고려대학교
KOREA UNIVERSITY

KU-The Future

Ne-Ka-La-Kube

0. Introduction

Distributed & Cloud Computing Lab.

석박사통합과정 김명현

2021.05.28

Table of Contents

Section 1

“Concept of Kubernetes”

What Kubernetes can do

Service Discovery and Load Balancing

- **DNS Name**이나 **자체 IP 주소**를 사용해서 컨테이너를 노출 가능
- 네트워크 트래픽에 대한 로드 밸런싱을 통해 안정적인 배포를 달성함

Storage Orchestration

- 로컬 스토리지, 퍼블릭 클라우드 스토리지 등 원하는 스토리지 시스템을 자동으로 마운트 가능

Automated Rollouts and Rollbacks

- 컨테이너의 배포에 대해 **원하는 상태(desired state)**를 미리 작성해둘 수 있음
- 쿠버네티스는 원하는 상태를 달성하기 위해 자동으로 Rollout과 Rollback을 시도함

What Kubernetes can do

Automatic Bin Packing

- 각 컨테이너가 리소스(CPU, RAM 등)를 얼마나 필요로 하는지를 설정 가능
- 쿠버네티스는 리소스 사용률을 자동으로 최적화함

Self-Healing

- 컨테이너가 fail이 발생하면 다시 시작하고, 교체함
- *user-defined health check*에 응답하지 않는 컨테이너들은 강제로 종료함
- 이런 과정을 클라이언트에게서 감춤

Secret and Configuration Management

- OAuth Token, SSH Key등과 같은 정보들을 저장하고 관리

Section 2

“ Installation ”

Managed Kubernetes Services

Google Kubernetes Engine (GKE)

- **GCP**(Google Cloud Platform)를 통해서 제공됨
- 웹 콘솔을 통해서도 클러스터 구축 및 애플리케이션 배포 가능
- **워커 노드 수**를 기준으로 과금하는 모델

High Availability

- 인프라 관련 구축 및 유지 보수 작업을 GCP에서 전부 대신해준다.
- 여러 **Failure Zone** (AWS의 AZ와 대응)에 걸쳐서 워커 노드를 분산시켜 **Multizone** 클러스터를 구성할 수 있음

2. Installati

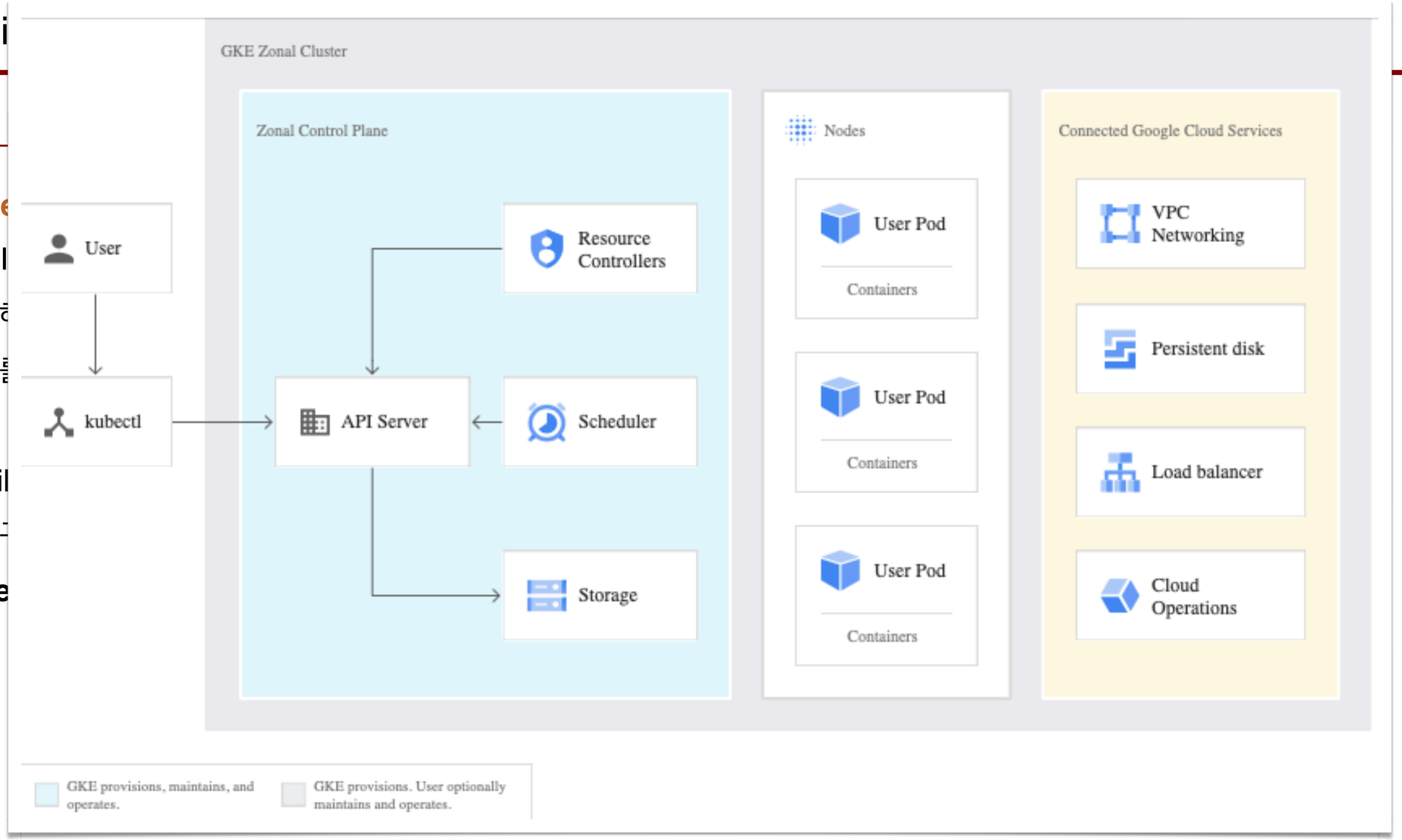
Managed

Google Kube

- **GCP**(Googl
- 웹 콘솔을 통
- **워커 노드 수**

High Availabil

- 인프라 관련
- 여러 **Failure**



[Figure 1]

Managed Kubernetes Services

AWS Elastic Kubernetes Service (EKS)

- **AWS**(Amazon Web Service)를 통해서 제공됨
- 구글이나 MS의 서비스와 다르게 **마스터 노드도 비용이 청구**됨
- GKE에 비해 최근에 런칭된 서비스라 Node Health Monitoring 등 여러 기능에서 아직 열세를 보임 [1]

Microsoft Azure Kubernetes Service (AKS)

- **MS Azure**를 통해서 제공됨
- GKE와 마찬가지로 마스터 노드는 내부적으로 관리되며, 직접 접근 불가
- 워커 노드 수를 기준으로 비용이 청구됨

