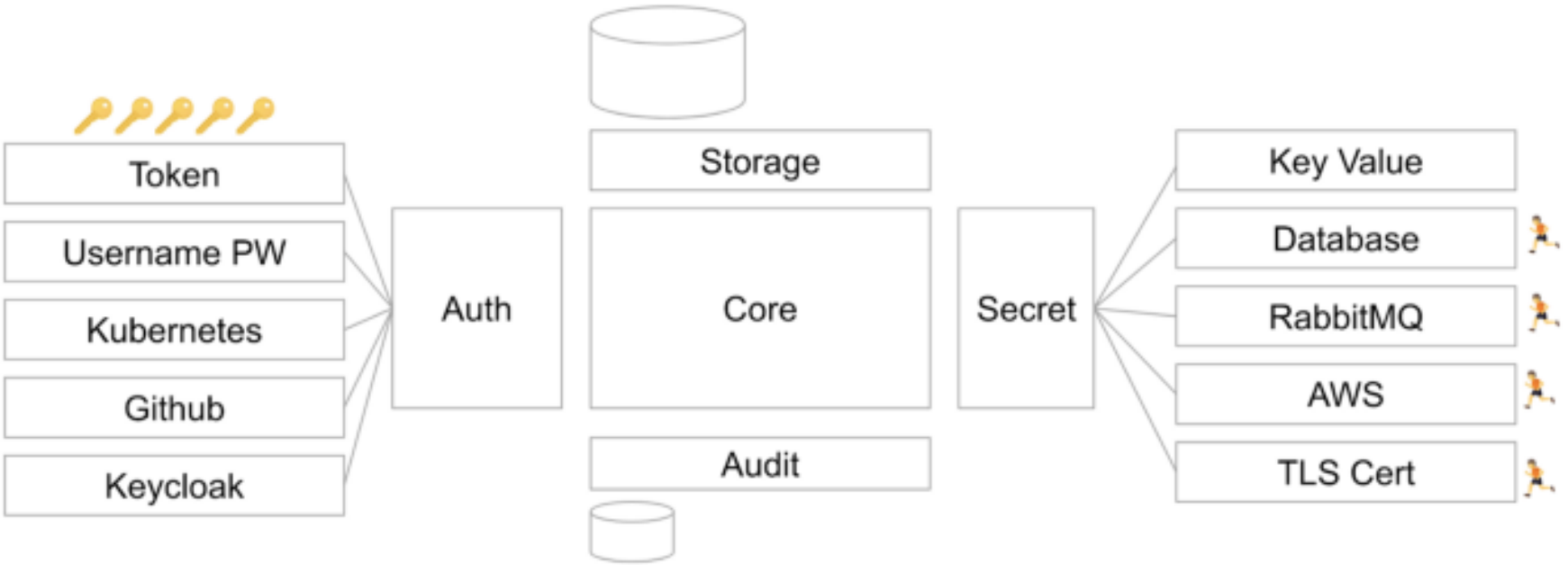
HashiCorp에서 제공하는 **크로스플랫폼 패스워드 및 인증 관리 시스템**

- UI, CLI, HTTP API 등의 인터페이스를 제공하고 있으며, 저장된 비밀 정보를 안전하게 사용할 수 있는 방법들을 제공
- Vault의 storage backend는 암호화된 데이터를 저장하기 위한 스토리지를 담당하며, 스토리지의 종류, 가용성 등을 책임지지 않음

출처 : https://dealicious-inc.github.io/2021/12/21/hashicorp-vault.html

02 Vault

» Vault 구성요소 체계

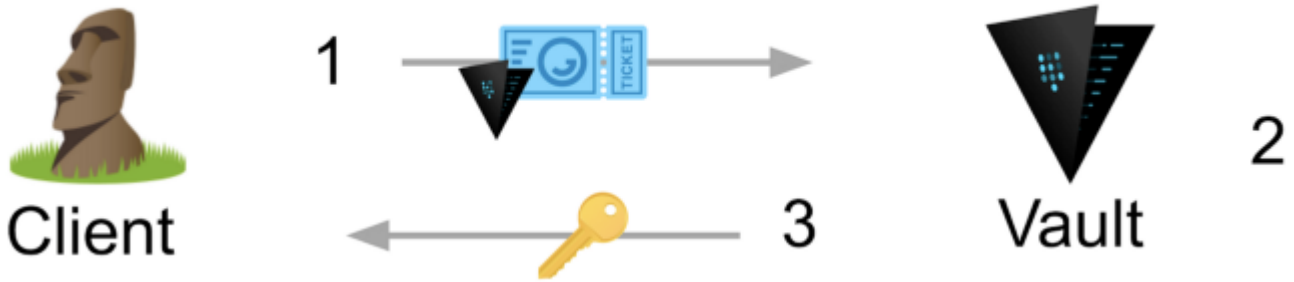


Vault 구성 요소 체계

출처 : https://dealicious-inc.github.io/2021/12/21/hashicorp-vault.html

02 Vault

» Vault 구조



1. 클라이언트는 Vault에 토큰과 시크릿 경로를 보냄
2. Vault는 클라이언트가 원하는 데이터(시크릿 정보)를 가지고 있음
3. Vault는 클라이언트의 토큰 권한을 확인, 허용되면 요청된 시크릿 정보를 반환

출처 : https://dealicious-inc.github.io/2021/12/21/hashicorp-vault.html

03 Ceph

» Ceph 란?

오픈소스 소프트웨어(Software Defined Storage) 스토리지 플랫폼

- 단일 분산 컴퓨터 클러스터에 object 스토리지를 구현하고 object, block 및 file level의 스토리지 기능을 제공
- Single point of failure이 없는 완전히 분산된 운영을 주소 목표로 하며 엑사바이트 수준으로 scale-out 가능

출처 : https://chhanz.github.io/ceph/2021/08/18/ceph-architecture/

03 Ceph

» Ceph 특징

소프트웨어 정의 스토리지 (Software Defined Storage) 이점

- **비용 대비 성능 절충** : 애플리케이션이 동일한 소프트웨어 스택을 사용하여 서로 다른 하드웨어 및 내구성 구성의 성능 및 비용 절충을 선택할 수 있음
- **유연한 인터페이스** : 산업 표준 API를 선택하거나, 애플리케이션에 클라이언트 라이브러리를 내장하거나, 필요한 경우 독점 API 사용 가능
- **다양한 스토리지 구현** : object, block 및 file 추상화 전반에 걸쳐 동일한 스토리지 소프트웨어 스택을 활용하여 R&D 및 운영 비용을 절감

출처 : https://chhanz.github.io/ceph/2021/08/18/ceph-architecture/

03 Ceph

» Ceph 구조

- **Ceph OSD daemon**
  Ceph 클라이언트를 대신하여 데이터를 저장하며, 데이터 관리 기능을 수행
- **Ceph Monitor**
  Ceph 스토리지 클러스터의 현재 상태에 대한 Ceph 스토리지 클러스터 맵의 마스터 복사본을 유지
- **Ceph Manager**
  Ceph Monitor 대신 placement groups(PG), 프로세스 메타데이터 및 호스트 메타데이터에 대한 자세한 정보를 유지하여 규모에 맞게 성능을 크게 향상시킴
- **MDS(Metadata Servers)**
  클라이언트에 의한 효율적인 POSIX 명령 실행을 위해 CephFS 에서 사용하는 메타데이터를 저장

출처 : https://chhanz.github.io/ceph/2021/08/18/ceph-architecture/