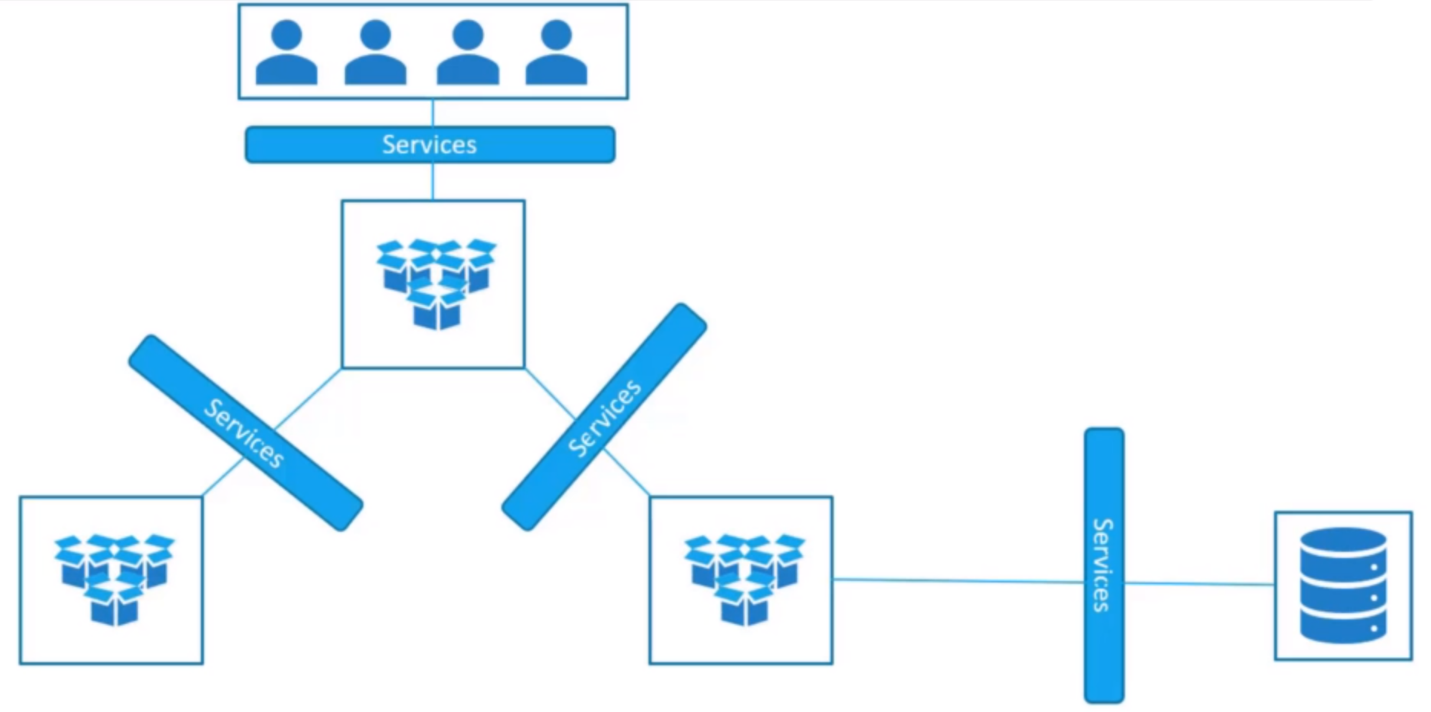
Kubernetes Services enable communication between various components within and outside of the application. Kubernetes Services helps us connect applications together with other applications or users. For example, our application has groups of pods running various sections, such as a group for serving a front-end load to users and other group for running back-end 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 front-end application to be made available to end users. It helps communication between back-end and front-end pods and helps in establishing connectivity to an external data source. Thus, services enable loose coupling between micro services in our application.

