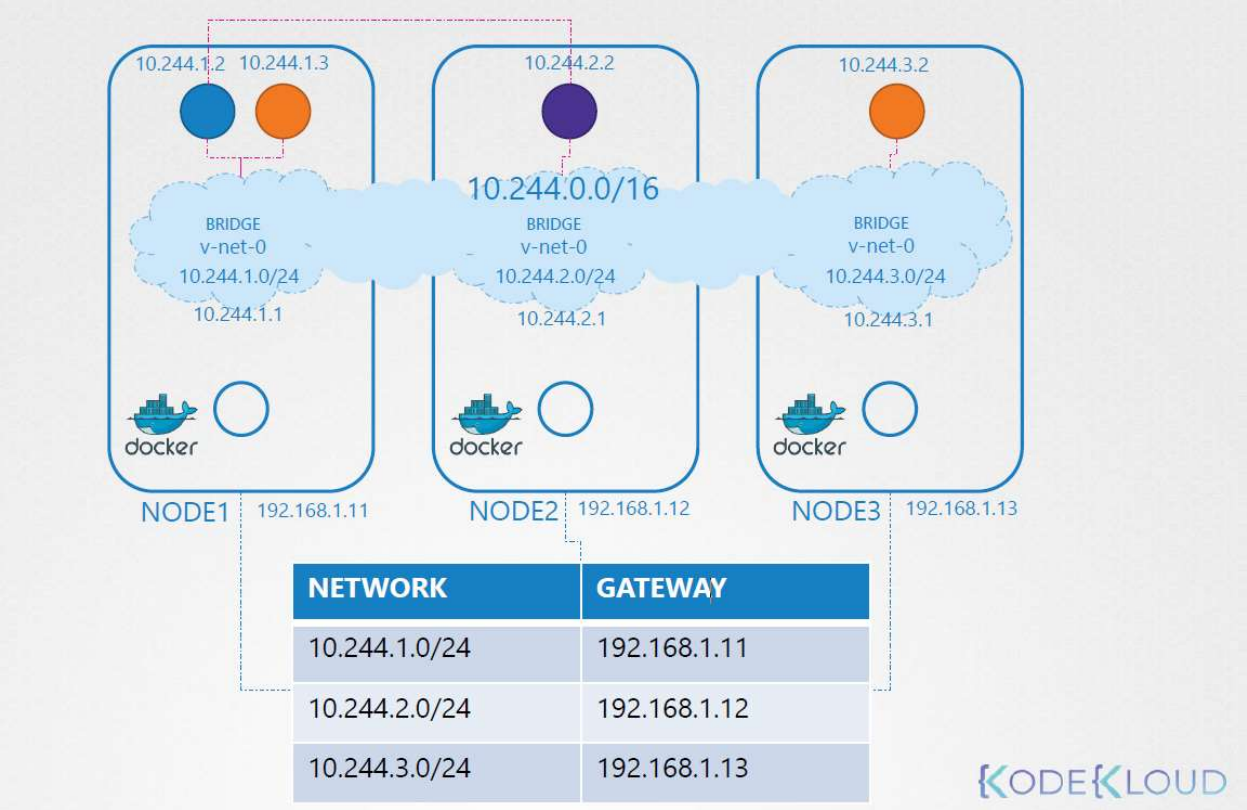
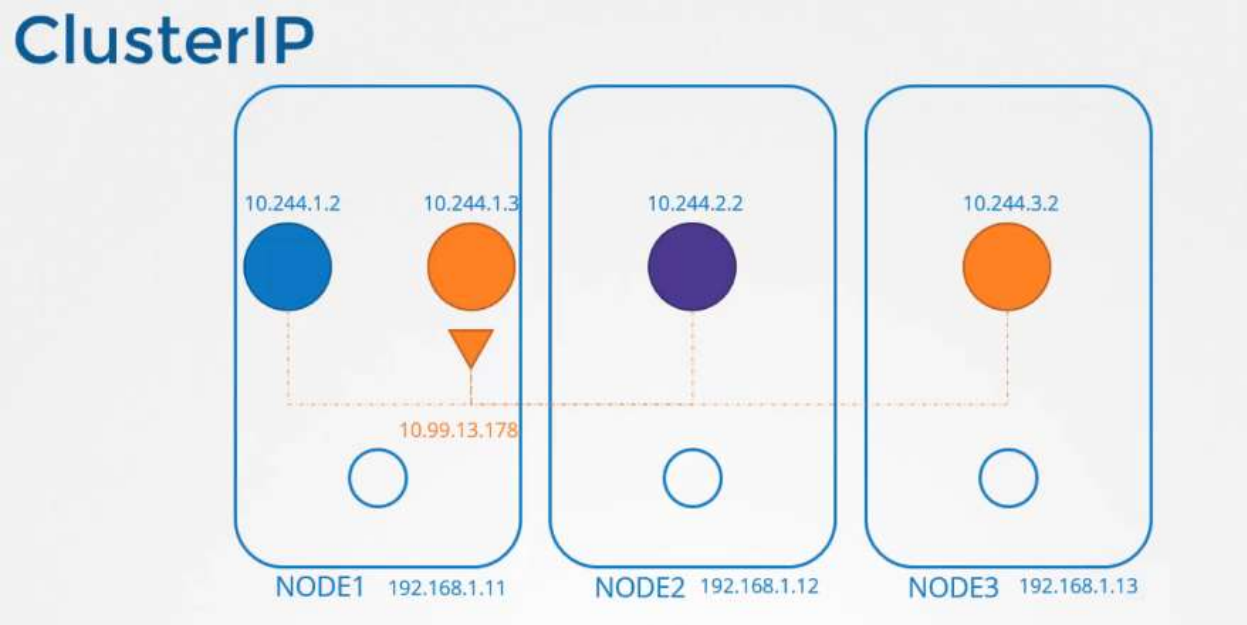
In this lecture, we will discuss about service networking. In the previous lectures, we talked about pod networking, how bridge networks are created within each node, and how pods get a namespace created for them, and how interfaces are attached to those namespaces, and how pods get an IP address assigned to them within the subnet assigned for that node. And we also saw through routes, or other overlay techniques, we can get the pods in different nodes to talk to each other, forming a large virtual network where all pods can reach each other. Now, you would rarely configure your pods to communicate directly with each other. If you want a pod to access services hosted on another pod, you would always use a service.



