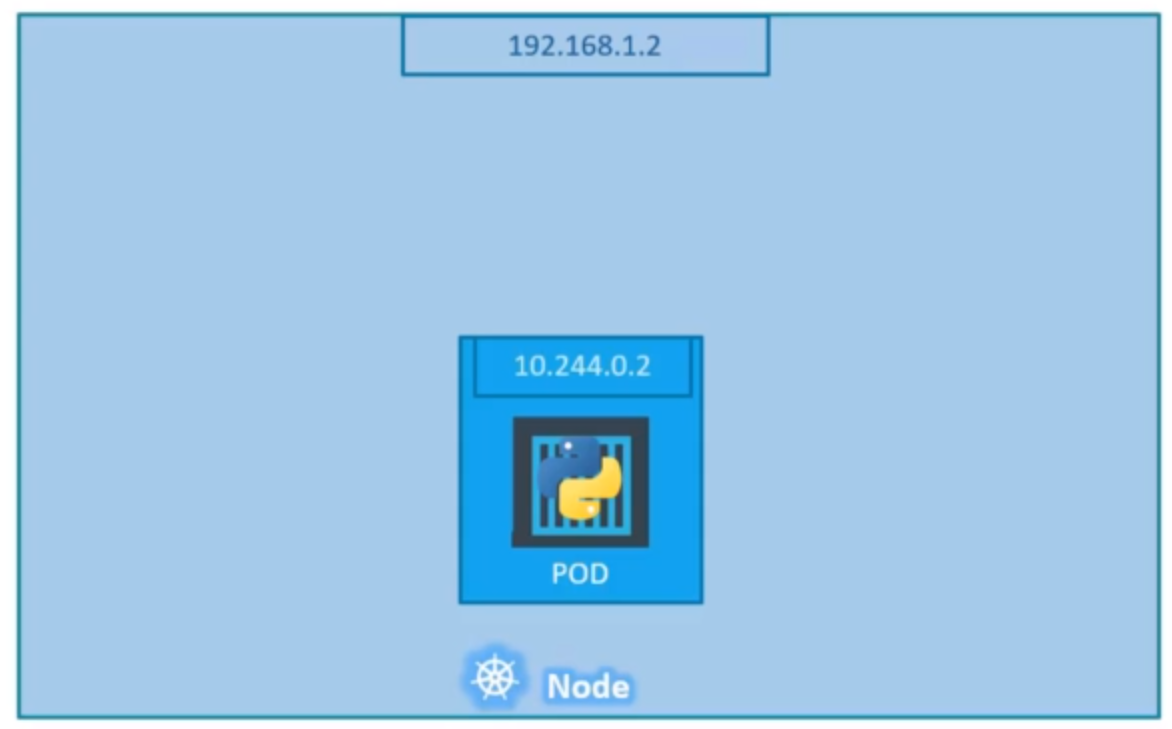
Let’s take an example of a single node Kubernetes cluster, the node has an IP address 192.168.1.2, this is the IP address we use to access the Kubernetes. On a side note, if you’re using a mini cube setup, then I’m talking about the IP address of the mini cube virtual machine inside your hypervisor. Your laptop may be having a different IP like 192.168.1.10, so it’s important to understand how your VMs are set up.