OpenShift

- 쿠버네티스 서비스 + CI, 빌드 도구, 테스트 도구, 애플리케이션 배포, 모니터링 등 소프트웨어 개발 주기 관리 기능도 포함

Kubernetes Installers for Self-Hosting

kops

- 쿠버네티스 클러스터에서 provisioning, orchestration을 **직접 수행**함
- AWS에서 쿠버네티스를 직접 호스팅할 경우에 좋은 선택

kubespray

- 온프레미스 베어 메탈 서비스에 쿠버네티스 클러스터를 구축하는데에 중점을 두며, **다중 플랫폼** 또한 지원된다.
- provisioning, orchestration을 위해서 **Ansible**을 사용함

  (kops처럼 직접 수행하지 않으므로 더 유연함)
- Ansible에 익숙한 경우 편리함
- 하나의 플랫폼만 사용할 것이라면 kops가 더 좋은 선택이 될 수 있음. [2]

Kubernetes Installers for Self-Hosting

kubeadm

- 쿠버네티스에서 제공하는 클러스터 설치 및 관리 툴
- 클러스터에 인프라를 프로비저닝 하지는 않으며, 부트스트랩까지만 관여함
- 대부분의 설치 도구들이 내부적으로 kubeadm을 사용함
- 베어 메탈 서버나 클라우드 인스턴스에 설치하는데 적합

Installation via kubeadm

Before beginning

- 각 노드는 **2GB** 이상의 RAM을 갖출 것
- 각 노드는 2개 이상의 코어를 가진 CPU를 가질 것
- **Swap**을 비활성화할 것

Costs of Self-Hosting Kubernetes

Control-Plane이 고가용성을 보장하는가?

- 마스터 노드가 fail이 발생해도 클러스터가 정상적으로 작동하는지.
- Control Plane 없이도 실행 중인 애플리케이션이 fault-tolerant 한지.

워커 노드가 고가용성을 보장하는가?

- 여러 워크 노드에서 fail이 발생한 경우 워크로드 실행이 중단되지 않는지.
- 새로운 노드를 자동으로 프로비저닝해서 복구할 수 있는지. 혹은 수동 조치가 필요하지 않은지.

클러스터는 안전하게 설정되었는가?

- 내부 컴포넌트들은 TLS 암호화와 신뢰할 수 있는 인증서로 통신하는지.
- 사용자와 애플리케이션에게 최소한의 권한이 부여되었는지.
- etcd에 대한 접근이 적절하게 제어되는지.

Section 3

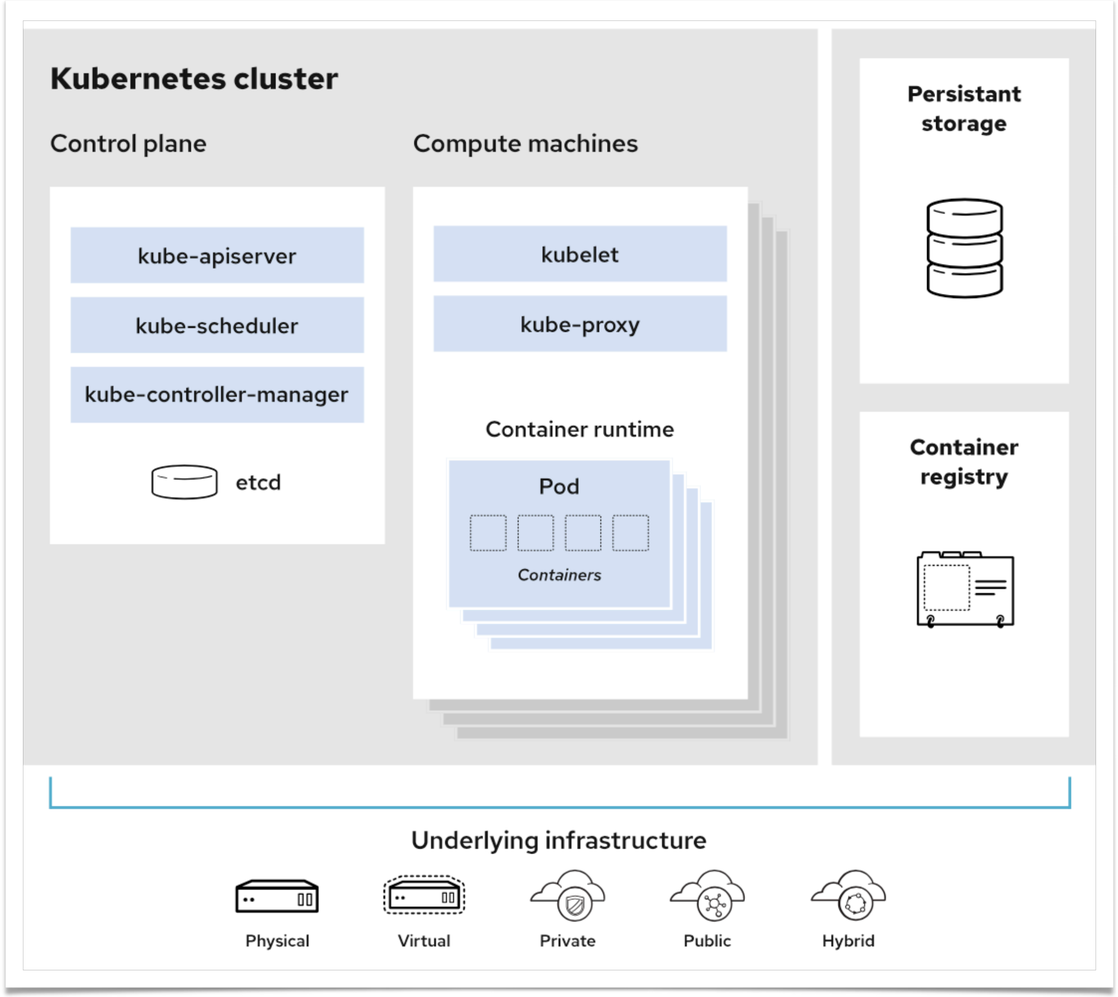
“Architecture”

Kubernetes Cluster

- 쿠버네티스 클러스터는 온 프레미스 서버, 프라이빗/퍼블릭 클라우드 등 자유롭게 구축 가능

- Control Plane이 마스터 노드 역할
  - 마스터 노드는 여러 머신으로 구성할 수 있음
  - 실제 구동에서는 한 노드만이 Control Plane 역할을 수행하다가, fail이 발생하면 다른 백업 노드로 migration을 함

- 워커 노드는 Container Runtime(docker, containerd)과 kubelet, kube-proxy가 구동되며, 이는 마스터 노드에서도 구동됨



[Figure 2]

Control Plane Components

kube-apiserver (pod)

