35.

Services

we've deployed a pod (inside web-app)
we want to access this web-app from our lap, for which we use "Service"
Communication b/w various components

→ Kubernetes Services enable communication and collab of the application

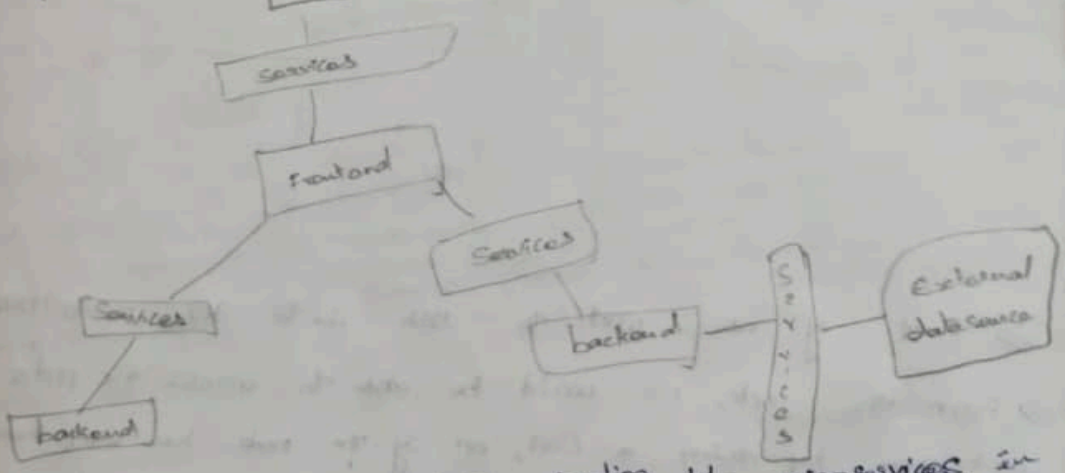
→ Kubernetes Services helps us connect applications together with other applications or users.

eg: Our application has groups of pods running various sections, such as a group for serving frontend load to users & other group for running backend processes, and a third group connecting to an external data source.

→ It is 'Services' that enable connectivity b/w these groups of pods

Services enable the front and application to be made available to end users

& helps communication b/w backend and front end pods and helps in establishing connectivity to an external data source



Thus, services enable loose coupling b/w microservices in an application.

Use Case of Services:

We're discussing about external application.

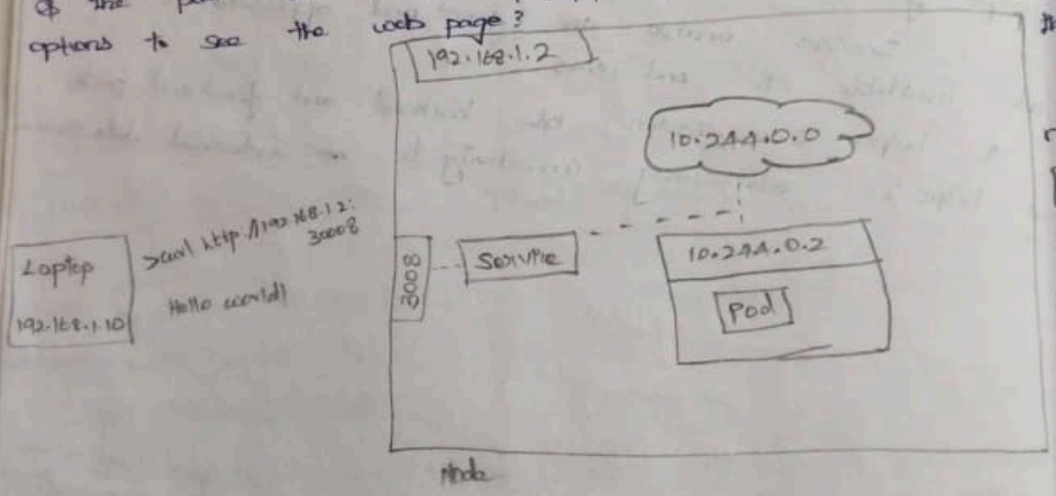
We've deployed our pod, having a web application running on it.

How do we, as an external user access the webpage.

