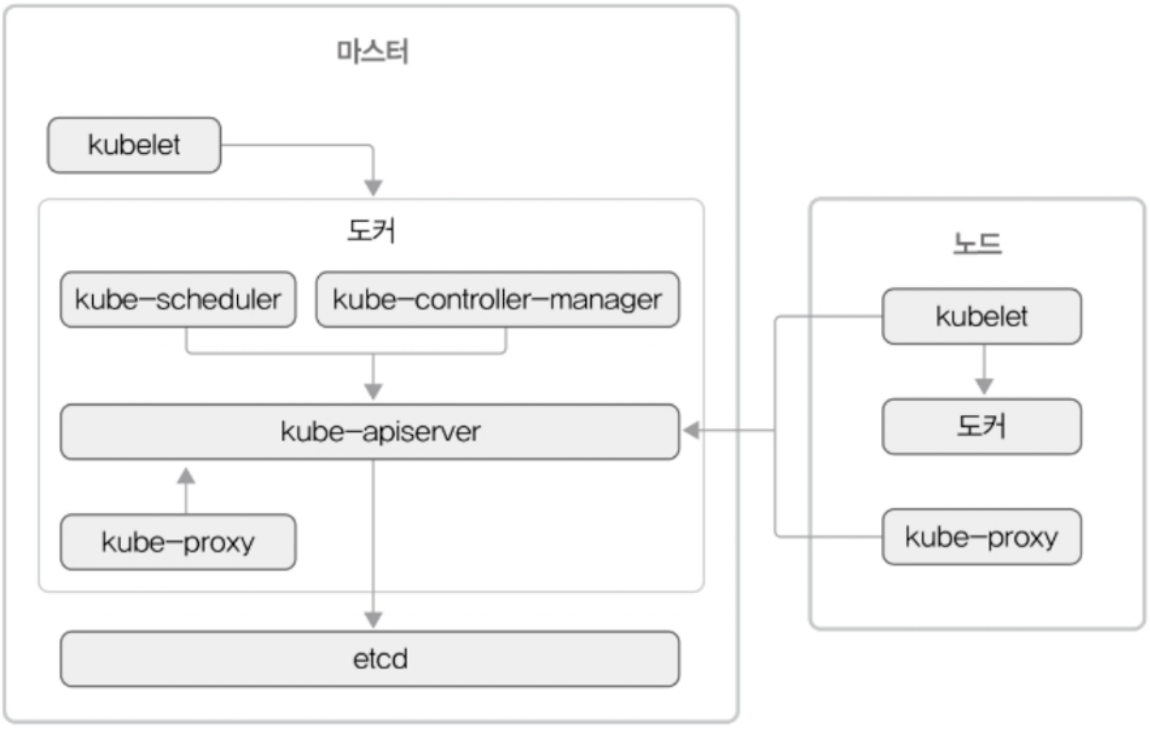
쿠버네티스란

- 쿠버네티스는 컨테이너 오케스트레이션을 위한 솔루션
  - 오케스트레이션(Orchestration)은 복잡한 단계를 관리하고 요소들의 유기적인 관계를 미리 정의해 손쉽게 사용하도록 서비스를 제공하는 것
  - 도커 스윔, 메소스, 노매드, 쿠버네티스등이 오케스트레이션 솔루션

