

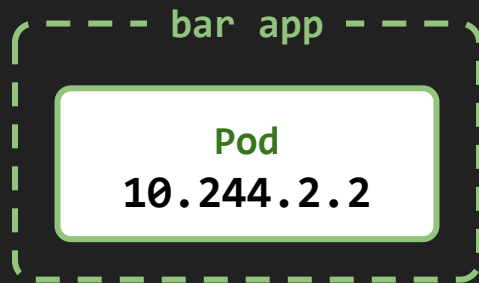
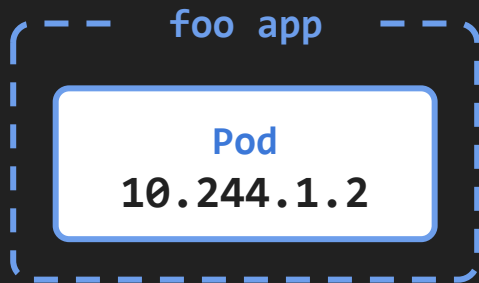
Kubernetes 네트워크 이해하기 (2)

: 서비스 개념과 동작 원리

전호준 (<https://hyojun.me>)

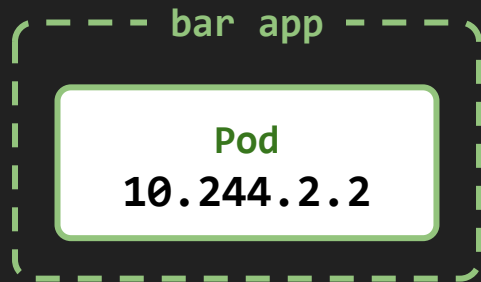
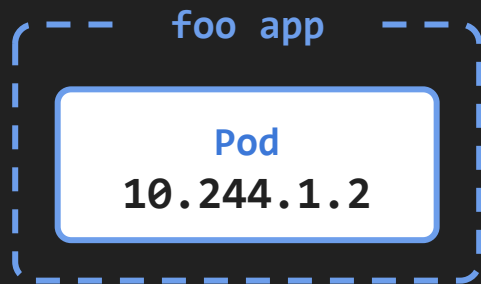
Kubernetes에 배포된 두 개의 Application

Kubernetes Cluster



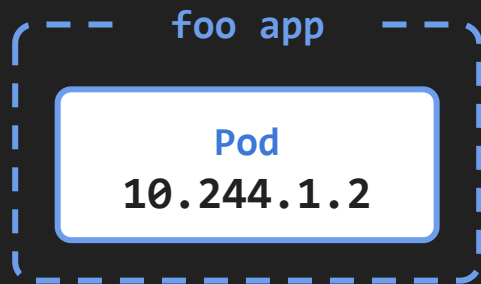
Pod들은 클러스터 내에서 고유한 IP로 서로 통신

Kubernetes Cluster



bar app의 Pod이 변경 또는 종료되어 새로운 Pod이 생성되면?

Kubernetes Cluster

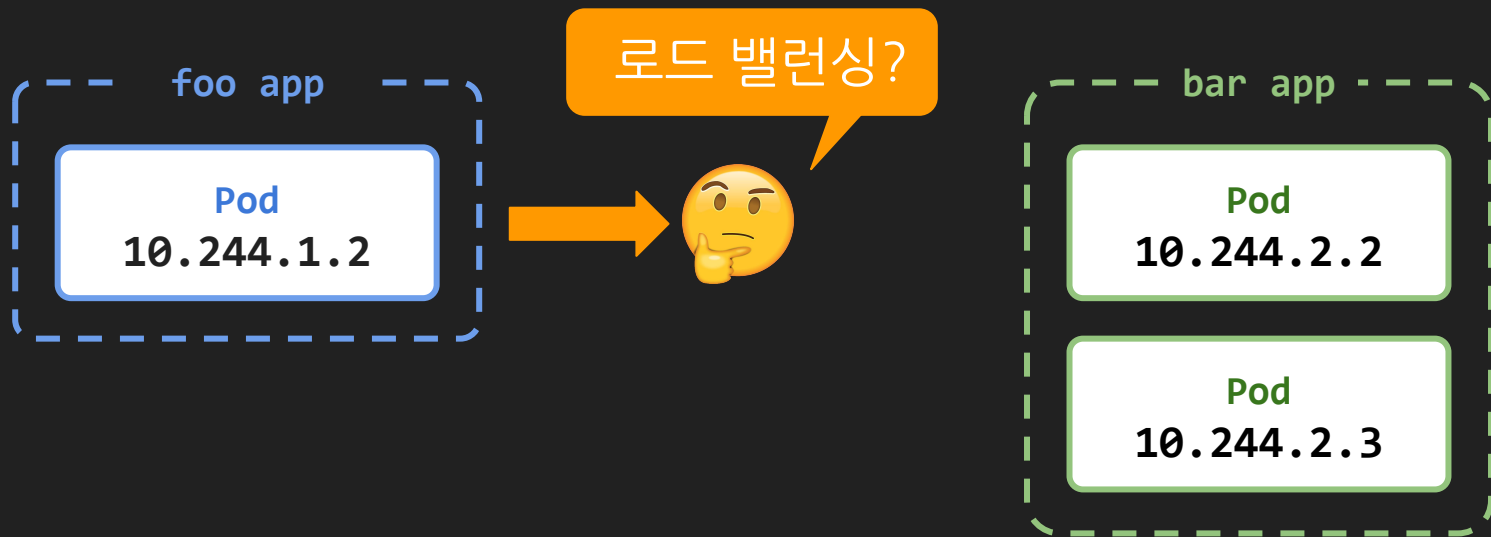


변경된 Pod IP를 어떻게 알려주지?



bar app이 여러 개의 Pod으로 구성된 경우

Kubernetes Cluster



서비스(Service)

- Pod 집합에 대한 접근을 추상화
 - Selector를 통한 대상 Pod 집합 정의
 - 단일 고정 Endpoint를 제공 (IP + Port)
- 클러스터 외부에 존재하는 백엔드에 대한 추상화도 제공
 - Services without selector
- 로드 밸런싱 지원

replicaset.yaml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: hello
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hello
  template:
    metadata:
      labels:
        app: hello
    spec:
      containers:
        - name: hello-world
          image: gcr.io/google-samples/node-hello:1.0
          ports:
            - containerPort: 8080
              protocol: TCP
```

service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: hello-service
spec:
  selector:
    app: hello
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

replicaset.yaml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: hello
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hello
  template:
    metadata:
      labels:
        app: hello
    spec:
      containers:
        - name: hello-world
          image: gcr.io/google-samples/node-hello:1.0
          ports:
            - containerPort: 8080
              protocol: TCP
```

service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: hello-service
spec:
  selector:
    app: hello
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

Selector를 이용한 대상 Pod 집합 설정

A yellow line with an arrow at the end originates from the 'selector: app: hello' field in the service.yaml file and points to the 'labels: app: hello' field in the replicaset.yaml file, illustrating how the service uses the replica set's labels to select its target pods.

replicaset.yaml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: hello
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hello
  template:
    metadata:
      labels:
        app: hello
    spec:
      containers:
        - name: hello-world
          image: gcr.io/google-samples/node-hello:1.0
          ports:
            - containerPort: 8080
              protocol: TCP
```

service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: hello-service
spec:
  selector:
    app: hello
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 8080
```

<Service IP>:80 → <Pod IP>:8080



Service는 Pod 집합의 변경과 상관없이 항상 고정된 IP를 가진다.

```
vagrant@hyojun:~$ kubectl describe svc hello-service
```

```
Name: hello-service
```

```
Namespace: default
```

```
Labels: <none>
```

```
Annotations: <none>
```

```
Selector: app=hello
```

```
Type: ClusterIP
```

```
IP: 10.225.30.68
```

```
Port: <unset> 80/TCP
```

```
TargetPort: 8080/TCP
```

```
Endpoints: 10.244.0.131:8080,10.244.1.131:8080,10.244.1.3:8080
```

```
Session Affinity: None
```

```
Events: <none>
```

```
vagrant@hyojun:~$ kubectl get pods -l app=hello -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
hello-9k6dl	1/1	Running	0	12m	10.244.0.131	worker-node3
hello-b8k8r	1/1	Running	0	12m	10.244.1.3	worker-node1
hello-pj59s	1/1	Running	0	12m	10.244.1.131	worker-node2

Selector와 일치하는 Pod 집합의 IP/Port를 Endpoints 리소스로 관리

```
vagrant@hyojun:~$ kubectl describe svc hello-service
```

```
Name:          hello-service
Namespace:     default
Labels:        <none>
Annotations:   <none>
Selector:      app=hello
Type:          ClusterIP
IP:            10.225.30.68
Port:          <unset> 80/TCP
TargetPort:    8080/TCP
```

```
Endpoints:      10.244.0.131:8080,10.244.1.131:8080,10.244.1.3:8080
```

```
Session Affinity: None
```

```
Events:         <none>
```

```
vagrant@hyojun:~$ kubectl get pods -l app=hello -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
hello-9k6dl	1/1	Running	0	12m	10.244.0.131	worker-node3
hello-b8k8r	1/1	Running	0	12m	10.244.1.3	worker-node1
hello-pj59s	1/1	Running	0	12m	10.244.1.131	worker-node2

Selector와 일치하는 Pod 집합의 IP/Port를 Endpoints 리소스로 관리

```
vagrant@hyojun:~$ kubectl describe endpoints hello-service
```

Name: hello-service

Namespace: default

Endpoint 리소스는 Service와 같은 이름으로 생성됨

Labels: <none>

Annotations: endpoints.kubernetes.io/last-change-trigger-time: 2021-04-18T16:08:29Z

Subsets:

Addresses: 10.244.0.131,10.244.1.131,10.244.1.3

NotReadyAddresses: <none>

Ports:

Name	Port	Protocol
------	------	----------

----	----	-----
------	------	-------

<unset>	8080	TCP
---------	------	-----

Events: <none>

hello-9k6dl	1/1	Running	0	12m	10.244.0.131	worker-node3
hello-b8k8r	1/1	Running	0	12m	10.244.1.3	worker-node1
hello-pj59s	1/1	Running	0	12m	10.244.1.131	worker-node2

Selector가 없는 경우 (Endpoint 수동 생성)

```
apiVersion: v1
kind: Service
metadata:
  name: hello-service-no-selector
spec:
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
---
```

```
apiVersion: v1
kind: Endpoints
metadata:
  name: hello-service-no-selector
subsets:
  - addresses:
      - ip: 10.244.1.3
    ports:
      - port: 8080
```

Endpoint는 항상 Service 이름과 동일하게 지정해 주어야 한다.

Selector가 없는 경우 (Endpoint 수동 생성)

```
apiVersion: v1
kind: Service
metadata:
  name: hello-service-no-selector
spec:
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

```
apiVersion: v1
kind: Endpoints
metadata:
  name: hello-service-no-selector
subsets:
```

```
- addresses:
  - ip: 10.244.1.3
  ports:
  - port: 8080
```

참고: Endpoint에 지정된 IP는 외부에 존재하는 서버의 IP가 될 수 있다.
(클러스터 외부 IP로 트래픽을 전달하는 서비스)

Pod에서 서비스 식별 방법 (Service discovery)

1. 환경 변수(Environment variable)

- “서비스가 생성된 이후”에 만들어진 Pod에서만 사용 가능
- `{SVCNAME}_SERVICE_HOST` / `{SVCNAME}_SERVICE_PORT`
 - e.g. Service name: “foo-bar”
→ `FOO_BAR_SERVICE_HOST` / `FOO_BAR_SERVICE_PORT`

Pod에서 서비스 식별 방법 (Service discovery)

2. DNS 사용

- 클러스터 내에서 DNS 서버 역할을 수행하는 CoreDNS(또는 kube-dns)가 존재하는 경우
- DNS name format
 - `<service-name>.<namespace>.svc.cluster.local`
 - `<service-name>.<namespace>` (같은 로컬 클러스터인 경우)
 - `<service-name>` (같은 namespace인 경우)
- SRV Record 지원 ([RFC 2782](#))
 - 서비스 port에 이름을 지정한 경우,
SRV Record 조회를 통해 IP 주소와 함께 사용 가능한 port도 같이 확인 가능

Pod에서 서비스 식별 방법 (Service discovery)

2. DNS 사용

```
apiVersion: v1
kind: Service
metadata:
  name: hello-service
spec:
  selector:
    app: hello
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 8080
```

SRV Record 조회하기

```
vagrant@control-plane:~$ kubectl apply -f service.yaml
service/hello-service created
vagrant@control-plane:~$ kubectl apply -f https://k8s.io/examples/admin/dns/dnsutils.yaml
pod/dnsutils created
vagrant@control-plane:~$ kubectl exec -i -t dnsutils -- nslookup -type=SRV _http._tcp.hello-service
Server:      169.254.25.10
Address:     169.254.25.10#53

