Life of a packet in Amazon EKS

Dumlu Timuralp

Solutions Architect at Amazon Web Services

https://www.linkedin.com/in/dumlutimuralp/

https://dumlutimuralp.medium.com/

https://github.com/dumlutimuralp

Disclaimer

- All data provided about packet walks are observed using Linux commands (“ip”, ”tcpdump”), SSH access to Amazon EKS worker nodes, and VPC flow logs
- All of the content in this session is based on public information and resources are explicitly highlighted on the respective slides.
- The opinions expressed in this session are solely my own and do not represent those of my employer.

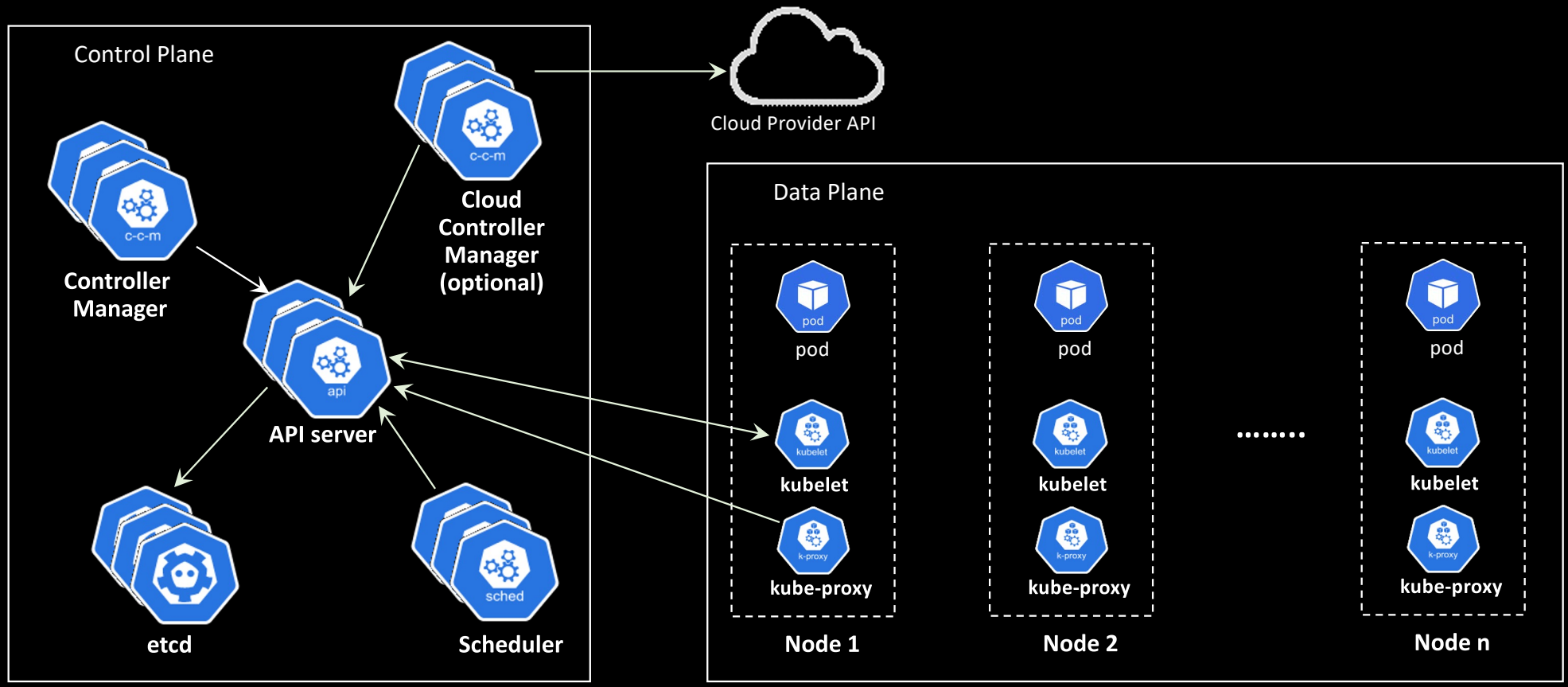
Agenda

- Kubernetes and Amazon EKS high level architecture
- Kubernetes network model and pod connectivity
- Packet walks
- Kubernetes Services and Ingress
- more packet walks

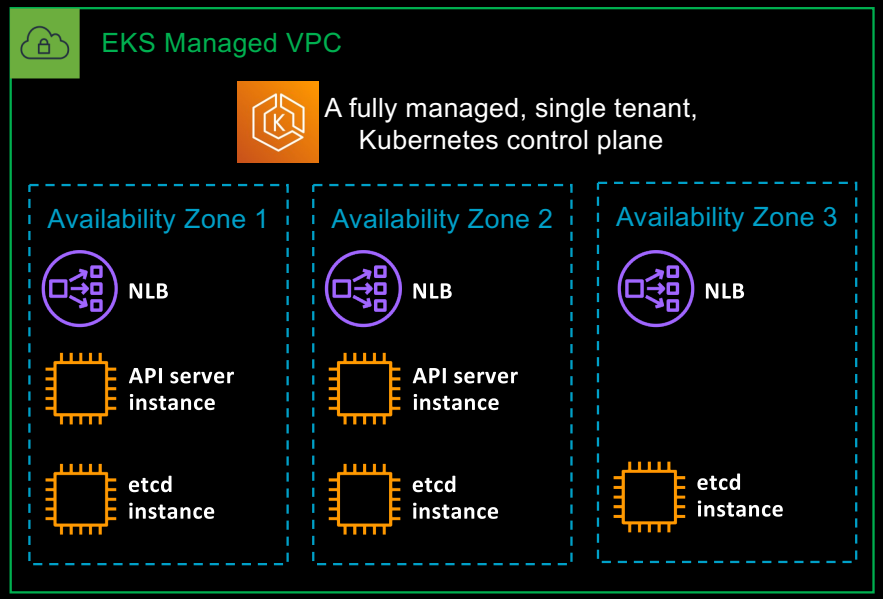
Out of scope

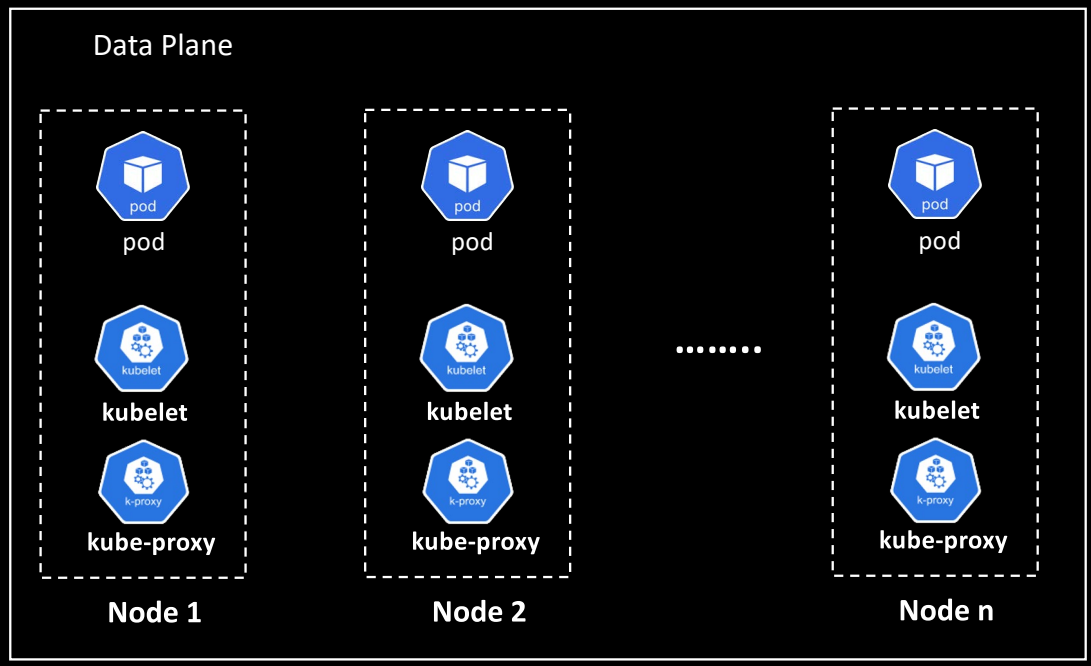
- IPv6
- Kubernetes DNS specifics and details
- EKS cluster communication details (EKS node <-> EKS Control Plane)
- Developer <-> EKS Control Plane (Kubernetes API) communication details
- EKS on Fargate , EKS Anywhere, EKS on AWS Outposts, EKS Hybrid Nodes, Windows Nodes

Kubernetes Arhitecture



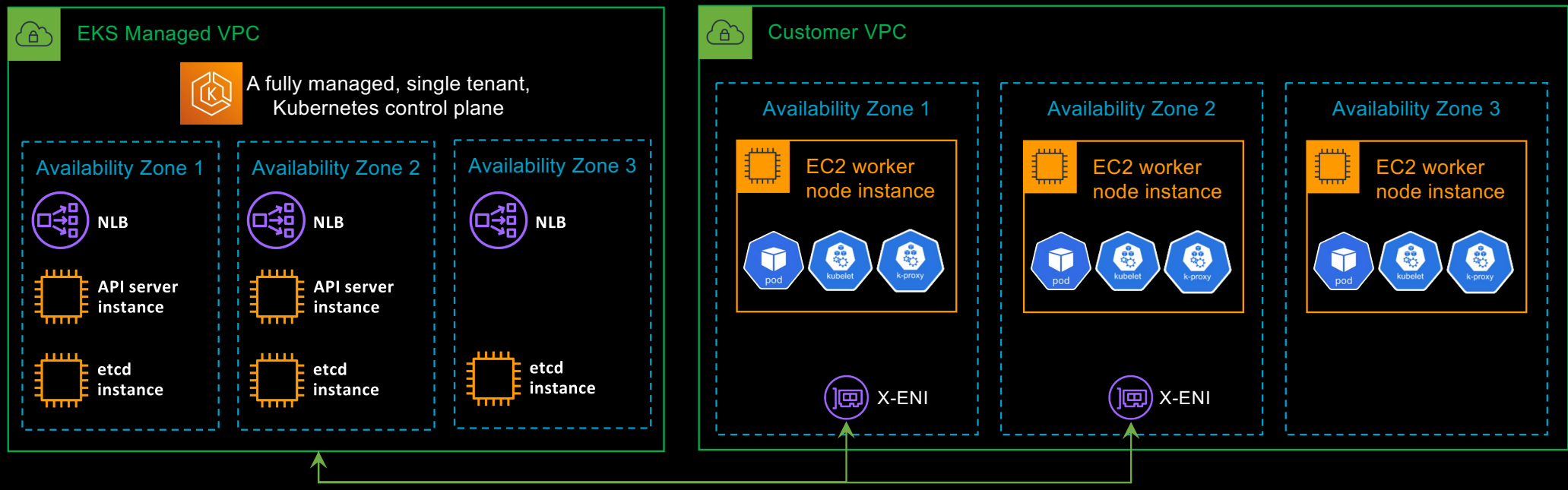
EKS Architecture





Source :
https://docs.aws.amazon.com/eks/latest/best-practices/control-plane.html

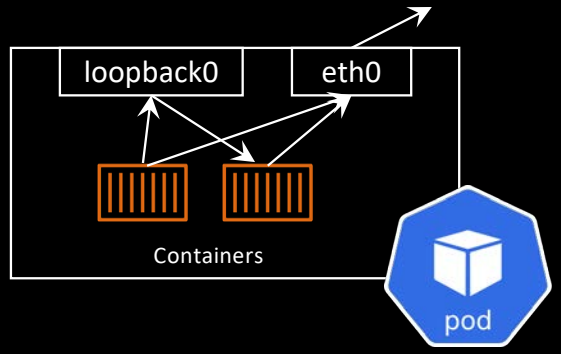
EKS Architecture



Source :
https://docs.aws.amazon.com/eks/latest/best-practices/control-plane.html

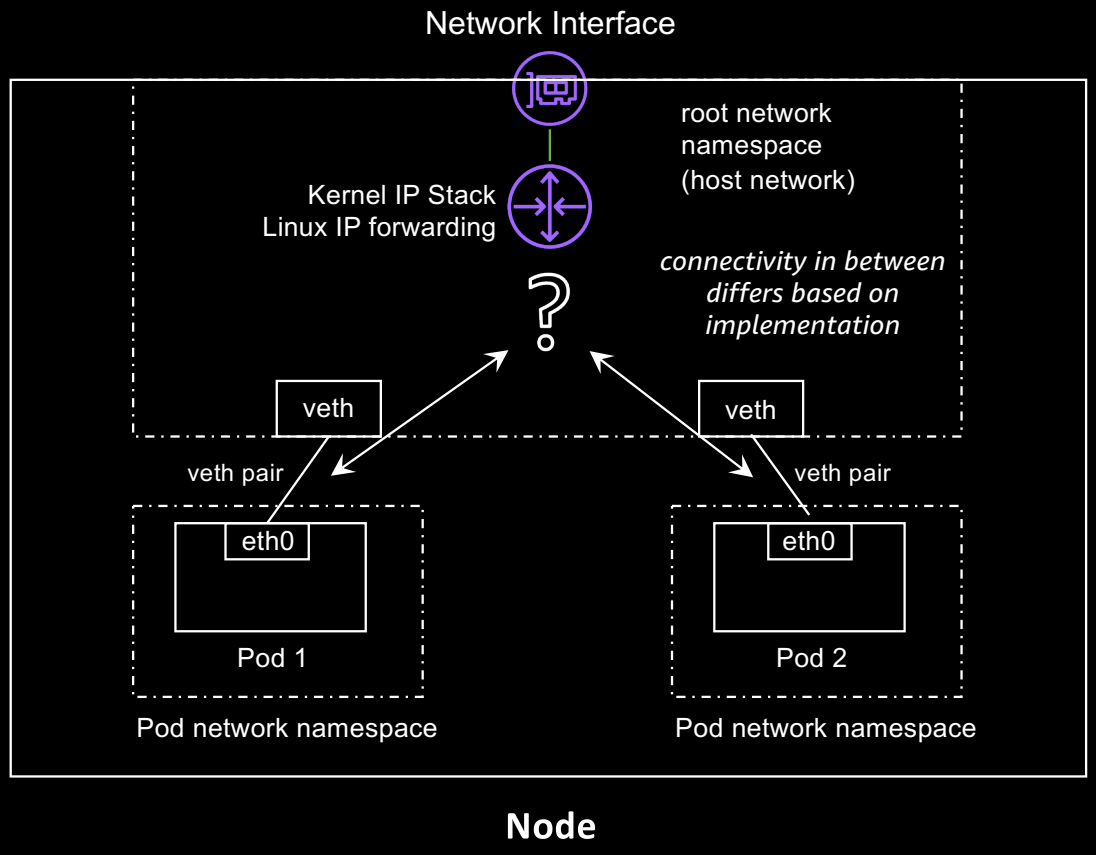
Kubernetes Network Model

- Every pod gets its own unique cluster-wide IP address
- Pods can communicate with all other pods on any other node without NAT
- Agents on a node (e.g. system daemons, kubelet) can communicate with all pods on that node



Source :
https://kubernetes.io/docs/concepts/services-networking/
https://kubernetes.io/docs/concepts/cluster-administration/networking/

