Groundbreakers

10th Oracle Developer Meetup

Kubernetes : from Beginner to Advanced EP02

강인호
inho.kang@oracle.com

2019.04.18

ORACLE

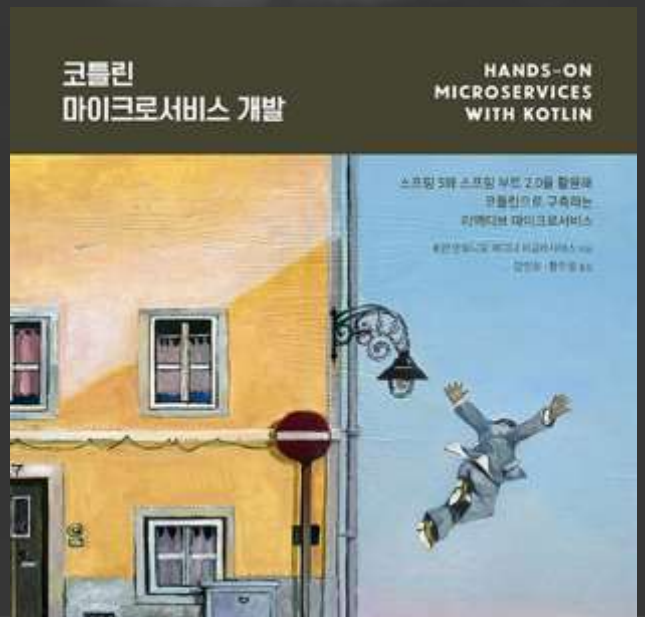
Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Kang In Ho

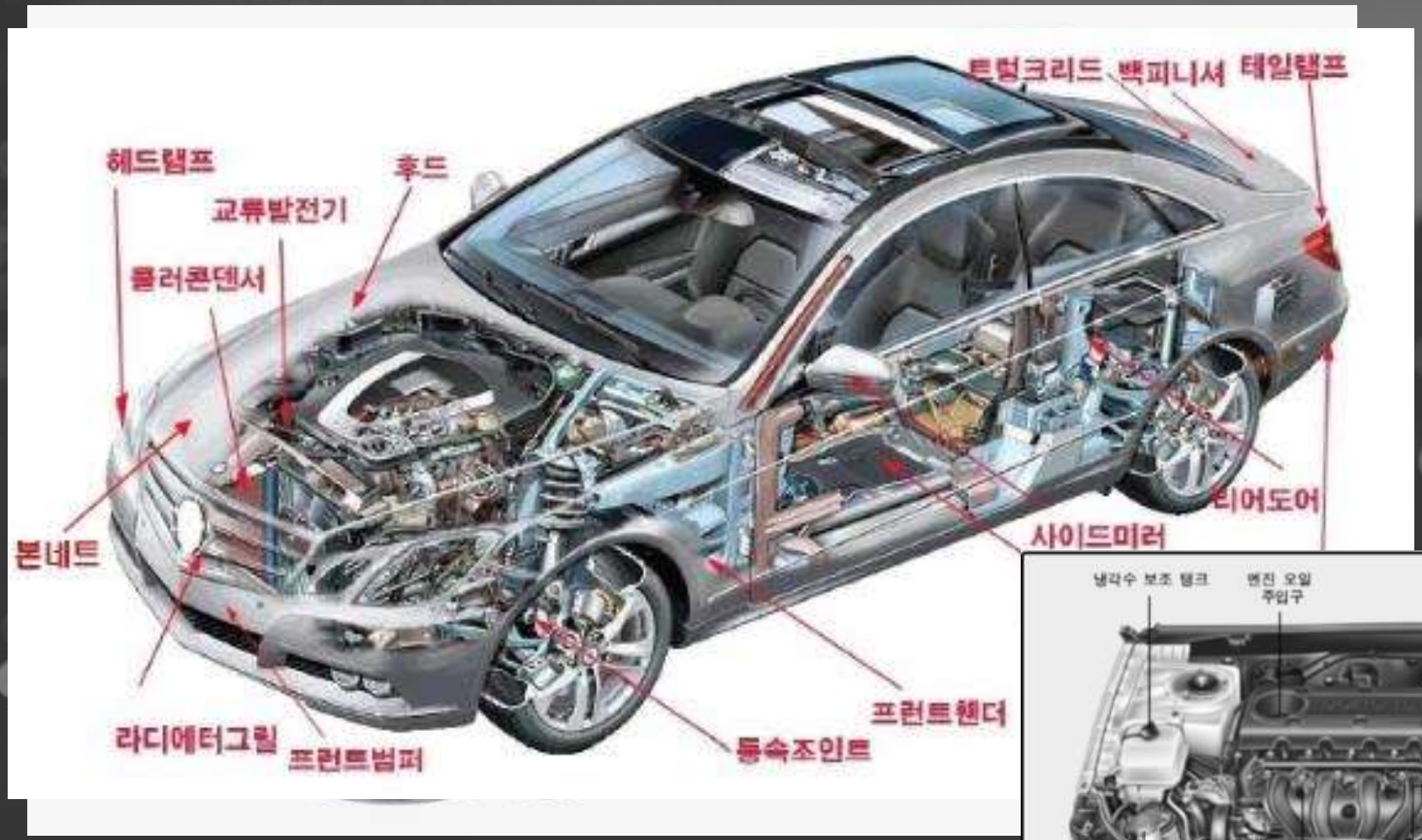
- .Net Developer
- CBD, SOA Methodology Consulting
- ITA/EA, ISP Consulting
- Oracle Corp.
  - Middleware
  - Cloud Native Application, Container Native
  - Emerging Technology Team
- k8s korea user group

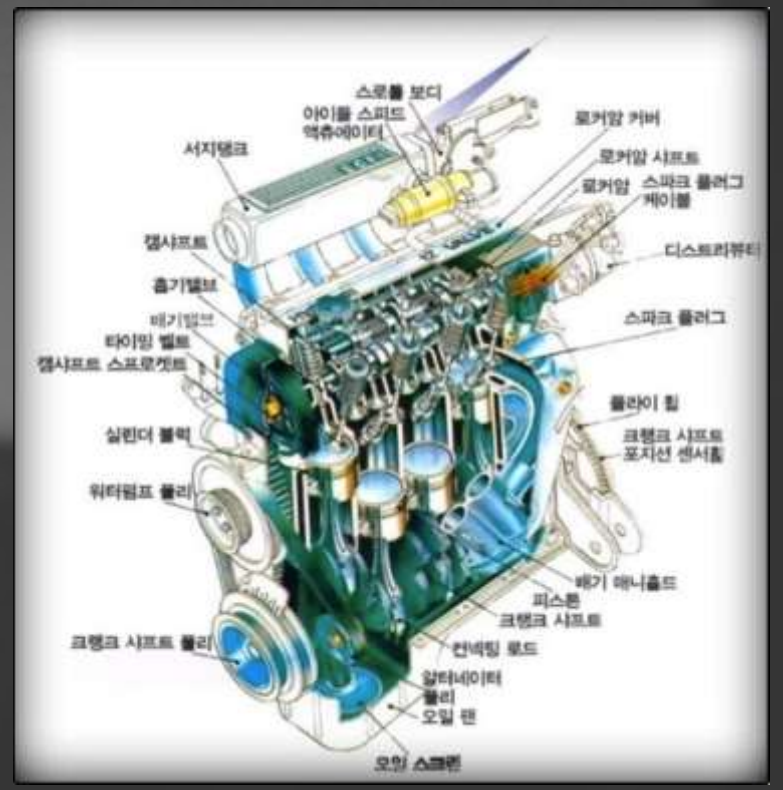
innoshom@gamil.com

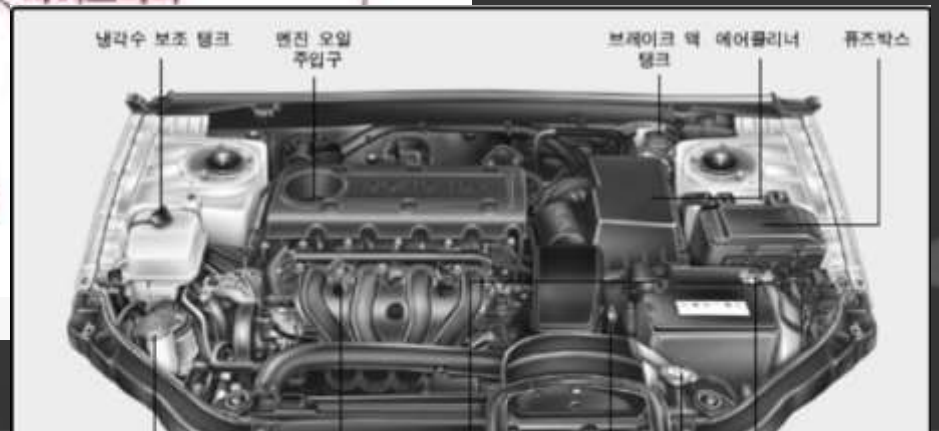


Kubernetes Concept

1 자동차


트렁크리드
백피니셔
테일램프
헤드램프
후드
교류발전기
콜러콘덴서
사이드미러
본네트
프런트휀더
라디에터그릴
프런트범퍼
등속조인트

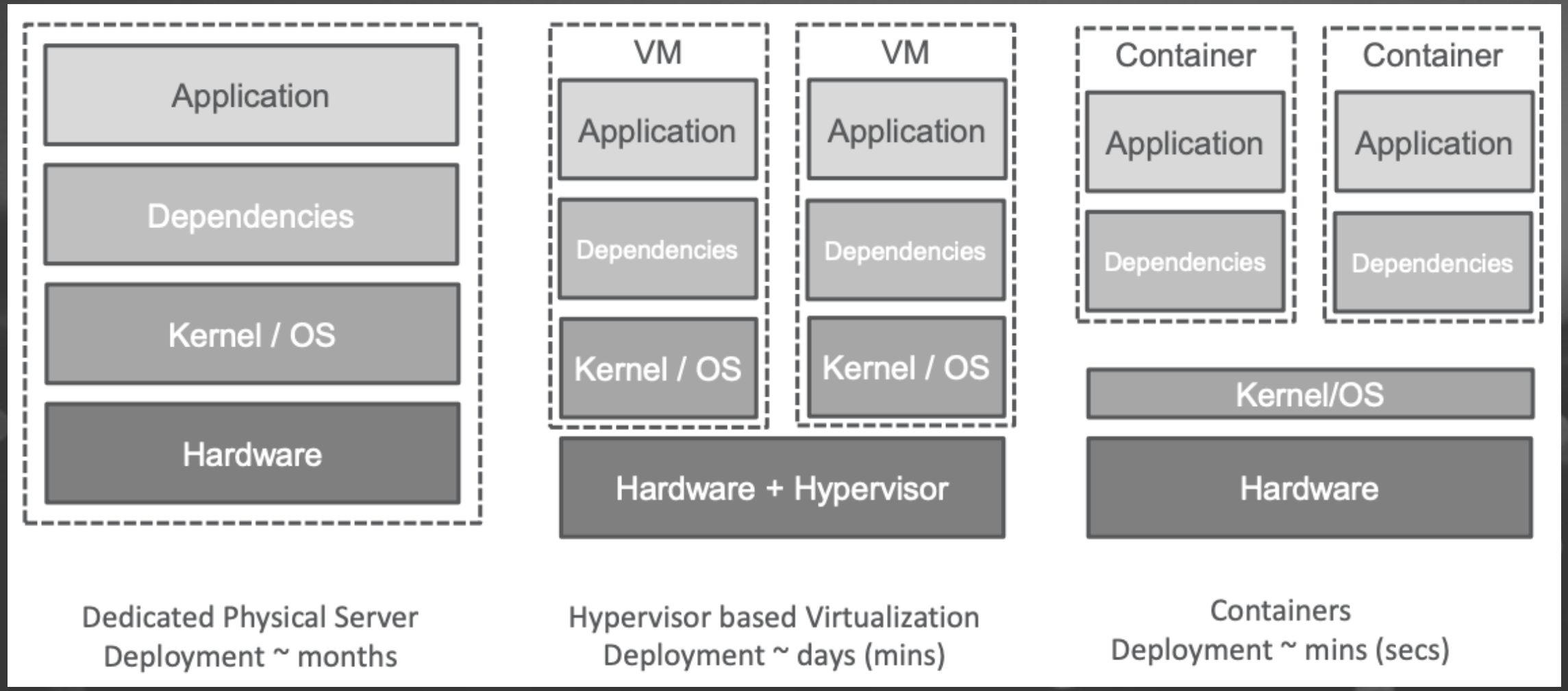

스로틀 보디
아이들 스피드
액츄에이터
서지탱크
로커암 커버
로커암 샤프트
로커암
스파크 플러그
케이블
캠샤프트
흡기캠프
배기캠프
타이밍 벨트
캠샤프트 스프로켓트
디스트리뷰터
스파크 플러그
플라이 휠
실린더 블럭
크랭크 샤프트
포지션 센서
워터펌프 풀리
배기 매니폴드
피스톤
크랭크 샤프트
크랭크 샤프트 풀리
컨넥팅 로드
얼터네이터
풀리
오일 팬
오일 스크린


냉각수 보조 탱크
엔진 오일
주입구
브레이크 액
탱크
에어클리너
퓨즈박스

Kubernetes Concept

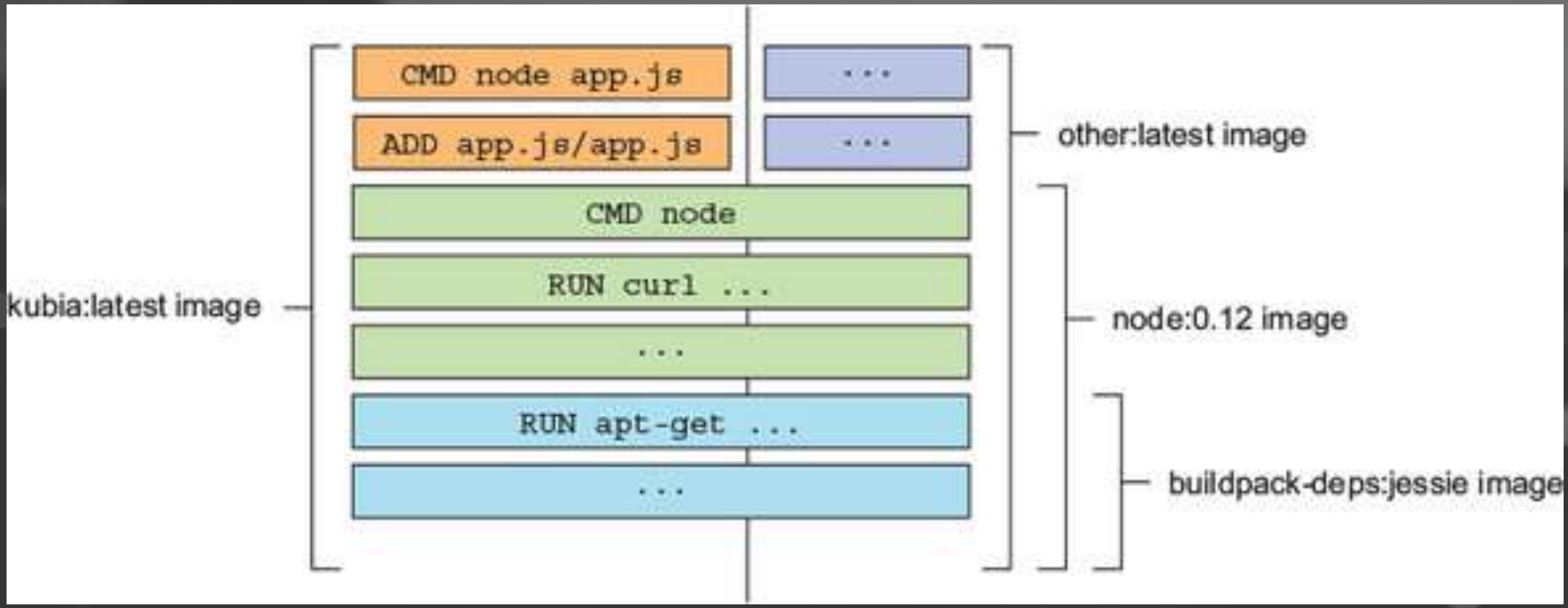
- Container

1 Container


Application
Dependencies
Kernel / OS
Hardware
VM
Application
Dependencies
Kernel / OS
VM
Application
Dependencies
Kernel / OS
Hardware + Hypervisor
Container
Application
Dependencies
Container
Application
Dependencies
Kernel/OS
Hardware
Dedicated Physical Server
Deployment ~ months
Hypervisor based Virtualization
Deployment ~ days (mins)
Containers
Deployment ~ mins (secs)

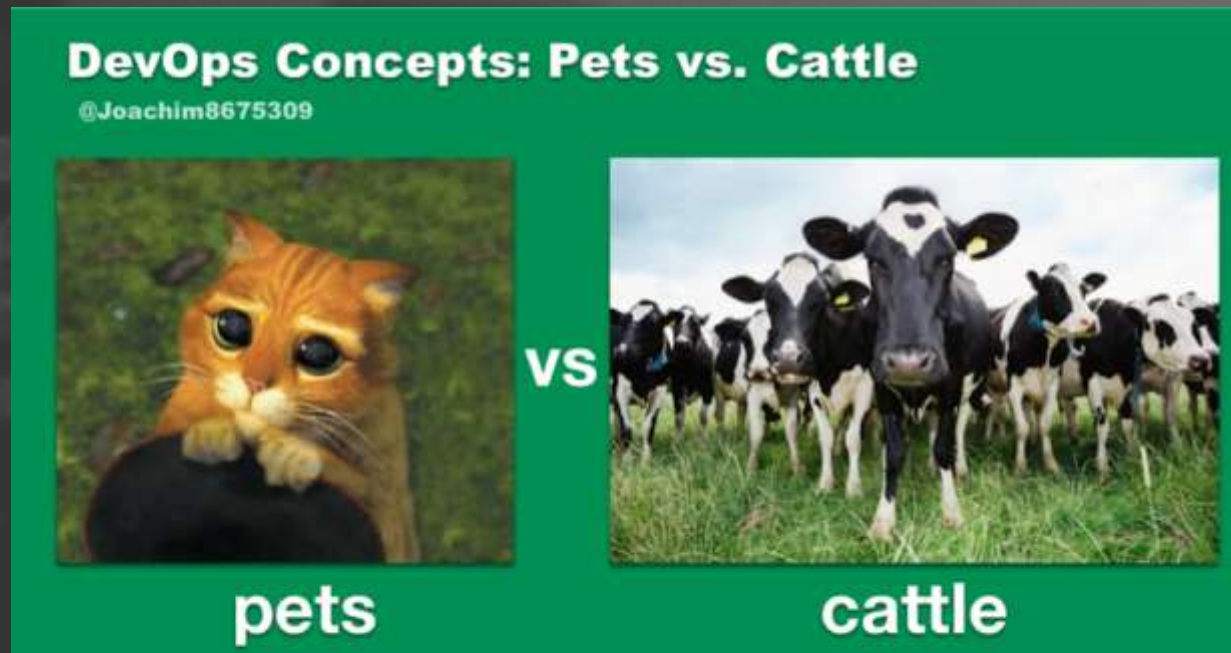
1 Container

Namespace, Cgroup, Union FS



출처 : kubernetes in action

1 Container – Pet vs Cattle



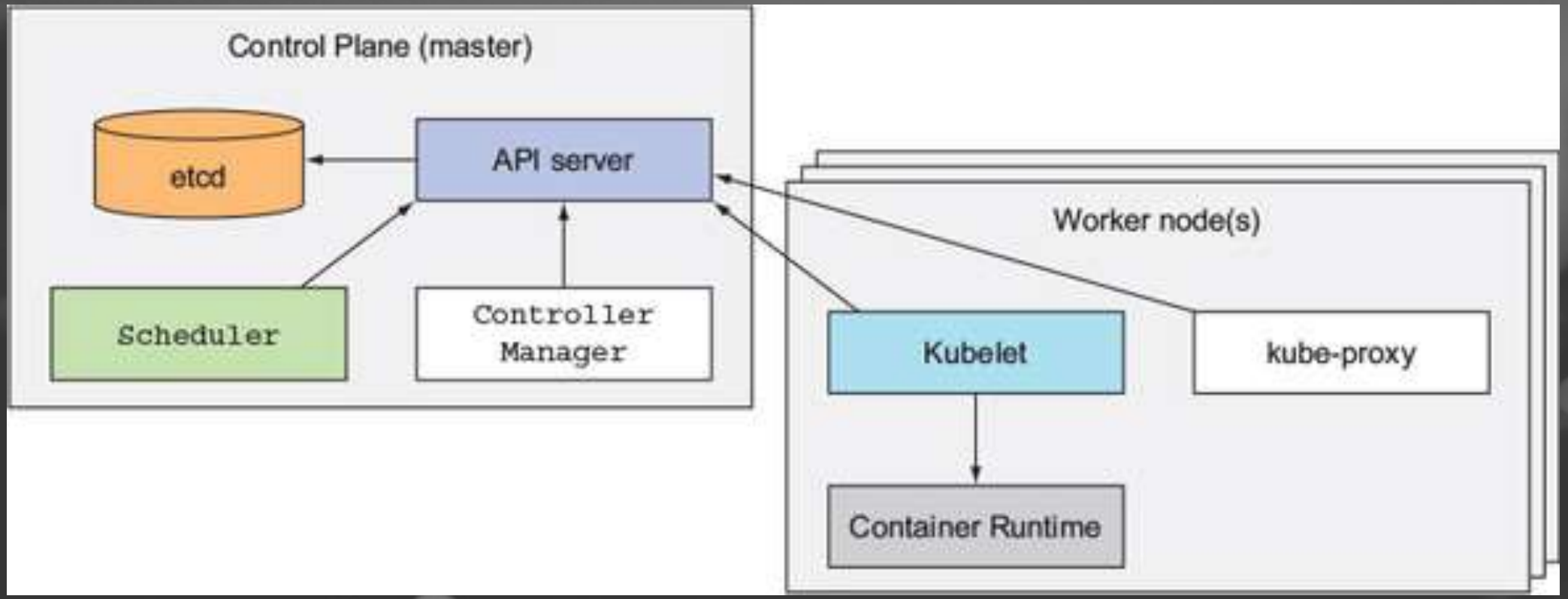
Chicken

Insect

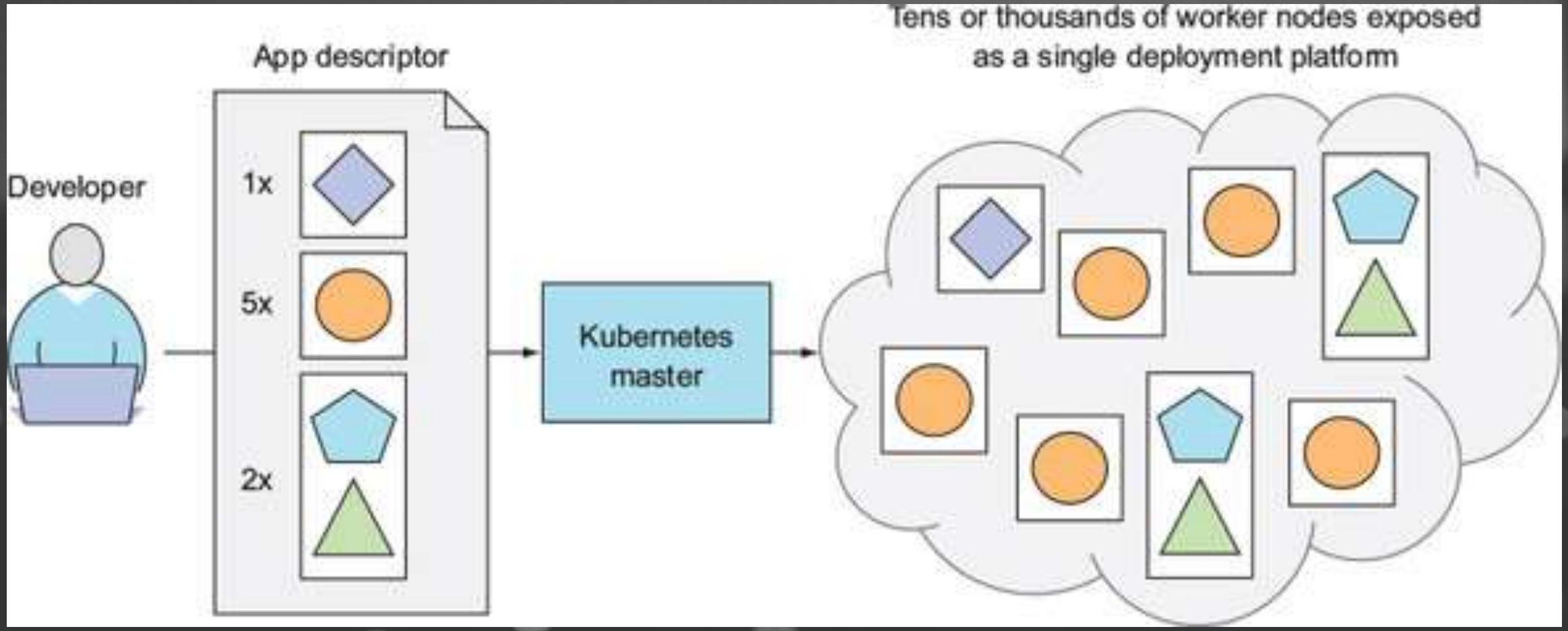
1 Container – Orchestration

kubernetes

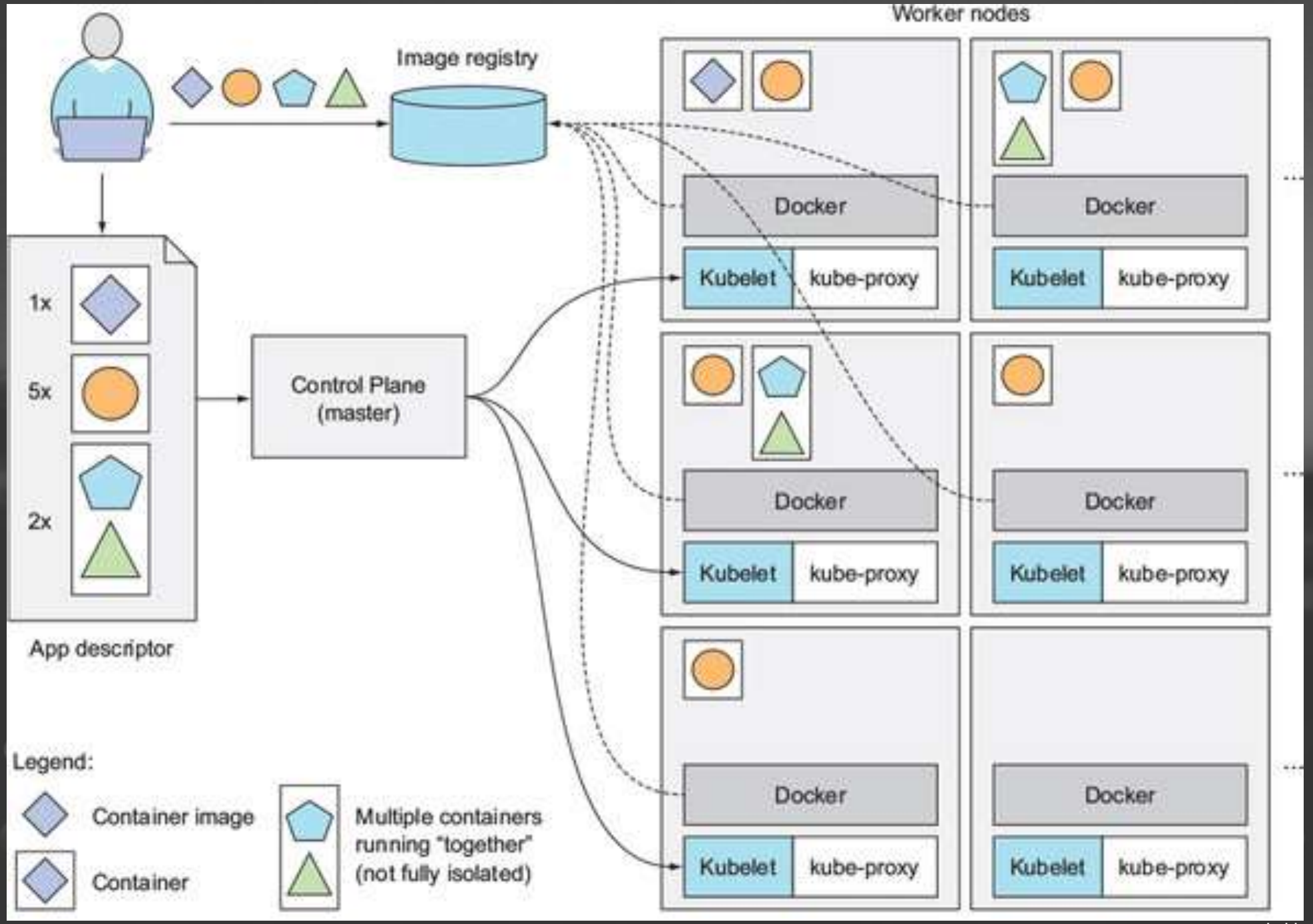
1 Kubernetes Architecture – Bird's View



1 Kubernetes – Bird View

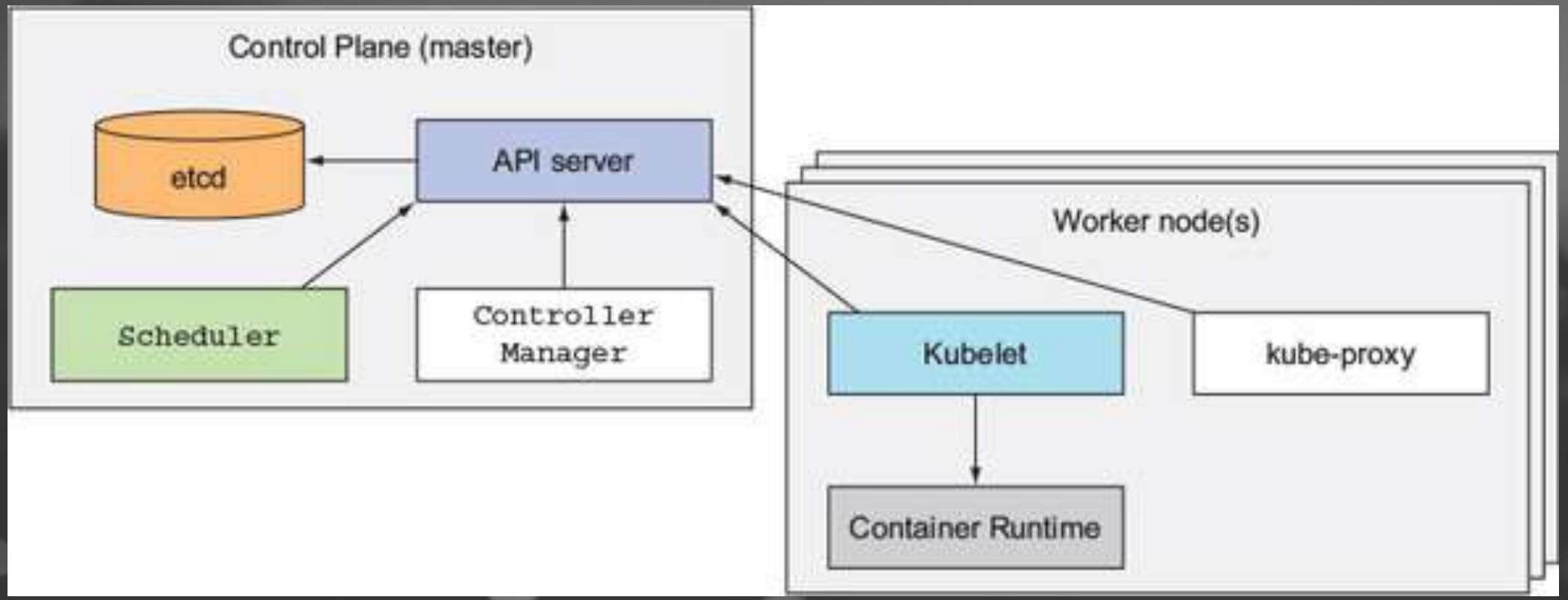


1

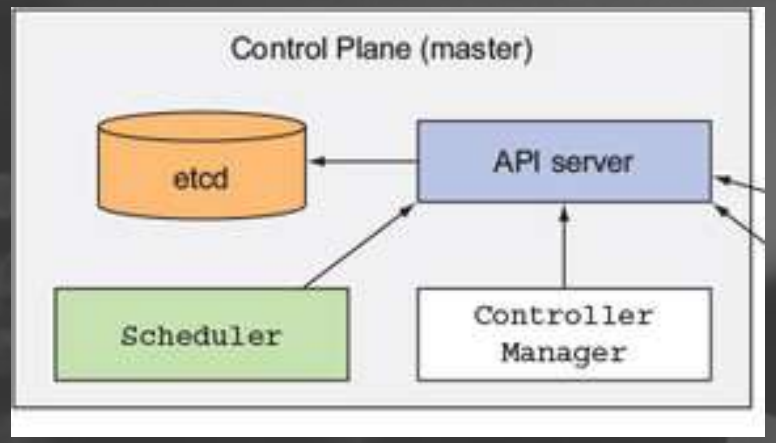


출처 : kubernetes in action

1 Kubernetes Architecture – Bird's View



1 Kubernetes Concept – Master



kube-apiserver

- 쿠버네티스 API를 노출하는 마스터 상의 컴포넌트. 쿠버네티스 컨트롤 플레인에 대한 **프론트엔드**이다.
- 수평적 스케일(즉, 더 많은 인스턴스를 디플로이하는 스케일)을 위해 설계되었다.

etcd

- 모든 클러스터 데이터를 담는 쿠버네티스 뒷단의 저장소로 사용되는 일관성·고가용성 **키-값 저장소**.
- 쿠버네티스 클러스터 정보를 담고 있는 etcd 데이터에 대한 **백업 계획은 필수**이다.

kube-scheduler

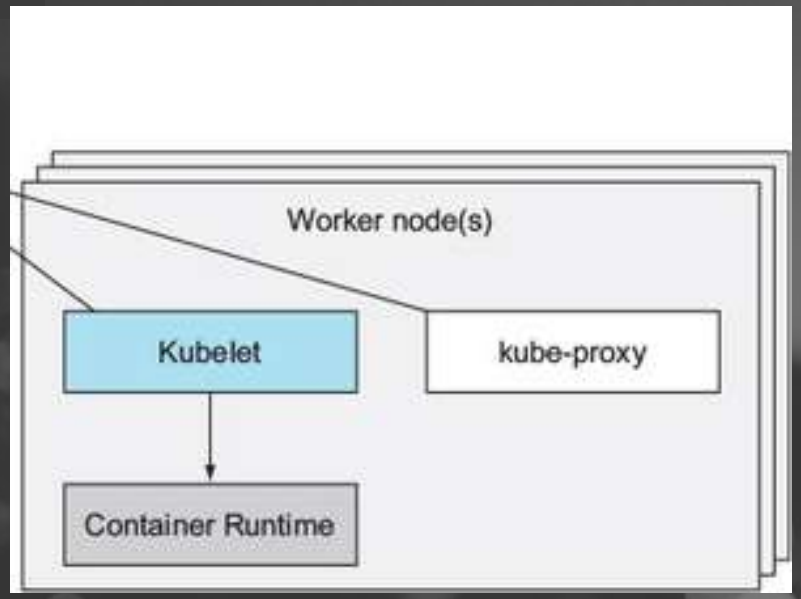
- **노드가 배정되지 않은 새로 생성된 파드를 감지하고 그것이 구동될 노드를 선택하**는 마스터 상의 컴포넌트.
- 스케줄링 결정을 위한 어카운트 내 요소들로는 개별 및 공동의 리소스 요건, 하드웨어/소프트웨어/정책 제약, 친밀 및 배격 명세, 데이터 지역성, 워크로드-간 간섭, 데드라인들이 포함된다.

kube-controller-manager

- **컨트롤러를 구동하는 마스터 상의 컴포넌트**.
- 논리적으로, 각 컨트롤러는 개별 프로세스이지만, 복잡성을 낮추기 위해 모두 단일 바이너리로 컴파일되고 단일 프로세스 내에서 실행된다.