_http._tcp.hello-service.default.svc.cluster.local    service = 0 100 80 hello-service.default.svc.cluster.local.
```

Record 지원 ([RFC 2782](#))

service = <priority> <weight> <port> <RECORD>

- 서비스 port에 이름을 지정한 경우,
SRV Record 조회를 통해 IP 주소와 함께 사용 가능한 port도 같이 확인 가능

서비스로 들어온 트래픽을 Pod으로 전달하기까지

- 각 노드에는 kube-proxy Pod가 실행(daemonset)
 - kube-proxy는 Proxy mode에 따라 역할이 달라진다.
- Proxy mode
 - userspace mode
 - iptables mode
 - ipvs mode
- 필요한 사전 지식
 - iptables

서비스로 들어온 트래픽을 Pod으로 전달하기까지

- 각 노드에는 kube-proxy Pod가 실행(daemonset)
 - kube-proxy는 Proxy mode에 따라 역할이 달라진다.
- Proxy mode
 - userspace mode
 - iptables mode
 - ipvs mode
- 필요한 사전 지식
 - iptables

iptables

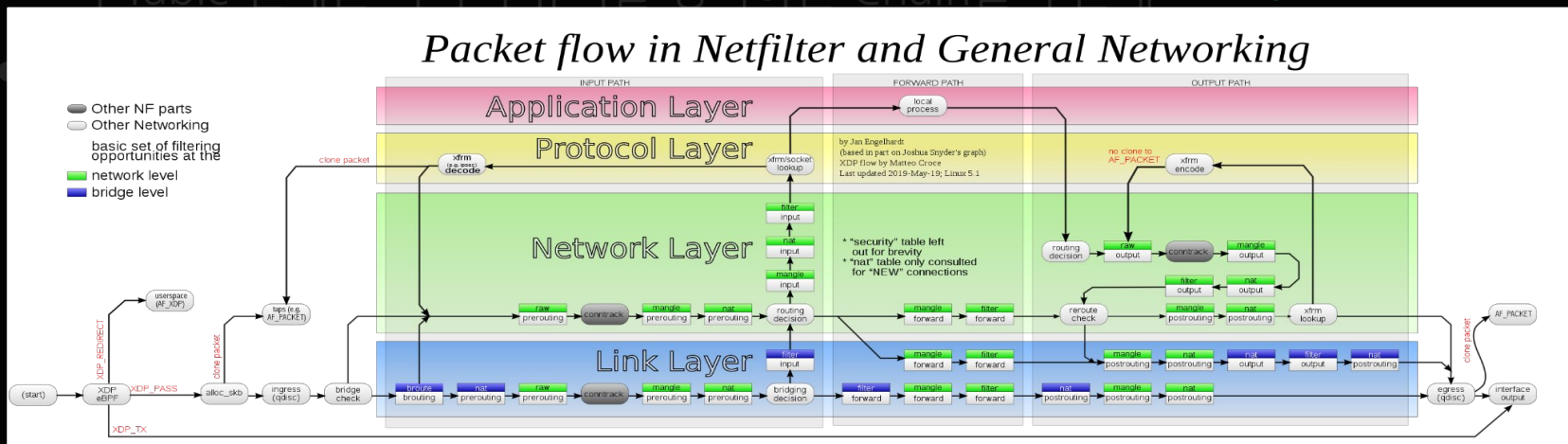
- Netfilter 기반으로 패킷을 필터링하거나 변환하는 방화벽 규칙을 정의
 - Netfilter - Linux 커널의 네트워크 관련 작업을 구현하기 위한 프레임워크
- 여러 종류의 Table 이 존재
 - Filter - 패킷을 필터링하거나 차단하는 규칙을 정의
 - NAT - 네트워크 주소 변환에 대한 규칙을 정의
 - mangle - 패킷 헤더 정보를 변경하는 규칙을 정의
 - 그 외 raw, security

iptables

- 각 Table 안에는 각각의 규칙을 정의하는 Chain들이 존재
- iptables에서 기본적으로 정의된 Chain들
 - PREROUTING - 패킷이 네트워크 인터페이스로 유입되면 가장 먼저 적용
 - FORWARD - 현재 호스트를 통해 라우팅 되는 패킷에 적용
 - INPUT - 해당 호스트에 존재하는 로컬 프로세스에 패킷이 유입될 때 적용
 - POSTROUTING - 패킷이 호스트 외부로 나갈 때 적용
 - OUTPUT - 로컬 프로세스에서 패킷이 생성된 직후 적용

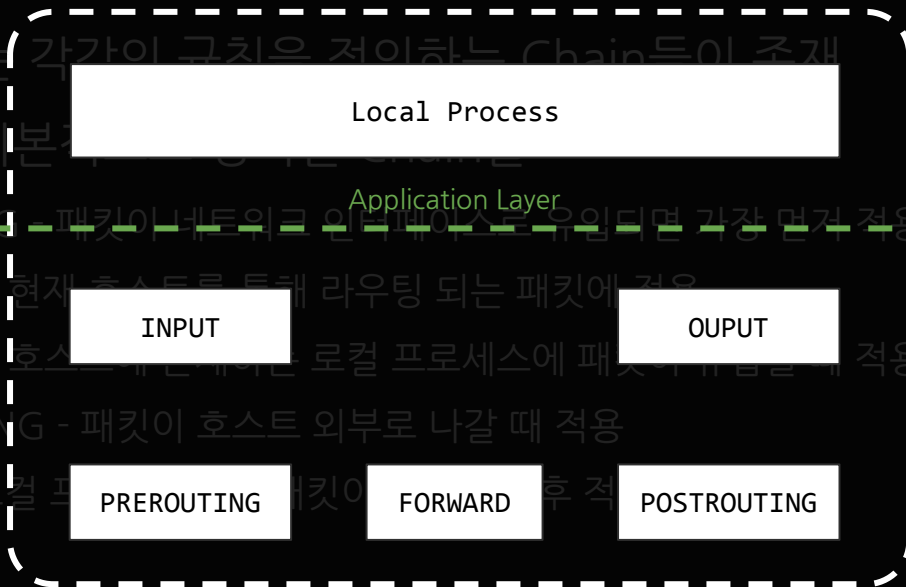
iptables 패킷 흐름

- 각 Table 안에는 각각의 규칙을 정의하는 ... <https://en.wikipedia.org/wiki/Iptables>

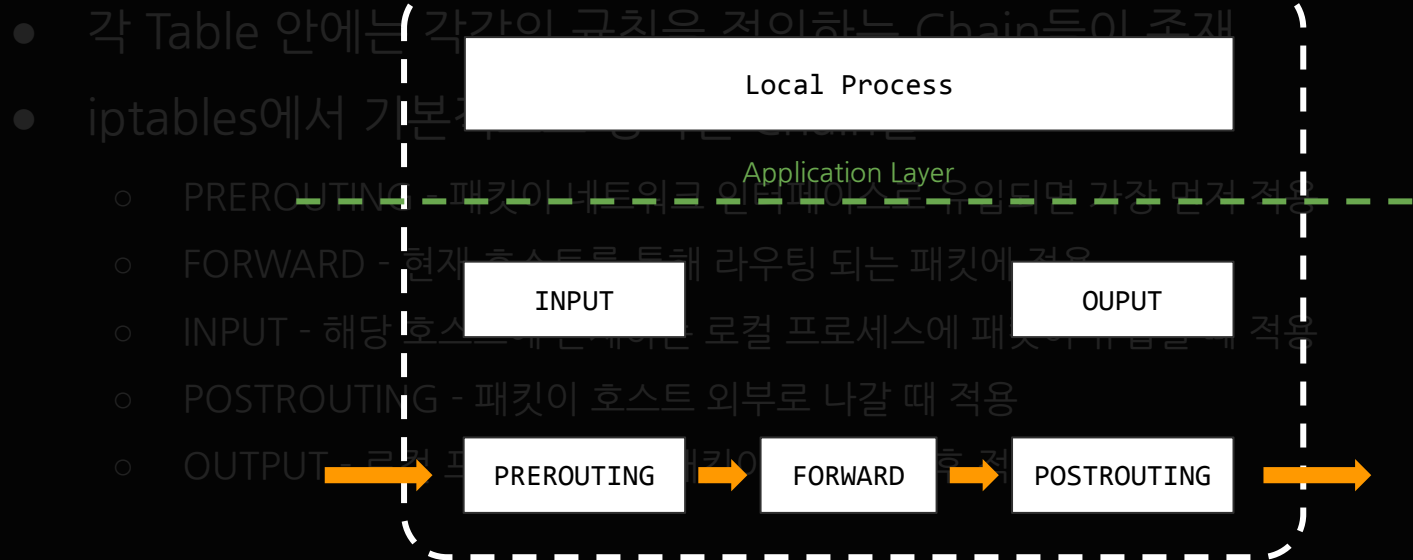


“간단하게 살펴보는” iptables Packet 흐름

- 각 Table 안에는 각각의 규칙을 정의하는 Chain들이 존재
- iptables에서 기본 Chain은 4가지로 나뉘어 있음
 - PREROUTING - 패킷이 네트워크 인터페이스로 유입되면 가장 먼저 적용
 - FORWARD - 현재 호스트를 통해 라우팅 되는 패킷에 적용
 - INPUT - 해당 호스트에 도착한 로컬 프로세스에 패킷이 유입될 때 적용
 - POSTROUTING - 패킷이 호스트 외부로 나갈 때 적용
 - OUTPUT - 로컬 프로세스에서 생성된 패킷이 호스트 외부로 나갈 때 적용



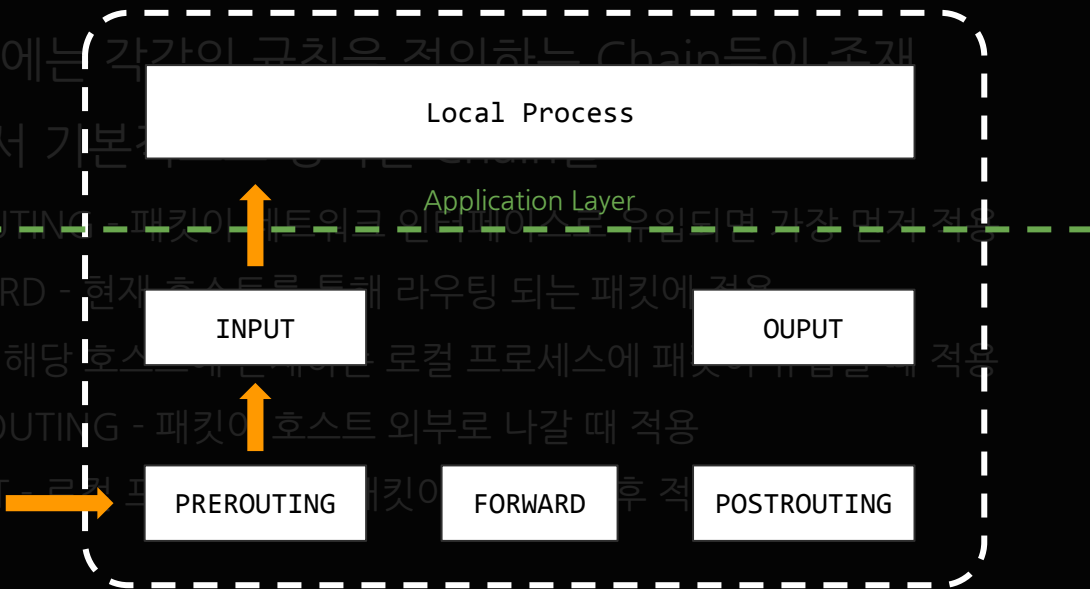
“간단하게 살펴보는” iptables Packet 흐름



호스트로 들어온 패킷이 다른 호스트로 포워딩되는 경우

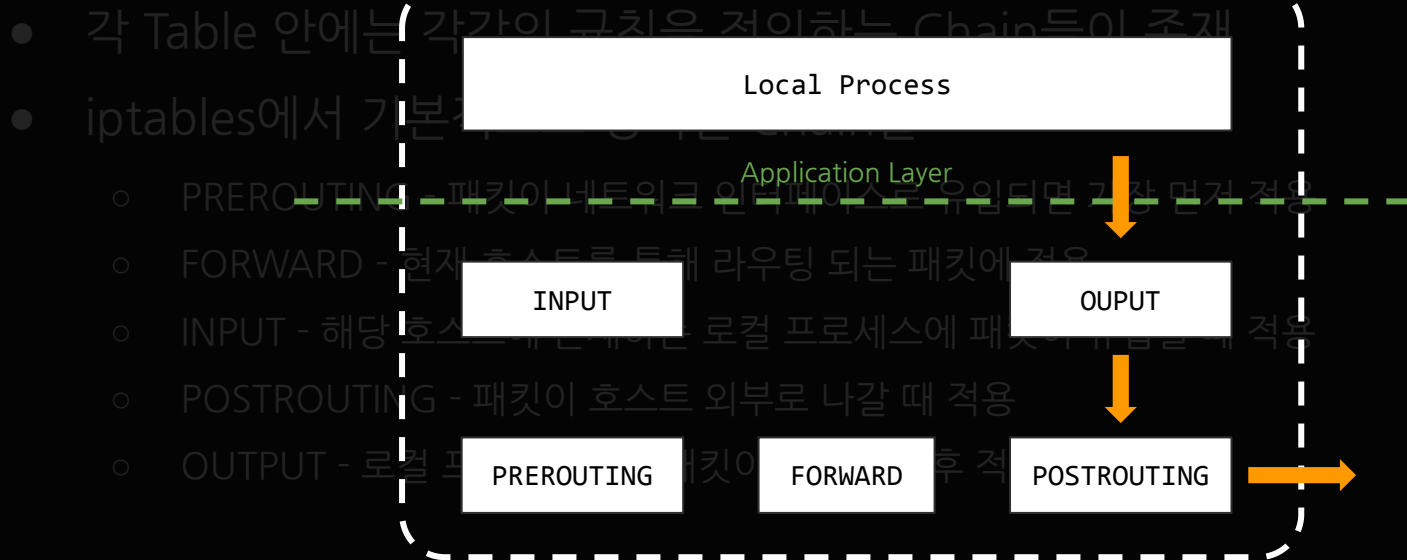
“간단하게 살펴보는” iptables Packet 흐름

- 각 Table 안에는 각각의 규칙을 정의하는 Chain들이 존재
- iptables에서 기본적인 Packet 흐름은 다음과 같다
 - PREROUTING - 패킷이 네트워크 인터페이스로 유입되면 가장 먼저 적용
 - FORWARD - 현재 호스트를 통해 라우팅 되는 패킷에 적용
 - INPUT - 해당 호스트에 유입되는 로컬 프로세스에 패킷이 유입될 때 적용
 - POSTROUTING - 패킷이 호스트 외부로 나갈 때 적용
 - OUTPUT - 로컬 프로세스에서 패킷이 생성될 때 적용



호스트로 들어온 패킷이 로컬 프로세스로 전달되는 경우

“간단하게 살펴보는” iptables Packet 흐름



로컬 프로세스에서 생성된 패킷이 호스트 밖으로 나가는 경우

NAT (Network Address Translation)

- 네트워크 주소 변환
 - DNAT (Destination Network Address Translation)
 - 목적지 네트워크 주소 변환
 - 예) 로드 밸런싱, 포트 포워딩
 - SNAT (Source Network Address Translation)
 - 출발지 네트워크 주소 변환
 - 예) 외부 네트워크(인터넷) 접속 시 패킷 Source IP를 Public IP로 변환

iptables를 이용한 DNAT 규칙 생성 예시

NAT Table(-t nat)의 PREROUTING Chain에 규칙 추가(-A PREROUTING)

```
$ iptables -t nat -A PREROUTING \  
-d 192.168.101.2 --dport 80 -p tcp \  
-j DNAT --to 1.2.3.4:80
```

iptables를 이용한 DNAT 규칙 생성 예시