Pod Connectivity

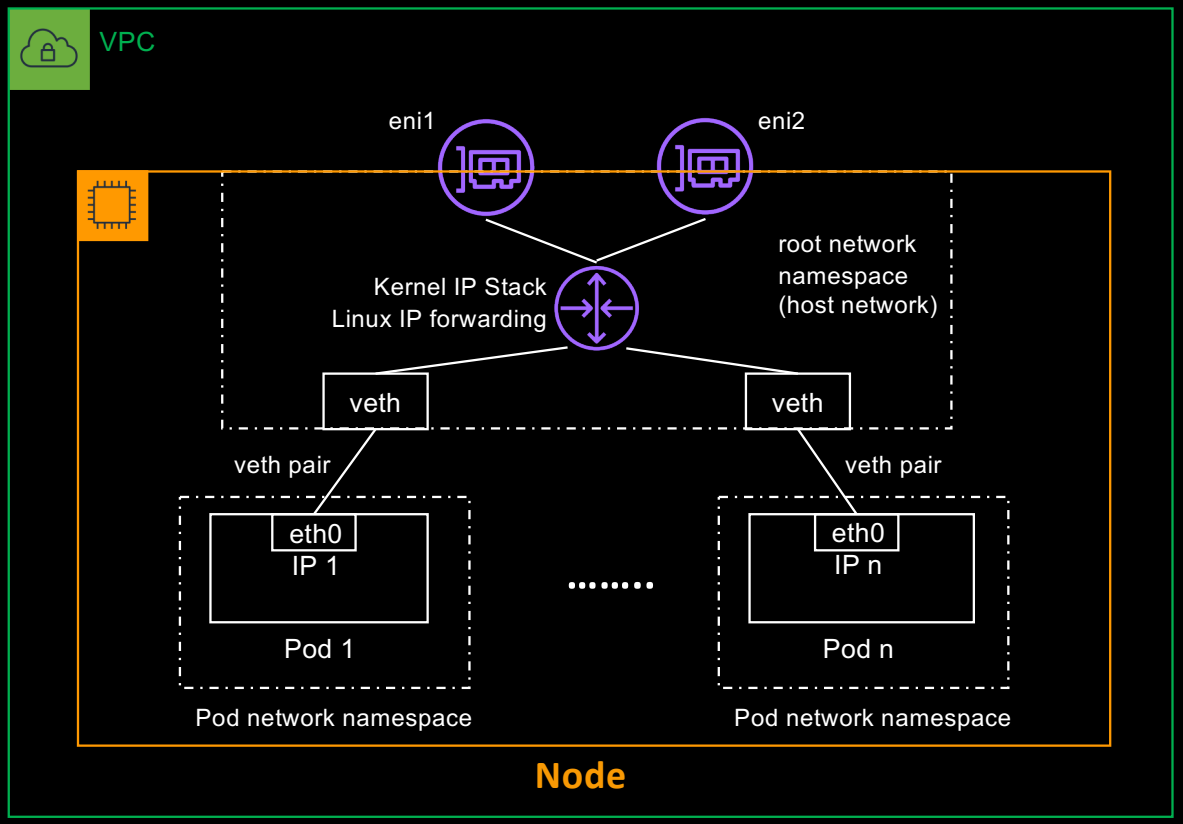


At a minimum, Container Network Interface (CNI) plugin is responsible for :

- adding/deleting the network interface in the pod network namespace
- calling the IPAM plugin to pull an available IP and configure it on the network interface of the pod (also reverse operation)
- adding/deleting the connectivity to the host network namespace (veth pair)

Source :
https://manpages.ubuntu.com/manpages/focal/man4/veth.4.html
https://cni.dev/docs/spec/#cni-operations

Pod Connectivity with VPC CNI

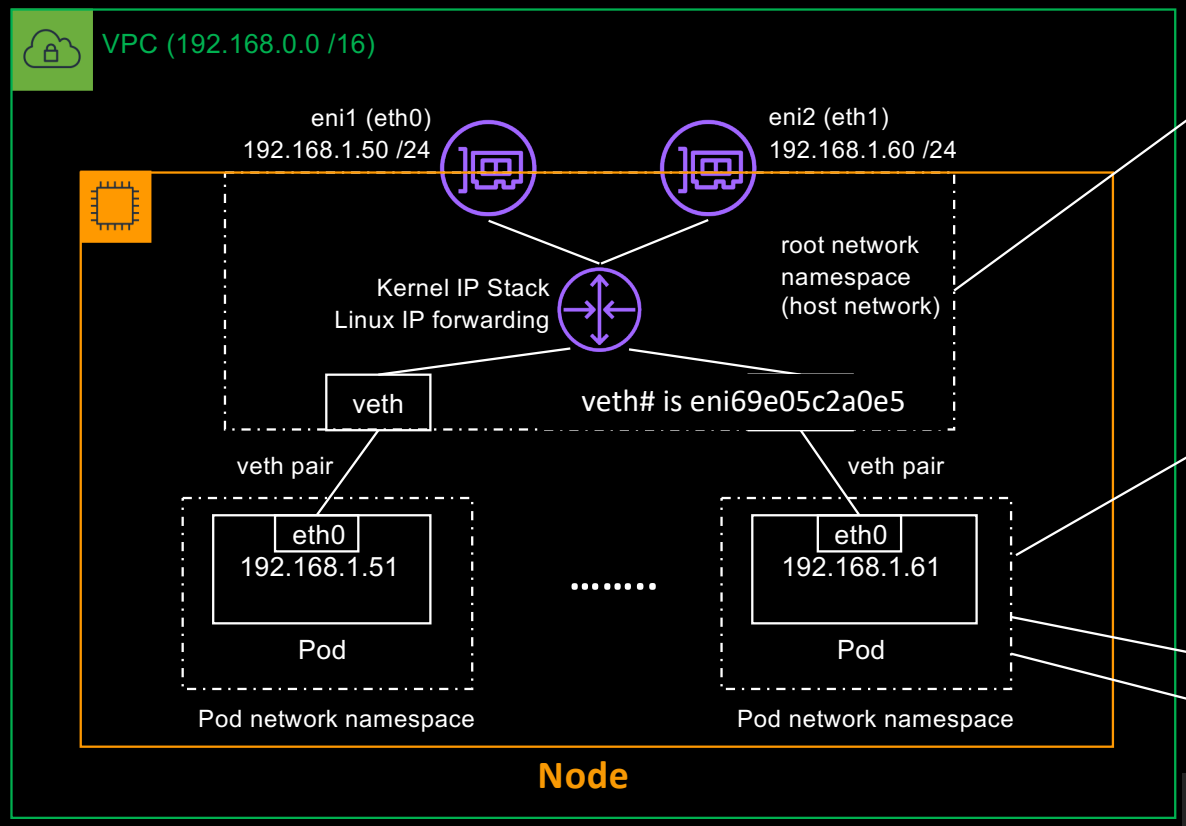


VPC CNI plugin runs the following tasks :

- sets up/decommissions pod connectivity
- leverages 1/EC2 Secondary IP mode or 2/EC2 Prefix mode feaure to assign an IP address to the pod from the VPC CIDR (pod works on/mapped to an ENI)
- manages the ENIs of the node, to make sure of available IPs for pods
- configures respective routing tables, routing entries and policy based routing rules in the node root network namespace
- configures routing/ARP entries in the pod network namespace

Source :
https://github.com/aws/amazon-vpc-cni-k8s/blob/master/docs/cni-proposal.md#solution-components
https://docs.aws.amazon.com/eks/latest/best-practices/vpc-cni.html

Examining interface with VPC CNI



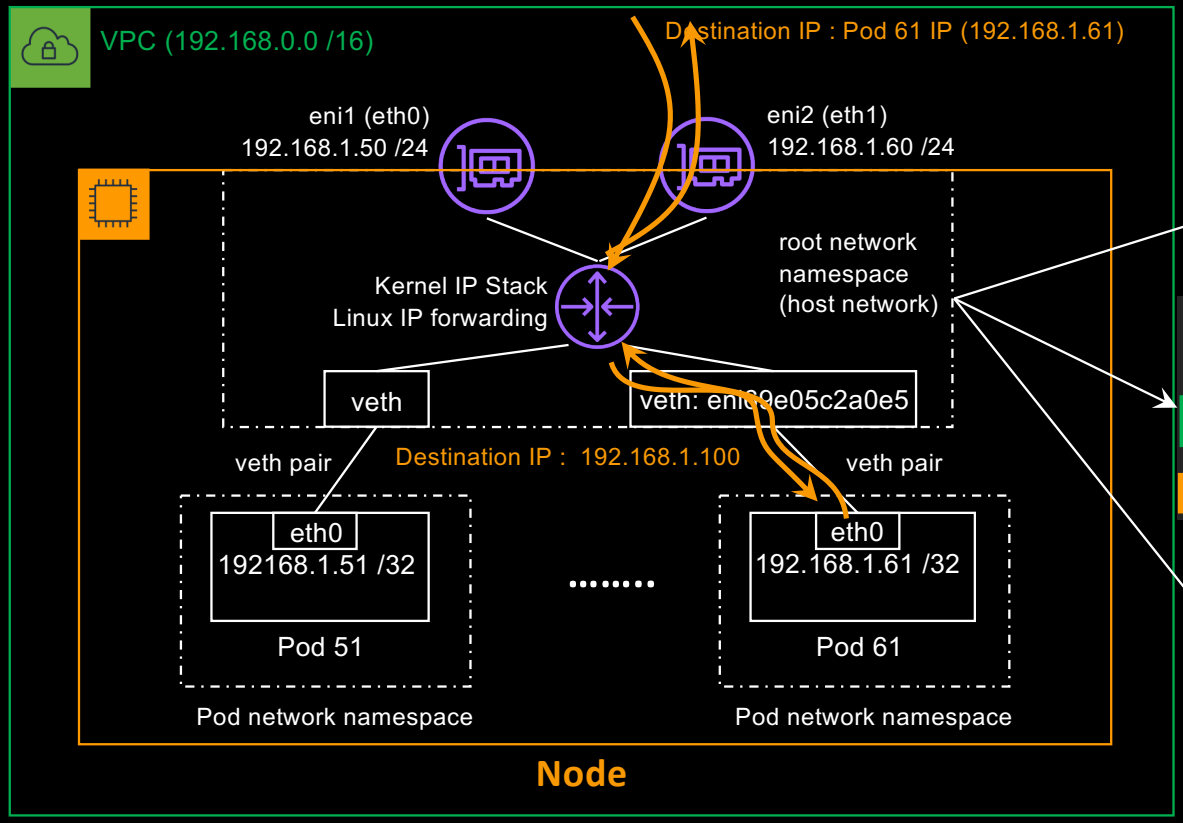

```
$ ip a | grep eni69e05c2a0e5 -A 3
39: eni69e05c2a0e5@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001
qdisc noqueue state UP group default
    link/ether 06:5e:c8:ae:85:be brd ff:ff:ff:ff:ff:ff link-netns cni-
04b6218d-0a3c-2466-aea7-ee1f8d4cb935
    inet6 fe80::45e:c8ff:feae:85be/64 scope link
       valid_lft forever preferred_lft forever
```

```
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
group default qlen 1000
    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
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: eth0@if39: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc noqueue
state UP group default
    link/ether 6a:40:16:85:c3:34 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.1.61/32 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::6840:16ff:fe85:c334/64 scope link
       valid_lft forever preferred_lft forever
```

```
/ # ip route
default via 169.254.1.1 dev eth0
169.254.1.1 dev eth0 scope link
```

```
/ # arp -n
Address         HWtype  HWaddress           Flags Mask  Iface
169.254.1.1     ether   06:5e:c8:ae:85:be   CM          eth0
```

Ingress/egress on a node with VPC CNI



eni specific route table (to route Pod's outgoing traffic) (since each pod is using an IP from the respective eni)

```
$ ip route show table 2
default via 192.168.1.1 dev eth1
192.168.1.1 dev eth1 scope link
```

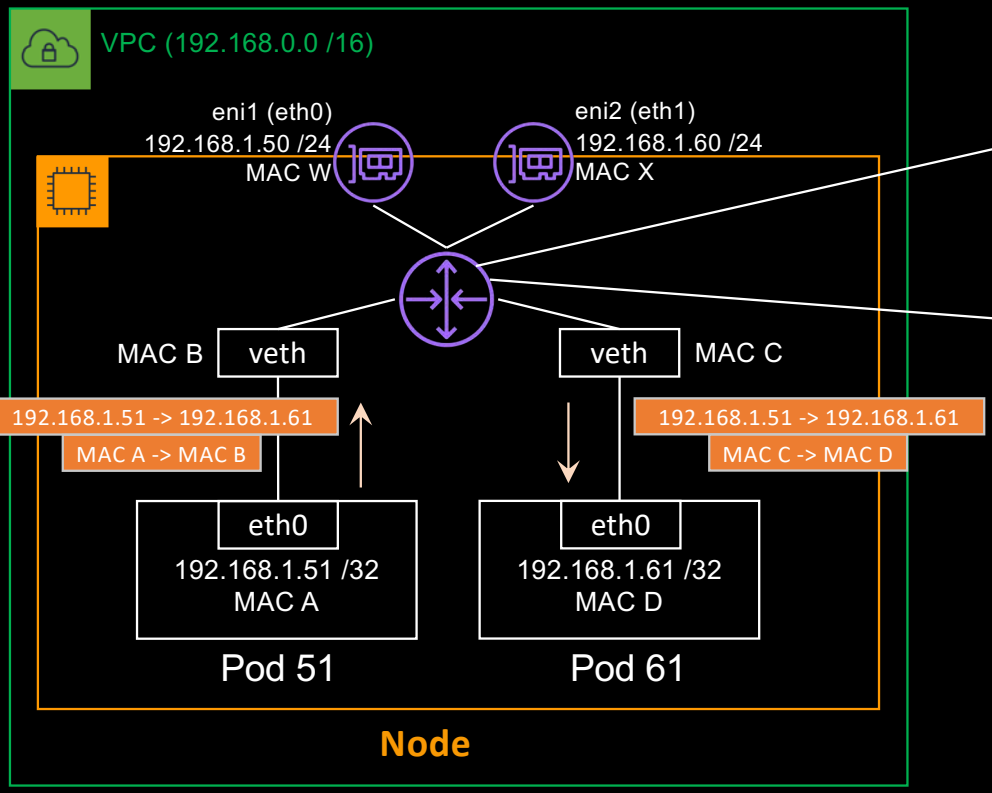
main route table (to route traffic destined to a Pod)

```
$ ip route show table main
default via 192.168.1.1 dev eth0
169.254.169.254 dev eth0
192.168.1.0/24 dev eth0 proto kernel scope link src 192.168.1.50
192.168.1.51 dev enib9b00b57dd4 scope link
192.168.1.61 dev eni69e05c2a0e5 scope link
```

policy based routing rules

```
$ ip rule show
0:      from all lookup local
512:    from all to 192.168.1.51 lookup main
512:    from all to 192.168.1.61 lookup main
1024:   from all fwmark 0x80/0x80 lookup main
1536:   from 192.168.1.61 lookup 2
32766:  from all lookup main
32767:  from all lookup default
```