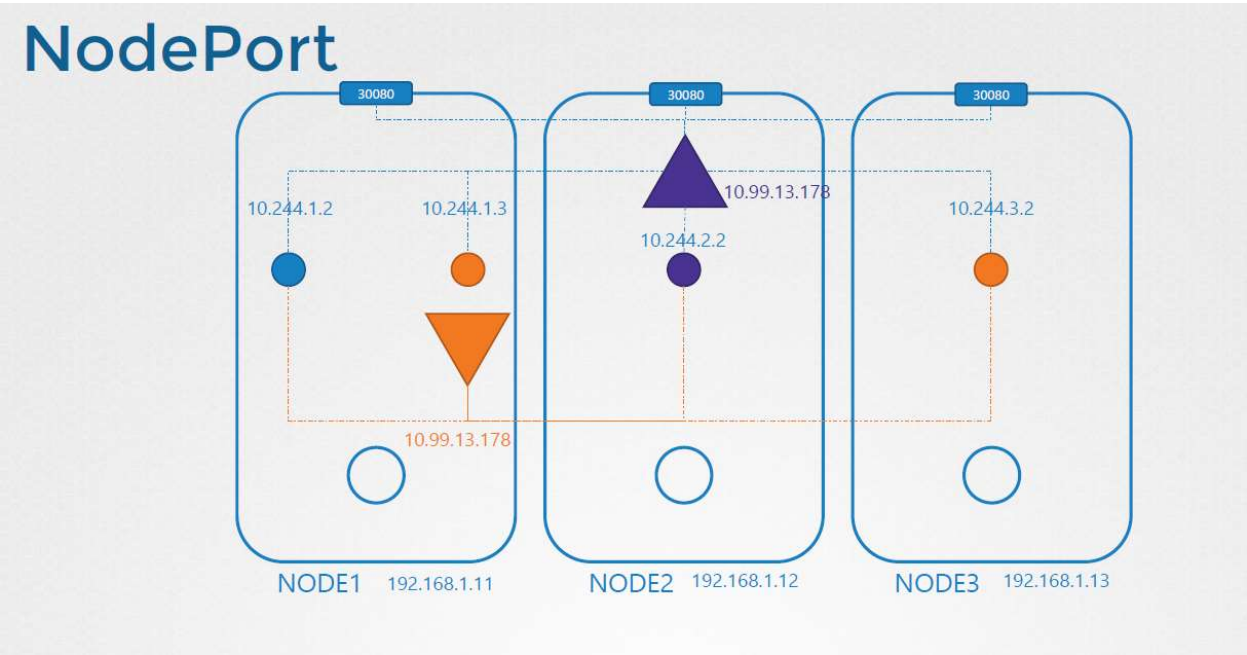
Let's quickly recap the different kinds of services. To make the orange pod accessible to the blue pod, we create an orange service. The orange service gets an IP address, and a name assigned to it. The blue pod can now access the orange pod through the orange services IP or it's name. We'll talk about name resolution in the upcoming lectures. For now, let's just focus on IP addresses. The blue and orange pod are on the same node.

