A full-stack web application typically has different kinds of pods hosting different parts of an application. You may have a number of pods running a frontend web server, another set of pods running a backend server, a set of pods running a key value store like Redis and another set of pods running persistent database like SQL.

The Web frontend server needs to communicate to the backend servers and the backend servers need to communicate to the database as well as Redis services. So, what is the right way to establish connectivity between these services or tiers of the application?

The pods have an IP address assigned to them, but these IPs are not static. These pods can go down any time and new pods are created all the time. So, you cannot rely on these IP addresses for internal communication between the application. Also, what if the first frontend pod 10.244.0.3 need to connect to a backend service? Which of the three would it go to and who makes that decision?

A Kubernetes service can help us group the pods together and provide a single interface to access the pods in a group. For example, a service created for the backend pods will help group all the backend pods together and provide a single interface for other pods to access the service. The requests are forwarded to one of the pods under the service randomly.

Similarly, create additional services for Redis and allow the backend pods to access the Redis systems through the service. This enables us to easily and effectively deploy a microservices based application on Kubernetes cluster. Each layer can now scale or move as required without impacting communication between the various services.



Each service gets an IP and name assigned to it inside the cluster, and that is the name that should be used by other pods to access the service. This type of service is known as ClusterIP. To create such a service, use a definition file in the service definition file.



From the above image, the default value for type is ClusterIP. The targetPort is the port where the backend is exposed, which in this case is 80, and the port is where the service is exposed, which is 80 as well. To link the service to a set of ports we use selector.