마스터와 워커 노드 의 구조



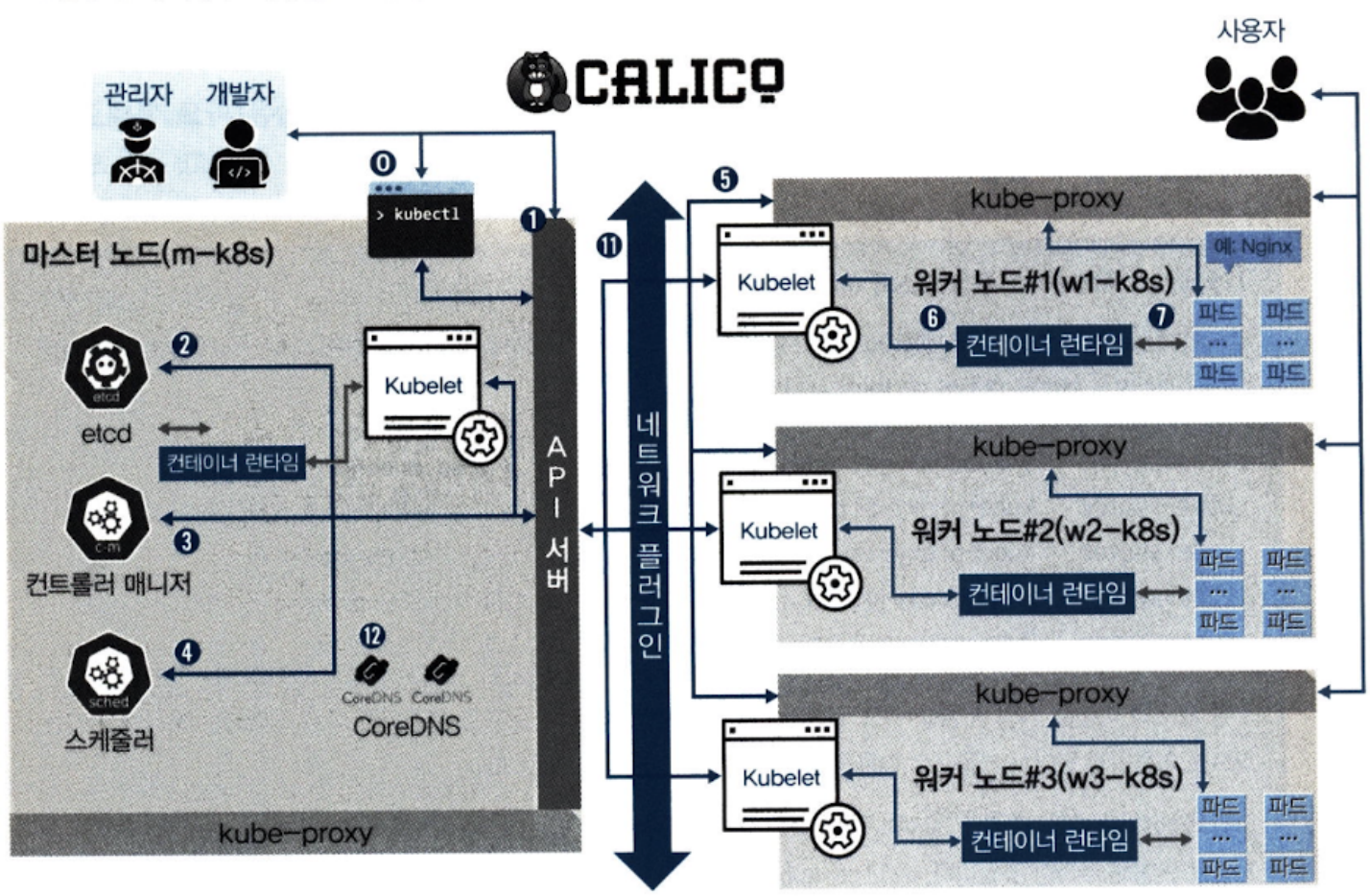
- kube-apiserver가 컴포넌트의 중심
- k8s의 모든 통신은 kube-apiserver를 통해서 다른 컴포넌트와 필요한 정보를 주고 받음
- kubelet은 마스터의 kube-apisever와 통신하면서 파드의 생성, 관리, 삭제를 담당
- etcd에는 kube-apiserver만 접근할 수 있음
- k8s 관리용 컴포넌트 kube-scheduler, kube-controller-manager, kube-apisever, kube-proxy
  - 초기 k8s는 관리용 컴포넌트들은 컨테이너가 아니라 직접 서버 프로세스로 실행했지만 최근에는 컨테이너로 변경
- etcd는 컨테이너가 아니라 별도의 프로세스로 설정

(2) 구성요소

- kubectl, kubelet, API 서버, 캘리코 등은 모두 쿠버네티스 클러스터를 이루는 구성 요소
- etcd, 컨트롤러 매니저, 스케줄러, kube-proxy, 컨테이너 런타임, 파드

```
[k8s@m-k8s ~]$ kubectl get pods --all-namespaces
NAMESPACE     NAME                                       READY   STATUS    RESTARTS
AGE
kube-system   calico-kube-controllers-99c9b6f64-dnlr4    1/1     Running   0
95m
kube-system   calico-node-2c9zh                          1/1     Running   0
95m
kube-system   calico-node-gf9vp                          1/1     Running   0
93m
kube-system   calico-node-xxb77                          1/1     Running   0
94m
kube-system   coredns-66bff467f8-672km                   1/1     Running   0
96m
kube-system   coredns-66bff467f8-bfsnp                   1/1     Running   0
96m
kube-system   etcd-m-k8s                                 1/1     Running   0
96m
kube-system   kube-apiserver-m-k8s                       1/1     Running   0
96m
kube-system   kube-controller-manager-m-k8s              1/1     Running   0
96m
kube-system   kube-proxy-djbw9                           1/1     Running   0
94m
kube-system   kube-proxy-l2qvt                           1/1     Running   0
93m
kube-system   kube-proxy-qg6lf                           1/1     Running   0
96m
kube-system   kube-scheduler-m-k8s                       1/1     Running   0
96m
```