Life of a packet – pod to pod on the same node



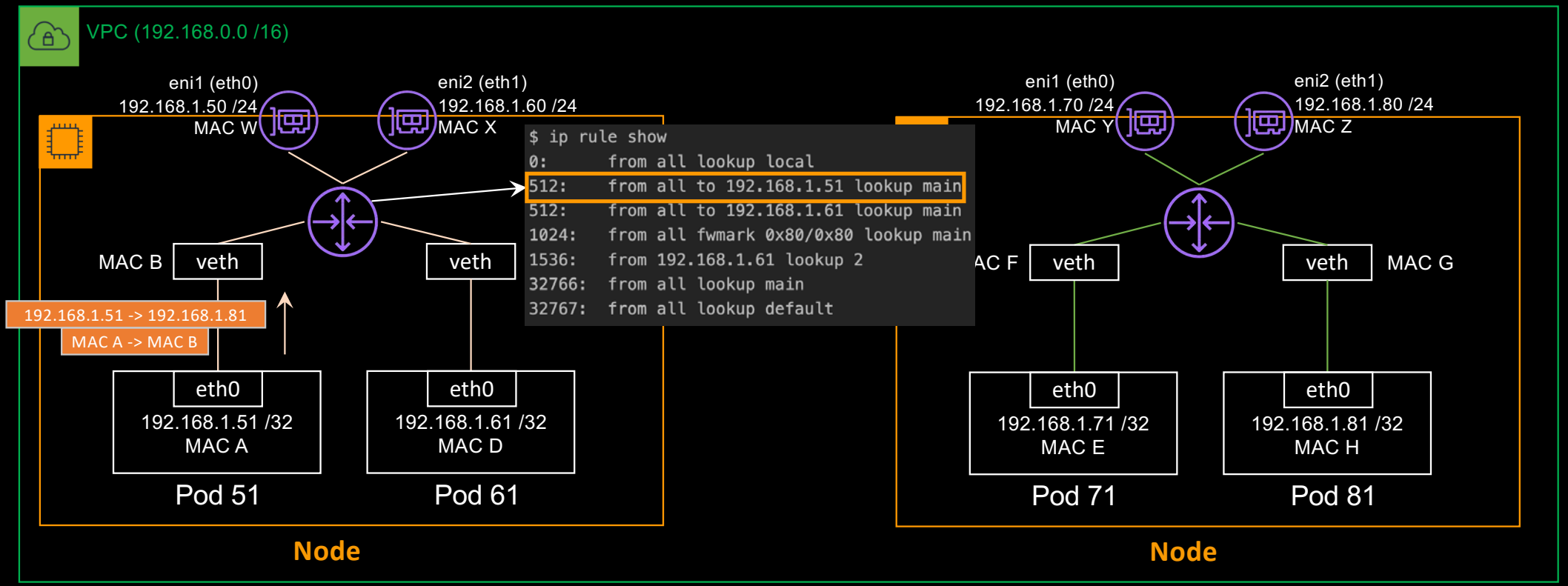
policy based routing rules

```
$ ip rule show
0:      from all lookup local
512:    from all to 192.168.1.51 lookup main
512:    from all to 192.168.1.61 lookup main
1024:   from all fwmark 0x80/0x80 lookup main
1536:   from 192.168.1.61 lookup 2
32766:  from all lookup main
32767:  from all lookup default
```

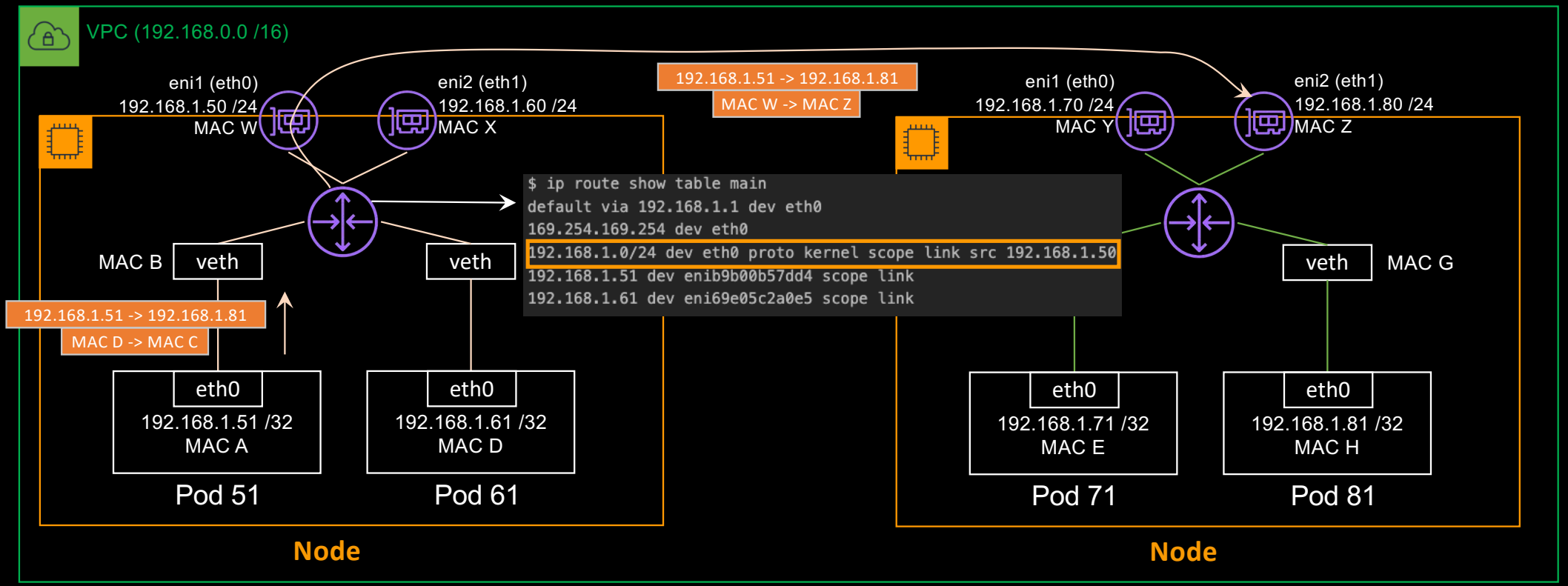
main route table

```
$ ip route show table main
default via 192.168.1.1 dev eth0
169.254.169.254 dev eth0
192.168.1.0/24 dev eth0 proto kernel scope link src 192.168.1.50
192.168.1.51 dev enib9b00b57dd4 scope link
192.168.1.61 dev eni69e05c2a0e5 scope link
```

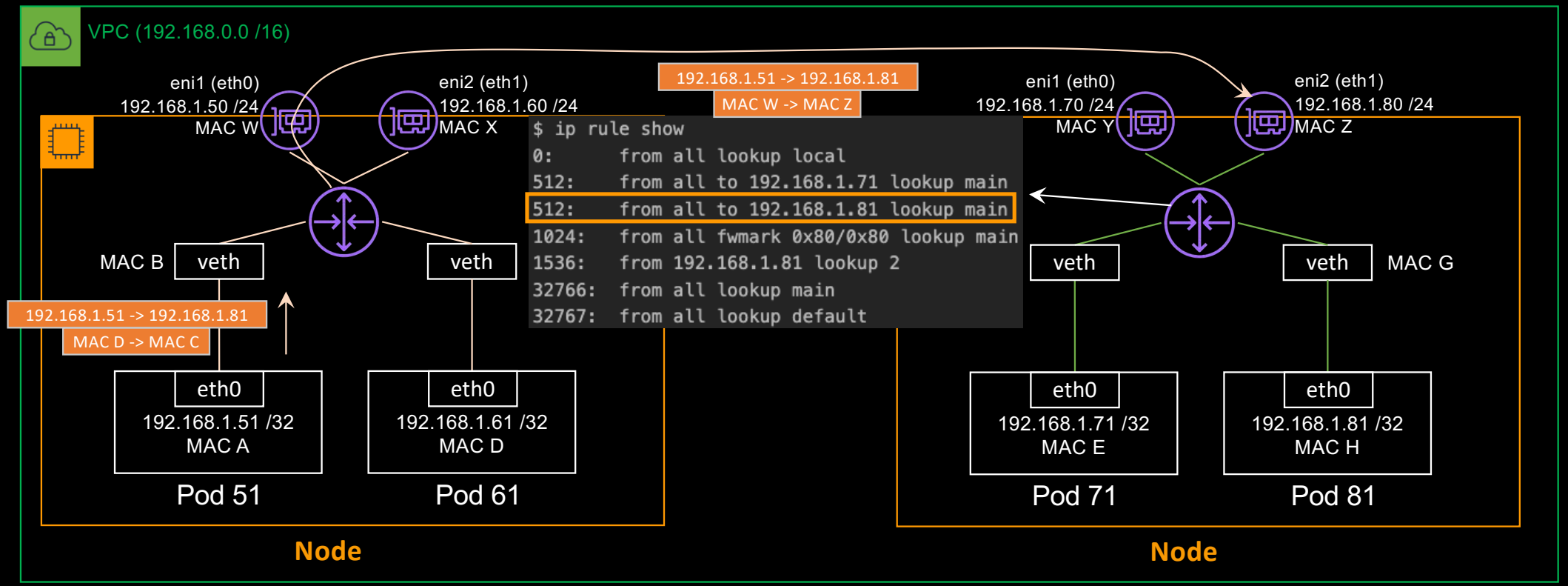
Life of a packet – pod to pod across nodes



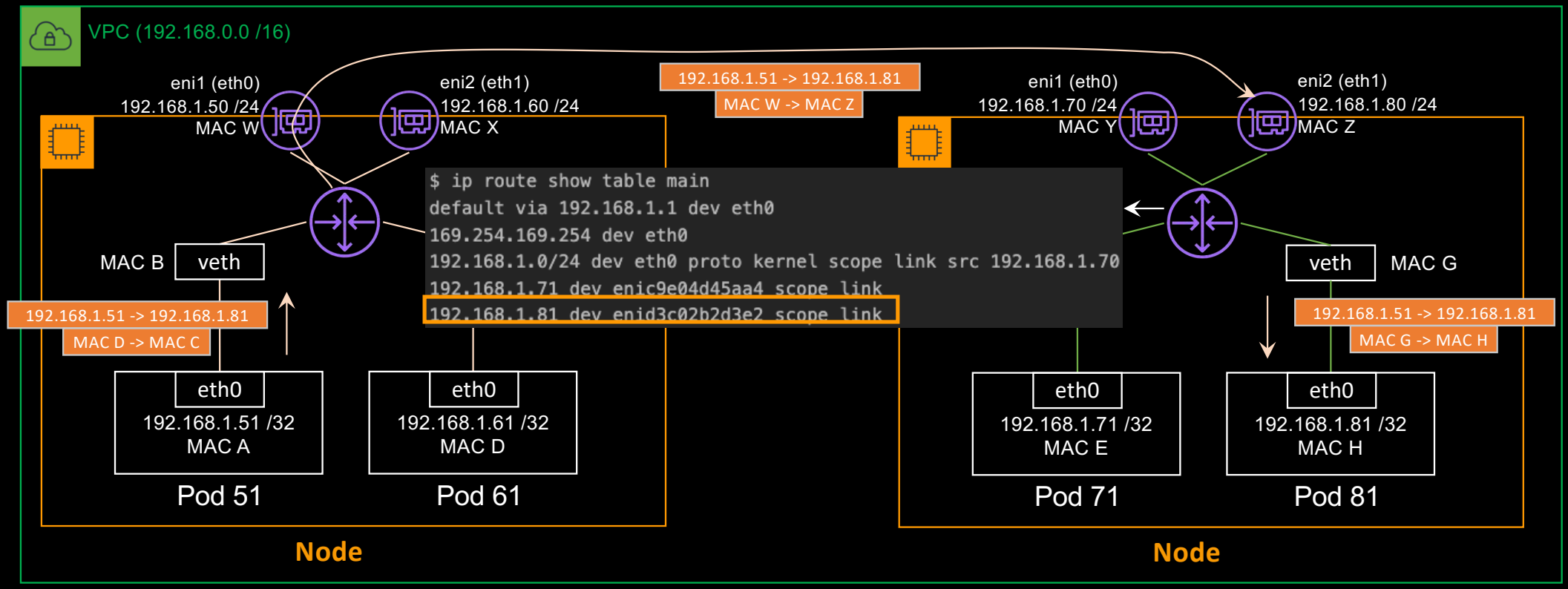
Life of a packet – pod to pod across nodes



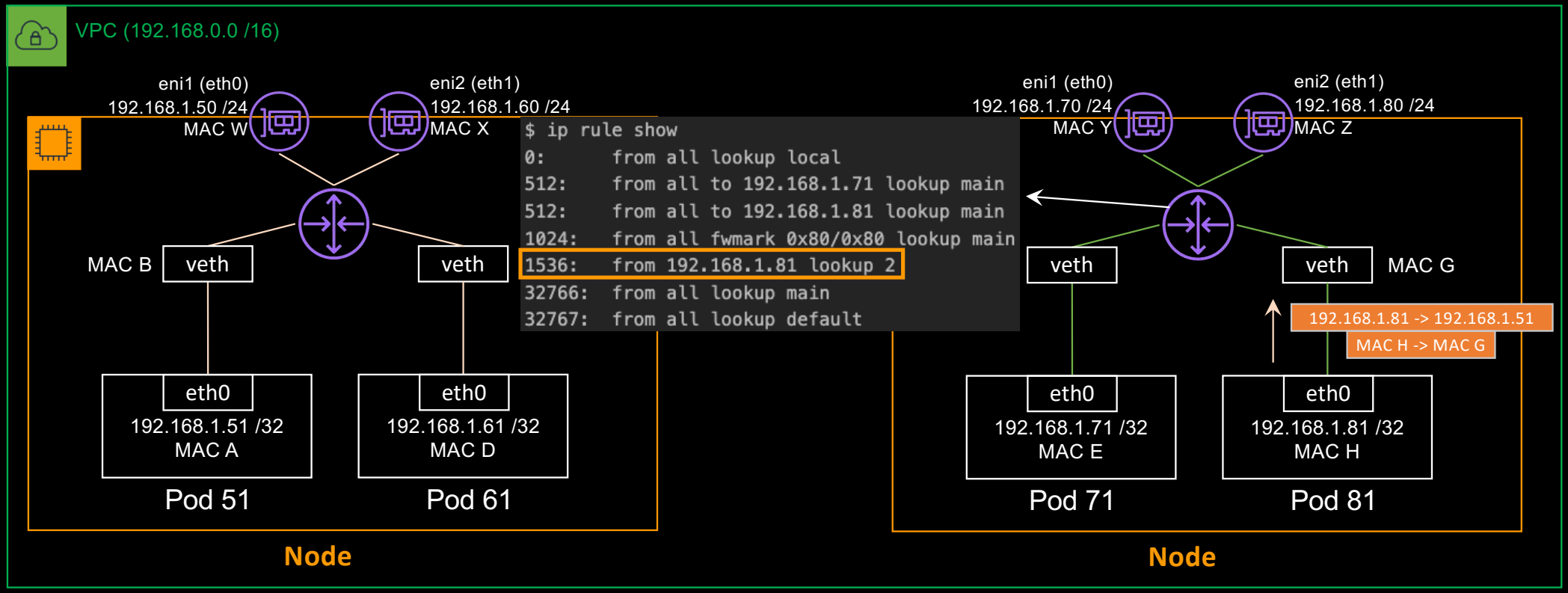
Life of a packet – pod to pod across nodes



