# < Kubernetes >

## 1. Kubernetes 란?

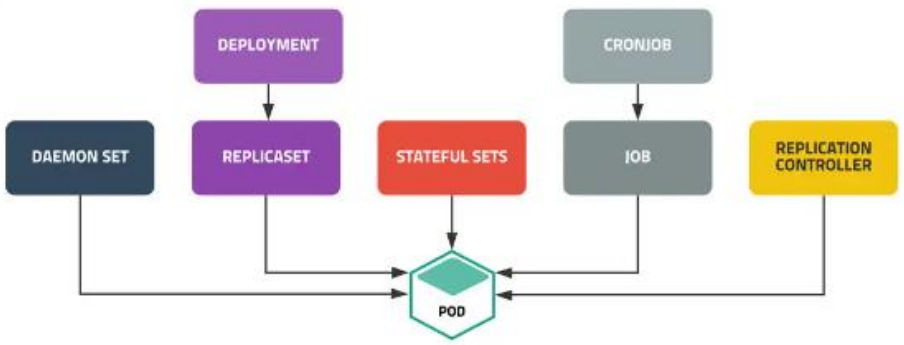

*kubernetes logo*

**쿠버네티스는 컨테이너를 쉽고 빠르게 배포/확장하고 관리를 자동화해주는 오픈소스 플랫폼입니다.**

## 2. Kubernetes 특징

컨테이너 오케스트레이션의 기본 기능 외에 쿠버네티스가 가지는 차별화된 특징은 다음과 같습니다.

### 2.1) 다양한 배포 방식

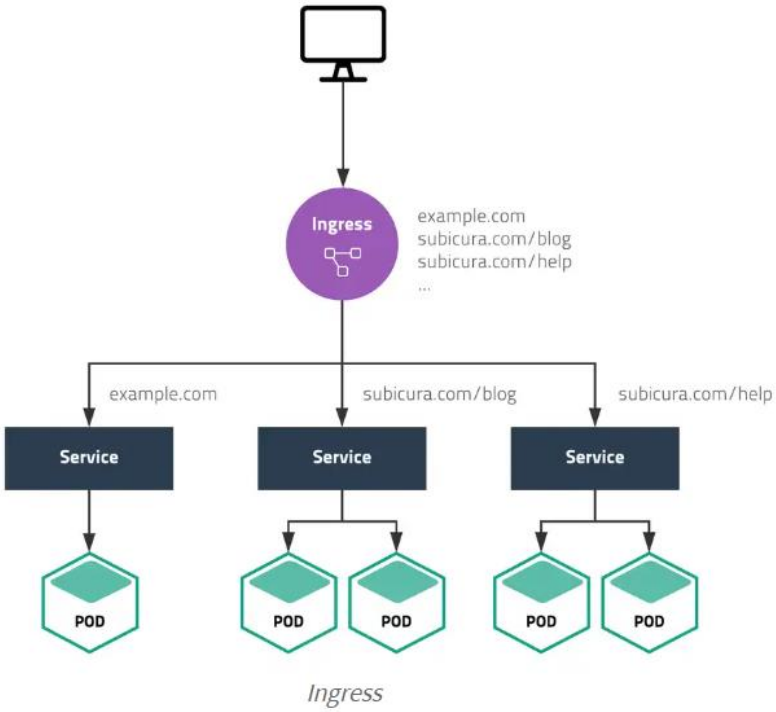



*쿠버네티스 배포 방식*

쿠버네티스는 Deployment, StatefulSets, DaemonSet, Job, CronJob 등 다양한 배포 방식을 지원합니다. Deployment 는 새로운 버전의 애플리케이션을 다양한 전략으로 무중단 배포할 수 있습니다. StatefulSets 은 실행 순서를 보장하고 호스트 이름과 볼륨을 일정하게 사용할 수 있어 순서나 데이터가 중요한 경우에 사용할 수 있습니다. 로그나

모니터링 등 모든 노드에 설치가 필요한 경우엔 DaemonSet 을 이용하고 배치성 작업은 Job 이나 CronJob 을 이용하면 됩니다.