Lets see the K8s setup

The K8s node has an IP address of that is 192.168.1.2 my laptop is on the same n/w, so it has an IP address 192.168.1.10

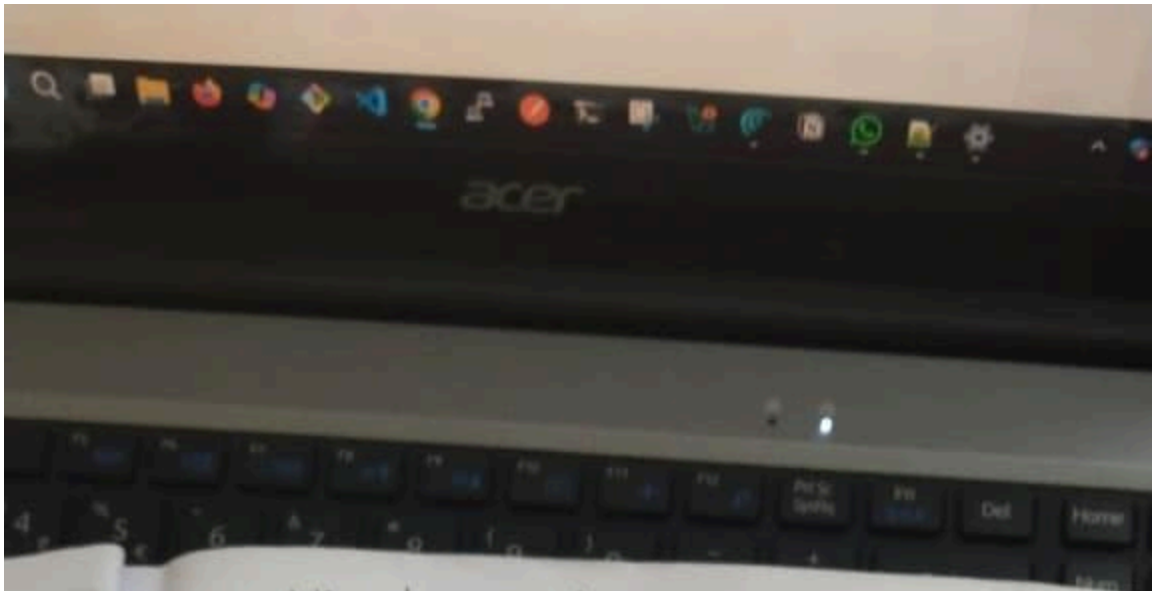
The internal pod n/w is in the range 10.244.0.0 & the pod has an IP 10.244.0.2. So what are the options to see the web page?



⇒ First of we want to ssh in to K8s node 192.168.1.2

⇒ From the node, we would be able to access the pod's web page by doing a curl, or if the node has GUI, then we would fire up a browser & see the webpage in a browser following the address http://10.244.0.2.

But this is inside K8s node, which we don't want



the expectation here is to access the web server from my own laptop without having to SSH into the node and simply by accessing the IP of the K8S node.

So we need something in the middle to help us route requests to the node from our laptop through the node to the pod running the web container.

[In other words, we need something in the middle which maps the request from our laptop to the pod through the node]

This is where the K8S service comes into play.

The K8S service is an object, just like Pod, ReplicaSet, Deployments that we worked with before.

One of the use cases is to listen to a [port on the node] & forward requests on that [port-to-a-pod] on the [pod-running-the-web-application]. This type of service is known as "node port service" because the service listens to a port on the node & forward requests to the port.

Service types

* NodePort * ClusterIP * LoadBalancer

NodePort: Listening to the node port, forwards the traffic to the port of the pod, where the web application is running.

NodePort, where the Service makes an internal port to the node.

the expectation here is to access the web server from my own laptop without having to SSH into the node and simply by accessing the IP of the K8S node.

So we need something in the middle to help us route requests from our laptop through the node to the port running the web container.

[In other words, we need something in the middle which maps the request from our laptop to the port through the node]

This is where the K8S service comes into play.

The K8S service is an object, just like Pod, ReplicaSet, Deployments that we worked with before.

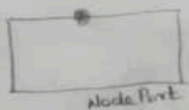
One of the use cases is to listen to a [port on the node] & forward requests on that [port-to-a-pod] on the [port-running-the-web-application]. This type of service is known as "node port service" because the service listens to a port on the node & forward requests to the port.

Service types

* NodePort * ClusterIP * LoadBalancer

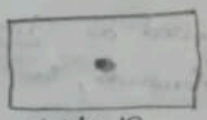
NodePort:- Listening to the node port, forwards the traffic to the port of the pod, where the web application is running.

NodePort, where the service makes an internal port accessible to a port on the node.



