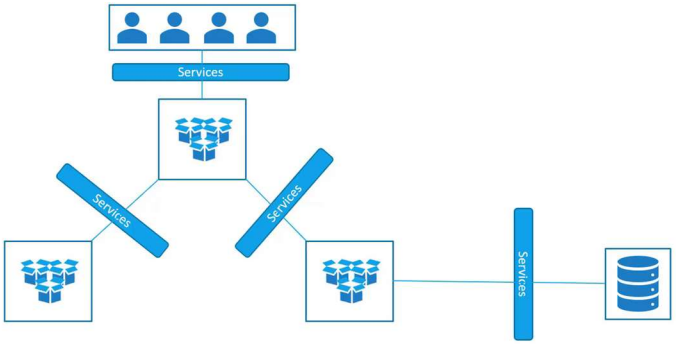
Services

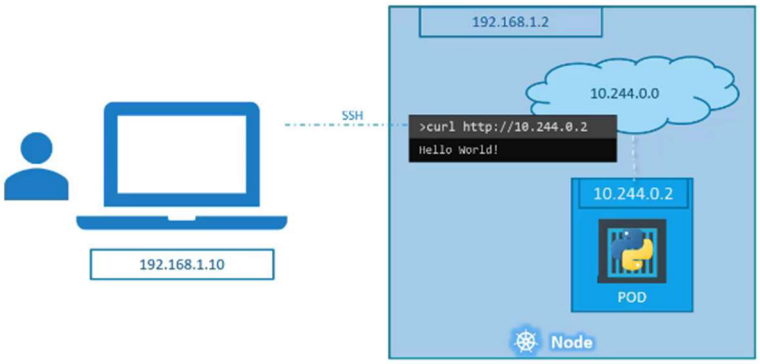
NodePort service

Kubernetes Services enable communication between various components within and outside of the application. Kubernetes Services helps us connect applications together with other applications or users. For example, our application has groups of Pods running various sections, such as a group for serving frontend loads to users, and other group for running backend processes, and a third group connecting to an external data source. It is Services that enable connectivity between these groups of Pods. Services enable the frontend application to be made available to end users. It helps communication between backend and frontend Pods, and helps in establishing connectivity to an external data source. Thus Services enable loose coupling between microservices in our application.



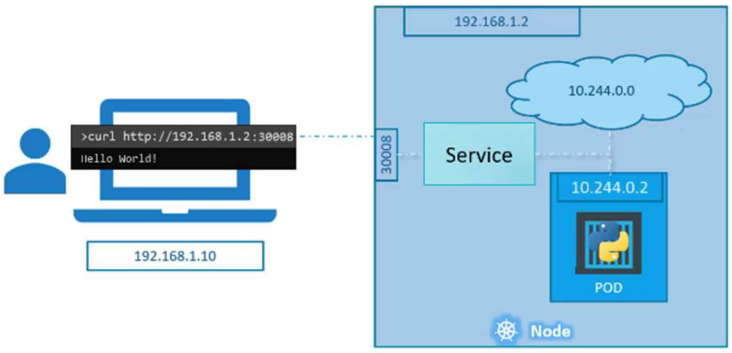
Let's take a look at one use case of Services. So far we talked about how Pods communicate with each other through internal networking. Let's look at some other aspects of networking in this lecture. Let's start with external communication. So we deployed our Pod having a web application running on it. How do we, as an external user, access the webpage? First of all, let's look at the existing setup. The Kubernetes node has an IP address and that is one 192.168.1.2. My laptop is on the same network as well, so it has an IP address, 192.168.1.10. The internal Pod network is in the range 10.244.0.0. And the Pod has an IP 10.244.0.2. Clearly I cannot ping or access the Pod at address 10.244.0.2 as it's in a separate network.

So what are the options to see the webpage? First, if we were to SSH into the Kubernetes node at 192.168.1.2 from the node, we would be able to access the Pod's webpage by doing a curl. Or if the node has a GUI we would fire up a browser and see the webpage in a browser following the address http://10.244.0.2. But this is from inside the Kubernetes node and that's not what I really want.



