Service

- A Kubernetes Service is an abstraction which defines a logical set of Pods running somewhere in your cluster, that all provide the same functionality.
- When created, each Service is **assigned a unique IP address (also called clusterIP). This address is tied to the lifespan of the Service, and will not change while the Service is aliv**e
- a Service is backed by a group of Pods. These **Pods are exposed through endpoints**
- The Service's selector will be evaluated continuously and the results will be POSTed to an **Endpoints object also named my-nginx**. When a Pod dies, it is automatically removed from the endpoints, and new Pods matching the Service's selector will automatically get added to the endpoints.
- Kubernetes supports 2 primary modes of finding a Service - **environment variables and DNS.**

```
kubectl exec my-nginx-3800858182-jr4a2 -- printenv | grep SERVICE
```

```
kubectl run curl --image=radial/busyboxplus:curl -i --tty
nslookup my-nginx
```

- **While creating a service own IP address can be chosen. You can specify your own cluster IP address as part of a Service creation request. To do this, set the .spec.clusterIP field. For example, if you already have an existing DNS entry that you wish to reuse, or legacy systems that are configured for a specific IP address and difficult to re-configure.**

Securing service with Certs (HTTPS)

https://kubernetes.io/docs/concepts/services-networking/connect-applications-service/#securing-the-service

https://github.com/kubernetes/examples/tree/master/staging/https-nginx/

https://phoenixnap.com/kb/openssl-tutorial-ssl-certificates-private-keys-csrs

Service Types

ClusterIP

With Selector

- Endpoints are auto created
- Will have IP address of target Pods as per selector

Without Selector

- No endpoints are created
- Endpoint can be manually created (using yaml) and assign static IP
- Useful for connecting to external services like mongo db.

Create Endpoints

```yaml
kind: Service
apiVersion: v1
metadata:
 name: mongo
spec:
 ports:
 - port: 27017
   targetPort: 49763
```

```yaml
kind: Endpoints
apiVersion: v1
metadata:
 name: mongo
subsets:
 - addresses:
     - ip: 35.188.8.12
   ports:
     - port: 49763
```

NodePort

- Kubernetes control plane allocates a port from a range specified by --service-node-port-range flag (default: 30000-32767).
- Each node proxies that port (the same port number on every Node) into your Service.

LoadBalancer

ExternalName

- Maps the Service to the contents of the externalName field (e.g. foo.bar.example.com), by returning a CNAME record with its value. No proxying of any kind is set up.
- Useful for connecting to external services like DB..
- When the external service in brought to kubernetes, no need of change in client application… Just update the service type…

ExternalIPs

If there are external IPs that route to one or more cluster nodes, Kubernetes Services can be exposed on those externalIPs. Traffic that ingresses into the cluster with the external IP (as destination IP), on the Service port, will be routed to one of the Service endpoints. externalIPs are not managed by Kubernetes and are the responsibility of the cluster administrator.

In the Service spec, **externalIPs can be specified along with any of the ServiceTypes**

Headless Services

- **cluster IP is not allocated**, kube-proxy does not handle these Services, and there is no load balancing or proxying done by the platform for them
- .spec.clusterIP: None
- You can use a **headless Service to interface with other service discovery mechanisms,** without being tied to Kubernetes' implementation.

With Selector

- Endpoints are auto created
- the endpoints controller creates Endpoints records in the API, and modifies the DNS configuration to return records (addresses) that point directly to the Pods backing the Service

Without Selector

- No endpoints are created
- Endpoint can be manually created (using yaml) and assign static IP

DNS system looks for and configures either:

- CNAME records for ExternalName-type Services.
- A records for any Endpoints that share a name with the Servic

DNS

For Services

Every service has A record and SRV record

My-svc.my-namespace.svc.cluster-domain.example

_my-port-name._my-port-protocol.my-svc.my-namespace.svc.cluster-domain.example

For Pods

Hostnames and subdomain

DNS policy

DNS Config

Very Important..

https://kubernetes.io/docs/tasks/administer-cluster/dns-custom-nameservers/

CoreDNS - Configuring subdomain and upstream name server

If a cluster operator has a Consul domain server located at 10.150.0.1, and all Consul names have the suffix .consul.local. To configure it in CoreDNS, the cluster administrator creates the following stanza in the CoreDNS ConfigMap.

```
consul.local:53 {
```

```
        errors
        cache 30
        forward . 10.150.0.1
    }
```

To explicitly force all non-cluster DNS lookups to go through a specific nameserver at 172.16.0.1, point the `forward` to the nameserver instead of `/etc/resolv.conf`

```
forward .  172.16.0.1
```

The final ConfigMap along with the default `Corefile` configuration looks like:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns
  namespace: kube-system
data:
  Corefile: |
    .:53 {
        errors
        health
        kubernetes cluster.local in-addr.arpa ip6.arpa {
           pods insecure
           fallthrough in-addr.arpa ip6.arpa
        }
        prometheus :9153
        forward . 172.16.0.1
        cache 30
        loop
        reload
        loadbalance
    }
    consul.local:53 {
        errors
        cache 30
        forward . 10.150.0.1

    }
```

Kube DNS

Kube-dns is now available as an optional DNS server since CoreDNS is now the default.

The running DNS Pod holds 3 containers: **CoreDNS only has one**

- "`kubedns`": watches the Kubernetes master for changes in Services and Endpoints, and maintains in-memory lookup structures to serve DNS requests.
- "`dnsmasq`": adds DNS caching to improve performance.
- "`sidecar`": provides a single health check endpoint to perform healthchecks for `dnsmasq` and `kubedns`.

Effects on Pods

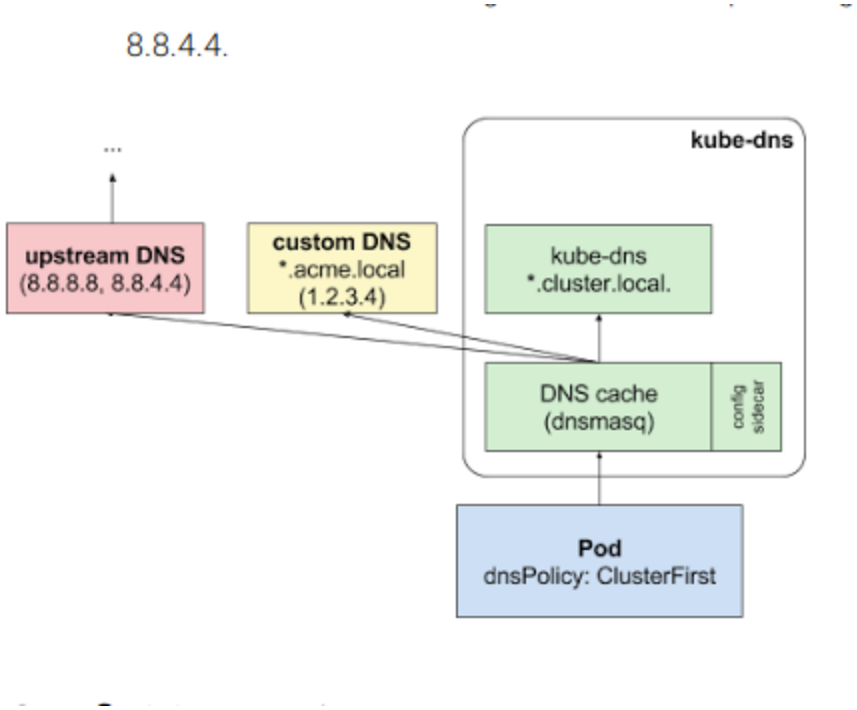
Custom upstream nameservers and stub domains do not affect Pods with a `dnsPolicy` set to "`Default`" or "`None`".

If a Pod's `dnsPolicy` is set to "`ClusterFirst`", its name resolution is handled differently, depending on whether stub-domain and upstream DNS servers are configured.

Without custom configurations: Any query that does not match the configured cluster domain suffix, such as "www.kubernetes.io", is forwarded to the upstream nameserver inherited from the node.

With custom configurations: If stub domains and upstream DNS servers are configured, DNS queries are routed according to the following flow:

1. The query is first sent to the DNS caching layer in kube-dns.
2. From the caching layer, the suffix of the request is examined and then forwarded to the appropriate DNS, based on the following cases:
    - *Names with the cluster suffix*, for example ".cluster.local": The request is sent to kube-dns.
    - *Names with the stub domain suffix*, for example ".acme.local": The request is sent to the configured custom DNS resolver, listening for example at 1.2.3.4.
    - *Names without a matching suffix*, for example "widget.com": The request is forwarded to the upstream DNS, for example Google public DNS servers at 8.8.8.8 and 8.8.4.4.



CoreDNS configuration equivalent to kube-dns

CoreDNS supports the features of kube-dns and more. A ConfigMap created for kube-dns to support `StubDomains`and `upstreamNameservers` translates to the `forward`plugin in CoreDNS. Similarly, the `Federations` plugin in kube-dns translates to the `federation` plugin in CoreDNS.

Example

This example ConfigMap for kubedns specifies federations, stubdomains and upstreamnameservers:

```
apiVersion: v1
data:
  federations: |
    {"foo" : "foo.feddomain.com"}
  stubDomains: |
```