```
$ iptables -t nat -A PREROUTING \
```

```
-d 192.168.101.2 --dport 80 -p tcp \
```

```
-j DNAT --to 1.2.3.4:80
```

192.168.101.2:80을 목적지 주소로 하는, TCP 패킷이 유입되면

iptables를 이용한 DNAT 규칙 생성 예시

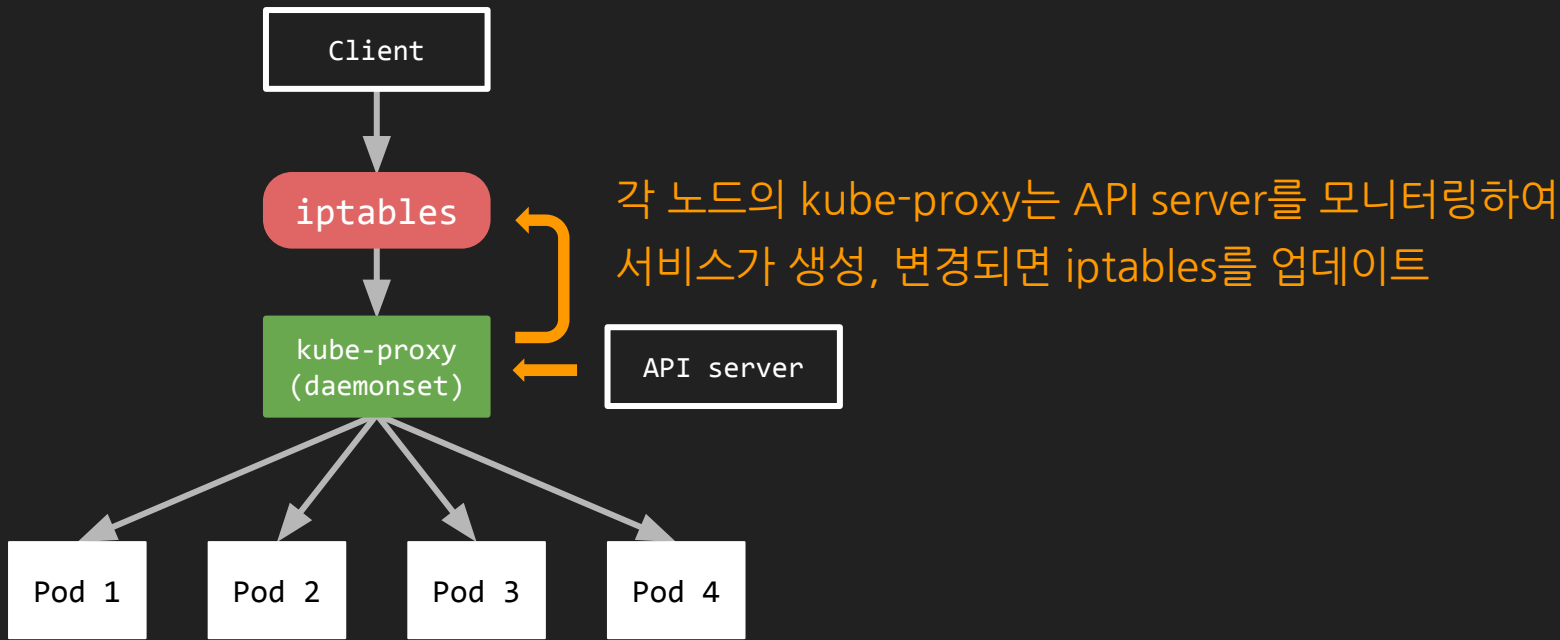
```
$ iptables -t nat -A PREROUTING \  
-d 192.168.101.2 --dport 80 -p tcp \  
-j DNAT --to 1.2.3.4:80
```

목적지 주소를 1.2.3.4:80으로 변환한다.

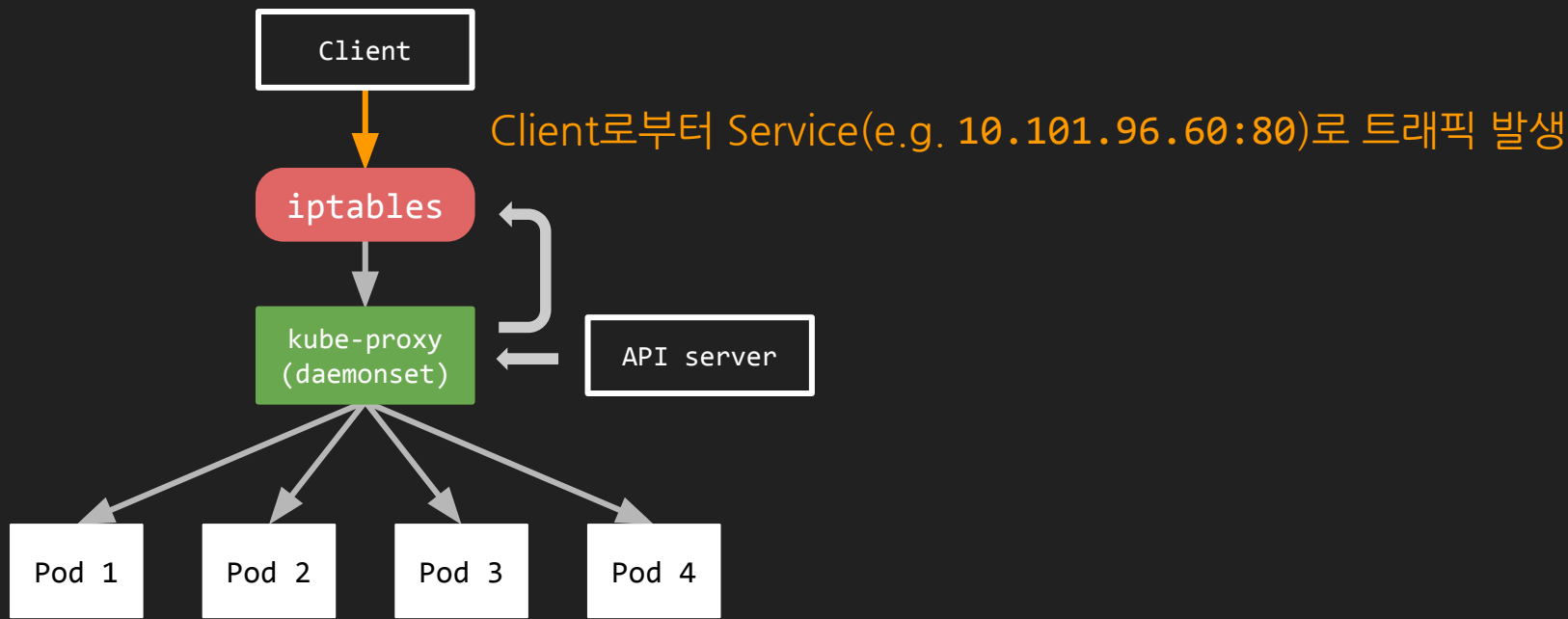
Proxy mode - (1) userspace

- kube-proxy는 API server로부터 Service 변경 감시하며 서비스 IP/Port로 향하는 패킷이 자신에게 전달되도록 iptables 업데이트
- 전달된 패킷은 kube-proxy가 백엔드 Pod로 직접 중개(Proxy)

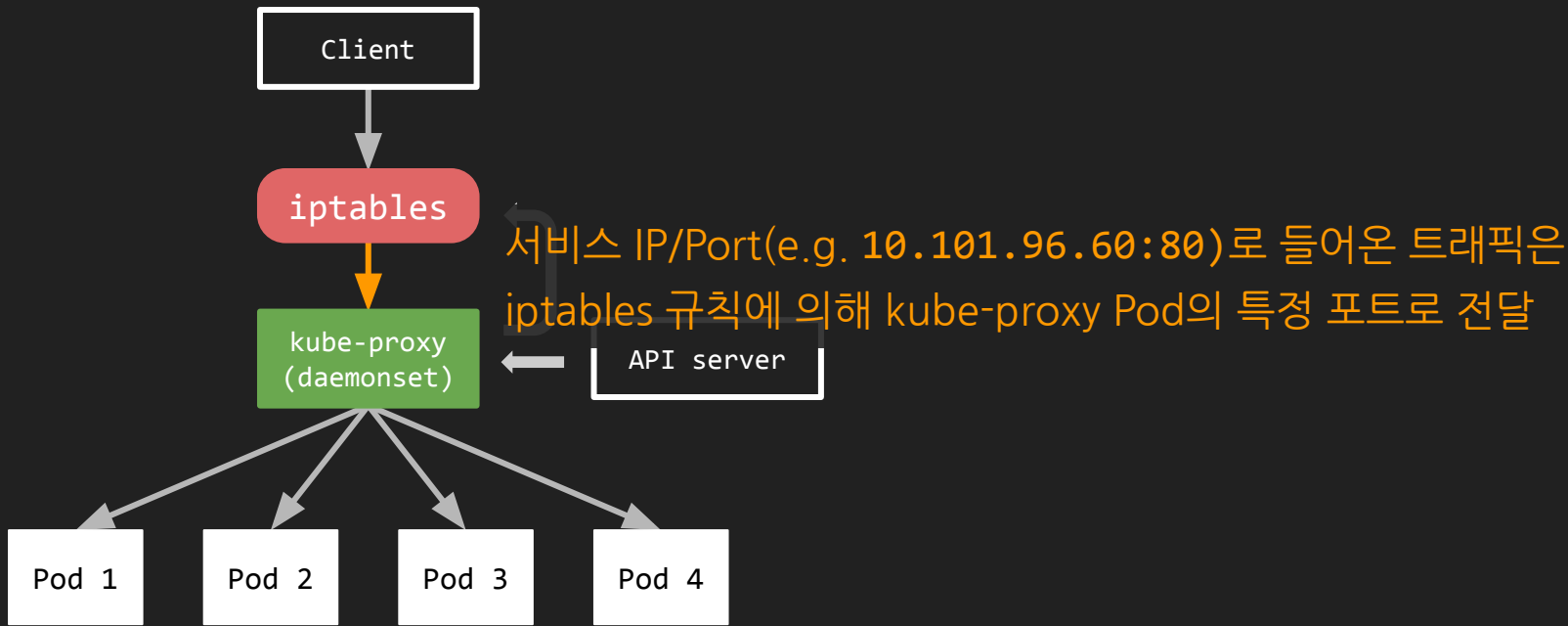
Proxy mode - (1) userspace



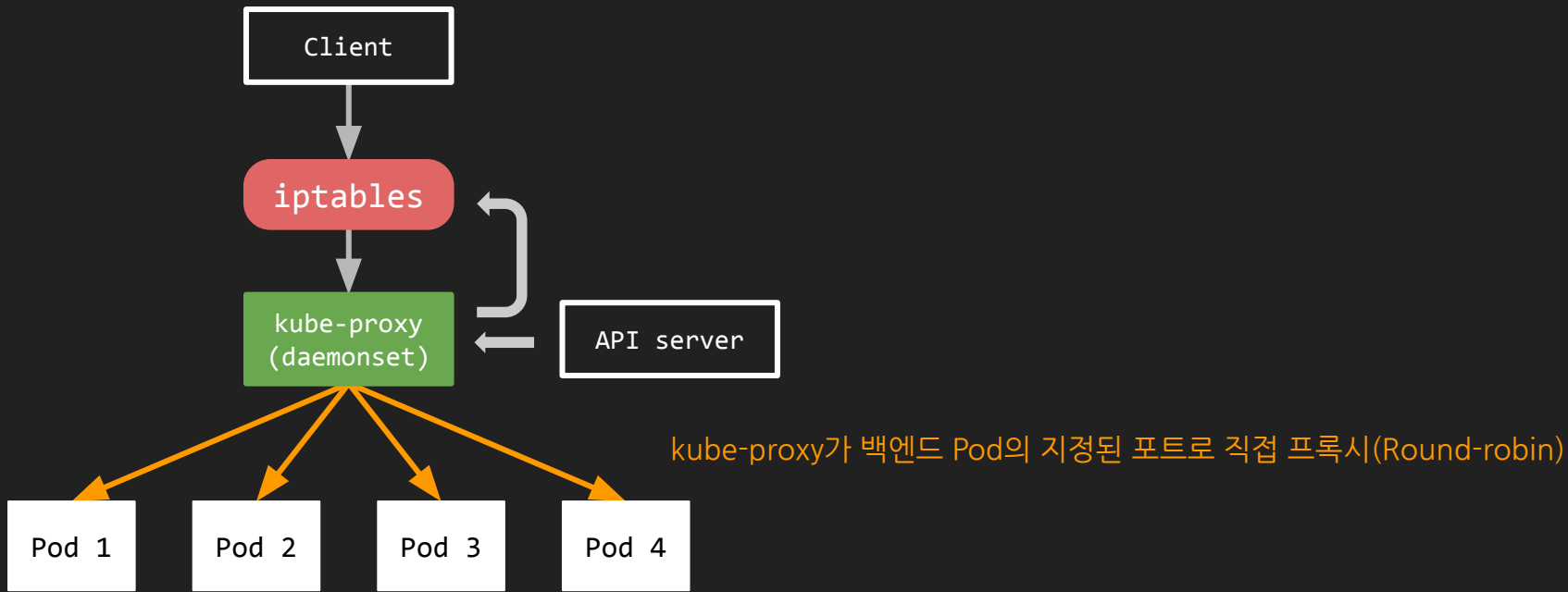
Proxy mode - (1) userspace



Proxy mode - (1) userspace



Proxy mode - (1) userspace



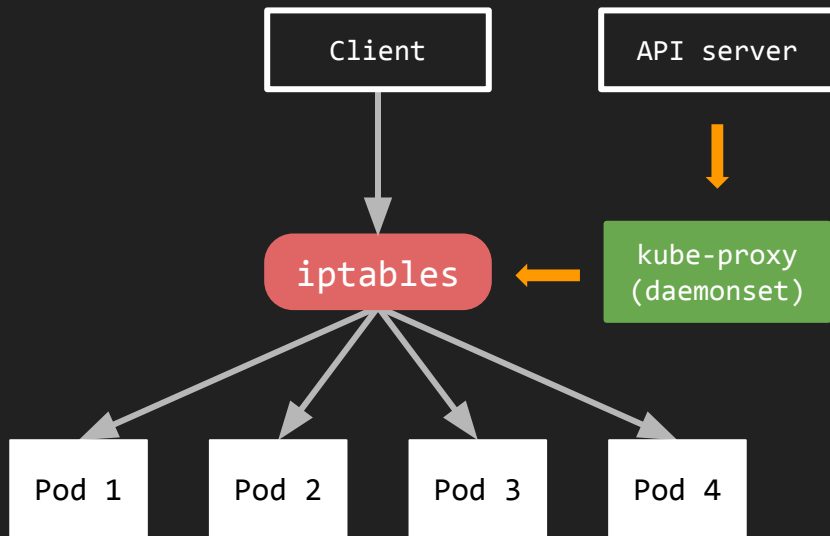
Proxy mode - (2) iptables

- 백엔드 Pod로 트래픽을 전달하는 것까지 모두 iptables 규칙으로 처리
 - 서비스 IP/Port로 향하는 트래픽은 iptables로 정의된 목적지 주소 변환(DNAT)에 의해 백엔드 Pod에 전달
 - DNAT = Destination Network Address Translation
- kube-proxy는 Kubernetes API server로부터 Service, Endpoint의 변경을 감지하여 iptables를 업데이트

Proxy mode - (2) iptables

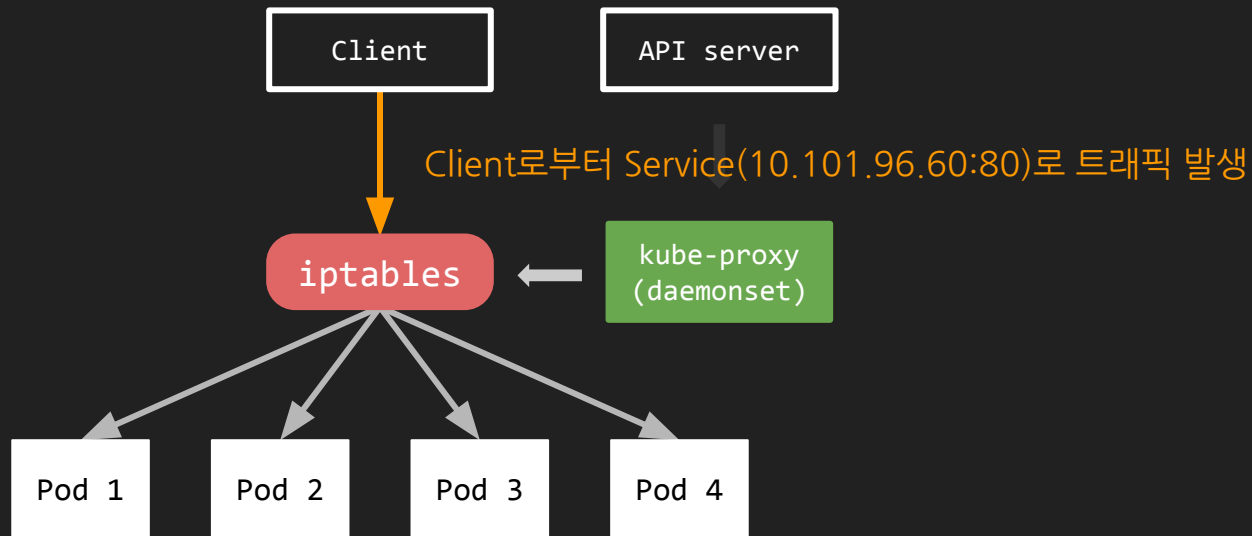
- userspace mode와 비교
 - iptables mode에서는 kube-proxy가 Proxy 서버 역할을 하지 않음
 - 커널 공간에서 처리되기 때문에 userspace 대비 오버헤드 감소
 - Pod이 응답하지 않는 경우?
 - userspace mode
 - kube-proxy가 다른 Pod에 Proxy
 - iptables mode
 - iptables 규칙으로 목적지 IP 주소가 이미 변환되었기 때문에, 다른 Pod로 트래픽 전달이 불가

Proxy mode - (2) iptables

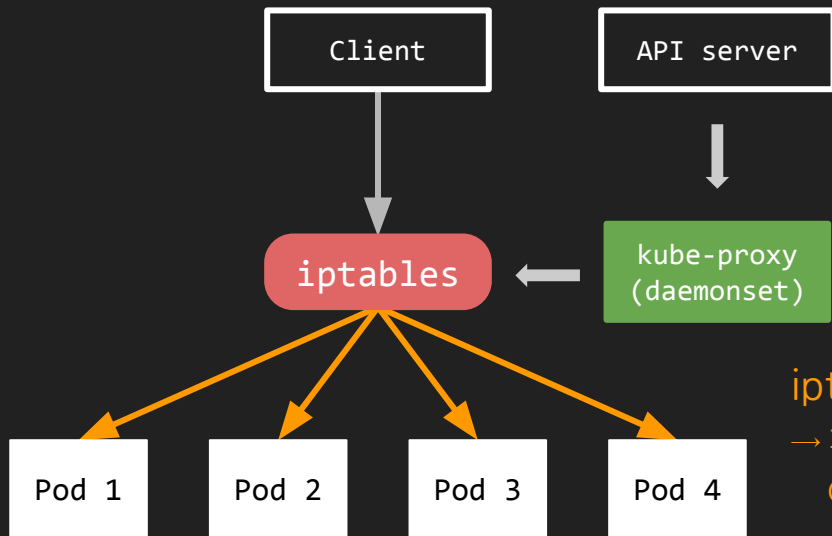


각 노드의 kube-proxy는 API server를 모니터링하여 Service 및 Endpoint가 생성, 변경되면 iptables를 업데이트

Proxy mode - (2) iptables



Proxy mode - (2) iptables



iptables 규칙에 의해 백엔드 Pod로 로드밸런싱
→ 패킷의 목적지 주소 및 포트가 백엔드 Pod로 변환되기 때문에,
Client는 대상 Pod와 직접 통신하게 된다.

Proxy mode - (2) iptables