- **노드 컨트롤러**: 노드가 다운되었을 때 통지와 대응에 관한 책임을 가진다.
- **레플리케이션 컨트롤러**: 알맞는 수의 파드들을 유지시켜 주는 책임을 가진다.
- **엔드포인트 컨트롤러**: 서비스와 파드를 연결시킨다

1 Kubernetes Concept – Node



kubelet

- 클러스터의 각 노드에서 실행되는 **에이전트**로 **컨테이너가 포드에서 실행 중**인지 확인합니다.
- kubelet은 다양한 메커니즘을 통해 제공되는 PodSpec을 가져 와서 해당 PodSpec에 설명된 컨테이너가 실행 중이며 **건강한 상태를 유지**하도록 합니다 .
- kubelet은 Kubernetes가 생성하지 않은 컨테이너를 관리하지 않습니다.

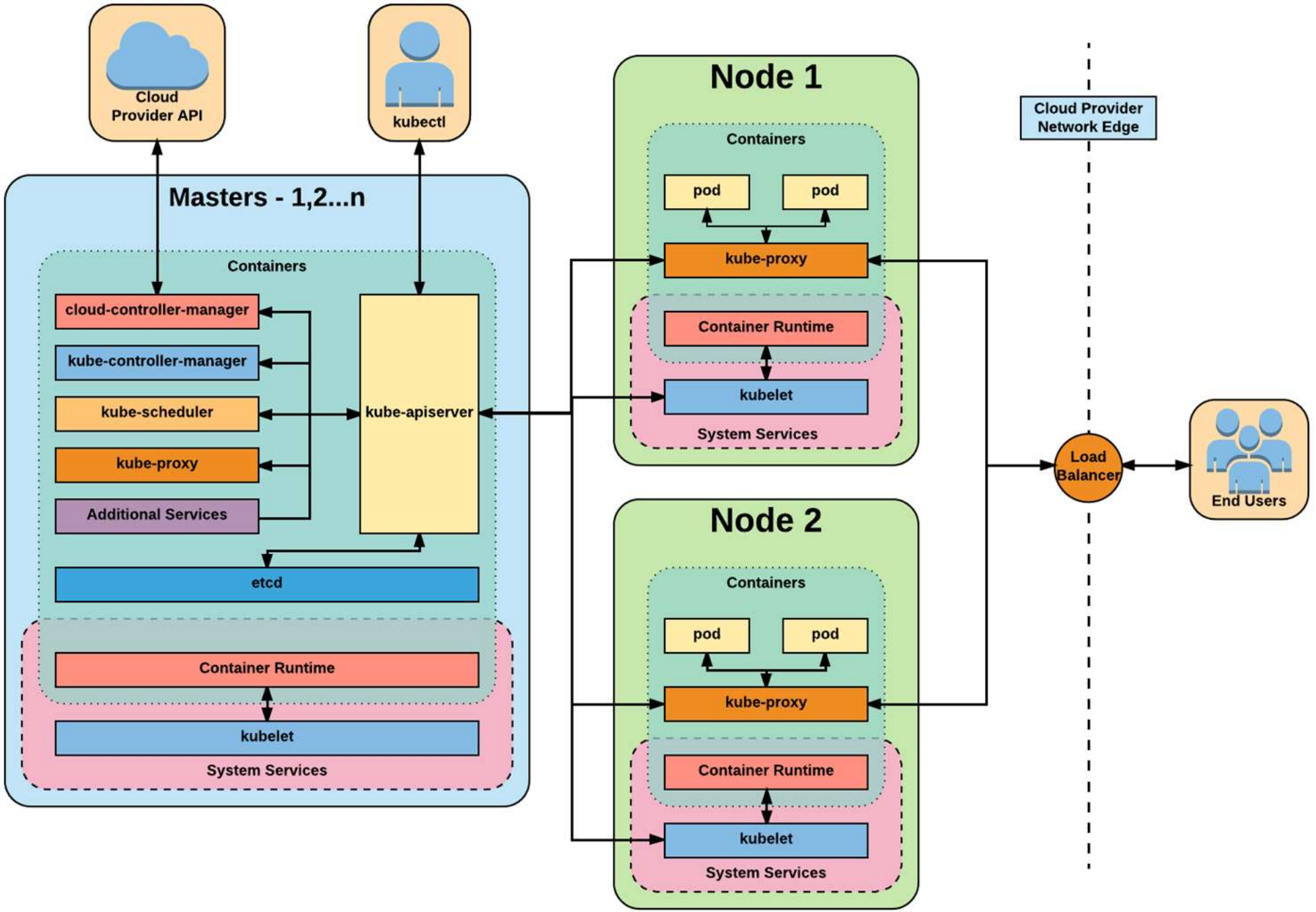
kube-proxy

- kube-proxy는 호스트 상에서 네트워크 규칙을 유지하고 **연결에 대한 포워딩을 수행함**으로서 쿠버네티스 서비스 추상화가 가능하도록 해준다.

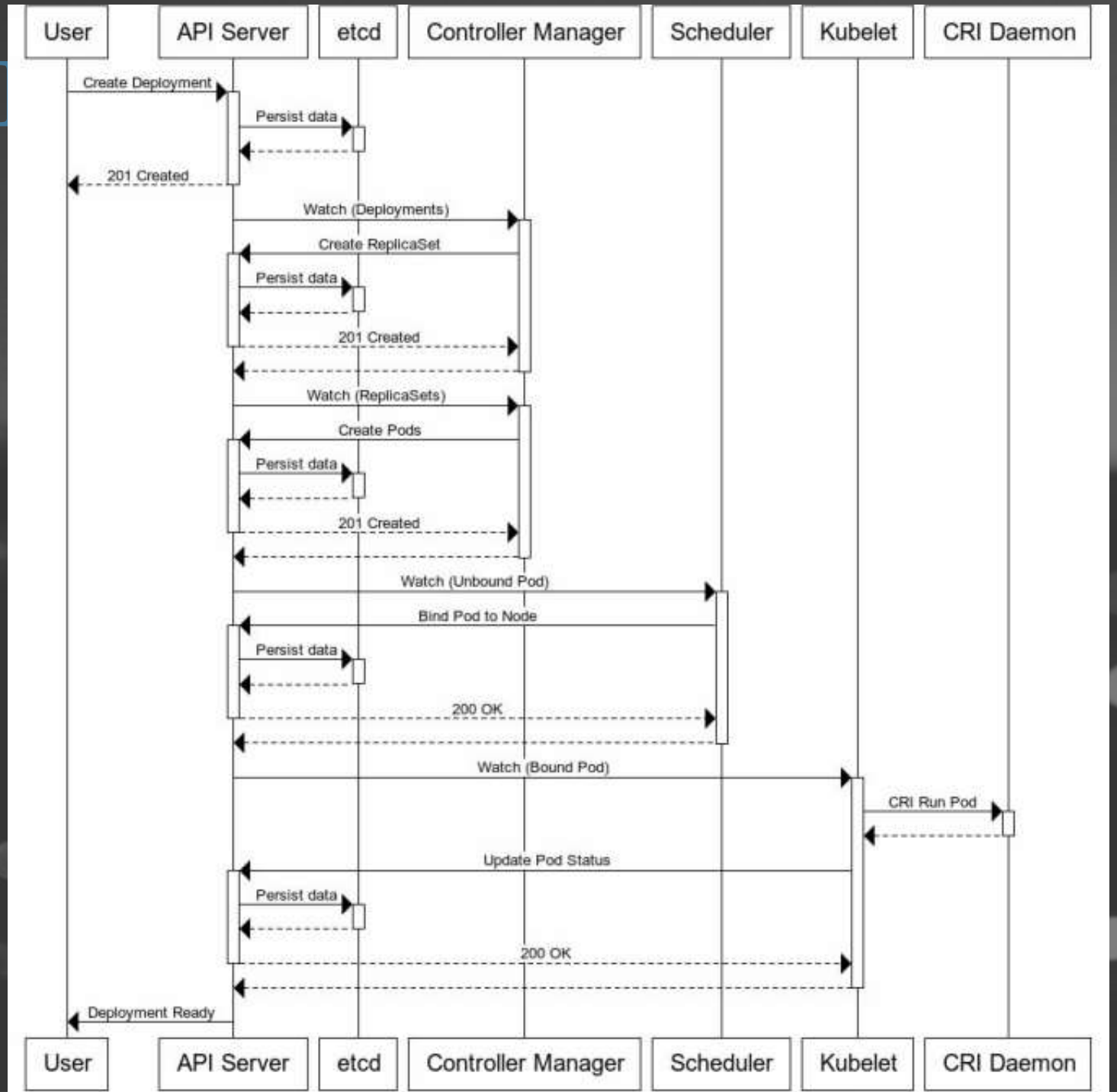
Container Runtime

- 컨테이너 런타임은 **컨테이너의 동작**을 책임지는 소프트웨어다. 쿠버네티스는 몇몇의 런타임을 지원하는데 Docker, containerd, cri-o, rktlet 그리고 Kubernetes CRI (Container Runtime Interface)를 구현한 모든 런타임이다.

Dive into Kubernetes Internal


Cloud Provider API
kubectl
Masters - 1,2...n
Containers
cloud-controller-manager
kube-controller-manager
kube-scheduler
kube-proxy
Additional Services
kube-apiserver
etcd
Container Runtime
kubelet
System Services
Node 1
Containers
pod
pod
kube-proxy
Container Runtime
kubelet
System Services
Node 2
Containers
pod
pod
kube-proxy
Container Runtime
kubelet
System Services
Cloud Provider Network Edge
Load Balancer
End Users

1

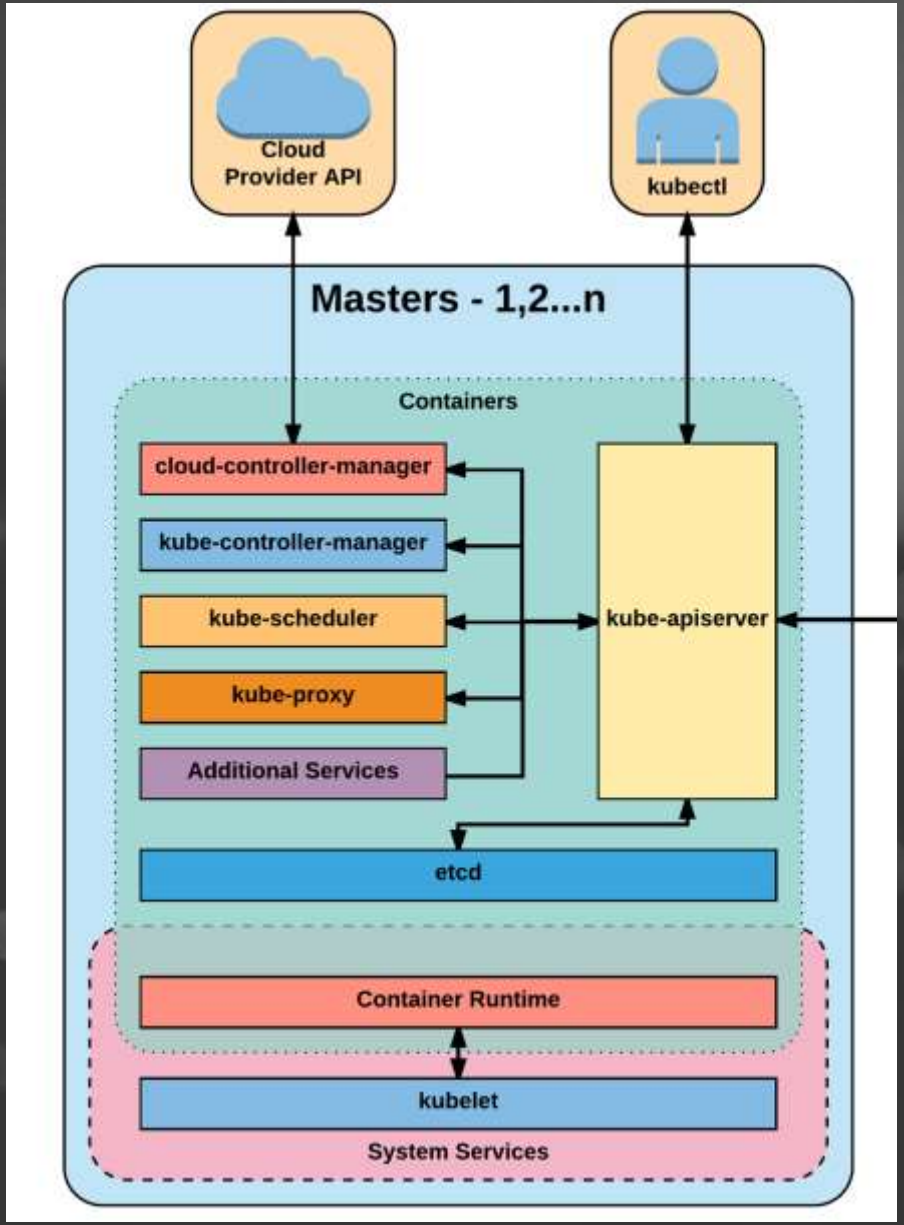


- Deployment ->
  ReplicatSet ->
- Pods

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```


Master Component

1 Master Component



- Kube-apiserver
- Etcd
- Kube-controller-manager
- Cloud-controller-manager
- Kube-scheduler

1 Master Component

- Components 상태
  - $kubectl get componentstatuses

```
vagrant@k8smaster:~$ docker ps
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS               NAMES
c9ef4c34aef6        f59dcacceff4           "/coredns -conf /etc…"   10 hours ago        Up 10 hours                             k8s_cored
c4bc724ca552        c264dff3420b           "td-agent"               10 hours ago        Up 10 hours                             k8s_fluen
73bc5f07b6c9        k8s.gcr.io/pause:3.1   "/pause"                 10 hours ago        Up 10 hours                             k8s_POD_c
c38152a79f1c        f59dcacceff4           "/coredns -conf /etc…"   10 hours ago        Up 10 hours                             k8s_cored
aad1a03246ce        k8s.gcr.io/pause:3.1   "/pause"                 10 hours ago        Up 10 hours                             k8s_POD_f
533da4e156cf        k8s.gcr.io/pause:3.1   "/pause"                 10 hours ago        Up 10 hours                             k8s_POD_c
cea7ae2d85c9        f0fad859c909           "/opt/bin/flanneld -…"   10 hours ago        Up 10 hours                             k8s_kube-
c135f3cf7be1        fdb321fd30a0           "/usr/local/bin/kube…"   10 hours ago        Up 10 hours                             k8s_kube-
fa2e03cb4df4        k8s.gcr.io/pause:3.1   "/pause"                 10 hours ago        Up 10 hours                             k8s_POD_k
f43f3723281b        k8s.gcr.io/pause:3.1   "/pause"                 10 hours ago        Up 10 hours                             k8s_POD_k
645435360a44        40a63db91ef8           "kube-apiserver --au…"   10 hours ago        Up 10 hours                             k8s_kube-
f82da7056889        3cab8e1b9802           "etcd --advertise-cl…"   10 hours ago        Up 10 hours                             k8s_etcd_
73f13ad62145        26e6f1db2a52           "kube-controller-man…"   10 hours ago        Up 10 hours                             k8s_kube-
7091223a069d        ab81d7360408           "kube-scheduler --ad…"   10 hours ago        Up 10 hours                             k8s_kube-
3ebc10b9348e        k8s.gcr.io/pause:3.1   "/pause"                 10 hours ago        Up 10 hours                             k8s_POD_k
1229e2671e8e        k8s.gcr.io/pause:3.1   "/pause"                 10 hours ago        Up 10 hours                             k8s_POD_k
e72b60da0a2e        k8s.gcr.io/pause:3.1   "/pause"                 10 hours ago        Up 10 hours                             k8s_POD_e
0989aede2225        k8s.gcr.io/pause:3.1   "/pause"                 10 hours ago        Up 10 hours                             k8s_POD_k
```

Master Component

- API Server

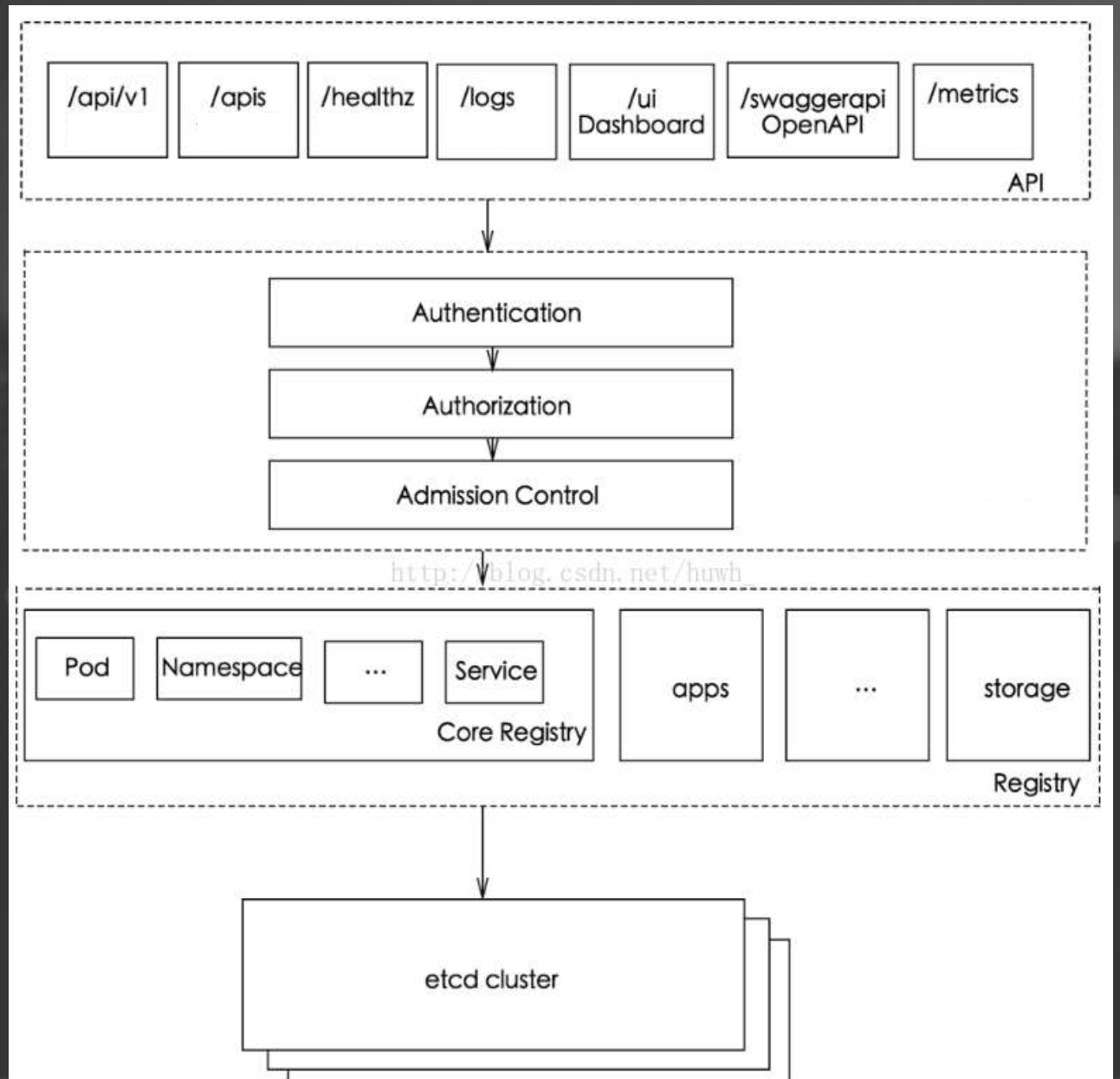
1 The Heart of the Cluster – API Server

- RESTful API를 통해서 kubectl와 같은 클라이언트와 통신
- RESTful API를 통해서 클러스터 상태를 쿼리 및 수정(CRUD)하고
- 그정보를 etcd에 저장한다.
- 모든 클라이언트와 컴포넌트는 오직 API 서버를 통해서만 상호작용한다.
- 클러스터에 대한 Gatekeeper 역할을 하며, 인증(Authen), 권한(Authoriz), 객체의 유효성 검사(Validation) and admission control
- JSON 기반의 HTTP API 가 기본이지만, 클러스터 내부 통신은 Protocol Buffer 도 지원

OpenAPI 규격을 조회하는 예제

1.10 이전	쿠버네티스 1.10 이상
GET /swagger.json	GET /openapi/v2 **Accept**: application/json
GET /swagger-2.0.0.pb-v1	GET /openapi/v2 **Accept**: application/com.github.proto-openapi.spec.v2@v1.0+protobuf
GET /swagger-2.0.0.pb-v1.gz	GET /openapi/v2 **Accept**: application/com.github.proto-openapi.spec.v2@v1.0+protobuf **Accept-Encoding**: gzip

1 The Heart of the Cluster – API Server

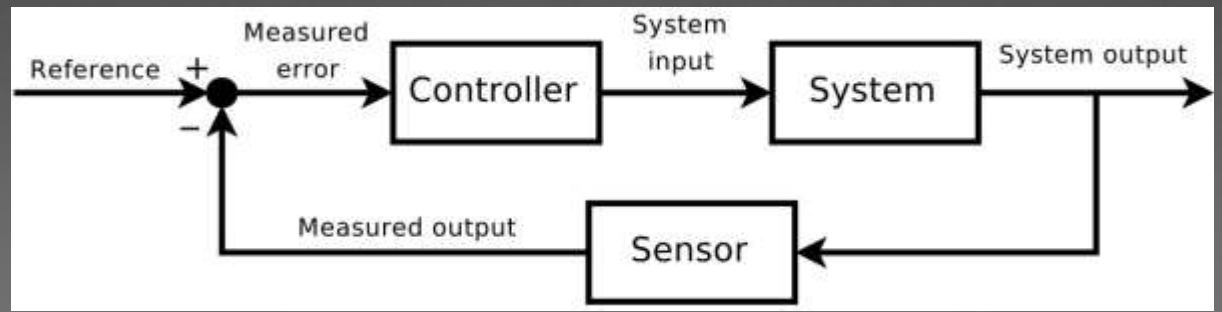


- kubectl create -f xxx.yaml
  → json Converting
- 인증, 권한
  - 요청한 Client의 인증서(key, token)를 검사
  - 요청된 액션을 수행할 권한 체크
  - Admission Control
    - 리소스 스펙에 누락된 것을 디폴트값이나 상속된 값으로 채움
    - Ex) AlwaysPullImages
      NamespaceLifeCycle
  - Resource Validation
  - ETCD에 /Registry 밑에 계층구조로 저장

Master Component

- Controller

1 Controller



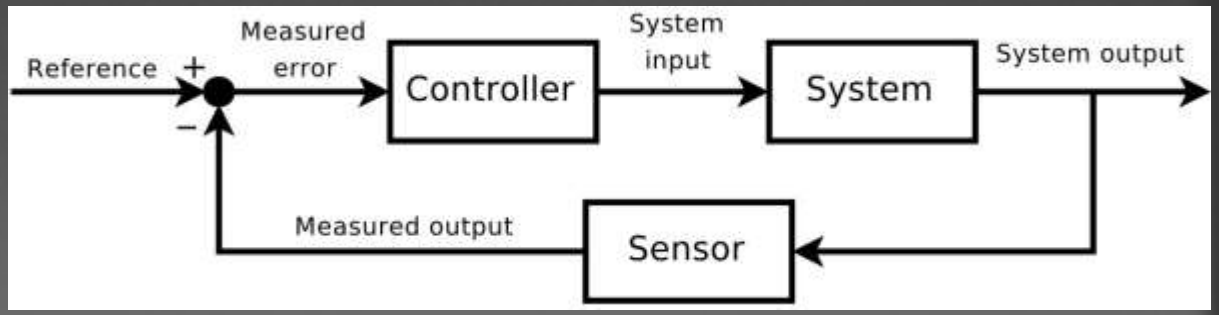
kube-controller-manager

컨트롤러를 구동하는 마스터 상의 컴포넌트.
논리적으로, 각 컨트롤러는 개별 프로세스이지만, 복잡성을 낮추기 위해 모두 **단일 바이너리로 컴파일**되고 단일 프로세스 내에서 실행된다.

API 서버를 통해서 Shared State(ETCD)의 정보를 Control loop를 통해 감시(watch)하고 있다가 **Current State를 Desired State로 변경을 시도**한다.

이들 컨트롤러는 다음을 포함한다. (약 26개)
- 노드 컨트롤러: 노드가 다운되었을 때 통지와 대응에 관한 책임을 가진다.
- 레플리케이션 컨트롤러: 시스템의 모든 레플리케이션 컨트롤러 오브젝트에 대해 알맞는 수의 파드들을 유지시켜 주는 책임을 가진다.
- 엔드포인트 컨트롤러: 엔드포인트 오브젝트를 채운다(즉, 서비스와 파드를 연결시킨다.)
- 서비스 어카운트 & 토큰 컨트롤러: 새로운 네임스페이스에 대한 기본 계정과 API 접근 토큰을 생성한다.

1 Controller Loop



단순화한 Controller Loop

```
for {
  desired := getDesiredState()
  current := getCurrentState()
  makeChanges(desired, current)
}
```

- Controller의 가장 중요한 역할을 Object의 current state와 desired state를 watch(감시)
- 이를 위해 API Server에 값을 조회하는데 매번 조화하는 것을 오버헤드가 있기 때문에 **event을 전달해주는 client-go library의 "Listwatcher interface"** 사용

```
lw := cache.NewListWatchFromClient(
      client,
      &v1.Pod{},
      api.NamespaceAll,
      fieldSelector)
```

https://engineering.bitnami.com/articles/a-deep-dive-into-kubernetes-controllers.html

1 Controller 정리

모든 컨트롤러는 API 서버를 통해 API 오브젝트를 가지고 동작한다.

컨트롤러는 Kubelet와 직접 통신하거나 어떤 종류의 명령도 내리지 않는다. 사실 kubelet의 존재하는지 조차 알지 못한다.