- --all-namespaces는 기본 네임스페이스인 default 외에 모든 것을 표시하겠다는 의미



(2.1) 마스터 노드 컴포넌트

- control-plane이라고 부름
- etcd, kube-apiserver, kube-scheduler, kube-controller-manager, cloud-controller-manager등이 마스터용 컴포넌트

etcd

- etcd는 코어os에서 개발한 고가용성을 제공하는 key-value 저장소
- k8s에서는 필요한 모든 데이터를 저장하는 데이터베이스 역할을 담당하며 구성 요소들의 상태 값이 모두 저장되는 곳
- etcd는 서버 하나당 프로세스 1개만 사용할 수 있으며 보통 etcd 자체를 클러스터링한 후 여러 개 마스터 서버에 분산해서 실행해 데이터의 안정성을 보장하도록 구성
  - 마스터 서버는 기본 3개 정도 설정(수업에는 리소스 문제로 1개만)
  - 여러 마스터 서버에 복제해 두면 하나의 etcd에서 장애가 나더라도 시스템의 가용성을 확복할 수 있음
- etcd는 리눅스의 구성 정보를 주로 가지고 있는 etc 디렉터리와 distributed의 합성어. 따라서 etcd는 구성 정보를 퍼뜨려 저장하겠다는 의미
- 회사의 관리자가 모든 보고 내용을 기록하는 장부라고 보면 됨
- 실제로 etcd 외의 다른 구성 요소는 상태 값을 관리하지 않음. 그러므로 etcd의 정보만 백업돼 있다면 긴급한 장애 상황에서도 쿠버네티스 클러스터는 복구할 수 있음

kube-apiserver

- 쿠버네티스 클러스터의 중심 역할을 하는 통로
- 클러스터의 api를 사용할 수 있도록 하는 컴포넌트
- 클러스터로 온 요청이 유효한지 검증
  - 예) 클러스터의 특정 네임스페이스에 존재하는 디폴로이먼트 목록 조회 요청을 받으면, 이 요청에 사용된 토큰이 해당 네임스페이스와 자원을 대상으로 실행할 권한이 있는지 검사하고 권한이 있다면 디폴로이먼트 목록을 조회하여 되돌려 줌
- k8s는 MSA(Micro Service Architecture)이므로 서로 분리된 컴포넌트 여러 개로 구성되어 있음
- k8s에 보내는 모든 요청은 kube-apiserver를 이용해서 다른 컴포넌트로 전달
- 수평적으로 확장할 수 있도록 설계하여 여러 대에 서버에 kube-apiserver를 실행할 수 있도록 함
- 주로 상태 값을 저장하는 etcd와 통신하지만, 그 밖의 요소들 또한 API 서버를 중심에 두고 통신하므로 API 서버의 역할이 매우 중요
- 회사에 비유하면 모든 직원과 상황을 관리하고 목표를 설정하는 관리자에 해당

kube-scheduler

- 노드의 상태와 자원, 레이블, 요구 조건 등을 고려해 파드를 어떤 워커 노드에 생성할 것인지를 결정하고 할당
  - 현재 클러스터 안에서 자원 할당이 가능한 노드 중 알맞은 노드를 선택해서 새롭게 만든 파드를 실행
- 스케줄러라는 이름에 걸맞게 파드를 조건에 맞는 워커 노드에 지정하고, 파드가 워커 노드에 할당되는 일정을 관리하는 역할을 담당
- 파드는 처음 실행할 때 여러 가지 조건을 설정하여, kube-scheduler가 조건에 맞는 노도를 찾음
  - 조건에는 아래와 같은 내용이 있음
  - 하드웨어 요구사항
  - 함께 있어야 하는 파드들을 같은 노드에 실행하는 어피니티(affinity)와 파드를 다양한 노드로 분산해서 실행하는 안티 어피니티(anti-affinity) 만족 여부
  - 특정 데이터가 있는 노드에 할당