ClusterIP:-

In this case, the service creates a virtual IP inside the cluster to enable communication b/w different services, such as a set of frontend servers to a set of backend servers



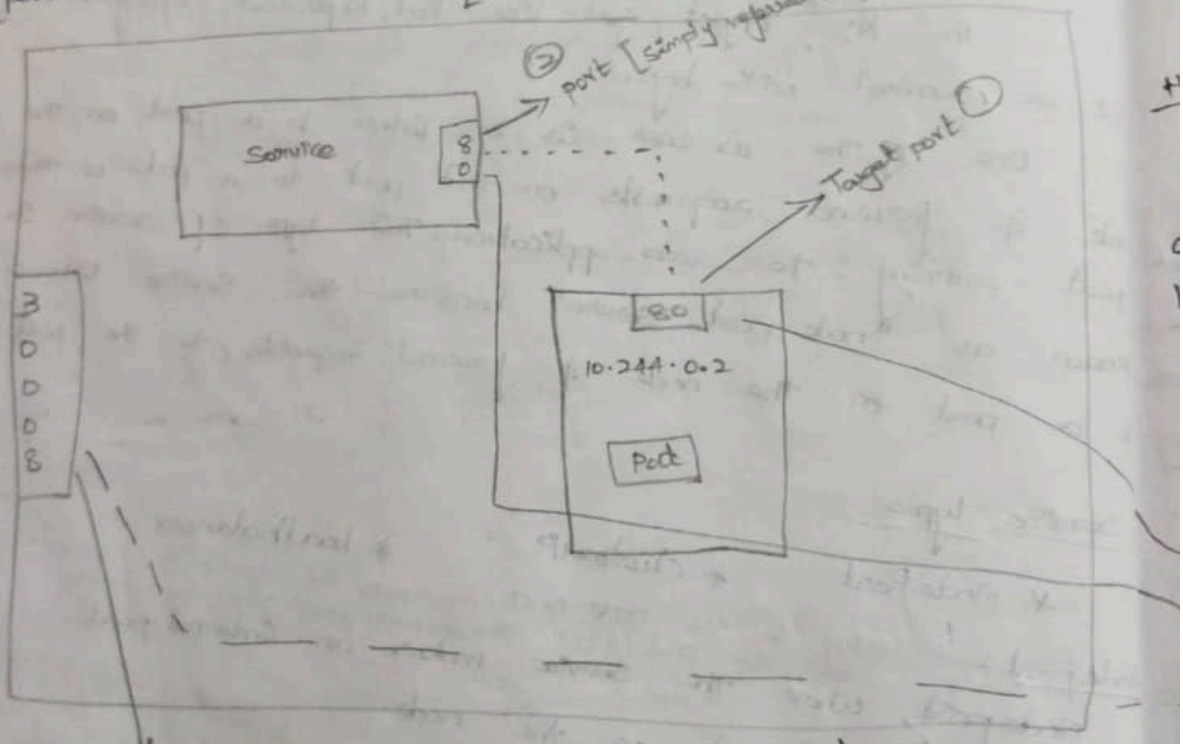
Load balancer:- it is for load distribution

* It provisions a load balancer for our application in supported cloud providers

A good example of that would be to distribute load across the different web servers in your frontend tier

Service - Node Port

We've said that a service can help us by mapping a [port-on-the-node] to a [port-on-the-pod]



3 Node port
[The node itself has a port, called node port]

These are 3 Ports available

The port on the pod, where the actual web server is running [Port: 80], it is referred as "target port" because that is where the service forwards, the request

The request to the 3rd port is the [port on the service] itself. This is referred as "port" itself.

The service is infact like a virtual server inside the node inside the cluster, it has its own IP address & that IP address is called the **cluster IP** of the service.

And finally we have the port on the node itself, which we use to access the web server externally & that is known as the node port. [nodeport - 30008]