```
$ sudo iptables -L -t nat
```

NAT table(-t nat)에 존재하는 모든 Chain을 listing(-L)

확인해봅시다 😎

Proxy mode - (2) iptables

```
vagrant@worker-node1:~$ sudo iptables -L -t nat
```

```
Chain PREROUTING (policy ACCEPT)
```

target	prot	opt	source	destination
--------	------	-----	--------	-------------

KUBE-SERVICES	all	--	anywhere	anywhere	/* kubernetes service portals */
---------------	-----	----	----------	----------	----------------------------------

DOCKER	all	--	anywhere	anywhere	ADDRTYPE match dst-type LOCAL
--------	-----	----	----------	----------	-------------------------------

Proxy mode - (2) iptables

```
vagrant@worker-node1:~$ sudo iptables -L -t nat
```

```
Chain PREROUTING (policy ACCEPT)
```

target	prot	opt	source	destination	
KUBE-SERVICES	all	--	anywhere	anywhere	/* kubernetes service portals */
DOCKER	all	--	anywhere	anywhere	ADDRTYPE match dst-type LOCAL



```
Chain KUBE-SERVICES (2 references)
```

target	prot	opt	source	destination	
KUBE-MARK-MASQ	tcp	--	!10.244.0.0/16	10.101.96.60	/* default/hello-service cluster IP */ tcp dpt:http
KUBE-SVC-WR2LASOULGGSX3JN	tcp	--	anywhere	10.101.96.60	/* default/hello-service cluster IP */ tcp dpt:http
KUBE-MARK-MASQ	tcp	--	!10.244.0.0/16	10.96.0.1	/* default/kubernetes-https cluster IP */ tcp dpt:https

Proxy mode - (2) iptables

```
vagrant@worker-node1:~$ sudo iptables -L -t nat
```

```
Chain PREROUTING (policy ACCEPT)
```

target	prot	opt	source	destination	
KUBE-SERVICES	all	--	anywhere	anywhere	/* kubernetes service portals */
DOCKER	all	--	anywhere	anywhere	ADDRTYPE match dst-type LOCAL

```
Chain KUBE-SERVICES (2 references)
```

target	prot	opt	source	destination	
KUBE-MARK-MASQ	tcp	--	!10.244.0.0/16	10.101.96.60	/* default/hello-service cluster IP */ tcp dpt:http
KUBE-SVC-WR2LASOULGGSX3JN	tcp	--	anywhere	10.101.96.60	/* default/hello-service cluster IP */ tcp dpt:http
KUBE-MARK-MASQ	tcp	--	!10.244.0.0/16	10.06.0.1	/* default/kubernetes https cluster IP */ tcp dpt:https

hello-service(10.101.96.60 + 80/tcp)로 들어온 패킷들에 대한 규칙

→ 서비스에 할당된 IP는 iptables로 정의된 가상의 IP 임을 알 수 있다. 항상 port와 함께 동작한다.

Proxy mode - (2) iptables

Chain KUBE-SERVICES (2 references)

target	prot	opt	source	destination	
KUBE-MARK-MASQ	tcp	--	!10.244.0.0/16	10.101.96.60	/* default/hello-service cluster IP */ tcp dpt:http
KUBE-SVC-WR2LASOULGGSX3JN	tcp	--	anywhere	10.101.96.60	/* default/hello-service cluster IP */ tcp dpt:http
KUBE-MARK-MASQ	tcp	--	10.244.0.0/16	10.96.0.1	/* default/kubernetes:https cluster IP */ tcp dpt:https

Chain KUBE-MARK-MASQ (14 references)

target	prot	opt	source	destination	
MARK	all	--	anywhere	anywhere	MARK or 0x4000

KUBE-MARK-MASQ

- 클러스터 외부에서 서비스로 들어온 패킷에 대해 SNAT 처리하도록 Marking
- Marking된 패킷은 호스트 밖으로 라우팅 되는 경우,
POSTROUTING chain을 통과하며 패킷의 출발지 IP가 현재 호스트 IP로 변환 (= Masquerade)

Proxy mode - (2) iptables

Chain KUBE-SERVICES (2 references)

target	prot	opt	source	destination	
KUBE-MARK-MASQ	tcp	--	!10.244.0.0/16	10.101.96.60	/* default/hello-service cluster IP */ tcp dpt:http
KUBE-SVC-WR2LASOULGGSX3JN	tcp	--	anywhere	10.101.96.60	/* default/hello-service cluster IP */ tcp dpt:http
KUBE-MARK-MASQ	tcp	--	!10.244.0.0/16	10.96.0.1	/* default/kubernetes:https cluster IP */ tcp dpt:https

hello-service(10.101.96.60 + 80/tcp)로 들어온 패킷은 “KUBE-SVC-WR2LASOULGGSX3JN” Chain으로 전달

Chain KUBE-SVC-WR2LASOULGGSX3JN (1 references)

target	prot	opt	source	destination	
KUBE-SEP-JYKRBLVI3TUDQCLU	all	--	anywhere	anywhere	/* default/hello-service */ statistic mode random probability 0.5000000000
KUBE-SEP-KC5MKKT3MKNRXG3M	all	--	anywhere	anywhere	/* default/hello-service */

Proxy mode - (2) iptables

Chain KUBE-SERVICES (2 references)

target	prot	opt	source	destination	
KUBE-MARK-MASQ	tcp	--	!10.244.0.0/16	10.101.96.60	/* default/hello-service cluster IP */ tcp dpt:http
KUBE-SVC-WR2LASOULGGSX3JN	tcp	--	anywhere	10.101.96.60	/* default/hello-service cluster IP */ tcp dpt:http
KUBE-MARK-MASQ	tcp	--	10.244.0.0/16	10.96.0.1	/* default/kubernetes:https cluster IP */ tcp dpt:https

Chain KUBE-SVC-WR2LASOULGGSX3JN (1 references)

target	prot	opt	source	destination	
KUBE-SEP-JYKRBLVI3TUDQCLU	all	--	anywhere	anywhere	/* default/hello-service */ statistic mode random probability 0.5000000000
KUBE-SEP-KC5MKKT3MKNRXG3M	all	--	anywhere	anywhere	/* default/hello-service */

Pod 별로 생성되는 “KUBE-SEP-...” Chain에는 Pod IP/Port로 DNAT 규칙이 정의됨.
서비스의 백엔드 Pod가 현재 2개인 상태로, 2개 Pod Chain 중에 50% 확률로 무작위 선택

Proxy mode - (2) iptables

Chain KUBE-SVC-WR2LASOULGGSX3JN (1 references)

target	prot	opt	source	destination
--------	------	-----	--------	-------------

KUBE-SEP-JYKRBLVI3TUDQCLU	all	--	anywhere	anywhere
---------------------------	-----	----	----------	----------

/* default/hello-service */ statistic mode random probability 0.5000000000

KUBE-SEP-KC5MKKT3MKNRXG3M	all	--	anywhere	anywhere
---------------------------	-----	----	----------	----------

/* default/hello-service */

Chain KUBE-SEP-JYKRBLVI3TUDQCLU (1 references)

target	prot	opt	source	destination
--------	------	-----	--------	-------------

KUBE-MARK-MASQ	all	--	10.244.1.3	anywhere
----------------	-----	----	------------	----------

/* default/hello-service */

DNAT	tcp	--	anywhere	anywhere
------	-----	----	----------	----------

/* default/hello-service */ tcp to:10.244.1.3:8080

10.244.1.3:8080 Pod로 DNAT

Chain KUBE-SEP-KC5MKKT3MKNRXG3M (1 references)

target	prot	opt	source	destination
--------	------	-----	--------	-------------

KUBE-MARK-MASQ	all	--	10.244.2.3	anywhere
----------------	-----	----	------------	----------

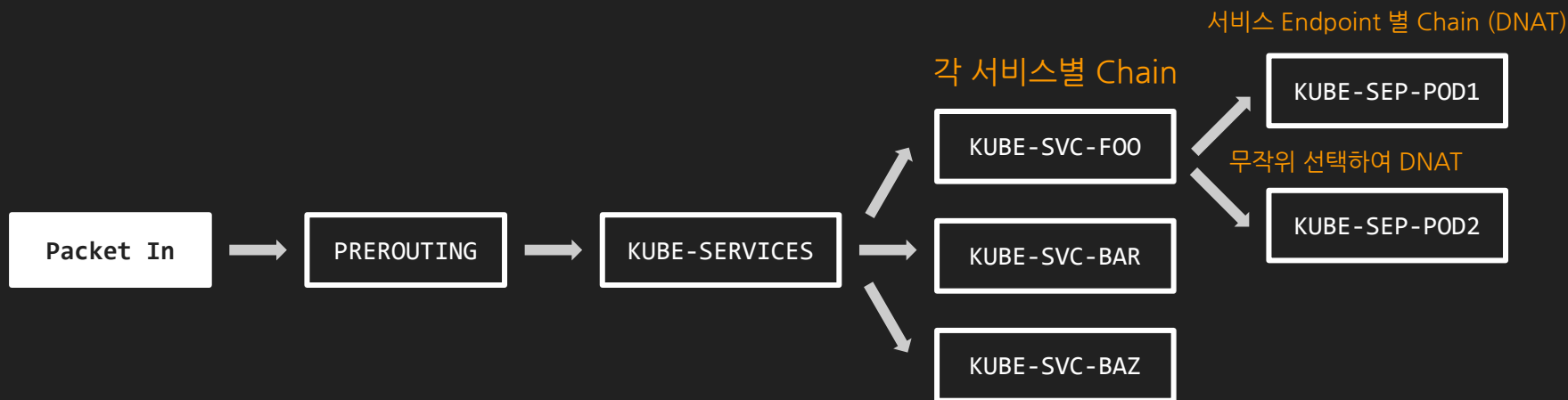
/* default/hello-service */

DNAT	tcp	--	anywhere	anywhere
------	-----	----	----------	----------

/* default/hello-service */ tcp to:10.244.2.3:8080

10.244.2.3:8080 Pod로 DNAT

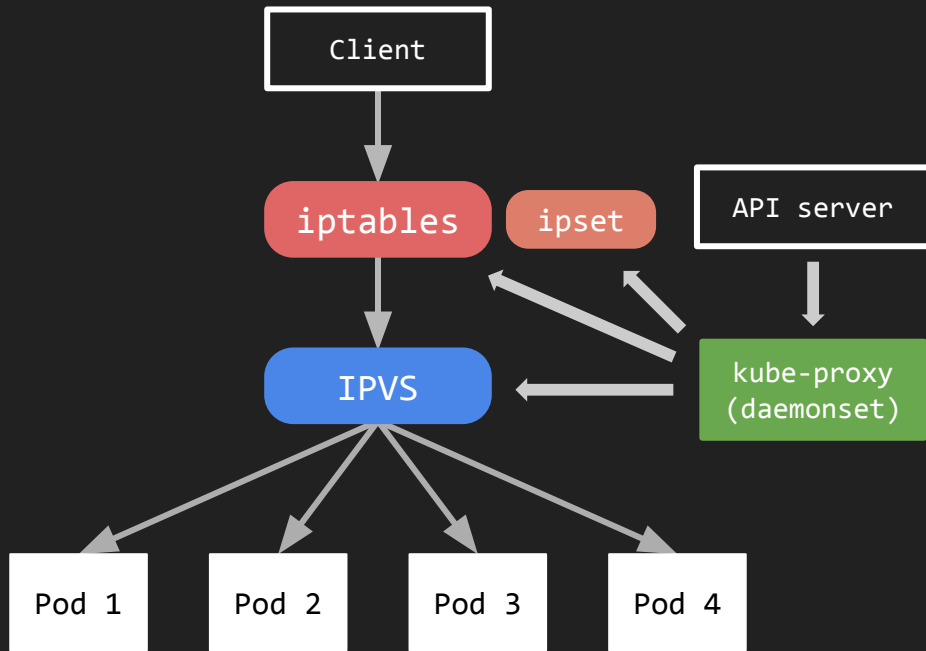
Proxy mode - (2) iptables



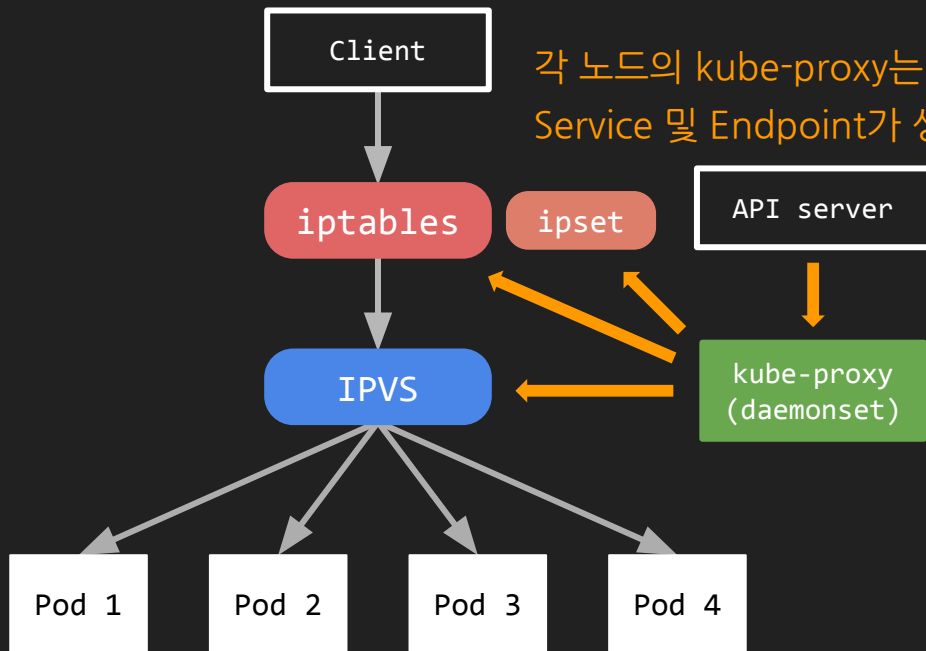
Proxy mode - (3) IPVS

- IPVS(IP Virtual Server)란?
 - IPVS는 커널 계층에서 제공되는 L4 Load Balancer
 - iptables는 패킷 필터링과 패킷 조작을 통한 유연성으로 방화벽을 구현하기 위해 설계됨
 - 반면에, ipvs는 로드밸런싱을 위해 설계된 커널 기능
 - iptables와 동일하게 Netfilter를 기반
 - 커널 내부의 해시테이블을 기반으로, 많은 수의 서비스와 Pod에도 일관된 성능
 - iptables의 경우 순차적으로 규칙들을 따라가면서 처리하다 보니 성능에 한계가 있음
 - 최단 예상 지연시간, 최소 연결 등 좀 더 정교한 로드 밸런싱 알고리즘 제공

Proxy mode - (3) IPVS

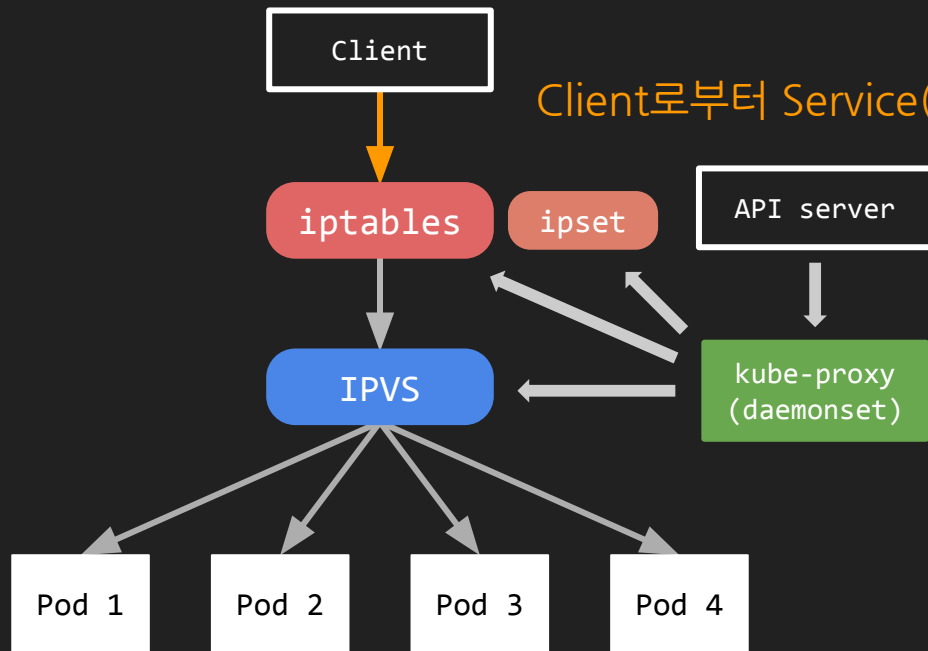


Proxy mode - (3) IPVS



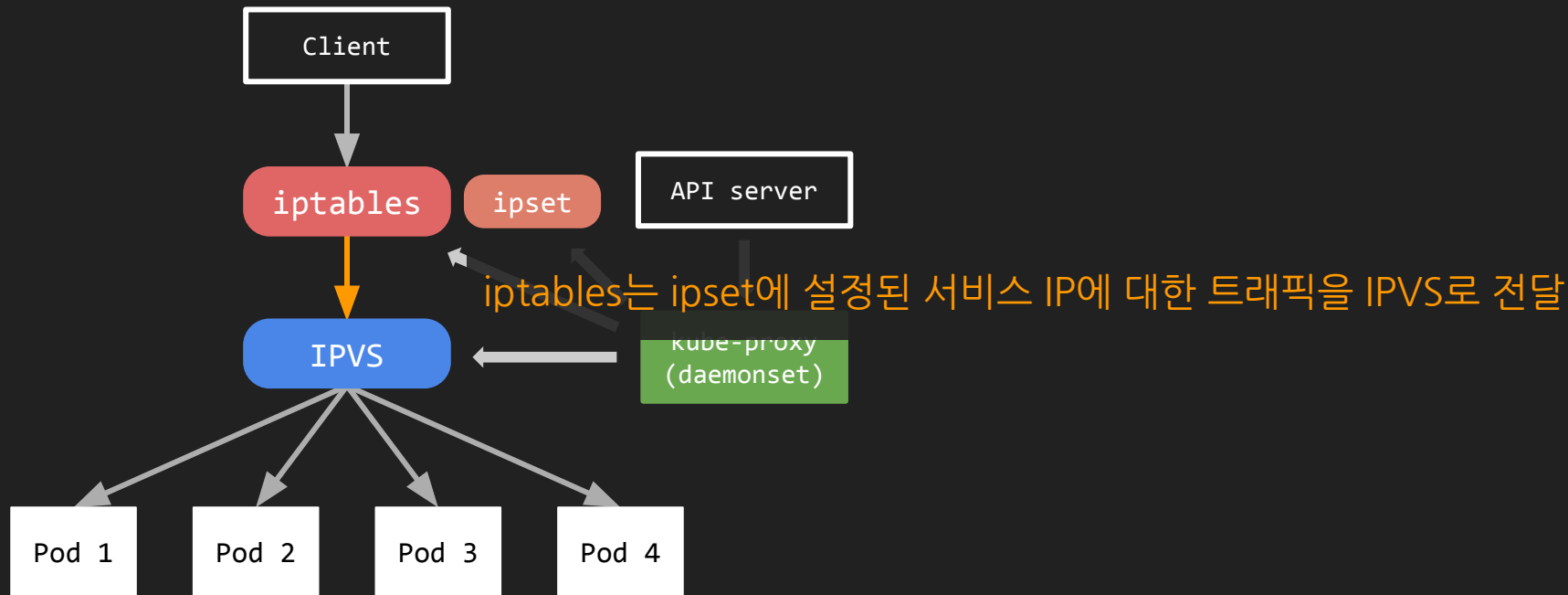
각 노드의 kube-proxy는 API server를 모니터링하다가 Service 및 Endpoint가 생성 또는 변경되면 iptables, ipset, IPVS를 업데이트

Proxy mode - (3) IPVS

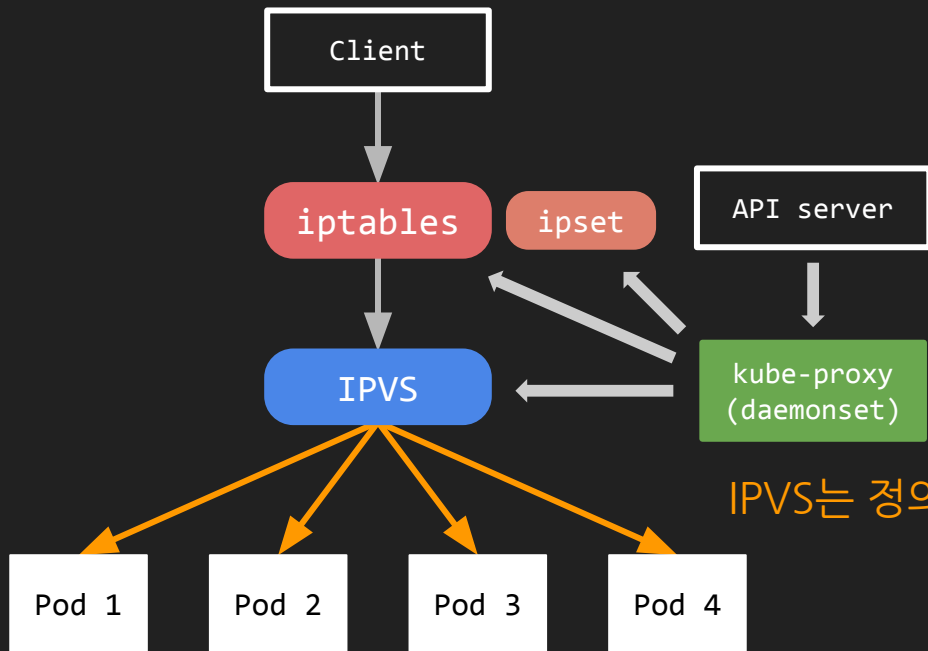


Client로부터 Service(e.g. 10.101.96.60:80)로 트래픽 발생

Proxy mode - (3) IPVS



Proxy mode - (3) IPVS



IPVS는 정의된 규칙에 따라 백엔드 Pod로 로드 밸런싱

Proxy mode - (3) IPVS