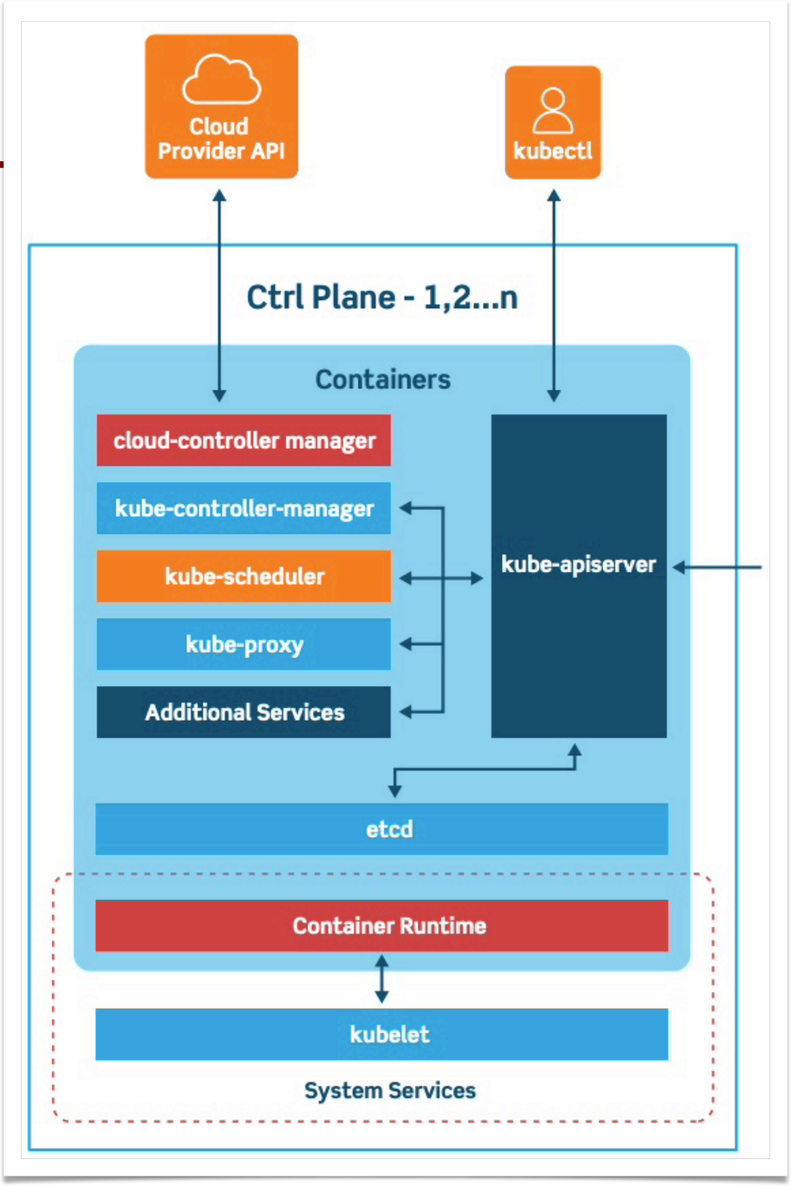
- 쿠버네티스 API를 노출하여 Control Plane의 프론트엔드 역할 수행
- kubectl, kubeadm을 통해 접근 가능, REST API로도 접근 가능

etcd (pod)

- 클러스터에 관련된 설정, 상태 데이터를 저장하는 key-value store
- kube-apiserver를 통해서만 접근이 가능함

kube-scheduler (pod)

- 노드가 할당되지 않은 새로운 파드를 감지하고, 실행할 노드를 선택하는 스케줄링을 담당



[Figure 3]

Control Plane Components

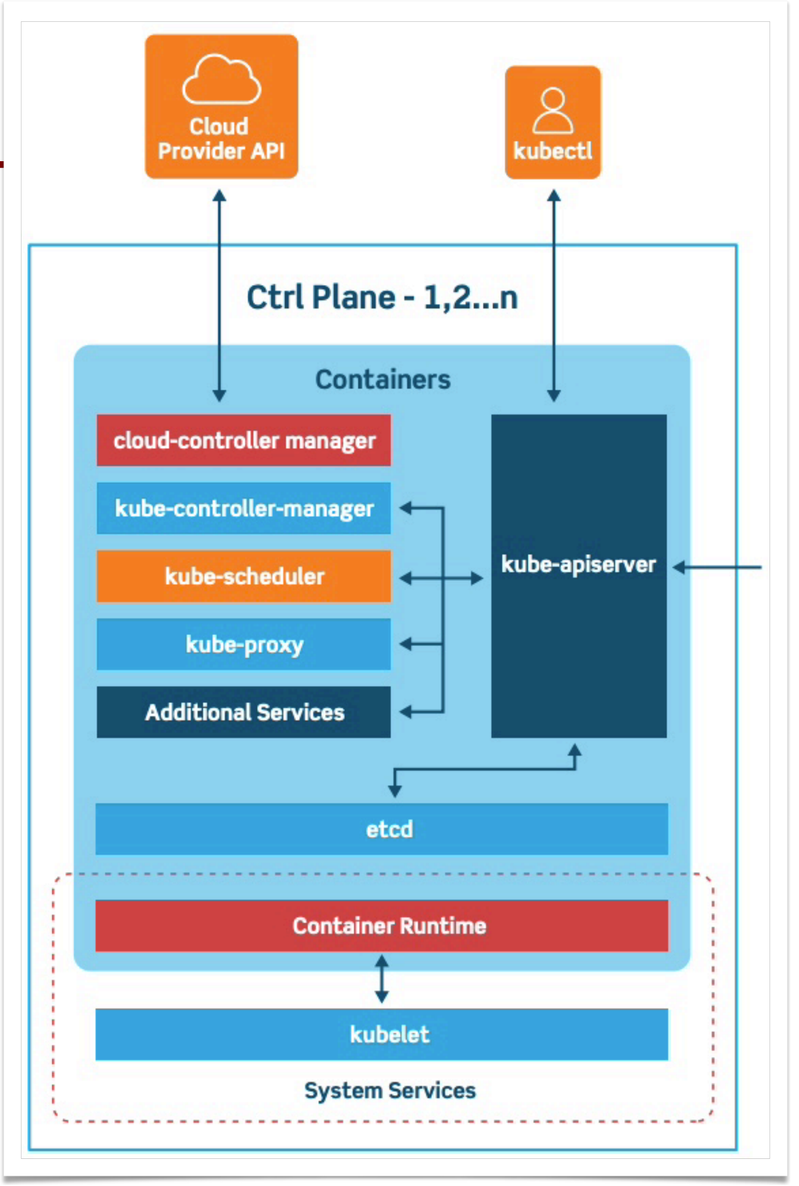
kube-controller-manager (pod)

- 하위 컨트롤러들을 단일 바이너리로 통합, 컴파일해서 구동

- *Node Controller* : 노드가 다운되었을 때 대응
- *Replication Controller* : 시스템의 모든 레플리케이션에 대해 알맞은 수의 파드를 유지함
- *Endpoint Controller* : 서비스와 파드를 연결
- *Service Account & Token Controller* : 새로운 네임스페이스의 기본 계정과 API 토큰을 생성

cloud-controller-manager (pod)

- 하위 컨트롤러들을 단일 바이너리로 통합, 컴파일해서 구동
- 클러스터를 cloud provider가 제공하는 API에 연결
- 외부 클라우드 플랫폼을 사용하지 않는 경우 존재하지 않음



[Figure 3]