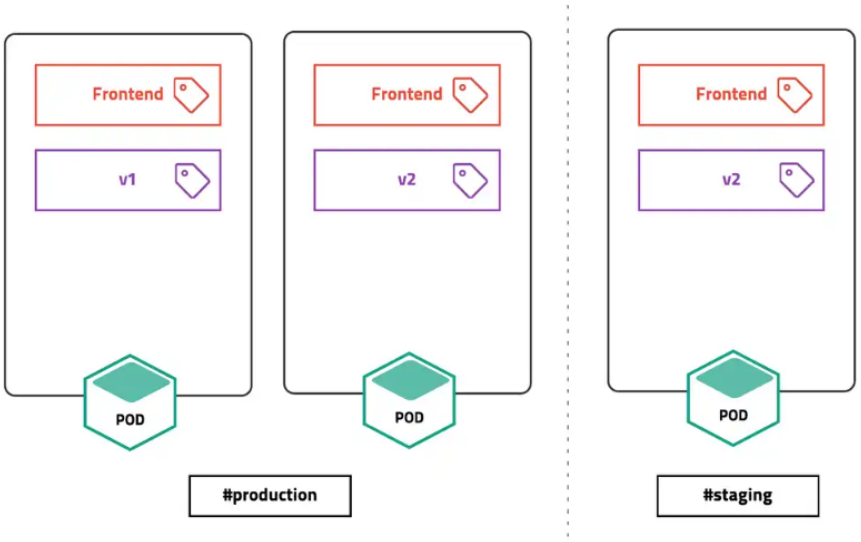
## 2.2) Ingress 설정



*Ingress*

다양한 웹 애플리케이션을 하나의 로드 밸런서로 서비스하기 위해 Ingress 기능을 제공합니다. 웹 애플리케이션을 배포하는 과정을 보면 외부에서 직접 접근할 수 없도록 애플리케이션을 내부망에 설치하고 외부에서 접근이 가능한 ALB 나 Nginx, Apache 를 프록시 서버로 활용합니다. 프록시 서버는 도메인과 Path 조건에 따라 등록된 서버로 요청을 전달하는데 서버가 바뀌거나 IP 가 변경되면 매번 설정을 수정해줘야 합니다.
쿠버네티스의 Ingress 는 이를 자동화하면서 기존 프록시 서버에서 사용하는 설정을 거의 그대로 사용할 수 있습니다. 새로운 도메인을 추가하거나 업로드 용량을 제한하기 위해 일일이 프록시 서버에 접속하여 설정할 필요가 없습니다.
하나의 클러스터에 여러 개의 Ingress 설정을 할 수 있어 관리자 접속용 Ingress 와 일반 접속용 Ingress 를 따로 관리할 수 있습니다.

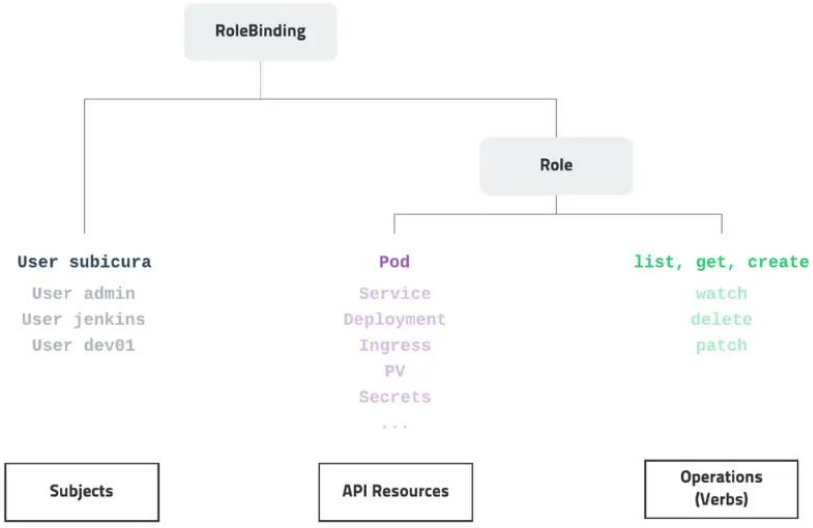
## 2.3) Namespace & Label




*Namespace & Label*

하나의 클러스터를 논리적으로 구분하여 사용할 수 있습니다. 하나의 클러스터에 다양한 프레임워크와 애플리케이션을 설치하기 때문에 기본(system, default)외에 여러 개의 네임스페이스를 사용하는 것이 일반적입니다. 더 세부적인 설정으로 라벨 기능을 적극적으로 사용하여 유연하면서 확장성 있게 리소스를 관리할 수 있습니다.