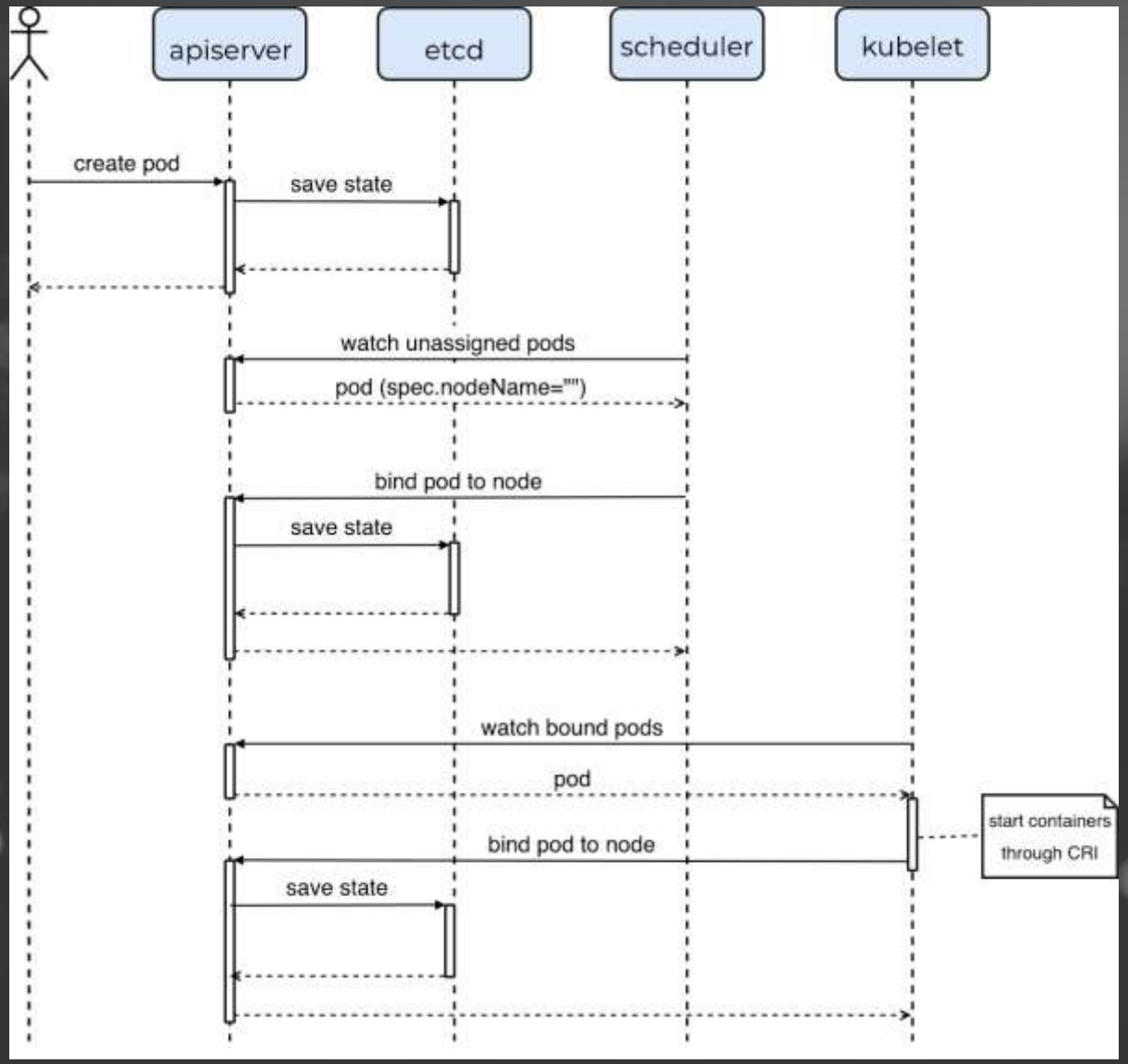
컨트롤러가 API 서버에서 리소스를 업데이트한 이후로 Kubelets와 쿠버네티스 서비스 프록시는 컨트롤러의 존재를 알지 못한다. 그리고 포드의 컨테이너를 회전시키고 네트워크 스토리지에 연결하거나 포드의 실제 로드 밸런싱을 설정한다.

컨트롤 플레인은 전체 시스템 동작의 한 부분을 처리하므로, 이런 동작이 쿠버네티스 클러스터에서 어떻게 펼쳐지는지 완전히 이해하려면 kubelet과 서비스 프록시가 하는 일을 이해할 필요가 있다.

Master Component

- Scheduler

1 Scheduler – Assign Node



So let's see how the scheduling lifecycle really looks like:

1. A pod is created and its desired state is saved to etcd with the node name unfilled.
2. The scheduler *somehow* notices that there is a **new pod with no node** bound.
3. It finds **the node that best fits that pod**.
4. Tells the apiserver to bind the pod to the node -> saves the new desired state to etcd.
5. Kubelets are watching bound pods through the apiserver, and start the containers on the particular node.

https://banzaicloud.com/blog/k8s-custom-scheduler/

1 Scheduler – Assign Node

단순화한 Scheduler

```
watch, err := s.clientset.CoreV1().Pods("").Watch(metav1.ListOptions{
  FieldSelector: fmt.Sprintf("spec.schedulerName=%s,spec.nodeName=" schedulerName ,
})

...

for event := range watch.ResultChan() {
  if event.Type != "ADDED" {
          continue
  }
  p := event.Object.(*v1.Pod)
  fmt.Println("found a pod to schedule:", p.Namespace, "/", p.Name)
  ...
}
```

- client-go library의 watch를 이용해 Event를 받는데 FieldSelector 에 spec.nodeName=”
- 노드가 할당되지 않은 pod에만 관심
- Scheduling 에 관련된 Factor

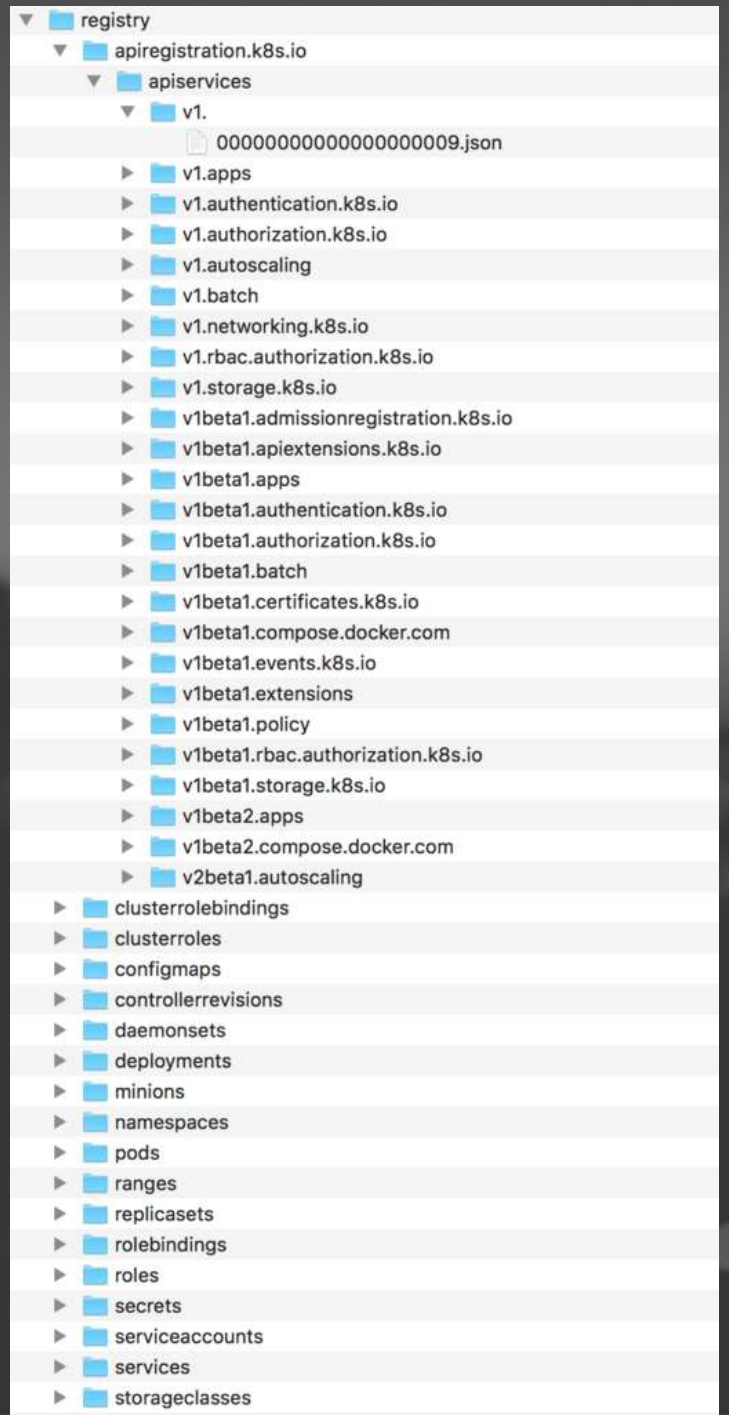
Master Component

- ETCD

1 ETCD

- 쿠버네티스 클러스터의 저장소

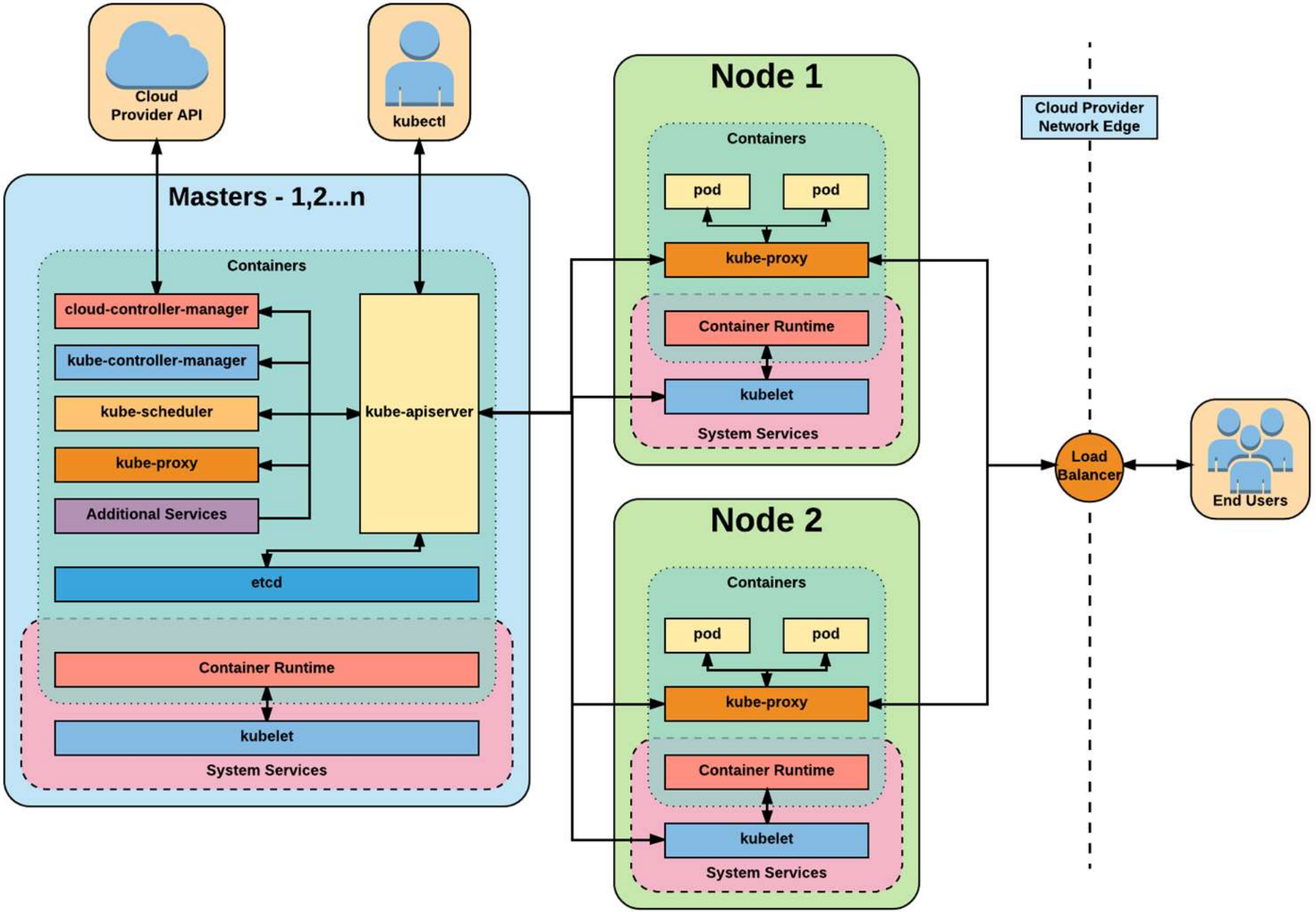
- 빠르고, 분산되며, 일관된 Key-Value Store
- Raft Consensus algorithm
- 클러스터는 fault-tolerant를 위해 3,5,7 과 같은 홀수개로 운영
  - 정족수(quorum)



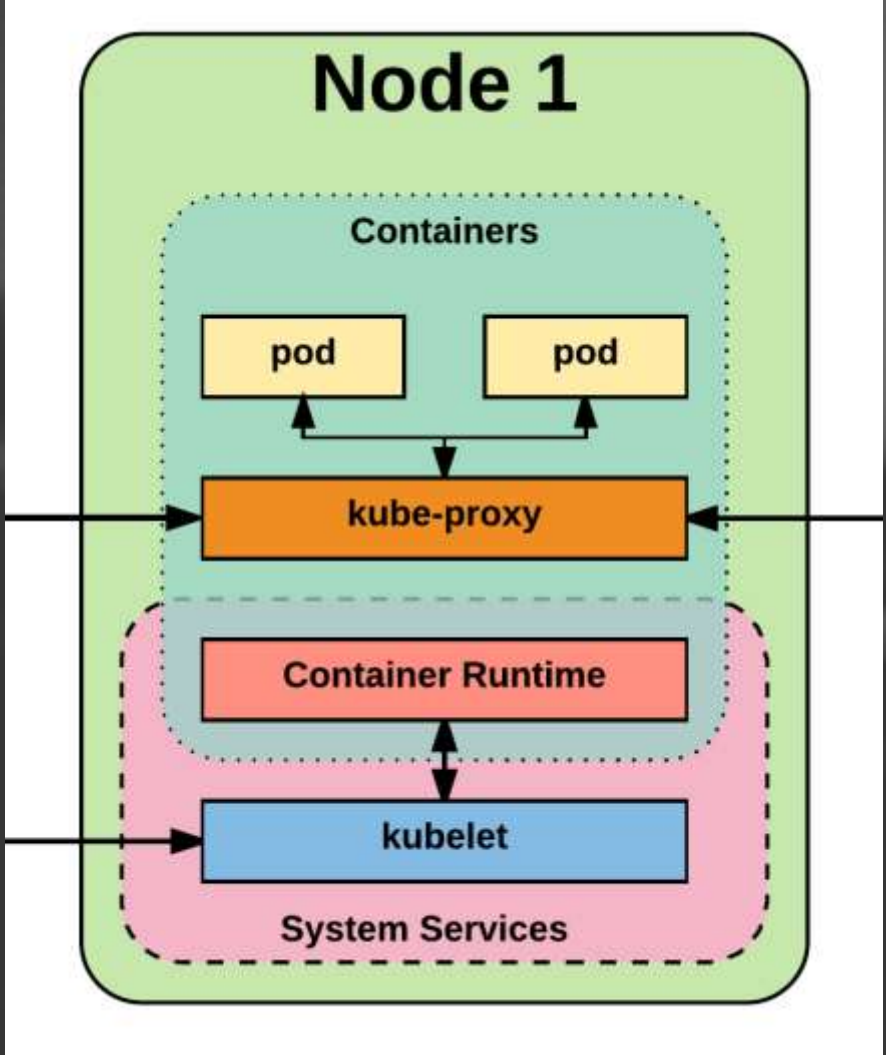
Node Component

- Kubelet
- Kube-proxy
- CRI

1

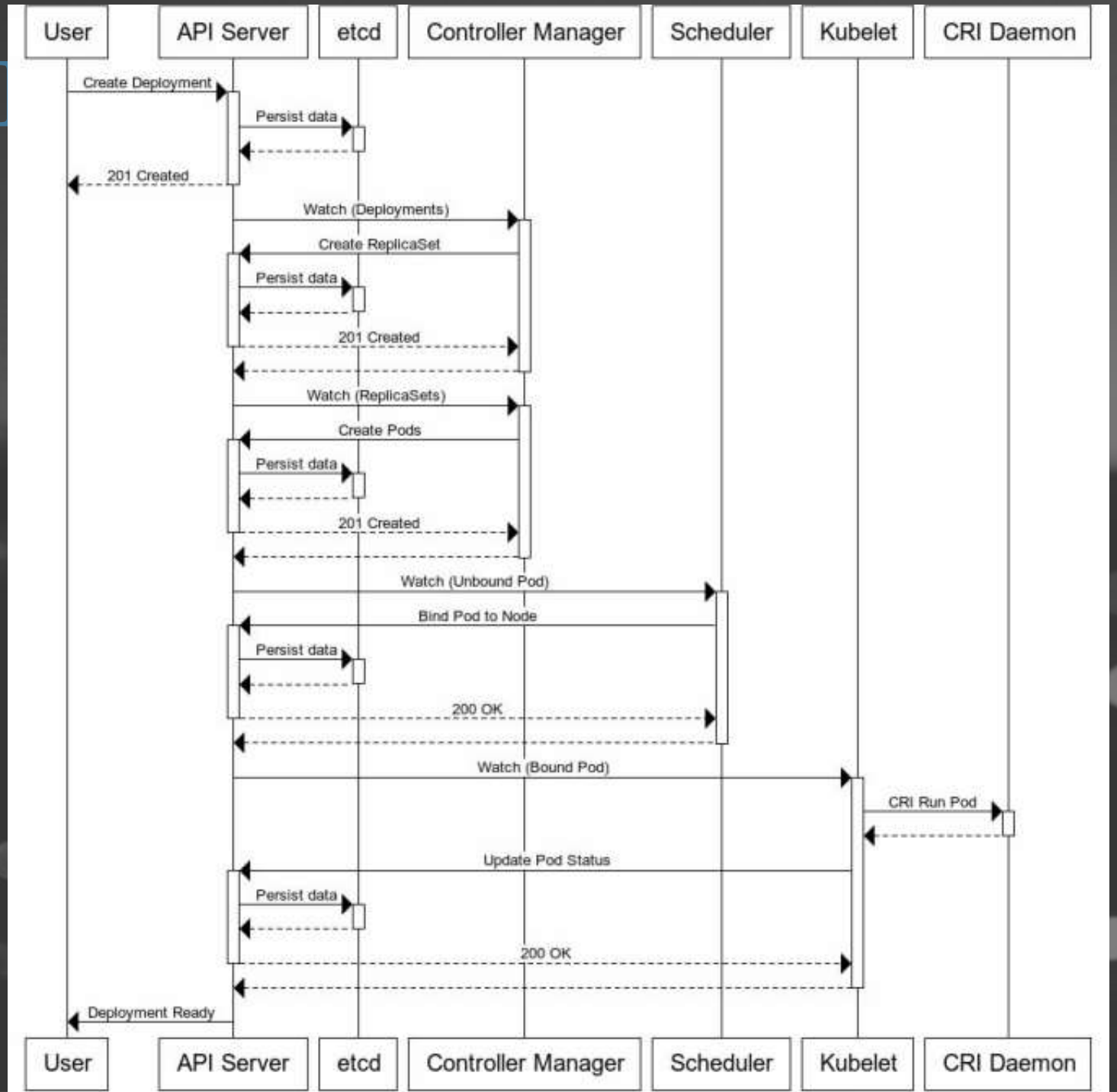

Cloud Provider API
kubectl
Masters - 1,2...n
Containers
cloud-controller-manager
kube-controller-manager
kube-scheduler
kube-proxy
Additional Services
kube-apiserver
etcd
Container Runtime
kubelet
System Services
Node 1
Containers
pod
pod
kube-proxy
Container Runtime
kubelet
System Services
Node 2
Containers
pod
pod
kube-proxy
Container Runtime
kubelet
System Services
Cloud Provider Network Edge
Load Balancer
End Users

1 Worker Node



- Kubelet
- Kube-proxy
- Container runtime engine

1



- Deployment ->
  ReplicatSet ->
- Pods

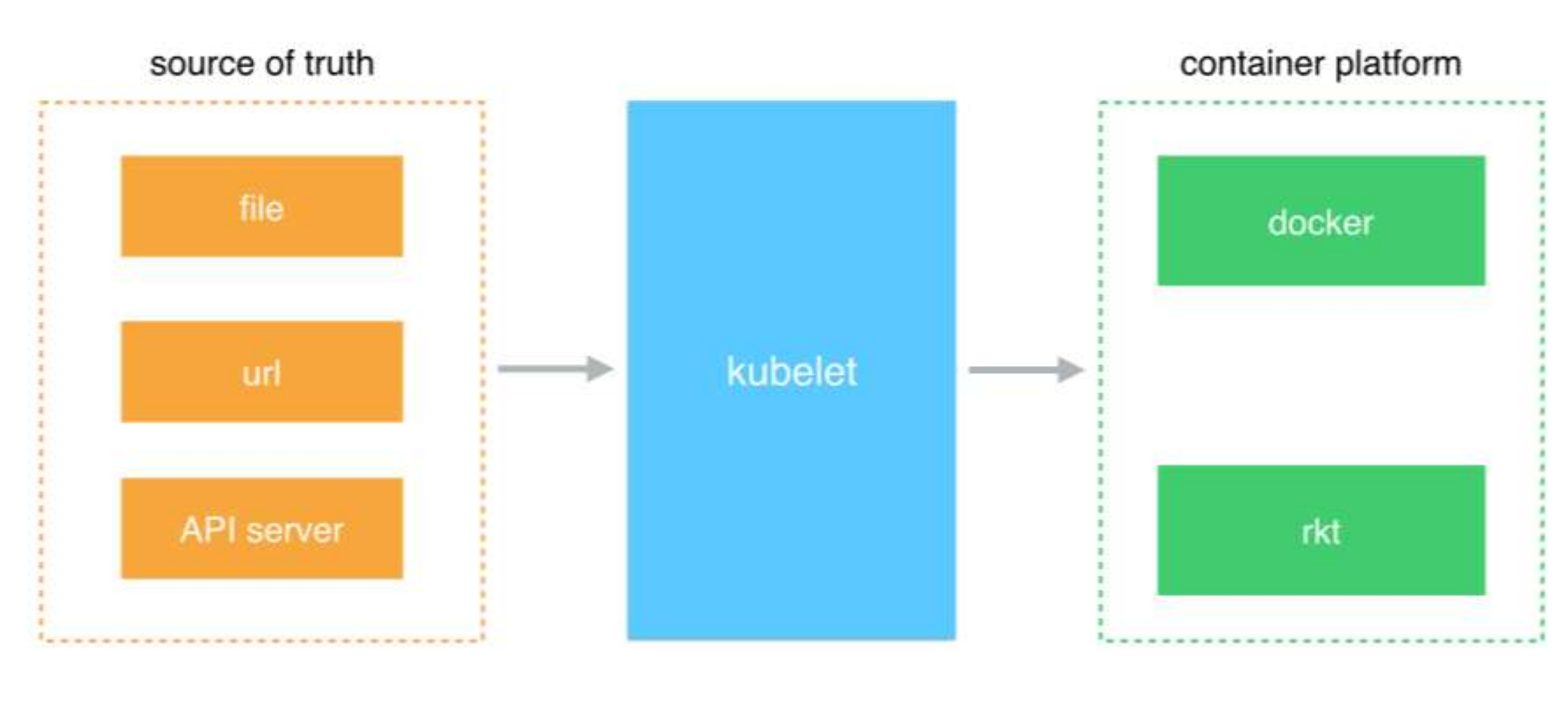
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

1 Kubelet

Node에서 Agent 역할을 하며 하며 Pod의 Lifecycle 을 관리한다.
지속적으로 실행중인 컨테이너를 모니터링하고 상태와 이벤트, 리소스등을 API 서버에 보고 한다.

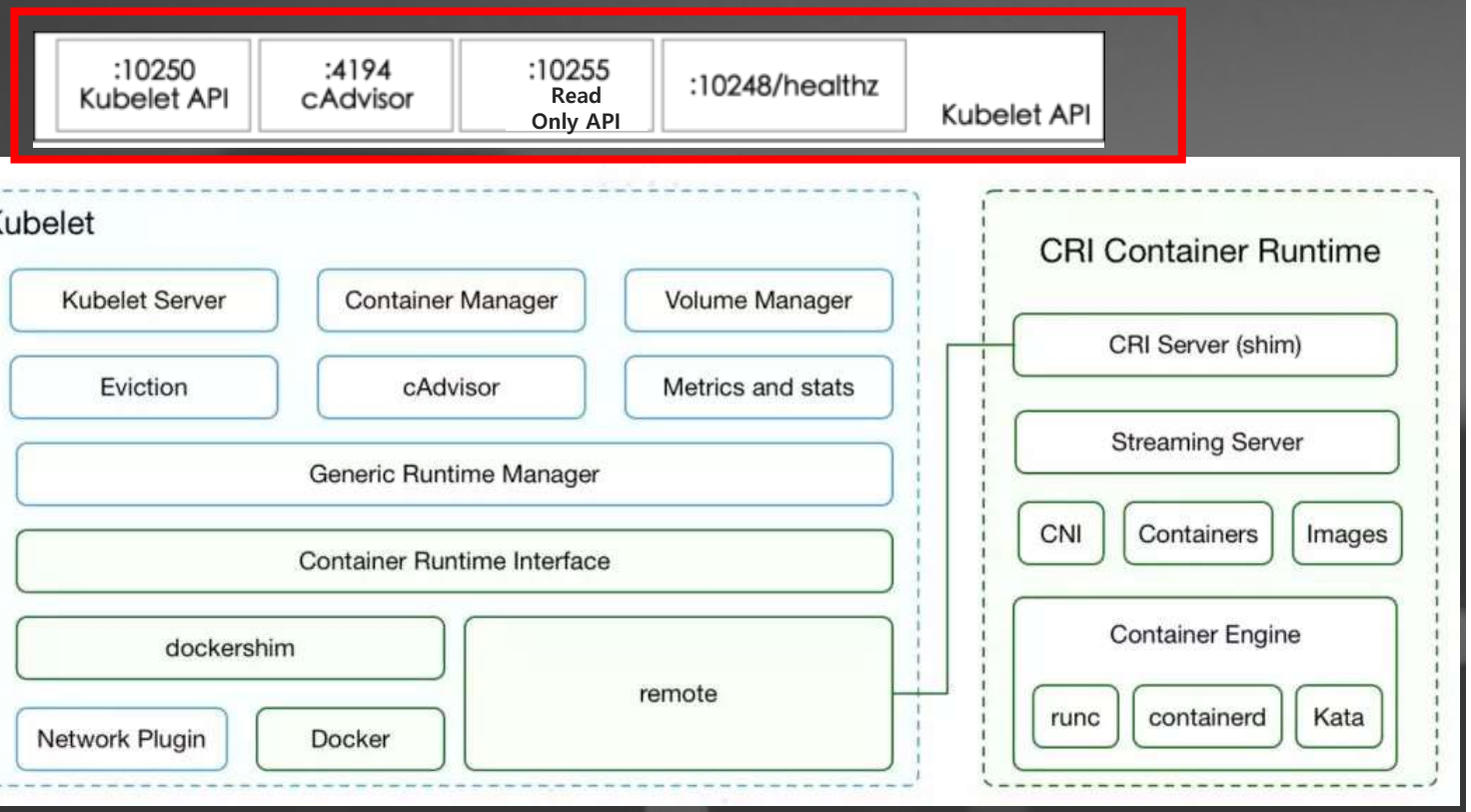
- Main features
  1) Pod Management
     주기적으로 pod/container의 상태를 읽어와서 desired status가 되도록 Container Runtime Interface에 명령
  2) Container Health Check
     container를 생성후 컨테이너의 health check
  3) Container Monitoring
     cAdvisor를 이용해서 노드의 리소스 사용량을 Master 노드 리포트

- File path
- URL HTTP Endpoint
- API Server



https://www.jianshu.com/p/f888020d7dcc

1 Kubelet



kubelet의 구성 요소

- Kubelet Server는 kube-apiserver 및 metrics-server와 같은 서비스 호출을 위한 API를 제공합니다. 예를 들어, kubectl exec는 Kubelet API / exec / {token}을 통해 컨테이너와 상호 작용해야합니다.

1. The kubelet listens by default to four ports, which are 10250, 10255, 10248, and 4194.

```
LISTEN    0    128          *:10250                *:*
LISTEN    0    128          *:10255                *:*
LISTEN    0    128          *:4194                 *:*
LISTEN    0    128    127.0.0.1:10248                *:*
```

- 10250 (kubelet API): The port on which the kubelet server communicates with the apiserver. It periodically requests the apiserver to obtain the tasks it should handle. Through this port, you can access the node resources and status.

- 10248 (health check port): You can determine kubelet is working properly by accessing the port, by the startup parameters kubelet `--healthz-port` and `--healthz-bind-address` to specify the listen address and port.

```
$ curl http://127.0.0.1:10248/healthz
ok
```

- 4194 (cAdvisor monitor): kublet can get through this port to environmental information and the node status container running on the node content, visit http: // localhost: 4194 you can see cAdvisor management interface, through the startup parameters kubelet `--cadvisor-port` can Specifies the port to start.

```
$ curl  http://127.0.0.1:4194/metrics
```

- 10255 (readonly API): Provides information about pod and node. The interface is exposed as read-only. Access to the port does not require authentication and authentication.

```
//  获取 pod 的接口，与 apiserver 的
// http://127.0.0.1:8080/api/v1/pods?fieldSelector=spec.nodeName=  接口类似
$ curl  http://127.0.0.1:10255/pods

// 节点信息接口,提供磁盘、网络、CPU、内存等信息
$ curl http://127.0.0.1:10255/spec/
```

1 Kubelet

- $netstat –tnlp
- $curl http://127.0.0.1:10248/healthz

```
[root@inho-k8s-worker2-setup opc]# netstat -ntpl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:10248         0.0.0.0:*               LISTEN      4452/kubelet
tcp        0      0 127.0.0.1:10249         0.0.0.0:*               LISTEN      16568/kube-proxy
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN      1/systemd
tcp        0      0 127.0.0.1:46163         0.0.0.0:*               LISTEN      4452/kubelet
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      5157/sshd
tcp        0      0 127.0.0.1:39011         0.0.0.0:*               LISTEN      5110/containerd
tcp6       0      0 :::10250                :::*                    LISTEN      4452/kubelet
tcp6       0      0 :::111                  :::*                    LISTEN      4438/rpcbind
tcp6       0      0 :::10256                :::*                    LISTEN      16568/kube-proxy
tcp6       0      0 :::30001                :::*                    LISTEN      16568/kube-proxy
tcp6       0      0 :::22                   :::*                    LISTEN      5157/sshd
[root@inho-k8s-worker2-setup opc]# curl http://127.0.0.1:10248/healthz
ok[root@inho-k8s-worker2-setup opc]#
```

1 Kubelet

Ensure Container Order - Init Container
Kubelet가 livenessProbe를 이용해서 Health check