Node Components

kubelet

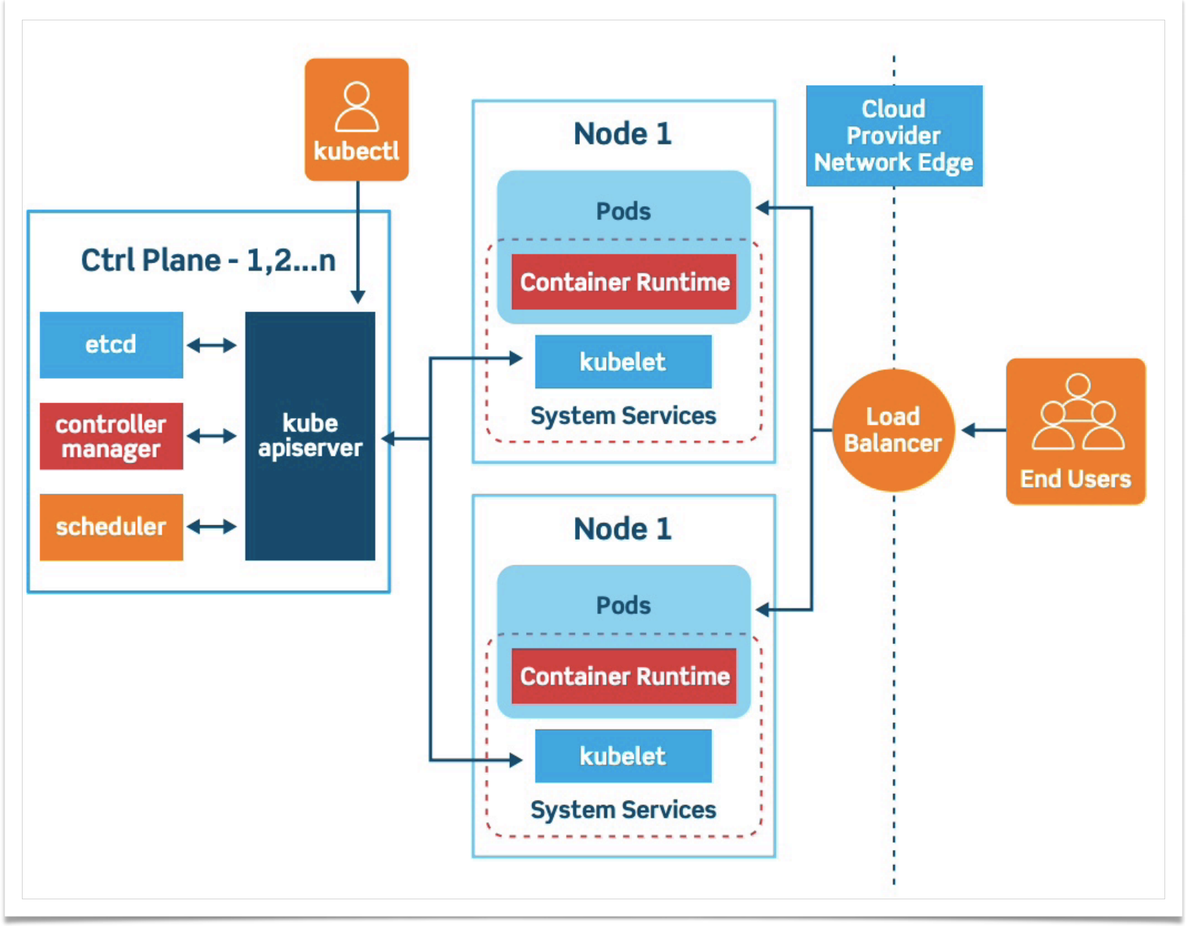
- Control Plane과 통신
- 컨테이너가 파드에서 실행되도록 Container Runtime을 제어
- 제공받은 PodSpec의 집합을 받아서, 스펙에 따라 파드가 동작하도록 관리

kube-proxy (pod)

- 네트워크 프록시 역할, *service* 개념의 구현부
- 노드의 네트워크 규칙을 유지 및 관리

Container Runtime

- Docker, containerd, CRI-O 등 Kubernetes CRI를 만족하는 컨테이너 런타임



[Figure 4]

Other Components

Persistent Storage

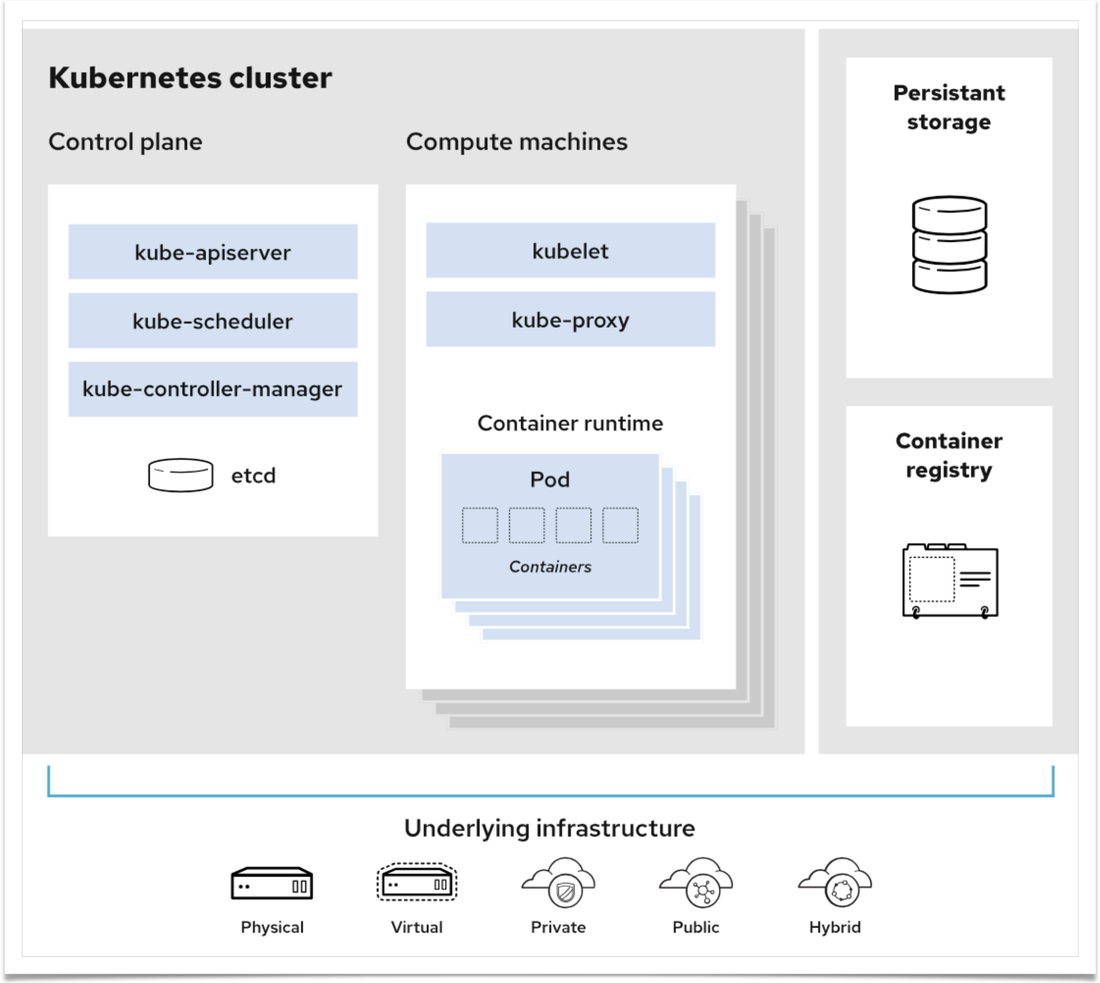
- *Persistent Volume*을 관리하는 스토리지

Persistent Volume (PV)