```
vagrant@control-plane:~$ kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
hello-service	ClusterIP	10.225.122.243	<none>	80/TCP	24h
kubernetes	ClusterIP	10.225.0.1	<none>	443/TCP	24h

Proxy mode - (3) IPVS

```
vagrant@control-plane:~$ kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
hello-service	ClusterIP	10.225.122.243	<none>	80/TCP	24h
kubernetes	ClusterIP	10.225.0.1	<none>	443/TCP	24h

Chain PREROUTING (policy ACCEPT)

target	prot	opt	source	destination	
KUBE-SERVICES	all	--	anywhere	anywhere	/* kubernetes service portals */
DOCKER	all	--	anywhere	anywhere	ADDRTYPE match dst-type LOCAL


Proxy mode - (3) IPVS

```
vagrant@control-plane:~$ kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
hello-service	ClusterIP	10.225.122.243	<none>	80/TCP	24h
kubernetes	ClusterIP	10.225.0.1	<none>	443/TCP	24h

```
Chain PREROUTING (policy ACCEPT)
```

target	prot	opt	source	destination	
KUBE-SERVICES	all	--	anywhere	anywhere	/* kubernetes service portals */
DOCKER	all	--	anywhere	anywhere	ADDRTYPE match dst-type LOCAL



```
Chain KUBE-SERVICES (2 references)
```

target	prot	opt	source	destination	
KUBE-MARK-MASQ	all	--	!control-plane/16	anywhere	/* Kubernetes service cluster ip + po
KUBE-NODE-PORT	all	--	anywhere	anywhere	ADDRTYPE match dst-type LOCAL
ACCEPT	all	--	anywhere	anywhere	match-set KUBE-CLUSTER-IP dst,dst

Proxy mode - (3) IPVS

Chain KUBE-SERVICES (2 references)

target	prot	opt	source	destination
KUBE-MARK-MASQ	all	--	!control-plane/16	anywhere
KUBE-NODE-PORT	all	--	anywhere	anywhere
ACCEPT	all	--	anywhere	anywhere

```
/* Kubernetes service cluster ip + port  
ADDRTYPE match dst-type LOCAL  
match-set KUBE-CLUSTER-IP dst,dst
```

Proxy mode - (3) IPVS

Chain KUBE-SERVICES (2 references)

target	prot	opt	source	destination	
KUBE-MARK-MASQ	all	--	!control-plane/16	anywhere	/* Kubernetes service cluster ip + po
KUBE-NODE-PORT	all	--	anywhere	anywhere	ADDRTYPE match dst-type LOCAL
ACCEPT	all	--	anywhere	anywhere	match-set KUBE-CLUSTER-IP dst,dst

```
vagrant@control-plane:~$ sudo ipset list KUBE-CLUSTER-IP
```

Name: KUBE-CLUSTER-IP

Type: hash:ip,port

Revision: 5

Header: family inet hashsize 1024 maxelem 65536

Size in memory: 536

References: 2

Number of entries: 7

Members:

10.225.116.240,tcp:8000

10.225.0.1,tcp:443

10.225.122.243,tcp:80

10.225.88.218,tcp:443

10.225.0.3,tcp:9153

10.225.0.3,udp:53

10.225.0.3,tcp:53

ipset 을 사용하여 각 서비스마다 너무 많은 iptables Chain이 생기는 것을 방지

iptables proxy mode에서...



클러스터에 정의된 서비스들의 IP, Protocol:Port

Proxy mode - (3) IPVS

각 서비스 IP가 Dummy NIC(Network Interface Controller)에 binding 된 것을 알 수 있다.

```
vagrant@control-plane:~$ sudo ipset list KUBE-CLUSTER-IP
Name: KUBE-CLUSTER-IP
Type: hash:ip,port
Revision: 5
Header: family inet hashsize 1024 maxelem 65536
Size in memory: 536
References: 2
Number of entries: 7
Members:
10.225.116.240,tcp:8000
10.225.0.1,tcp:443
10.225.122.243,tcp:80
10.225.88.218,tcp:443
10.225.0.3,tcp:9153
10.225.0.3,udp:53
10.225.0.3,tcp:53
```

```
vagrant@control-plane:~$ ip addr show kube-ipvs0
5: kube-ipvs0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN group default
    link/ether b2:1f:27:8e:a5:11 brd ff:ff:ff:ff:ff:ff
    inet 10.225.0.1/32 scope global kube-ipvs0
        valid_lft forever preferred_lft forever
    inet 10.225.0.3/32 scope global kube-ipvs0
        valid_lft forever preferred_lft forever
    inet 10.225.88.218/32 scope global kube-ipvs0
        valid_lft forever preferred_lft forever
    inet 10.225.116.240/32 scope global kube-ipvs0
        valid_lft forever preferred_lft forever
    inet 10.225.122.243/32 scope global kube-ipvs0
        valid_lft forever preferred_lft forever
```

Proxy mode - (3) IPVS

10.225.122.243:80로 들어온 트래픽은 Pod IP와 정의된 targetPort로 로드 밸런싱

```
vagrant@control-plane:~$ ip addr show kube-ipvs0
5: kube-ipvs0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN group default
    link/ether b2:1f:27:8e:a5:11 brd ff:ff:ff:ff:ff:ff
    inet 10.225.0.1/32 scope global kube-ipvs0
        valid_lft forever preferred_lft forever
    inet 10.225.0.3/32 scope global kube-ipvs0
        valid_lft forever preferred_lft forever
    inet 10.225.88.218/32 scope global kube-ipvs0
        valid_lft forever preferred_lft forever
    inet 10.225.116.240/32 scope global kube-ipvs0
        valid_lft forever preferred_lft forever
    inet 10.225.122.243/32 scope global kube-ipvs0
        valid_lft forever preferred_lft forever
```

```
vagrant@control-plane:~$ sudo ipvsadm -ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  10.225.0.1:443 rr
  -> 192.168.101.2:6443            Masq    1      3          0
TCP  10.225.0.3:53 rr
  -> 10.245.0.2:53                  Masq    1      0          0
  -> 10.245.1.134:53                Masq    1      0          0
TCP  10.225.0.3:9153 rr
  -> 10.245.0.2:9153                Masq    1      0          0
  -> 10.245.1.134:9153              Masq    1      0          0
TCP  10.225.88.218:443 rr
  -> 10.245.1.130:8443              Masq    1      0          0
TCP  10.225.116.240:8000 rr
  -> 10.245.1.132:8000              Masq    1      0          0
TCP  10.225.122.243:80 rr
  -> 10.245.1.131:8080              Masq    1      0          0
  -> 10.245.1.133:8080              Masq    1      0          0
UDP  10.225.0.3:53 rr
  -> 10.245.0.2:53                  Masq    1      0          0
  -> 10.245.1.134:53                Masq    1      0          0
```

rr = Round robin

Round-robin DNS를 사용하지 않는 이유

- Round-robin DNS
 - 서비스마다 DNS를 부여하고 백엔드 Pod들의 IP를 Record로 나열하여 DNS Round-robin 방식으로 로드 밸런싱하는 방법
- Kubernetes에서는 DNS Round-robin을 정식 Proxy-mode로 지원하지 않는다.
 - DNS Record TTL를 고려하지 않고, DNS 검색 결과를 캐싱하는 경우가 있음
 - 일부 앱은 무기한으로 캐싱하는 경우도...
 - TTL이 낮은 경우 DNS 서버의 부하가 높아질 수 있음