kube-controller-manager

- 클러스터의 오브젝트 상태 관리 및 파드들을 관리하는 컨트롤러(controller)
  - 예를 들어 워커 노드에서 통신이 되지 않는 경우, 상태 체크와 복구는 컨트롤러 매니저에 속한 노드 컨트롤러에서 이루어짐
- 컨트롤러 각각은 논리적으로 개별 프로세스지만 복잡도를 줄이려고 모든 컨트롤러를 바이너리 파일 하나로 컴파일해 단일 프로세스로 실행
- kube-controller-manager는 컨트롤러 각각을 실행하는 컴포넌트
- 서비스와 파드를 연결하는 역할을 하는 엔드 포인트 컨트롤러 또한 컨트롤러 매니저
- 레플리카셋 컨트롤러는 레플리카셋에 요청받은 파드 개수대로 파드를 생성
- 이와 같이 다양한 상태 값을 관리하는 주체들이 컨트롤러 매니저에 소속돼 각자의 역할을 수행

(2.2) 노드용 컴포넌트

- 노드용 컴포넌트는 k8s 실행 환경을 관리
- 대표적으로 각 노드의 파드 실행을 관리

- 컴포넌트에는 Kubelet, kube-proxy, 컨테이너 런타임

kubelet

- 클러스터 안 모든 노드에서 실행되는 에이전트
- 파드의 구성 내용(PodSpec)을 받아서 컨테이너 런타임으로 전달하고, 파드 안의 컨테이너들이 정상적으로 작동하는지 모니터링
  - PodSpec에는 조건이 파드 생성 조건이 담김
- 노드 안에 있는 컨테이너라도 k8s가 만들지 않은 컨테이너는 관리하지 않음

kube-proxy

- k8s는 클러스터 안에 별도의 가상 네트워크를 설정하고 관리
- kube-proxy는 이런 가상 네트워크의 동작을 관리하는 컴포넌트
- 호스트의 네트워크 규칙을 관리하거나 연결을 전달할 수도 있음

컨테이너 런타임(CRI, Container Runtime Interface)

- 파드를 이루는 컨테이너의 실행을 담당
- 파드 안에서 다양한 종류의 컨테이너가 문제 없이 작동하게 만드는 표준 인터페이스
- 가장 많이 알려진 런타임으로 도커, containerd, runc 등이 존재
- 보통 컨테이너 표준을 정하는 OCI(Open Container Initiative)의 런타임 규격을 구현되 있거나 컨테이너 런타임이라면 k8s에 사용가능

Pod

- 한 개 이상의 컨테이너로 단일 목적의 일을 하기 위해서 모인 단위
- 만약 웹 서버 역할을 할 수도 있고, 로그나 데이터를 분석할 수도 있음
- Pod(파드)는 언제라도 죽을 수 있는 존재라는 것
- 파드는 언제라도 죽을 수 있다고 가정하고 설계됐기 때문에 k8s는 여러 대안을 디자인함