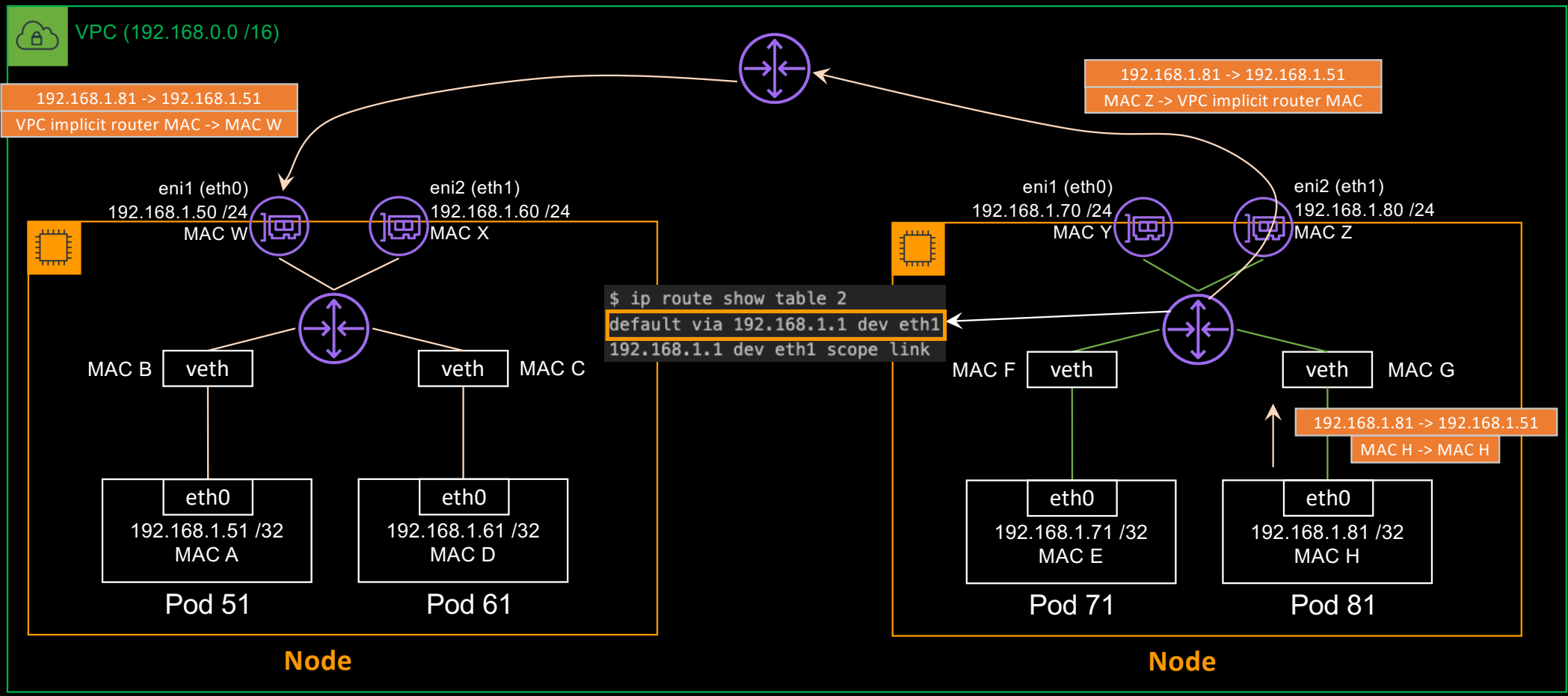
Life of a packet – pod to pod across nodes



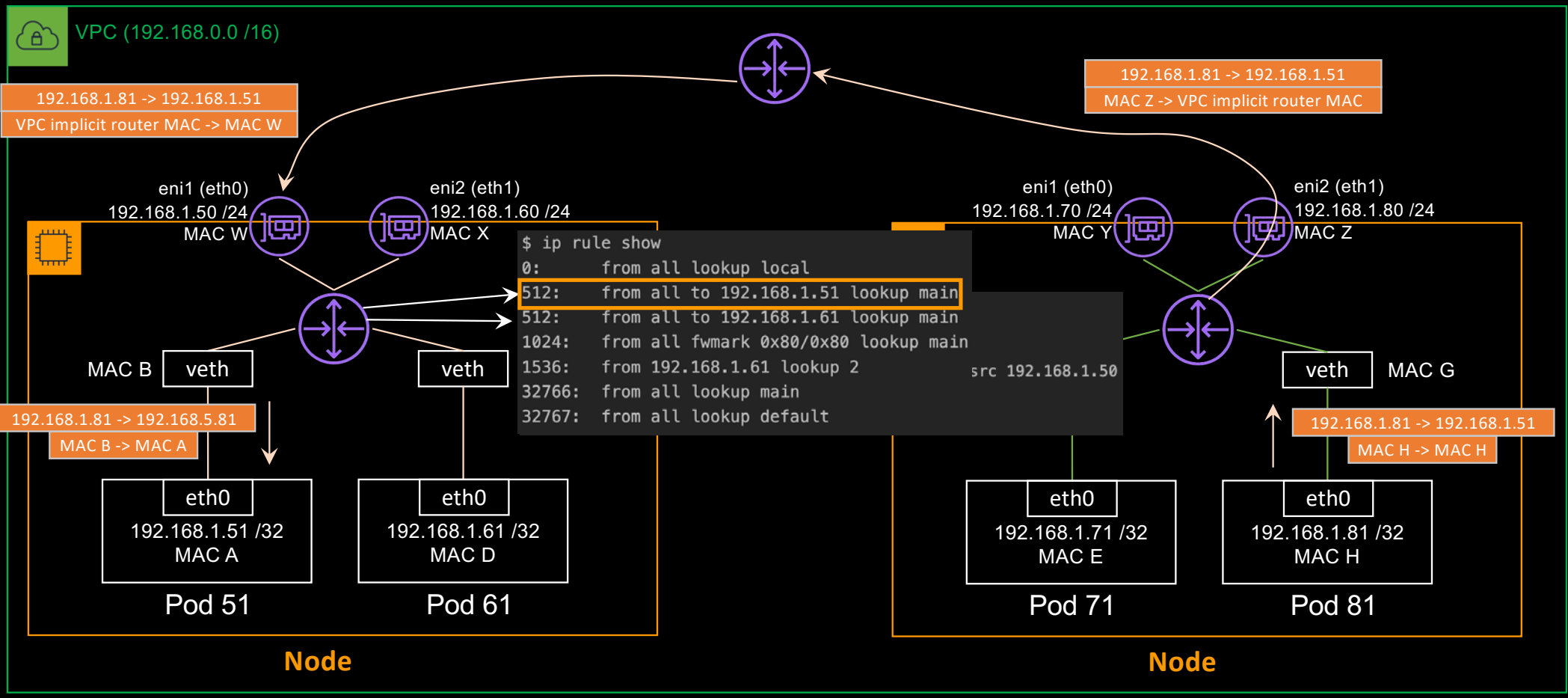
Life of a packet – pod to pod across nodes (return)



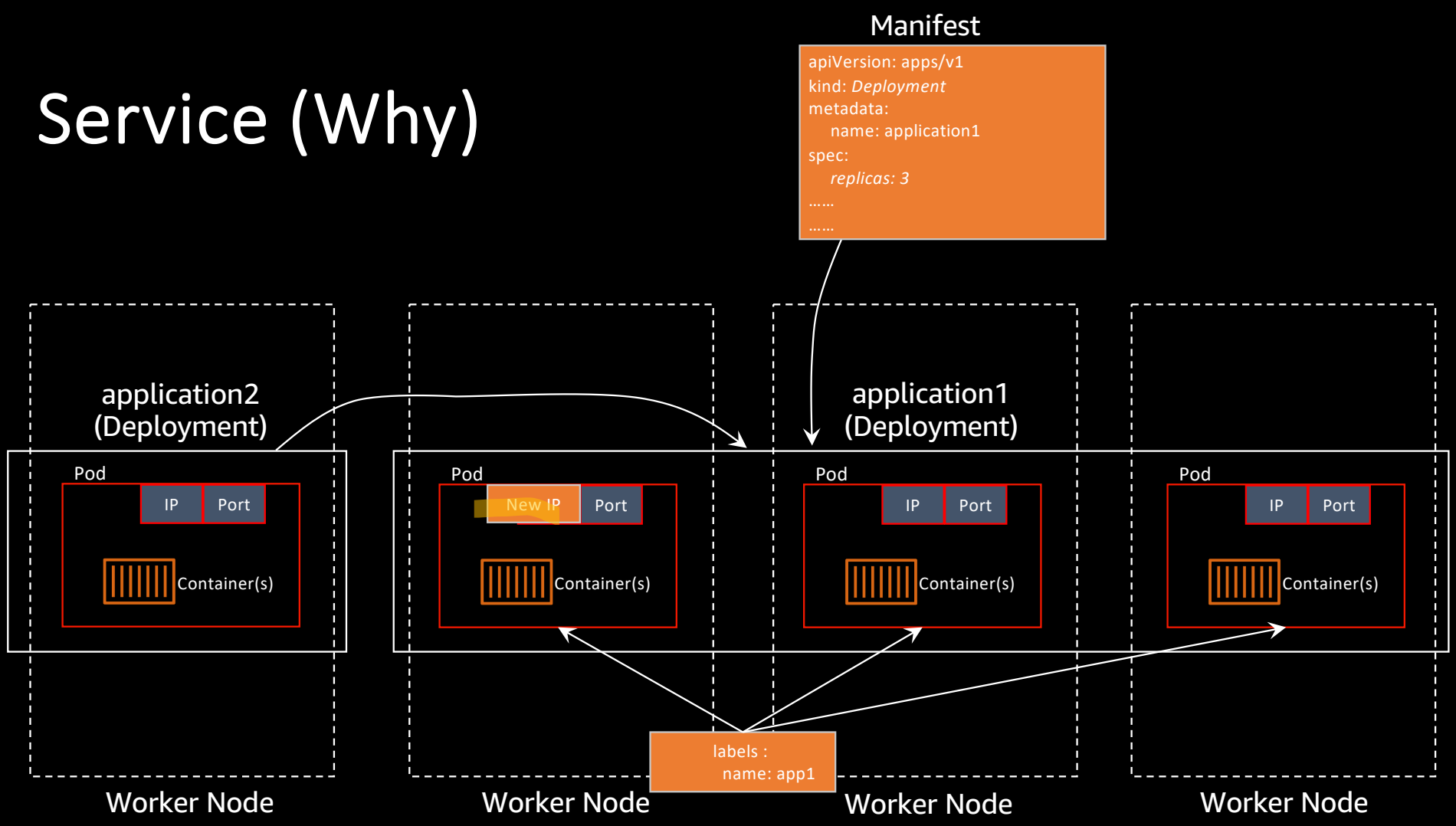
Life of a packet – pod to pod across nodes (return)



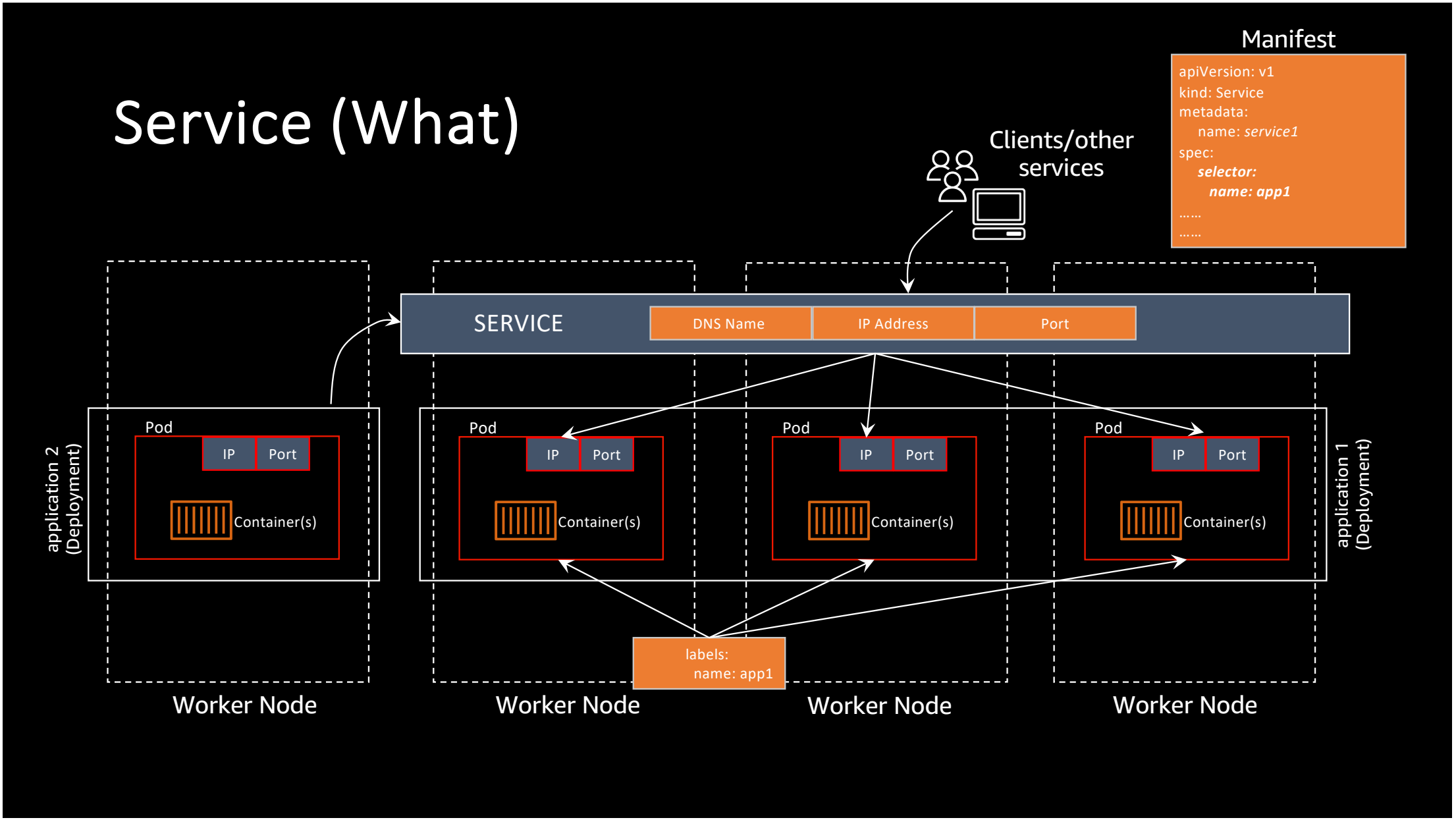
Life of a packet – pod to pod across nodes (return)