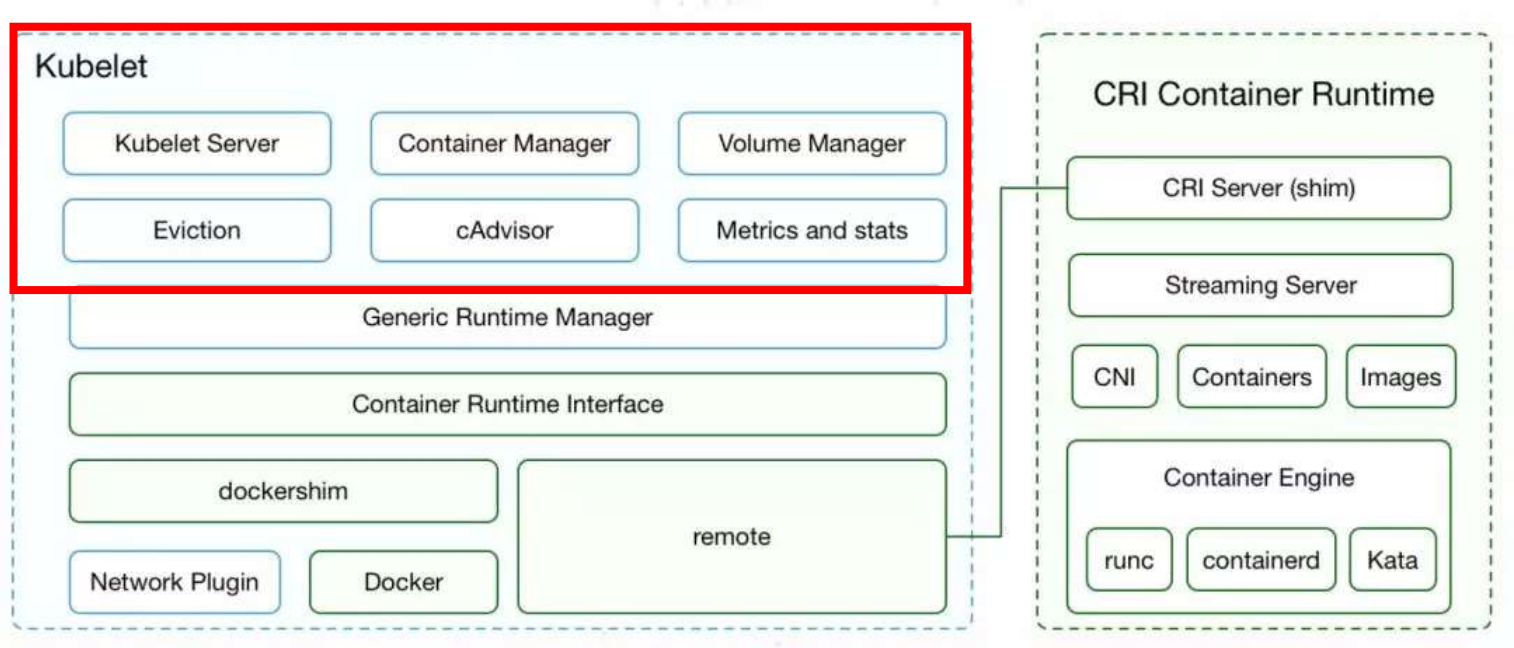
```yaml
livenessProbe:
      exec:
        command:
        - cat
        - /tmp/health
      initialDelaySeconds: 15
      timeoutSeconds: 1
```

```yaml
livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
        httpHeaders:
          - name: X-Custom-Header
            value: Awesome
      initialDelaySeconds: 15
      timeoutSeconds: 1
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: javaweb-2
spec:
  initContainers:
  - name: war
    image: resouer/sample:v2
    command: ["cp", "/sample.war", "/app"]
    volumeMounts:
    - mountPath: /app
      name: app-volume
  containers:
  - name: tomcat
    image: resouer/mytomcat:7.0
    command: ["sh","-c","/root/apache-tomcat-7.0.42-v2/bin/start.sh"]
    volumeMounts:
    - mountPath: /root/apache-tomcat-7.0.42-v2/webapps
      name: app-volume
    ports:
    - containerPort: 8080
      hostPort: 8001
  volumes:
  - name: app-volume
    emptyDir: {}
```

https://www.slideshare.net/resouer/k8s-meetup-3june-77322267

1 Kubelet

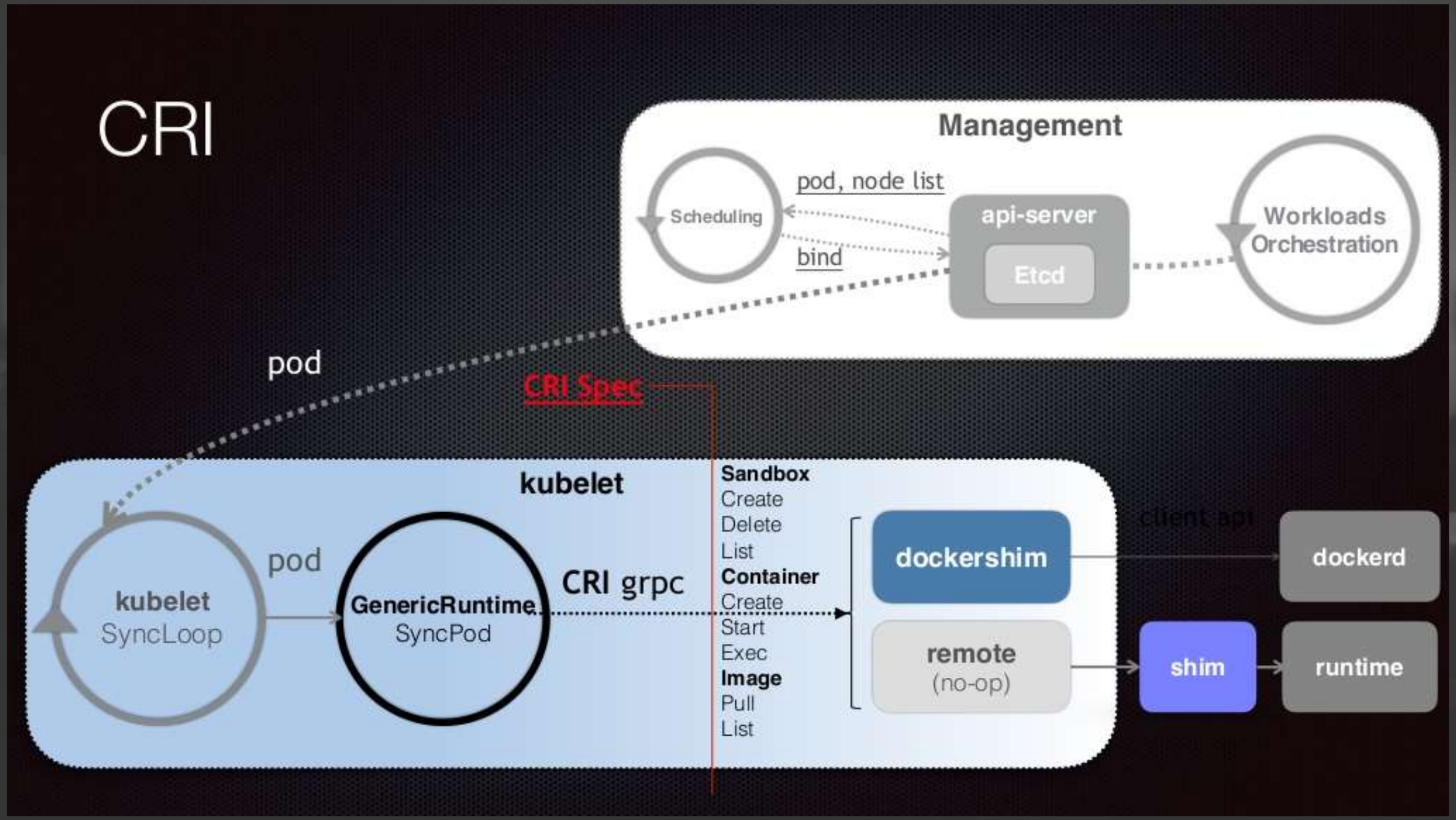


kubelet의 구성 요소

- 컨테이너 관리자는 CGroups, QoS, cpuset, device 등과 같은 컨테이너의 다양한 자원을 관리합니다.
- 볼륨 관리자는 디스크의 포맷, 노드 로컬에 마운트, 마지막으로 마운트 경로를 컨테이너에 전달하는 것과 같이 컨테이너의 스토리지 볼륨을 관리합니다.
- Eviction 관리자 는 자원이 충분하지 않을 때 우선 순위가 낮은 컨테이너의 퇴출 시키는 것과 같이 우선 순위가 높은 컨테이너의 운영을 보장하기 위해 컨테이너 퇴거를 담당합니다.
- cAdvisor는 컨테이너에 Metrics를 제공합니다.
- 메트릭 및 통계는 메트릭과 같은 컨테이너 및 노드에 대한 메트릭을 제공합니다. / stats / summary를 통해 추출 된 서버 메트릭은 HPA의 자동 확장을위한 기본 요소입니다.
- 일반 런타임 관리자는 CRI 상호 작용 및 컨테이너 및 미러링 관리를 담당하는 컨테이너 런타임 관리자입니다

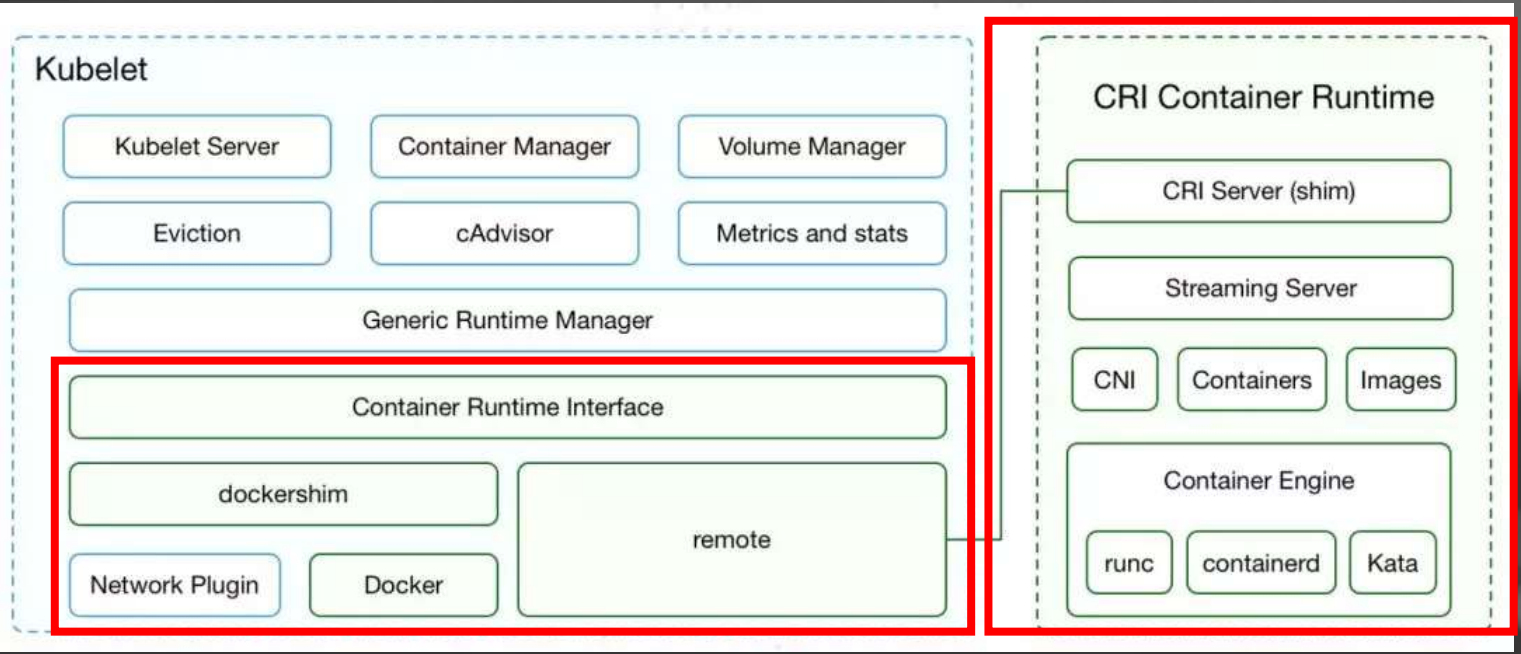
Node Component

- Container Runtime Interface



https://www.slideshare.net/resouer/k8s-meetup-3june-77322267

1 Kubelet



CRI에는 2개의 Container Runtime이 존재

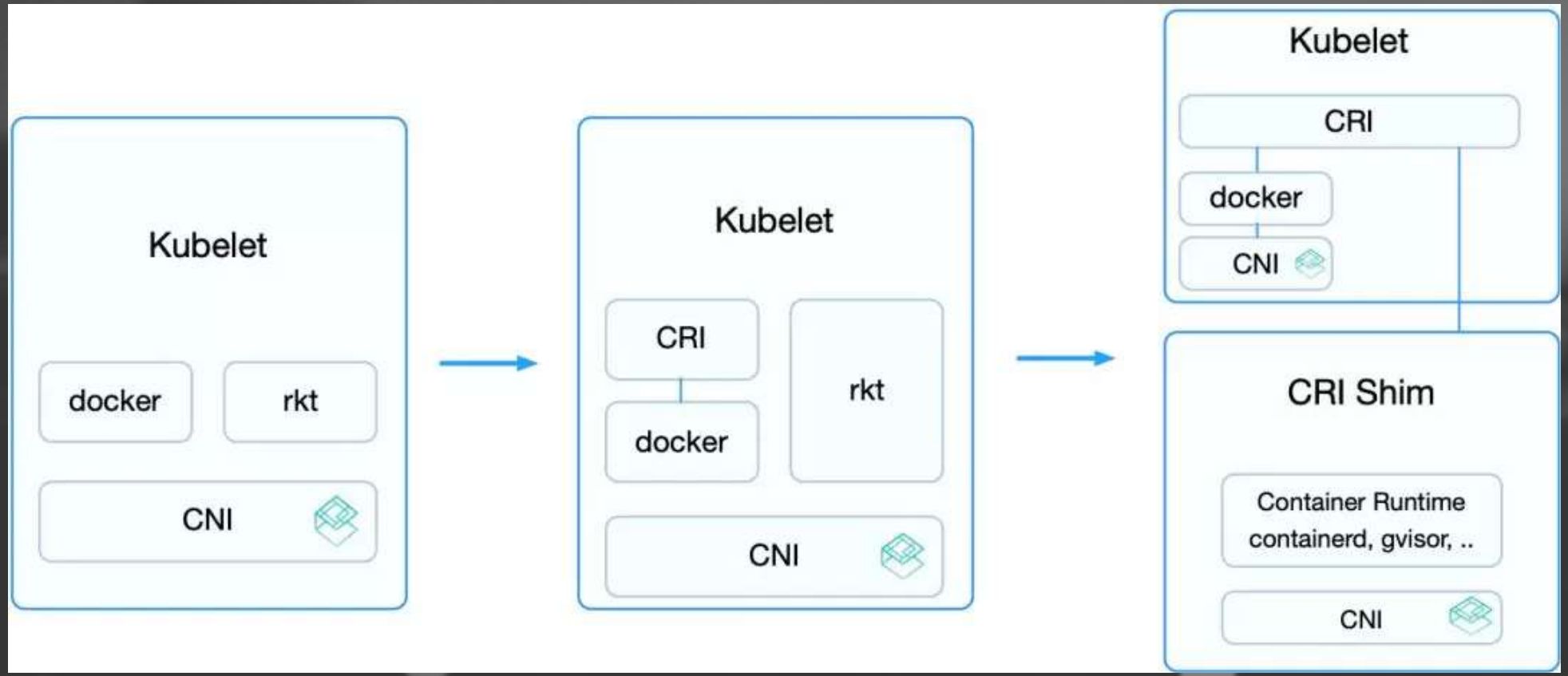
1) Docker , dockershim
2) CRI Container Runtime
   - runC, containerd, Kata 등

- CRI Server: CRI gRPC server, listening on the unix socket;
- Streaming Server: Provides streaming API, including Exec, Attach, and Port Forward;
- Management of containers and images, such as pulling images, creating and launching containers, etc.
- Support for the CNI network plugin for configuring the network for the container;
- Container engine management, such as support for runc, containerd, or support for multiple container

https://juejin.im/entry/5bc71b3d6fb9a05cf23029b8

1 Container Runtime

Container Runtime 변화



https://juejin.im/entry/5bc71b3d6fb9a05cf23029b8

1 Container Runtime

1.2015년 도커가 원칙을 가지고 플랫폼을 분리했다

2.runC : https://blog.docker.com/2015/06/runc/

- o2015년 6월 Docker platform에서 분리했다. 분리하면서 멋진 몇가지 원칙을 내놓았는데 이름하여 "**infrastructure plumbing manifesto**"
  - o**3가지 원칙**
    - **1.Whenever possible, re-use existing plumbing and contribute improvement back**
    - **2.Follow the unix principles : several simple components are better than a single, complicated one**
    - **3.Define standard interfaces which can be used to combine many simple components into a more sophisticated system**
  - o그 첫번째가 runC이다.

runC is a lightweight, portable container runtime. It includes all of the plumbing code used by Docker to interact with system features related to containers. It is designed with the following principles in mind:

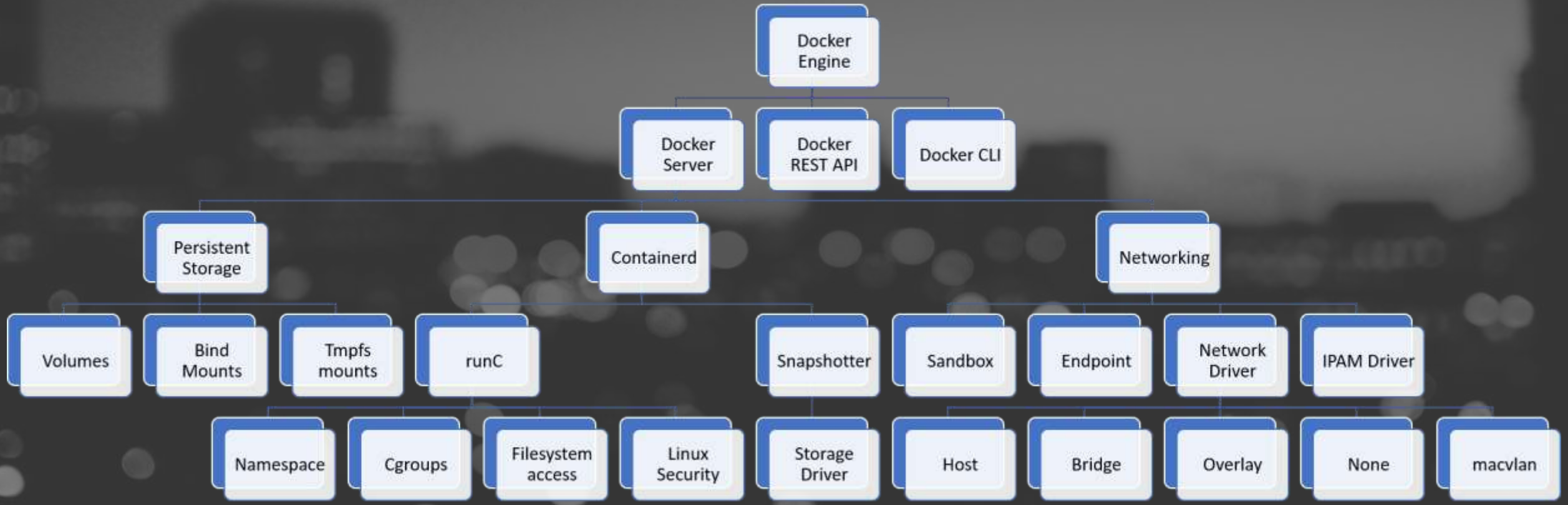
- Designed for security.
- Usable at large scale, in production, today.
- No dependency on the rest of the Docker platform: just the container runtime and nothing else.

Popular runC features include:

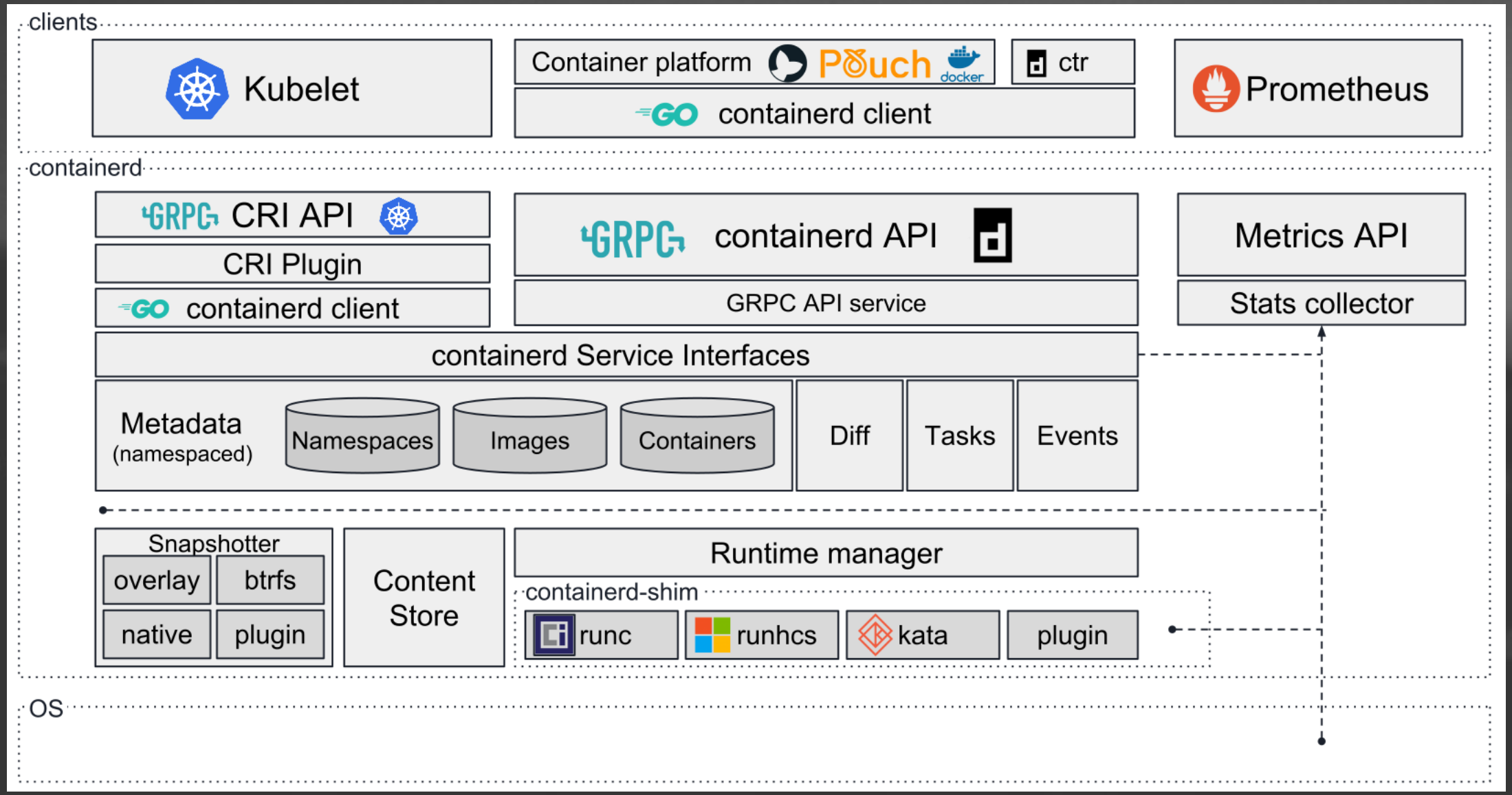
- Full support for Linux namespaces, including user namespaces
- Native support for all security features available in Linux: Selinux, Apparmor, seccomp, control groups, capability drop, pivot_root, uid/gid dropping etc. If Linux can do it, runC can do it.
- Native support for live migration, with the help of the CRIU team at Parallels
- Native support of Windows 10 containers is being contributed directly by Microsoft engineers
- Planned native support for Arm, Power, Sparc with direct participation and support from Arm, Intel, Qualcomm, IBM, and the entire hardware manufacturers ecosystem.

1 Container Runtime

Docker Architecture



1 Container Runtime



1 Container Runtime

docker-dev ›

Use of containerd-shim in docker-architecture

작성자 2명의 게시물 3개

☆ **Sridarshan Shetty** I'm still trying to warp my head around why containerd-shim is used in docker architecture. Can someone please help me with that? I noticed that, for exec and start, containerd communicates through shim, but for other operations (list, ps)

Michael Crosby

☆ **다른 수신자** srida...@gmail.com

한국어로 메시지 번역

The shim allows for daemonless containers. It basically sits as the parent of the container's process to facilitate a few things.

First it allows the runtimes, i.e. runc,to exit after it starts the container. This way we don't have to have the long running runtime processes for containers. When you start mysql you should only see the mysql process and the shim.

Second it keeps the STDIO and other fds open for the container incase containerd and/or docker both die. If the shim was not running then the parent side of the pipes or the TTY master would be closed and the container would exit.

Finally it allows the container's exit status to be reported back to a higher level tool like docker without having the be the actual parent of the container's process and do a wait4

I did a talk on this last week at dockercon US. You can see my slides here. https://github.com/crosbymichael/docker

Hopefully that will explain a little more about how containerd and the shim work.

- 받은메일 표시 -
- 받은메일 표시 -

--
You received this message because you are subscribed to the Google Groups "docker-dev" group.
To unsubscribe from this group and stop receiving emails from it, send an email to docke...@googlegroups.com.
For more options, visit https://groups.google.com/d/optout.

```
vagrant@ubuntu-xenial:~$ pstree -g
systemd(1)─┬─NetworkManager(1291)─┬─{gdbus}(1291)
           │                      └─{gmain}(1291)
           ├─VBoxService(1413)─┬─{automount}(1413)
           │                   ├─{control}(1413)
           │                   ├─{cpuhotplug}(1413)
           │                   ├─{memballoon}(1413)
           │                   ├─{timesync}(1413)
           │                   ├─{vminfo}(1413)
           │                   └─{vmstats}(1413)
           ├─accounts-daemon(1309)─┬─{gdbus}(1309)
           │                       └─{gmain}(1309)
           ├─acpid(1304)
           ├─agetty(1549)
           ├─agetty(1552)
           ├─at-spi-bus-laun(2726)─┬─dbus-daemon(2726)
           │                       ├─{dconf worker}(2726)
           │                       ├─{gdbus}(2726)
           │                       └─{gmain}(2726)
           ├─at-spi2-registr(2726)─┬─{gdbus}(2726)
           │                       └─{gmain}(2726)
           ├─atd(1266)
           ├─avahi-daemon(1268)───avahi-daemon(1268)
           ├─colord(2939)─┬─{gdbus}(2939)
           │              └─{gmain}(2939)
           ├─containerd(10997)─┬─containerd-shim(11583)─┬─bash(11770)
           │                   │                        ├─nginx(11601)───nginx(11601)
           │                   │                        └─{containerd-shim}(11583)
```

1 Pod

파드에 대해 이해하기

파드 는 쿠버네티스의 기본 구성 요소이다. 쿠버네티스 객체 모델 중 만들고 배포할 수 있는 가장 작고 간단한 단위이다. 파드는 클러스터에서의 Running 프로세스를 나타낸다.

파드는 애플리케이션 컨테이너(또는, 몇몇의 경우, 다중 컨테이너), 저장소 리소스, 특정 네트워크 IP 그리고, 컨테이너가 동작하기 위해 만들어진 옵션들을 캡슐화 한다. 파드는 배포의 단위를 말한다. 아마 단일 컨테이너로 구성되어 있거나, 강하게 결합되어 리소스를 공유하는 소수의 컨테이너로 구성되어 있는 *쿠버네티스에서의 애플리케이션 단일 인스턴스* 를 의미함.
Docker는 쿠버네티스 파드에서 사용되는 가장 대표적인 컨테이너 런타임이지만, 파
쿠버네티스 클러스터 내부의 파드는 주로 두 가지 방법으로 사용된다.

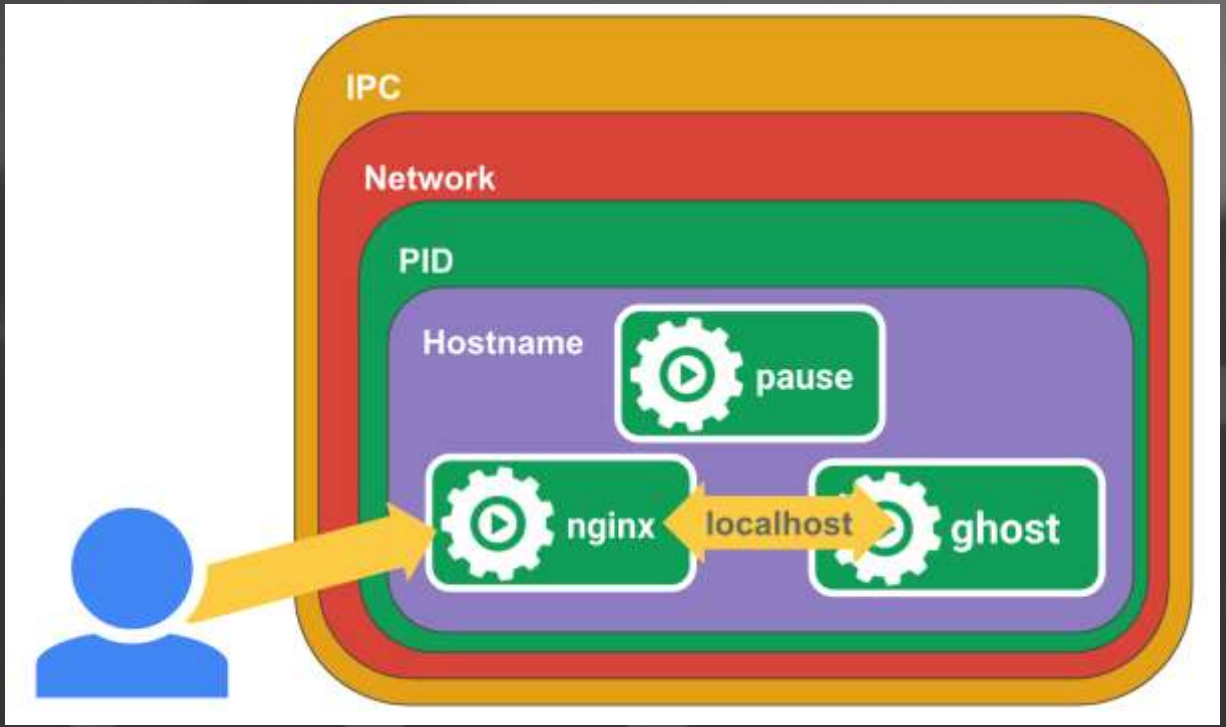
컨테이너간의

Hostname,
Namespace
IPC
Volume

- **단일 컨테이너만 동작하는 파드**. "단일 컨테이너 당 한 개의 파드" 모델은 쿠버너 가 단일 컨테이너를 감싸고 있다고 생각할 수 있으며, 쿠버네티스는 컨테이너가 아닌 파드를
- **함께 동작하는 작업이 필요한 다중 컨테이너가 동작하는 파드**. 아마 파드는 강하게 결합되어 있고 리소스 공유가 필요한 다중으로도 함께 배치된 컨테이너로 구성되어 있을 것이다. 이렇게 함께 배치되어 설치된 컨테이너는 단일 결합 서비스 단위일 것이다. 한 컨테이너는 공유 볼륨에서 퍼블릭으로 파일들을 옮기고, 동시에 분리되어 있는 "사이드카" 컨테이너는 그 파일들을 업데이트 하거나 복구한다. 파드는 이 컨테이너와 저장소 리소스들을 한 개의 관리 가능한 요소로 묶는다.

https://kubernetes.io/ko/docs/concepts/workloads/pods/pod-overview/

1 Pod



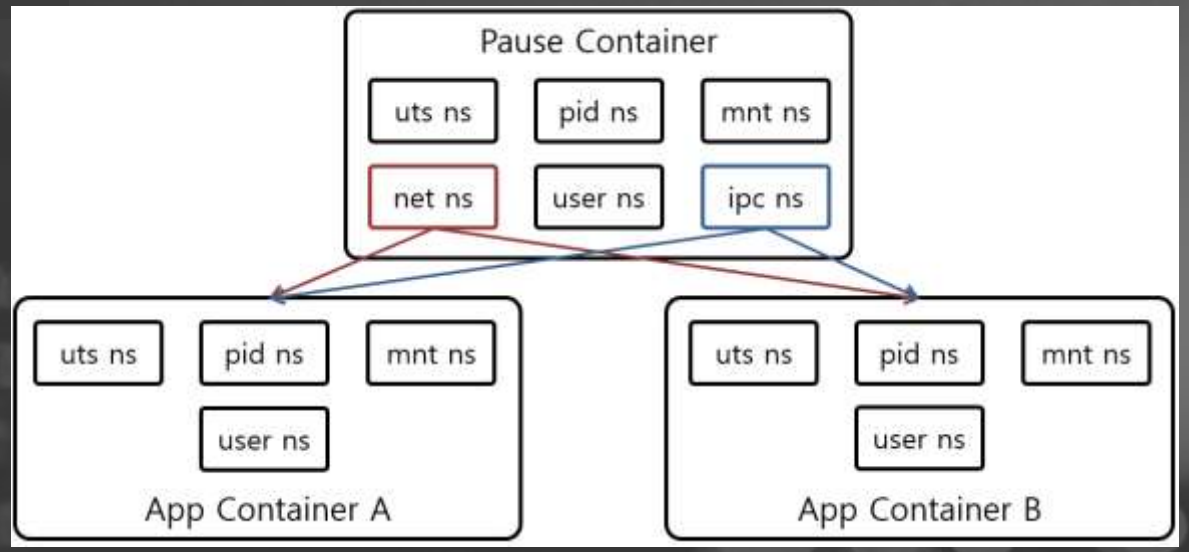
- Hostname
- Network
- IPC
- Volume 공유

- 특징
  - 단일 컨테이너 당 하나의 파드
  - 또는 다중 컨테이너 (Side - Car)
  - 스케일링의 단위

    - Deployment
    - StatefulSet
    - DamonSet

https://www.ianlewis.org/en/what-are-kubernetes-pods-anyway

1 Pod – pause container



Pod가 실행될 때 pause 라는 컨테이너가 먼저 실행되어 pause컨테이너의 네임스페이스를 Pod내부의 모든 컨테이너가 공유

Pod 내부의 컨테이너 프로세스가 좀비 프로세스가 되지 않도록 pause 컨테이너는 PID 1로 설정되어 모든 컨테이너 프로세스의 부모 프로세스가 되고, 컨테이너 프로세스의 문제가 발생해 SIGCHLD 시그널을 받으면 waitpid(해당 프로세스 ID)를 수행해서 중지된 컨테이너가 사용하던 자원을 회수하는 역할을 수행

https://blog.2dal.com/2018/03/28/kubernetes-01-pod/
https://ssup2.github.io/theory_analysis/Kubernetes_Pod/?fbclid=IwAR1IZtyjusnW1iX8C1s1c5QpQMigmiH-Y7pFDZ3dwtL44TJ8esDZqp-IXHg