- 관리자가 프로비저닝하거나, 동적으로 프로비저닝한 클러스터 스토리지
- 컨테이너가 죽더라도 PV는 유지되어, 새롭게 시작된 컨테이너에서 PV에 있는 데이터를 바로 활용할 수 있음
- 컨테이너가 PV를 사용하고자 할 때는 PVC(*PersistentVolumeClaim*)를 통해 요청해서 PV와 바인딩

Container Registry

- 쿠버네티스에서 사용하는 컨테이너들의 이미지가 저장되는 공간



[Figure 2]

Section 4

“YAML”

YAML

- 쿠버네티스에서 오브젝트의 Spec과 Status는 YAML이나 JSON으로 표현됨
- JSON보다 더 가독성이 높아 주로 YAML을 사용하므로, 필수로 익혀둬야 한다.
- YAML 내에 JSON 문법을 포함할 수도 있음 (우측 자료 사진 라인 10)

간단한 문법

- 확장자 : **.yml**, **.yaml**
- 주석 : #로 시작하는 문장
- **" : "** : map의 key
- **" - "** : list를 뜻함

```
1   apiVersion: v1
2   kind: Pod
3   metadata:
4     name: private-image-test-1
5   spec:
6     containers:
7       - name: uses-private-image
8         image: $PRIVATE_IMAGE_NAME
9         imagePullPolicy: Always
10        command: [ "echo", "SUCCESS" ]
```

YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: private-image-test-1
spec:
  containers:
    - name: uses-private-image
      image: $PRIVATE_IMAGE_NAME
      imagePullPolicy: Always
      command: [ "echo", "SUCCESS" ]
```

YAML 예시

```json
{
    "apiVersion": "v1",
    "kind": "Pod",
    "metadata": {
        "name": "private-image-test-1"
    },
    "spec": {
        "containers": [
            {
                "name": "uses-private-image",
                "image": "%PRIVATE_IMAGE_NAME",
                "imagePullPolicy": "Always",
                "command": [ "echo", "SUCCESS" ]
            }
        ]
    }
}
```

예시 YAML을 JSON으로 변환

Section 5

“ Pod ”

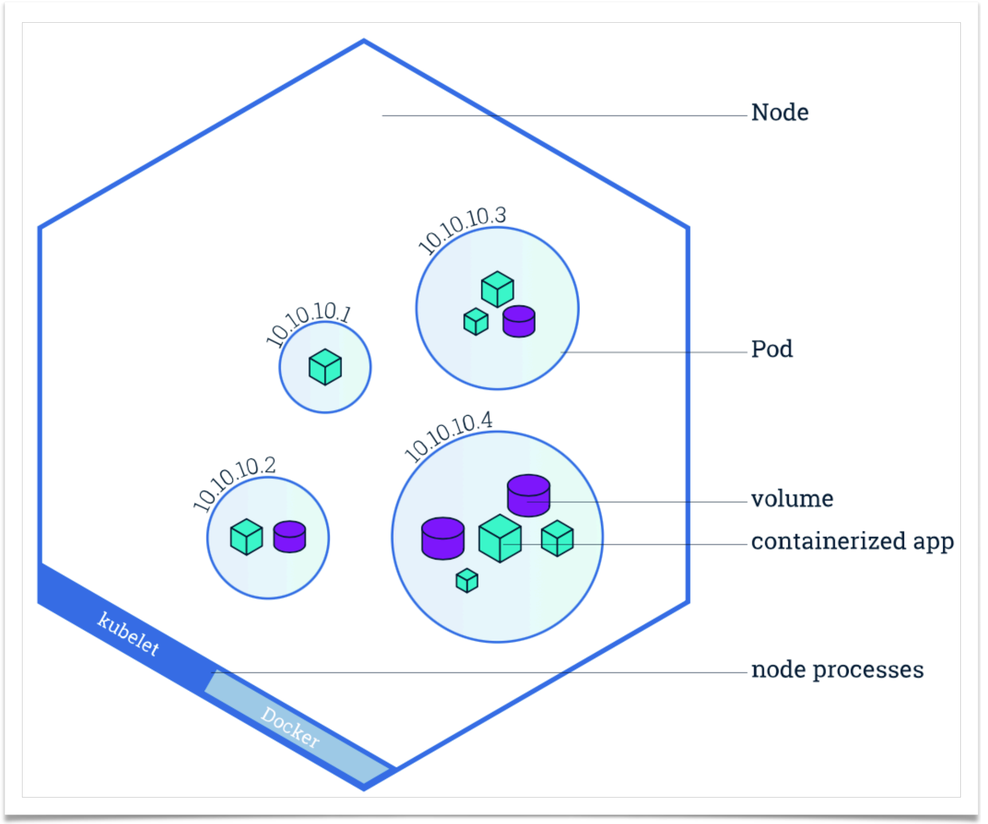
Pod

: 쿠버네티스에서 생성하고 관리할 수 있는 배포 가능한 가장 작은 컴퓨팅 단위로,
애플리케이션 별 "논리적 호스트"를 모델링한 오브젝트이다.

- 하나 이상의 컨테이너의 그룹으로 구성됨 (컨테이너들은 상대적으로 밀접하게 결합됨)
- 파드 내에서는 **스토리지와 네트워크를 공유**함

∴ Pod : 공유 네임스페이스와 공유 파일시스템 볼륨이 있는 컨테이너 그룹

! 주의 : 파드는 프로세스가 아니라, 컨테이너를 실행하기 위해 논리적으로 격리된 환경



[Figure 5]

Pod

- 사용자가 직접 개별 파드를 만드는 경우는 거의 없으며,

  워크로드 리소스를 사용해 생성함

  - 워크로드 리소스 : *Deployment*, *StatefulSet*, *DaemonSet* 등

PodTemplate

- 워크로드 리소스의 컨트롤러들은 파드 템플릿 (PodTemplate)을

  기반으로 실제 파드를 생성함

- 파드 템플릿을 수정해도 이미 존재하는 파드에는 영향이 없음

- 다만, 컨트롤러는 업데이트된 파드 템플릿을 기반으로 새로운 파드를 생성하고, 기존 파드를 종료함으로써 교체를 함

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: hello
spec:
  template:
    # 여기서부터 파드 템플릿이다
    spec:
      containers:
      - name: hello
        image: busybox
        command: ['sh', '-c', 'echo "Hello, Kubernetes!" && sleep 3600']
      restartPolicy: OnFailure
    # 여기까지 파드 템플릿이다
```