## 2.4) RBAC (role-based access control)




*Role based access control*

접근 권한 시스템입니다. 각각의 리소스에 대해 유저별로 권한을 손쉽게 지정할 수 있습니다. 클러스터 전체에 적용하거나 특정 네임스페이스에 적용할 수 있습니다.

## 2.5) 인증(Authentication) / 인가(Authorization)

- **Cert-manager**

쿠버네티스 내부에서 HTTPS 통신을 위한 인증서를 행성하고, 인증서의 만료기간이 되면 자동으로 인증서를 갱신해주는 역할을 하는 Certificate manager controller 입니다.

- **Keyclock**

Redhat 에서 개발한 서비스를 대상으로 하는 인증 & 권한 부여 오프소스입니다. SSO, 소셜 로그인, LDAP, RDBMS 등의 User Federation 도 지원합니다.

- **OIDC (OpenID Conccect)**

권한 허가 프로토콜인 0Auth2.0 기술을 이용하여 만들어진 ID 프로토콜입니다. 0Auth2.0 은 Athorization을 포함한다면, OIDC는 Authentication에 집중합니다. 0Atuth를 통해 Access_token을 받을 때 함께 전달받은 id_token을 활용합니다. OIDC는 JWT를 사용한 id_token을 전달해 주므로 해당 token의 위변조 확인 후 관련 정보를 활용합니다.

### 2.6) CRD(Custom Resource Definition)

쿠버네티스가 제공하지 않는 기능을 기본 기능과 동일한 방식으로 적용하고 사용할 수 있습니다. 예를 들어, 쿠버네티스는 기본적으로 SSL 인증서 관리 기능을 제공하지 않지만, cert-manager를 설치하고 Certificate 리소스를 이용하면 익숙한 쿠버네티스 명령어로 인증서를 관리할 수 있습니다. 또 다른 도구, 방식을 익힐 필요 없이 다양한 기능을 손쉽게 확장할 수 있습니다.

### 2.7) Auto Scaling

CPU, memory 사용량에 따른 확장은 기본이고 현재 접속자 수와 같은 값을 사용할 수도 있습니다. 컨테이너의 개수를 조정하는 Horizontal Pod Autoscaler(HPA), 컨테이너의 리소스 할당량을 조정하는 Vertical Pod Autoscaler(VPA), 서버 개수를 조정하는 Cluster Autosclaer(CA) 방식이 있습니다.

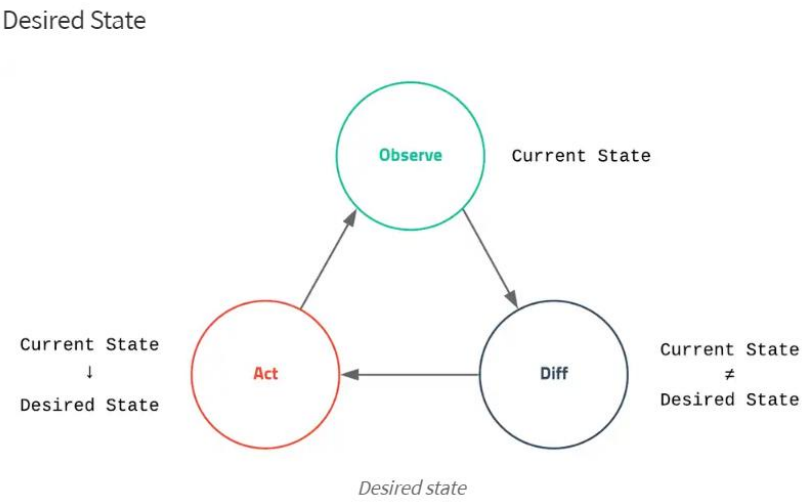
### 2.8) Federation, Multi Cluster

클라우드에 설치한 쿠버네티스 클러스터와 자체 서버에 설치한 쿠버네티스를 묶어서 하나로 사용할 수 있습니다.

