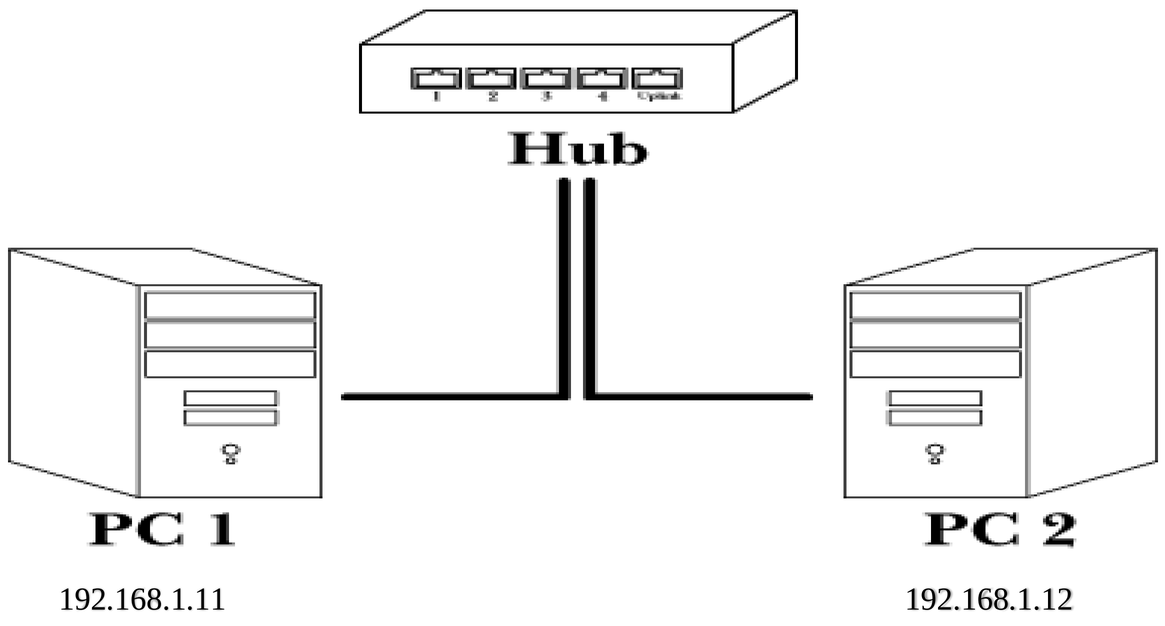
Kubernetes Network

➢ What is Network?

We have two computers, A and B, laptops, desktops, VMs on the cloud, wherever, how does system A reach B? We connect them to a switch and the switch creates a network containing the two systems. To connect them to a switch, we need an interface on each host: physical or virtual, depending on the host. To see the interfaces for the host, we use the IP link command. In this case, we look at the interface named Eth0 that we will be using to connect to the switch. Let's assume it's a network with the address 192.168.1.0.
We then assign the systems with IP addresses on the same network. For this, we use the command ip addr. Once the links are up and the IP addresses are assigned, the computers can now communicate with each other through the switch. The switch can only enable communication within a network, which means it can receive packets from a host on the network and deliver it to other systems within the same network.



➢ Kubernetes Network

In Kubernetes we are using multiple Nodes and Pods and the communication between them Very necessary.
how can we interconnect them?
So we created a program or script that performs all the required tasks to get the container attached to a bridge network. For example, you could run this program using its name, Bridge and specify that you want to add this container to a particular network name space. The Bridge program takes care of the rest so that the container runtime environments are relieved of those tasks. For example, when Rocket or Kubernetes creates a new container, they call the Bridge Program and pass the container ID and namespace to get networking configured for that container. So what if you wanted to create such a program for yourself, maybe for a new networking type?
If you were doing so,

- what arguments and commands should it support? How do you make sure the program you create will work currently with these run times?

- How do you know that container run times like Kubernetes or Rocket_will invoke your program correctly?

That's where we need some standards defined. A standard that defines how a program should look,
how container run times will invoke them so that everyone can adhere to a single set of standards
and develop solutions that work across run times. That's where container network interface comes in.
The CNI is a set of standards that define how programs should be developed to solve networking challenges_in a container runtime environment.

- CNI defines how the plugin should be developed and how container run times should invoke them.
- CNI defines a set of responsibilities for container run times and plugins. For container run times,
- CNI specifies that it is responsible for creating a network name space for each container.

CNI comes with a set of supported plugins already such as Bridge, VLAN, IP VLAN, MAC VLAN, one for Windows as well as IPAM plugins like Host Local and DHCP.
There are other plugins available from third party organizations as well.
Some examples are Weave, Flannel, Cilium, VMware NSX, Calico, Infoblox, et cetera.
All of these containers run times implement CNI standards so any of them can work with any of these plugins.
But there is one that is not in this list.
Docker.
Docker does not implement CNI. Docker has its own set of standards known as CNM which stands for container network model which is another standard that aims at solving container networking challenges similar to CNI but with some differences. Due to the differences, these plugins don't natively integrate with Docker, meaning you can't run a Docker container and specify the network plugin to use a CNI and specify one of these plugins.
But that doesn't mean you can't use Docker with CNI at all.