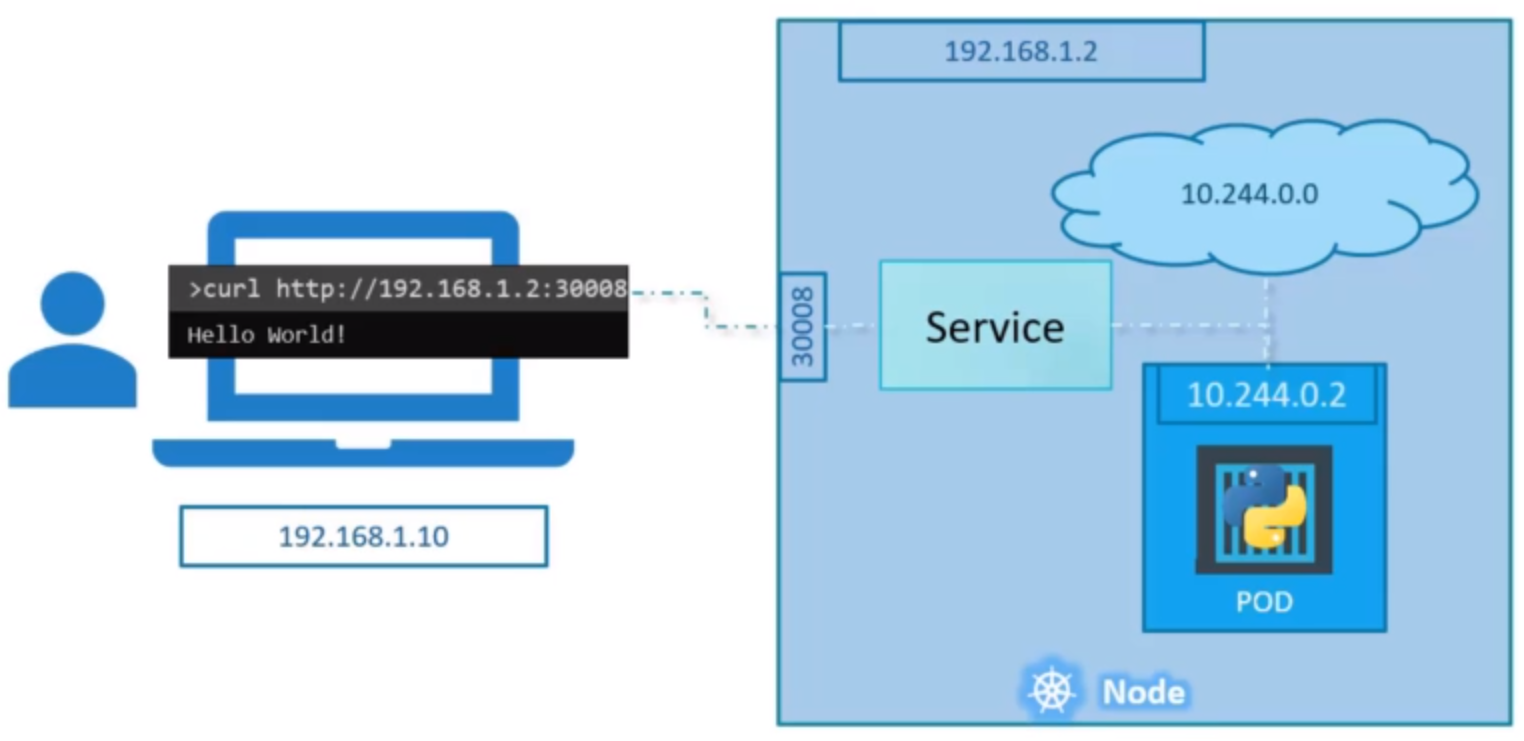


Let’s take a look at one use case of services. So far, we talked about how pods communicate with each other through internal networking, let’s look at some other aspects of networking. Let’s start with external communication. So, we deployed our pod having a web application running on it. How do we, as an external user access the web page?

First of all, let’s look at the existing set up. The Kubernetes node has an IP address 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 web page?