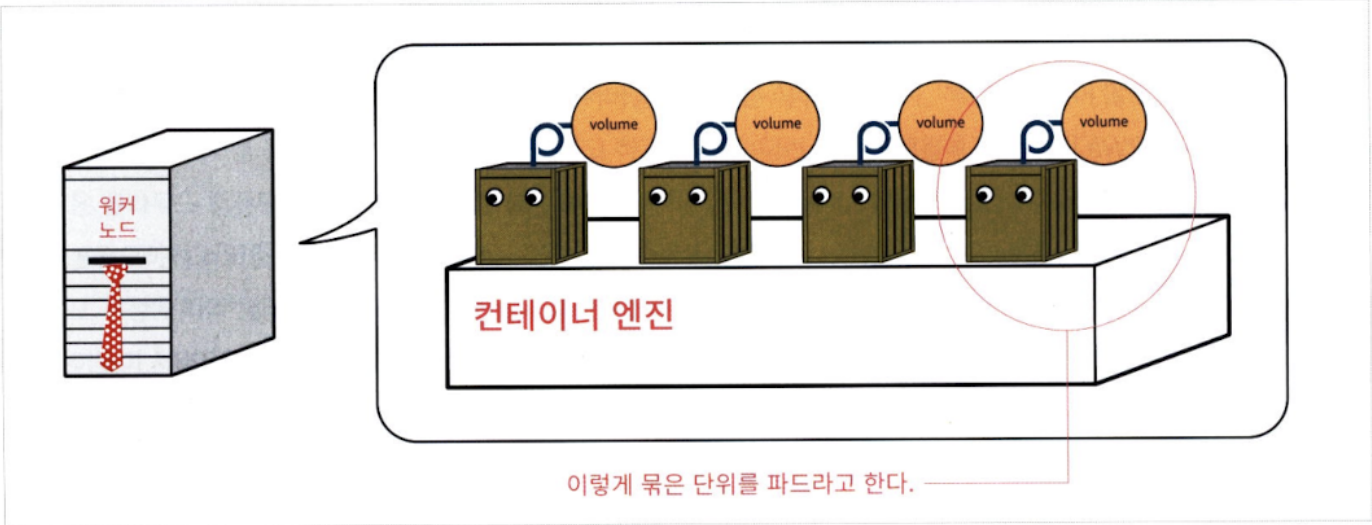


그림 8-3-1 파드는 컨테이너와 볼륨을 묶은 것이다.

(2.3) 애드온

- 클러스터 안에서 필요한 기능을 실행하는 파드
- 네임스페이스는 kube-system이며 애드온으로 사용하는 파드들은 디폴리이먼트, 리플리케이션 컨트롤러 등으로 관리

DNS 애드온

- DNS 애드온은 클러스터 안에서 동작하는 DNS 서버
- 빠르고 유연한 DNS 서버
- k8s 서비스에 DNS 레코드를 제공
- k8s 안에 실행된 컨테이너들은 자동으로 DNS 서버에 등록
- 클러스트에서 도메인 이름을 이용해 통신하는 데 사용
- kube-dns, CoreDNS가 주로 사용됨
- 실무에서 쿠버네티스 클러스터를 구성하여 사용할 때는 IP보다 도메인 네임을 편리하게 관리해 주는 CoreDNS를 사용하는 것이 일반적

(3) 오브젝트와 컨트롤러

- k8s는 크게 오브젝트와 오브젝트를 관리하는 컨트롤러 구분
- 오브젝트에는 pod, service, volume, namespace등이 있음
- 컨트롤러에는 Replicaset, Deployment, StatefulSet, DaemonSet, Job 등이 있음

(3.1) 네임스페이스

- 네임스페이스는 k8s 클러스터 하나를 여러 개 논리적인 단위로 구분해서 사용하는 것
- 네임스페이스 덕분에 하나의 클러스터를 여러 개 팀이나 사용자가 함께 공유할 수 있음
- 클러스터 안에서 용도에 따라 실행해야 하는 앱을 구분할 때도 네임스페이스를 사용
- 네임스페이스별로 별도이 쿼터를 설정해서 특정 네임스페이스의 사용량을 제한할 수 있음
- k8s를 처음 설치하면 기본적으로 몇 개의 네임스페이스가 생성

네트워크 플러그인

- 쿠버네티스 클러스터의 통신을 위해서 네트워크 플러그인을 선택하고 구성해야 함
- 네트워크 플러그인은 일반적으로 CNI로 구성하는데, 주로 많이 사용하는 CNI에는 아래와 같음
  - Calico
  - Flannel
  - Cilium
  - Kube-router
  - Romana
  - WeaveNet