1 Pod

- Pause Container
  - 평소에는 pause()
    - (시스널 수시까지 대기)
  - SIGTERM, SIGCHLD에 반응

```c
#ifndef VERSION
#define VERSION HEAD
#endif

static void sigdown(int signo) {
  psignal(signo, "Shutting down, got signal");
  exit(0);
}

static void sigreap(int signo) {
  while (waitpid(-1, NULL, WNOHANG) > 0)
    ;
}

int main(int argc, char **argv) {
  int i;
  for (i = 1; i < argc; ++i) {
    if (!strcasecmp(argv[i], "-v")) {
      printf("pause.c %s\n", VERSION_STRING(VERSION));
      return 0;
    }
  }

  if (getpid() != 1)
    /* Not an error because pause sees use outside of infra containers. */
    fprintf(stderr, "Warning: pause should be the first process\n");

  if (sigaction(SIGINT, &(struct sigaction){.sa_handler = sigdown}, NULL) < 0)
    return 1;
  if (sigaction(SIGTERM, &(struct sigaction){.sa_handler = sigdown}, NULL) < 0)
    return 2;
  if (sigaction(SIGCHLD, &(struct sigaction){.sa_handler = sigreap,
                                             .sa_flags = SA_NOCLDSTOP},
                NULL) < 0)
    return 3;

  for (;;)
    pause();
  fprintf(stderr, "Error: infinite loop terminated\n");
  return 42;
}
```

1 Pod

```
vagrant@k8snode2:~$ docker ps
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS               NAMES
7d8f799807bc        luksa/kubia            "node app.js"            3 minutes ago       Up 3 minutes                            k8s_kubia_kubia-rsmgp_default_a9585433-415e-11e9-9be4-026c728e02ff_9
904f7faf77cc        luksa/kubia            "node app.js"            3 minutes ago       Up 3 minutes                            k8s_kubia_kubia-4zgqc_default_3cd6e66c-45b5-11e9-88b1-026c728e02ff_4
3570e95cd23a        c264dff3420b           "td-agent"               3 minutes ago       Up 3 minutes                            k8s_fluentd-elasticsearch_fluentd-elasticsearch-5db68_kube-system_ce1c12d3-416d-11
e9-9be4-026c728e02ff_9
240edcc861da        k8s.gcr.io/pause:3.1   "/pause"                 3 minutes ago       Up 3 minutes                            k8s_POD_fluentd-elasticsearch-5db68_kube-system_ce1c12d3-416d-11e9-9be4-026c728e02
ff_27
f0951e8e1078        f9aed6605b81           "/dashboard --insecu…"   3 minutes ago       Up 3 minutes                            k8s_kubernetes-dashboard_kubernetes-dashboard-57df4db6b-fvmsm_kube-system_3cd67a7f
-45b5-11e9-88b1-026c728e02ff_4
41c394e7cdb3        luksa/kubia            "node app.js"            3 minutes ago       Up 3 minutes                            k8s_kubia_kubia-4zgqc_default_3cd6e66c-45b5-11e9-88b1-026c728e02ff_4
7c8ea344c40f        k8s.gcr.io/pause:3.1   "/pause"                 3 minutes ago       Up 3 minutes                            k8s_POD_kubia-rsmgp_default_a9585433-415e-11e9-9be4-026c728e02ff_23
96d205c9e586        c038466384ab           "apache2-foreground"     3 minutes ago       Up 3 minutes                            k8s_php-redis_frontend-jtsv5_default_3cdc5844-45b5-11e9-88b1-026c728e02ff_4
8668b3409301        4c7ea8709739           "/bin/sh -c 'node se…"   3 minutes ago       Up 3 minutes                            k8s_hello-world_hello-world-696b6b59bd-q8ddq_default_3cd7b0fd-45b5-11e9-88b1-026c7
28e02ff_4
e3c48c9d73c9        k8s.gcr.io/pause:3.1   "/pause"                 3 minutes ago       Up 3 minutes                            k8s_POD_kubia-68x6b_default_fd8996ca-415d-11e9-9be4-026c728e02ff_25
a47751e7b1e1        k8s.gcr.io/pause:3.1   "/pause"                 3 minutes ago       Up 3 minutes                            k8s_POD_kubernetes-dashboard-57df4db6b-fvmsm_kube-system_3cd67a7f-45b5-11e9-88b1-0
26c728e02ff_10
9a66d4d1c1a7        c038466384ab           "apache2-foreground"     3 minutes ago       Up 3 minutes                            k8s_php-redis_frontend-db2td_default_268d17ce-416a-11e9-9be4-026c728e02ff_9
40016f9861fd        k8s.gcr.io/pause:3.1   "/pause"                 3 minutes ago       Up 3 minutes                            k8s_POD_frontend-jtsv5_default_3cdc5844-45b5-11e9-88b1-026c728e02ff_11
4257f8fd150a        k8s.gcr.io/pause:3.1   "/pause"                 3 minutes ago       Up 3 minutes                            k8s_POD_hello-world-696b6b59bd-q8ddq_default_3cd7b0fd-45b5-11e9-88b1-026c728e02ff_
11
4185f856c41e        k8s.gcr.io/pause:3.1   "/pause"                 3 minutes ago       Up 3 minutes                            k8s_POD_frontend-db2td_default_268d17ce-416a-11e9-9be4-026c728e02ff_22
f29e4151d567        4c7ea8709739           "/bin/sh -c 'node se…"   3 minutes ago       Up 3 minutes                            k8s_hello-world_hello-world-696b6b59bd-gxpk9_default_e08633f4-13de-11e9-b61a-026c7
28e02ff_16
02374c97717e        k8s.gcr.io/pause:3.1   "/pause"                 3 minutes ago       Up 3 minutes                            k8s_POD_hello-world-696b6b59bd-gxpk9_default_e08633f4-13de-11e9-b61a-026c728e02ff_
59
7cb66507a724        c038466384ab           "apache2-foreground"     3 minutes ago       Up 3 minutes                            k8s_php-redis_frontend-gf999_default_268db974-416a-11e9-9be4-026c728e02ff_9
b70ec99a3c74        k8s.gcr.io/pause:3.1   "/pause"                 3 minutes ago       Up 3 minutes                            k8s_POD_kubia-4zgqc_default_3cd6e66c-45b5-11e9-88b1-026c728e02ff_11
a7c7c5047135        k8s.gcr.io/pause:3.1   "/pause"                 3 minutes ago       Up 3 minutes                            k8s_POD_frontend-gf999_default_268db974-416a-11e9-9be4-026c728e02ff_26
f2f59f8e0548        f0fad859c909           "/opt/bin/flanneld -…"   3 minutes ago       Up 3 minutes                            k8s_kube-flannel_kube-flannel-ds-amd64-q44d4_kube-system_befbe46d-13de-11e9-b61a-0
26c728e02ff_17
82dd99ce242f        fdb321fd30a0           "/usr/local/bin/kube…"   3 minutes ago       Up 3 minutes                            k8s_kube-proxy_kube-proxy-nrmvt_kube-system_befb3195-13de-11e9-b61a-026c728e02ff_1
7
6d96d6a675c9        k8s.gcr.io/pause:3.1   "/pause"                 3 minutes ago       Up 3 minutes                            k8s_POD_kube-flannel-ds-amd64-q44d4_kube-system_befbe46d-13de-11e9-b61a-026c728e02
ff_17
8d9f6f712291        k8s.gcr.io/pause:3.1   "/pause"                 3 minutes ago       Up 3 minutes                            k8s_POD_kube-proxy-nrmvt_kube-system_befb3195-13de-11e9-b61a-026c728e02ff_17
vagrant@k8snode2:~$ docker --help
```

1 Pod

pause와 app container가 net namespace 공유하는 것을 증명 방법

- #docker ps : 전체 프로세스로 container_id 을 찾고
- #docker inspect —format '{{.State.Pid}}' <container_id> : pid를 찾고
- #cd /proc/pid/ns : pid의 namespace 를 보면
- #lsns -t net : 으로 찾은 net-namespace가 동일한것을 볼 수 있다.

```
[root@k8s-worker-ad3-0 ns]# lsns -t net
        NS TYPE NPROCS   PID USER  COMMAND
4026531993 net     264     1 root  /usr/lib/systemd/systemd --switched-root --system --deseria
4026532231 net       2 17187 root  /pause
4026532311 net       2 17898 root  /pause
4026532386 net       2 18251 root  /pause
4026532458 net       4 18293 root  /pause
4026532535 net       7 18950 root  /pause
4026532608 net       5  6644 10001 sh
[root@k8s-worker-ad3-0 ns]# docker inspect --format '{{.State.Pid}}' e919f0b5a0c1
19353
[root@k8s-worker-ad3-0 ns]# cd ../
[root@k8s-worker-ad3-0 17140]# cd ../
[root@k8s-worker-ad3-0 proc]# cd 19353
[root@k8s-worker-ad3-0 19353]# ls
attr        clear_refs       cpuset   fd       limits     mem         net         oom_score
autogroup   cmdline          cwd      fdinfo   loginuid   mountinfo   ns          oom_score_adj
auxv        comm             environ  gid_map  map_files  mounts      numa_maps   pagemap
cgroup      coredump_filter  exe      io       maps       mountstats  oom_adj     personality
[root@k8s-worker-ad3-0 19353]# cd ns
[root@k8s-worker-ad3-0 ns]# ls
cgroup  ipc  mnt  net  pid  pid_for_children  user  uts
[root@k8s-worker-ad3-0 ns]# ls -al
합계 0
dr-x--x--x. 2 10001 10001 0  4월  15 05:26 .
dr-xr-xr-x. 9 10001 10001 0  4월  15 02:16 ..
lrwxrwxrwx. 1 10001 10001 0  4월  15 14:25 cgroup -> cgroup:[4026531835]
lrwxrwxrwx. 1 10001 10001 0  4월  15 09:58 ipc -> ipc:[4026532420]
lrwxrwxrwx. 1 10001 10001 0  4월  15 05:26 mnt -> mnt:[4026532683]
lrwxrwxrwx. 1 10001 10001 0  4월  15 09:58 net -> net:[4026532458]
lrwxrwxrwx. 1 10001 10001 0  4월  15 05:26 pid -> pid:[4026532685]
lrwxrwxrwx. 1 10001 10001 0  4월  15 14:25 pid_for_children -> pid:[4026532685]
lrwxrwxrwx. 1 10001 10001 0  4월  15 09:58 user -> user:[4026531837]
lrwxrwxrwx. 1 10001 10001 0  4월  15 05:26 uts -> uts:[4026532684]
```

Node Component

- Kube-proxy

1 Kube-proxy

kube-proxy는 호스트 상에서 네트워크 규칙을 유지하고 연결에 대한 포워딩을 수행함으로서 쿠버네티스 서비스 추상화가 가능하도록 해준다.

1) DaemonSet

```
kih@IHMac-2 ~ kb get daemonset -n kube-system
NAME              DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
kube-flannel-ds   4         4         4       4            4           beta.kubernetes.io/arch=amd64
kube-proxy        4         4         4       4            4           <none>
kih@IHMac-2 ~ kb describe ds kube-proxy -n kube-system
Name:           kube-proxy
Selector:       k8s-app=kube-proxy
Node-Selector:  <none>
Labels:         k8s-app=kube-proxy
Annotations:    deprecated.daemonset.template.generation: 1
Desired Number of Nodes Scheduled: 4
Current Number of Nodes Scheduled: 4
Number of Nodes Scheduled with Up-to-date Pods: 4
Number of Nodes Scheduled with Available Pods: 4
Number of Nodes Misscheduled: 0
Pods Status:  4 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:           k8s-app=kube-proxy
  Annotations:      scheduler.alpha.kubernetes.io/critical-pod:
  Service Account:  kube-proxy
  Containers:
   kube-proxy:
    Image:      container-registry.oracle.com/kubernetes/kube-proxy:v1.12.7
    Port:       <none>
    Host Port:  <none>
```

1 Kube-proxy

kube-proxy는 호스트 상에서 네트워크 규칙을 유지하고 연결에 대한 포워딩을 수행함으로서 쿠버네티스 서비스 추상화가 가능하도록 해준다.

1) DaemonSet
2) DaemonSet도 Pod다

```
[root@inho-k8s-worker1-setup opc]# docker ps | grep proxy
e74839cc3b80        istio/proxyv2                                      "/usr/local/bin/pilo…"   25 hours ago
                       k8s_istio-proxy_kubia-k922j_default_23097bb6-6193-11e9-a534-0000170298b5_0
c77e248d3394        38e58c880d10                                       "/usr/local/bin/kube…"   25 hours ago
                       k8s_kube-proxy_kube-proxy-wpt6q_kube-system_2a843bc1-605c-11e9-a725-0000170298b5_1
c262201f7915        container-registry.oracle.com/kubernetes/pause:3.1   "/pause"                 25 hours ago
                       k8s_POD_kube-proxy-wpt6q_kube-system_2a843bc1-605c-11e9-a725-0000170298b5_1
```

```
[root@inho-k8s-worker1-setup opc]# docker images
REPOSITORY                                                       TAG                 IMAGE ID            CREAT
istio/proxyv2                                                    1.1.3               70a4a6110ae0        6 day
istio/proxy_init                                                 1.1.3               fa8a07a7f877        6 day
mongo                                                            latest              b1c8142b3efc        7 day
container-registry.oracle.com/kubernetes/kube-proxy              v1.12.7             38e58c880d10        13 da
busybox                                                          latest              af2f74c517aa        2 wee
container-registry.oracle.com/kubernetes/kubernetes-dashboard-amd64   v1.10.1             65d210ea5183        2 mon
container-registry.oracle.com/kubernetes/coredns                 1.2.2               eeb7f9f4edc2        2 mon
B
container-registry.oracle.com/kubernetes/flannel                 v0.10.0             0a95ca9313eb        2 mon
container-registry.oracle.com/kubernetes/pause                   3.1                 641963956871        8 mon
```

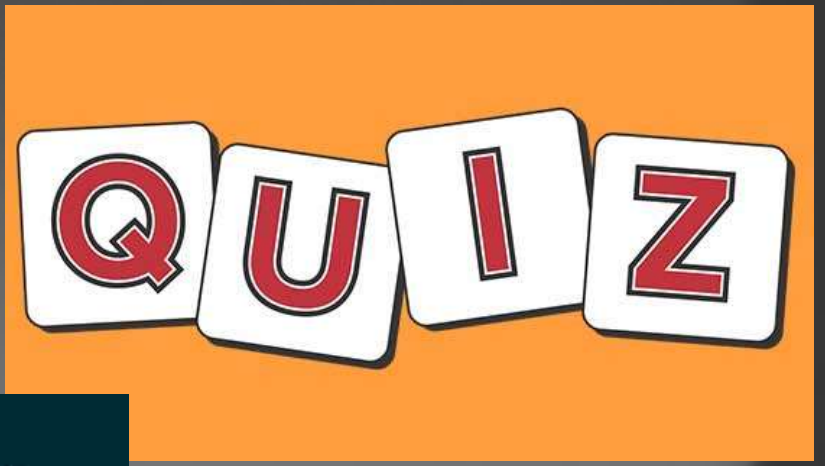
1 Kube-proxy

- Kube proxy Listen Port
  - NodePort

```
[root@inho-k8s-worker2-setup opc]# netstat -ntpl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:10248         0.0.0.0:*               LISTEN      4452/kubelet
tcp        0      0 127.0.0.1:10249         0.0.0.0:*               LISTEN      16568/kube-proxy
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN      1/systemd
tcp        0      0 127.0.0.1:46163         0.0.0.0:*               LISTEN      4452/kubelet
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      5157/sshd
tcp        0      0 127.0.0.1:39011         0.0.0.0:*               LISTEN      5110/containerd
tcp6       0      0 :::10250                :::*                    LISTEN      4452/kubelet
tcp6       0      0 :::111                  :::*                    LISTEN      4438/rpcbind
tcp6       0      0 :::10256                :::*                    LISTEN      16568/kube-proxy
tcp6       0      0 :::30001                :::*                    LISTEN      16568/kube-proxy
tcp6       0      0 :::22                   :::*                    LISTEN      5157/sshd
[root@inho-k8s-worker2-setup opc]# curl http://127.0.0.1:10248/healthz
ok[root@inho-k8s-worker2-setup opc]#
```

1 Kube-proxy

- 왜 다양한 서비스가 Listen하고 있는 포트가 netstat –ntlp 에서 잡히지 않는 걸까?




```
[root@inho-k8s-worker2-setup opc]# netstat -ntpl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:10248         0.0.0.0:*               LISTEN      4452/kubelet
tcp        0      0 127.0.0.1:10249         0.0.0.0:*               LISTEN      16568/kube-proxy
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN      1/systemd
tcp        0      0 127.0.0.1:46163         0.0.0.0:*               LISTEN      4452/kubelet
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      5157/sshd
tcp        0      0 127.0.0.1:39011         0.0.0.0:*               LISTEN      5110/containerd
tcp6       0      0 :::10250                :::*                    LISTEN      4452/kubelet
tcp6       0      0 :::111                  :::*                    LISTEN      4438/rpcbind
tcp6       0      0 :::10256                :::*                    LISTEN      16568/kube-proxy
tcp6       0      0 :::30001                :::*
tcp6       0      0 :::22                   :::*
```

```
kih@IHMac-2 ~ kb get svc -n sock-shop
NAME           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
carts          ClusterIP   10.111.9.53      <none>        80/TCP           25h
carts-db       ClusterIP   10.111.15.248    <none>        27017/TCP        25h
catalogue      ClusterIP   10.109.198.55    <none>        80/TCP           25h
catalogue-db   ClusterIP   10.100.164.170   <none>        3306/TCP         25h
front-end      NodePort    10.99.4.188      <none>        80:30001/TCP     25h
orders         ClusterIP   10.111.90.77     <none>        80/TCP           25h
orders-db      ClusterIP   10.101.231.237   <none>        27017/TCP        25h
payment        ClusterIP   10.105.157.220   <none>        80/TCP           25h
queue-master   ClusterIP   10.98.196.234    <none>        80/TCP           25h
rabbitmq       ClusterIP   10.105.237.245   <none>        5672/TCP         25h
shipping       ClusterIP   10.103.187.205   <none>        80/TCP           25h
user           ClusterIP   10.98.138.49     <none>        80/TCP           25h
user-db        ClusterIP   10.97.70.63      <none>        27017/TCP        25h
```

1 Kube-proxy

- 이전 페이지에서 kube proxy가 listen하고 있는 모든 포트에 대해서 iptable의 nat를 조사해봤다.
- 왜 NodePort로 오픈한 서비스만 나오고, 나머지 서비스는 netstat에 잡히지 않을까??
  - 답은 root name space에 대해서만 netstat 가 가능하기 때문이다. Container가 열고 있는 포트는 컨테이너 netns에서만 보이니까!~ → 이건 뒤에 network 파트에서 계속

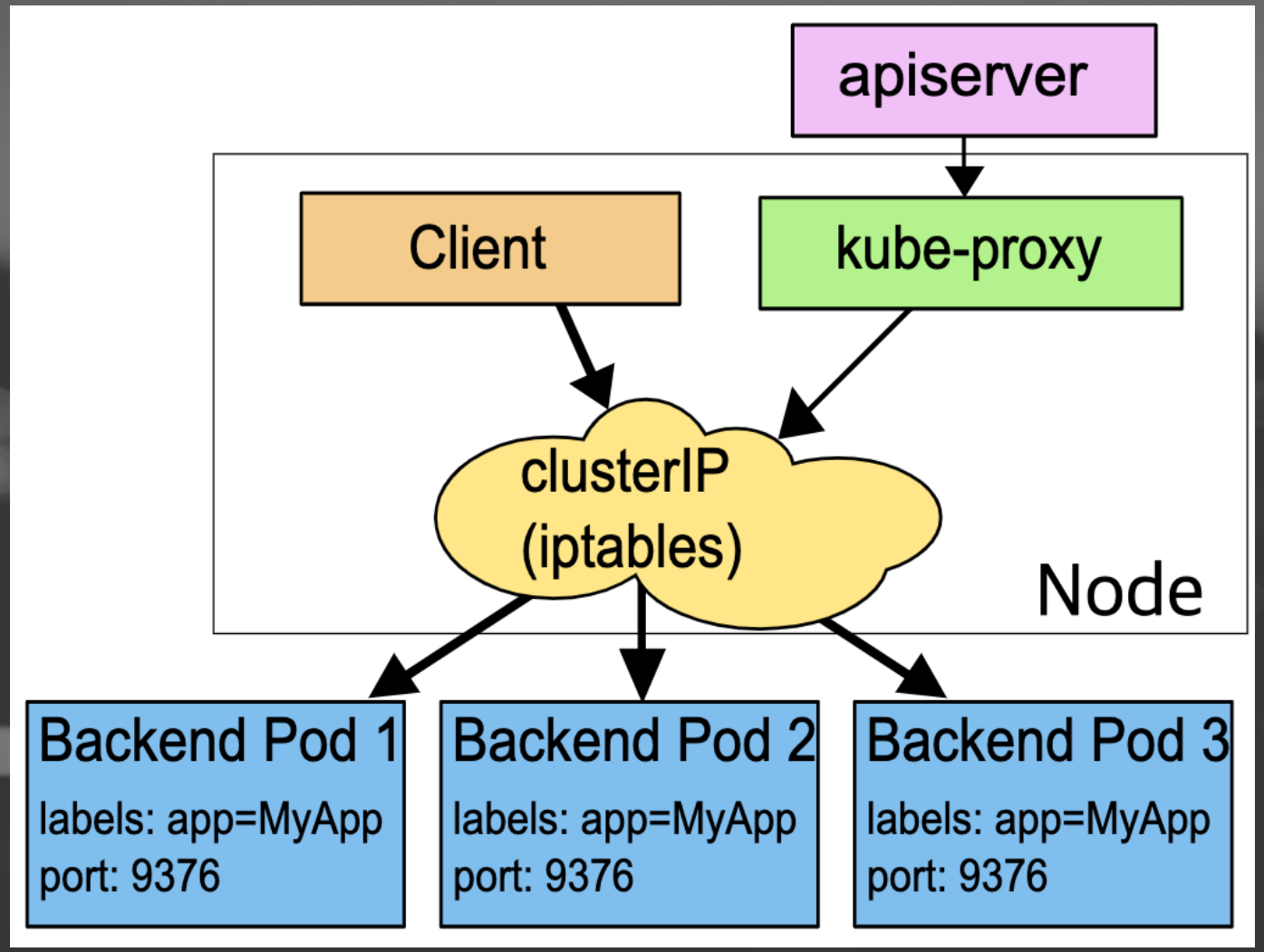
```
kih@IHMac-2 ~ kb get svc -n sock-shop
NAME           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
carts          ClusterIP   10.111.9.53      <none>        80/TCP         25h
carts-db       ClusterIP   10.111.15.248    <none>        27017/TCP      25h
catalogue      ClusterIP   10.109.198.55    <none>        80/TCP         25h
catalogue-db   ClusterIP   10.100.164.17
front-end      NodePort    10.99.4.188
orders         ClusterIP   10.111.90.77
orders-db      ClusterIP   10.101.231.23
payment        ClusterIP   10.105.157.22
queue-master   ClusterIP   10.98.196.234
rabbitmq       ClusterIP   10.105.237.24
shipping       ClusterIP   10.103.187.20
user           ClusterIP   10.98.138.49
user-db        ClusterIP   10.97.70.63
```

```
[root@inho-k8s-worker1-setup opc]# iptables -t nat -S | grep '31494\|31622\|31400\|31917\|30001\|31985\|31380\
-A KUBE-NODEPORTS -p tcp -m comment --comment "istio-system/istio-ingressgateway:status-port" -m tcp --dport 31
-A KUBE-NODEPORTS -p tcp -m comment --comment "istio-system/istio-ingressgateway:status-port" -m tcp --dport 31
-A KUBE-NODEPORTS -p tcp -m comment --comment "istio-system/istio-ingressgateway:https" -m tcp --dport 31390 -j
-A KUBE-NODEPORTS -p tcp -m comment --comment "istio-system/istio-ingressgateway:https" -m tcp --dport 31390 -j
-A KUBE-NODEPORTS -p tcp -m comment --comment "istio-system/istio-ingressgateway:https-grafana" -m tcp --dport
-A KUBE-NODEPORTS -p tcp -m comment --comment "istio-system/istio-ingressgateway:https-grafana" -m tcp --dport
-A KUBE-NODEPORTS -p tcp -m comment --comment "sock-shop/front-end:" -m tcp --dport 30001 -j KUBE-MARK-MASQ
-A KUBE-NODEPORTS -p tcp -m comment --comment "sock-shop/front-end:" -m tcp --dport 30001 -j KUBE-SVC-LFMD53S3
-A KUBE-NODEPORTS -p tcp -m comment --comment "istio-system/istio-ingressgateway:http2" -m tcp --dport 31380 -j
-A KUBE-NODEPORTS -p tcp -m comment --comment "istio-system/istio-ingressgateway:http2" -m tcp --dport 31380 -j
-A KUBE-NODEPORTS -p tcp -m comment --comment "istio-system/istio-ingressgateway:tcp" -m tcp --dport 31400 -j K
-A KUBE-NODEPORTS -p tcp -m comment --comment "istio-system/istio-ingressgateway:tcp" -m tcp --dport 31400 -j K
-A KUBE-NODEPORTS -p tcp -m comment --comment "istio-system/istio-ingressgateway:https-prometheus" -m tcp --dpo
-A KUBE-NODEPORTS -p tcp -m comment --comment "istio-system/istio-ingressgateway:https-prometheus" -m tcp --dpo
-A KUBE-NODEPORTS -p tcp -m comment --comment "istio-system/istio-ingressgateway:https-kiali" -m tcp --dport 31
-A KUBE-NODEPORTS -p tcp -m comment --comment "istio-system/istio-ingressgateway:https-kiali" -m tcp --dport 31
-A KUBE-NODEPORTS -p tcp -m comment --comment "istio-system/istio-ingressgateway:tls" -m tcp --dport 31985 -j K
-A KUBE-NODEPORTS -p tcp -m comment --comment "istio-system/istio-ingressgateway:tls" -m tcp --dport 31985 -j K
-A KUBE-SEP-5SKKMOWDDPRPEXT6 -p tcp -m tcp -j DNAT --to-destination 10.244.1.17:31400
-A KUBE-SERVICES ! -s 10.244.0.0/16 -d 10.108.57.135/32 -p tcp -m comment --comment "istio-system/istio-ingress
```

1 Kube-proxy

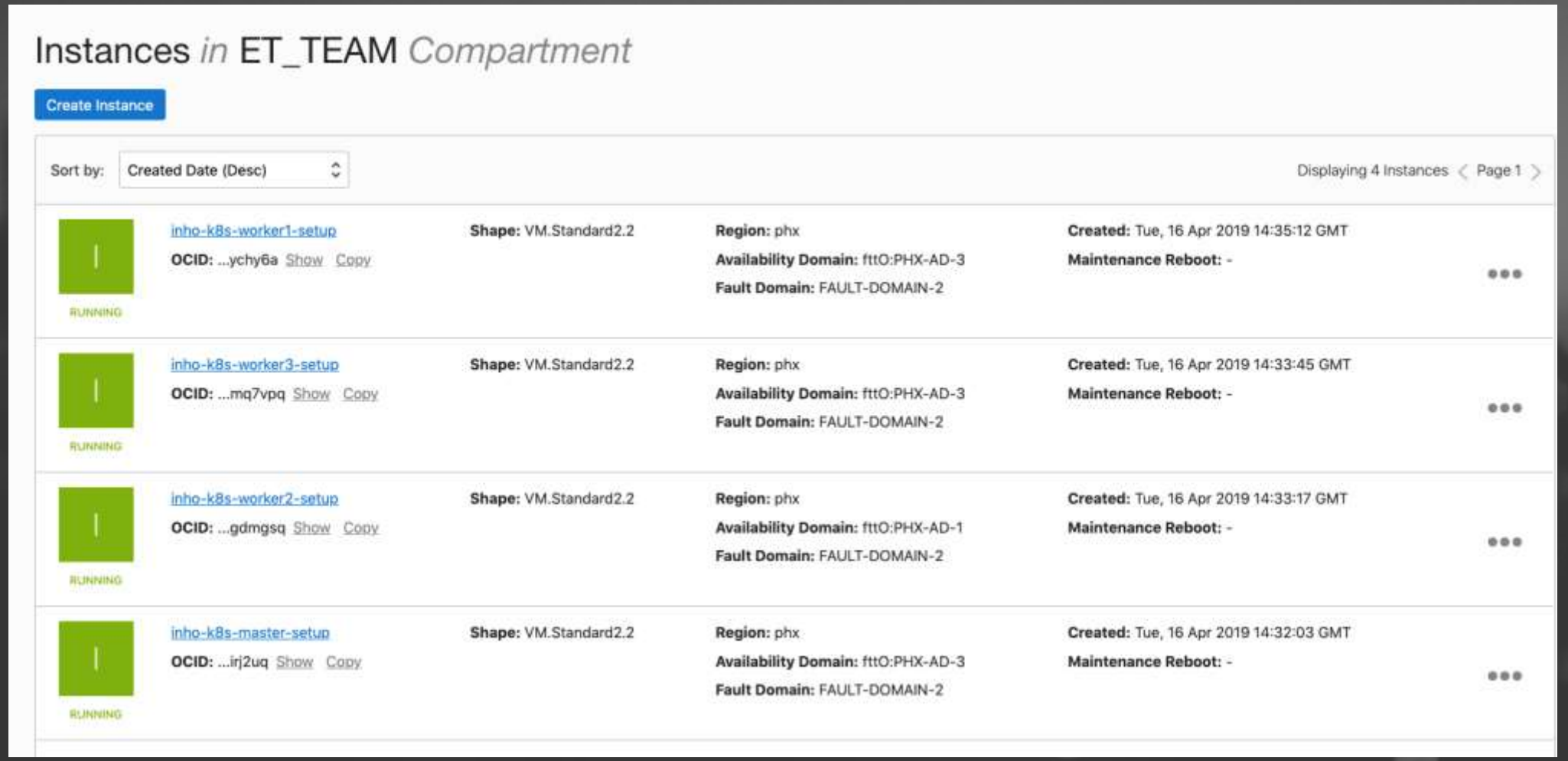
kube-proxy는 호스트 상에서 네트워크 규칙을 유지하고 연결에 대한 포워딩을 수행함으로서 쿠버네티스