I want to be able to access the web server from my own laptop without having to SSH into the node and simply by accessing the IP of the Kubernetes node. So we need something in the middle to help us map requests to the node from our laptop, through the node, to the Pod running the web container.

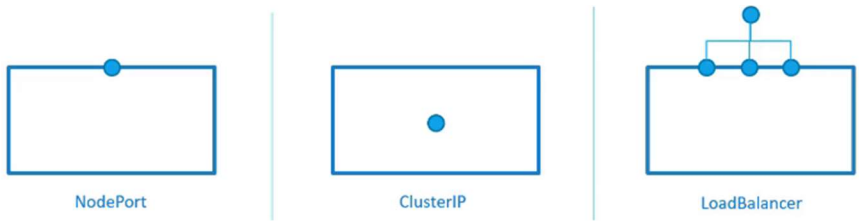
This is where the Kubernetes Service comes into play. The Kubernetes Service is an object, just like Pods, ReplicaSet, or Deployments that we worked with before. One of its use case is to listen to a port on the node and forward request on that port to a port on the Pod running the web application. This type of service is known as a **NodePort service** because the service listens to a port on the node and forward request to the Pods.



There are other kinds of services available, which we will now discuss. The first one is what we discussed already, NodePort where the service makes an internal port accessible on a port on the node.

The second is CluserIP, and in this case, the service creates a virtual IP inside the luster to enable communication between different services, such as a set of frontend servers to a set of backend servers.

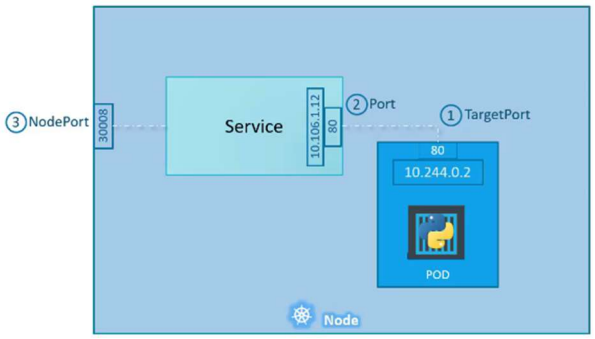
The third type is a LoadBalancer, where it provisions a load balancer for our application in supported cloud providers. A good example of that would be to distribute load across the different web servers in your frontend tier. We will now look at each of these in a bit more detail along with some demos.



In this lecture, we will discuss about the NodePort Kubernetes service. Getting back to NodePort, few slides back, we discussed about external access to the application.

We said that a service can help us by mapping a port on the node to a port on the Pod. Let's take a closer look at the service. If you look at it, there are three ports involved. The port on the Pod where the actual web server is running is 80, and it is referred to as the target port because that is where the service forwards their request to. The second port is the port on the service itself. It is simply referred to as the port. Remember, these terms are from the viewpoint of the service. The service is, in fact, like a virtual server inside the node. Inside the cluster it has its own IP address, and that IP address is called the ClusterIP of the service.

And finally, we have the port on the node itself which we use to access the web server externally, and that is as the node port. As you can see, it is set to 30,008. That is because node ports can only be in a valid range which by default is from 30,000 to 32,767.



Let's now look at how to create the service.

Just like how we created a deployment ReplicaSet or Pod in the past, we will use a definition file to create a service.

The high-level structure of the file remains the same. As before, we have the API version, kind, metadata, and spec sections. The API version is going to be v1. The kind is of course service. The metadata will have a name, and that will be the name of the service. It can have labels, but we don't need that for now.

Next, we have spec, and as always, this is the most crucial part of the file, as this is where we will be defining the actual services, and this is the part of a definition file that differs between different objects.

In the spec section of a service, we have type and ports. The type refers to the type of service we are creating. As discussed before, it could be ClusterIP, NodePort, or LoadBalancer. In this case, since we are creating a NodePort we will set it as NodePort. The next part of a spec is ports. This is where we input information regarding what we discussed on the left side of the screen.