## 3. Kubernetes 기본 개념

### 3.1) Desired State

쿠버네티스의 디자인과 구성 요소, 각각의 동작 방식입니다.

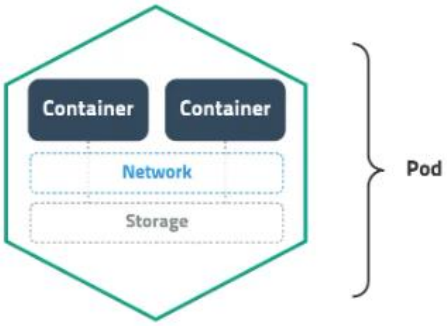



쿠버네티스에서 가장 중요한 것은 **desired state - 원하는 상태** 라는 개념입니다. 원하는 상태라 함은 관리자가 바라는 환경을 의미하고 좀 더 구체적으로는 얼마나 많은 웹서버가 떠 있으면 좋은지, 몇 번 포트로 서비스하기를 원하는지 등을 말합니다. 쿠버네티스는 복잡하고 다양한 작업을 하지만 자세히 들여다보면 **현재 상태** current state 를 모니터링하면서 관리자가 설정한 **원하는 상태**를 유지하려고 내부적으로 이런저런 작업을 하는 로직을 가지고 있습니다.

### 3.2) Kubernetes Object

쿠버네티스는 상태를 관리하기 위한 대상을 오브젝트로 정의합니다. 기본으로 수십 가지 오브젝트를 제공하고 새로운 오브젝트를 추가하기가 매우 쉽기 때문에 확장성이 좋습니다. 여러 오브젝트 중 주요 오브젝트는 다음과 같습니다.

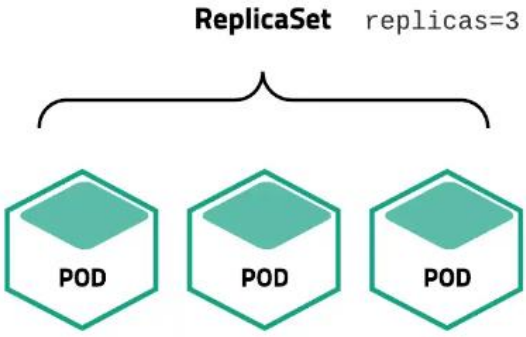
- **Pod**




*Pod*

쿠버네티스에서 배포할 수 있는 가장 작은 단위로 한 개 이상의 컨테이너와 스토리지, 네트워크 속성을 가집니다. Pod에 속한 컨테이너는 스토리지와 네트워크를 공유하고 서로 localhost로 접근할 수 있습니다. 컨테이너를 하나만 사용하는 경우도 반드시 Pod으로 감싸서 관리합니다.

- **ReplicaSet**




*ReplicaSet*

Pod을 여러 개(한 개 이상) 복제하여 관리하는 오브젝트입니다. Pod을 생성하고 개수를 유지하려면 반드시 ReplicaSet을 사용해야 합니다. ReplicaSet은 복제할 개수, 개수를 체크할 라벨 선택자, 생성할 Pod의 설정값(템플릿)등을 가지고 있습니다. 직접적으로 ReplicaSet을 사용하기보다는 Deployment등 다른 오브젝트에 의해서 사용되는 경우가 많습니다.

- **Service**

네트워크와 관련된 오브젝트입니다. Pod을 외부 네트워크와 연결해주고 여러 개의 Pod을 바라보는 내부 로드 밸런서를 생성할 때 사용합니다. 내부 DNS에 서비스 이름을 도메인으로 등록하기 때문에 서비스 디스커버리 역할도 합니다.

- **Volume**

저장소와 관련된 오브젝트입니다. 호스트 디렉토리를 그대로 사용할 수도 있고 EBS 같은 스토리지를 동적으로 생성하여 사용할 수도 있습니다. 사실상 인기 있는 대부분의 저장 방식을 지원합니다.

