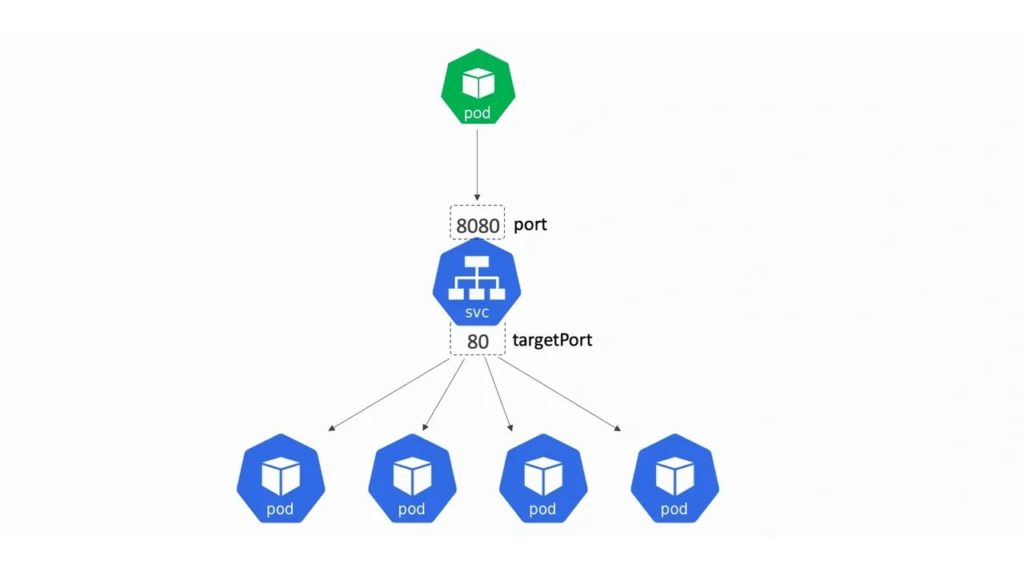
**Explained: Kubernetes Service Ports**



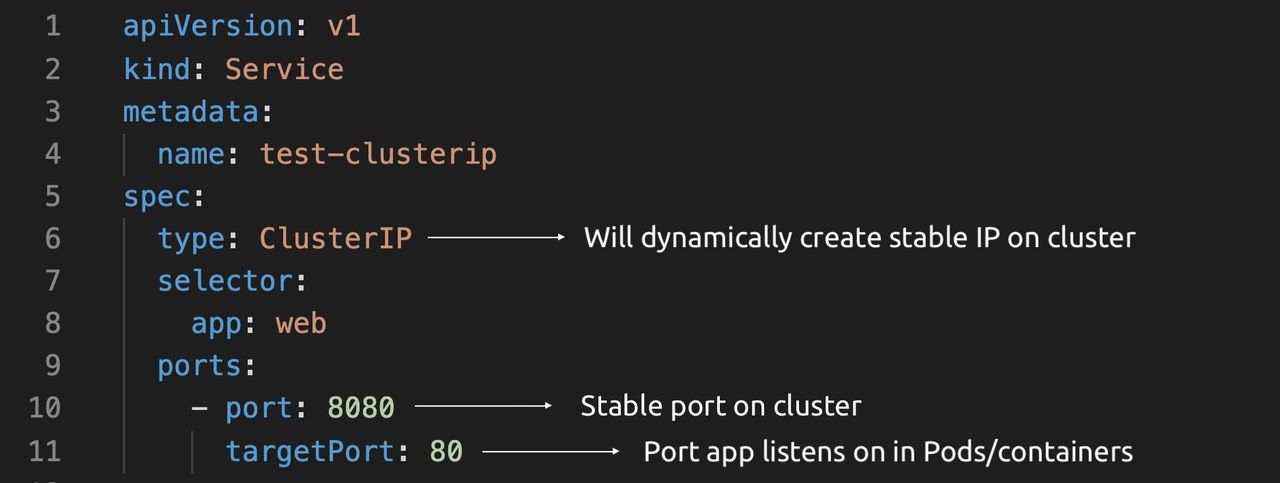
While I was creating the brand new version of my [**Getting Started with Kubernetes**](https://www.pluralsight.com/courses/kubernetes-getting-started) course on Pluralsight (right click the link and open in new window!!!), I realised port mappings on *Kubernetes Service objects* can be confusing. So here goes with an explanation…

Kubernetes has three major Service types: **ClusterIP**, **NodePort**, and **LoadBalancer**…

**ClusterIP**

This is the default and most basic type. Its job is to provide a *stable IP and port* that passes traffic to Pods/containers on the same cluster. The stable IP and port are only accessible from other Pods running in the cluster.