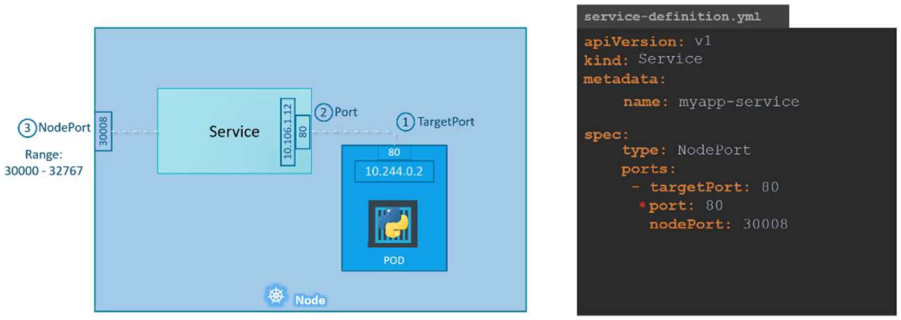
The first type of port is the targetPort, which we will set to 80. The next one is simply port, which is a port on the service object, and we will set that to 80 as well. The third is nodePort, which we will set to 30,008 or any number in the valid range.

Remember, that out of these, the only mandatory field is port. If you don't provide a targetPort, it is assumed to be the same as port. And if you don't provide a nodePort, a free port in the valid range between 30,00 and 32,767 is automatically allocated.

Also, note that ports is an array, so note the dash under the port section that indicate the first element in the array. You can have multiple such port mappings within a single service.

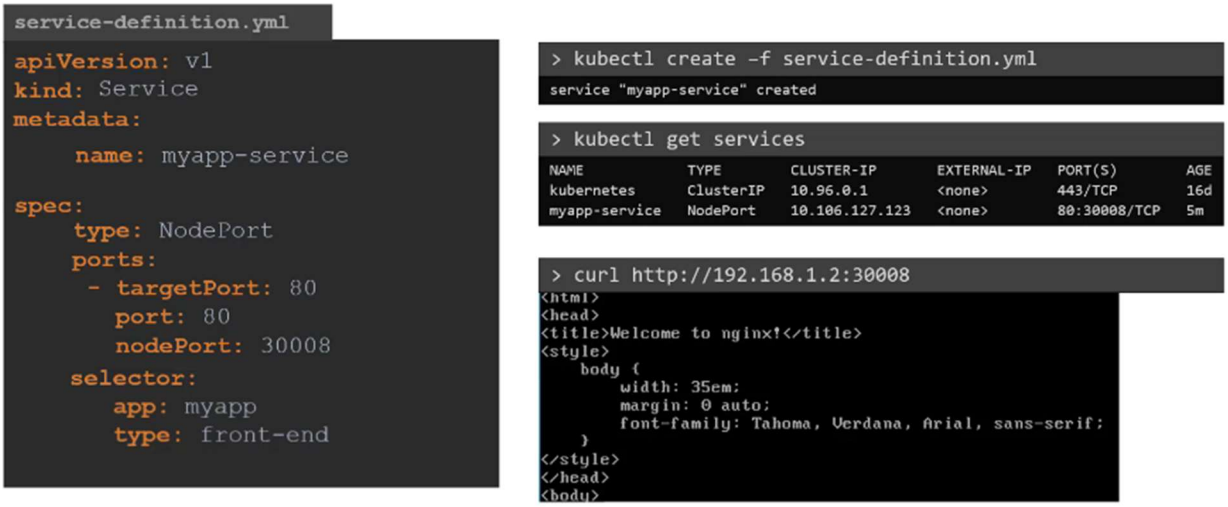
So, we have all the information in, but something is really missing. There is nothing here in the definition file that connects the service to the Pod. We have simply specified the target port but we didn't mention the target port on which Pod. There could be hundreds of other Pods with web services running on port 80. So how do we do that? As we did with the ReplicaSets previously, and a technique that you will see very often in Kubernetes, 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 the service definition file.