node port default valid range 30000 to 32767

How to Create a Service

Service-definition.yml

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  type: NodePort   #type refers to the type of service we are creating
                   Type we have used Nodeport, it could be also clusterIP, nodeport, loadbalancer
  ports:
  - targetPort: 80      # Pod Port
    port: 80            # Service Port
    nodePort: 30008
  selector:
    app: myapp
```

Under "ports", the only mandatory field is "port"

⇒ If we dont provide a target port, it is assumed to be the same as port, and if you don't provide a node known port, a free port in the valid range b/w 30000 to 32,767 is automatically allocated

NOTE:

Port is an array, so note the dash under the port section that indicates the first element in the array, we can have multiple such port mappings within single service

Something missing here.

There is nothing here in the definition file that connects the Services to the pod

⇒ We've simply specified the target port, but we didn't mention the target port on which pod. There could be many other pods with webservices running on port 80, So how do we do that

⇒ As we did with the replica sets previously and a technique that you will see very often in K8S.

⇒ We will use **labels and selectors** to link these together

⇒ We know that the pod was created with a label

⇒ We need to bring that label into Service-definition file

In the before page Service-definition.yml file

Under 'spec' section, we have to add a new property called "selector", just like in "replicaset" and "deployment definition files", under the selector provide a list of labels to identify a pod

For this refer to the pod definition file used to create the pod

Pull the labels from the pod-definition-file and place it under the selector section. This links the service to the pod

```
Selector:
  app: myapp
  type: front-end
```

Once done, Create the Service using "kubectl"

```
>kubectl create -f service-definition.yml
```

Service "myapp-service" created

```
>kubectl get services
```

Stop here

pod-definition.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: front-end
spec:
  containers:
  - name: nginx-container
    image: nginx
```

Continue here

```
>kubectl get services

NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes    ClusterIP   10.96.0.1       <none>        443/TCP        16d
myapp-service NodePort    10.106.127.123  <none>        80:30008/TCP   5m
```

```
>curl http://192.168.1.2:30008
```
(node IP, node port)

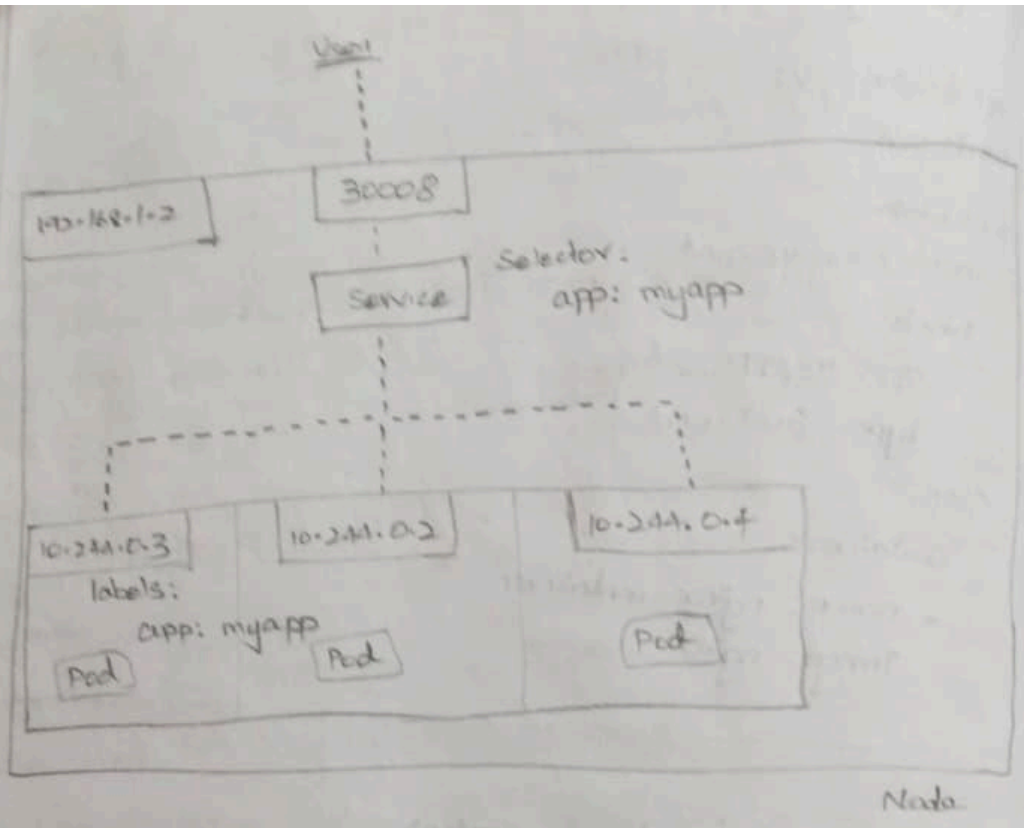
So far what we've discussed was a service mapped to a single pod but that's not the case all the time, what you do, when you have multiple pods

In production environment, there are multiple instances of web application running [reason: HA, load balancing]

Here, we've multiple similar pods running our web applicati...