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, what if you want to access the web server from your own laptop without having to switch 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, replica sets or deployments that we worked with before. One of its use cases is to listen to a pod on the node and forward requests on that pod to a port 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 requests to the ports. There are different types of services available:

1. **NodePort:** Where the service makes an internal port accessible on a pod on the node.
2. **ClusterIP:** The service creates a virtual IP inside the cluster to enable communication between different services, such as a set of frontend servers to a set of backend servers.
3. **LoadBalancer:** It provisions a LoadBalancer 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.



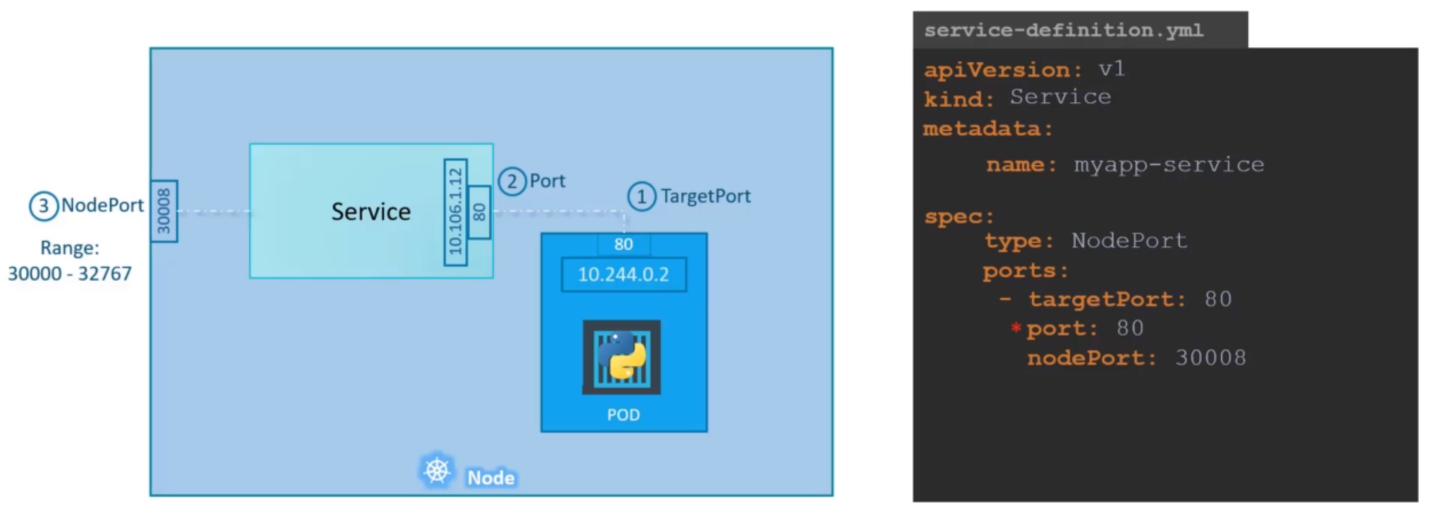
In the NodePort, lets take a closer look at the Service. In Service, there are three ports involved. The port on the pod where the actual web server is running is 80. And it is referred to as the target port because that is where the service forwards the request.

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 known as the NodePort and it is set to 30008 that is because NodePort can only be in a valid range, which by default is from 30000 to 32767.

How to create a Service?

Just like how we created a deployment, replica set 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.





Out of the above information, port 80 is only mandatory. If you don’t provide the Target Port it is assumed to be same as of port and for the NodePort it is assigned any free port between the valid range of 30000 to 32767. The tag ports is an array, the dash (-) under the ports section indicates the first element in the array. You can have multiple such port mappings within a single service.