Service - NodePort



So, we have a new property in the specs section and that is called selector, just like in a ReplicaSet and deployment definition files. Under the selector provide a list of labels to identify the 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.

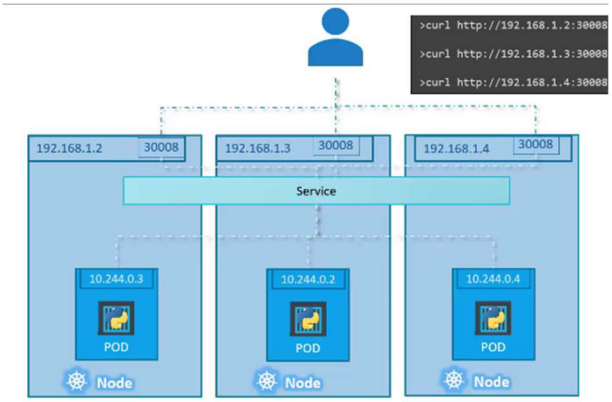
Once done, create the service using the kubectl create command, and input the service definition file. And there you have the service created. To see the created service run the kubectl get services command that lists the service, the ClusterIP, and the map ports. The type is nodePort as we created and the port on the node is set to 30,008, because that's the port that we specified in the definition file. We can now use this port to access the web service using curl or a web browser. So curl to 192.168.1.2, which is the IP of the node. And then I use the port 30,008 to access the web server.



So far we talked about a service mapped to a single Pod. But that's not the case all the time. What do you do when you have multiple Pods? In a production environment, you have multiple instances of your web application running for high availability and load balancing purposes. In this case, we have multiple similar Pods running our web application. They all have the same labels with a key app and set to a value of my app. The same label is used as a selector during the creation of the 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 three Pods as endpoints to forward the external request coming from the user. You don't have to do any additional configuration to make this happen.

And if you're wondering what algorithm it uses to balance the load across the three different Pods, it uses a random algorithm. Thus, the service acts as a built-in load balancer to distribute load across different Pods.

And finally, let's look at what happens when the Pods are distributed across multiple nodes. In this case, we have the web application on Pods on separate nodes in the cluster. When we create a service, without us having to do any additional configuration Kubernetes 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 and using the same port number which in this case is 30,008. 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 is made available on all the nodes part of the cluster.



To summarize, in any case, whether it be a single Pod on a single node, multiple Pods 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 won't typically have to make any additional configuration changes.

ClusterIP service

In this lecture we will discuss about the Kubernetes service cluster IP. A full stack web application typically has different kinds of pods hosting different parts of an application.

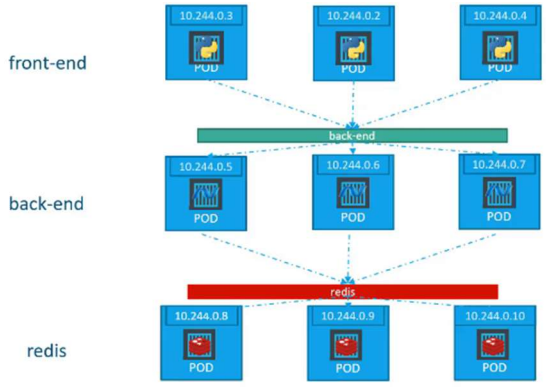
You may have a number of pods running a front end web server, another set of pods running a backend server, a set of pods running a key value store like Redis and another set of pods may be running a persistent database like MySQL. The web front end server needs to communicate to the backend servers, and the backend servers need to communicate to the database as well as the Redis services, et cetera

So what is the right way to establish connectivity between these services or tiers of my application? The pods all have an IP address assigned to them, as we can see on the screen, but these IPs, as we know, are not static. These pods can go down anytime and new pods are created all the time and so you cannot rely on these IP addresses for internal communication between the application.

Also, what if the first front end pod at 102440.03 need to connect to a backend service? Which, which of the three would it go to and who makes that decision?