```
    {"abc.com" : ["1.2.3.4"], "my.cluster.local" : ["2.3.4.5"]}
  upstreamNameservers: |
    ["8.8.8.8", "8.8.4.4"]

kind: ConfigMap
```

The equivalent configuration in CoreDNS creates a Corefile:

For federations:

```
federation cluster.local {
       foo foo.feddomain.com
```

- `    }`

For stubDomains:

```
abc.com:53 {
    errors
    cache 30
    forward . 1.2.3.4
}
my.cluster.local:53 {
    errors
    cache 30
    forward . 2.3.4.5
```

- `}`

The complete Corefile with the default plugins:

```
.:53 {
        errors
        health
        kubernetes cluster.local in-addr.arpa ip6.arpa {
           pods insecure
           fallthrough in-addr.arpa ip6.arpa
        }
        federation cluster.local {
           foo foo.feddomain.com
```

```
        }
        prometheus :9153
        forward .  8.8.8.8 8.8.4.4
        cache 30
    }
    abc.com:53 {
        errors
        cache 30
        forward . 1.2.3.4
    }
    my.cluster.local:53 {
        errors
        cache 30
        forward . 2.3.4.5

    }
```

Ingress

- Ingress **exposes HTTP and HTTPS routes** from outside the cluster to services within the cluster.
- Traffic routing is controlled by rules defined on the Ingress resource.
- An Ingress controller is responsible for fulfilling the Ingress,
- An Ingress can be configured to give
    - **Services externally-reachable URLs,**
    - **load balance traffic,**
    - **terminate SSL / TLS,**
    - **offer name based virtual hosting.**
- Ingress resource **only supports rules for directing HTTP traffic**.

Ingress rules

- **Host (optional)**
    - No host - rule applies to all inbound HTTP traffic through the IP address specified
    - Host present - the rules apply to that host
- Paths - Path have associated backend
- Backend --- serviceName and servicePort

Request not matching rules is sent to **default backend** which is configured on the ingress controller. **An Ingress with no rules sends all traffic to a single default backend.**
If none of the hosts or paths match the HTTP request in the Ingress objects, the traffic is routed to your default backend

Example:

```
apiVersion: networking.k8s.io/v1beta1 # for versions before 1.14 use extensions/v1beta1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
 rules:
 - host: hello-world.info
   http:
     paths:
     - path: /*
       backend:
         serviceName: web
         servicePort: 8080
```

Types on ingress

- Single service ingress
  - All traffic to single service
  - Alternatives - Define service type as LoadBalancer or NodeType

`kubectl get ingress test-ingress`

```
NAME           HOSTS     ADDRESS          PORTS     AGE
test-ingress   *         107.178.254.228  80        59s
```

107.178.254.228 is the IP allocated by the Ingress controller to satisfy this Ingress.
This rule is only applied to traffic coming through this IP

- Simple Fanout
  - routes traffic from a single IP address to more than one Service, based on the HTTP URI being requested.

- Name based virtual hosting
  - **Hostname based routing**
  - **Name-based virtual hosts support routing HTTP traffic to multiple host names at the same IP address.**

- TLS
  - secure an Ingress by specifying a Secret that contains a TLS private key and certificate
  - **Referencing this secret in an Ingress tells the Ingress controller to secure the channel from the client to the load balancer using TLS.**
  - You need to make sure the TLS secret you created came from a certificate that contains a CN for sslexample.foo.com.
  - assumes TLS termination

- Loadbalancing
  - An Ingress controller is bootstrapped with some load balancing policy settings that it applies to all Ingress, such as the load balancing algorithm, backend weight scheme, and others

Ingress.class annotation

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: test
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  tls:
  - secretName: tls-secret
  backend:
```

serviceName: echoheaders-https
servicePort: 80

- The **GCE controller** will only act on Ingresses with the annotation value of "gce" or empty string "" (the default value if the annotation is omitted).
- The nginx controller will only act on Ingresses with the annotation value of "nginx" or empty string "" (the default value if the annotation is omitted).

Rewrite-target

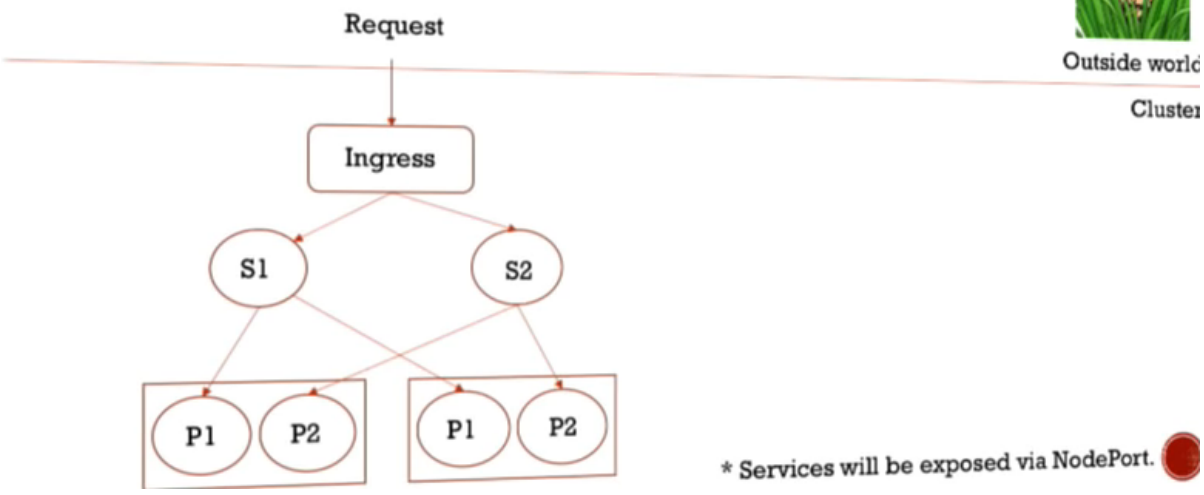
Traffic path URL redirected to “/” .. Verify..

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: simple-fanout-example
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        backend:
          serviceName: service1
          servicePort: 4200
      - path: /bar
        backend:
          serviceName: service2
          servicePort: 8080
```

WHAT IS INGRESS?

- **Ingress:** The action or fact of going in or entering
- **Ingress Resource:** An API object that manages external access to the services in a cluster, typically HTTP
- **Service:** Is an abstraction which defines a logical set of Pods and exposes them to the outside world
- **Ingress Controller:** Monitors Ingress resources via the Kubernetes API and updates the configuration of a load balancer in case of any changes
- **Nginx:** Open source software for web serving, reverse proxy, caching, load balancing, media streaming, and more
- **Ingress-Nginx:** Ingress controller for NGINX

WHAT'S IT ALL LOOK LIKE?



* Services will be exposed via NodePort.

INGRESS-NGINX VS. KUBERNETES-INGRESS

Kubernetes/Ingress-Nginx	Nginxinc/Kubernetes-Ingress
Kubernetes Community	NGINX Inc. and Community
Custom NGINX build with several 3rd party modules	Official NGINX build
One Configuration File generated from all Ingress Resources	One Configuration File per Ingress Resource
Some different annotations, Configuration Map Values, Templates, and Supported Configurations	Some different annotations, Configuration Map Values, Templates, and Supported Configurations

CONFIGURING INGRESS-NGINX

- **Annotations:** Sets a specific configuration for a particular Ingress rule
- **Configuration Map:** Sets global configurations in NGINX

Ingress Controller

- ingress controllers are not started automatically with a cluster
- Kubernetes as a project currently supports and maintains GCE and nginx controllers.
- Multiple ingress controllers can be running a time .. example GCE and nginx
- When you create an ingress, you should annotate each ingress with the appropriate ingress.class to indicate which ingress controller should be used if more than one exists within your cluster.

  kubernetes.io/ingress.class: "gce"

  kubernetes.io/ingress.class: "nginx"

- **Race condition** - Deploying **multiple Ingress controllers**, of different types (e.g., ingress-nginx & gce), and **not specifying a class annotation** will result in both or all controllers fighting to satisfy the Ingress, and all of them racing to update Ingress status field in confusing ways.
- **Multiple nginx controllers -**
  To do this, the option **--ingress-class must be changed to a value unique for the cluster** within the definition of the replication controller. Here is a partial example:

```
spec:
  template:
    spec:
      containers:
        - name: nginx-ingress-internal-controller
          args:
            - /nginx-ingress-controller
            - '--election-id=ingress-controller-leader-internal'
            - '--ingress-class=nginx-internal'
            - '--configmap=ingress/nginx-ingress-internal-controller'
```

- **Best practice - one controller for each namespace**

Network Policy

- A network policy is a specification of **how groups of pods are allowed to communicate with each other and other network endpoints**.
- use labels to select pods and define rules which specify what traffic is allowed to the selected pods
- Network policies **are implemented by the network plugin**
- **Networking solution used should support network policies**
- **There needs to be a controller that implement this networkpolicy**
- **Non-Isolated Pods -** By default, pods are non-isolated; they accept traffic from any source.
- **Isolated Pods** - achieved by applying networking policy
- Policy is applied to a given namespace..

- **podSelector - pods to which policy will be applied .. Pods from same namespace as policy namespace**
- **NetworkPolicy - type of policy , ingress or & egress**
- **Ingress**
  - **From**
    - **IP block**
    - **Namespace -- all pods from this name space .. can be further filtered based on podSelector and labels.**
    - **Pods - Group of Pods from same namespace as policy.**
  - **Ports**
    - **Protocol and port**