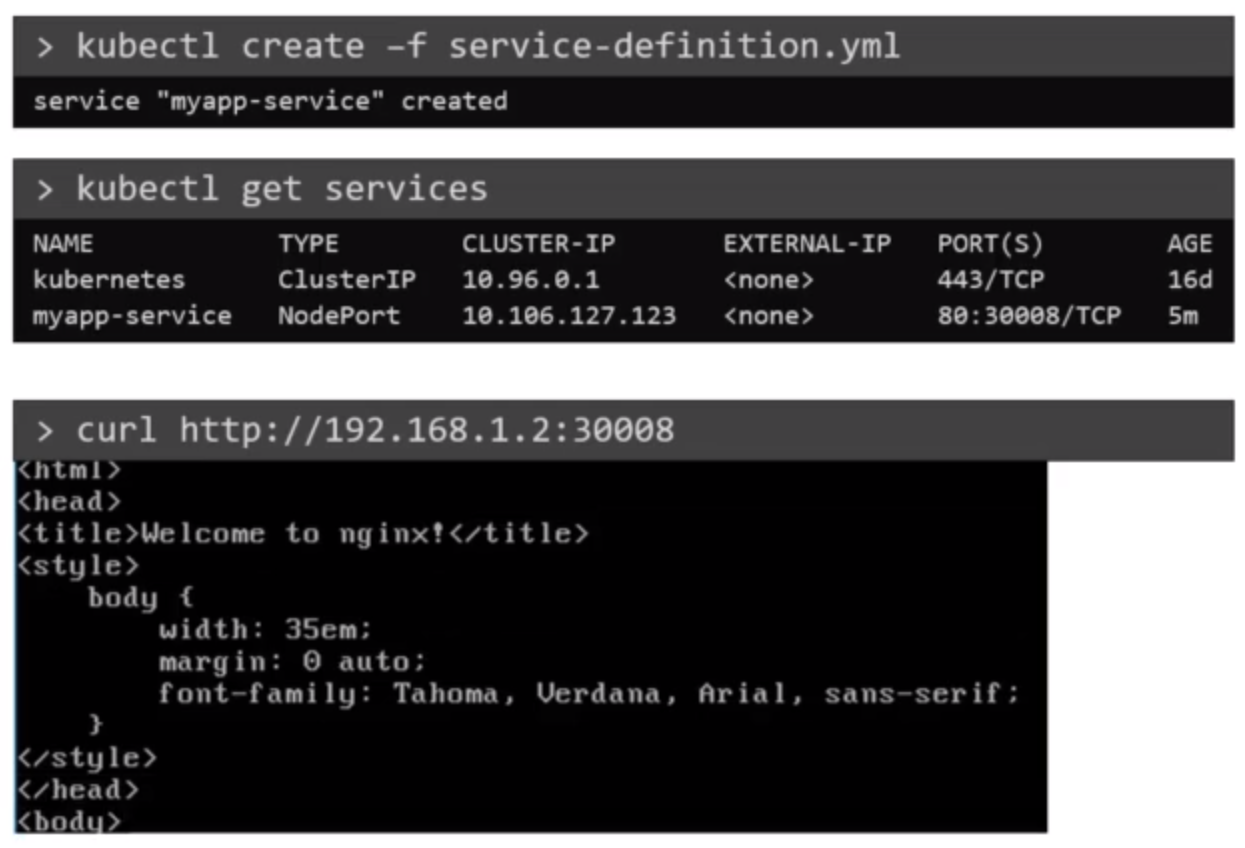
To connect the service to the pod, we have to use labels and selectors to link these together. We know that the pod was created with a label, we need to bring that label into the service definition file. For that we will use a new property in the spec section and that is called Selector, just like in replica set and deployment definition files, under the selector provide a list of labels to identify 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.