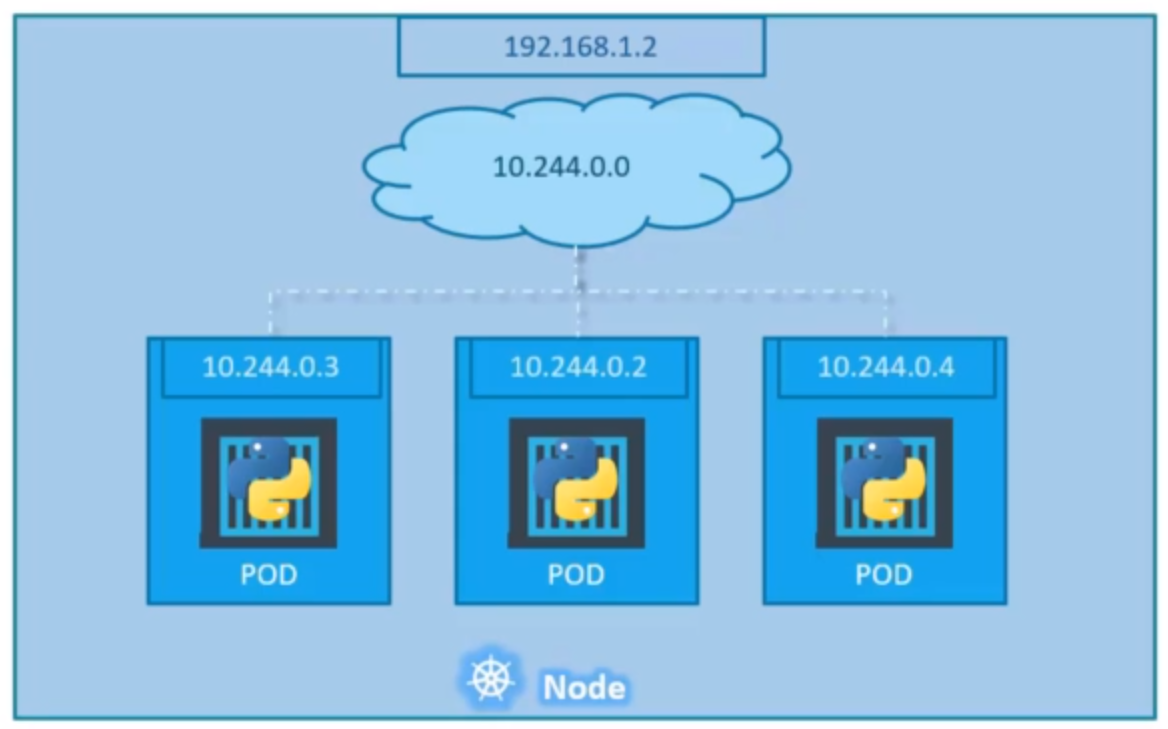


On the single node Kubernetes cluster, we have created a single pod. As you know, a pod holds a container, unlike in the Docker world where an IP address is always assigned to a Docker container. In the Kubernetes world, the IP address is assigned to a pod. Each pod in the Kubernetes gets its own internal IP address.

In this case, it’s in the range of 10.244 series and the IP assigned to the pod is 10.244.0.2, so how is it getting this IP address?



When Kubernetes is initially configured, we create an internal private network with the address 10.244.0.0 and all the pods are attached to it. When you deploy multiple pods, they all get a separate IP assigned from this network. The pods can communicate to each other through this IP but accessing the other pods using this internal IP address may not be good idea as it subject’s to change when pods are recreated, we will see better ways to establish communication between pods in the next section.