A Kubernetes service can help us group the pods together and provide a single interface to access the pods in a group. For example, a service created for the backend pods will help group all the backend pods together and provide a single interface for other pods to access this service. The requests are forwarded to one of the pods under the service randomly. Similarly, create additional services for Redis and allow the backend pods to access the Redis systems through the service. This enables us to easily and effectively deploy a microservices based application on Kubernetes cluster.

Each layer can now scale or move as required without impacting communication between the various services. Each service gets an IP and name assigned to it inside the cluster, and that is the name that should be used by other pods to access the service. This type of service is known as cluster IP.



To create such a service, as always, use a definition file in the service definition file first use the default template, which has API version, kind, metadata, and spec. The API version is V1, kind is service, and we will give a name to our service. We will call it backend.

Under specification, we have type and ports. The type is cluster IP. In fact, cluster IP is the default type so even if you didn't specify it it will automatically assume the type to be cluster IP. Under ports, we have a target port and port. The target port is the port where the backend is exposed which in this case is 80, and the port is where the service is exposed, which is 80 as well. To link the service to a set of ports we use selector we will refer to the pod definition file and copy the labels from it and move it under selector, and that should be it.

We can now create the service using the Cube Control Create command, and then check its status using the Cube Control Get Services command. The service can be accessed by other pods using the cluster IP or the service name.

service-definition.yml

```yaml
apiVersion: v1
kind: Service
metadata:
    name: back-end

spec:
    type: ClusterIP
    ports:
     - targetPort: 80
       port: 80

    selector:
       app: myapp
       type: back-end
```

```
> kubectl create -f service-definition.yml
service "back-end" created
```

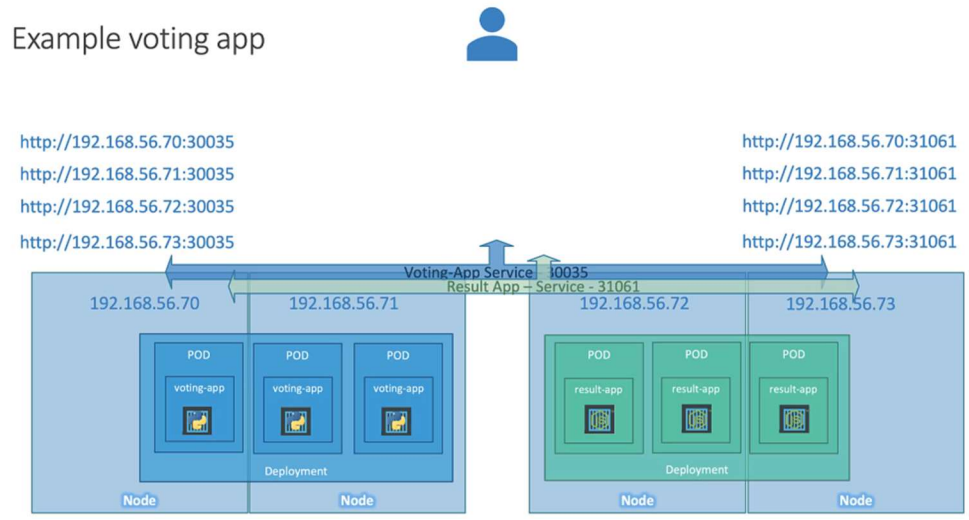
```
> kubectl get services
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP    16d
back-end     ClusterIP   10.106.127.123  <none>        80/TCP     2m
```

LoadBalancer service

Let us now look at another type of service, known as the load balancer type. So we have seen the node port service that helps us make an external facing application available on a port, on the worker nodes. So let's turn our focus to the front end applications. which are the voting app and the result app.

Now we know that these pods are hosted on the worker nodes in a cluster. So let's say we have a four node cluster. And to make the applications accessible to external users, we create these services of type node port. Now the services with type node port help in receiving traffic on the ports on the nodes and routing the traffic to the respective pods.