```
...
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          user: alice
      podSelector:
        matchLabels:
          role: client
  ...
```

Different than

```
...
  ingress:
  - from:
```

```
    - namespaceSelector:
        matchLabels:
          user: alice
    - podSelector:
        matchLabels:
          role: client

  ...
```

- **Egress**
  - **TO**
    - **IP block**
  - **Ports**
    - **Protocol and port**
- **By default, if no policies exist in a namespace, then all ingress and egress traffic is allowed to and from pods in that namespace**

Note:

If there is at least one NetworkPolicy with a rule allowing the traffic, it means the traffic will be routed to the pod regardless of the policies blocking the traffic.

Default Policies

https://kubernetes.io/docs/concepts/services-networking/network-policies/#default-deny-all-ingress-traffic

Examples

https://kubernetes.io/docs/tasks/administer-cluster/declare-network-policy/

https://github.com/ahmetb/kubernetes-network-policy-recipes/blob/master/01-deny-all-traffic-to-an-application.md

Adding entries to Pod /etc/hosts with HostAliases

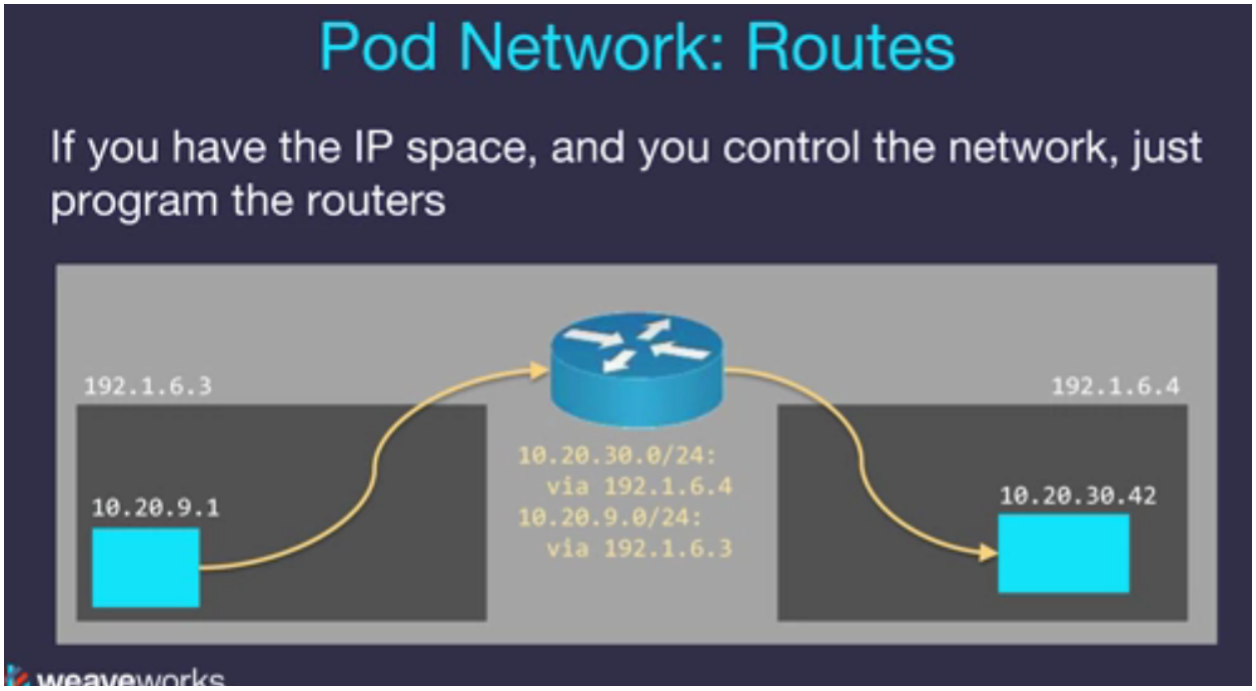
- **/etc/hosts file is managed by kubelet**
- **Manual updates will get overwritten during restart..**

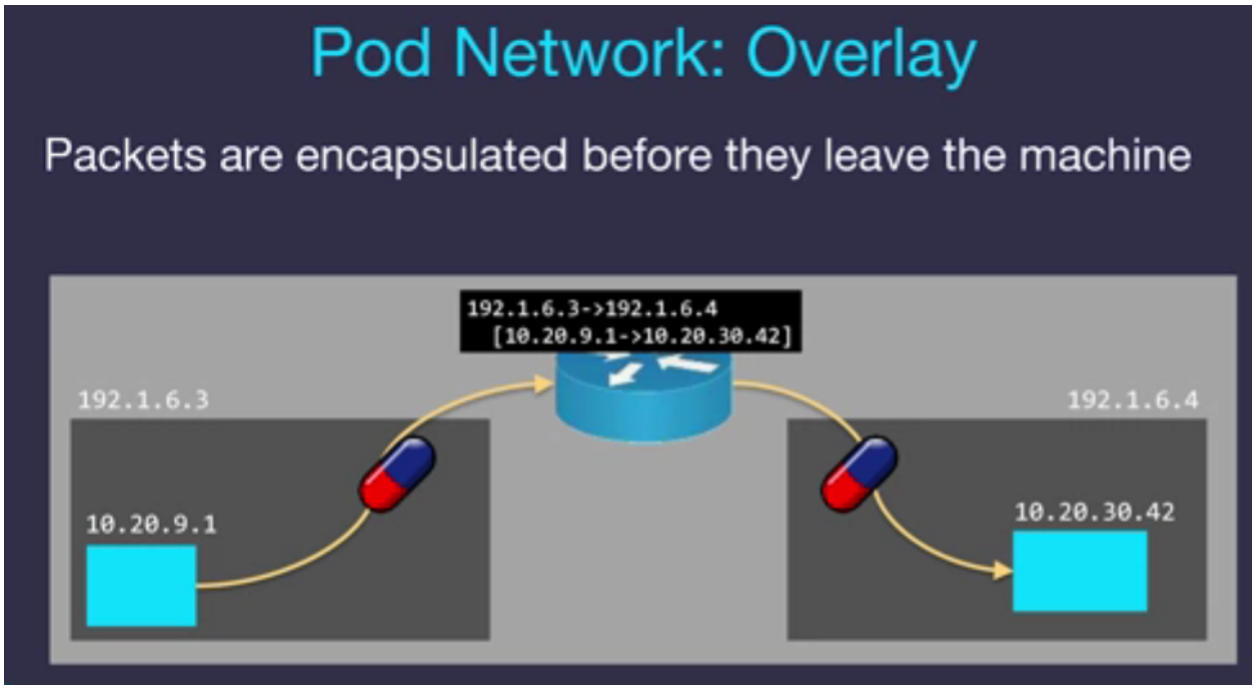
- **Modification not using HostAliases is not suggested because the file is managed by Kubelet and can be overwritten on during Pod creation/restart.**