[Figure 6] PodTemplate 예제

Pod

모든 오브젝트에 포함되는 메타데이터

PodSpec

PodStatus

```go
// Pod is a collection of containers that can run on a host. This resource is created
// by clients and scheduled onto hosts.
type Pod struct {
    metav1.TypeMeta `json:",inline"`
    // Standard object's metadata.
    // More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata
    // +optional
    metav1.ObjectMeta `json:"metadata,omitempty" protobuf:"bytes,1,opt,name=metadata"`

    // Specification of the desired behavior of the pod.
    // More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status
    // +optional
    Spec PodSpec `json:"spec,omitempty" protobuf:"bytes,2,opt,name=spec"`

    // Most recently observed status of the pod.
    // This data may not be up to date.
    // Populated by the system.
    // Read-only.
    // More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status
    // +optional
    Status PodStatus `json:"status,omitempty" protobuf:"bytes,3,opt,name=status"`
}
```

Pod Lifecycle

Phase

- *PodStatus* 오브젝트의 *phase* 필드로 알 수 있다.

- **Pending** : 파드가 클러스터에서 승인되었지만, 컨테이너들이 설정되지 않았고 실행할 준비가 되지 않은 상황.
  - 즉, 컨테이너 이미지 다운로드 시간을 포함한 스케줄되기 이전.
- **Running** : 파드가 노드에 바인딩되었고, 모든 컨테이너가 생성됨.
- **Succeeded** : 모든 컨테이너가 성공적으로 종료되었고, 재시작되지 않을 것임.
- **Failed** : 모든 컨테이너가 종료되었고, 하나 이상의 컨테이너가 실패로 종료됨.
- **Unknown** : 파드 상태가 추적이 불가능한 경우.

Pod

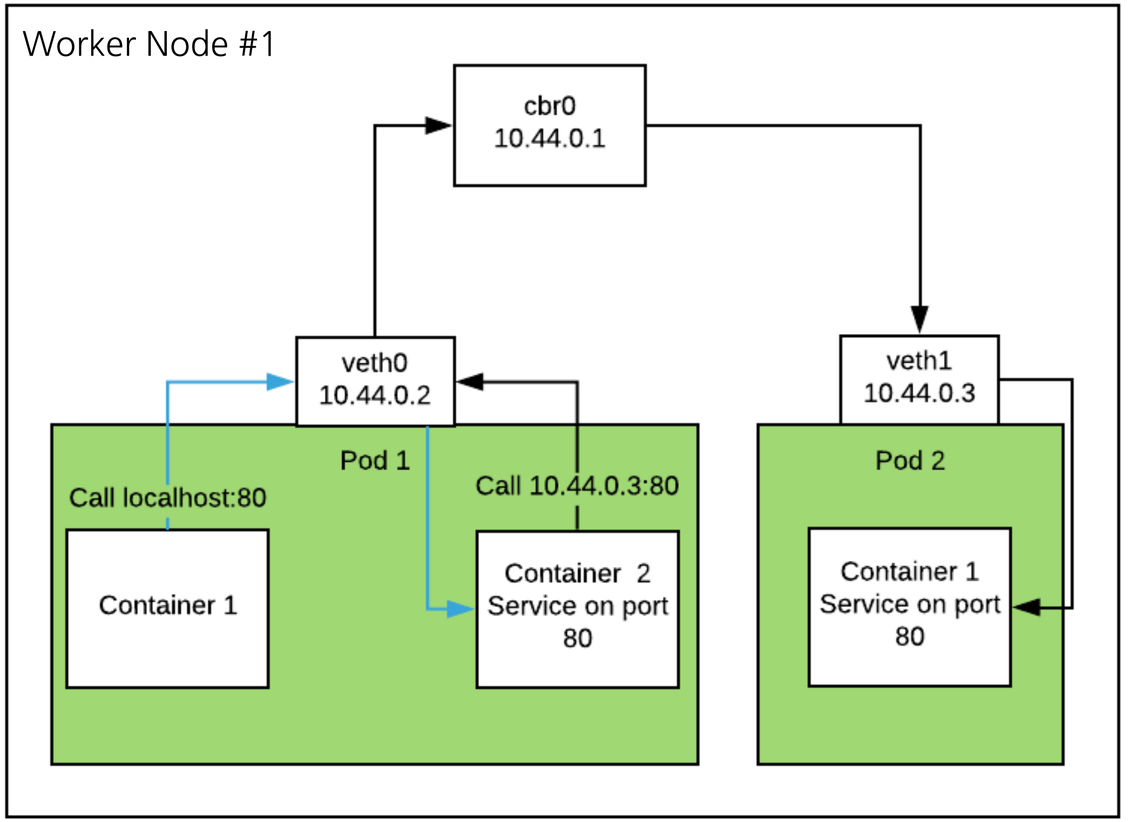
Networking in Pods

- 각 파드에는 고유한 IP 주소가 할당되며,

  이 네트워크 네임스페이스는 파드 내 모든 컨테이너가 공유함

  (각 컨테이너는 localhost를 사용해서 통신 가능)

- 파드를 각각 격리된 호스트로 생각할 수 있다.



[Figure 7]

Section 6

“ ReplicaSet ”

ReplicaSet

- 파드 집합의 실행을 안정적으로 유지하기 위해서, 파드의 개수(레플리카)를 일정하게 유지함

- ReplicaSet Controller는 Label Selector를 통해 ReplicaSet에 연결된 파드를 파악하고, 이 파드들의 개수를 ReplicaSet 스펙에 따라 유지하는 역할만을 한다.

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # 케이스에 따라 레플리카를 수정한다.
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

[Figure 8]

```
NAME             READY   STATUS    RESTARTS   AGE
frontend-b2zdv   1/1     Running   0          6m36s
frontend-vcmts   1/1     Running   0          6m36s
frontend-wtsmm   1/1     Running   0          6m36s
```

[Figure 9]

Section 7

“Deployment”