- **Object Spec -YAML**

```yaml
apiVersion: v1
kind: Pod
metadata:
    name: example
spec:
    containers:
    - name: busybox
        image: busybox:1.25
```

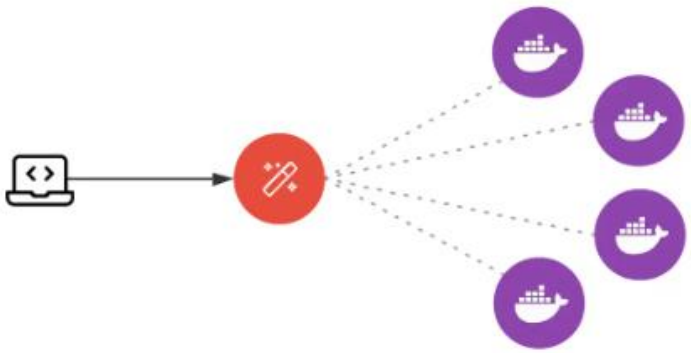
오브젝트의 명세Spec는 YAML 파일로 정의하고 여기에 오브젝트의 종류와 원하는 상태를 입력합니다. 이러한 명세는 생성, 조회, 삭제로 관리할 수 있기 때문에 REST API로 쉽게 노출할 수 있습니다. 접근 권한 설정도 같은 개념을 적용하여 누가 어떤 오브젝트에 어떤 요청을 할 수 있는지 정의할 수 있습니다.

### 3.3) **Kubernetes 배포 방식**

쿠버네티스는 애플리케이션을 배포하기 위해 원하는 상태(desired state)를 다양한 오브젝트(object)에 라벨 Label 을 붙여 정의(yaml)하고 API 서버에 전달하는 방식을 사용합니다.
"컨테이너를 2 개 배포하고 80 포트로 오픈해줘" 라는 간단한 작업을 위해 다음과 같은 구체적인 명령을 전달해야 합니다.

## 4. Kubernetes Architecture

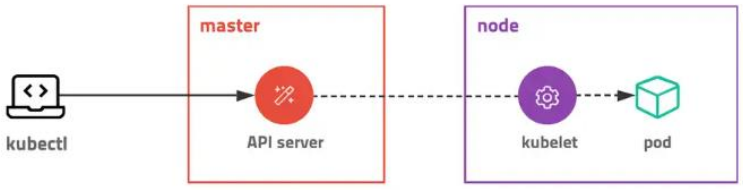
컨테이너는 아주 심플하고 우아하게 동작합니다. run을 하면 실행되고 stop을 하면 멈춥니다. 서버-클라이언트 구조를 안다면 컨테이너를 관리하는 에이전트를 만들고 중앙에서 API를 이용하여 원격으로 관리하는 모습을 쉽게 그려볼 수 있습니다.



*Server - Agent*

쿠버네티스 또한 중앙(Master)에 API 서버와 상태 저장소를 두고 각 서버(Node)의 에이전트(kubelet)와 통신하는 구조입니다.

### 4.1) 마스터 – 워커 노드 구조




*Master - Node*

쿠버네티스는 전체 클러스터를 관리하는 **마스터**와 컨테이너가 배포되는 **워커 노드**로 구성되어 있습니다. 모든 명령은 마스터의 API 서버를 호출하고 워커 노드는 마스터와 통신하면서 필요한 작업을 수행합니다. 특정 워커 노드의 컨테이너에 명령하거나 로그를 조회할 때도 워커 노드에 직접 명령하는 게 아니라 마스터에 명령을 내리고 마스터가 워커 노드에 접속하여 대신 결과를 응답합니다.

- **Master**

마스터 서버는 다양한 모듈이 확장성을 고려하여 기능별로 쪼개져 있는 것이 특징입니

다. 관리자만 접속할 수 있도록 보안 설정을 해야 하고 마스터 서버가 죽으면 클러스터를 관리할 수 없기 때문에 보통 3대를 구성하여 안정성을 높입니다. AWS EKS 같은 경우 마스터를 AWS에서 자체 관리하여 안정성을 높였고(마스터에 접속 불가) 개발 환경이나 소규모 환경에선 마스터와 워커 노드를 분리하지 않고 같은 서버에 구성하기도 합니다.