```
apiVersion: v1
kind: Pod
metadata:
  name: hostaliases-pod
spec:
  restartPolicy: Never
  hostAliases:
  - ip: "127.0.0.1"
    hostnames:
    - "foo.local"
    - "bar.local"
  - ip: "10.1.2.3"
    hostnames:
    - "foo.remote"
    - "bar.remote"
  containers:
  - name: cat-hosts
    image: busybox
    command:
    - cat
    args:
    - "/etc/hosts"
```

https://www.digitalocean.com/community/tutorials/how-to-inspect-kubernetes-networking

Docker ps
docker inspect bdf8bb7f159f | grep Pid
nsenter -t 53901 -n ip addr
→ shows pipe number..
Host: ip addr
→ virtual ethernet pipe to pod



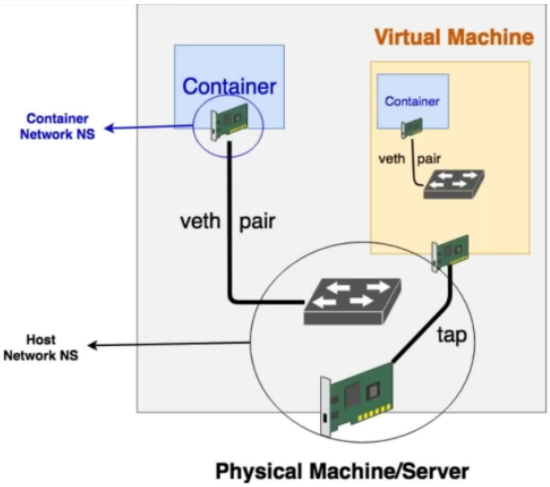


https://www.youtube.com/watch?v=GXq3FS8M_kw&t=791s

Out of the box linux networking...

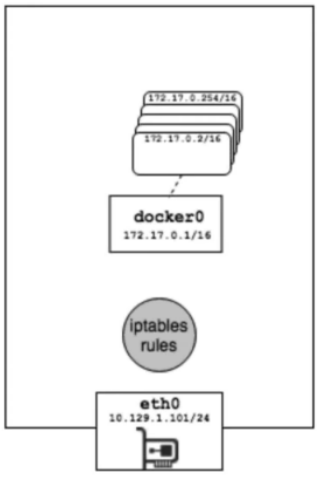
Linux Virtual Networking

- Virtual Switch/Bridge
  - Example: docker0 bridge
- Virtual Interfaces
  - Example: veth, tap
- Namespaces
  - A network namespace is logically another copy of the network stack, with its own routes, firewall rules, and network devices.



Docker Networking

- Creates a virtual bridge called "**`docker0`**"
- Uses Source-NAT rules to provide connectivity for the containers to the outside world via the Host IP address.
- Uses Destination NAT rules to expose services running inside the containers.
- The docker network is completely isolated.(host private networking)



iptables

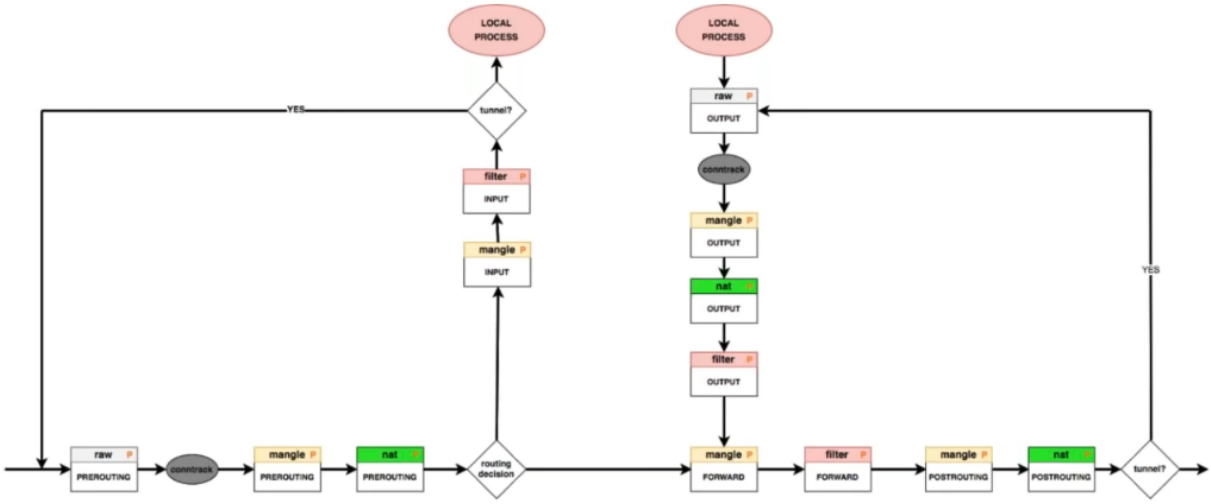
It is a user-space utility program used to configure the tables provided by the Linux kernel firewall (netfilter) and the chains and rules it stores.