We define it like this in a YAML file:

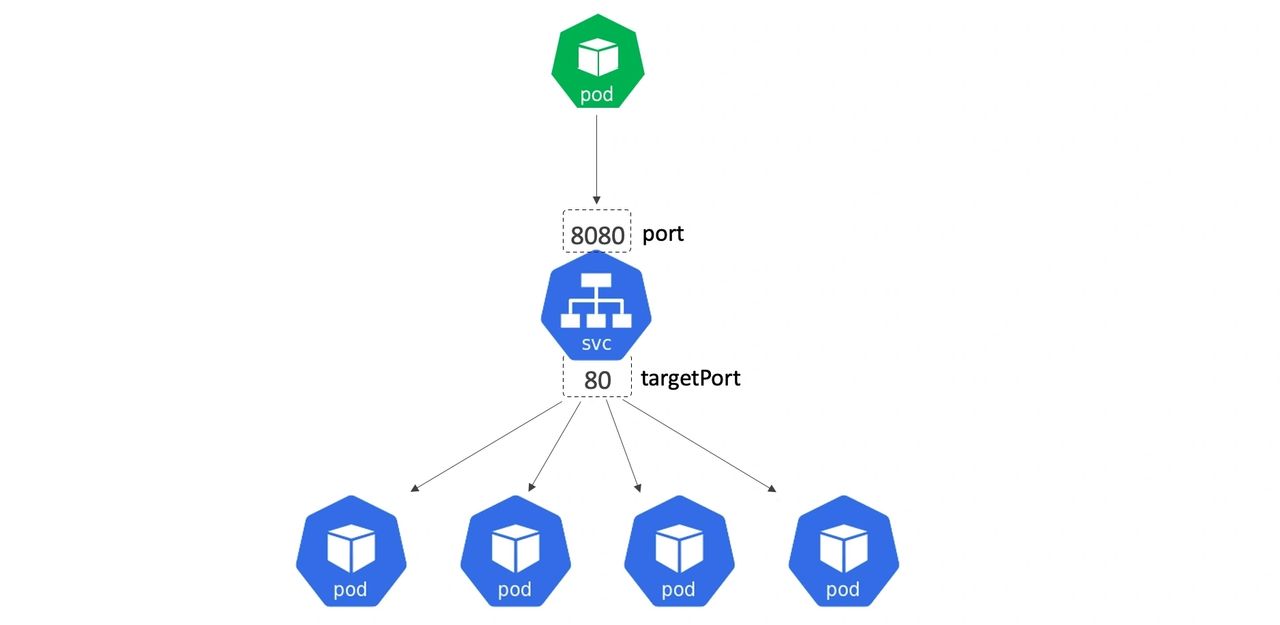


The YAML defines two ports:

* port
* targetPort

**port** is the stable port the Service exposes inside the cluster — other Pods in the cluster send traffic to this port (8080 in our example). **targetPort** is the port that the application listens on in the Pods/containers.