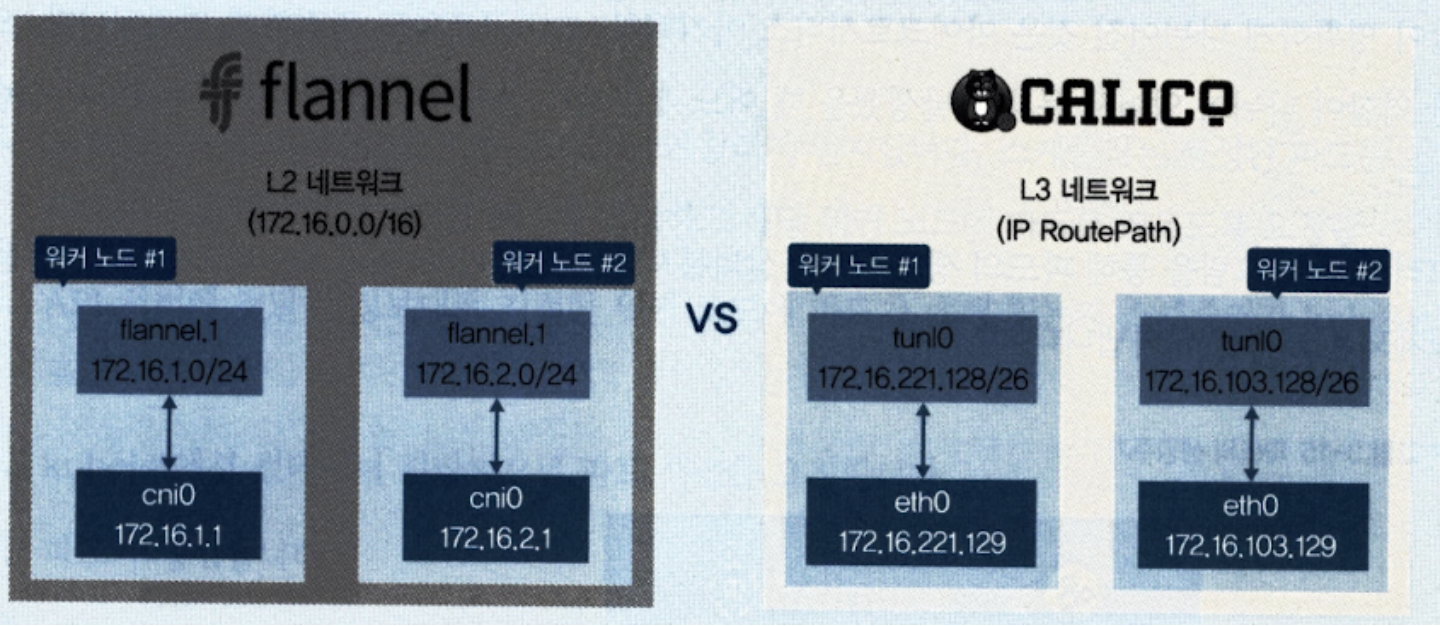
- Canal

CoreDNS

- k8s 클러스터에서 도메인 이름을 이용해 통신하는데 사용
- IP보다 도메인 네임을 편리하게 관리해 주는 CoreDNS를 사용하는 것이 일반적임

CNI

- CNI(Container Network Interface)는 클라우드 네이티브 컴퓨팅 재단의 프로젝트로, 컨테이너의 네트워크 안정성과 확장성을 보장하기 위해서 개발
- CNI에 사용할 수 있는 네트워크 플러그인은 다양한데, 구성 방식과 지원하는 기능, 성능이 각기 다르므로 사용 목적에 맞게 선택하면 됨
  - Calico는 L3로 컨테이너 네트워크를 구성
  - Flannel은 L2로 컨테이너 네트워크를 구성



사용자가 배포된 파드에 접속할 때

- kube-proxy
  - 쿠버네티스 클러스터는 파드가 위치한 노드에 kube-proxy를 통해 파드가 통신할 수 있는 네트워크를 설정
  - 이때 실제 통신은 br_netfilter와 iptables로 관리
  - 이 두 기능은 서버세팅할 때 사용했음
- 파드
  - 이미 배포된 파드에 접속하고 필요한 내용을 전달받음
  - 이 때 대부분 사용자는 파드가 어느 워커 노드에 위치하는지 신경 쓰지 않아도 됨