1) proxy-mode : userspace
2) Proxy-mode : iptables
3) Proxy-mode : IPVS

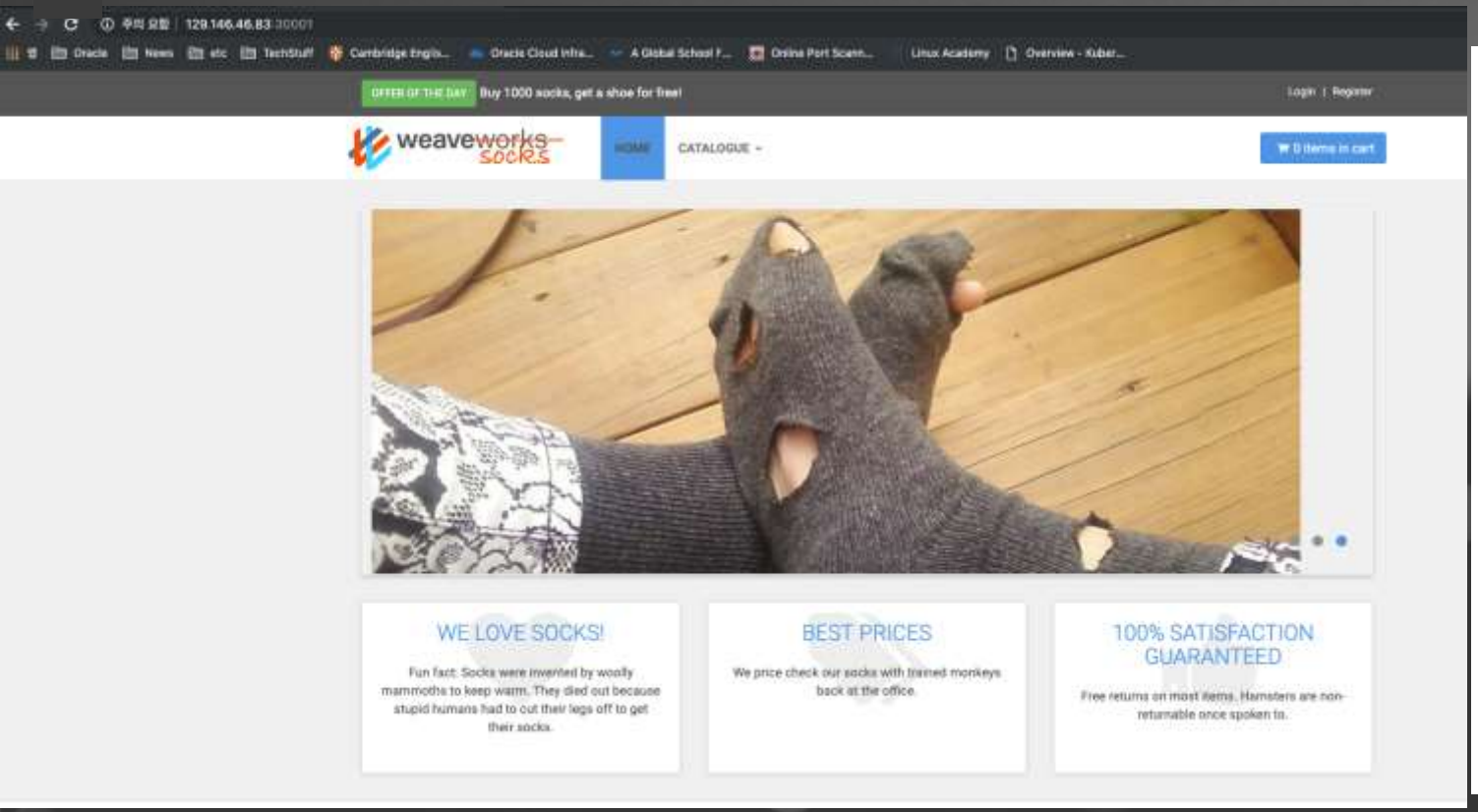


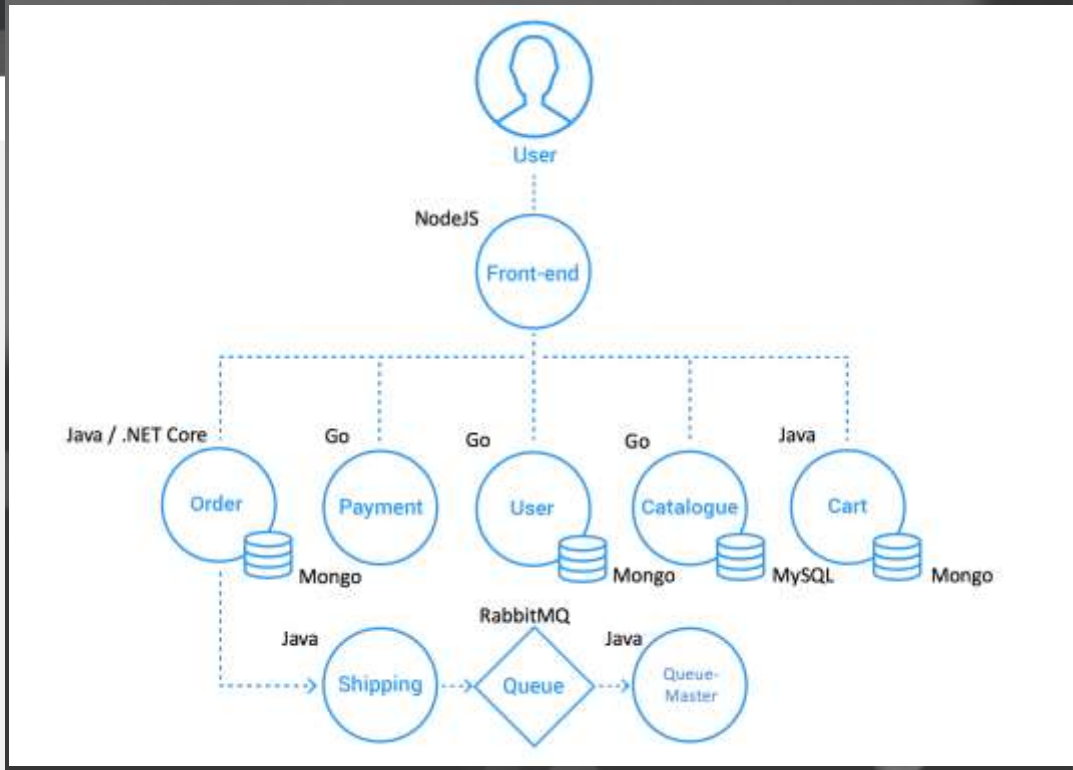
Kubernetes Network

1 환경

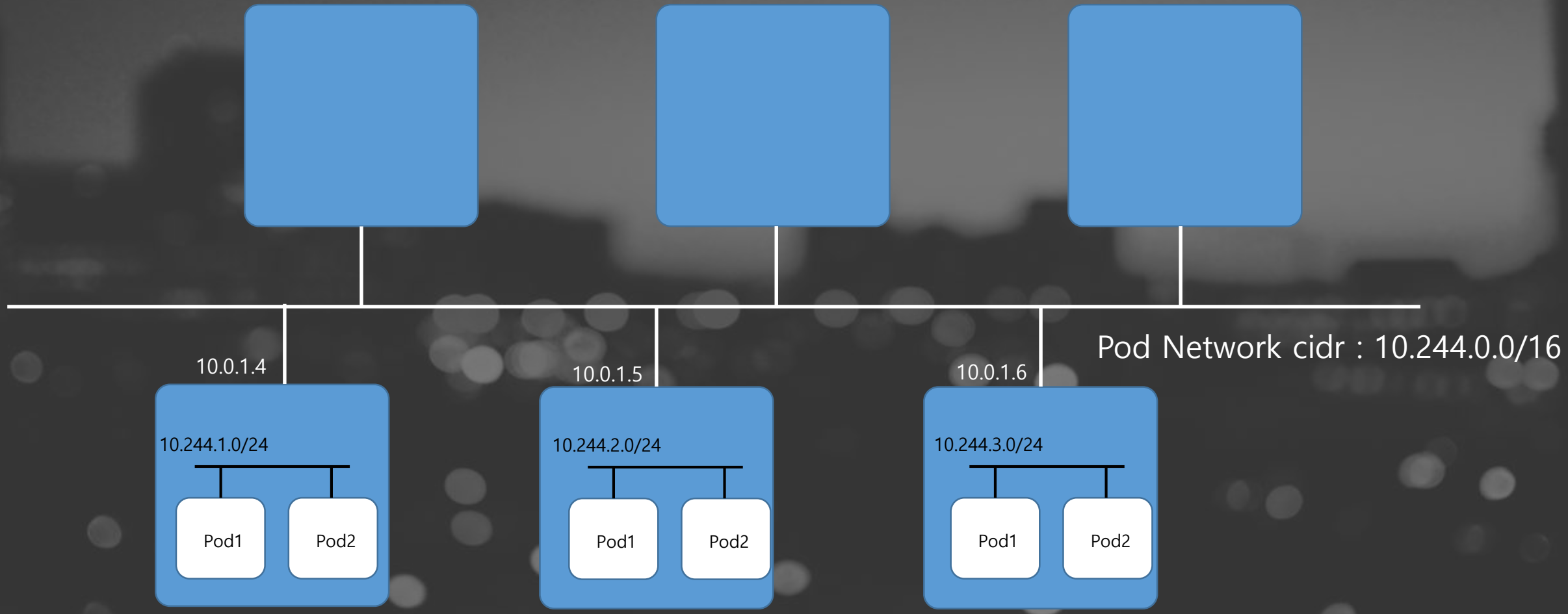

Instances in ET_TEAM Compartment
Create Instance
Sort by: Created Date (Desc)
Displaying 4 Instances
Page 1
inho-k8s-worker1-setup
OCID: ...ychy6a Show Copy
Shape: VM.Standard2.2
Region: phx
Availability Domain: fttO:PHX-AD-3
Fault Domain: FAULT-DOMAIN-2
Created: Tue, 16 Apr 2019 14:35:12 GMT
Maintenance Reboot: -
RUNNING
inho-k8s-worker3-setup
OCID: ...mq7vpq Show Copy
Shape: VM.Standard2.2
Region: phx
Availability Domain: fttO:PHX-AD-3
Fault Domain: FAULT-DOMAIN-2
Created: Tue, 16 Apr 2019 14:33:45 GMT
Maintenance Reboot: -
RUNNING
inho-k8s-worker2-setup
OCID: ...gdmgsq Show Copy
Shape: VM.Standard2.2
Region: phx
Availability Domain: fttO:PHX-AD-1
Fault Domain: FAULT-DOMAIN-2
Created: Tue, 16 Apr 2019 14:33:17 GMT
Maintenance Reboot: -
RUNNING
inho-k8s-master-setup
OCID: ...irj2uq Show Copy
Shape: VM.Standard2.2
Region: phx
Availability Domain: fttO:PHX-AD-3
Fault Domain: FAULT-DOMAIN-2
Created: Tue, 16 Apr 2019 14:32:03 GMT
Maintenance Reboot: -
RUNNING

1 환경


128.146.46.83:30001
OFFER OF THE DAY
Buy 1000 socks, get a shoe for free!
Login | Register
weaveworks
socks
HOME
CATALOGUE
0 items in cart
WE LOVE SOCKS!
Fun fact: Socks were invented by woolly mammoths to keep warm. They died out because stupid humans had to cut their legs off to get their socks.
BEST PRICES
We price check our socks with trained monkeys back at the office.
100% SATISFACTION GUARANTEED
Free returns on most items. Hamsters are non-returnable once spoken to.

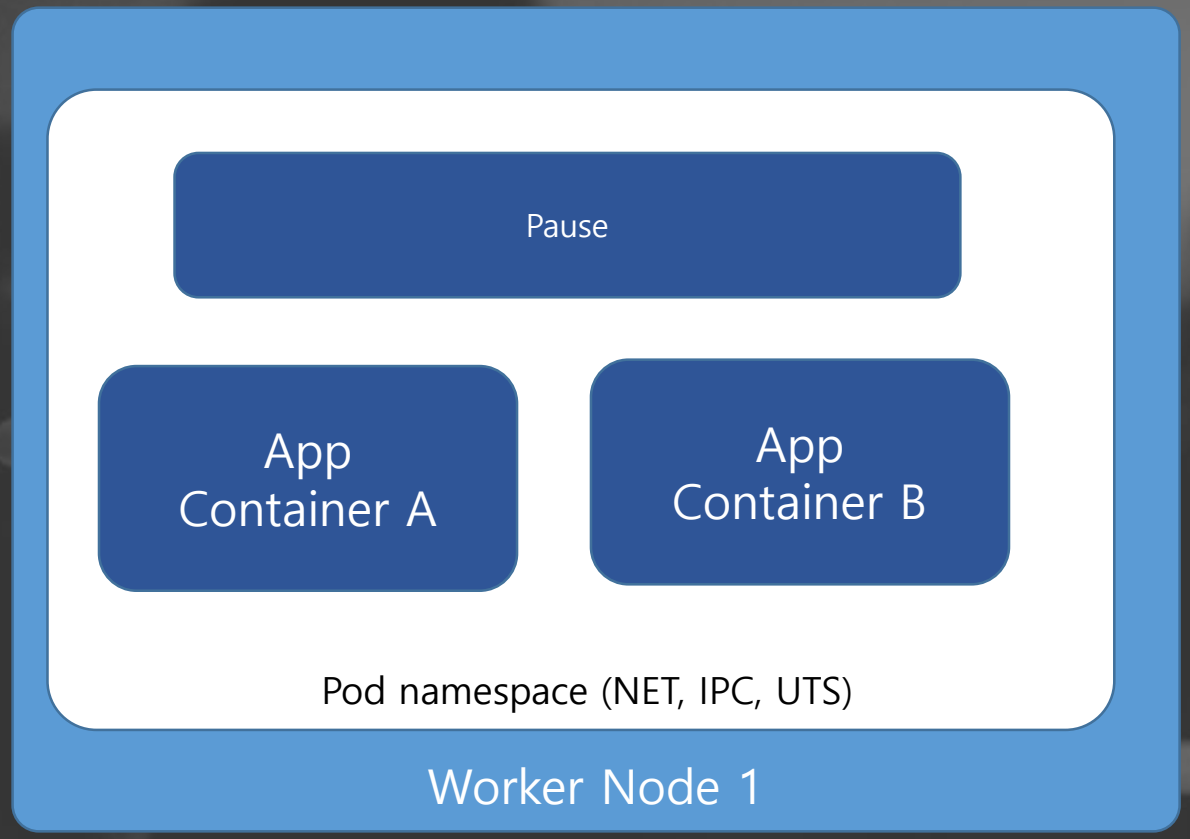

User
NodeJS
Front-end
Java / .NET Core
Order
Mongo
Go
Payment
Go
User
Mongo
Go
Catalogue
MySQL
Java
Cart
Mongo
RabbitMQ
Java
Shipping
Queue
Java
Queue-Master

1 Container-to-Container Networking



1 Container-to-Container Networking

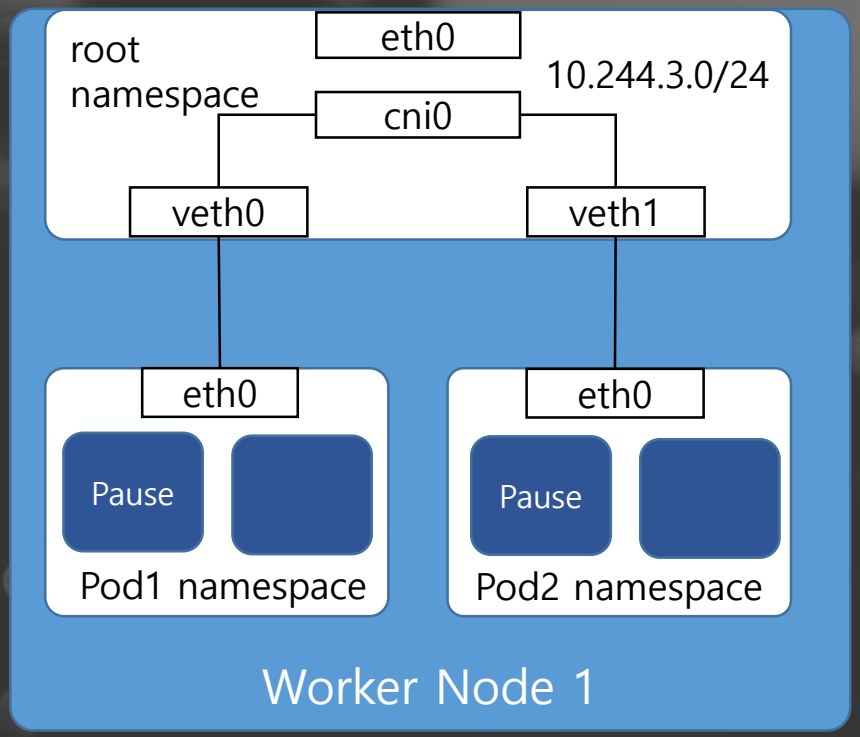
Pause Container 의 Namespace(NET, IPC) 공유
- Localhost 를 통한 통신 가능

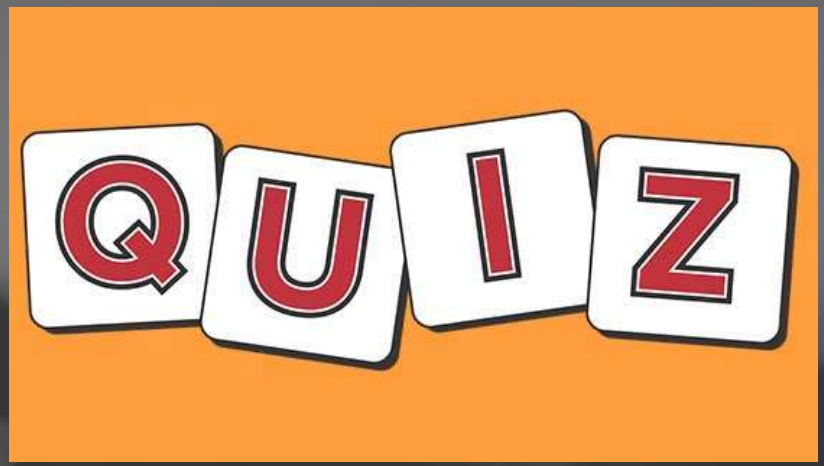



```
[root@k8s-worker-ad3-0 ns]# lsns -t net
        NS TYPE NPROCS   PID USER  COMMAND
4026531993 net     264     1 root  /usr/lib/systemd/systemd --switched-root --syste
4026532231 net       2 17187 root  /pause
4026532311 net       2 17898 root  /pause
4026532386 net       2 18251 root  /pause
4026532458 net       4 18293 root  /pause
4026532535 net       7 18950 root  /pause
4026532608 net       5  6644 10001 sh
[root@k8s-worker-ad3-0 ns]# docker inspect --format '{{.State.Pid}}' e919f0b5a0c1
19353
[root@k8s-worker-ad3-0 ns]# cd ../
[root@k8s-worker-ad3-0 17140]# cd ../
[root@k8s-worker-ad3-0 proc]# cd 19353
[root@k8s-worker-ad3-0 19353]# ls
attr        clear_refs       cpuset  fd       limits     mem         net         oom
autogroup   cmdline          cwd     fdinfo   loginuid   mountinfo   ns          oom
auxv        comm             environ gid_map  map_files  mounts      numa_maps   pag
cgroup      coredump_filter  exe     io       maps       mountstats  oom_adj     per
[root@k8s-worker-ad3-0 19353]# cd ns
[root@k8s-worker-ad3-0 ns]# ls
cgroup  ipc  mnt  net  pid  pid_for_children  user  uts
[root@k8s-worker-ad3-0 ns]# ls -al
합계 0
dr-x--x--x. 2 10001 10001 0  4월 15 05:26 .
dr-xr-xr-x. 9 10001 10001 0  4월 15 02:16 ..
lrwxrwxrwx. 1 10001 10001 0  4월 15 14:25 cgroup -> cgroup:[4026531835]
lrwxrwxrwx. 1 10001 10001 0  4월 15 09:58 ipc -> ipc:[4026532420]
lrwxrwxrwx. 1 10001 10001 0  4월 15 05:26 mnt -> mnt:[4026532683]
lrwxrwxrwx. 1 10001 10001 0  4월 15 09:58 net -> net:[4026532458]
lrwxrwxrwx. 1 10001 10001 0  4월 15 05:26 pid -> pid:[4026532685]
lrwxrwxrwx. 1 10001 10001 0  4월 15 14:25 pid_for_children -> pid:[4026532685]
lrwxrwxrwx. 1 10001 10001 0  4월 15 09:58 user -> user:[4026531837]
lrwxrwxrwx. 1 10001 10001 0  4월 15 05:26 uts -> uts:[4026532684]
```

1 Pod-to-Pod in a Same Node

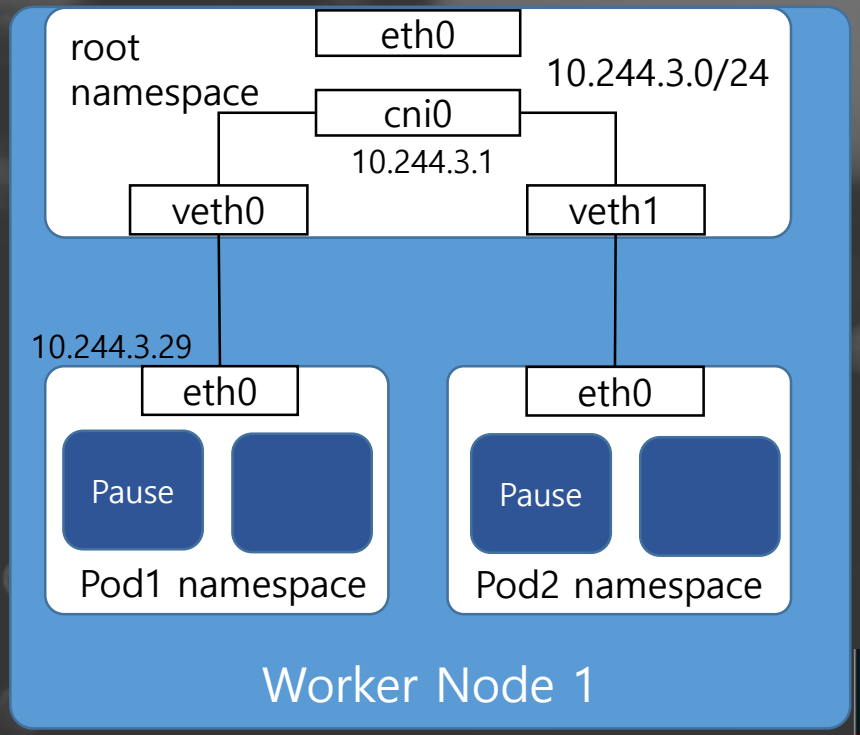
Destination IP는 알겠는데 통신하려는 상대방 MAC Address를 모를 때 사용하는 프로토콜?





1 Pod-to-Pod in a Same Node

Pause Container 의 Namespace(NET, IPC) 공유
- Localhost 를 통한 통신 가능



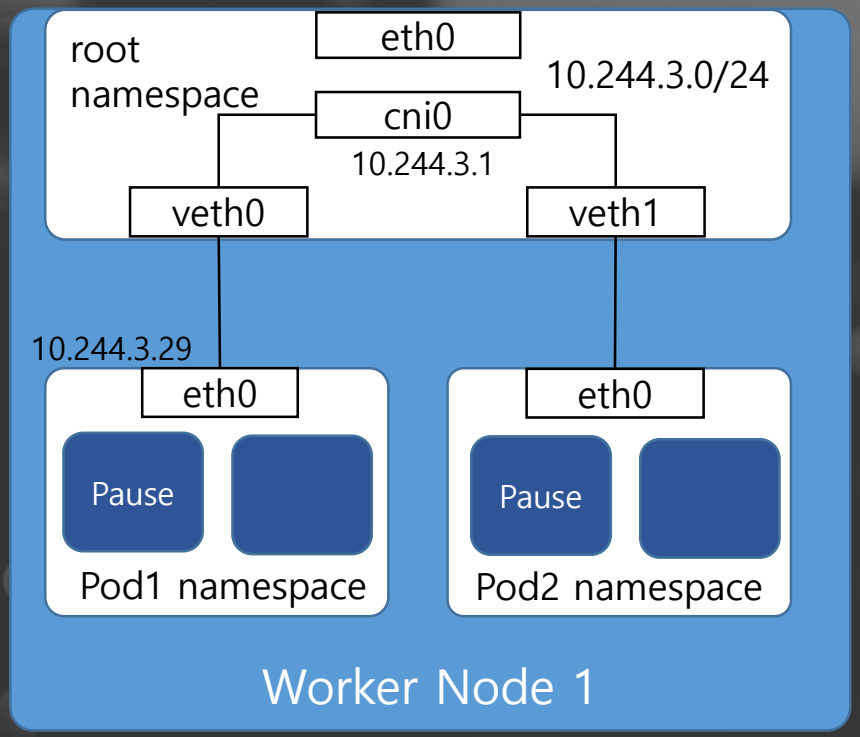
Node : k8s_worker3
Container : order

```
[root@inho-k8s-worker3-setup opc]# docker exec -it 9132d180b913 ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 10
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
3: eth0@if20: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 8950 qdisc noque
    link/ether ea:e3:e6:bf:54:07 brd ff:ff:ff:ff:ff:ff
    inet 10.244.3.29/24 scope global eth0
       valid_lft forever preferred_lft forever
```

```
[root@inho-k8s-worker3-setup opc]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN g
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP
    link/ether 02:00:17:05:f1:a9 brd ff:ff:ff:ff:ff:ff
19: veth926e89bf@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 8950 qdis
    link/ether 8e:3c:e7:6a:d8:8f brd ff:ff:ff:ff:ff:ff link-netnsid 1
20: veth4ae0ee7b@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 8950 qdis
    link/ether 32:0c:f7:d2:ed:16 brd ff:ff:ff:ff:ff:ff link-netnsid 1
```

1 Pod-to-Pod in a Same Node

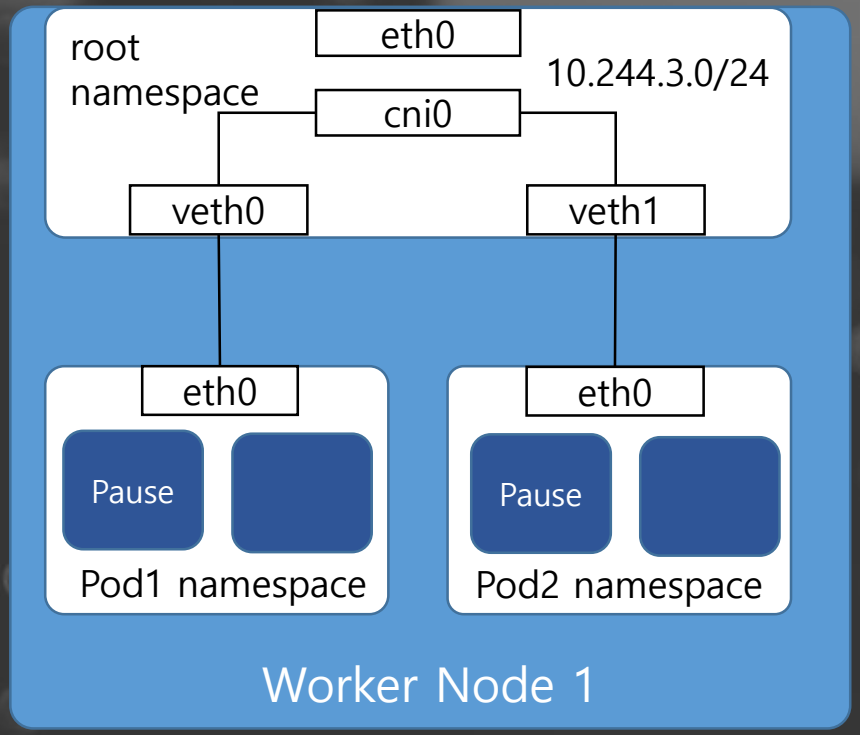
- Pod 속 routing table




```
[root@inho-k8s-worker3-setup opc]# docker exec -it 9132d180b913 sh
/usr/src/app $ ls
app.jar
/usr/src/app $ ip route
default via 10.244.3.1 dev eth0
10.244.0.0/16 via 10.244.3.1 dev eth0
10.244.3.0/24 dev eth0  src 10.244.3.29
/usr/src/app $
```

1 Pod-to-Pod in a Same Node

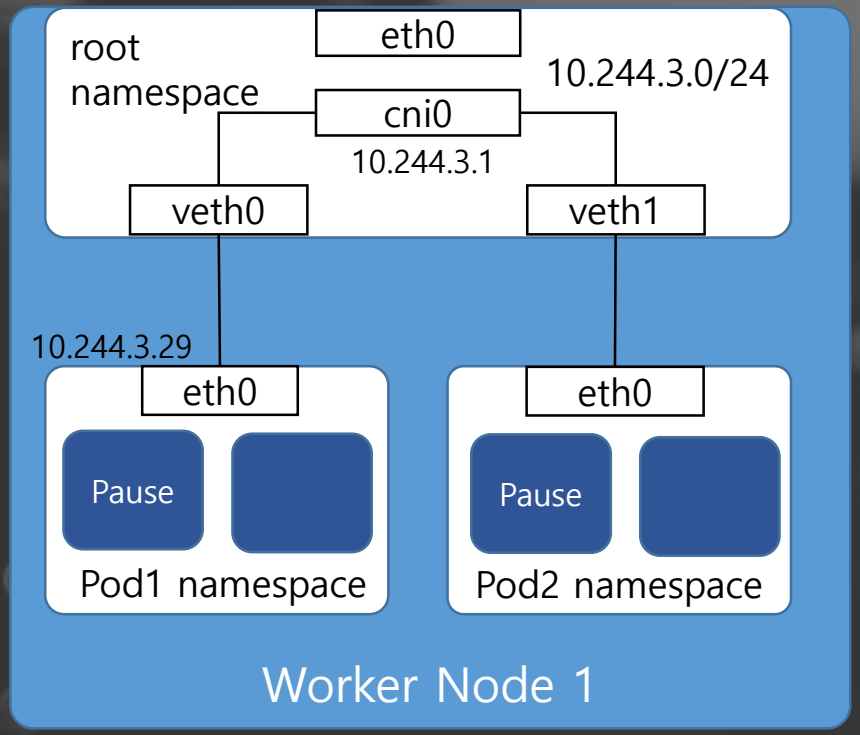
Cni0 bridge의 Interface 목록




```
[opc@inho-k8s-worker3-setup ~]$ brctl show
bridge name     bridge id               STP enabled     interfaces
cni0            8000.42a909af94bf       no              veth02db76dd
                                                        veth032b02e3
                                                        veth4ae0ee7b
                                                        veth607bc733
                                                        veth74eeab3b
                                                        veth8084be33
                                                        veth86086003
                                                        veth89db3028
                                                        veth926e89bf
                                                        veth97b8928e
                                                        vetha7d16da1
                                                        vethbe3a53b6
                                                        vethc431fcc8
                                                        vethe3501802
                                                        vetheb580c5f
docker0         8000.02428030e52b       no
[opc@inho-k8s-worker3-setup ~]$ brctl showmacs cni0
port no mac addr                is local?       ageing timer
  7     02:dc:6a:ff:f0:7e       yes                0.00
  7     02:dc:6a:ff:f0:7e       yes                0.00
  8     0a:16:e4:6f:17:13       yes                0.00
  8     0a:16:e4:6f:17:13       yes                0.00
  3     22:6a:e4:e8:5b:67       no                 0.80
 15     32:0c:f7:d2:ed:16       yes                0.00
 15     32:0c:f7:d2:ed:16       yes                0.00
 10     32:85:c7:68:a1:a3       yes                0.00
 10     32:85:c7:68:a1:a3       yes                0.00
```