What about access from the other pods on other nodes? When a service is created, it is accessible from all pods on the cluster, irrespective of what nodes the pods are on. While a pod is hosted on a node, a service is hosted across the cluster. It is not bound to a specific node, but remember, the service is only accessible from within the cluster. This type of service is known as ClusterIP.

If the orange pod was hosting a database application that is to be only accessed from within the cluster, then this type of service works just fine.



Say for instance, the purple pod was hosting a web application, to make the application on the pod accessible outside the cluster, we create another service of type NodePort. This service also gets an IP address assigned to it and works just like ClusterIP, as in all the other pods can access this service using it's IP. But in addition, it also exposes the application on a port on all nodes in the cluster. That way external users or applications have access to the service.



So that's the topic of our discussion for this lecture. Our focus is more on services and less on pods. How are the services getting these IP addresses? And how are they made available across all the nodes in the

cluster? How is the service made available to external users through a port on each node? Who is doing that and how and where do we see it?

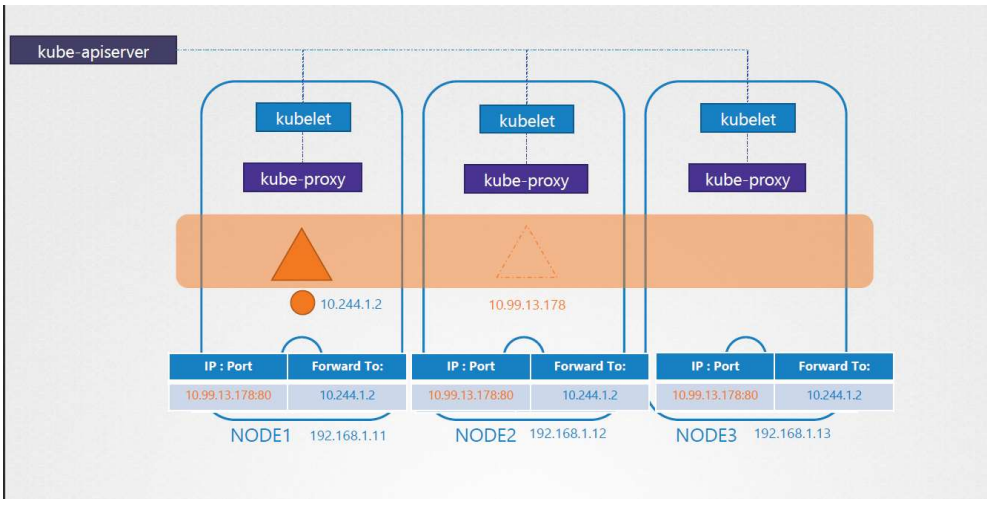
Let's start on a clean slate.

We have a three node cluster, no pods or services yet. We know that every Kubernetes node runs a kubelet process, which is responsible for creating pods. Each kubelet service on each node watches the changes in the cluster through the Kube API server, and every time a new pod is to be created, it creates the pod on the nodes. It then invokes the CNI plugin to configure networking for that pod.