클러스터 외부에서 접근 가능한 서비스

- 서비스는 기본적으로 클러스터 내부 IP로 노출 (서비스 타입: Cluster IP)
- 클러스터 외부로 서비스를 노출하려면?
 - 서비스 타입
 - NodePort
 - LoadBalancer
 - External IP 설정

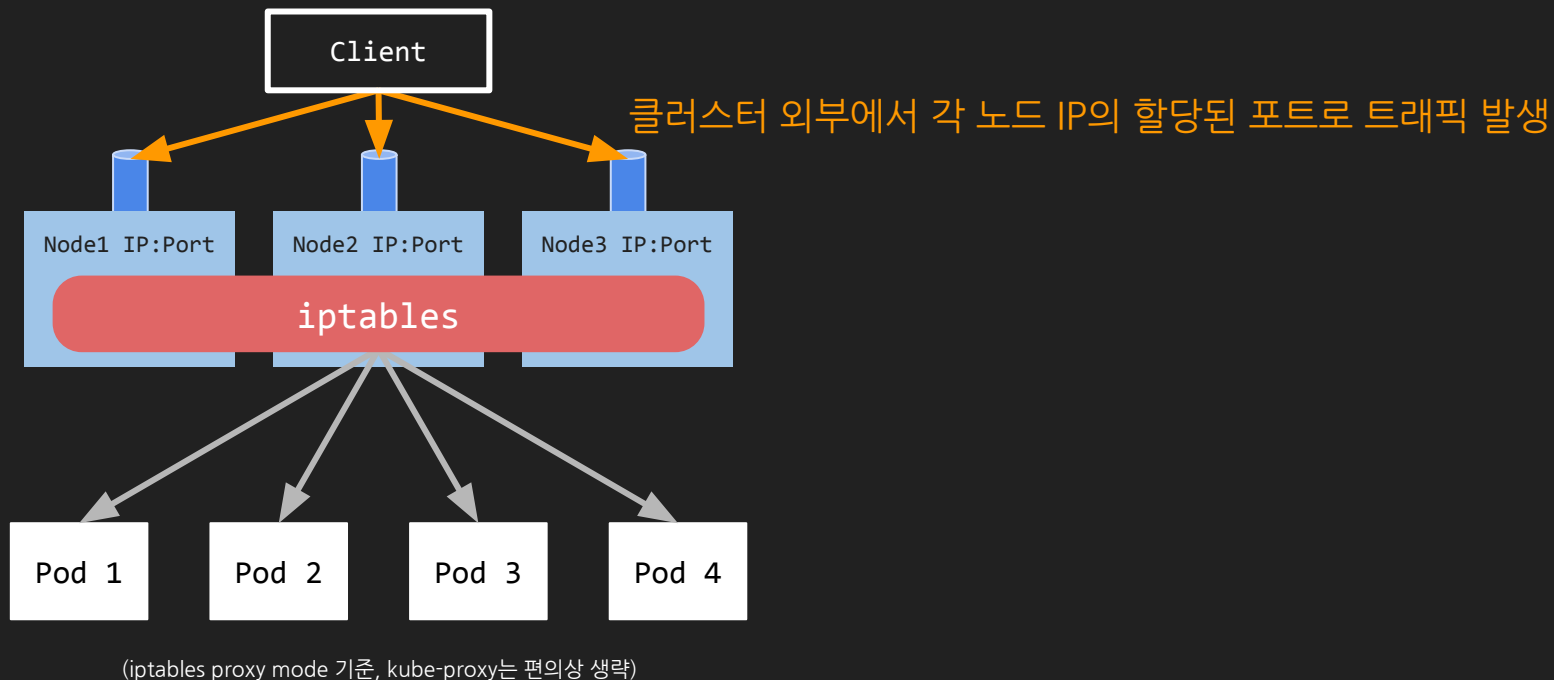
Service Type - (1) NodePort

- 클러스터 외부에서 “<NodeIP>:<NodePort>”로 서비스에 접근 가능한 서비스 타입
- `nodePort`를 지정하는 경우
 - 노드 IP들의 지정된 포트로 서비스 노출
- `nodePort`를 지정하지 않는 경우
 - `--service-node-port-range`로 지정된 포트 범위 안에서 노드 IP에 포트를 자동 할당 (기본값 : 30000-32767)
 - 포트 충돌을 방지하기 위해, 따로 포트를 설정하지 않고 Kubernetes에 포트 할당을 위임하는 것 이 좋다.

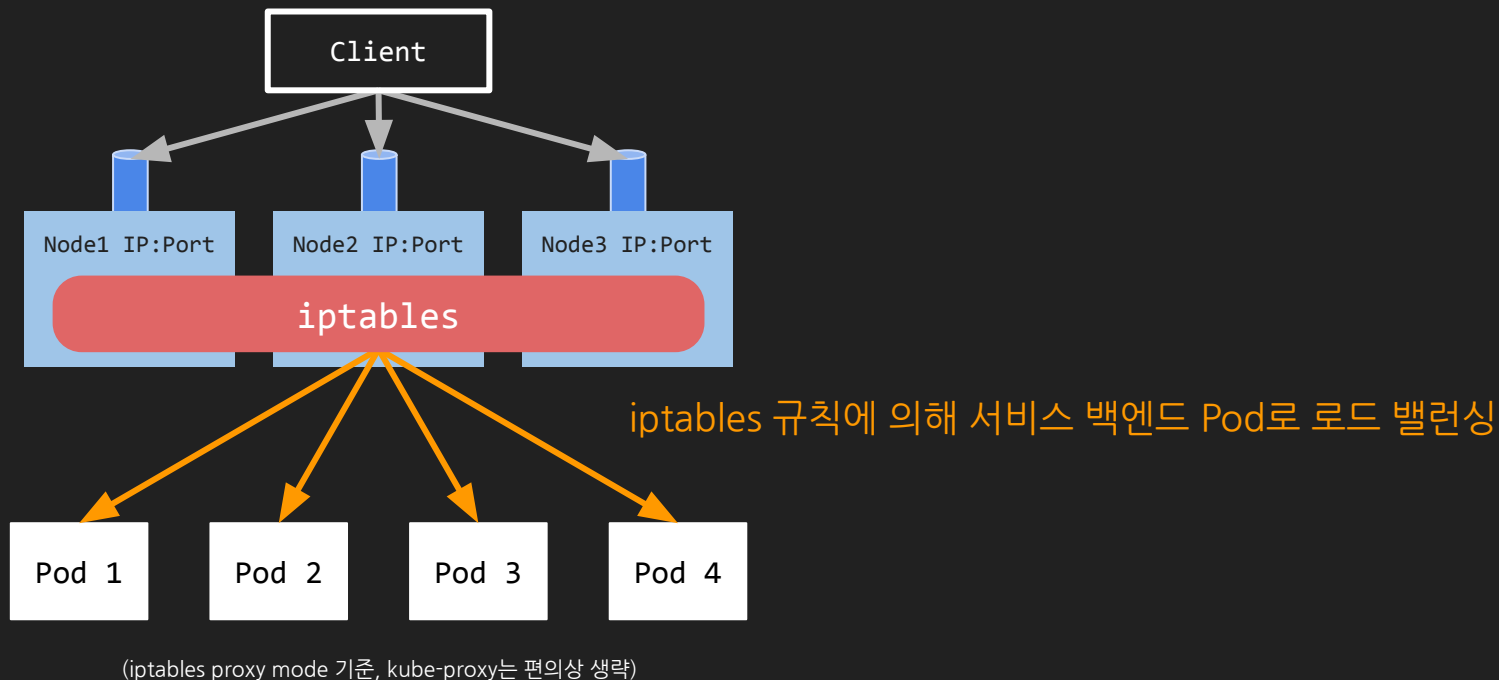
service-nodeport.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: hello-nodeport-service
spec:
  type: NodePort
  selector:
    app: hello
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 8080
      nodePort: 30080 # optional
```

Service Type - (1) NodePort



Service Type - (1) NodePort



Service Type - (1) NodePort

```
vagrant@control-plane:~$ sudo iptables -L -t nat
Chain PREROUTING (policy ACCEPT)
target     prot opt source                destination
KUBE-SERVICES all  -- anywhere             anywhere             /* kubernetes service portals */
DOCKER     all  -- anywhere             anywhere             ADDRTYPE match dst-type LOCAL
```

Chain KUBE-SERVICES (2 references)