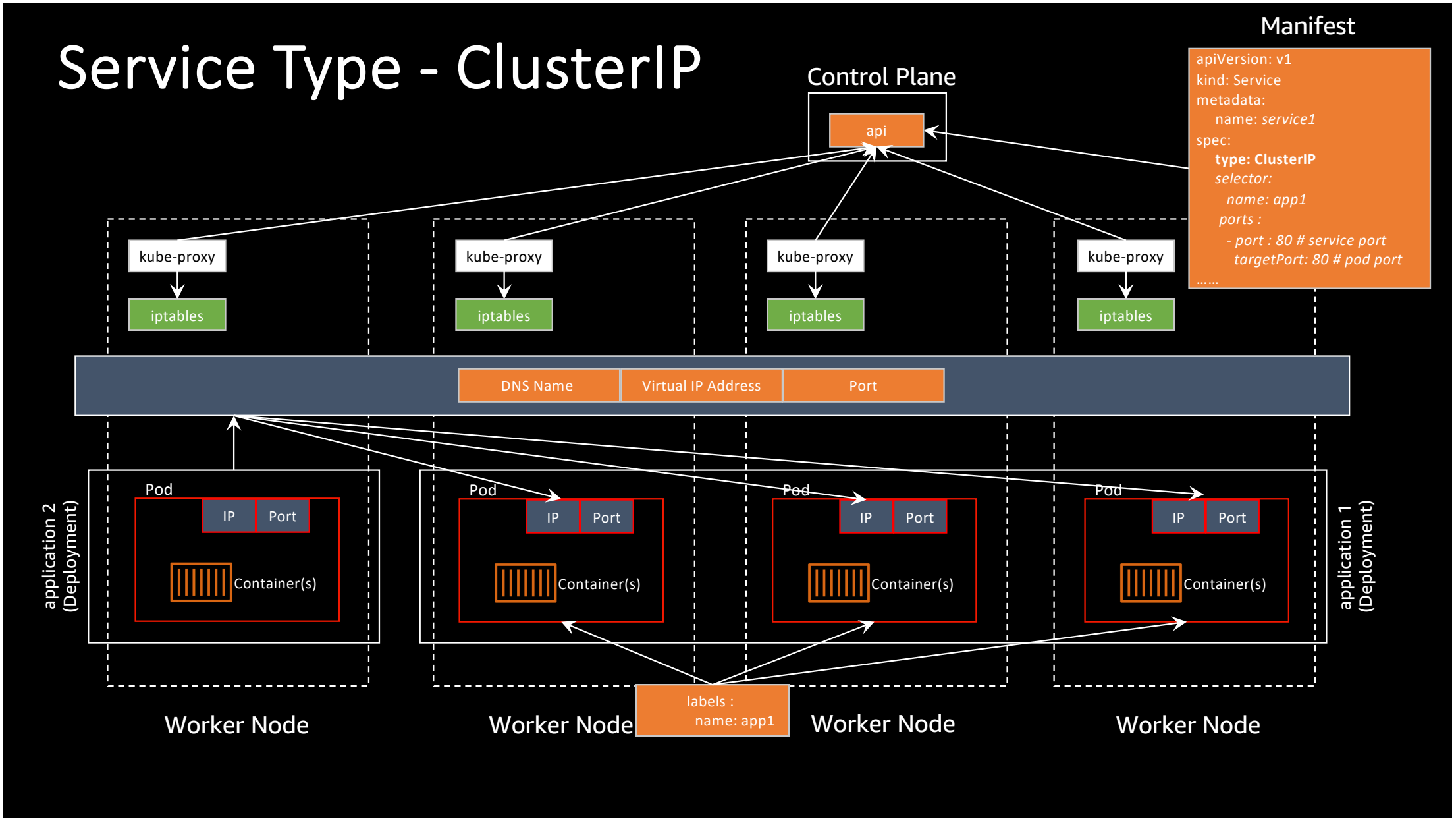


Service (Why)

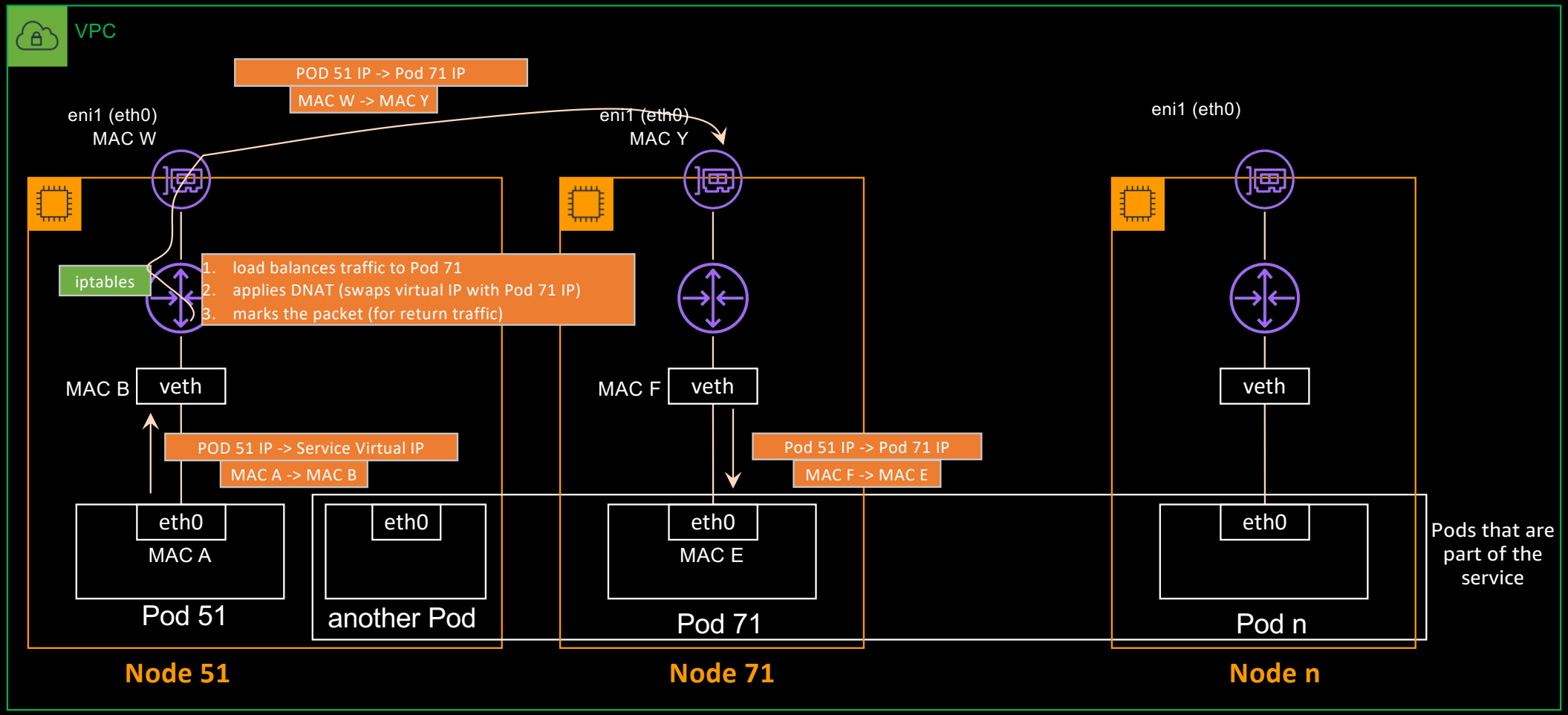

Manifest
apiVersion: apps/v1
kind: Deployment
metadata:
name: application1
spec:
replicas: 3
……
……
application2
(Deployment)
application1
(Deployment)
Pod
IP
Port
Container(s)
Pod
New IP
Port
Container(s)
Pod
IP
Port
Container(s)
Pod
IP
Port
Container(s)
labels :
name: app1
Worker Node
Worker Node
Worker Node
Worker Node

Service (What)

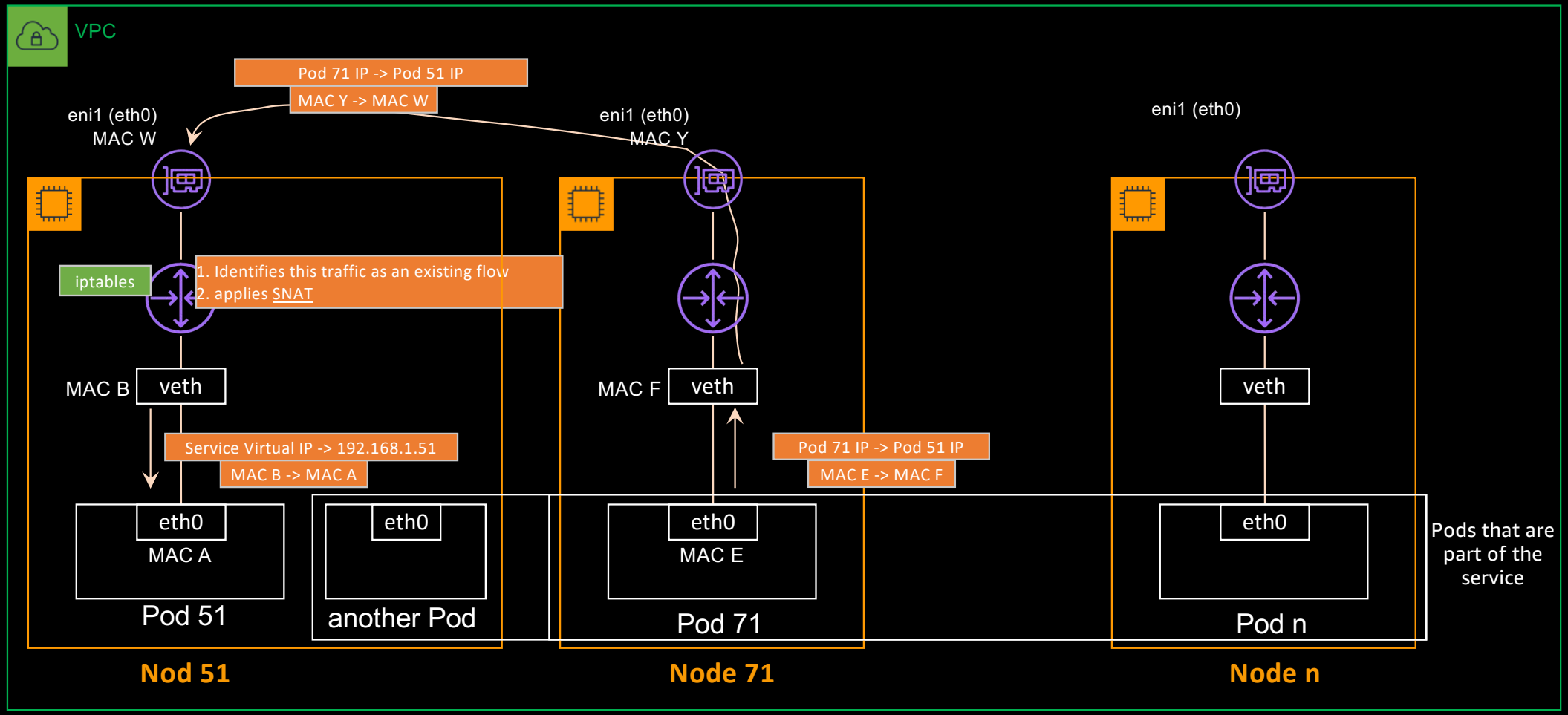



Service Type - ClusterIP
Control Plane
api
Manifest
apiVersion: v1
kind: Service
metadata:
name: service1
spec:
type: ClusterIP
selector:
name: app1
ports :
- port : 80 # service port
targetPort: 80 # pod port
……
kube-proxy
kube-proxy
kube-proxy
kube-proxy
iptables
iptables
iptables
iptables
DNS Name
Virtual IP Address
Port
application 2 (Deployment)
application 1 (Deployment)
Pod
IP
Port
Container(s)
labels :
name: app1
Worker Node
Worker Node
Worker Node
Worker Node