➢ Kubernetes Services

We have seen what is network and Kubernetes how to use network in our environment but if you want to create communication between nodes then the Services comes in the picture.
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.

a) **Cluster IP** - 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.

Suppose you have created cluster of nodes inside cluster we have our master and worker nodes available and we deployed multiple pods inside our nodes the with the help of CLUSTERIP service all pods can communicate with each other but the limitation is that they can communicate

Within our cluster only. Like front-end and database pods communication .

Each pod has its own IP Address 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.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
     type: clusterIP
     ports:
-    targetPort: 80
port: 80
```

b) NodePort –

NodePort service in Kubernetes is a service that is used to expose the application to the internet from where the end-users can access it. If you create a NodePort Service Kubernetes will assign the port within the range of (30000-32767). The application can be accessed by end-users using the node's IP address. The port must be in the range between 30000 and 32767 and if you don’t specify the NodePort value Kubernetes will assign it randomly.

```
apiVerision: v1
kind: service
metadata:
     name: my-service
spec:
     type: NodePort
     selector:
       app.kubernetes.io/name: MyApp
     ports:
                  -    Port: 80
```

c) LoadBalancer:

It operates at the transport level (TCP) that is at level 4. It means that is unable to make decisions based on content and it uses a simple algorithm such as a round-robin across the selected paths. Whereas Ingress operates at the application level which is at level 7. It is able to make decisions based on the actual content of each message. More intelligent load-balancing decisions and content optimizations. In other words, Ingress is like a LoadBalancer but more intelligent.

```
apiVerision: v1
kind: service
metadata:
          name: my-service
```

```
spec:
          selector:
            app.kubernetes.io/name: MyApp
          ports:
                - Port: 80
clusterIP: 192.168.1.1
           type: LoadBalancer
```

➢ DNS

• What is DNS?

We have two computers, A and B, both part of the same network, and they've been assigned with IP addresses 192.168.1.10, and 1.11. You're able to ping one computer from the other
using the other computer's IP address. You know that system B has database services on them.
So instead of having to remember the IP address of system B, you decide to give it a name db.
Going forward, you would like to ping system B using the name db instead of its IP address.
If you try to ping db now,
you would see that host A is unaware of a host named db.
So how do you fix that?
Basically, you wanna tell system A that system B at IP address 192.168.1.11 has a name db.
You wanna tell system A that when I say db, I mean the IP 192.168.1.11. You can do that by adding an entry into the /etc/hosts file on system A. Mention the IP address and the name you want your host to see system B has. We told system A that the IP at 192.168.1.11 is a host named db.
Similarly we can use multiple system and write there host name into /etc/host .
into a single server who will manage it centrally. We call that our DNS server. And then we point all hosts to look up that server if they need to resolve a host name to an IP address .
there are so many concepts in DNS like.
host NS lookup,

and the dig utility and the different types of DNS records like C-name, etc .

➢ DNS in Kubernetes

Whenever a service is created, the Kubernetes DNS service, creates a record for the service.
It maps the service name to the IP address, so, within the cluster, any pod can now reach this service using its service name.
that everyone within the name space address each other just with their first names, and to address anyone in another name space, you use their full names.
In this case, since the test pod and the web pod, and its associated service are all in the same name space, the default name space, you were able to simply reach the web service from the test pod using just the service name Web-Service.

➢ Ingress

In simple terms we can say ingress in Kubernetes is work like loadkbalancer.
We will start with a simple scenario. You are deploying an application on Kubernetes for a company that has an online store selling products.
Onlineshop.com and it has its own data base suppose the application is running on port number 38080 and working properly with the help of load balancer it manages the traffic but after some time the number of customer will increase and company wants to deploy more products and some videos and images in our portal .
to access your new video streaming service by going to myonlinestore.com/watch.
You'd like to make your old application accessible at myonlinestore.com/wear.
Your developers developed the new video streaming application as a completely different application as it has nothing to do with the existing one. However, in order to share the same cluster resources, you deploy the new application as a separate deployment within the same cluster.
You create a service called video service of type load balancer. Kubernetes provisions port 38282 for this service, and also provisions a network load balancer on the cloud.
The new load balancer has a new IP.

Remember, you must pay for each of these load balancers and having many such load balancers can inversely affect your cloud bill.

So how do you direct traffic between each of these load balancers based on the URL that the users type in? You need yet another proxy or load balancer that can redirect traffic based on URLs to the different services. Every time you introduce a new service, you have to reconfigure the load balancer. And finally, you also need to enable SSL for your applications.

That's where ingress comes in.

Ingress helps your users access your application using a single externally accessible URL that you can configure to route to different services within your cluster based on the URL path.
At the same time, implement SSL security as well. Simply put, think of ingress as a layer seven load balancer built in to the Kubernetes cluster that can be configured using native Kubernetes primitives
just like any other object in Kubernetes. Now remember, even with ingress, you still need to expose it
to make it accessible outside the cluster.
So you still have to either publish it as a node port or with a cloud native load balancer, but that is just a one-time configuration. Going forward, you're going to perform all your load balancing authentication,
SSL and URL-based routing configurations On the ingress controller.

Find more information and examples in the below reference link:-

https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#-em-ingress-em-

References:-

https://kubernetes.io/docs/concepts/services-networking/ingress

https://kubernetes.io/docs/concepts/services-networking/ingress/#path-types