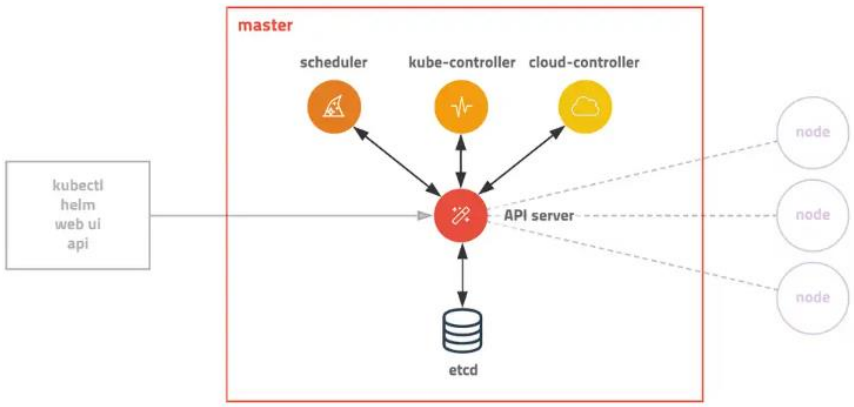
- **Worker Node**

워커 노드 서버는 마스터 서버와 통신하면서 필요한 Pod을 생성하고 네트워크와 볼륨을 설정합니다. 실제 컨테이너들이 생성되는 곳으로 수백, 수천대로 확장할 수 있습니다. 각각의 서버에 라벨을 붙여 사용목적을 정의할 수 있습니다.

- **Kubectl**

Kubectl 아웃클러스터 모드에서 사용하는 명령어입니다. Kubeconfig 파일을 물고 통신합니다.

### 4.2) Master 구성 요소




*Master Component*

- **API 서버 kube-apiserver**

API 서버는 모든 요청을 처리하는 마스터의 핵심 모듈입니다. kubectl의 요청 뿐 아니라 내부 모듈의 요청도 처리하며 권한을 체크하여 요청을 거부할 수 있습니다. 실제로 하는 일은 원하는 상태를 key-value 저장소에 저장하고 저장된 상태를 조회하는 매우 단순한 작업입니다. Pod을 노드에 할당하고 상태를 체크하는 일은 다른 모듈로 분리되어 있습니

다. 노드에서 실행 중인 컨테이너의 로그를 보여주고 명령을 보내는 등 디버거 역할도 수행합니다.

- **분산 데이터 저장소 etcd**

RAFT 알고리즘을 이용한 key-value 저장소입니다. 여러 개로 분산하여 복제할 수 있기 때문에 안정성이 높고 속도도 빠른 편입니다. 단순히 값을 저장하고 읽는 기능 뿐 아니라 watch 기능이 있어 어떤 상태가 변경되면 바로 체크하여 로직을 실행할 수 있습니다.
클러스터의 모든 설정, 상태 데이터는 여기 저장되기 때문에 etcd 만 잘 백업해두면 언제든지 클러스터를 복구할 수 있습니다. etcd 는 오직 API 서버와 통신하고 다른 모듈은 API 서버를 거쳐 etcd 데이터에 접근합니다.

- **스케줄러 kube-scheduler**

스케줄러는 할당되지 않은 Pod을 여러 가지 조건(필요한 자원, 라벨)에 따라 적절한 노드 서버에 할당해주는 모듈입니다.