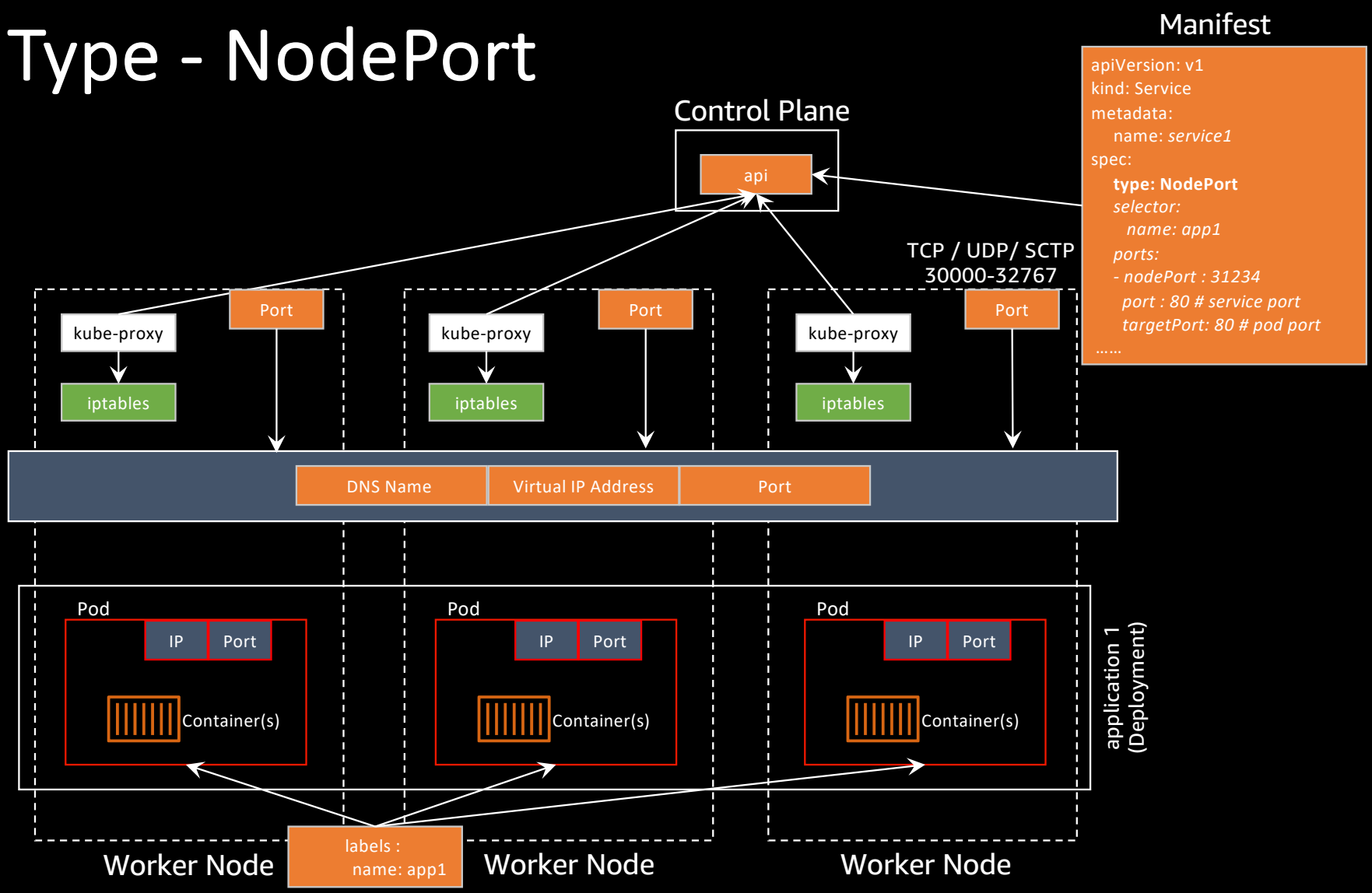
Life of a packet - ClusterIP



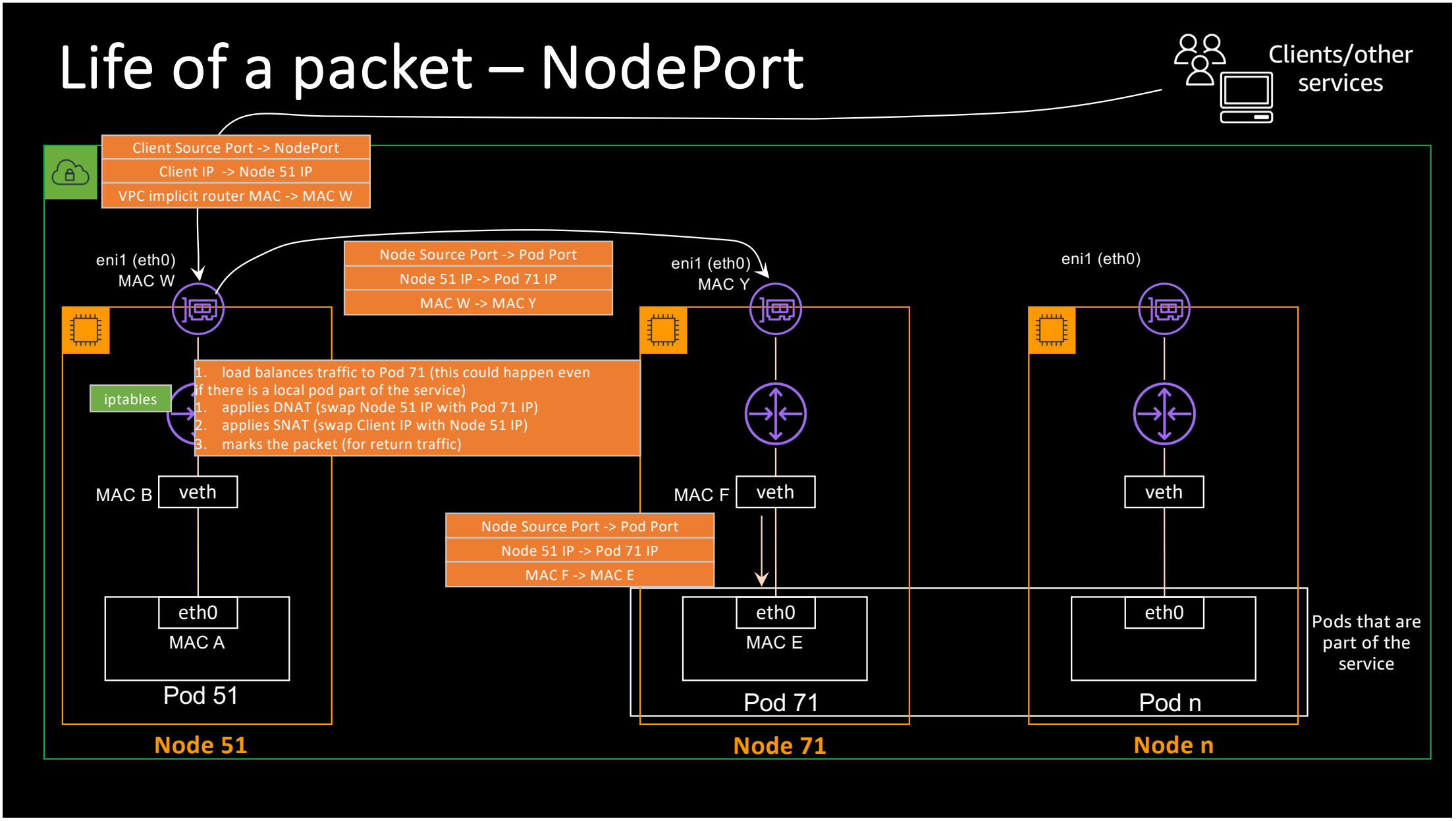
Life of a packet - ClusterIP



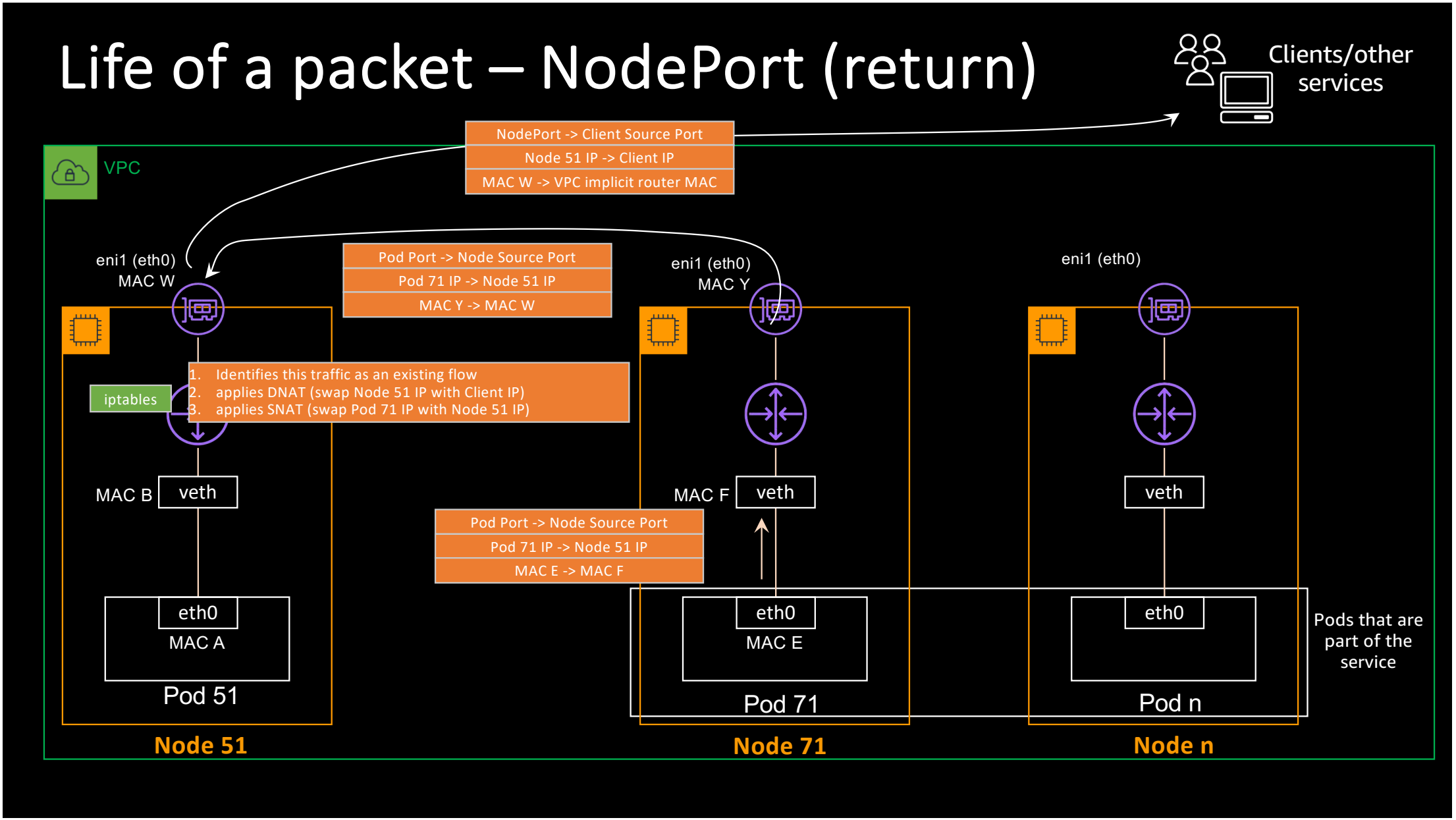
Service Type - NodePort



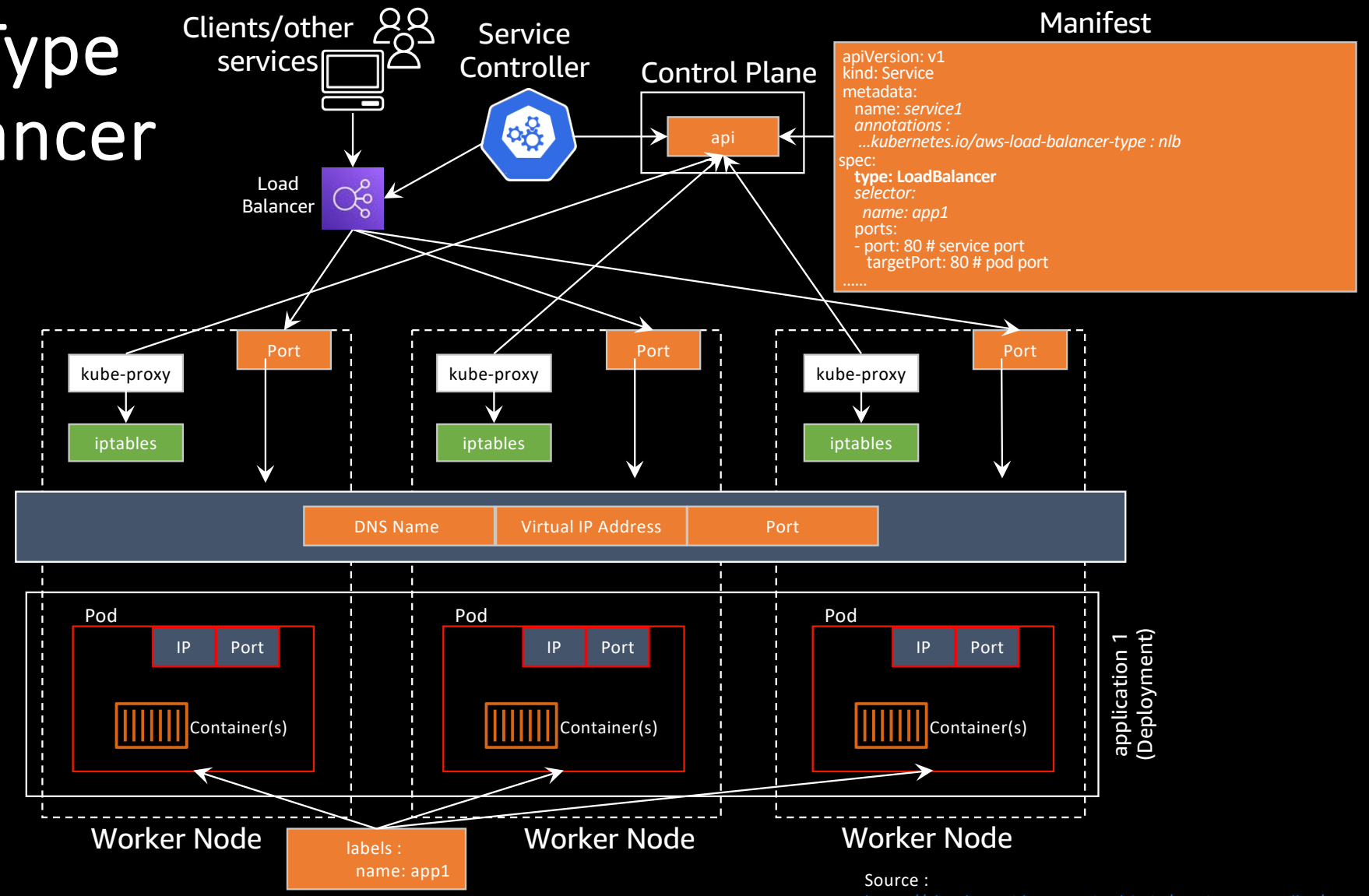
Life of a packet – NodePort



Life of a packet – NodePort (return)