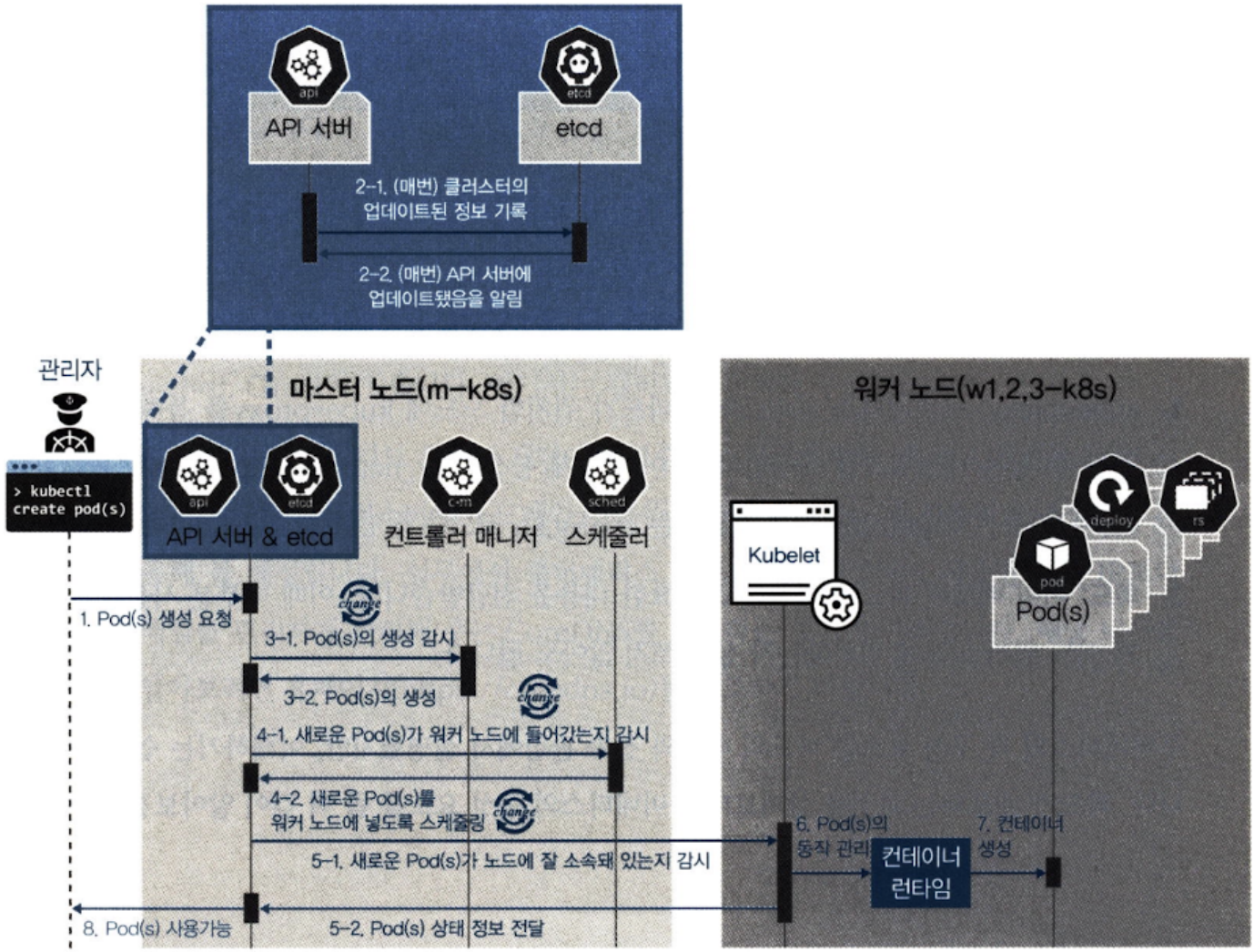
파드의 생명주기

- 쿠버네티스의 가장 큰 장점은 쿠버네티스의 구성 요소마다 하는 일이 명확하게 구분돼 각자의 역할만 충실하게 수행하면 클러스터 시스템이 안정적으로 운영된다는 점

- 이렇게 각자의 역할이 명확하게 나눠어진 것은 마이크로서비스 아키텍처(MSA) 구조와도 밀접하게 연관됨
- 또한 역할이 나뉘어 있어서 문제가 발생했을 때 어느 부분에서 문제가 발생했는지 디버깅하기 쉬움

생명주기

- 생성, 수정, 삭제되는 과정을 거침



1. kubectl을 통해 API 서버에 파드 생성 요청
2. API 서버에 전달된 내용이 있으면 API 서버는 etcd에 전달된 내용을 모두 기록해 클러스터의 상태 값을 최신으로 유지
    1. 따라서 각 요소가 상태를 업데이트할 때마다 모두 API 서버를 통해 etcd에 기록
3. API 서버에 파드 생성이 요청된 것을 컨트롤러 매니저가 인지하면 컨트롤러 매니저는 파드를 생성하고, 이 상태를 API 서버에 전달
    1. 아직 어떤 워커 노드에 파드를 적용할지는 결정되지 않은 상태
4. API 서버에 파드가 생성됐다는 정보를 스케줄러가 인지
    1. 스케줄러는 생성된 파드를 어떤 워커 노드에 적용할 지 조건을 고려해 결정하고 해당 워커 노드에 파드를 띄우도록 요청
5. API 서버에 전달된 정보대로 지정한 워커 노드에 파드가 속해 있는지 스케줄러가 kubelet으로 확인
6. kubelet에서 컨테이너 런타임으로 파드 생성 요청
7. 파드 생성 및 사용 가능한 상태로 변경

kubelet

- kubelet은 쿠버네티스에서 파드의 생성과 상태 관리 및 복구 등을 담당하는 매우 중요한 구성 요소
- 따라서 kubelet에 문제가 생기면 파드가 정상적으로 관리되지 않음

kubelet 테스트

1. 파드를 배포

```
kubectl create -f ./nginx-pod.xml
```

2. kubectl get pod 명령으로 배포된 파드가 정상적으로 배포된 상태(Running)인지 확인

```
kubectl get pod
```

