Agenda

Pod Networking

Services

Service Types and Load Balancers

Ingress Resource

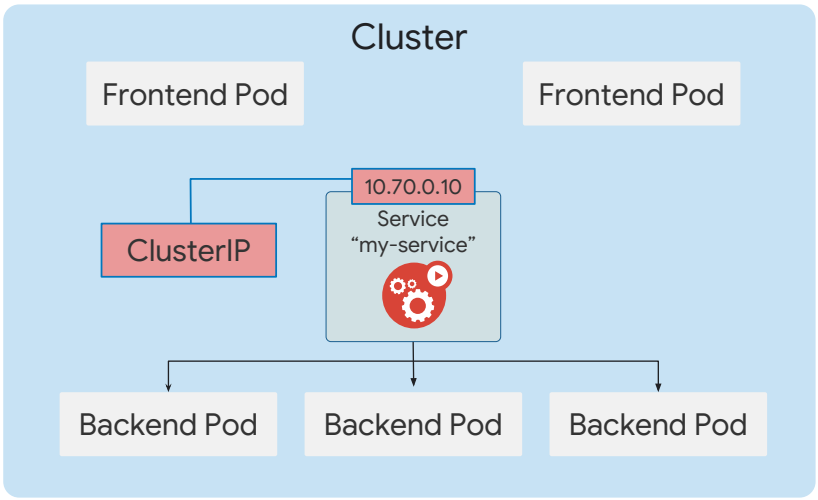
Container-Native Load Balancing

Network Security

In this lesson we'll identify the different types of Kubernetes Services as well as discuss Load Balancers.

There are three principal types of Services: ClusterIP, NodePort, and LoadBalancer. These Services build conceptually on one another, adding functionality with each step. We'll start with the basis of all the Kubernetes Services, the ClusterIP Service.

A ClusterIP Service has a static IP address



A Kubernetes ClusterIP Service has a static IP address and operates as a traffic distributor within the cluster, but ClusterIP Services aren’t accessible by resources outside the cluster.

Other Pods will use this ClusterIP as their destination IP address when communicating with the Service.