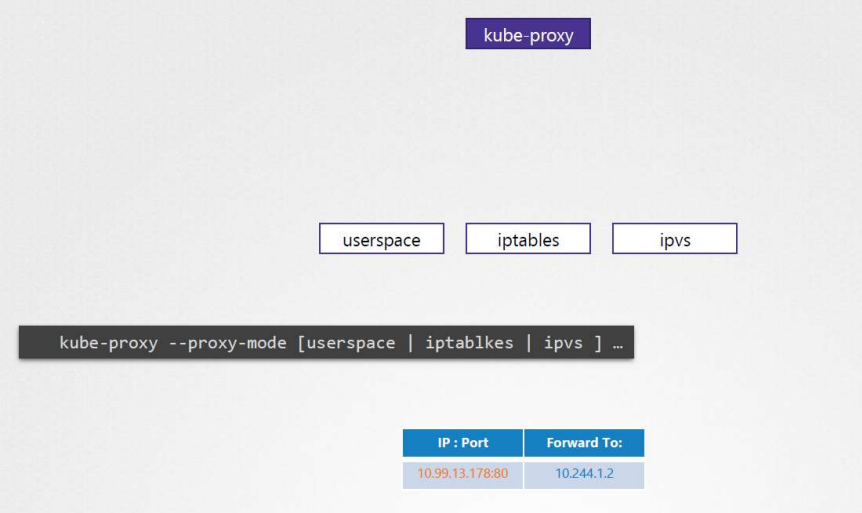
Similarly, each node runs another component known as kube-proxy. kube-proxy watches the changes in the cluster through Kube API server, and every time a new service is to be created, kube-proxy gets into action. Unlike pods, services are not created on each node or assigned to each node. Services are a cluster-wide concept. They exist across all the nodes in the cluster. As a matter of fact, they don't exist at all. There is no server or service really listening on the IP of the service.

We have seen that pods have containers, and containers have namespaces with interfaces, and IPs assigned to those interfaces. With services, nothing like that exists. There are no processes or namespaces or interfaces for a service. It's just a virtual object. Then how do they get an IP address, and how are we able to access the application on the pod through a service?

When we create a service object in Kubernetes, it is assigned an IP address from a predefined range. The kube-proxy components running on each node gets that IP address and creates forwarding rules on each node in the cluster. Saying, "Any traffic coming to this IP, the IP of the service, should go to the IP of the pod." Once that is in place, whenever a pod tries to reach the IP of the service, it is forwarded to the pod's IP address, which is accessible from any node in the cluster. Now, remember, it's not just the IP, it's an IP and port combination. Whenever services are created or deleted, the kube-proxy component creates or deletes these rules.

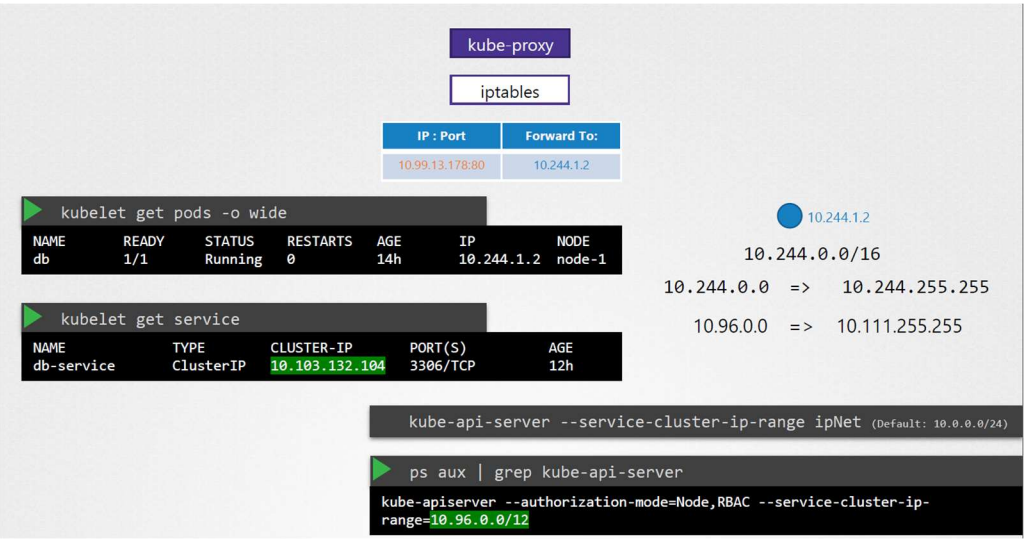


So how are these rules created? kube-proxy supports different ways, such as userspace where kube-proxy listens on a port for each service and proxy's connections to the pods by creating IPVS rules. Or the third and the default option, and the one familiar to us is using iptables. The proxy mode can be set using the proxy mode option, while configuring the kube-proxy service. If this is not set, it defaults to iptables.