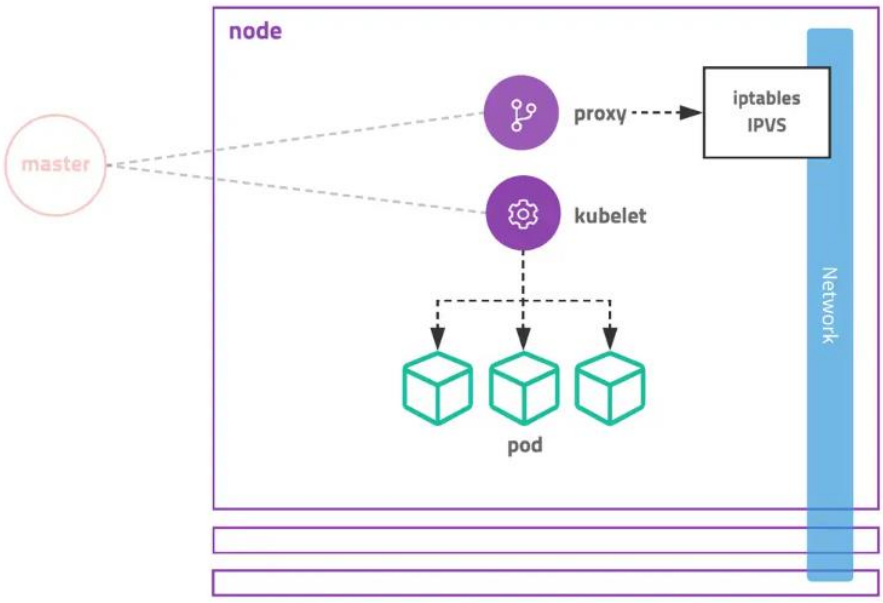
- **큐브 컨트롤러 kube-controller-manager**

큐브 컨트롤러는 다양한 역할을 하는 아주 바쁜 모듈입니다. 쿠버네티스에 있는 거의 모든 오브젝트의 상태를 관리합니다. 오브젝트별로 철저하게 분업화되어 Deployment는 ReplicaSet을 생성하고 ReplicaSet은 Pod을 생성하고 Pod은 스케줄러가 관리하는 식입니다.

- **클라우드 컨트롤러 cloud-controller-manager**

클라우드 컨트롤러는 AWS, GCE, Azure 등 클라우드에 특화된 모듈입니다. 노드를 추가/삭제하고 로드 밸런서를 연결하거나 볼륨을 붙일 수 있습니다. 각 클라우드 업체에서 인터페이스에 맞춰 구현하면 되기 때문에 확장성이 좋고 많은 곳에서 자체 모듈을 만들어 제공하고 있습니다.

### 4.3) Worker-node 구성요소




*Node Component*

- **큐블릿 kubelet**

노드에 할당된 Pod의 생명주기를 관리합니다. Pod을 생성하고 Pod 안의 컨테이너에 이상이 없는지 확인하면서 주기적으로 마스터에 상태를 전달합니다. API 서버의 요청을 받아 컨테이너의 로그를 전달하거나 특정 명령을 대신 수행하기도 합니다.

- **프록시 kube-proxy**

큐블릿이 Pod을 관리한다면 프록시는 Pod으로 연결되는 네트워크를 관리합니다. TCP, UDP, SCTP 스트림을 포워딩하고 여러 개의 Pod을 라운드로빈 형태로 묶어 서비스를 제공할 수 있습니다. 초기에는 kube-proxy 자체가 프록시 서버로 동작하면서 실제 요청을 프록시 서버가 받고 각 Pod에 전달해 주었는데 시간이 지나면서 iptables를 설정하는 방식으로 변경되었습니다. iptables에 등록된 규칙이 많아지면 느려지는 문제 때문에 최근 IPVS를 지원하기 시작했습니다.