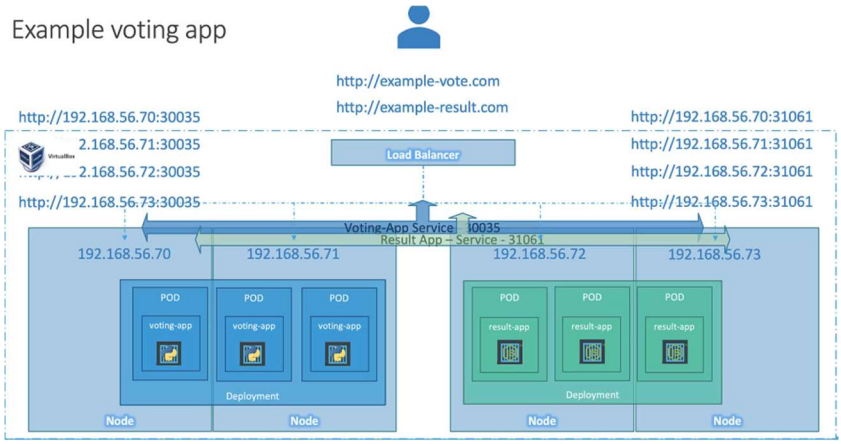
But what URL would you give your end users to access the applications? You could access any of these two applications using IP of any of the nodes and the high port with the services exposed on. So that would be four IP and port combinations for the voting app and four IP and port combinations for the result app. So note that, even if your pods are only hosted on two of the nodes, they will still be accessible on the IP's of all the nodes in the cluster. Say the pods for the voting app are only deployed on the nodes with IP 70 and 71. They would still be accessible on the ports of all the nodes in the cluster. So that's how a service is, is configured.



So you would share these URLs with your users to access the application. But that's not what the end users want. They need a single URL, like example, votingapp.com or the example, resultapp.com, to access the application.

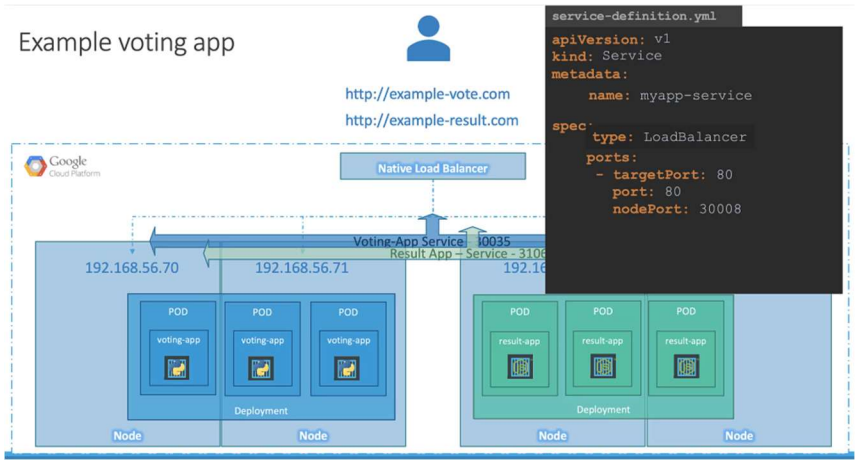
So how do you achieve that? One way to achieve this is to create a new VM for load balancer purposes. Install and configure a suitable load balancer on it like AJ/Proxy or nginx, etc. Then configure the load balancer to route traffic to the underlying nodes.

Setting up all of that external load balancing, and then maintaining and managing that can be a tedious task.



However, if we were on a supported cloud platform, like Google Cloud or AWS or Azure, I could leverage the native load balancer off that cloud platform. Kubernetes has support for integrating with the native load balancers of certain cloud providers in configuring that for us.

So all you need to do is set the service type for the front-end services to load balancer instead of node port. Now remember that this only works with supported cloud platforms. like GCP, AWS, and Azure



So if you set the type of service to load balancer in an unsupported environment, like VirtualBox, or any other environments, then it would have the same effect as setting it to node port. There, the services are exposed on a high-end port on the nodes. It just won't do any kind of external load balancing or configuration.