They all have the same label with the key/value app: myapp

⇒ The same label is used as a selector during the creation of service.



⇒ So, when the service is created, it looks for a matching pod with the label and finds three of them.

The service then automatically selects all the 3 pods as endpoints to forward the external request coming from the user

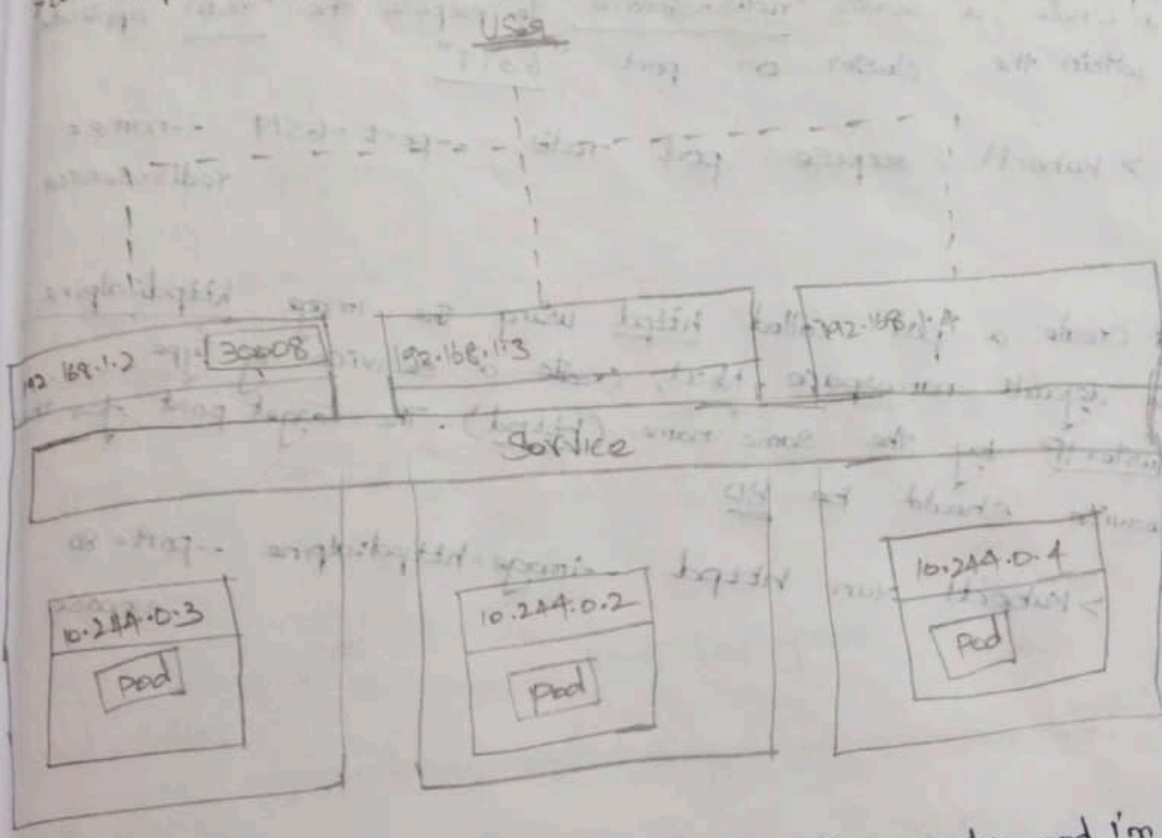
Now, finally let's look at what happens when the pods are distributed across multiple nodes

In this case, we've the web application on pod on separate nodes in the cluster

When we create a service without us having to do any additional configuration, k8s automatically creates a service that spans across all the nodes in the cluster and maps the target port to the same node port on all the nodes in the cluster this way you can access your application using the IP of any node in the cluster & using the same port number, which in this case is 30008 > curl http://192.168.1.2:30008

> curl http://192.168.1.3:30008

curl https://192.168.1.4:30008



P:24

As you can see, using the IP of any of these nodes and I'm trying to curl to the same port and the same port made available on all the nodes part of the cluster

To Summarize :-

In any case, whether it be a single port on a single node, multiple ports on a single node or multiple pods on multiple nodes, the service is created exactly the same without you having to do any additional steps during the service creation.

(*) > When pods are removed or added, the service is automatically updated, making it highly flexible and adaptive

> Once created, you don't typically have to make any additional configuration changes.