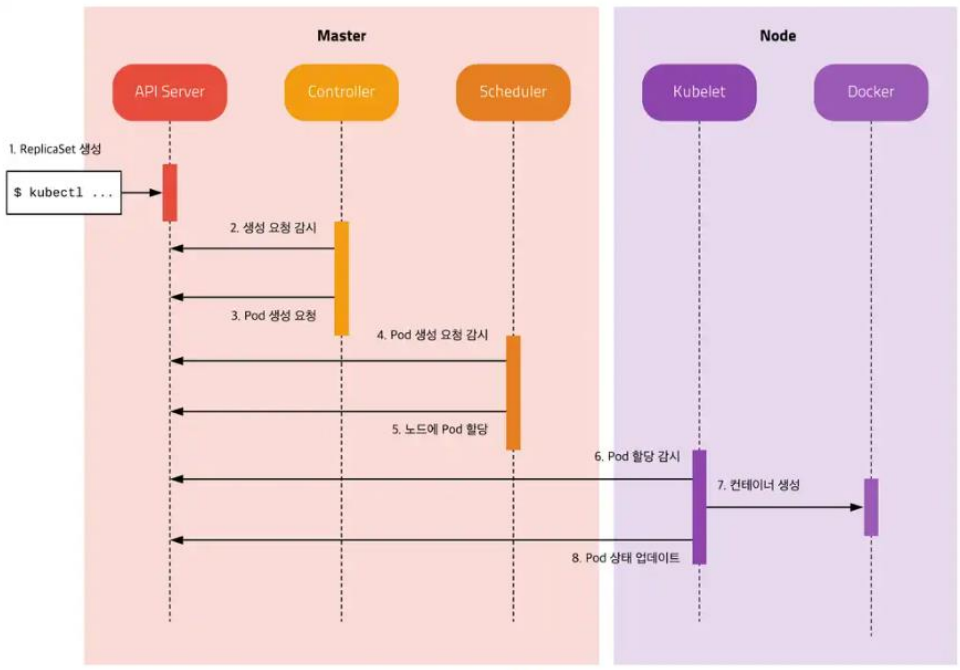
- **추상화**

쿠버네티스는 CRI(Container runtime interface)를 구현한 다양한 컨테이너 런타임을 지원합니다.

Api 서버를 통해서 모든 컴포넌트(리소스)가 통신하는데 Kubernetes 네트워크를 설정하는 방법론인 CNI(Container Network Interface)가 있습니다.

## 5. 하나의 Pod가 생성되는 과정

관리자가 애플리케이션을 배포하기 위해 ReplicaSet을 생성하면 다음과 같은 과정을 거쳐 Pod을 생성합니다.




*ReplicaSet 만들기*

흐름을 보면 각 모듈은 서로 통신하지 않고 오직 API Server와 통신하는 것을 알 수 있습니다. API Server를 통해 etcd에 저장된 상태를 체크하고 현재 상태와 원하는 상태가 다르면 필요한 작업을 수행합니다. 각 모듈이 하는 일을 보면 다음과 같습니다.

**Kube Controller**

- Kube Controller 에 포함된 ReplicaSet Controller 가 ReplicaSet 을 감시하다가 ReplicaSet 에 정의된 Label Selector 조건을 만족하는 Pod 이 존재하는지 체크

- 해당하는 Label 의 Pod 이 없으면 ReplicaSet 의 Pod 템플릿을 보고 새로운 Pod(no assign)을 생성. 생성은 역시 API Server 에 전달하고 API Server 는 etcd 에 저장

**Scheduler**

- Scheduler 는 할당되지 않은(no assign) Pod 이 있는지 체크
- 할당되지 않은 Pod 이 있으면 조건에 맞는 Node 를 찾아 해당 Pod 을 할당

**Kubelet**

- Kubelet 은 자신의 Node 에 할당되었지만 아직 생성되지 않은 Pod 이 있는지 체크
- 생성되지 않은 Pod 이 있으면 명세를 보고 Pod 을 생성
- Pod 의 상태를 주기적으로 API Server 에 전달

모든 노드에 Pod을 배포하는 DaemonSet도 동일한 방식으로 동작합니다. DaemonSet controller와 Scheduler가 전체 노드에 대해 Pod을 할당하면 kubelet이 자기 노드에 할당된 Pod을 생성하는 식입니다.

**Reference**


1. KubernetesArchitecture – '시작하세요!도커/쿠버네티스' 책 내용 공부하여 정리

2. KubernetesArchitecture – '쿠버네티스 입문' 책 내용 공부하여 정리

3. KubernetesArchitecture - https://subicura.com/2019/05/19/kubernetes-basic-1.html

4. Storage - https://kubernetes.io/docs/concepts/storage/#types-of-volumes

5. 인증/인가 - https://ooeunz.tistory.com/143

6. 인증/인가 - https://developers.google.com/identity/protocols/oauth2/openid-connect