1 Pod-to-Pod in a Same Node

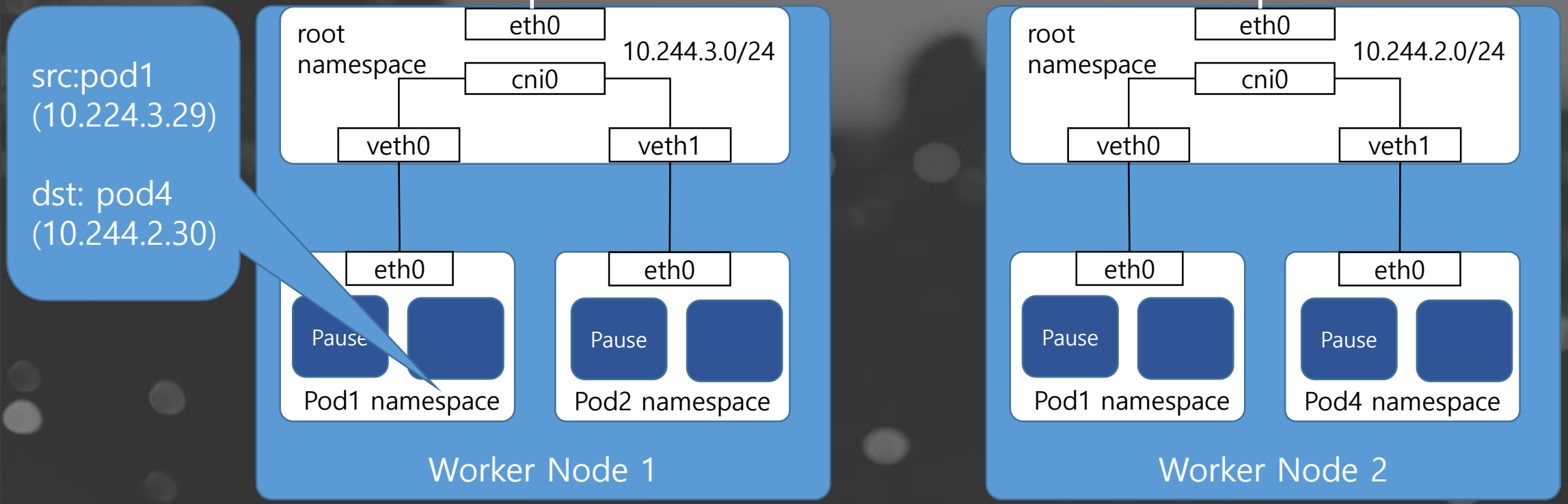
Root namespace의 routing table
- 10.244.0.0/24, 10.244.0.1/24. -> flannel.1
- 10.244.3.0/24 dev cni0




```
[opc@inho-k8s-worker3-setup ~]$ ip route
default via 10.0.1.1 dev ens3
10.0.1.0/24 dev ens3 proto kernel scope link src 10.0.1.5
10.244.0.0/24 via 10.244.0.0 dev flannel.1 onlink
10.244.1.0/24 via 10.244.1.0 dev flannel.1 onlink
10.244.2.0/24 via 10.244.2.0 dev flannel.1 onlink
10.244.3.0/24 dev cni0 proto kernel scope link src 10.244.3.1
169.254.0.0/16 dev ens3 proto static scope link
169.254.0.0/16 dev ens3 scope link metric 1002
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1 li
[opc@inho-k8s-worker3-setup ~]$
```

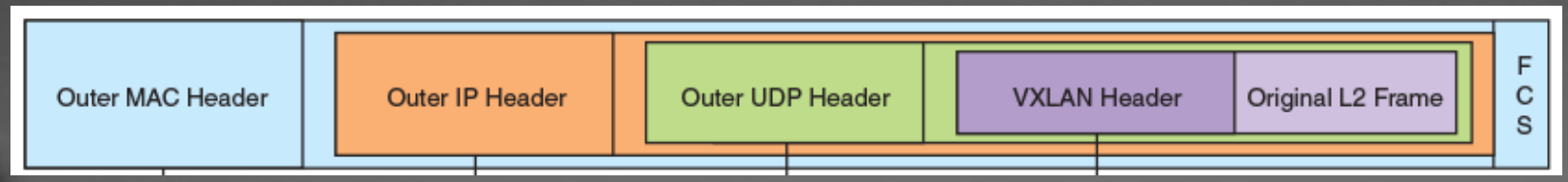
1 Pod-to-Pod across Node

Overlay Network
- Flannel (VXLAN)



1 Pod-to-Pod across Node

Overlay Network
- Flannel (VXLAN)



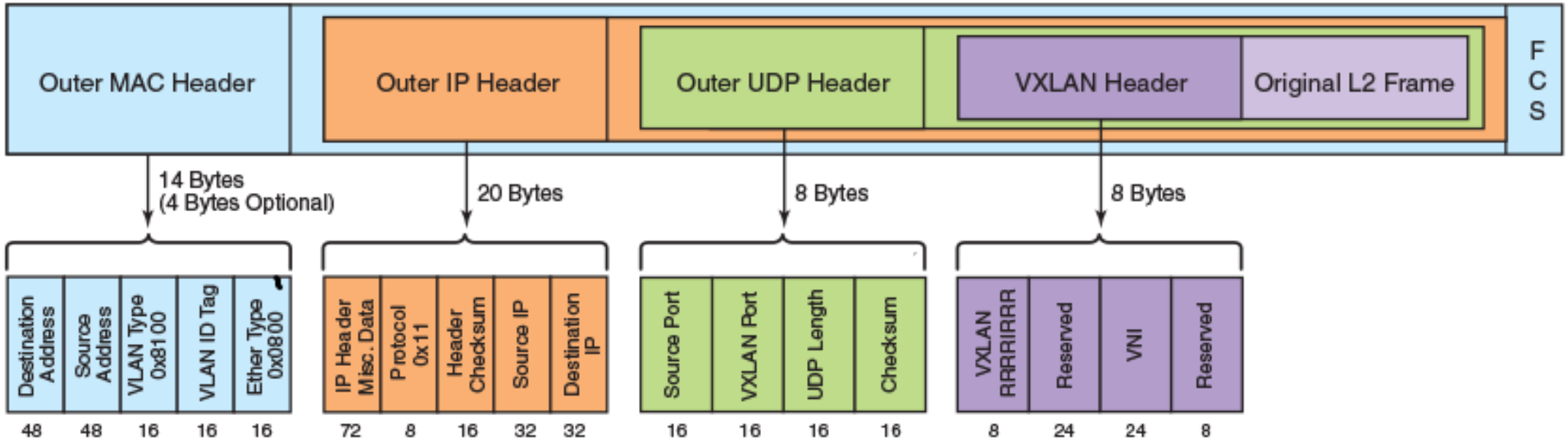
src:pod1
(10.224.3.29)

dst: pod4
(10.244.2.30)

root
namespace
eth0
10.244.3.0/24
cni0
veth0
veth1
eth0
Pause
Pod1 namespace
eth0
Pause
Pod2 namespace
Worker Node 1

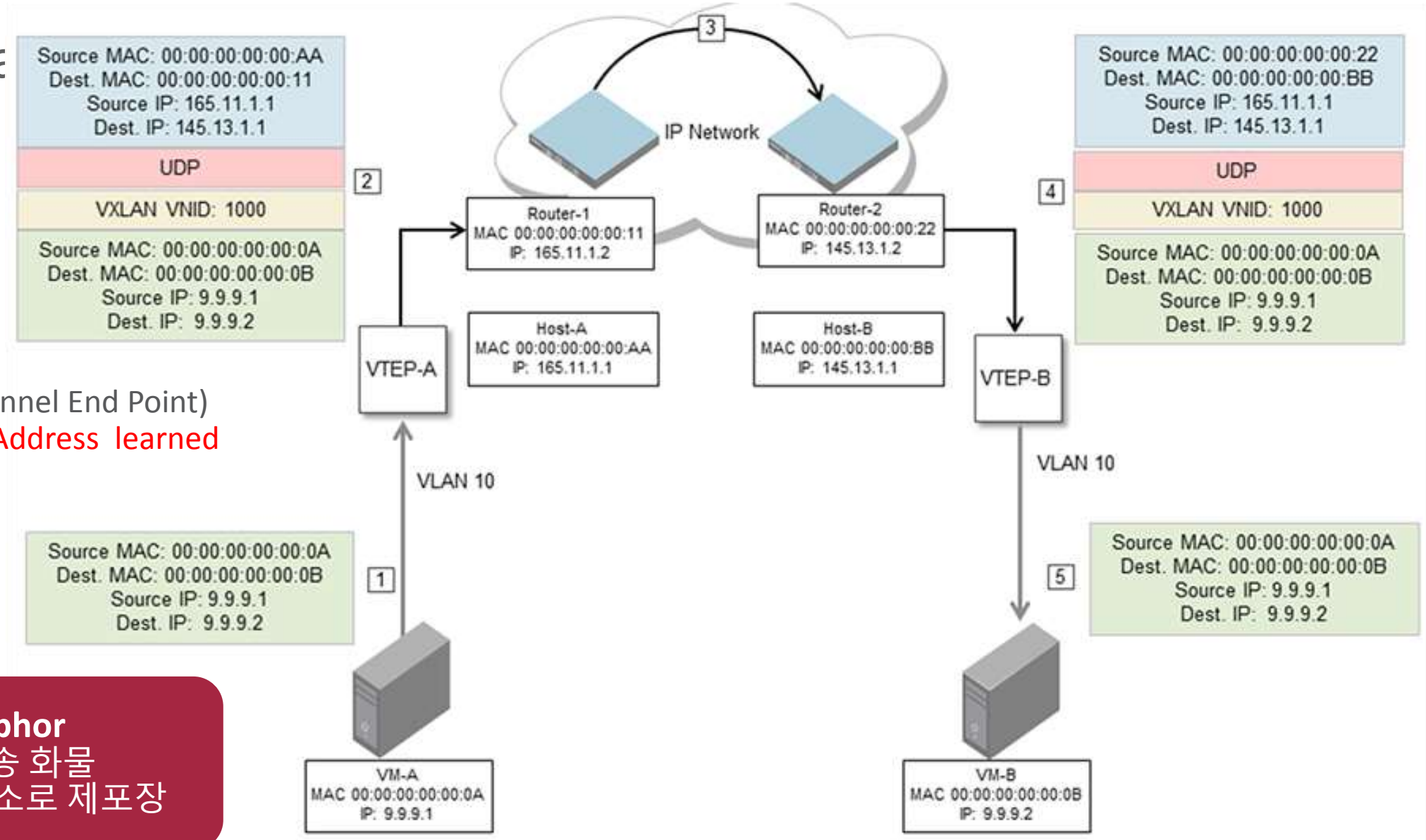
root
namespace
eth0
10.244.2.0/24
cni0
veth0
veth1
eth0
Pause
Pod1 namespace
eth0
Pause
Pod4 namespace
Worker Node 2

VxLAN Packet format



http://www.brocade.com/content/html/en/deployment-guide/brocade-vcs-gateway-vmware-dp/GUID-5A5F6C36-E03C-4CA6-9833-1907DD928842.html

VxLAN Unica

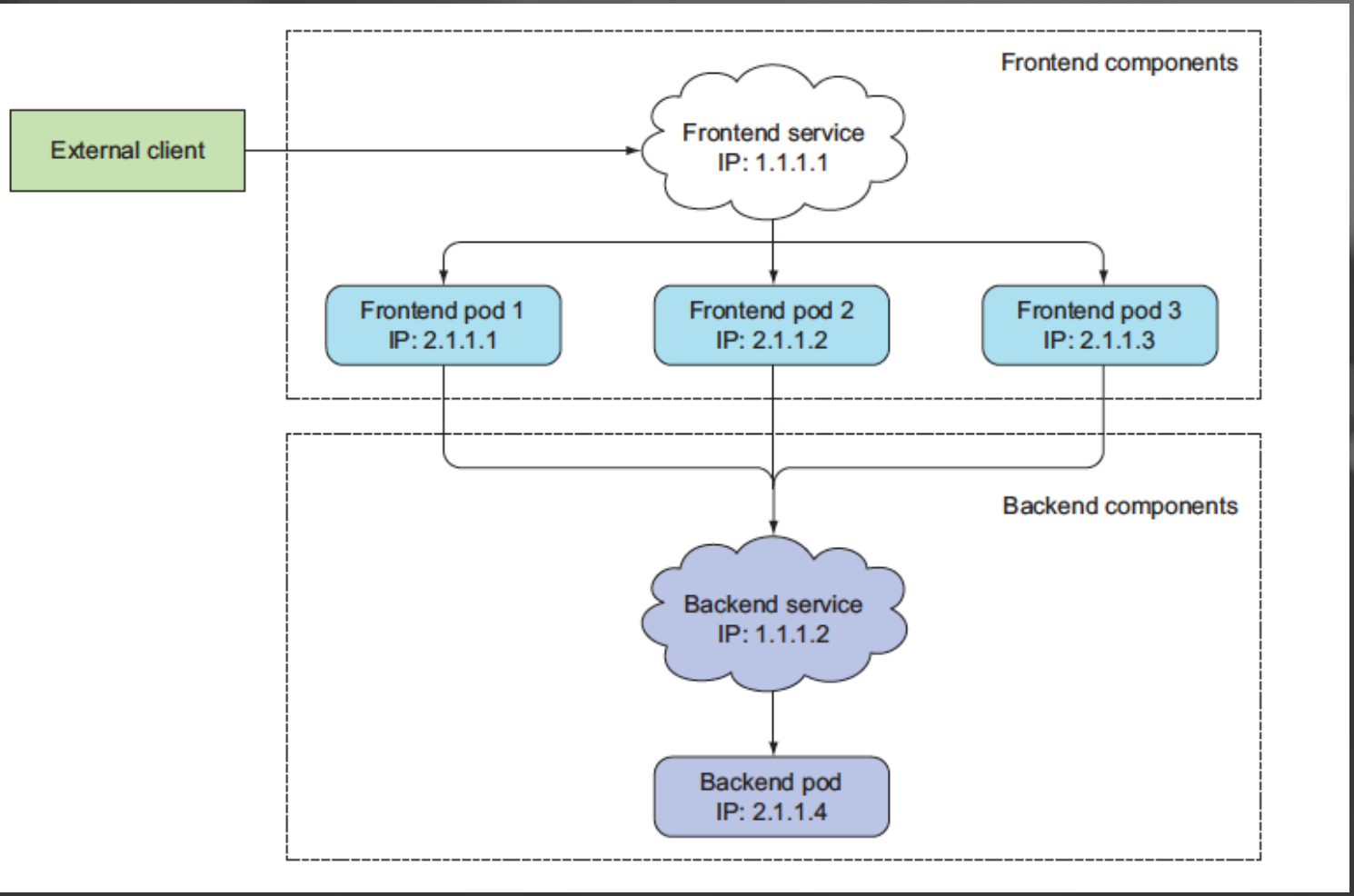


- VTEP (VxLAN Tunnel End Point)
- Unknown Mac Address learned by IP Multicast

Metaphor
국제 탁송 화물
국제 우편 주소로 제포장

1 Service

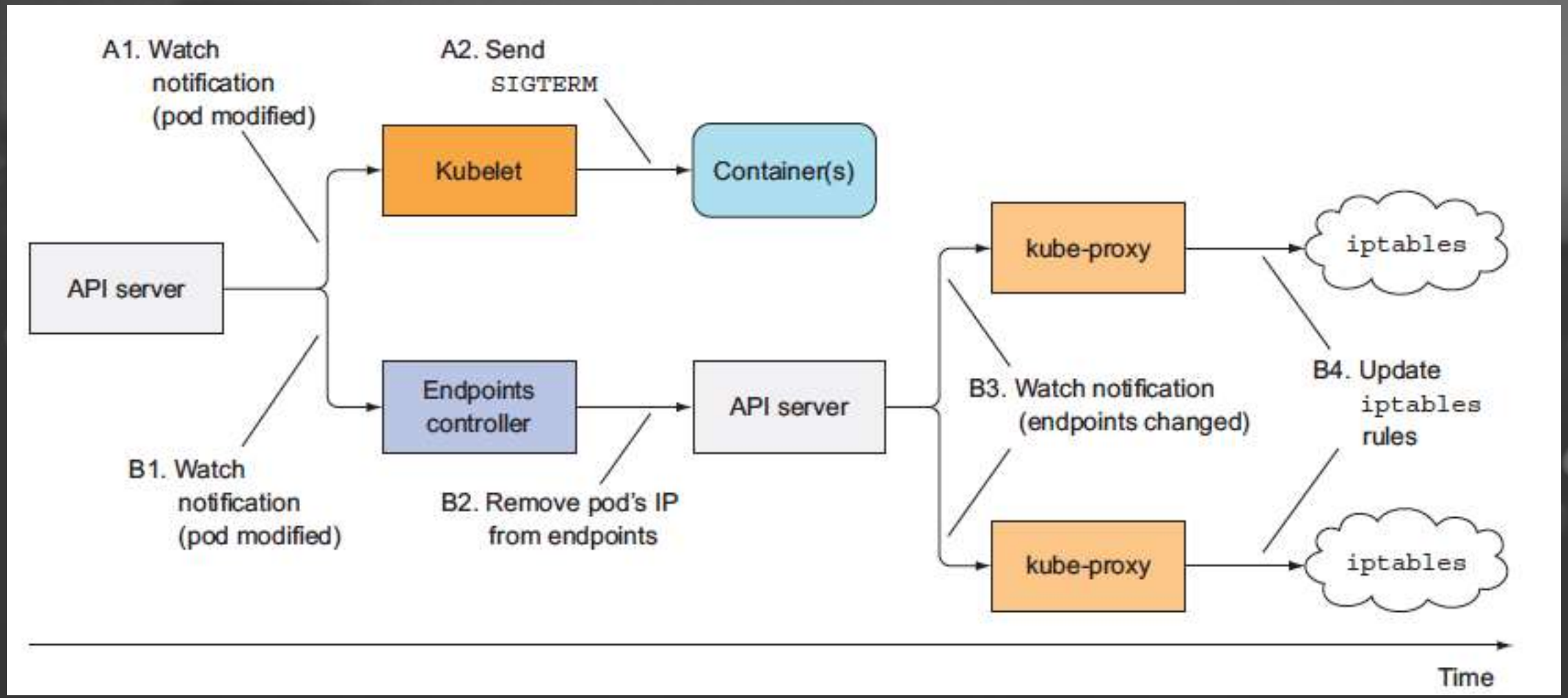
Service의 Cluster IP는 실제로 존재하는 IP가 아니다. Kube-proxy를 통해서 iptable에 기록되는 가상 IP이다.




```
kih@IHMac-2 ~ kb get svc -n sock-shop
NAME           TYPE        CLUSTER-IP       EX
carts          ClusterIP   10.111.9.53      <r
carts-db       ClusterIP   10.111.15.248    <r
catalogue      ClusterIP   10.109.198.55    <r
catalogue-db   ClusterIP   10.100.164.170   <r
front-end      NodePort    10.99.4.188      <r
orders         ClusterIP   10.111.90.77     <r
orders-db      ClusterIP   10.101.231.237   <r
payment        ClusterIP   10.105.157.220   <r
queue-master   ClusterIP   10.98.196.234    <r
rabbitmq       ClusterIP   10.105.237.245   <r
shipping       ClusterIP   10.103.187.205   <r
user           ClusterIP   10.98.138.49     <r
user-db        ClusterIP   10.97.70.63      <r
```

1 Service

Service의 Cluster IP는 실제로 존재하는 IP가 아니다. Kube-proxy를 통해서 iptable에 기록되는 가상 IP이다.



1 Service – iptables chain 예시

order라는 service를 호출하는 경우 cluster ip : 10.21.50.112를 호출하는데 이 IP는 Worker Node의 물리적인 IP가 아니다.

```
kih@IHMac-2 ~/Dev/Terraform/terraform-kubernetes-installer  master ● ?  kb get svc -n sock-shop
NAME           TYPE           CLUSTER-IP       EXTERNAL-IP      PORT(S)        AGE
carts          ClusterIP      10.21.222.195    <none>           80/TCP         141m
carts-db       ClusterIP      10.21.8.57       <none>           27017/TCP      141m
catalogue      ClusterIP      10.21.216.161    <none>           80/TCP         141m
catalogue-db   ClusterIP      10.21.59.153     <none>           3306/TCP       141m
front-end      LoadBalancer   10.21.224.37     129.146.93.195   80:30001/TCP   141m
orders         ClusterIP      10.21.50.112     <none>           80/TCP         141m
orders-db      ClusterIP      10.21.64.179     <none>           27017/TCP      141m
payment        ClusterIP      10.21.48.44      <none>           80/TCP         141m
queue-master   ClusterIP      10.21.20.233     <none>           80/TCP         141m
rabbitmq       ClusterIP      10.21.245.206    <none>           5672/TCP       141m
shipping       ClusterIP      10.21.28.137     <none>           80/TCP         141m
user           ClusterIP      10.21.73.241     <none>           80/TCP         141m
user-db        ClusterIP      10.21.241.128    <none>           27017/TCP      141m
```

1 Service – iptables chain 예시

Service 를 생성하면 Kube-proxy가 iptable에 아래와 같은 rule을 추가한다.
destination(-d)이 10.21.50.112이고 tcp(-p) 이며 port=80(-dport)인 경우에 해당 하면 KUBE-SVC-BWR2E3DUEHRRCI5E로 jump(-j)하도록 한다.

```
[opc@k8s-worker-ad3-0 pause-rootfs]$ sudo iptables -t nat -S | grep 10.21.50.112
-A KUBE-SERVICES -d 10.21.50.112/32 -p tcp -m comment --comment "sock-shop/orders: cluster IP" -m tcp --dport 80 -j KUBE-MARK-MASQ
-A KUBE-SERVICES -d 10.21.50.112/32 -p tcp -m comment --comment "sock-shop/orders: cluster IP" -m tcp --dport 80 -j KUBE-SVC-BWR2E3DUEHRRCI5E
```

Chain을 따라가 보면 KUBE-SVC-BWR2E3DUEHRRCI5E -> KUBE-SEP-5UUIRYMHBRG2PBN3 -> 10.99.2.5(pod IP)

```
[opc@k8s-worker-ad3-0 pause-rootfs]$ sudo iptables -t nat -S | grep 10.21.50.112
-A KUBE-SERVICES -d 10.21.50.112/32 -p tcp -m comment --comment "sock-shop/orders: cluster IP" -m tcp --dport 80 -j KUBE-MARK-MASQ
-A KUBE-SERVICES -d 10.21.50.112/32 -p tcp -m comment --comment "sock-shop/orders: cluster IP" -m tcp --dport 80 -j KUBE-SVC-BWR2E3DUEHRRCI5E
[opc@k8s-worker-ad3-0 pause-rootfs]$ sudo iptables -t nat -S | grep KUBE-SVC-BWR2E3DUEHRRCI5E
-N KUBE-SVC-BWR2E3DUEHRRCI5E
-A KUBE-SERVICES -d 10.21.50.112/32 -p tcp -m comment --comment "sock-shop/orders: cluster IP" -m tcp --dport 80 -j KUBE-SVC-BWR2E3DUEHRRCI5E
-A KUBE-SVC-BWR2E3DUEHRRCI5E -m comment --comment "sock-shop/orders:" -j KUBE-SEP-5UUIRYMHBRG2PBN3
[opc@k8s-worker-ad3-0 pause-rootfs]$ ^C
[opc@k8s-worker-ad3-0 pause-rootfs]$ sudo iptables -t nat -S | grep KUBE-SEP-5UUIRYMHBRG2PBN3
-N KUBE-SEP-5UUIRYMHBRG2PBN3
-A KUBE-SEP-5UUIRYMHBRG2PBN3 -s 10.99.2.5/32 -m comment --comment "sock-shop/orders:" -j KUBE-MARK-MASQ
-A KUBE-SEP-5UUIRYMHBRG2PBN3 -p tcp -m comment --comment "sock-shop/orders:" -m tcp -j DNAT --to-destination 10.99.2.5:80
-A KUBE-SVC-BWR2E3DUEHRRCI5E -m comment --comment "sock-shop/orders:" -j KUBE-SEP-5UUIRYMHBRG2PBN3
```

1 Service – LoadBalancing 예시

Service 가 3개의 Endpoint가 있을때 3개를 Loadbalancing 하는 메커니즘은 아래 Work Node에 있는 iptable에 비밀이 숨어 있다.
 - 우선 Order라는 Service에 3개의 Pod가 있어서 Endpoint가 3개 있다.

```
kih@IHMac-2 ~ kb get po -n sock-shop
NAME                            READY   STATUS    RESTARTS   AGE
carts-55f7f5c679-vpqkk          1/1     Running   0          78m
carts-db-5c55874946-ft65s       1/1     Running   0          78m
catalogue-5764fdf6d-mkjx9       1/1     Running   0          78m
catalogue-db-66ff5bbbf5-v7rz2   1/1     Running   0          78m
front-end-f99dbcb9c-7bmx2       1/1     Running   0          78m
orders-7b69bf5686-9d4f6         1/1     Running   0          72m
orders-7b69bf5686-xf8nk         1/1     Running   0          78m
orders-7b69bf5686-zddmc         1/1     Running   0          72m
orders-db-7bc46bdb98-8m68l      1/1     Running   0          78m
payment-79769c5874-s6whr        1/1     Running   0          78m
queue-master-56894bf9b8-r7mh6   1/1     Runnin
rabbitmq-bf496c754-wqjmz        1/1     Runnin
shipping-7d6f7bf67b-drwbd       1/1     Runnin
user-7d4d7c9675-s5ddk           1/1     Runnin
user-db-5d6b6cd769-kvz6g        1/1     Runnin
```

```
kih@IHMac-2 ~ kb describe svc orders -n sock-shop                    4162  17:33:02
Name:              orders
Namespace:         sock-shop
Labels:            name=orders
Annotations:       kubectl.kubernetes.io/last-applied-configuration:
                     {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"labels"
:{"name":"orders"},"name":"orders","namespace":"sock-shop"},"spec...
Selector:          name=orders
Type:              ClusterIP
IP:                10.111.90.77
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         10.244.1.23:80,10.244.2.12:80,10.244.3.29:80
Session Affinity:  None
Events:            <none>
```

1 Service – 예시 LB

이 3개의 EndPoint는 kube proxy가 Worker Node의 iptable에 기록한다.
아래 보면 Service 의 Cluster IP (10.111.90.77) 로 iptable의 nat table을 검색하면 KUBE-SVC-BWR2E3DUEHRRCI5E 로 chaining 되는 것을 볼 수 있고, 이 KUBE-SVC-BWR2E3DUEHRRCI5E 는 KUBE-SEP-CHSFBYOEYZ7WWARH, KUBE-SEP-CC74YOAJP33X6HJV, KUBE-SEP-RUZQ7LYDHDDBPMDG 로 chaining 되는 데 이때 random module을 이용해서
LoadBalancing을 수행하고 이 Entity가 다시 DNAT해서 Pod 를 가리키는 IP/Port로 변환 된다.

```
-A PRE_public -j PRE_public_allow
[root@inho-k8s-worker3-setup opc]# iptables -t nat -S | grep 10.111.90.77
-A KUBE-SERVICES ! -s 10.244.0.0/16 -d 10.111.90.77/32 -p tcp -m comment --comment "sock-shop/orders: cluster IP" -m tcp --dport 80 -j KUBE-MARK-MASQ
-A KUBE-SERVICES -d 10.111.90.77/32 -p tcp -m comment --comment "sock-shop/orders: cluster IP" -m tcp --dport 80 -j KUBE-SVC-BWR2E3DUEHRRCI5E
[root@inho-k8s-worker3-setup opc]# iptables -t nat -S | grep KUBE-SVC-BWR2E3DUEHRRCI5E
-N KUBE-SVC-BWR2E3DUEHRRCI5E
-A KUBE-SERVICES -d 10.111.90.77/32 -p tcp -m comment --comment "sock-shop/orders: cluster IP" -m tcp --dport 80 -j KUBE-SVC-BWR2E3DUEHRRCI5E
-A KUBE-SVC-BWR2E3DUEHRRCI5E -m statistic --mode random --probability 0.33332999982 -j KUBE-SEP-CHSFBYOEYZ7WWARH
-A KUBE-SVC-BWR2E3DUEHRRCI5E -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-CC74YOAJP33X6HJV
-A KUBE-SVC-BWR2E3DUEHRRCI5E -j KUBE-SEP-RUZQ7LYDHDDBPMDG
[root@inho-k8s-worker3-setup opc]# iptables -t nat -S | grep KUBE-SEP-CHSFBYOEYZ7WWARH
-N KUBE-SEP-CHSFBYOEYZ7WWARH
-A KUBE-SEP-CHSFBYOEYZ7WWARH -s 10.244.1.23/32 -j KUBE-MARK-MASQ
-A KUBE-SEP-CHSFBYOEYZ7WWARH -p tcp -m tcp -j DNAT --to-destination 10.244.1.23:80
-A KUBE-SVC-BWR2E3DUEHRRCI5E -m statistic --mode random --probability 0.33332999982 -j KUBE-SEP-CHSFBYOEYZ7WWARH
[root@inho-k8s-worker3-setup opc]# iptables -t nat -S | grep KUBE-SEP-CC74YOAJP33X6HJV
-N KUBE-SEP-CC74YOAJP33X6HJV
-A KUBE-SEP-CC74YOAJP33X6HJV -s 10.244.2.12/32 -j KUBE-MARK-MASQ
-A KUBE-SEP-CC74YOAJP33X6HJV -p tcp -m tcp -j DNAT --to-destination 10.244.2.12:80
-A KUBE-SVC-BWR2E3DUEHRRCI5E -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-CC74YOAJP33X6HJV
[root@inho-k8s-worker3-setup opc]# iptables -t nat -S | grep KUBE-SEP-RUZQ7LYDHDDBPMDG
-N KUBE-SEP-RUZQ7LYDHDDBPMDG
-A KUBE-SEP-RUZQ7LYDHDDBPMDG -s 10.244.3.29/32 -j KUBE-MARK-MASQ
-A KUBE-SEP-RUZQ7LYDHDDBPMDG -p tcp -m tcp -j DNAT --to-destination 10.244.3.29:80
-A KUBE-SVC-BWR2E3DUEHRRCI5E -j KUBE-SEP-RUZQ7LYDHDDBPMDG
[root@inho-k8s-worker3-setup opc]# packet_write_wait: Connection to 129.146.46.83 port 22: Broken pipe
```

1 Iptable-sync-period

bash

```
$ while sleep 1;
  do date +%X; curl -sS http://<your load balancer ip>/ | grep ^Hello;
done
11:29:55 Hello world! via k8s-hello-world-55f48f8c94-xzvlc
11:29:56 Hello world! via k8s-hello-world-55f48f8c94-tjg4n
# this is when `iptables -F` was issued
11:30:25 Hello world! via k8s-hello-world-55f48f8c94-npkn6
11:30:27 Hello world! via k8s-hello-world-55f48f8c94-vrkr9

$ _
```

There was a gap of 29 seconds, from 11:29:56 to 11:30:25, but the cluster is back to normal.

Why does it take about 30 seconds to reply?

Is the node receiving traffic despite no routing table?

Maybe you could investigate what happens to the node in this 30 seconds.

Digging in the official documentation for `kube-pro`
flags:

- `--iptables-sync-period` - The maximum int
  are refreshed (e.g. '5s', '1m', '2h22m'). Must b
- `--iptables-min-sync-period` - The minimum
  iptables rules can be refreshed as endpoints a
  '1m', '2h22m'). (default 10s)

https://learnk8s.io/blog/kubernetes-chaos-engineering-lessons-learned/

1 Iptable vs IPVS

https://kubernetes.io/blog/2018/07/09/ipvs-based-in-cluster-load-balancing-deep-dive/

앱 Oracle News etc TechStuff Cambridge Englis... Oracle Cloud Infra... A Global School F... Online Port Scann... Linux Academy Overview - Kuber...

Documentation Blog Partners Cor

IPVS-Based In-Cluster Load Balancing Deep Dive

Author: Jun Du(Huawei), Haibin Xie(Huawei), Wei Liang(Huawei)

Editor's note: this post is part of a series of in-depth articles on what's new in Kubernetes 1.11

Introduction

Per the Kubernetes 1.11 release blog post , we announced that IPVS-Based In-Cluster Service Load Balancing graduates to General Availability. In this blog, we will take you through a deep dive of the feature.

What Is IPVS?

IPVS (IP Virtual Server) is built on top of the Netfilter and implements transport-layer load balancing as part of the Linux kernel.

IPVS is incorporated into the LVS (Linux Virtual Server), where it runs on a host and acts as a load balancer in front of a cluster of real servers. IPVS can direct requests for TCP- and UDP-based services to the real servers, and make services of the real servers appear as virtual services on a single IP address. Therefore, IPVS naturally supports Kubernetes Service.

Why IPVS for Kubernetes?

As Kubernetes grows in usage, the scalability of its resources becomes more and more important. In particular, the scalability of services is paramount to the adoption of Kubernetes by developers/companies running large workloads.

Kube-proxy, the building block of service routing has relied on the battle-hardened iptables to implement the core supported Service types such as ClusterIP and NodePort. However, iptables struggles to scale to tens of thousands of Services because it is designed purely for firewalling purposes and is based on in-kernel rule lists.

