K8s N/w Overview : [Section 40]

1. K8s N/w & model.
2. CNI Plugin Concept
3. Kubernetes DNS.
4. N/w Policy & Components.
5. Ingress or Egress.
6. To and from Selector.

SECTION 10 [255. K8s N/w Overview]

K8s N/w Overview:-

1. K8s N/w
2. K8s N/w model

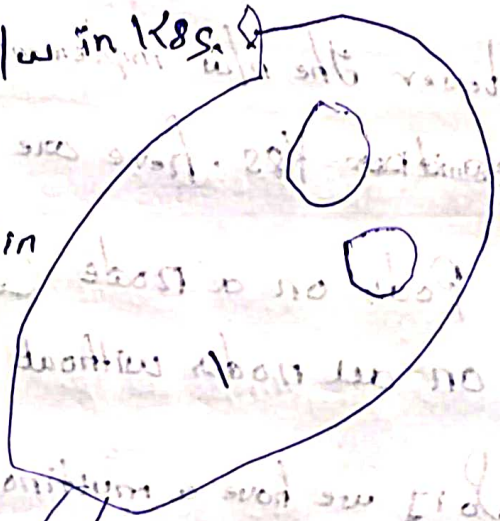
K8s N/w :-

1) Kubernetes N/w model define how networking b/w pods behave. (or) how n/w will work b/w the pods

we are very well aware that within the K8s we will Create the Containers & that Container are basically wrapped up in a single unit called pod. & within a single pod. if you want you can have the multiple Containers as well

So as we aware that pods are the basic object of K8s & n/w mode in K8s or in N/w in K8s.

Basically define that how that parts which are running within K8s cluster, they will behave & how they can Communicate.



2) In K8s many N/w. Implementation available in K8s

Whenever you will Search for the K8s you will get a K8s offical document page & you will see multiple n/w. models are available for the Kubernetes.

if you remember, Then whenever we have done the setup of Kubernetes cluster, we used the n/w called Calico.

3) So we downloaded the Calico YAML from internet & then we executed this Calico YAML.

just like the Calico.

There are multiple n/w implementations available for the K8s, but if you are using the K8s with the KubeAdm Then Calico is best Supported n/w with the KubeAdm.

4] Kubernetes imposes the following fundamental Requirements On any networking implementation -

Whatever the n/w implementation you are going to use in K8s. here are the fundamental list:

1] Pods on a node Can Communicate with all pods on all nodes without NAT.

So if we have a multinode Kubernetes cluster & any pod Want to Communicate with the Other pods.

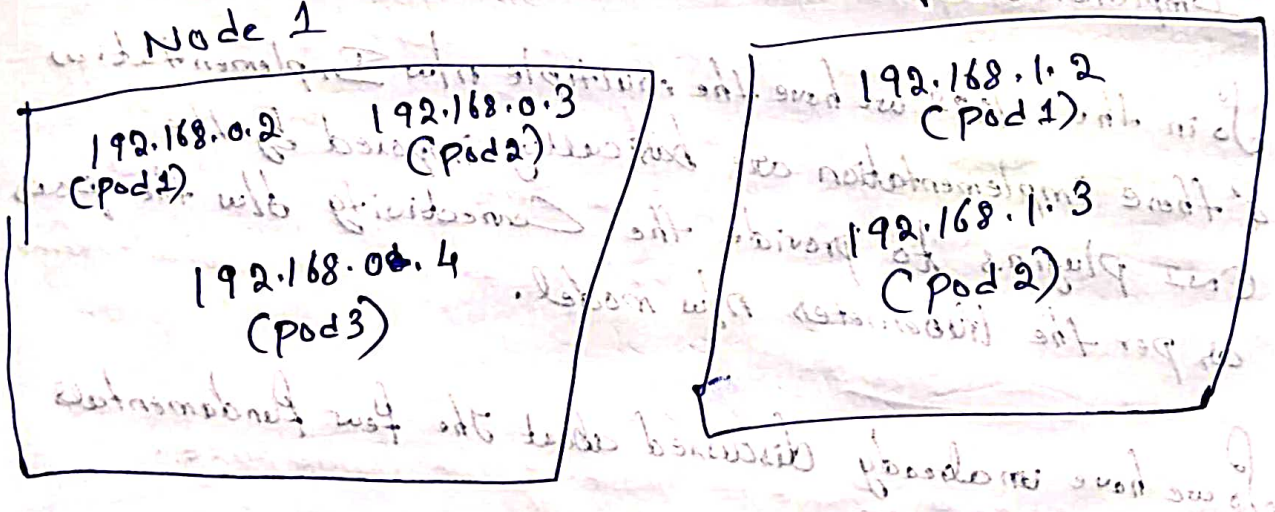
Then if the n/w allows, Then the pod Can easily Communicate to the Other pods without using the n/w address. [NAT]