```
target     prot opt source                destination
KUBE-MARK-MASQ tcp  -- !control-plane/16    10.96.0.10          /* kube-system/kube-dns:metrics cluster IP */ tcp dpt:9153
KUBE-SVC-JD5MR3NA4I4DYORP tcp  -- anywhere            10.96.0.10          /* kube-system/kube-dns:metrics cluster IP */ tcp dpt:9153
KUBE-MARK-MASQ tcp  -- !control-plane/16    10.101.53.58        /* default/hello-nodeport-service:http cluster IP */ tcp dpt:http
KUBE-SVC-KGTTXZQT54FX5FOU tcp  -- anywhere            10.101.53.58        /* default/hello-nodeport-service:http cluster IP */ tcp dpt:http
KUBE-MARK-MASQ tcp  -- !control-plane/16    10.101.96.60        /* default/hello-service cluster IP */ tcp dpt:http
KUBE-SVC-WR2LASOULGGSX3JN tcp  -- anywhere            10.101.96.60        /* default/hello-service cluster IP */ tcp dpt:http
KUBE-MARK-MASQ tcp  -- !control-plane/16    10.96.0.1           /* default/kubernetes:https cluster IP */ tcp dpt:https
KUBE-SVC-NPX46M4PTMTKRN6Y tcp  -- anywhere            10.96.0.1           /* default/kubernetes:https cluster IP */ tcp dpt:https
KUBE-MARK-MASQ udp  -- !control-plane/16    10.96.0.10          /* kube-system/kube-dns:dns cluster IP */ udp dpt:domain
KUBE-SVC-TCOU7JCQXEZGVUNU udp  -- anywhere            10.96.0.10          /* kube-system/kube-dns:dns cluster IP */ udp dpt:domain
KUBE-MARK-MASQ tcp  -- !control-plane/16    10.96.0.10          /* kube-system/kube-dns:tcp cluster IP */ tcp dpt:domain
KUBE-SVC-FRTFTISQEP7F70F4 tcp  -- anywhere            10.96.0.10          /* kube-system/kube-dns:tcp cluster IP */ tcp dpt:domain
KUBE-NODEPORTS all  -- anywhere            anywhere            /* kubernetes service nodeports; NOTE: this must be the last rule in this chain */ ADDRTYPE match dst-type LOCAL
```

노드 IP + nodePort로 유입된 트래픽은 KUBE-NODEPORTS Chain으로 전달

Service Type - (1) NodePort

Chain KUBE-SERVICES (2 references)

target	prot	opt	source	destination	
KUBE-MARK-MASQ	tcp	--	!control-plane/16	10.96.0.10	/* kube-system/kube-dns:metrics cluster IP */ tcp dpt:9153
KUBE-SVC-JD5MR3NA4I4DYORP	tcp	--	anywhere	10.96.0.10	/* kube-system/kube-dns:metrics cluster IP */ tcp dpt:9153
KUBE-MARK-MASQ	tcp	--	!control-plane/16	10.101.53.58	/* default/hello-nodeport-service:http cluster IP */ tcp dpt:http
KUBE-SVC-KGTTXZQT54FX5FOU	tcp	--	anywhere	10.101.53.58	/* default/hello-nodeport-service:http cluster IP */ tcp dpt:http
KUBE-MARK-MASQ	tcp	--	!control-plane/16	10.101.96.60	/* default/hello-service cluster IP */ tcp dpt:http
KUBE-SVC-WR2LASOULGGSX3JN	tcp	--	anywhere	10.101.96.60	/* default/hello-service cluster IP */ tcp dpt:http
KUBE-MARK-MASQ	tcp	--	!control-plane/16	10.96.0.1	/* default/kubernetes:https cluster IP */ tcp dpt:https
KUBE-SVC-NPX46M4PTMTKRN6Y	tcp	--	anywhere	10.96.0.1	/* default/kubernetes:https cluster IP */ tcp dpt:https
KUBE-MARK-MASQ	udp	--	!control-plane/16	10.96.0.10	/* kube-system/kube-dns:dns cluster IP */ udp dpt:domain
KUBE-SVC-TCOU7JCQXEZGVUNU	udp	--	anywhere	10.96.0.10	/* kube-system/kube-dns:dns cluster IP */ udp dpt:domain
KUBE-MARK-MASQ	tcp	--	!control-plane/16	10.96.0.10	/* kube-system/kube-dns:dns-tcp cluster IP */ tcp dpt:domain
KUBE-SVC-ERITXISQEP7E70E4	tcp	--	anywhere	10.96.0.10	/* kube-system/kube-dns:dns-tcp cluster IP */ tcp dpt:domain
KUBE-NODEPORTS	all	--	anywhere	anywhere	/* kubernetes service nodeports; NOTE: this must be the last rule in this chain */ ADDRTYPE match dst-type LOCAL

Chain KUBE-NODEPORTS (1 references)

target	prot	opt	source	destination	
KUBE-MARK-MASQ	tcp	--	anywhere	anywhere	/* default/hello-nodeport-service:http */ tcp dpt:30080
KUBE-SVC-KGTTXZQT54FX5FOU	tcp	--	anywhere	anywhere	/* default/hello-nodeport-service:http */ tcp dpt:30080

Service Type - (1) NodePort

Chain KUBE-NODEPORTS (1 references)

target	prot	opt	source	destination	
KUBE-MARK-MASQ	tcp	--	anywhere	anywhere	/* default/hello-nodeport-service:http */ tcp dpt:30080
KUBE-SVC-KGTTXZQT54FX5F0U	tcp	--	anywhere	anywhere	/* default/hello-nodeport-service:http */ tcp dpt:30080



지정된 nodePort로 유입된 TCP Packet은 해당 서비스 Chain으로 전달

Chain KUBE-SVC-KGTTXZQT54FX5F0U (2 references)

target	prot	opt	source	destination	
KUBE-SEP-BUGHN5FR5NR2IMX3	all	--	anywhere	anywhere	/* default/hello-nodeport-service:http */ statistic mode random probability 0.5000000000
KUBE-SEP-K05D5IEUXAZK7RXL	all	--	anywhere	anywhere	/* default/hello-nodeport-service:http */

Service Type - (1) NodePort

Chain KUBE-NODEPORTS (1 references)

target	prot	opt	source	destination	
KUBE-MARK-MASQ	tcp	--	anywhere	anywhere	/* default/hello-nodeport-service:http */ tcp dpt:30080
KUBE-SVC-KGTTXZQT54FX5FOU	tcp	--	anywhere	anywhere	/* default/hello-nodeport-service:http */ tcp dpt:30080

Chain KUBE-SVC-KGTTXZQT54FX5FOU (2 references)

target	prot	opt	source	destination	
KUBE-SEP-BUGHN5FR5NR2IMX3	all	--	anywhere	anywhere	/* default/hello-nodeport-service:http */ statistic mode random probability 0.5000000000
KUBE-SEP-K05D5IEUXAZK7RXL	all	--	anywhere	anywhere	/* default/hello-nodeport-service:http */

해당 서비스의 백엔드 Pod으로 로드 밸런싱

Chain KUBE-SEP-BUGHN5FR5NR2IMX3 (1 references)

target	prot	opt	source	destination	
KUBE-MARK-MASQ	all	--	10.244.1.3	anywhere	/* default/hello-nodeport-service:http */
DNAT	tcp	--	anywhere	anywhere	/* default/hello-nodeport-service:http */ tcp to:10.244.1.3:8080

Chain KUBE-SEP-K05D5IEUXAZK7RXL (1 references)

target	prot	opt	source	destination	
KUBE-MARK-MASQ	all	--	10.244.2.3	anywhere	/* default/hello-nodeport-service:http */
DNAT	tcp	--	anywhere	anywhere	/* default/hello-nodeport-service:http */ tcp to:10.244.2.3:8080

Service Type - (2) LoadBalancer

- 외부 로드 밸런서를 지원하는 Cloud Service (AWS, GCP, Azure, etc.)에서, 로드 밸런서를 프로비저닝 하여 서비스를 외부에 노출
 - AWS EKS
 - <https://docs.aws.amazon.com/eks/latest/userguide/load-balancing.html>
 - GCP GKE
 - https://cloud.google.com/kubernetes-engine/docs/how-to/exposing-apps#creating_a_service_of_type_loadbalancer
 - Azure
 - <https://docs.microsoft.com/azure/aks/load-balancer-standard>

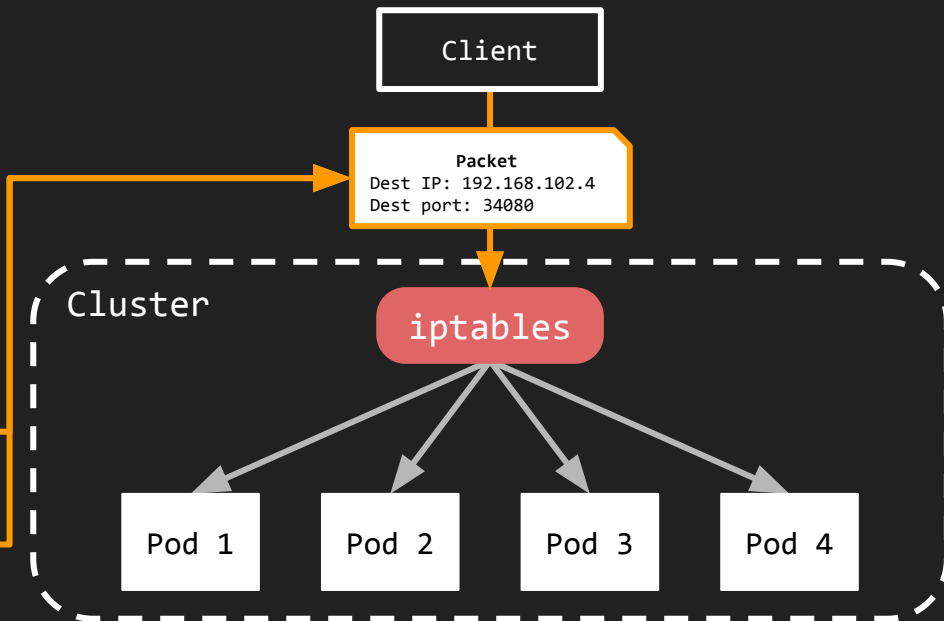
External IP 설정

- NodePort, LoadBalancer 타입의 서비스를 정의하지 않고도, 클러스터로 트래픽을 전달하는 특정 외부 IP로 서비스를 노출할 수 있다.
 - External IP는 Kubernetes에서 관리하지 않는다.
- 외부 IP가 목적지이면서, 목적지 포트가 특정 서비스 포트와 일치하는 패킷이 클러스터로 들어오면 해당 서비스의 백엔드 Pod으로 전달된다.

External IP 설정

패킷의 목적지 IP가 서비스의 External IP와 일치하고,
목적지 Port가 서비스 Port와 일치하는 패킷이 클러스터로 유입

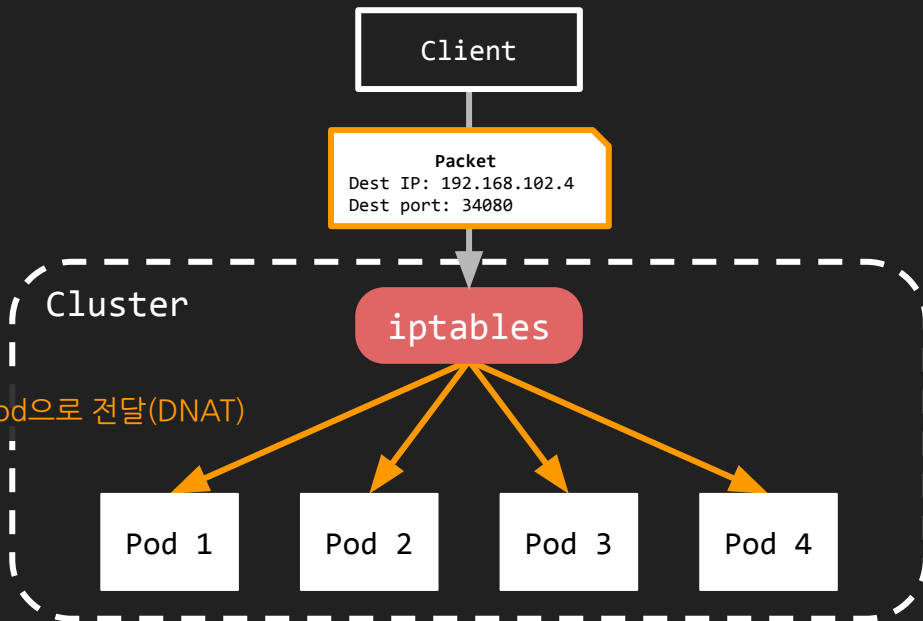
```
apiVersion: v1
kind: Service
metadata:
  name: hello-external-ip-service
spec:
  selector:
    app: hello
  ports:
    - protocol: TCP
      port: 34080
      targetPort: 8080
  externalIPs:
    - 192.168.102.4
```



External IP 설정

```
apiVersion: v1
kind: Service
metadata:
  name: hello-external-ip-service
spec:
  selector:
    app: hello
  ports:
    - protocol: TCP
      port: 34080
      targetPort: 8080
  externalIPs:
    - 192.168.102.4
```

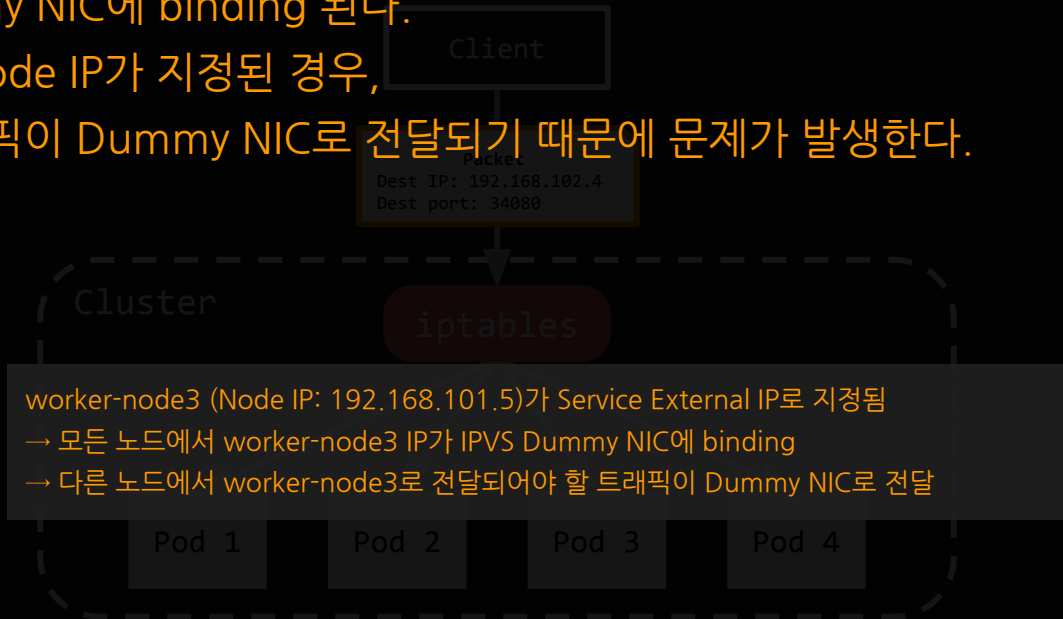
패킷은 서비스의 백엔드 Pod으로 전달(DNAT)



IPVS Proxy mode에서 External IP 설정 시 주의 사항

- IPVS proxy mode에서는 External IP도 서비스 Cluster IP와 마찬가지로 IPVS 로드 밸런싱을 위해 Dummy NIC에 binding 된다.
- 따라서, 서비스 External IP로 Node IP가 지정된 경우, 해당 노드로 전달되어야 할 트래픽이 Dummy NIC로 전달되기 때문에 문제가 발생한다.

```
vagrant@worker-node1:~$ sudo ip addr show kube-ipvs0
5: kube-ipvs0: <BROADCAST,NOARP> mtu 1500 qdisc noop
    link/ether 22:28:ba:73:9a:93 brd ff:ff:ff:ff:ff:f
    inet 10.225.0.1/32 scope global kube-ipvs0
        valid_lft forever preferred_lft forever
    inet 10.225.0.3/32 scope global kube-ipvs0
        valid_lft forever preferred_lft forever
    inet 10.225.88.218/32 scope global kube-ipvs0
        valid_lft forever preferred_lft forever
    inet 10.225.116.240/32 scope global kube-ipvs0
        valid_lft forever preferred_lft forever
    inet 10.225.114.94/32 scope global kube-ipvs0
        valid_lft forever preferred_lft forever
    inet 192.168.101.5/32 scope global kube-ipvs0
        valid_lft forever preferred_lft forever
```



외부 도메인을 참조하는 Service Type - ExternalName

- 클러스터 외부의 FQDN(Fully Qualified Domain Name)를 참조하는 서비스를 정의하여, 서비스 DNS Name으로 접근 가능
- Cluster IP가 할당되지 않고, DNS level에서 구현

apiVersion: v1

kind: Service

metadata:

name: search-engine

spec:

type: ExternalName

externalName: google.com

```
vagrant@control-plane:~$ kubectl exec -i -t dnsutils -- nslookup search-engine.default.svc.cluster.local
Server:      10.96.0.10
Address:     10.96.0.10#53

search-engine.default.svc.cluster.local canonical name = google.com.
Name:   google.com
Address: 172.217.174.110
Name:   google.com
Address: 2404:6800:4004:810::200e
```

외부 도메인을 참조하는 Service Type - ExternalName

- 클러스터 외부의 FQDN(Fully Qualified Domain Name)를 참조하는 서비스를 정의하여, 서비스 DNS Name으로 접근 가능
- Cluster IP가 할당되지 않고, DNS level에서 구현

apiVersion: v1

kind: Service

metadata:

name: search-engine

spec:

type: ExternalName

externalName: google.com

```
vagrant@control-plane:~$ kubectl exec -i -t dnsutils -- nslookup search-engine.default.svc.cluster.local
Server:      10.96.0.10
Address:     10.96.0.10#53

search-engine.default.svc.cluster.local canonical name = google.com.
Name:   google.com
Address: 172.217.174.110
Name:   google.com
Address: 2404:6800:4004:810::200e
```

서비스 DNS

search-engine.default.svc.cluster.local

search-engine.default (같은 로컬 클러스터인 경우)

search-engine (같은 namespace인 경우)



CNAME

google.com

Headless 서비스

- Cluster IP를 명시적으로 할당하지 않는 서비스
 - 서비스 IP, 로드 밸런싱이 필요하지 않은 경우 사용
예) 서비스 DNS Name가 EndPoint IP를 직접 반환하도록 함
 - Kubernetes 서비스 구현에 의존하지 않고
다른 Service discovery mechanism을 사용할 수 있도록 보장

```
apiVersion: v1
kind: Service
metadata:
  name: hello-headless-service
spec:
  clusterIP: None
  selector:
    app: hello
```

```

vagrant@control-plane:~$ kubectl exec -i -t dnsutils -- nslookup hello-headless-service
Server:      10.96.0.10
Address:     10.96.0.10#53 Headless 서비스는 selector에 일치하는 Pod IP들이 DNS Record로 반환

Name:   hello-headless-service.default.svc.cluster.local
Address: 10.244.1.3
Name:   hello-headless-service.default.svc.cluster.local
Address: 10.244.2.3

vagrant@control-plane:~$ kubectl exec -i -t dnsutils -- nslookup hello-service
Server:      10.96.0.10
Address:     10.96.0.10#53 일반적인 서비스의 경우, 서비스 IP가 DNS Record로 반환