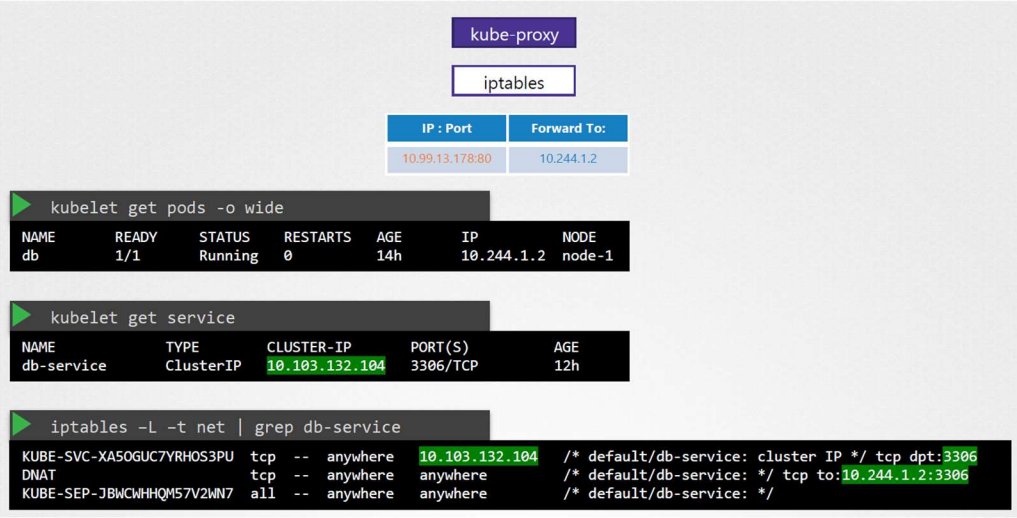
So we'll see how iptables are configured by kube-proxy, and how you can view them on the nodes. We have a pod named db deployed on node one. It has IP address 10.244.1.2. We create a service of that cluster IP to make this pod available within the cluster. When the service is created, Kubernetes assigns an IP address to it. It is set to 10.103.132.104. This range is specified in the Kube API server's option called service cluster IP range, which is by default set to 10.0.0.0/24. In my case, if I look at my Kube API server option, I see it is set to 10.96.0.0/12. That gives my services IP anywhere from 10.96.0.0 to 10.111.255.255.

A relative point to mention here, when I set up my pod networking, I provided a pod network CIDR range of 10.244.00/16, which gives my pods IP addresses from 10.244.0.0 to 10.244.255.255. The reason I brought this up here is because whatever range you specify for each of these networks, it shouldn't overlap, which it doesn't in this case. Both of these should have it's own dedicated range of IPs to work with. There shouldn't be a case where a pod, and a service are assigned the same IP address.



So getting back to services, that's how my service got an IP address of 10.103.132.104. You can see the rules created by kube-proxy in the IP table's NAT table output. Search for the name of the service as all

rules created by kube-proxy have a comment with the name of the service on it. This rules mean any traffic going to the IP address 10.103.132.104 on port 3306, which is the IP of the service, should have it's destination address changed to 10.244.1.2 and port 3306, which is the IP of the pod. This is done by adding a DNAT rule to iptables. Similarly, when you create a services of type NodePort, kube-proxy creates IP table rules to forward all traffic coming on a port on all nodes to the respective backend pods.



You can also see kube-proxy create these entries in the kube-proxy logs itself. In the logs, you will find what proxier it uses. In this case, it's iptables, and then adds an entry when it added a new service for the database. Note that the location of this file might vary depending on your installation. If you don't see these entries, you must also check the verbosity level of the process as well.