2] Agents on a node (e.g. System daemons, Kubelet) Can Communicate with all pods on that node.

That is also a basic fundamental of any n/w plugin or any n/w, whatever you are going to use in K8s.

3] Every pod gets its own IP address.

So whenever we are Creating the pod, that pod must have their own IP address so that the other pod Can Communicate to that particular pod either by IP or DNS.



The pods with in node 1 Can Communicate with IP or DNS.

The pods on different nodes (node1 to node2) Can Communicate with IP address.

[256] CNI Plugins:

1. CNI plugin Concept
2. Selecting Network plugin
3. Install N/w plugin.

CNI plugin Concept :-

1. CNI plugins are K8s N/w plugins.

   CNi plugin are the Components which provide these Implementations.

   So in the K8s we have the multiple n/w Implementation & these implementation are basically imposed by the

2. CNI plugins to provide the Connectivity b/w the pods as per the Kubernetes n/w model.

   So we have already discussed about the few fundamentals of K8s n/w model.

   CNI plugin will basically provide these fundamentals & additional features as per the n/w plugin.

3. multiple CNI N/w plugins available for the K8s.

   In browser Search of K8s N/w, click on the K8s Doc. and we can see lots of plugins.

Selecting N/w plugin

① Selection of CNI plugin depends on your Business needs.
You can go through the documentation of this plugin & you can go through the your business need.
You can compare that what is the plugin which is close to your business need & you can use that.

② In our HA Setup, we used Calico n/w plugin.
Because Calico supports Kubeadm very well.

Install n/w plugin :-

① Each plugin have it's own installation way.

② In HA Setup, we installed Calico using Calico Yml file.
Go through the documentation for installing plugins.

③ Note :- K8s nodes will remain NotReady until you install N/w plugin & users won't able to Run pods.
If you are not using any n/w plugin in your Kubernetes Cluster, the nodes which are the part of your Cluster, they will be not Ready & you won't be able to execute any kind of pod Creation or any kind of object Creation in your Kubernetes Cluster.

[25] Kubernetes DNS.

1) DNS in K8s
2) Pod Domain Names
3] Hands On Demonstration

DNS in K8s :-

(1) K8s Virtual n/w uses DNS to allow PODs to locate other PODs & Services using Domain Name.

So whenever we are Creating the Kubernetes cluster, we are also Creating the n/w so that the pods can Communicate to each other & the value which are basically allowing the pods to Communicate each other or the value which allows the Communication b/w the pods & the Services that is called the domain name or the DNS record.

(2) DNS runs as a Service in Kube-System namespace:

So whenever you are using the K8s cluster, you can go to your Kube-System-namespace & you can identify if any DNS Service is running in your Kube-System-namespace or not.

(3) KubeAdm/minicube both use the Core DNS

(4) All pods in Kubeadm cluster are automatically given a Domain Name. like-

1. pod-ip.namespace-name.pod.cluster.local.

(5) Pod DNS in Default Namespace with IP: 190.168.0.20 would look like:-

190-168-0-20.default.pod.cluster.local.

Hand On Demo :-

(1) we are in Singlenode Cluster.