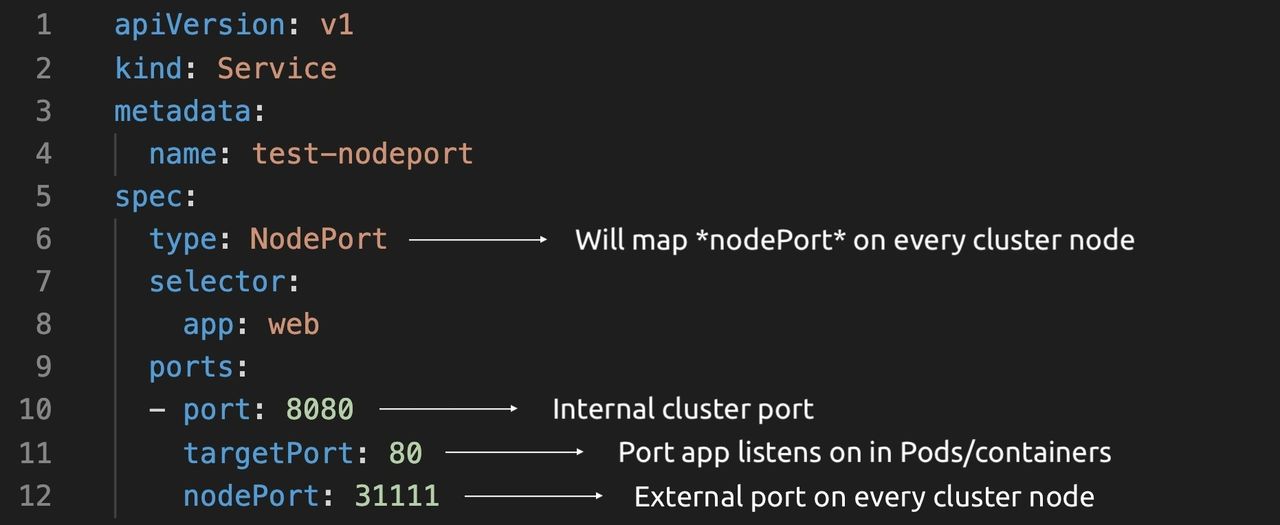
The diagram below shows how other Pods send traffic to port 8080 and how the Service redirects that to port 80 in the Pods/containers.



If you don’t specify a **targetPort** value, it defaults to the value specified in **port**.

**NodePort**

A **NodePort** Service exposes an app to the outside world via a port mapping on every node in the cluster. It looks like this in a YAML file:

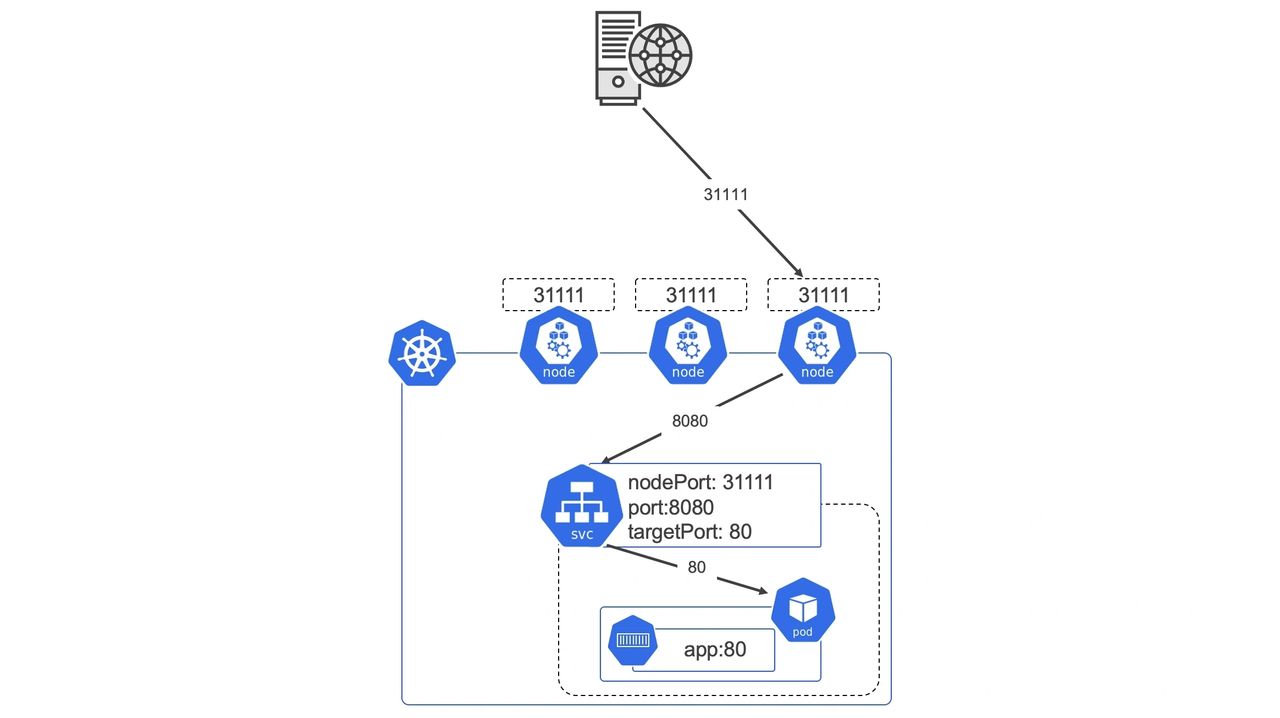


This time the YAML defines three ports:

* port
* targetPort
* nodePort

**port** and **targetPort** work the same as they do with ClusterIP Services. The **nodePort** is a TCP/UDP port between 30,000 and 32,767 that is mapped on every cluster node and exposes the Service outside of the cluster. Basically, any client outside of the cluster can hit any cluster node on the nodePort value (31111 in our example) and reach the ClusterIP Service inside the cluster and eventually reach Pods/containers.

The diagram below shows how external clients send traffic to cluster nodes on port 31111, get routed to the ClusterIP Service on port 8080, and eventually to a Pod/container listening on port 80.



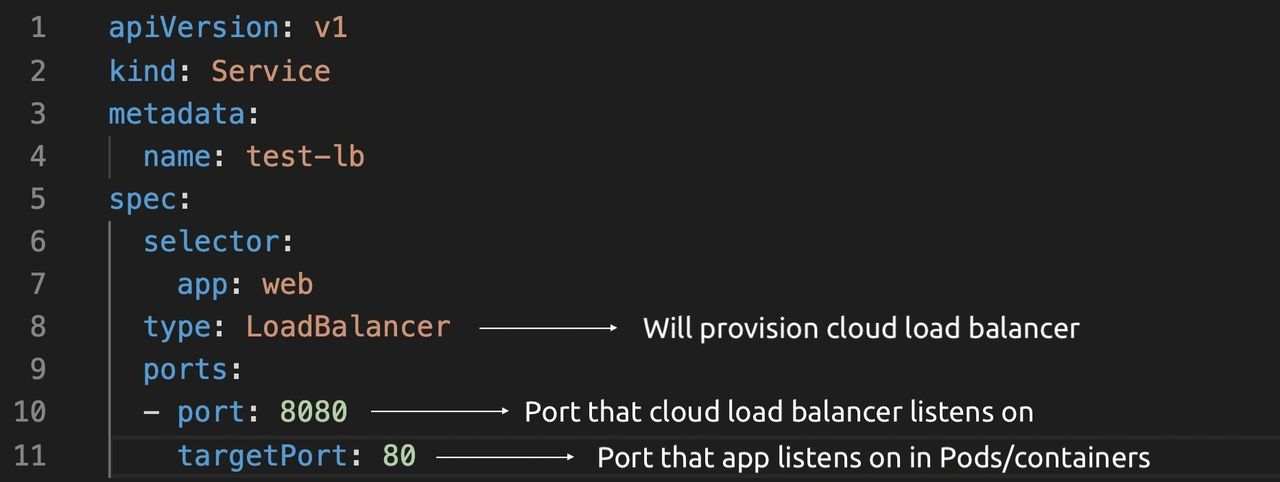
It’s important to understand that NodePort Services build on top of ClusterIP Services. However, when you define a NodePort Service, Kubernetes takes care of creating any ClusterIPs and mapping ports etc.

**LoadBalancer**

Last but not least, Kubernetes offers a **LoadBalancer Service**. This builds on top of *NodePort*and *ClusterIP* constructs and exposes a Service to the internet via one of your cloud’s native load balancers.

It’s important to understand that a Kubernetes LoadBalancer Service will build an internet-facing load balancer on your cloud platform as well as all the constructs required to route traffic all the way back to Pods/containers running in your Kubernetes cluster.

They’re defined like this in YAML files.