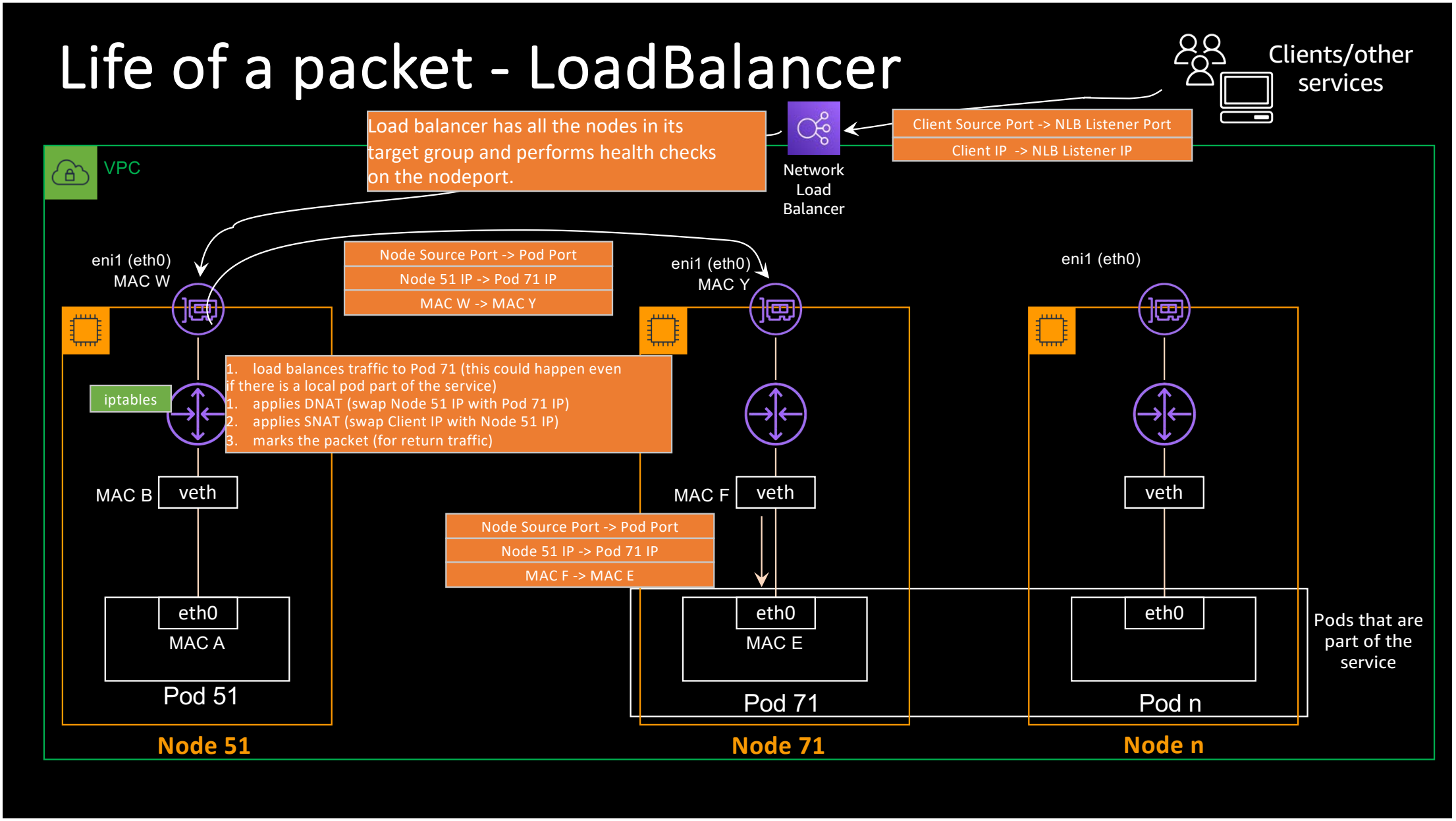


Service Type LoadBalancer

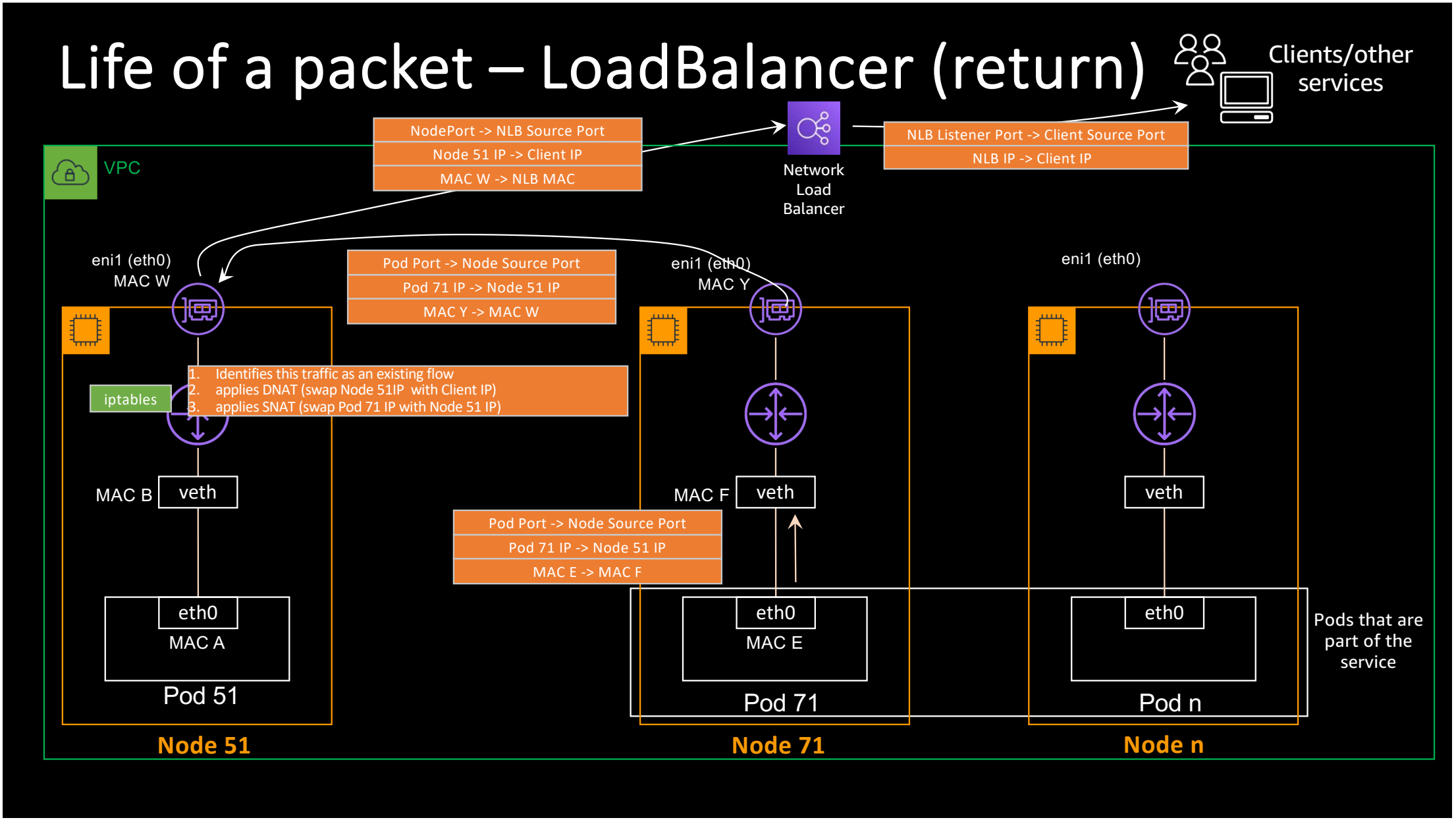


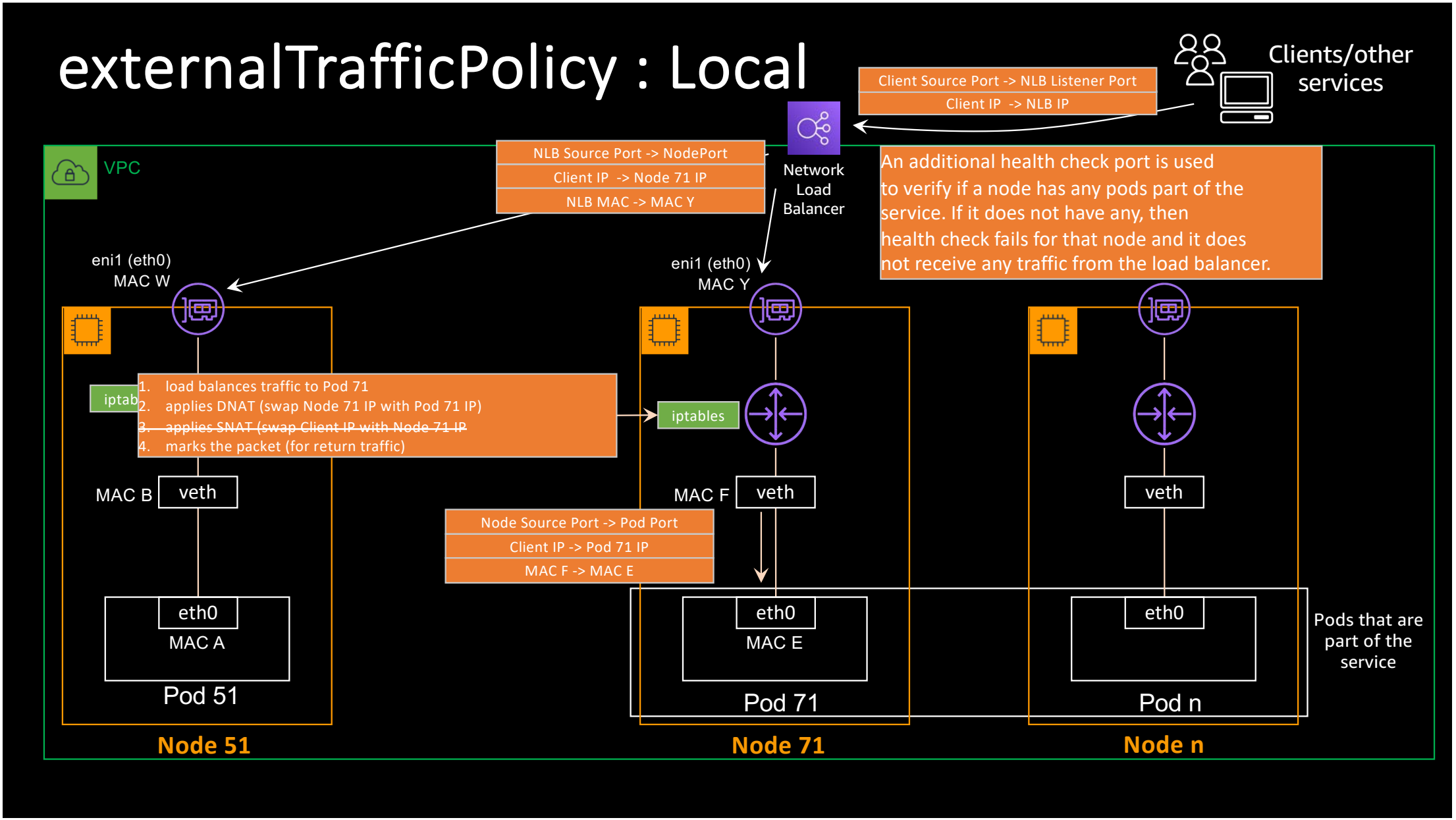
Source :
https://cloud-provider-aws.sigs.k8s.io/service_controller/
https://kubernetes-sigs.github.io/aws-load-balancer-controller

Life of a packet - LoadBalancer

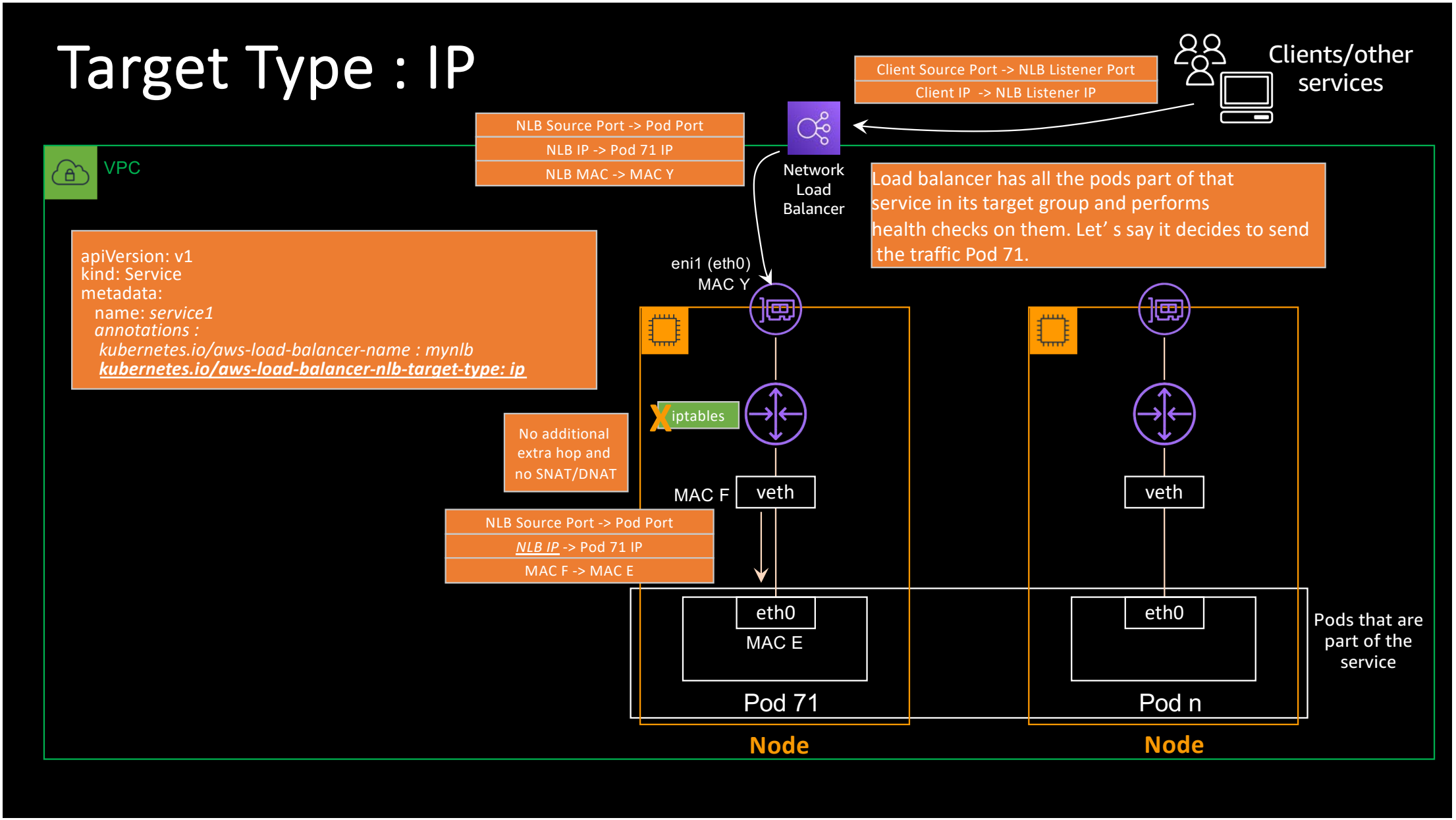


Life of a packet – LoadBalancer (return)

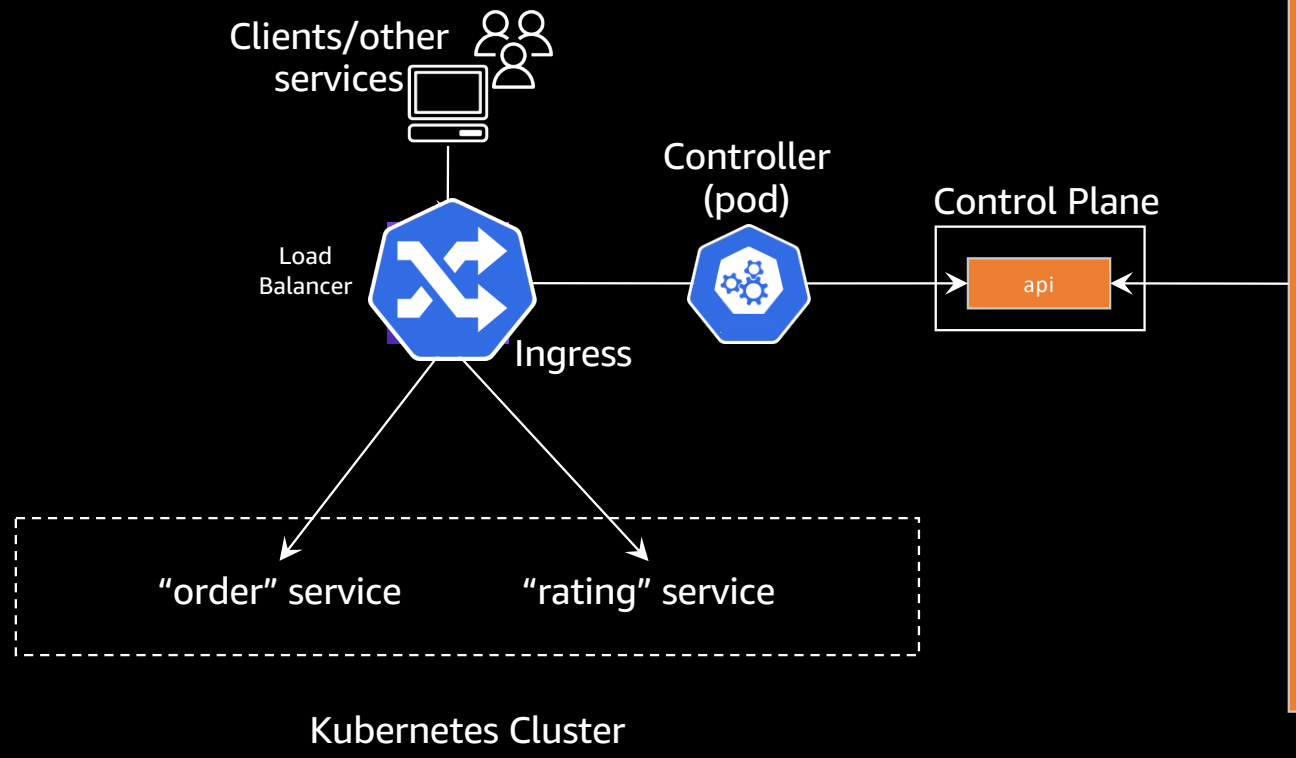



externalTrafficPolicy : Local
Clients/other services
Client Source Port -> NLB Listener Port
Client IP -> NLB IP
Network Load Balancer
NLB Source Port -> NodePort
Client IP -> Node 71 IP
NLB MAC -> MAC Y
An additional health check port is used to verify if a node has any pods part of the service. If it does not have any, then health check fails for that node and it does not receive any traffic from the load balancer.
VPC
eni1 (eth0) MAC W
eni1 (eth0) MAC Y
iptab
1. load balances traffic to Pod 71
2. applies DNAT (swap Node 71 IP with Pod 71 IP)
3. applies SNAT (swap Client IP with Node 71 IP
4. marks the packet (for return traffic)
iptables
MAC B
veth
MAC F
veth
veth
Node Source Port -> Pod Port
Client IP -> Pod 71 IP
MAC F -> MAC E
eth0
MAC A
eth0
MAC E
eth0
Pod 51
Pod 71
Pod n
Pods that are part of the service
Node 51
Node 71
Node n