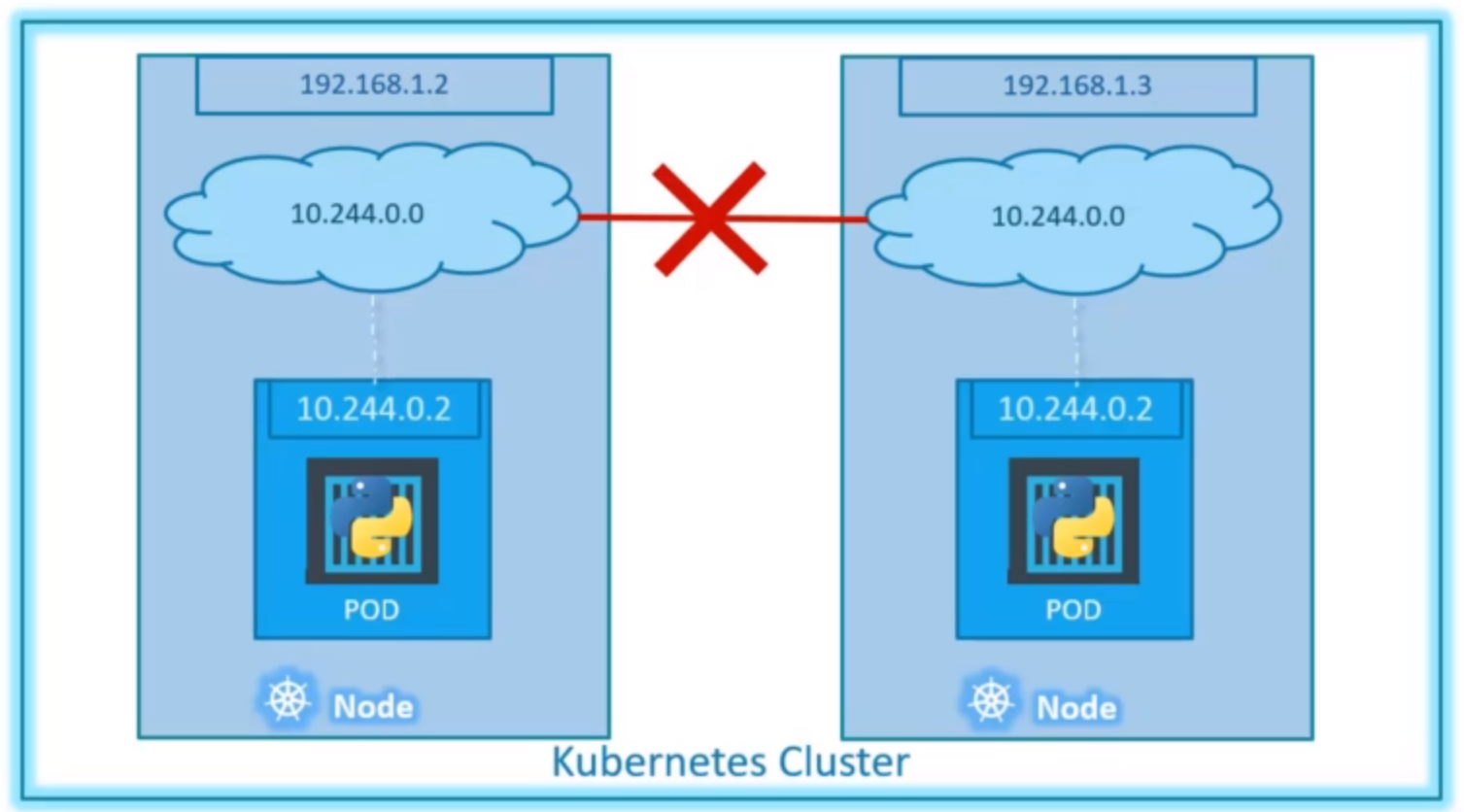
It's all easy and simple to understand when it comes to networking on a single node. But how does it work when you have multiple nodes in your cluster?

In this case, we have two nodes running Kubernetes and they have IP addresses 192.168.1.2 and 192.168.1.3 assigned to them. Note, they are not part of the cluster yet. Each of them has a single pod deployed, these pods are attached to an internal network and they have their own IP addresses assigned. However, if you look at the internal network addresses, you can see that they are the same.

The two networks have an address 10.244.0.2 and the pods deployed have the same address too. This is not going to work well when the nodes are part of the same cluster, the pods have the same IP addresses assigned to them, and that will lead to IP conflicts in the network. That’s the one problem. When a Kubernetes cluster is set up, Kubernetes does not automatically set up any kind of networking to handle these issues.

As a matter of fact, Kubernetes expects us to set up networking to meet certain fundamental requirements. Some of these are:

1. All containers or pods can communicate to one another without NAT
2. All nodes can communicate with all containers and vice versa without NAT.