Name:   hello-service.default.svc.cluster.local
Address: 10.99.239.179
```

Session Affinity - Client IP

- 특정 클라이언트의 연결이 매번 동일한 파드로 전달되도록 설정
 - `sessionAffinity`를 "ClientIP"로 설정(기본값: None)
 - 클라이언트의 IP 주소를 기반
 - `sessionAffinityConfig.clientIP.timeoutSeconds`
 - 최대 세션 고정시간 (기본값: 3시간)

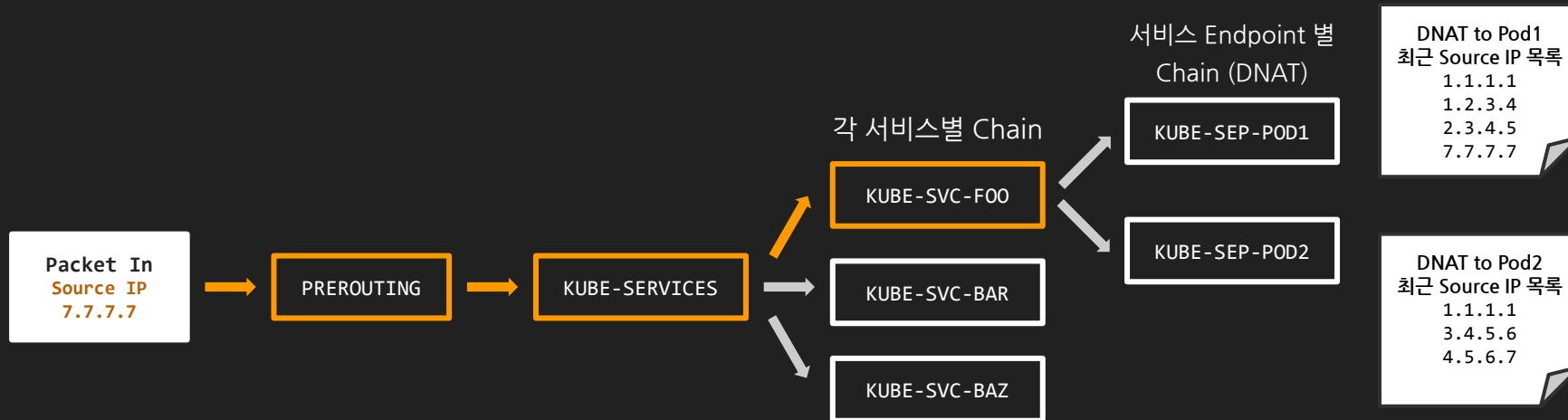
Session Affinity - Client IP

iptables proxy mode에서 동작 원리

- “recent” 모듈로 SessionAffinity 구현
 - 동적으로 IP 주소 목록을 생성하고 확인 가능한 iptables 모듈
 - recent 모듈을 사용한 방화벽 구현 사례)
특정 목적지 IP로 전달된 패킷들의 Source IP 목록을 저장해두고,
1분 동안 100번 이상 패킷을 전송한 Source IP를 차단

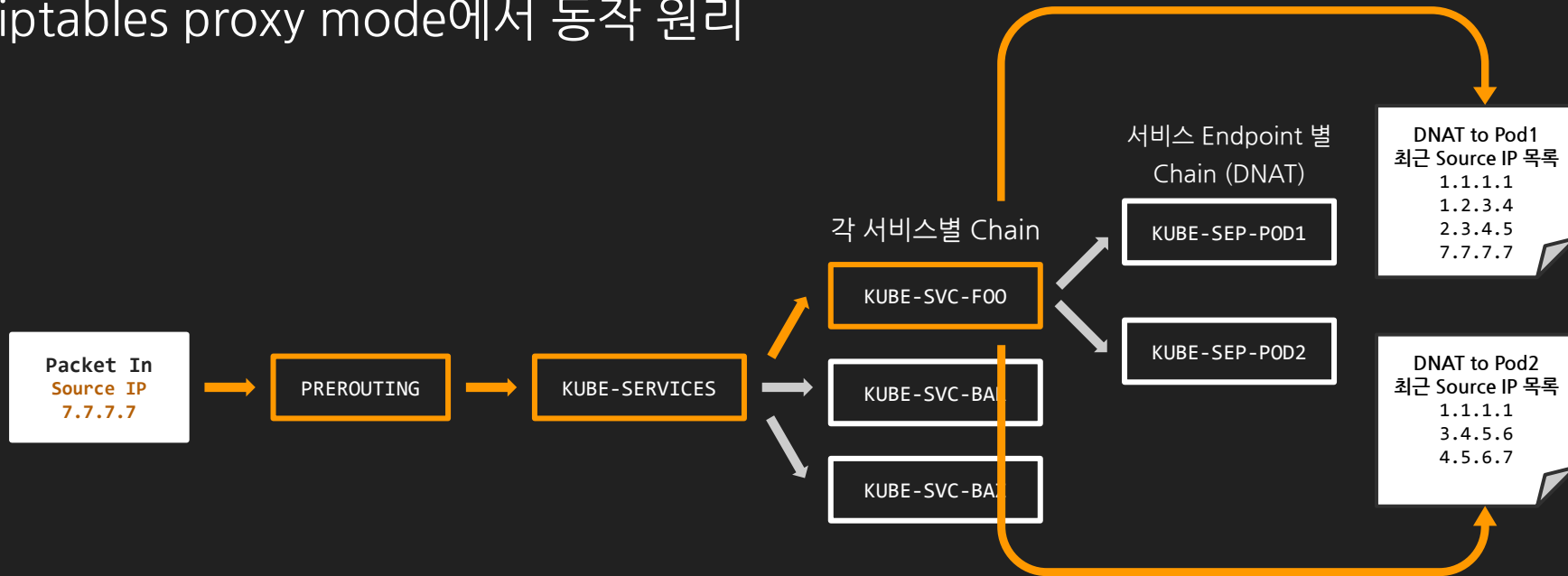
Session Affinity - Client IP

iptables proxy mode에서 동작 원리



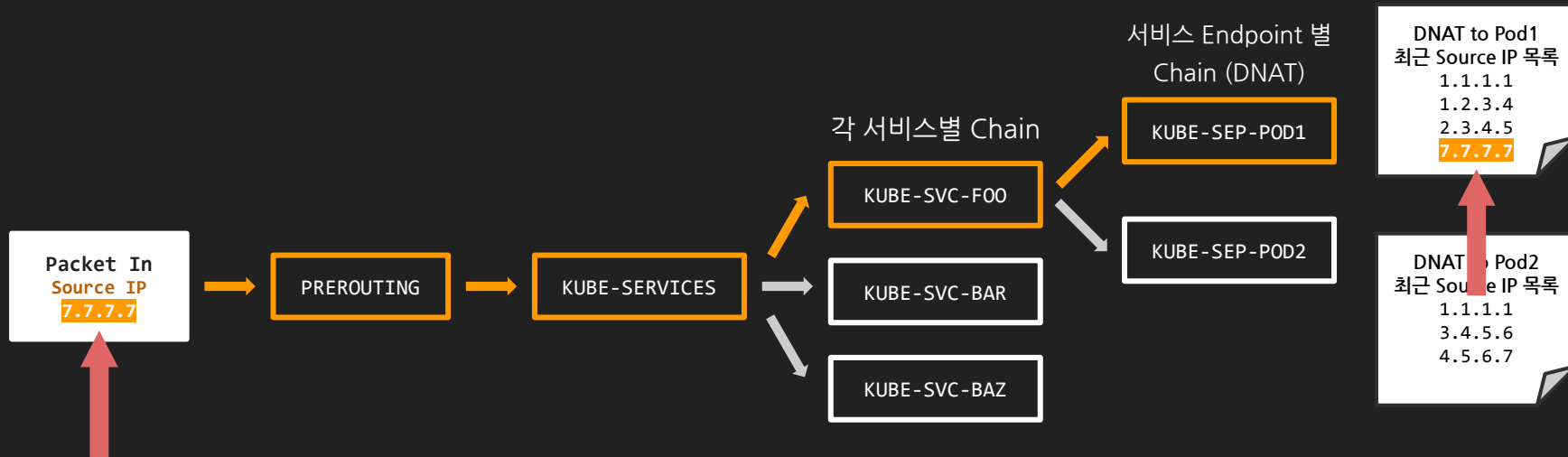
Session Affinity - Client IP

iptables proxy mode에서 동작 원리



Session Affinity - Client IP

iptables proxy mode에서 동작 원리

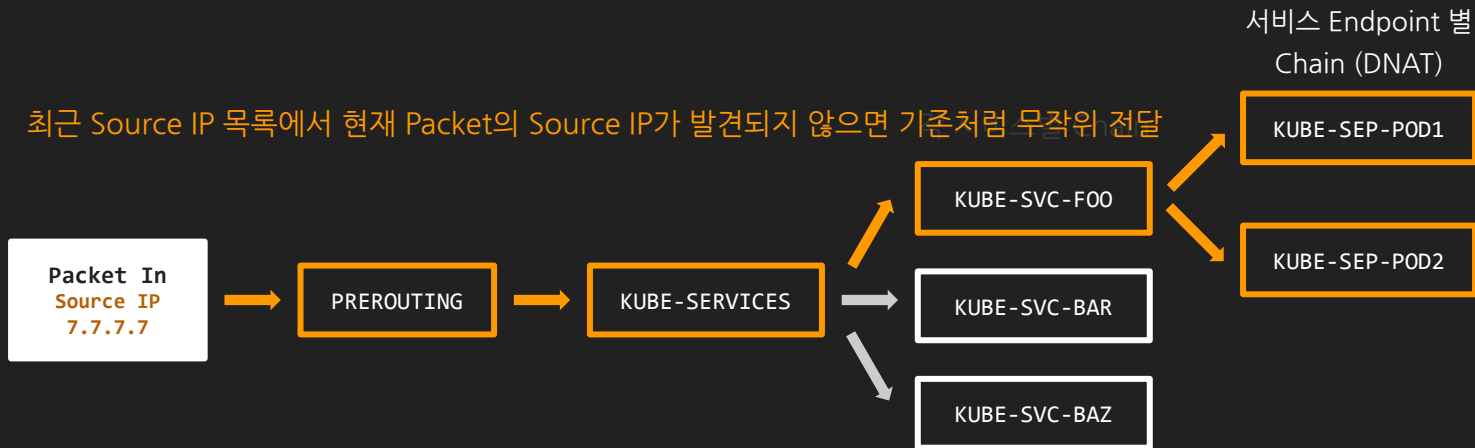


최근 Source IP 목록에서 현재 Packet의 Source IP를 발견하면 해당 Chain으로 패킷 전달

Session Affinity - Client IP

iptables proxy mode에서 동작 원리

최근 Source IP 목록에서 현재 Packet의 Source IP가 발견되지 않으면 기존처럼 무작위 전달



~~DNAT to Pod1
최근 Source IP 목록
1.1.1.1
1.1.1.4
1.3.4.5~~

~~DNAT to Pod2
최근 Source IP 목록
1.1.1.1
3.3.3.6
1.5.6.7~~

Session Affinity - Client IP

iptables proxy mode에서 동작 원리

iptables로 확인하기

```
Chain KUBE-SVC-QDYXZQLFVWQ62CGK (1 references)
target     prot opt source                destination
KUBE-SEP-7Z645BKY753FL3V3 all  --  anywhere              anywhere             /* default/hello-sessoin-affinity-service */ recent: CHECK seconds: 10800 reap name: KUBE-SEP-7Z645BKY753FL3V3 side: source
KUBE-SEP-LRH07F3KTMU43LSF all  --  anywhere              anywhere             /* default/hello-sessoin-affinity-service */ recent: CHECK seconds: 10800 reap name: KUBE-SEP-LRH07F3KTMU43LSF side: source
KUBE-SEP-7Z645BKY753FL3V3 all  --  anywhere              anywhere             /* default/hello-sessoin-affinity-service */ statistic mode random probability 0.50000000000
KUBE-SEP-LRH07F3KTMU43LSF all  --  anywhere              anywhere             /* default/hello-sessoin-affinity-service */
```

Pod 별 Chain으로 전달하기 전에, recent 모듈로부터 기록된 최근 Source IP를 확인 (최대 세션 고정시간 10800초 = 3시간) 목록
→ 현재 패킷의 Source IP가 최근 Source IP 목록에 존재하는 경우, 해당 Pod Chain으로 전달

최근 Source IP 목록에서 현재 Packet의 Source IP를 발견하면 해당 Chain으로 패킷 전달

Session Affinity - Client IP

iptables proxy mode에서 동작 원리

iptables로 확인하기

최근 Source IP 목록에서 최근 Packet의 Source IP가 발견되지 않으면, 기존처럼 무작위 전달

```
Chain KUBE-SVC-QDYXZQLFVWQ62CGK (1 references)
target    prot opt source                destination
KUBE-SEP-7Z645BKY753FL3V3 all -- anywhere             anywhere             /* default/hello-sessoin-affinity-service */ recent: CHECK seconds: 10800 reap name: KUBE-SEP-7Z645BKY753FL3V3 side: source
KUBE-SEP-LRH07F3KTMU43LSF all -- anywhere             anywhere             /* default/hello-sessoin-affinity-service */ recent: CHECK seconds: 10800 reap name: KUBE-SEP-LRH07F3KTMU43LSF side: source
KUBE-SEP-7Z645BKY753FL3V3 all -- anywhere             anywhere             /* default/hello-sessoin-affinity-service */ statistic mode random probability 0.500000000000
KUBE-SEP-LRH07F3KTMU43LSF all -- anywhere             anywhere             /* default/hello-sessoin-affinity-service */
```

최근 Source IP에 존재하지 않으면, 기존처럼 무작위 전달

서비스 Endpoint 별
Chain (DNAT)

~~DNAT to Pod
최근 Source IP 목록~~

~~10.1.1.1
10.1.1.4
3.4.5.6~~

Packet In
Source IP
7.7.7.7

KUBE-SVC-BAR

KUBE-SVC-BAZ

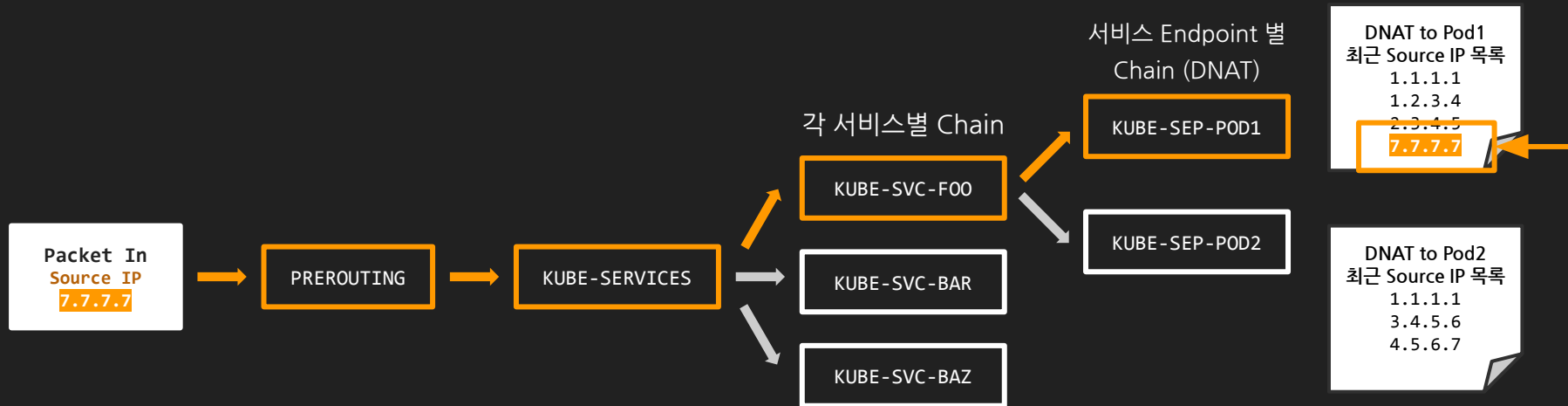
~~DNAT to Pod
최근 Source IP 목록~~

~~10.1.1.1
3.4.5.6
1.5.6.7~~

Session Affinity - Client IP

iptables proxy mode에서 동작 원리

DNAT를 적용한 Chain에서 최근 Source IP 목록에 저장



Session Affinity - Client IP

iptables proxy mode에서 동작 원리

iptables로 확인하기

Chain KUBE-SEP-LRH07F3KTMU43LSF (2 references)

target	prot	opt	source	destination
KUBE-MARK-MASQ	all	--	10.244.2.3	anywhere
DNAT	tcp	--	anywhere	anywhere

```
/* default/hello-sessoin-affinity-service */  
/* default/hello-sessoin-affinity-service */
```

recent: SET name: KUBE-SEP-LRH07F3KTMU43LSF side: source mask: 255.255.255.255 tcp to:10.244.2.3:8080

Packet In
Source IP
1.1.1.1

PREROUTING

KUBE-SERVICES

KUBE-SVC-BAR

KUBE-SVC-BAZ

KUBE-SEP-POD2

서비스 Endpoint 별
Chain (DNAT)

DNAT to Pod1
최근 Source IP 목록
1.1.1.1
1.2.3.4

DNAT to Pod2
최근 Source IP 목록
1.1.1.1
3.4.5.6
4.5.6.7

Pod Chain에서 DNAT 적용된 패킷의 Source IP(Client IP)를 저장

감사합니다.