Deployment

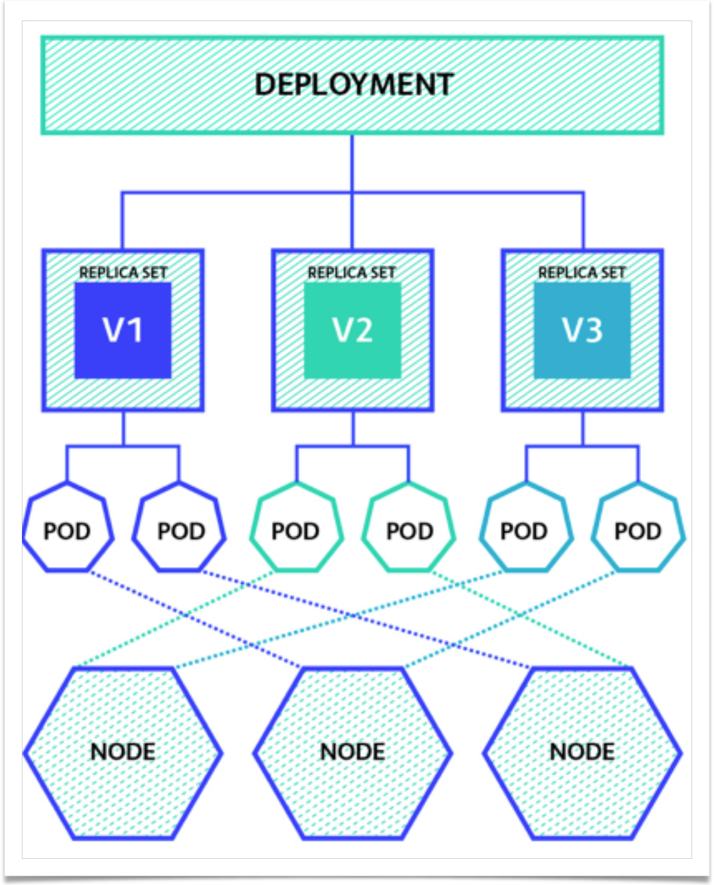
: 쿠버네티스가 애플리케이션 인스턴스를 어떻게 생성하고 업데이트해야 하는지 선언함.

- Deployment Controller는 선언한 스펙에 따라 파드의 개수, 상태 등을 **ReplicaSet**을 생성해 유지함

Deployment의 역할 [3]

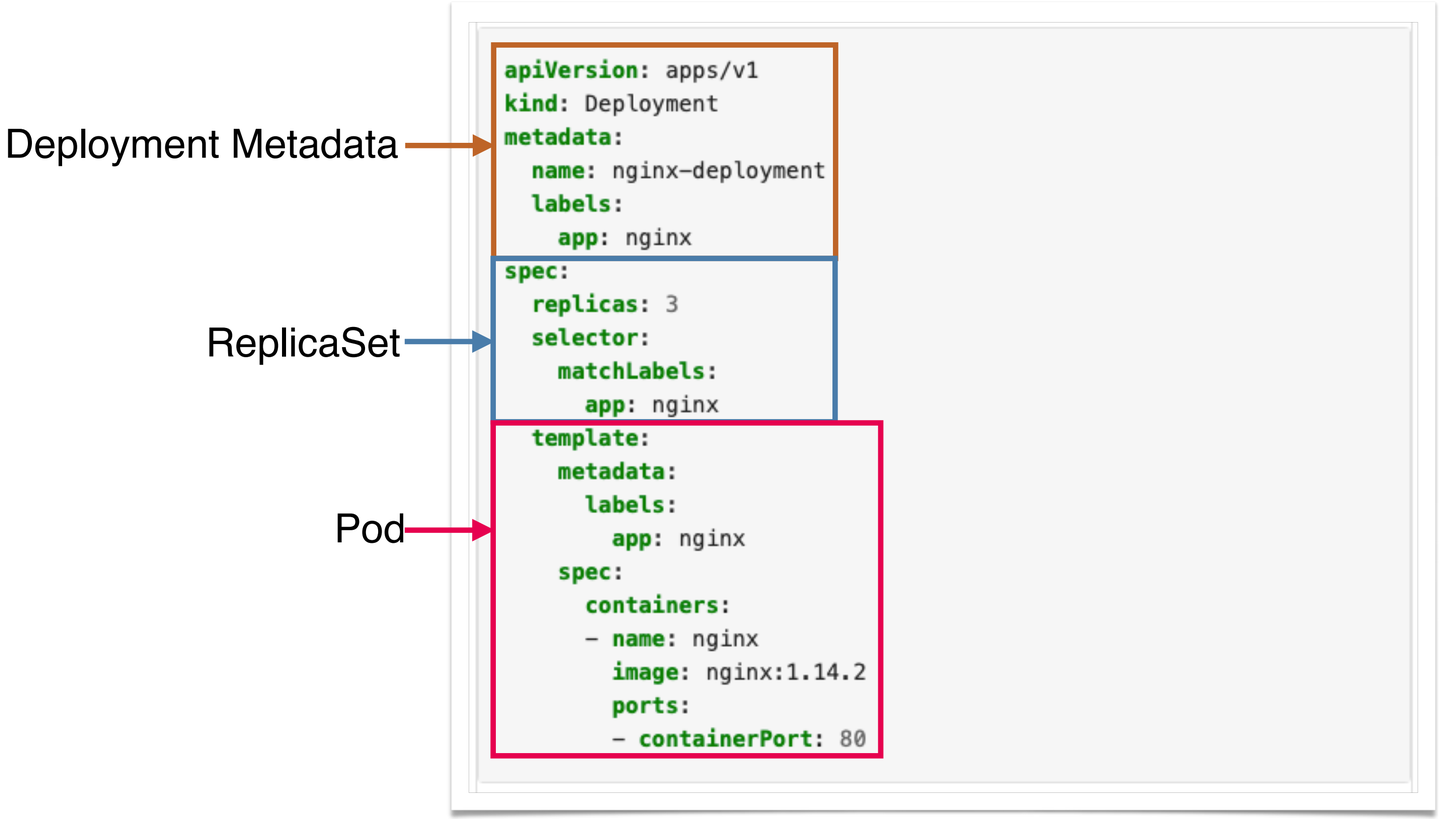
- 파드의 scale in / out하는 기준을 정의
- 파드의 배포된 버전을 모두 추적
- 배포된 파드에 대한 Rollback 수행

∴ Deployment : ReplicaSet보다 더 추상화된 오브젝트로, 각 파드의 레플리카 뿐만 아니라 버전도 추적하여 관리하는 기능도 제공함



[Figure 10]

Deployment



[Figure 11]

Section 8

“ Service ”

Kubernetes Networking

- 파드 내의 컨테이너는 루프백(loopback)을 통해 통신함
- 클러스터 네트워킹은 서로 다른 파드 간 통신을 제공함
- 파드에서 실행 중인 애플리케이션을 클러스터 외부에서 접근하기 위해서는 ***Service***라는 리소스를 사용