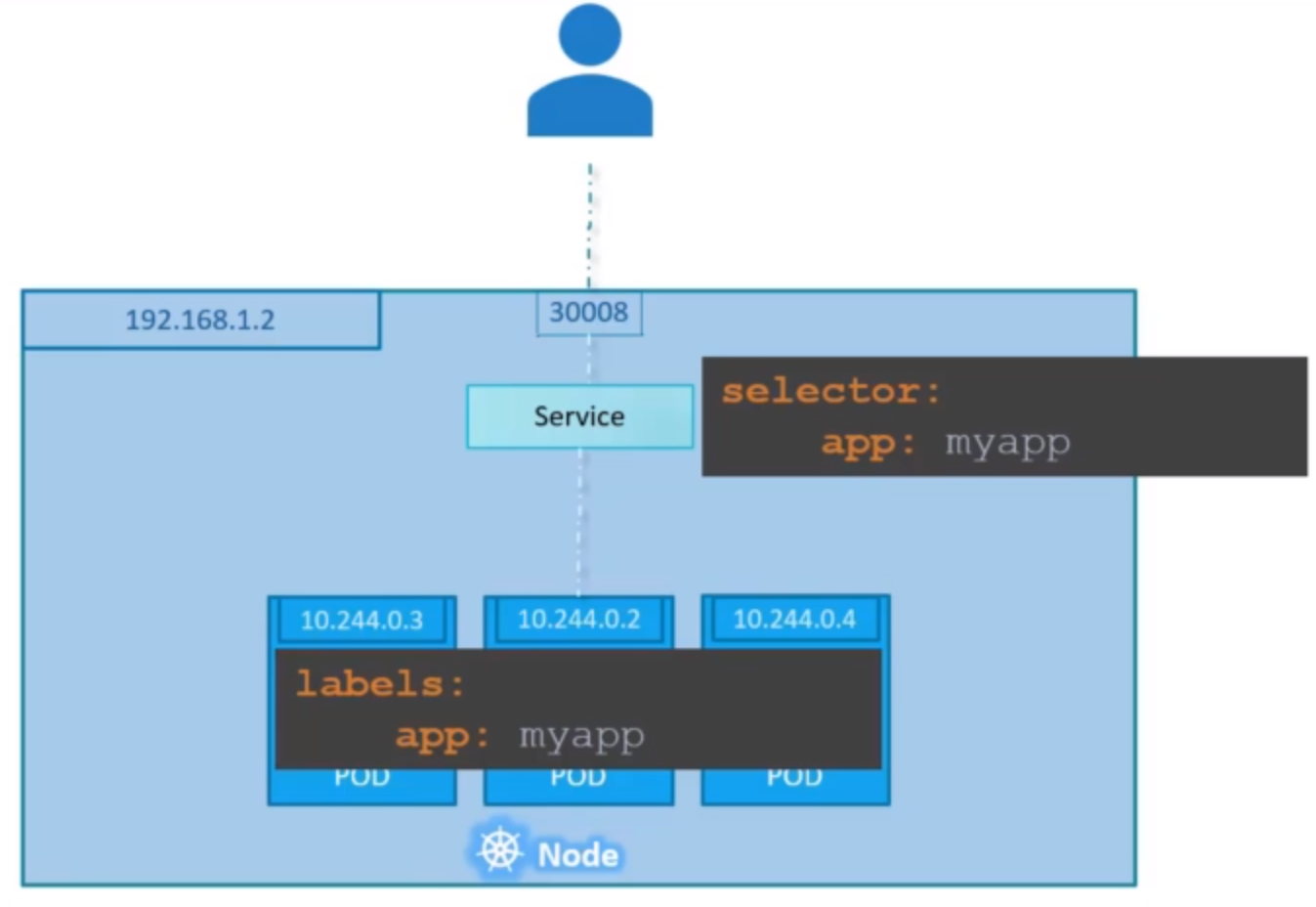
To see the Service created run the Kubectl get services command that lists the services, the ClusterIP and the mapped port.



So far, we have 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 the 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 myapp. The same label is used as a selector during the creation of the Service.

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.



Finally, lets look at what happens when the pods are distributed across multiple nodes. In this case, we have the web application on pod on a separate node 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 NodePort 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 30008. As you can see, using the IP of any of these nodes, 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.