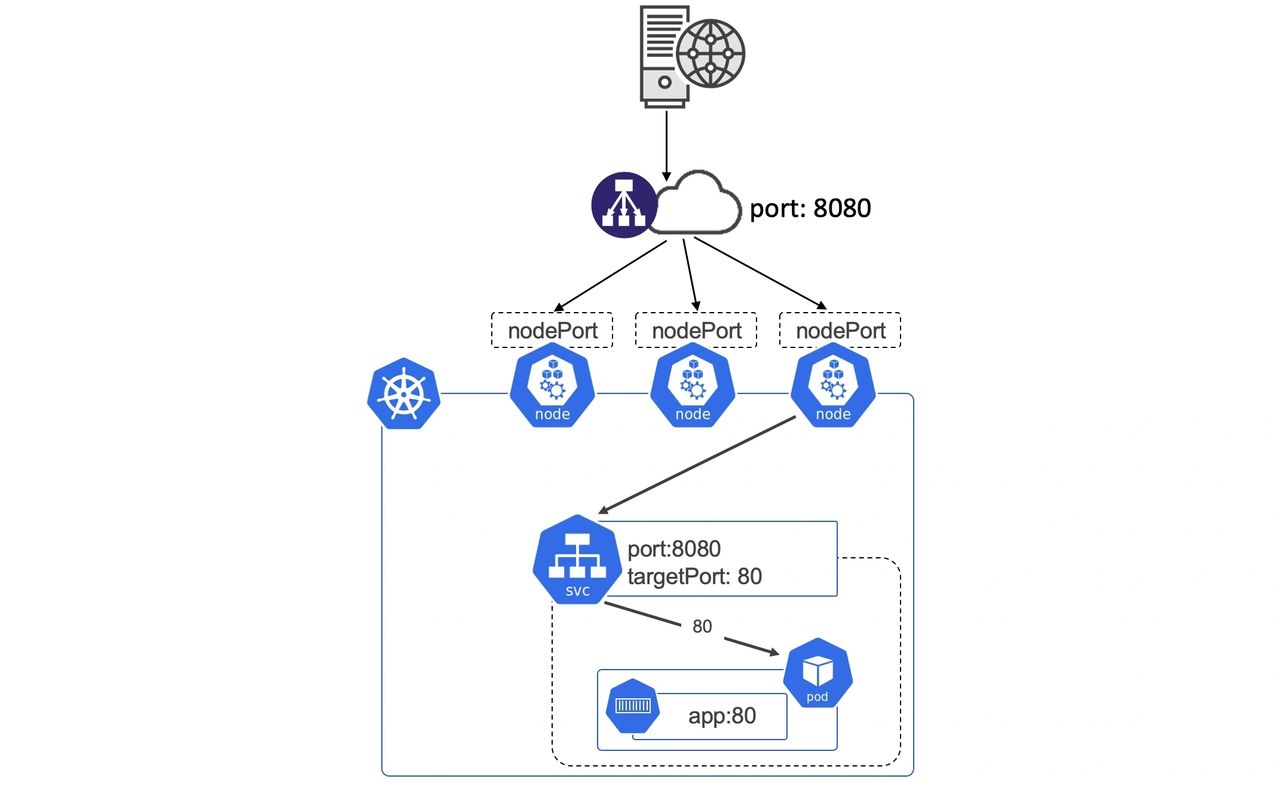


This time the YAML only defines two ports:

* port
* targetPort

**port** is the port the cloud load balancer will listen on (8080 in our example) and **targetPort** is the port the application is listening on in the Pods/containers. Kubernetes works with your cloud’s APIs to create a load balancer and everything needed to get traffic hitting the load balancer on port 8080 all the way back to the Pods/containers in your cluster listening on targetPort 80.

Behind the scenes, many implementations create NodePorts to glue the cloud load balancer to the cluster. The traffic flow is usually like this.



As you can see, LoadBalancer Services build on top of NodePorts which in turn build on top of ClusterIPs.

If you’re hungry for more, including examples and animated explanations, see my [**Getting Started with Kubernetes course on Pluralsight**](https://www.pluralsight.com/courses/kubernetes-getting-started) (right click and open in new window!!!). And feel free to reach on out the various socials where I’m more than happy to talk technology all day!