(2) Kubectl get pods -o wide -n kube-system.

we see that Coredns -- is Running ~~(other Kube)~~.

(Core dns pod in Running)

(3) To get the Services.

Kubectl get Services -o wide -n kube-system.

we can see that kube-dns Services are Running in ~~ng~~ namespace.

(4) Let See How pods can Communicate to each other by a DNS Record.

(a) Create a directory or Go to directory where Script or Stored.

(b) Vi pods-dns.yml.

```
apiVersion : V1
Kind : Pod
metadata :
   name: nginx-nodename
Spec :
  Containers:
    - name: nginx
      image : nginx
---

apiVersion : V1
Kind : Pod
metadata :
   name : frontend-app
Spec :
  Containers :
    - name: app
      image: alpine
      Command:
        - Sleep
        - '3600'
```

Save & Exit.

(c) Kubectl apply - f pods-DNS. yml.

we See that two pods.

(d) Kubectl get pods -o wide.

(e) Copy the IP of pod. & Execute Curl Command.

Curl 172.17.0.6

(f) Curl with dns name.

Curl 172-17-0-6.default.pod.cluster.local.

we see that we got error. Could not resolve host.

why? we are executing from host machine not from the Container (pods) we deployed.

(g) To Execute Command in Container

Kubectl exec -it <pod-name> sh.

we will look in to the pod.

Now update Apk update.

apk add curl

curl --version.

Curl 172-17-0-6.default.pod.cluster.local.

Note :- we have connected to Alpine pod & performed curl to nginx pod. & it is working.

These procedure is not only for Calico n/w. it is for all supported n/w plugins. (i mean rules to communicate via DNS)

[258] Using Kubernetes N/w policies

1. What is Network policy
2. pod Selector.
3. ingress & Egress
4. From & To Selector
5. Ports
6. Hands On Demo.

what is N/w policy :-

1. K8s n/w policies are used to Control the Traffic flow at IP address or Port level

2. N/w policy is object in K8s
   So whenever you want to Create the n/w policy. first you need to define the template. then you need to Create the n/w policy object in K8s.

3. Pods Can Communicate using three Identifiers :-
   a. Other pods that are allowed.
      So the pods which are allowed to Communicate to a Specific pod. Only these pods Can Communicate.
      if you have pod A, B, C & D. if only two pods are allowed to Communicate with pod A (ie, C & D).

Then only these two pods can communicate to pod A.

~~Then only these two pods can communicate to pod A~~

(B) Namespaces that are allowed.

you can also define the namespace & the pods which are belongs to that namespace can only communicate to the other pods.

(C) IP blocks

you can define the IP ranges within your n/w policy & on the basis of that IP range, the pods can communicate to each other.

(4) The purpose of n/w policies is to build a secure n/w by keeping the pods isolated from traffic.

(5) By default the traffic from pods are allowed from any to any. & that is also called non isolated pod.

(6) Pods become isolated by having a N/w policy that selects them.

N/w policy Components :-

(1) Pod Selector :- Determines to which Pods in Namespace the NetworkPolicy will be applied.

So within that pod Selector you will define the labels all the pods which are matching that particular labels are the Candidate of this particular n/w policy.

(2) PodSelector Can Select the pods using Labels.

(3) An empty podSelector Selects all pods in the namespace.

if you are defining the pod Selector in the n/w policy & that is empty, it means that don't have any matching labels, Then it will Contain or it will Select all the Pods within that particular Name Space. in which you are Creating the ~~Name~~ n/w policy in that

Namespace:

Same definition/manifest/template :-

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy.
metadata:
    name: sample-net-policy
    namespace: default.
Spec:
    podSelector:
    matchLabels:
        role: front-end
```

Ingress or Egress

1) Ingress: Incoming N/w traffic coming into the pod from another source.

2) Egress :- Outgoing N/w Traffic that leaving the pod for another Destination.

So n/w policy is basically applied on both kind of traffic as well.

you need to define the policy type whenever you are Creating the n/w policy & that is Completely optional.

To and from Selector :-

① fromSelector: Selects Ingress Traffic that will be allowed on pods.

② toSelector :- Selects Egress Traffic that will be allowed from pods.

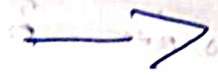
when we adding fromSelector or toSelector in manifest we are applying N/w policy on pods. (Note:- not on ports, it on pod) :-

```
...
ingress:
- from:
  - podSelector:
      matchLabels:
        role: client
...
```

NamespaceSelector:

Select namespaces to allow Traffic from/to.