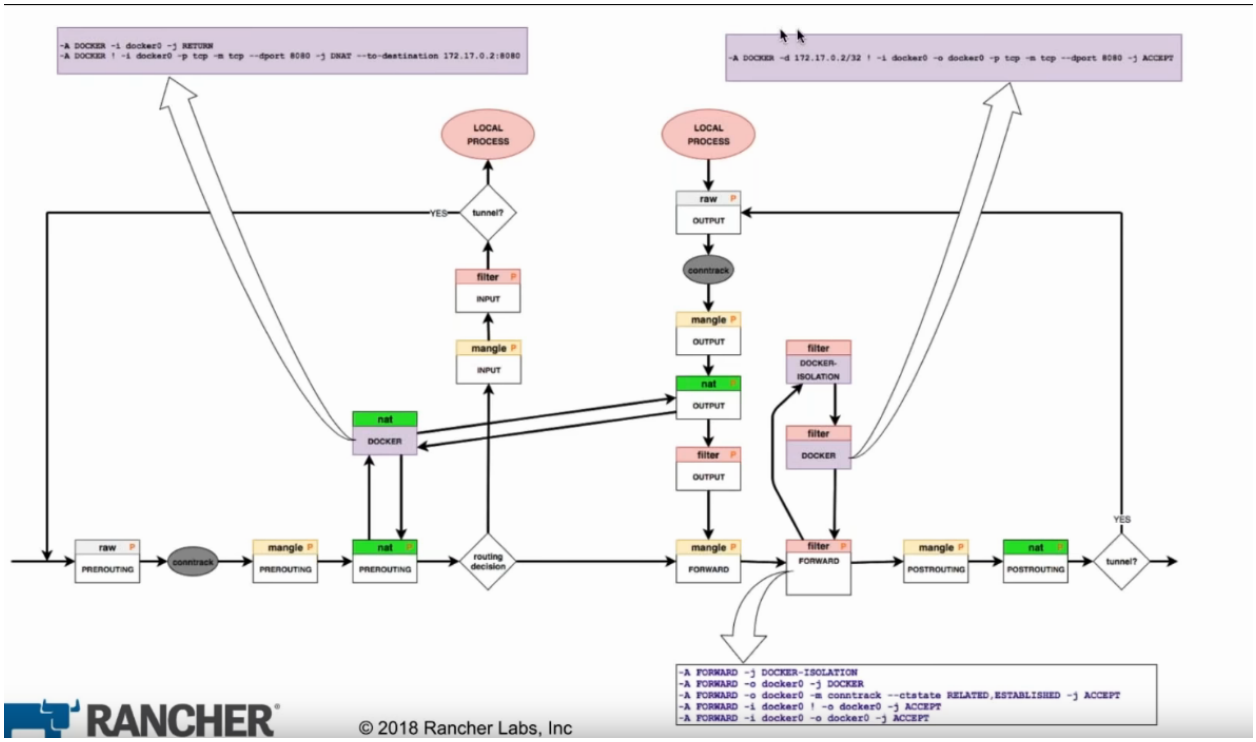
Tables:

- raw
- mangle
- nat
- filter

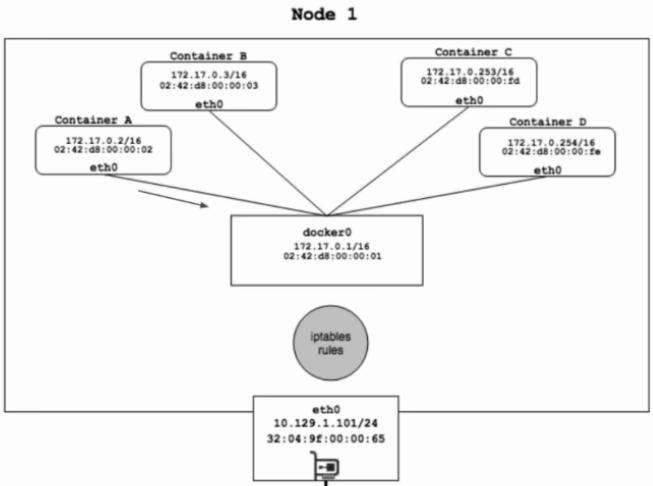
Chains:

- INPUT
- PREROUTING
- FORWARD
- POSTROUTING
- OUTPUT





Container to Container Communication

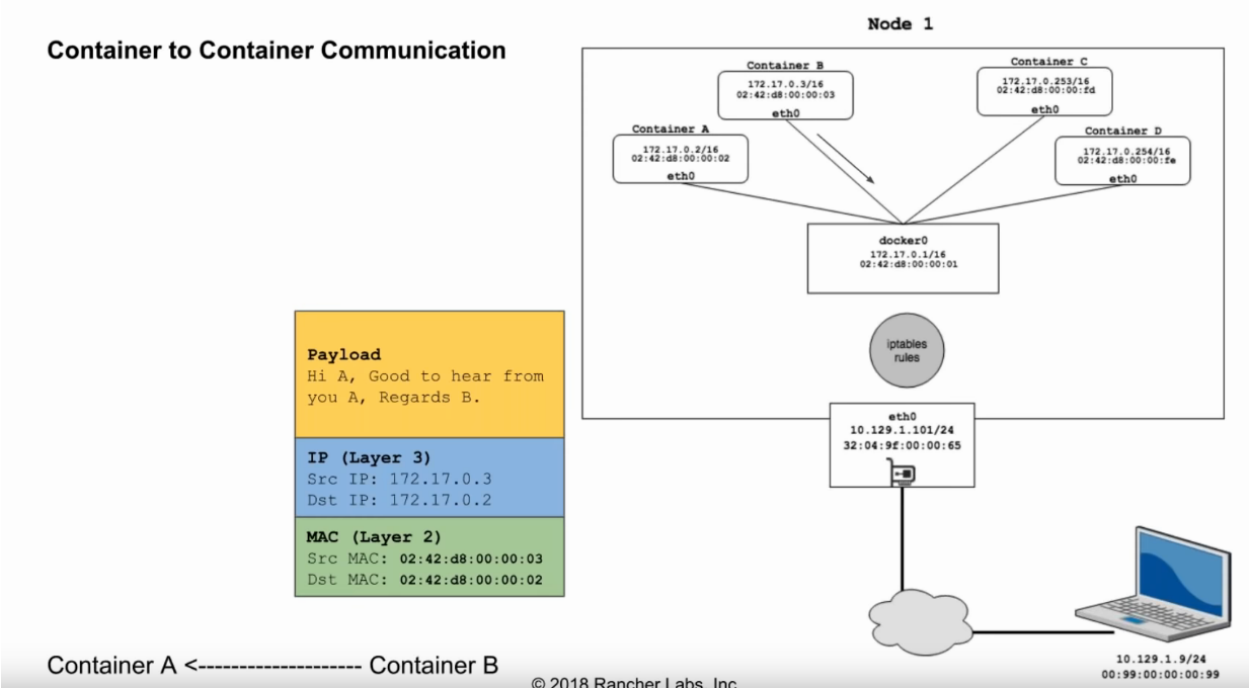


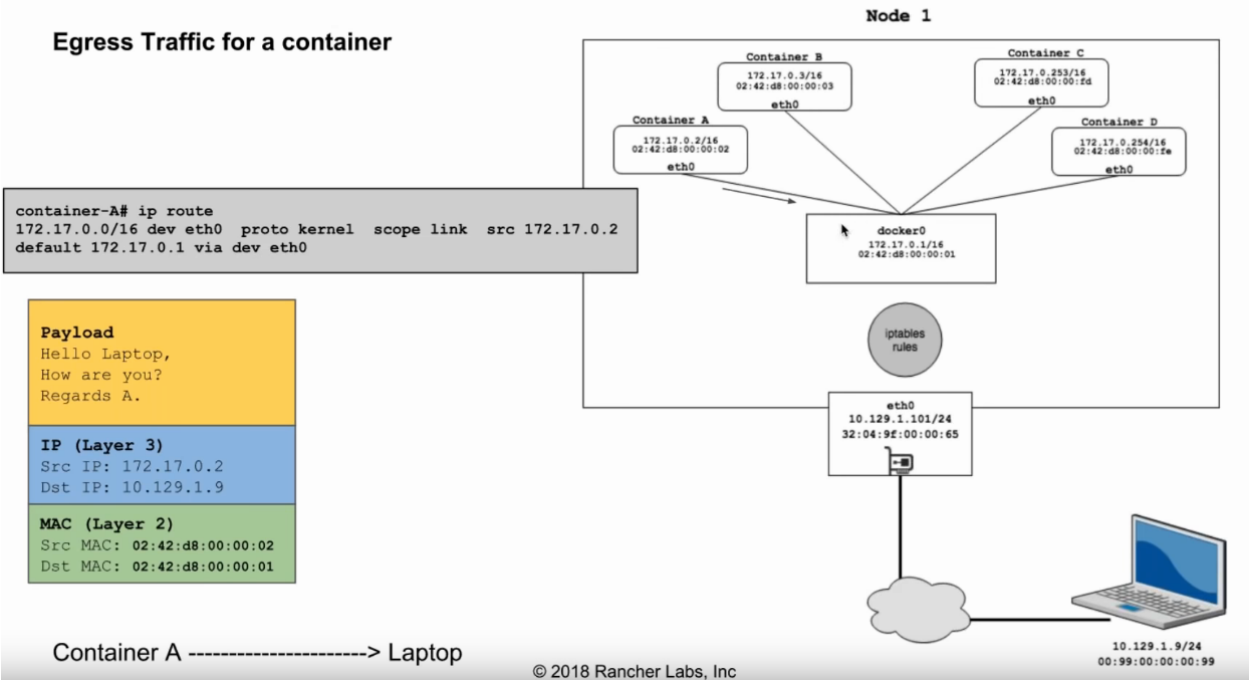
Payload
Hello B, a message from A.

IP (Layer 3)
Src IP: 172.17.0.2
Dst IP: 172.17.0.3

MAC (Layer 2)
Src MAC: 02:42:d8:00:00:02
Dst MAC: 02:42:d8:00:00:03

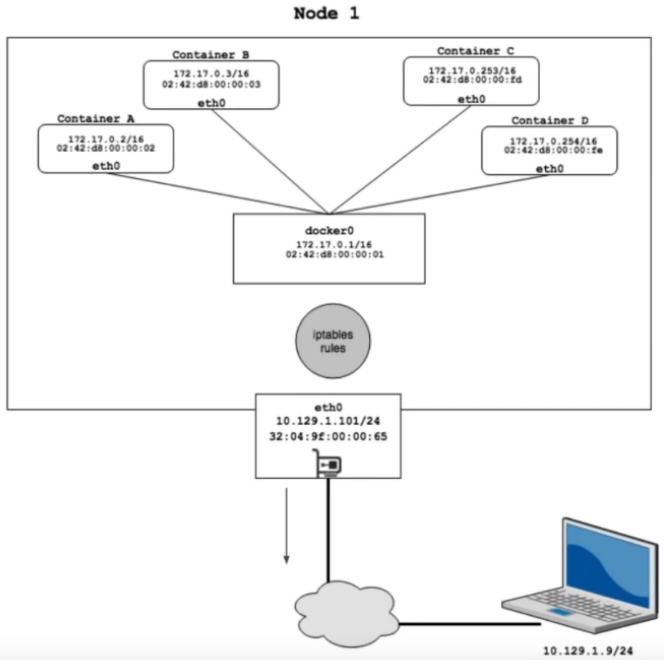
Container A ----------> Container B





Traffic to outside is netted using IP tables..
Source IP is netted ---> uses IP tables rules..

Egress Traffic for a container



Payload
Hello Laptop,
How are you?
Regards A.

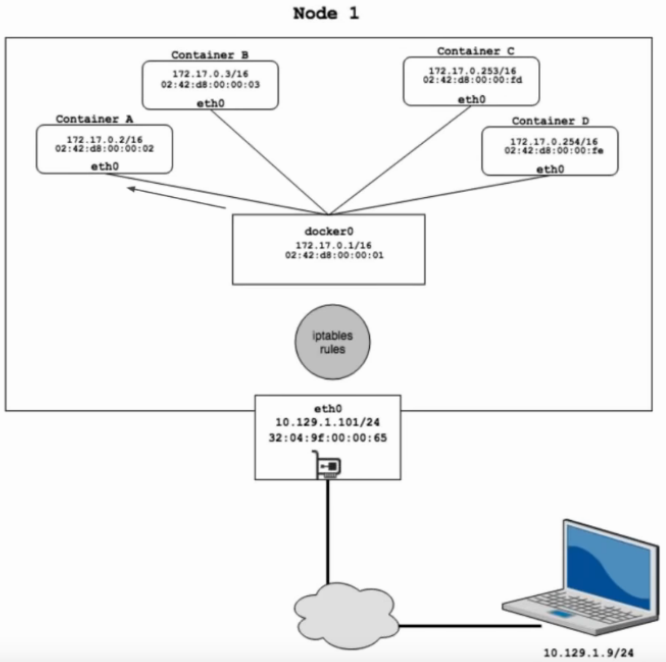
IP (Layer 3)
Src IP: 10.129.1.101
Dst IP: 10.129.1.9

MAC (Layer 2)
Src MAC: 32:04:9f:00:00:65
Dst MAC: 00:99:00:00:00:99

NAT iptables rules
172.17.0.2 -> 10.129.1.101

Container A ----------> Laptop

Egress Traffic for a container



Payload
Hello A,
I am doing good. You?
Regards Laptop.

IP (Layer 3)
Src IP: 10.129.1.9
Dst IP: 10.129.1.101

MAC (Layer 2)
Src MAC: 00:99:00:00:00:99
Dst MAC: 32:04:9f:00:00:65

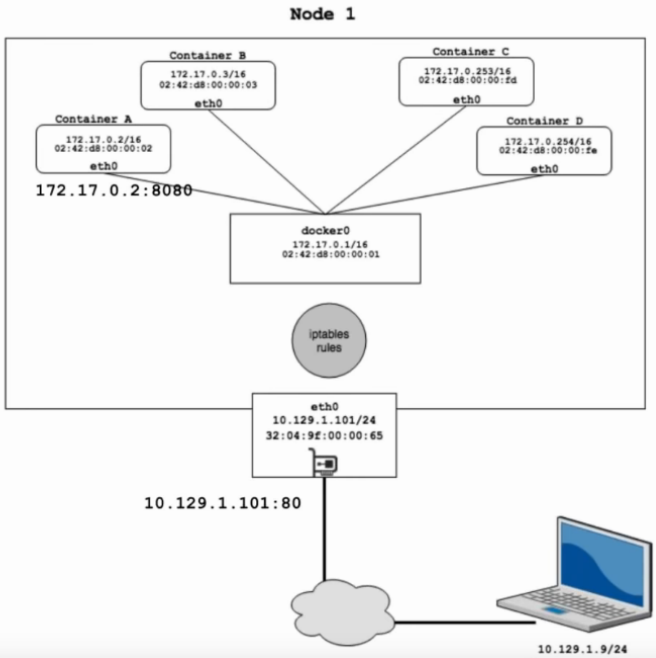
NAT iptables rules
172.17.0.2 <- 10.129.1.101

Container A <---------- Laptop

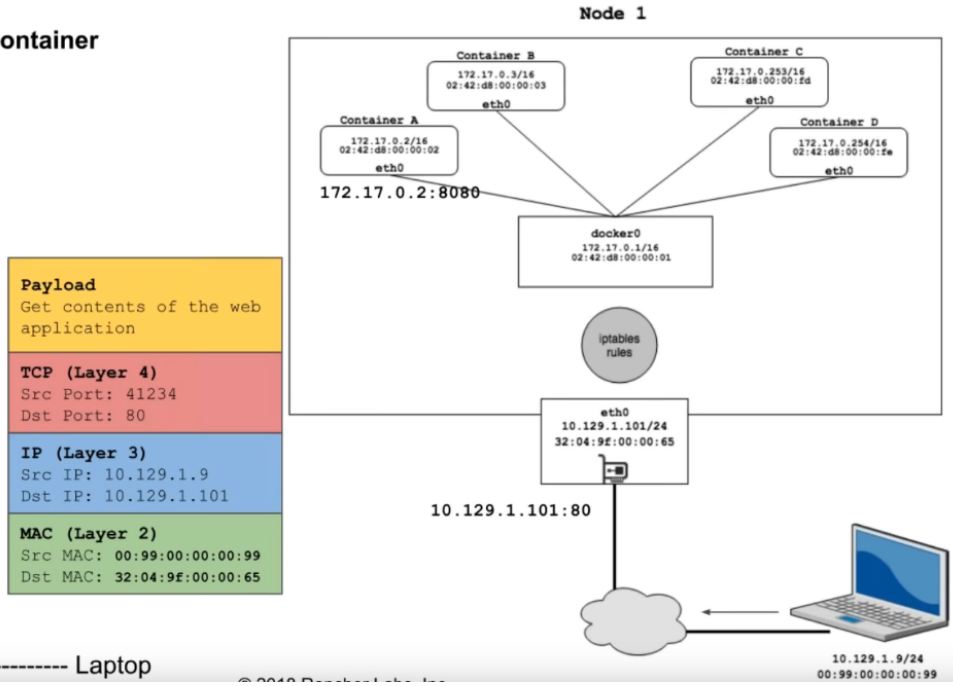
Ingress Traffic for a container

Assume Container A is a web server running on port 8080. This port is published on the host using port 80.