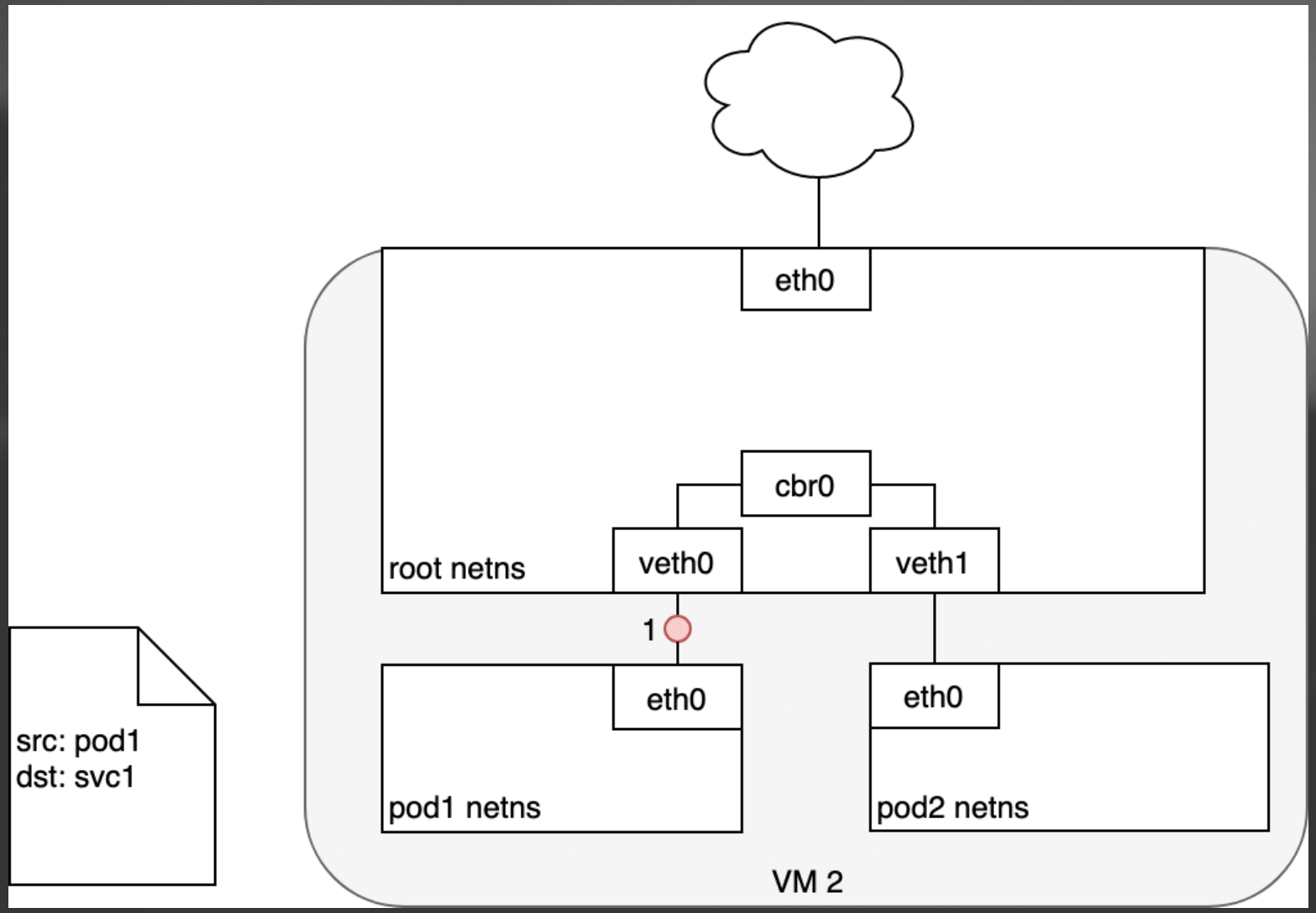
Even though Kubernetes already support 5000 nodes in release v1.6, the kube-proxy with iptables is actually a bottleneck to scale the cluster to 5000 nodes. One example is that with NodePort Service in a 5000-node cluster, if we have 2000 services and each services have 10 pods, this will cause at least 20000 iptable records on each worker node, and this can make the kernel pretty busy.

On the other hand, using IPVS-based in-cluster service load balancing can help a lot for such cases. IPVS is specifically designed for load balancing and uses more efficient data structures (hash tables) allowing for almost unlimited scale under the hood.

IPVS-based Kube-proxy

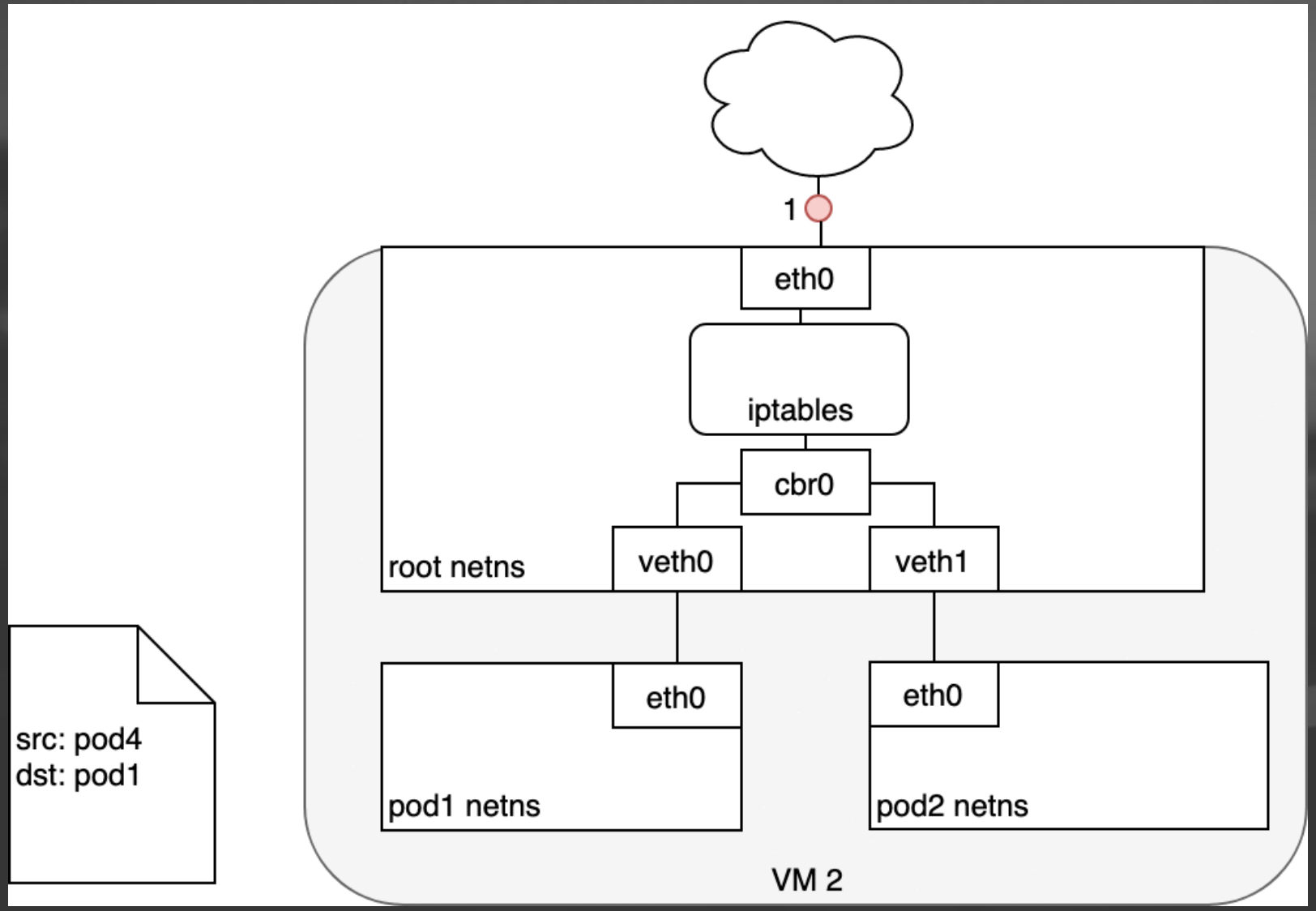
Parameter Changes

1 Pod to Service



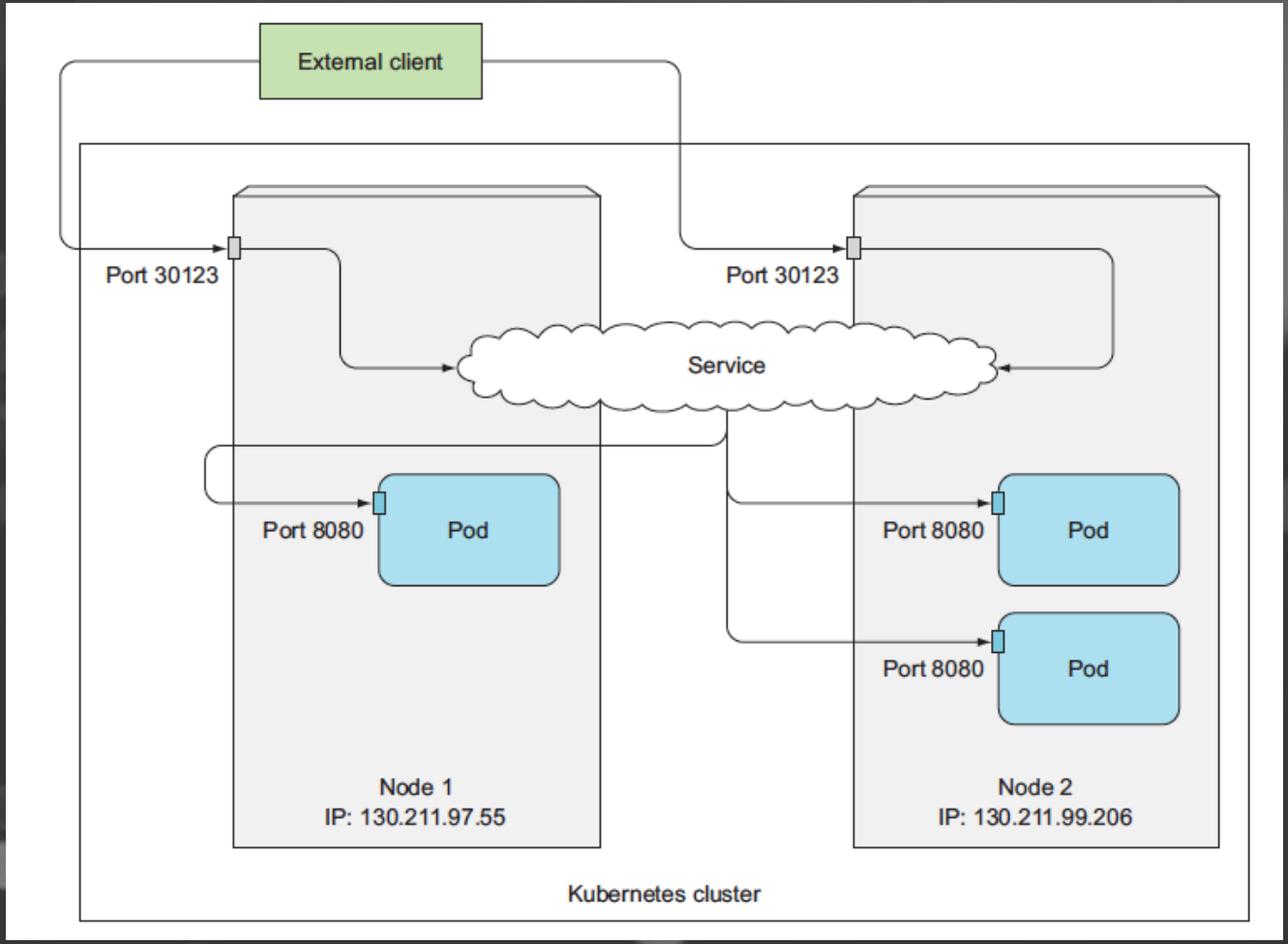
https://sookocheff.com/post/kubernetes/understanding-kubernetes-networking-model/

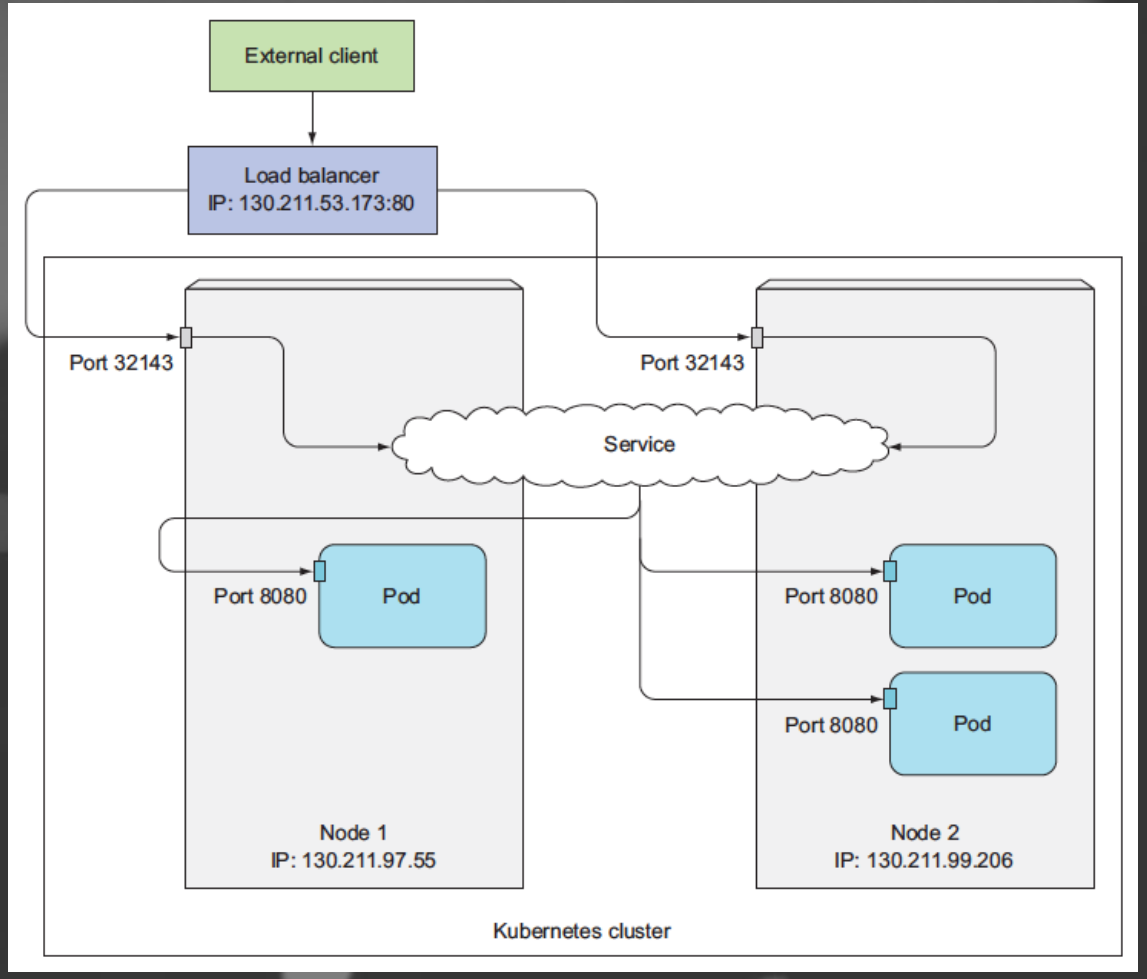
1 Service to Pod



https://sookocheff.com/post/kubernetes/understanding-kubernetes-networking-model/

1 Service – NodePort, LoadBalancer





Kubernetes Network - Demo

1 Service 추적

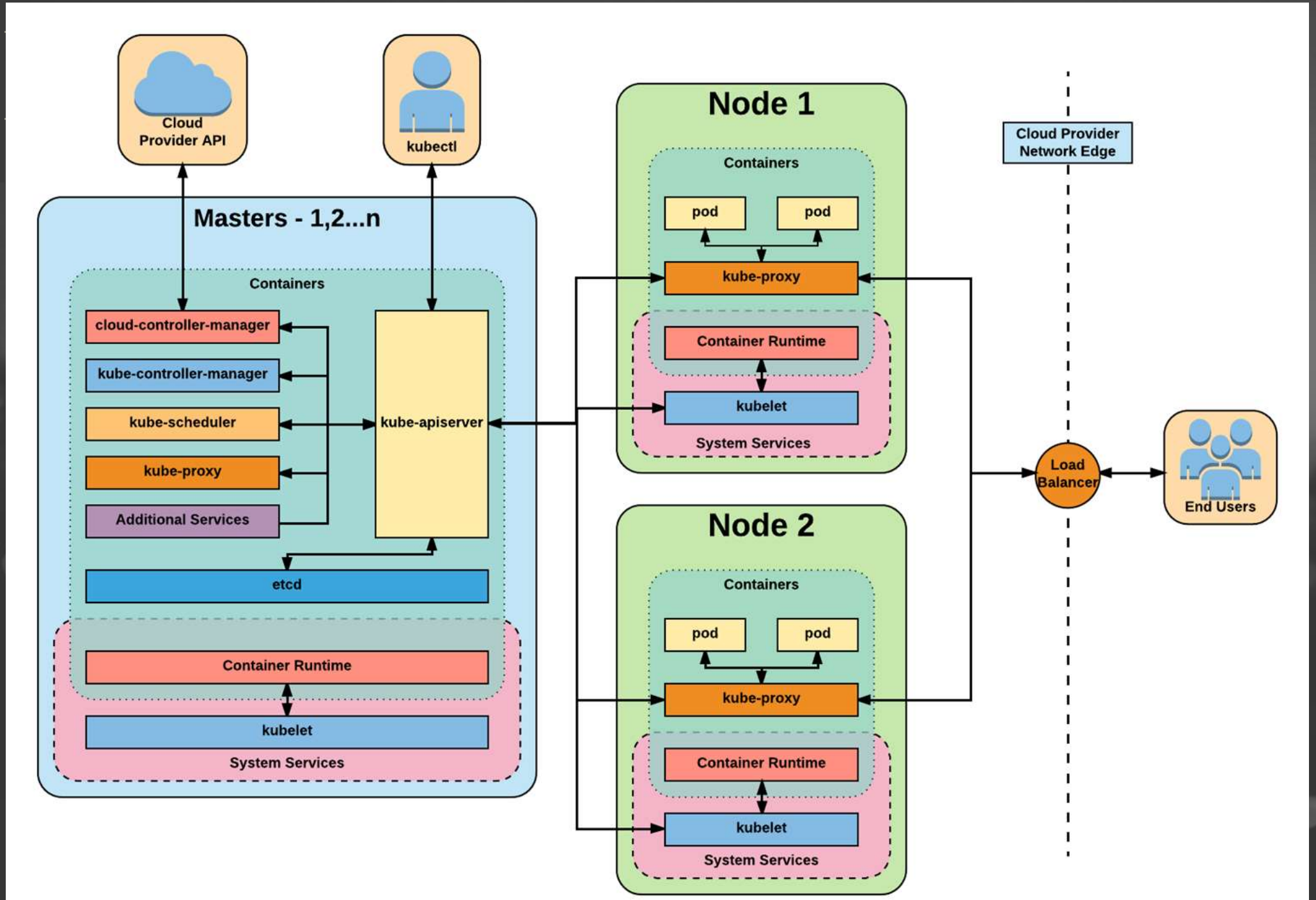
-$kubectl scale --replicats=3 deployment/orders
-$kubectl get svc –n sock-shop
 orders : ClusterIP

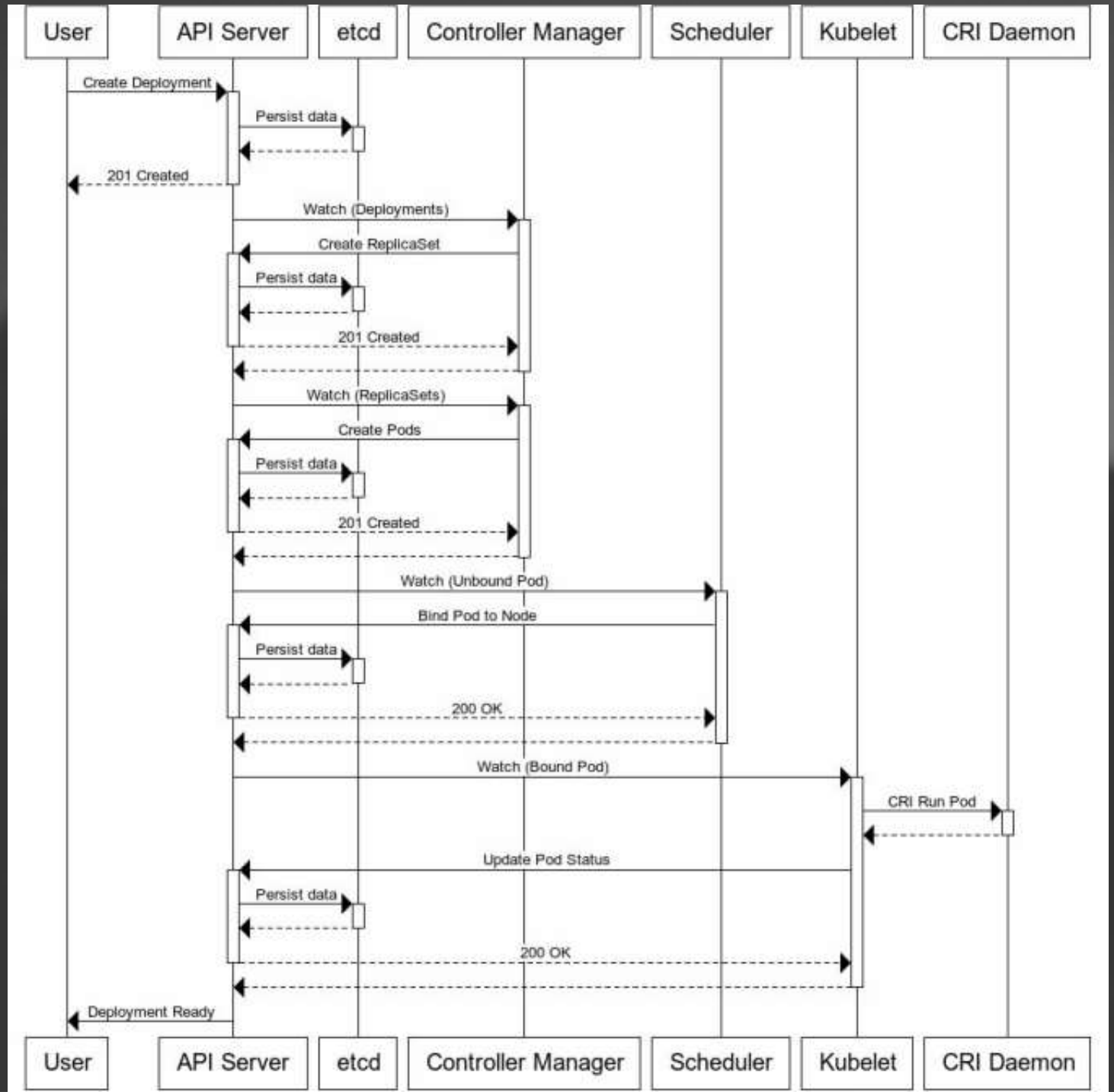
- [root@worker3] iptables –t nat –S | grep <clusterip>

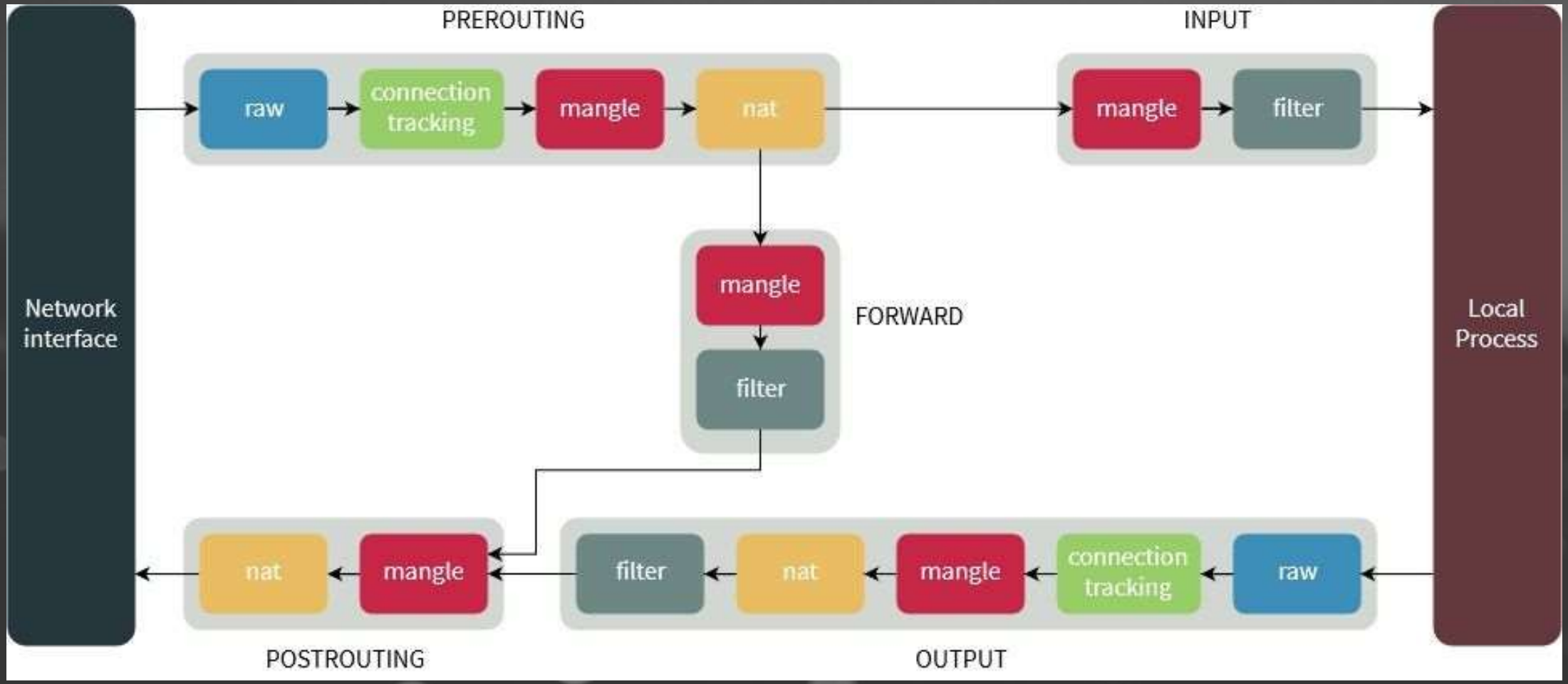
```
kih@IHMac-2 ~ kb get svc -n sock-shop
NAME           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
carts          ClusterIP   10.111.9.53      <none>        80/TCP           42h
carts-db       ClusterIP   10.111.15.248    <none>        27017/TCP        42h
catalogue      ClusterIP   10.109.198.55    <none>        80/TCP           42h
catalogue-db   ClusterIP   10.100.164.170   <none>        3306/TCP         42h
front-end      NodePort    10.99.4.188      <none>        80:30001/TCP     42h
orders         ClusterIP   10.111.90.77     <none>        80/TCP           42h
orders-db      ClusterIP   10.101.231.237   <none>        27017/TCP        42h
payment        ClusterIP   10.105.157.220   <none>        80/TCP           42h
queue-master   ClusterIP   10.98.196.234    <none>        80/TCP           42h
rabbitmq       ClusterIP   10.105.237.245   <none>        5672/TCP         42h
shipping       ClusterIP   10.103.187.205   <none>        80/TCP           42h
user           ClusterIP   10.98.138.49     <none>        80/TCP           42h
user-db        ClusterIP   10.97.70.63      <none>        27017/TCP        42h
kih@IHMac-2 ~ kb describe svc orders -n sock-shop
Name:              orders
Namespace:         sock-shop
Labels:            name=orders
Annotations:       kubectl.kubernetes.io/last-applied-configuration:
                     {"apiVersion":"v1","kind":"Service","metadata":{"annotations
rs","namespace":"sock-shop"},"spec...
Selector:          name=orders
Type:              ClusterIP
IP:                10.111.90.77
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         10.244.1.23:80,10.244.2.12:80,10.244.3.29:80
Session Affinity:  None
Events:            <none>
kih@IHMac-2 ~
```

```
[root@inho-k8s-worker2-setup opc]# iptables -t nat -S | grep  10.111.90.77
-A KUBE-SERVICES ! -s 10.244.0.0/16 -d 10.111.90.77/32 -p tcp -m comment --comment "sock-shop/orders: cluster IP" -m tcp --dport 80 -j KUBE-MARK-MASQ
-A KUBE-SERVICES -d 10.111.90.77/32 -p tcp -m comment --comment "sock-shop/orders: cluster IP" -m tcp --dport 80 -j KUBE-SVC-BWR2E3DUEHRRCI5E
[root@inho-k8s-worker2-setup opc]# iptables -t nat -S | grep KUBE-SVC-BWR2E3DUEHRRCI5E
-N KUBE-SVC-BWR2E3DUEHRRCI5E
-A KUBE-SERVICES -d 10.111.90.77/32 -p tcp -m comment --comment "sock-shop/orders: cluster IP" -m tcp --dport 80 -j KUBE-SVC-BWR2E3DUEHRRCI5E
-A KUBE-SVC-BWR2E3DUEHRRCI5E -m statistic --mode random --probability 0.33332999982 -j KUBE-SEP-CHSFBYOEYZ7WWARH
-A KUBE-SVC-BWR2E3DUEHRRCI5E -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-CC74YOAJP33X6HJV
-A KUBE-SVC-BWR2E3DUEHRRCI5E -j KUBE-SEP-RUZQ7LYDHDDBPMDG
```

Summary


Cloud
Provider API
kubectl
Masters - 1,2...n
Containers
cloud-controller-manager
kube-controller-manager
kube-scheduler
kube-proxy
Additional Services
kube-apiserver
etcd
Container Runtime
kubelet
System Services
Node 1
Containers
pod
pod
kube-proxy
Container Runtime
kubelet
System Services
Node 2
Containers
pod
pod
kube-proxy
Container Runtime
kubelet
System Services
Cloud Provider
Network Edge
Load
Balancer
End Users


User
API Server
etcd
Controller Manager
Scheduler
Kubelet
CRI Daemon
Create Deployment
Persist data
201 Created
Watch (Deployments)
Create ReplicaSet
Persist data
201 Created
Watch (ReplicaSets)
Create Pods
Persist data
201 Created
Watch (Unbound Pod)
Bind Pod to Node
Persist data
200 OK
Watch (Bound Pod)
CRI Run Pod
Update Pod Status
Persist data
200 OK
Deployment Ready
User
API Server
etcd
Controller Manager
Scheduler
Kubelet
CRI Daemon



https://msazure.club/kubernetes-services-and-iptables/

10 References

- Introduction to k8s workshop : https://docs.google.com/presentation/d/1zrfVIE5r61ZNQrmXKx5gJmBcXnoa_WerHEnTxu5SMco/edit?usp=sharing
- K8s comprehensive Overview: https://docs.google.com/presentation/d/1_xwLGM6U6EDK59s9Zny-zWGGAbQk47cZPuBbIU3Upus/edit#slide=id.g2c3848b8cd_0_158
- Kubernetes in Action
- A Crash Course on Container Orchestration & k8s : https://speakerd.s3.amazonaws.com/presentations/cb7394a1868b479eb6723d120995f259/Container_Orchestration_-_Interop_ITX__17.pdf
- Pod :https://ssup2.github.io/theory_analysis/Kubernetes_Pod/?fbclid=IwAR1IZtyjusnW1iX8C1s1c5QpQMigmiH-Y7pFDZ3dwtL44TJ8esDZqp-lXHg
- Pod : https://www.ianlewis.org/en/what-are-kubernetes-pods-anyway
- Pod : https://blog.2dal.com/2018/03/28/kubernetes-01-pod/

Integrated Cloud

Applications & Platform Services