```
...
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          role: client
```



instead of PodSelector, we are using namespaceSelector

IPBlock

Select an IP Range to allow Traffic from/to.

```
---
  ingress:
  --from:
  - ipBlock:
      cidr: 172.17.0.0/16
```

Ports in N/w policy :-

Ports : Specify one or more ports that allow traffic

```
---
  ingress:
  - from:
    ports:
    - protocol: TCP
      port: 80
---
```

```
---
  egress:
  - to:
    ports:
    - protocol: TCP
      port: 32000
      endPort: 32768
---
```

[259] LAB : Using K8s N/w Policies

Using HA Cluster Setup.

Why we not using minikube for this lab because minikube setup is not Compatible for the n/w policy.

policy n/w needs an agent and minicube n/w don't have that agent within it.

As we are using Calico N/w agent on Our HA setup. it work very fast.

(1) Go to the directory, where we have stored the script.

cd myscripts. (On master node)

(2) vi network-pol-pods.yml (Creating two pods):

```
apiversion: v1.
Kind: Pod
metadata:
   name: nginx-pod.
   namespace: network-policy.
   labels:
      app: frontend.
Spec:
   Containers:
      - name: nginx
      . image: nginx
---
```

apiversion : v1.

Kind : Pod

metadata :

name : busybox-pod.

Namespace : network-policy

labels :

app : client

Spec :

Containers :

- name : busybox

image : radial/busyboxplus : curl

Command :

- sleep

- '3600'

Save & close the file.

~~3) need to Create namespace~~

~~Kubectl Create namespace network-poli~~

(2) ~~Create a pods.~~

~~Kubectl apply -f network-policy-pod.yml~~

(4) ~~To check if pods are Created.~~

~~Kubectl get pod -o wide.~~

(5) ~~To check if namespace is Created.~~

Kubectl

(3) Create a namespace manually as to mentioned on manifest.

Kubectl Create namespace network-policy

(4) Check if namespace is Created.

Kubectl get namespace

(5) ~~need to create label to the namespace.~~

Check if Label are present in namespace.

Kubectl get namespace --show-labels.

we see there is no Label in namespace as we want.

(6) Create a Label into the Namespace.

Kubectl label namespace network-policy

role=test-network-policy

why are attaching Label to namespace, to send ingress/Egress traffic.

(7) Kubectl get namespace --show-label

(8) Create a pods.

Kubectl apply -f network-pol-pods.yml

(9) To check if pods are Created & Status.

Kubectl get pods -o wide -n network-policy

Note:- we have Created pods in these namespaces.
(not in default)

(10) Execute Command to check. if pods in reachable & accesible from other pod. (Second pod)

Kubectl exec -n network-policy busybox-pod
-- curl 192.168.36.195

we see that Status in Success.

Up to this we are not applied any n/w policy.
So all pods are able to Communicate (non isolated)
By default pods are able to Communicate with all pods.

11 Create manifest for network policy

Vi network-policy.yml:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: sample-network-policy
  namespace: network-policy
Spec:
  podSelector:
    matchLabels:
      app: frontend
  policyTypes:
    - Ingress:
    - Egress:
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            role: test-network-policy
      ports:
        - protocol: TCP
          port: 80
```

Save & Exit

12) Kubectl apply -f network-policy.yml :

13) Kubectl get networkpolicy -n network-policy -o wide

14) Kubectl exec -n network-policy busybox-pod -- curl 192.168.36.195