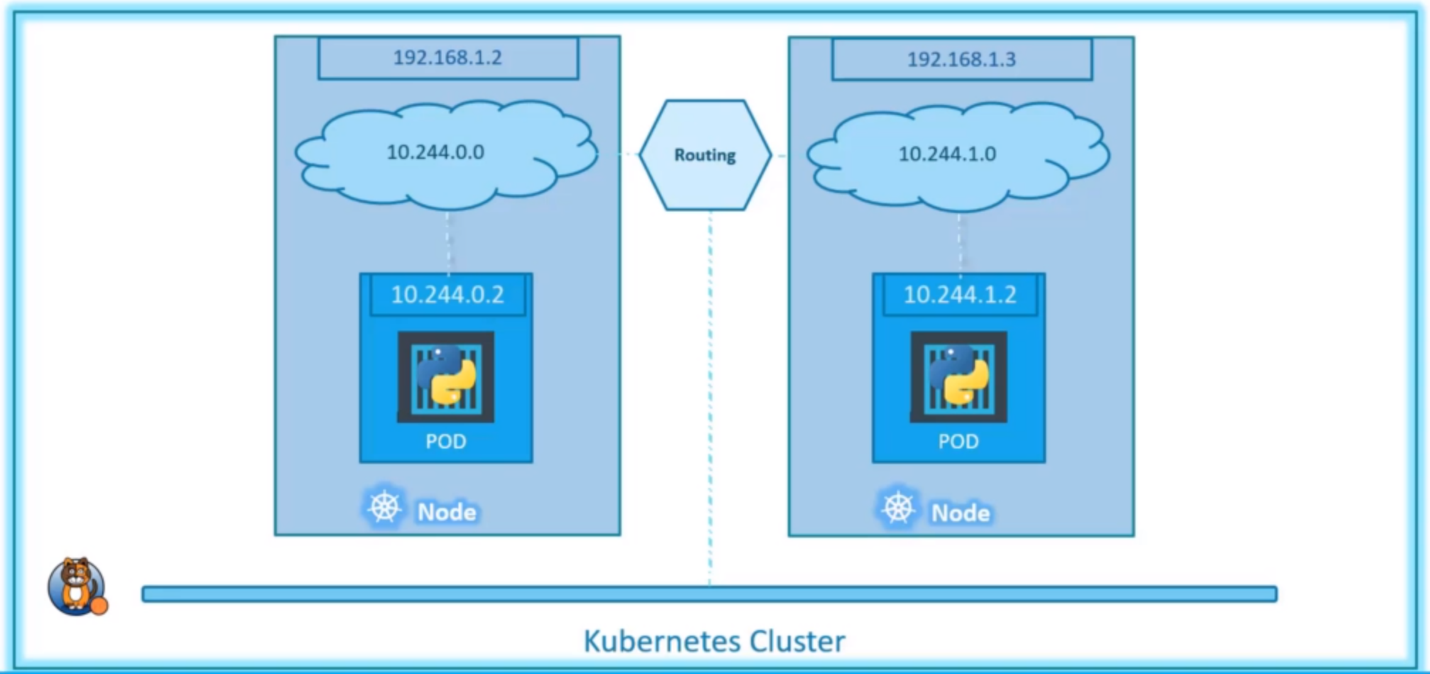


Fortunately, we don’t have to set it up all on our own as there are multiple pre-built solutions available.



Depending on the platform you’re deploying your Kubernetes cluster on, you may use one of these solutions. For example, if you are setting up a Kubernetes cluster from scratch on your own system, you may use any of the solutions like Calico or Flannel. If you are deploying on a VMWare environment, NSXT may be a good option.

So, back to our cluster with the custom networking either Flannel or Calico set up, it now manages the networks and IPs in the node and assigns a different network address for each network in the node. This create a virtual network of all pods and nodes where they are all assigned a unique IP address. And by using simple routing techniques, the cluster networking enables communication between the different pods or nodes to meet the networking requirements of Kubernetes.



Thus, all the pods now can communicate with each other using the assigned IP address.