```
docker run \
-itd \
-p 80:8080
--name container-A \
mywebapplicationimage
```



Ingress Traffic for a container



Payload
Get contents of the web application

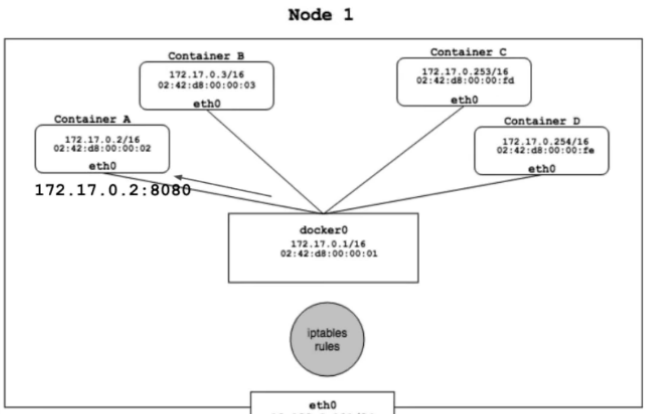
TCP (Layer 4)
Src Port: 41234
Dst Port: 80

IP (Layer 3)
Src IP: 10.129.1.9
Dst IP: 10.129.1.101

MAC (Layer 2)
Src MAC: 00:99:00:00:00:99
Dst MAC: 32:04:9f:00:00:65

HOST <---------- Laptop

Ingress Traffic for a container

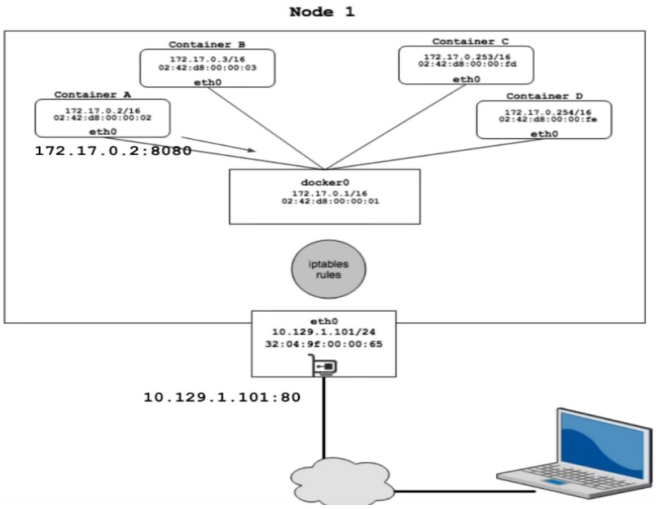


NAT iptables rules
172.17.0.2 <- 10.129.1.101
8080 <- 80

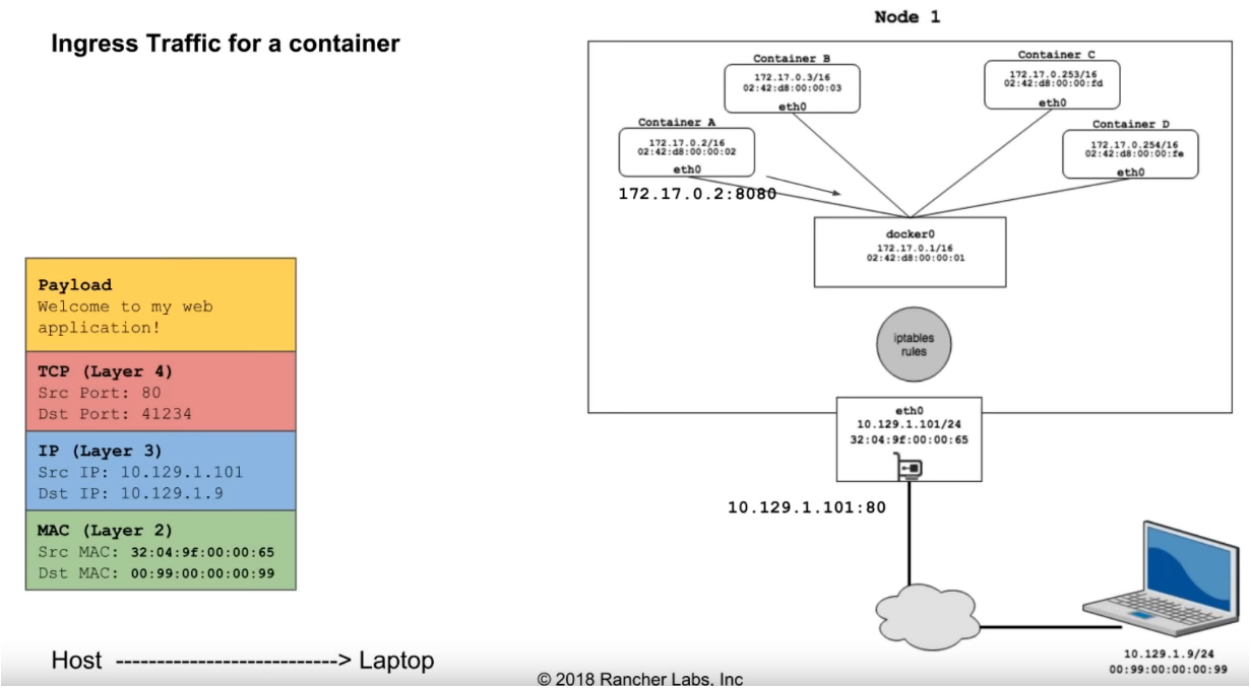
Payload Get contents of the web application
TCP (Layer 4) Src Port: 41234 Dst Port: 8080
IP (Layer 3) Src IP: 10.129.1.9 Dst IP: 172.17.0.2
MAC (Layer 2) Src MAC: 02:42:d8:00:00:01 Dst MAC: 02:42:d8:00:00:02

Container <---------- Laptop

Ingress Traffic for a container



Payload Welcome to my web application!
TCP (Layer 4) Src Port: 8080 Dst Port: 41234
IP (Layer 3) Src IP: 172.17.0.2 Dst IP: 10.129.1.9
MAC (Layer 2) Src MAC: 02:42:d8:00:00:02 Dst MAC: 02:42:d8:00:00:01

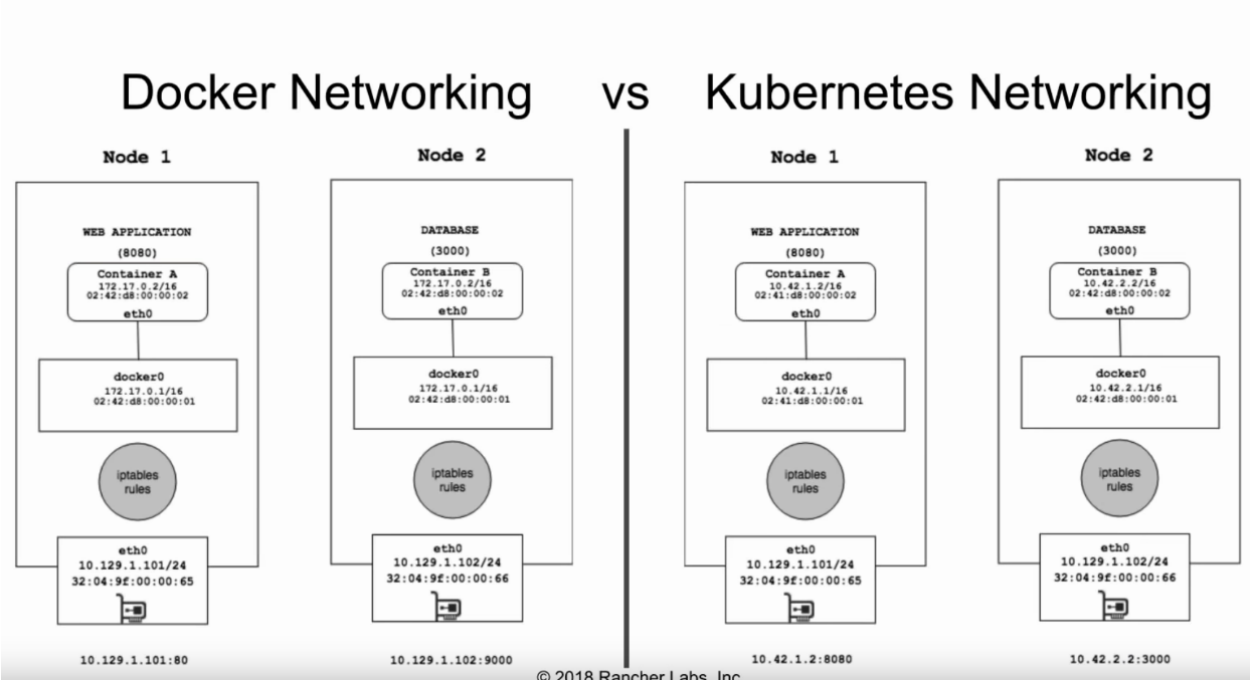


Docker networking

- Container have same Ips and different ports..
- Container port and host port mismatch..

Kubernetes networking

- Every pod have unique IP
- Host/container port can be same..



What is a POD? (In terms of networking)

- A POD is two(at the very least) or more containers running together.
  - pause container + actual container(s)
- Pause container has the actual network namespace.
- Actual container uses the network namespace of the pause container.

Docker equivalent:

```
docker run --name pause -itd ubuntu sleep infinity

docker run --name actual-app --net=container:pause -itd appimage
```

Pod Network CIDR

- Range of IP addresses that's shared by all the PODs in the entire cluster
- This big range/pool of addresses is split into multiple smaller chunks.
- Example: 10.42.0.0/16 (65536 IP addresses)
- Strategy depends on the actual needs.
- **Non overlapping with the host network CIDR.**

Strategy 1

- 10.42.0.0/24 (256 addresses for Node 1)
- 10.42.1.0/24 (256 addresses for Node 2)
- .
- .
- .
- 10.42.254.0/24 (256 addresses for Node 255)
- 10.42.255.0/24 (256 addresses for Node 256)

Strategy 2

- 10.42.0.0/25 (128 addresses for Node 1)
- 10.42.128.0/25 (128 addresses for Node 2)
- .
- .
- .
- 10.42.255.0/25 (128 addresses for Node 511)
- 10.42.255.128/25 (128 addresses for Node 512)

No network interface associated with cluster IP
IP tables mandatory - cannot be disabled.. Lot of services depend on it..

What is a Cluster IP?

- It is an indirection to reach the actual containers providing the service.
- It hides scaling up/down, failures, restarts of containers behind the service.
- Any microservice that provides a service consumed by another party should use a Cluster IP.
- There are no network interfaces associated with the Cluster IP.
- It's totally virtual and implemented using iptables.

POD CIDR and Service subnet different. No overlapping

Service/Cluster IP Range

- The subnet reserved to allocate Cluster IPs in a cluster for various Services.
- **Non overlapping with the host network CIDR.**
- The size of the subnet depends on how many services are planned to be hosted in the cluster.
- Can be much smaller than Pod subnet.

Scope for demo exercise
TWO PODS in same namespace.. One POD → curl to another pod service without namespace..
Another POD in differen name space..

Service discovery

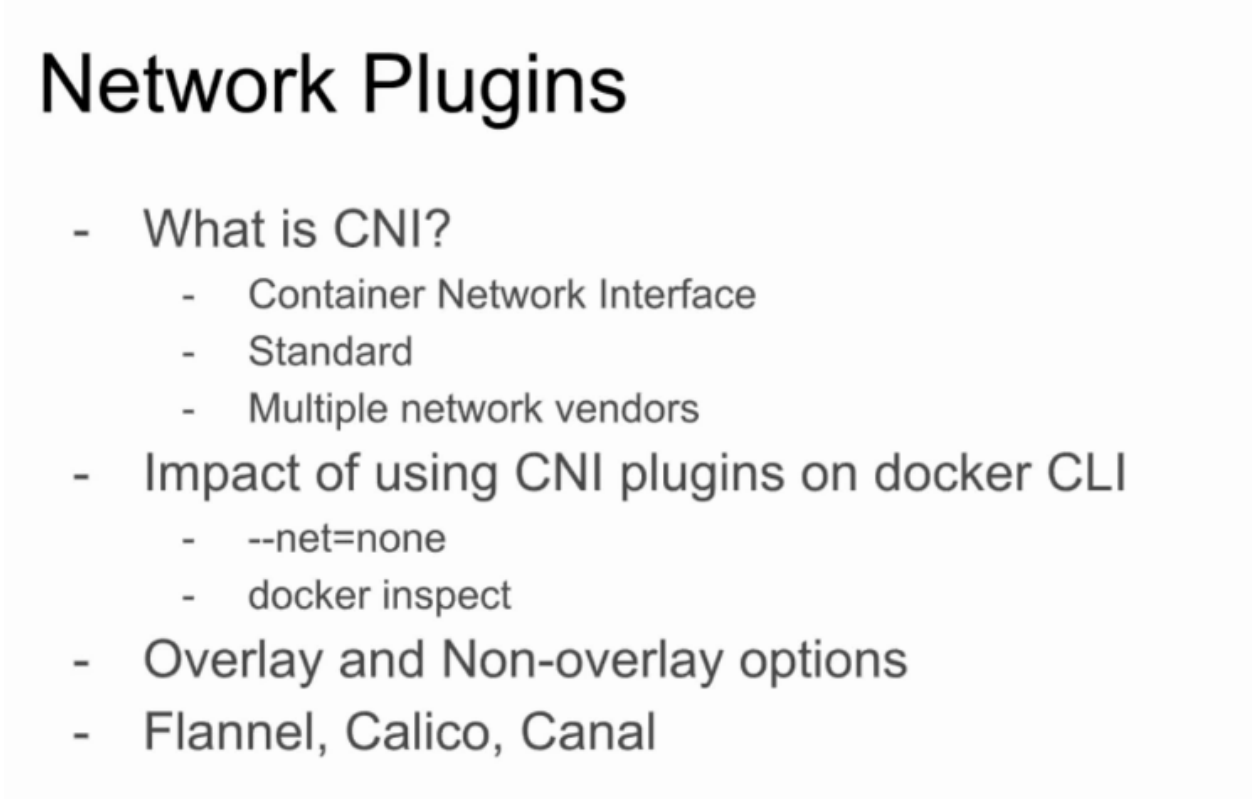