Service

기존의 문제점

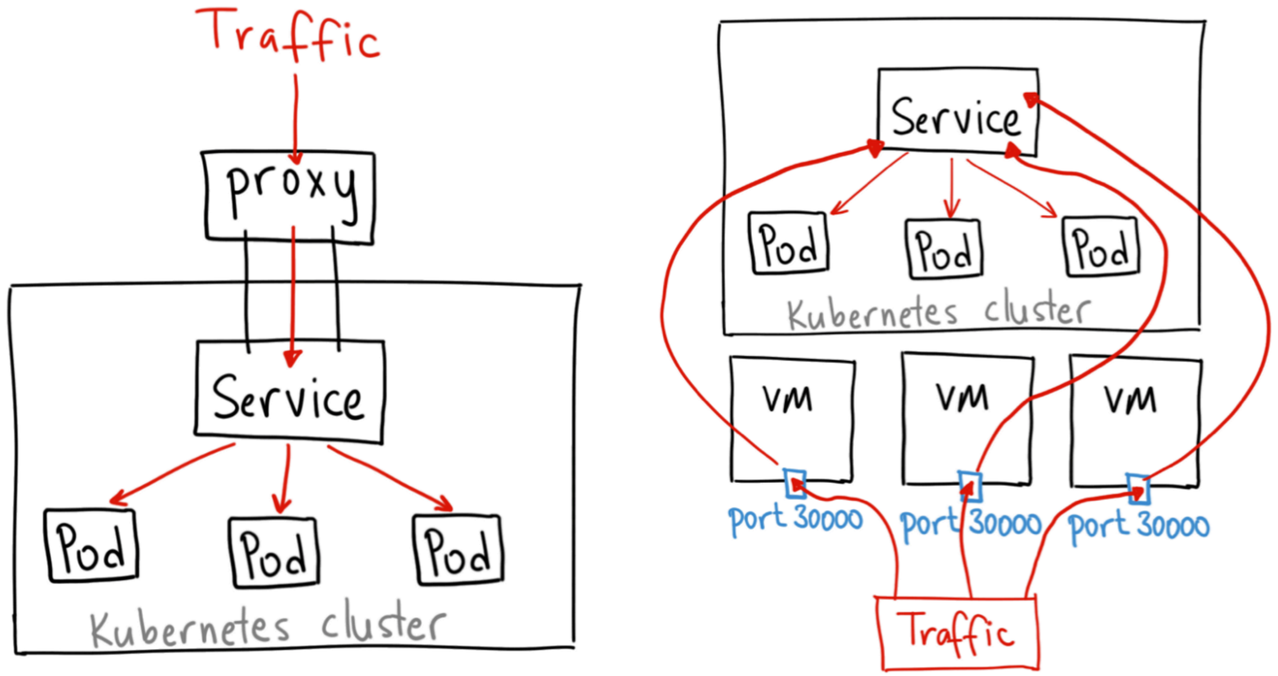
- 각 파드는 클러스터에서 고유 IP를 가지며, 동적으로 생성되고 제거될 수 있음
- Deployment 관점에서 봤을 때, 한 시점에 실행되는 파드가 잠시 후 실행되는 파드와 다를 수 있음

∴ 프론트엔드가 워크로드의 백엔드를 사용하기 위해, 연결할 IP 주소를 찾기 어려움.

Service

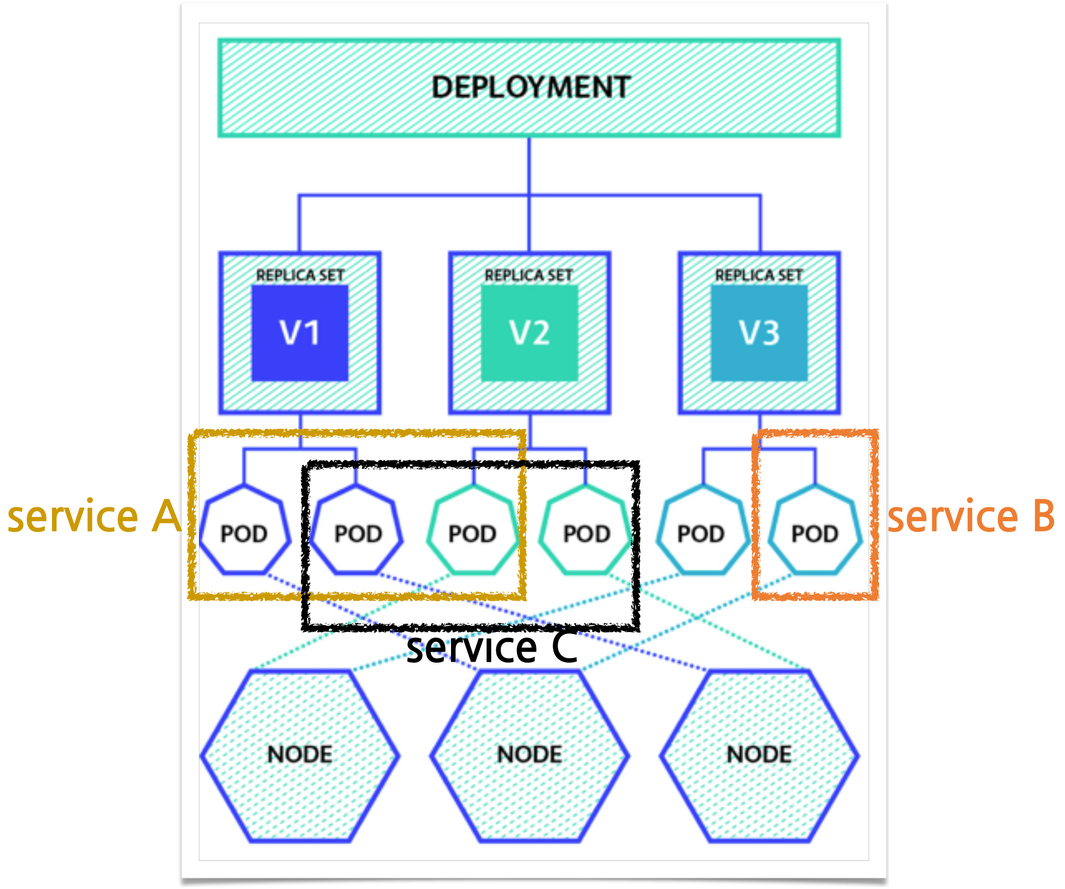
- 파드의 논리적 집합과, 그것들에 접근할 수 있는 정책을 정의하는 추상적 개념

Service



[Figure 12] ClusterIP & NodePort

Pod, ReplicaSet, Deployment, Service



Q & A

Appendix - References

J. Arundel, J. Domingus, *Cloud Native DevOps with Kubernetes*, O'Reilly Media, 2019.

(번역판) 쿠버네티스를 활용한 클라우드 네이티브 데브옵스 (최경현 옮김), 한빛미디어.

쿠버네티스 공식 문서 https://kubernetes.io/ko/docs/home/

2. Installation

- [Figure 1] https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-architecture
- [1] https://www.stackrox.com/post/2021/01/eks-vs-gke-vs-aks-jan2021/
- [2] https://github.com/kubernetes-sigs/kubespray/blob/master/docs/comparisons.md

3. Architecture

- [Figure 2] https://www.redhat.com/ko/topics/containers/kubernetes-architecture
- [Figure 3] https://platform9.com/blog/kubernetes-enterprise-chapter-2-kubernetes-architecture-concepts/
- [Figure 4] https://luispreciado.blog/posts/kubernetes/core-concepts/cluster-architecture

Appendix - References

5. Pod

- [Figure 5] https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/
- [Figure 6] https://kubernetes.io/docs/concepts/workloads/pods/
- [Figure 7] https://blog.neuvector.com/article/advanced-kubernetes-networking

6. ReplicaSet

- [Figure 8, 9] https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/

7. Deployment

- [Figure 10] https://ooeunz.tistory.com/124
- [3] https://blog.wonizz.tk/2019/09/17/kubernetes-deployment/
- [Figure 11] https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

8. Service

- [Figure 12] https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0

End of Document