3. kubectl get pods -o wide 명령을 실행해 파드가 배포된 워커 노드를 확인

```
kubectl get pods -o wide
```

4. 배포된 노드를 접속하여 systemctl stop kubelet으로 kubelet 서비스를 멈춤

```
sudo systemctl stop kubelet
```

5. master 서버에서 아래 명령어로 상태를 확인하고 delete 명령어를 입력하여 파드를 삭제 (삭제하는데 시간이 오래 걸림)

```
kubectl get pod
kubectl delete pod ngix-pod
```

6. 너무 시간이 오래 걸리면 강제 중지를 진행
7. 다시 kubectl get pod 명령을 실행해 파드의 상태를 확인
8. 해당 시스템에서 kubelet을 복구

```
sudo systemctl start kubelet
```

9 몇 분 뒤에 master 서버에서 kubelet get pod 명령을 실행해 nginx-pod가 삭제됐는지 확인

kube-proxy

- kube-proxy는 파드의 통신을 담당
- br_netfilter 커널 모듈을 적재하고 iptables를 거쳐 통신하도록 설정

kubectl delete pod nginx-pod -n default

YAML 파일

- 쿠버네티스는 YAML 파일로 컨테이너 리소스를 생성하거나 삭제할 수 있음
- 쿠버네티스에서 YAML 파일의 용도는 컨테이너뿐만 아니라 거의 모든 리소스 오브젝트들에 사용될 수 있다는 것이 가장 큰 특징
- 컨테이너는 물론, 설정값(configMap), Secrets 등 모두 YAML 파일로 정의해 사용함
- 쿠버네티스에서 실제로 서비스를 배포할 때에도 kubectl 명령어가 아닌 여러 개의 YAML 파일을 정의해 쿠버네티스에 적용시키는 방식으로 동작
- YAML 파일을 잘 작성하는 것이 쿠버네티스를 잘 사용하는 방법

여러 개의 컴포넌트

- 쿠버네티스 노드의 역할은 크게 마스터와 워커로 구분
- 마스터 노드는 쿠버네티스가 제대로 동작할 수 있게 클러스터를 관리하는 역할을 담당
- 워커 노드에는 애플리케이션 컨테이너가 생성
- 쿠버네티스 클러스터 구성을 위해 kubelet이라는 에이전트가 모든 노드에서 실행
- kubelet은 컨테이너의 생성,삭제뿐만 아니라 마스터와 워커 노드 간의 통신 역할을 함께 담당하는 중요한 에이전트
- kubelet이 정상적으로 실행되지 않으면 해당 노드는 쿠버네티스와 제대로 연결되지 않을 수도 있음
- 쿠버네티스에서 반드시 도커를 사용해야 하는 것은 아니며, OCI(Open Container Initiative)라는 컨테이너의 런타임 표준을 구현한 CRI(Container Runtime Interface)를 갖추고 있다면 어떠한 컨테이너를 써도 문제는 없음

Pod

- 컨테이너를 다루는 기본 단위
- 파드는 1개 이상의 컨테이너로 구성된 컨테이너의 집합
- 1개의 파드에는 1개의 컨테이너가 존재할 수도 있고, 여러 개의 컨테이너가 존재할 수도 있음
- Nginx 웹 서비스를 쿠버네티스에서 생성하려면 1개의 파드에 Nginx 컨테이너 1개만을 포함해 생성
- 동일한 Nginx 컨테이너를 여러 개 생성하고 싶다면 1개의 Nginx 컨테이너가 들어 있는 동일한 파드를 여러 개 생성하면 됨
- 파드는 컨테이너 애플리케이션을 나타내기 위한 기본 구성 요소가 됨

그림 6.3 포드 1개에 컨테이너 1개만 포함해 생성

kubectl

- 쿠버네티스 클러스터에 명령을 내리는 역할
- 다른 구성 요소들과 다르게 바로 실행되는 명령 형태인 바이너리로 배포되기 때문에 마스터 노드에 있을 필요는 없음
- 통상적으로 API 서버와 주로 통신하므로 API 서버가 위치한 마스터 노드에 구성