- Within the same namespace, services are discoverable using their names.
- Across different namespaces, services are discoverable using the format:
  - service-name.namespace-name
- Default domain: cluster.local
- DNS A records: service-name.namespace-name.svc.cluster.local

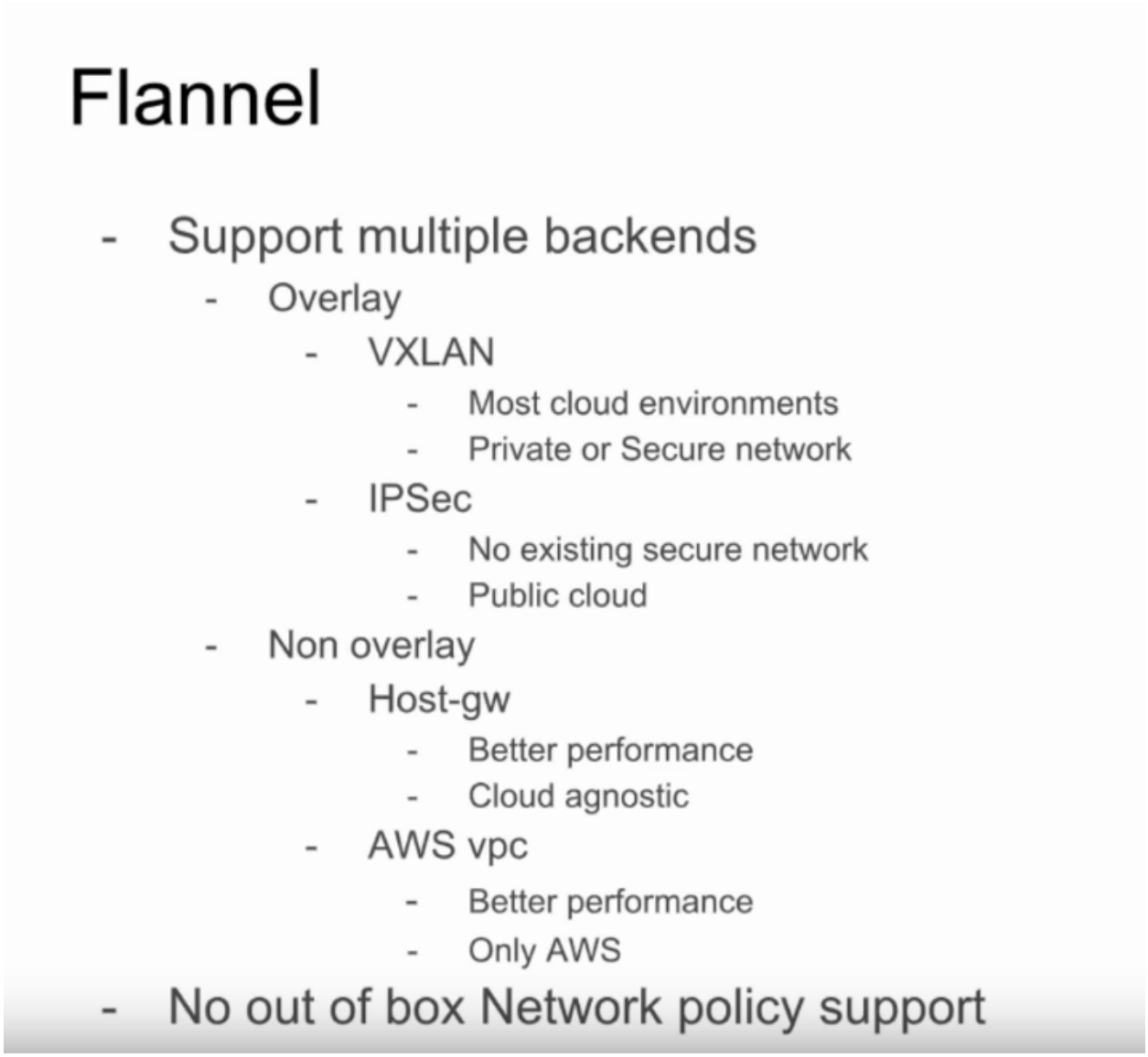
Kube/core-dns - Provides service discovery...

kube-dns

- The component responsible for providing Service Discovery functionality in a kubernetes cluster.
- The DNS pods are exposed to other pods in the cluster using DNS Service's a Cluster IP.
- The scale of the Deployment depends on the actual cluster size/load.

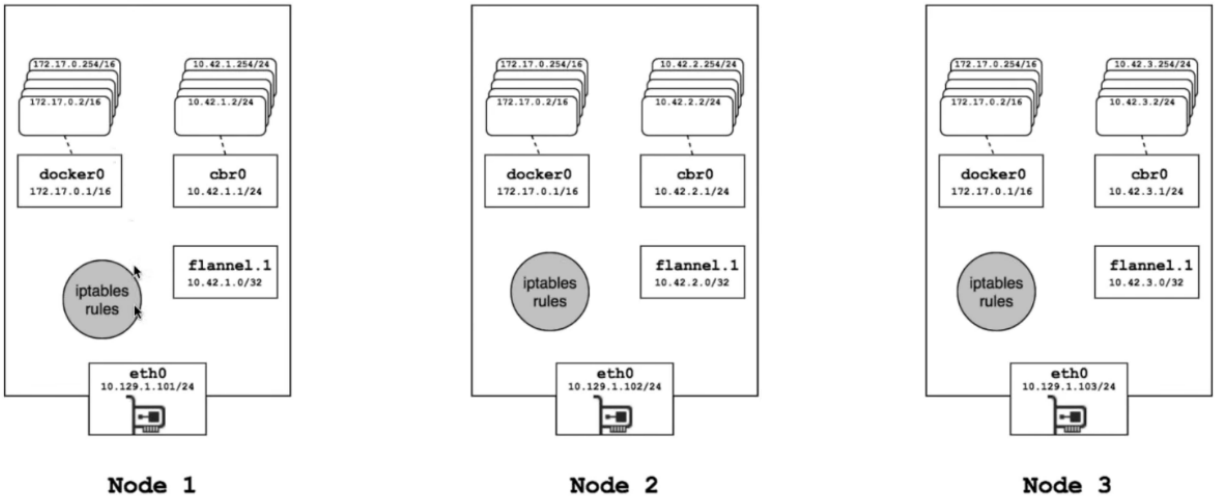
--net=none ??? ---- > Means docker networking is bpassed or not used..
If docker inspect <container ID> command is run.. Not network IP address will be shown..
But container will have an IP address assigned which can be confirmed by going to the container shell and running IP addr command...



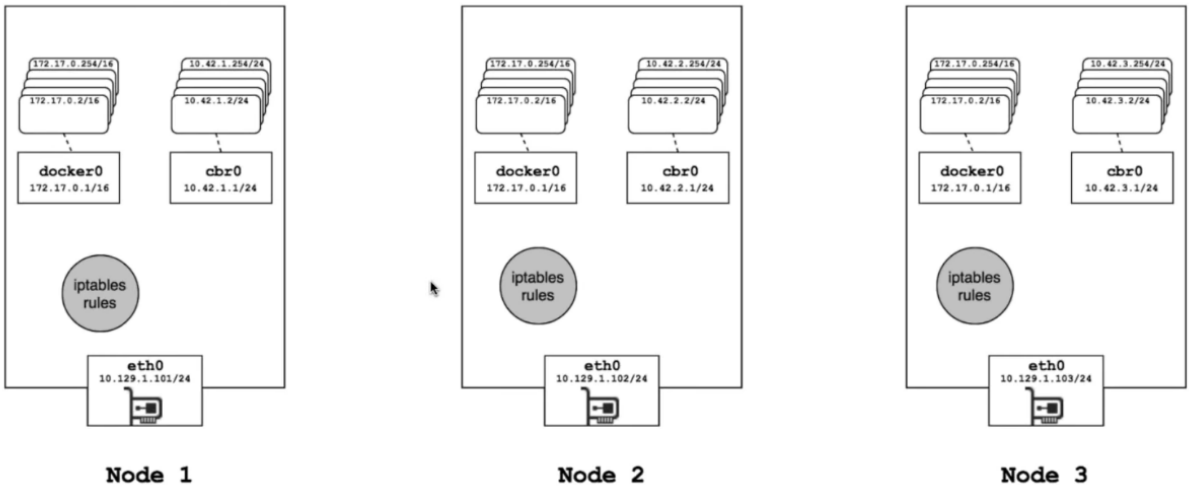


Flannel interface present - does encapsulation and decapsulation..

Flannel Architecture - VXLAN backend



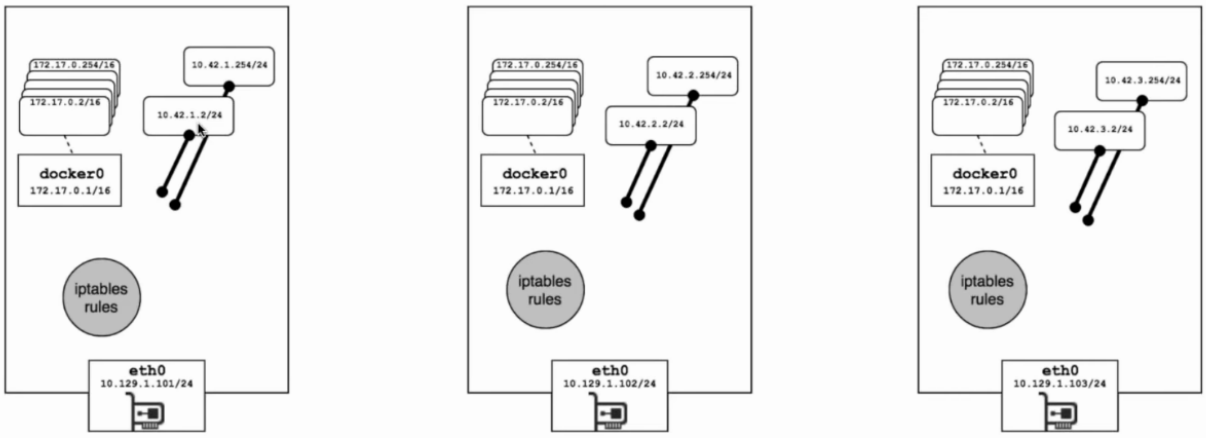
Flannel Architecture - Host Gateway backend



Calico

- Support multiple backends
  - Non overlay
    - L3
      - Better performance
      - Private datacenter
      - Direct routing to the container workloads (using BGP)
  - Overlay
    - IP-IP encapsulation
      - Most cloud environments
      - Private or Secure network
- Supports Network Policy

Calico Architecture



Host port ----> Pod IP and port..

HostPort

- The equivalent of expose port in the docker world.
- Exposes the service running inside the pod to the outside world using the desired port number on the Host IP address.
- NOT RECOMMENDED!
  - What happens if the node where the Pod is running dies? And there are no nodes available with the desired port available.