Target Type : IP



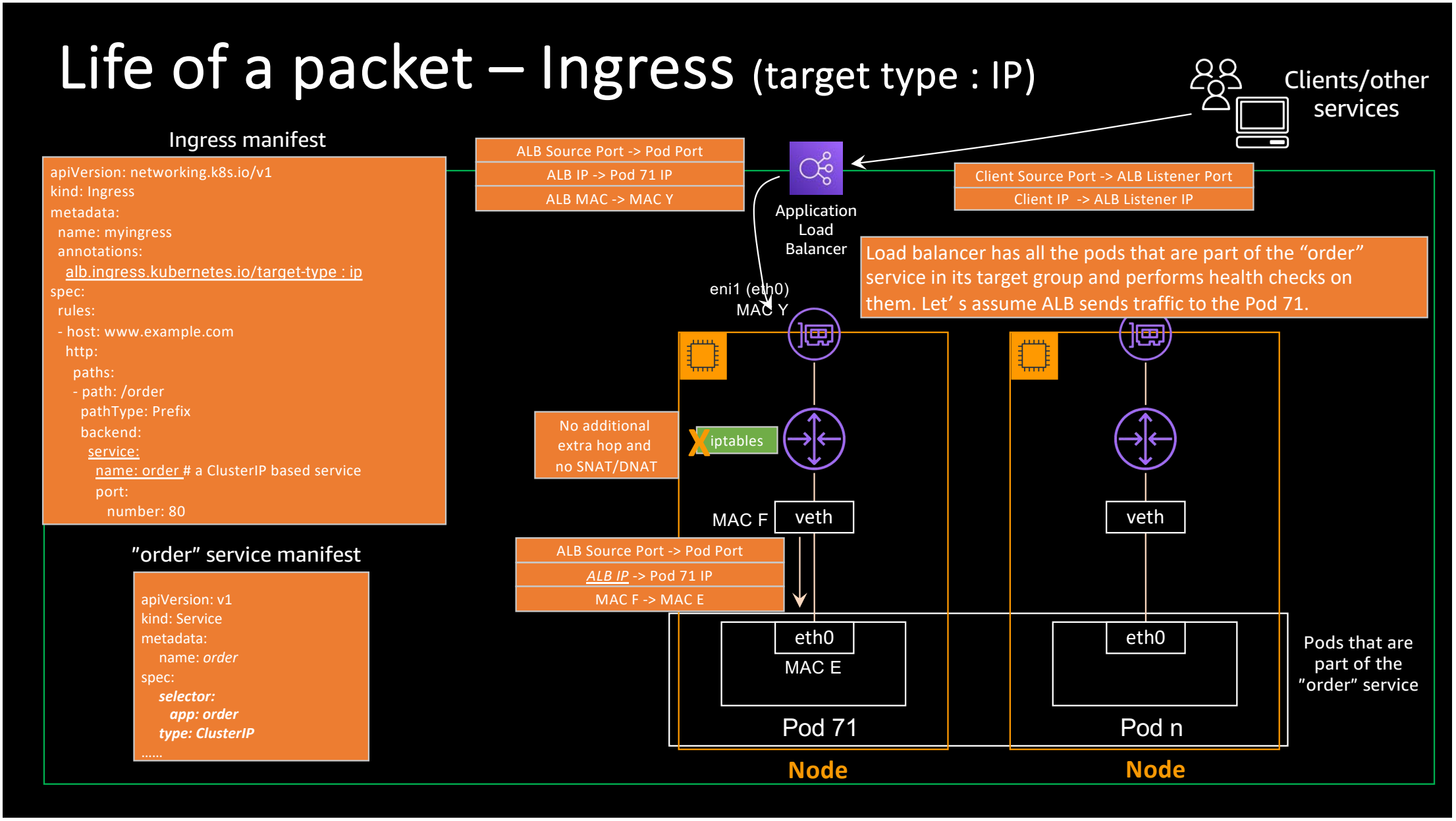
Ingress



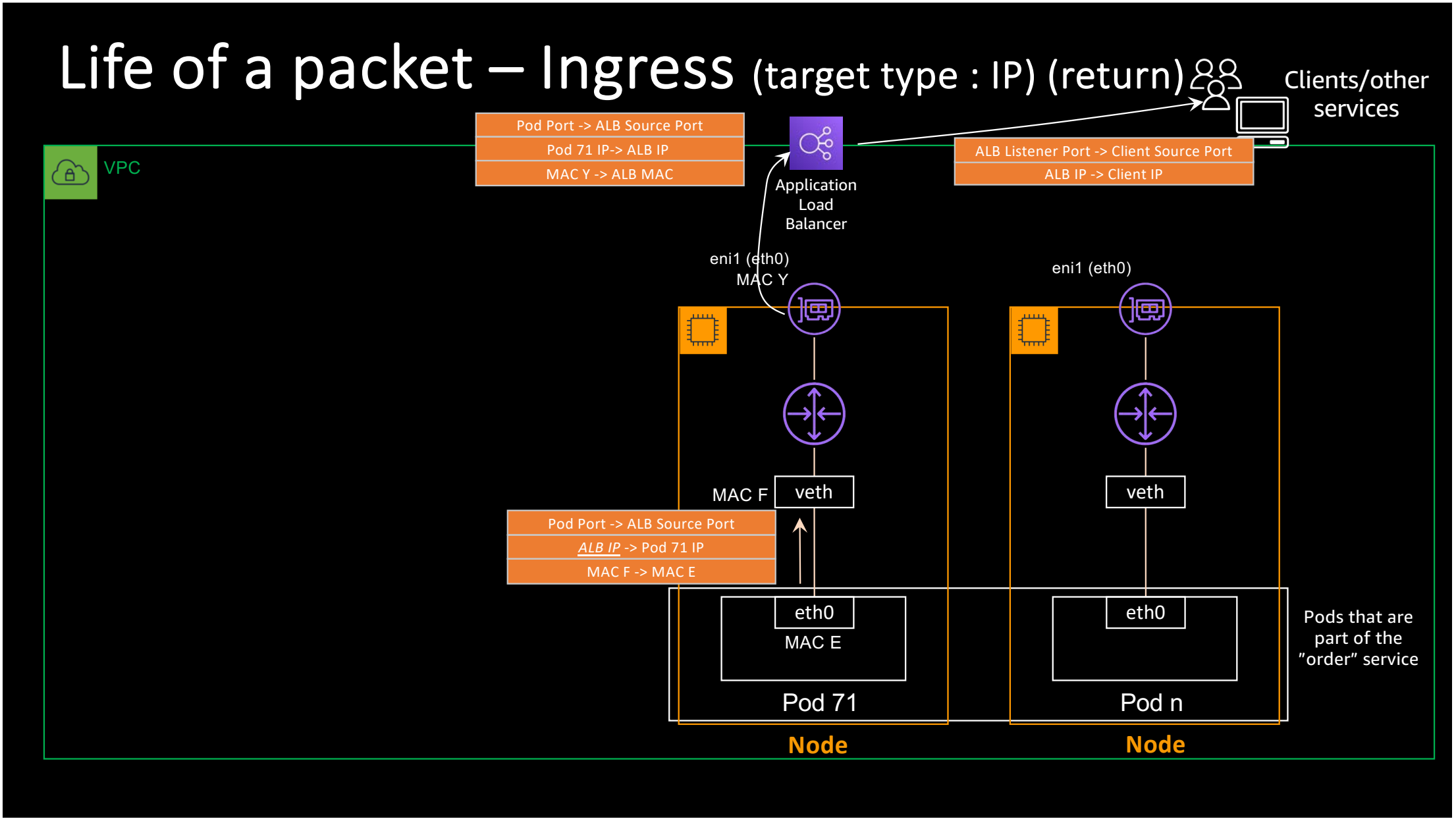
Manifest

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: myingress
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      - path: /order
        pathType: Prefix
        backend:
          service:
            name: order # a service
            port:
              number: 80
      - path: /rating
        pathType: Prefix
        backend:
          service:
            name: rating # another service
            port:
              number: 80
```

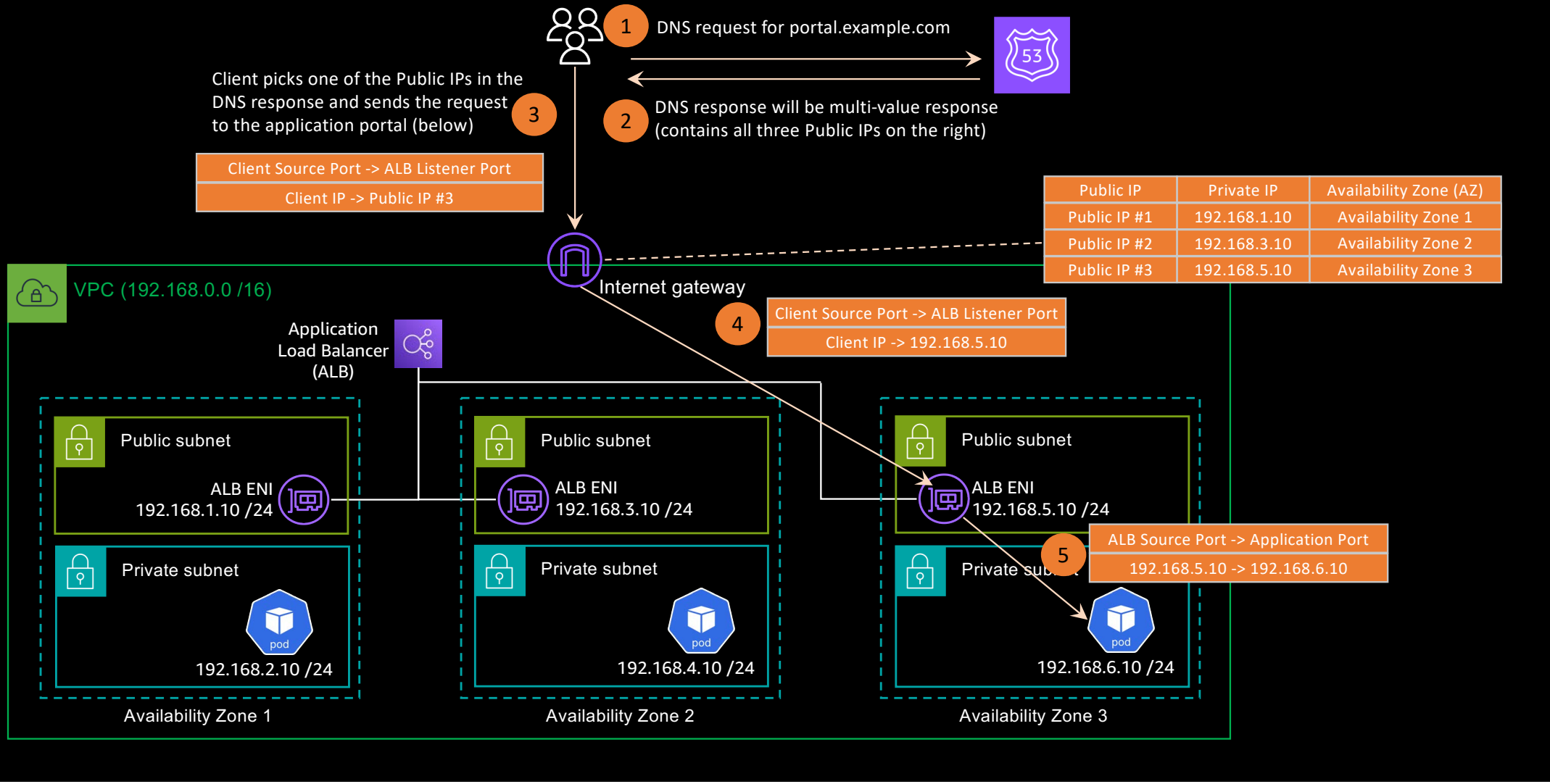
Life of a packet – Ingress (target type : IP)



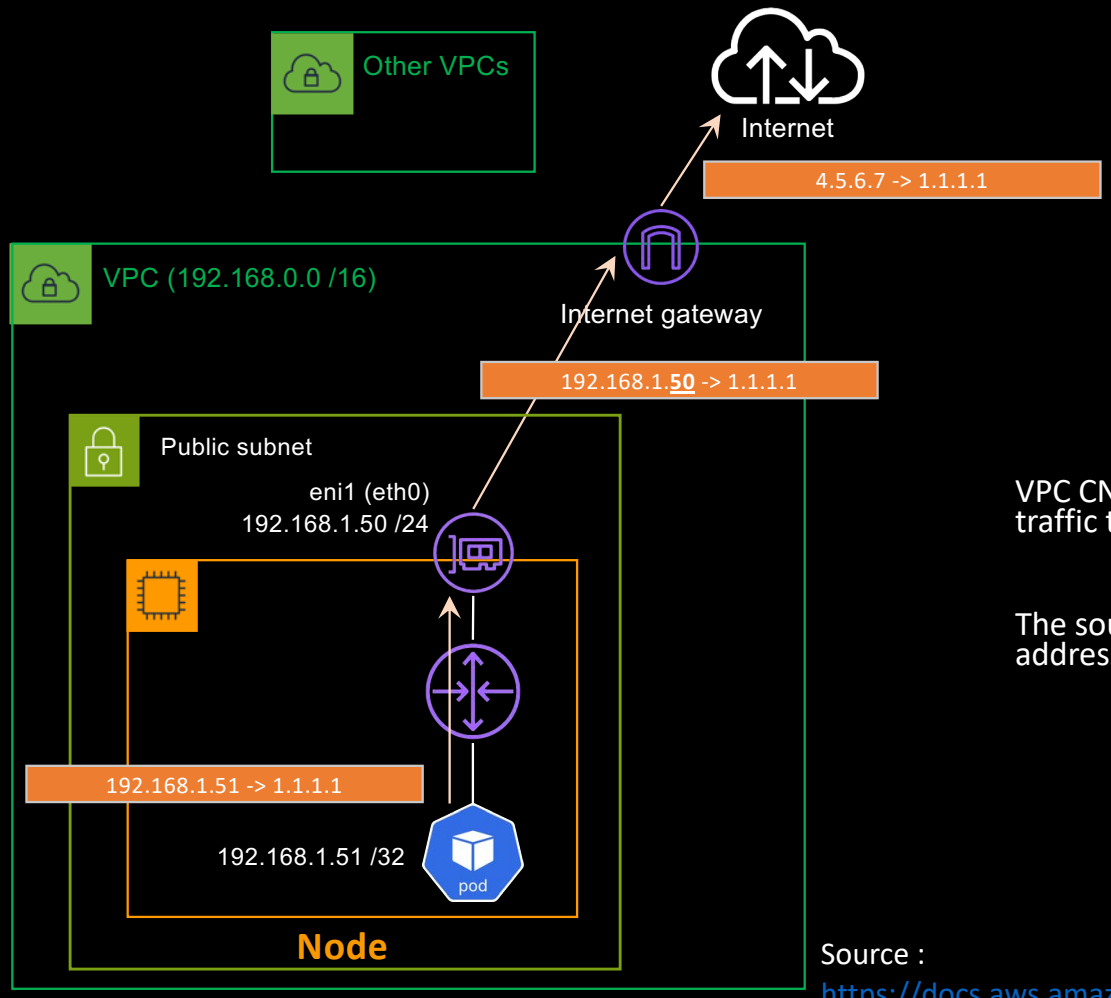
Life of a packet – Ingress (target type : IP) (return)



Stitching it all together in the context of AWS



VPC CNI SNAT



VPC CNI plugin by default, applies SNAT (source NAT) to all the traffic that is destined to external IP addresses. (non VPC CIDRs)

The source IP of the traffic is swapped with the primary IPv4 address of the primary ENI of the node.

Source :
https://docs.aws.amazon.com/eks/latest/userguide/external-snat.htm
https://github.com/aws/amazon-vpc-cni-k8s?tab=readme-ov-file#aws_vpc_k8s_cni_externalsnat

Additional Resources

Kubernetes

- Internal Traffic Policy : https://kubernetes.io/docs/concepts/services-networking/service-traffic-policy/
- Topology Aware Routing : https://kubernetes.io/docs/concepts/services-networking/topology-aware-routing/
- Traffic Distribution : https://kubernetes.io/docs/concepts/services-networking/service/#traffic-distribution

AWS

- EKS Best Practices Guide : https://docs.aws.amazon.com/eks/latest/best-practices/introduction.html
- ALB service quotas : https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-limits.html
- NLB service quotas : https://docs.aws.amazon.com/elasticloadbalancing/latest/network/load-balancer-limits.html

THANK YOU !

Dumlu Timuralp
Solutions Architect
Amazon Web Services


https://www.linkedin.com/in/dumlutimuralp/

https://dumlutimuralp.medium.com/

https://github.com/dumlutimuralp