Setting up a ClusterIP Service

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: ClusterIP
  selector:
    app: Backend
  ports:
    - protocol: TCP
      port: 3306
      targetPort: 6000
```

Let's consider the step-by-step process of setting up a ClusterIP Service type. Here, you create a Service object by defining its kind. If you don't specify a Service type during the creation of the service, it will default to a Service type of ClusterIP.

Next, you use a label selector to select the Pods that will run the target application. In this case, the Pods with a label of "app:Backend" are selected and included as endpoints for this Service. You should always create a Service before creating any workloads that need to access that Service. If you create a Service before its corresponding backend workloads, such as Deployments or StatefulSets, the Pods that make up that Service get a nice bonus: they get the hostname and IP address of the Service in an environment variable.

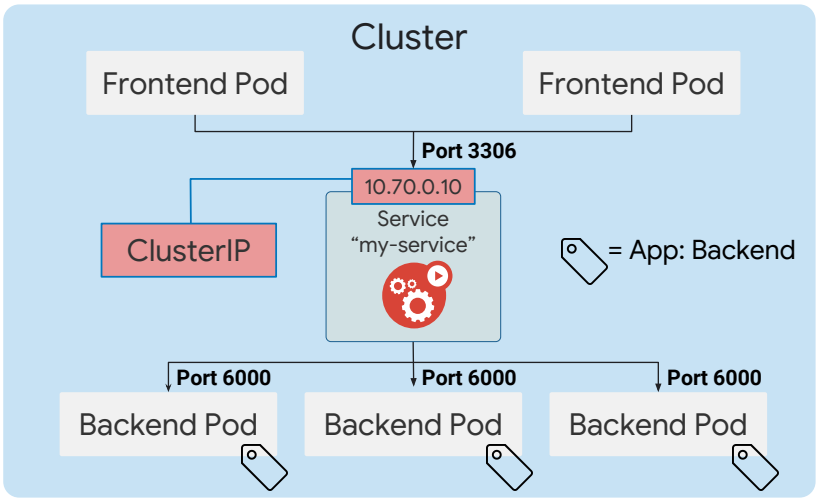
But remember that relying on environment variables for service

discovery is not as flexible as using DNS, so this practice isn't mandatory.

Next, you specify the port that the target containers are using, which in this case is TCP port 6000. This Service will receive traffic on port 3306 and then remap it to 6000 as it delivers it to the target Pods.

Creating, loading or applying this manifest will create the ClusterIP Service named "my-service". But how does this Service work?

How does the ClusterIP Service work?

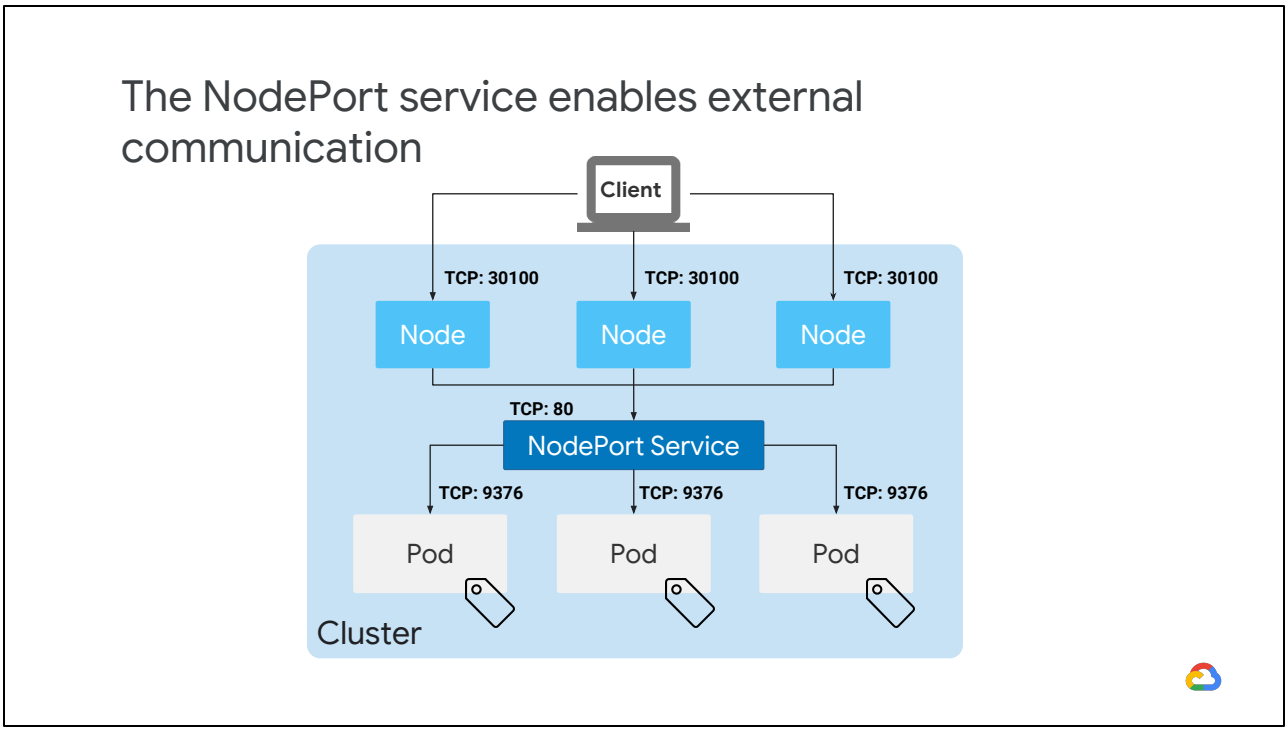


In this example, you have frontend Pods that must be able to locate the backend Pods.

When you create the Service, the cluster master assigns a virtual IP address—also known as ClusterIP—from a reserved pool of Alias IP addresses in the cluster's VPC. This IP address won't change throughout the lifespan of the Service.

The cluster master selects Pods to include in the Service's endpoints based on the label selector on the Service and the labels on the Pods.

The ip-addresses of these backend pods are mapped to the target port, TCP port 6000 in this example, to create the endpoint resources that the service forwards requests to. The ClusterIP Service will answer requests, on TCP port 3306 in this example, and forward the requests to the backend pods using their ip-addresses and target port as the endpoint resources.



ClusterIP is useful for internal communication within a cluster, but what about the external communication?
NodePort enables this. NodePort is built on top of ClusterIP Service; therefore, when you create a NodePort Service, a ClusterIP Service is automatically created in the process to distribute the inbound traffic internally across a set of pods.

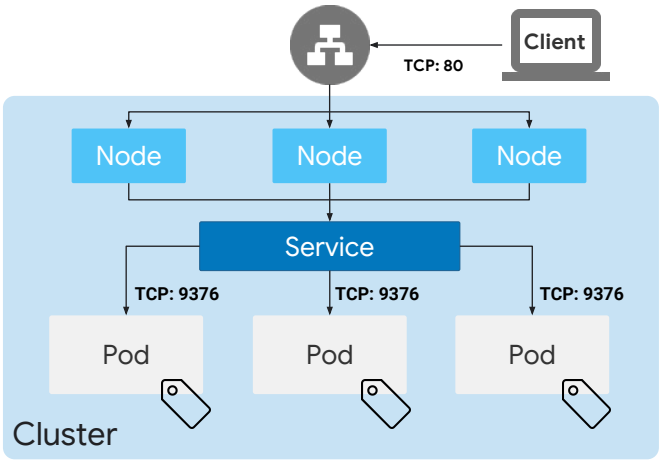
In this example we now have a Service that can be reached from outside of the cluster using the IP address of any node and the corresponding NodePort number. Traffic through this port is directed to a Service on Port 80 in this example and further redirected to one of the Pods on port 9376. NodePort Service can be useful to expose a Service through an external load balancer that you set up and manage yourself. Using this approach, you would have to deal with node management, making sure there are no port collisions.

Creating a NodePort Service

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
  selector:
    app: external
  ports:
    - protocol: TCP
      nodePort: 30100
      port: 80
      targetPort: 9376
```

In addition to the setup of the internal ClusterIP Service, a specific port is exposed for external traffic on every node. This port—also known as nodePort —is automatically allocated from the range 30,000 to 32767. In some cases users may want to manually specify it, which is allowed as long as the value also falls within that range, but this is not usually necessary.

The LoadBalancer Service can be used to expose a Service to resources outside the cluster



Conceptually, the LoadBalancer Service type builds on the ClusterIP Service and can be used to expose a Service to resources outside the cluster. With GKE, the LoadBalancer Service is implemented using GCP's network load balancer.

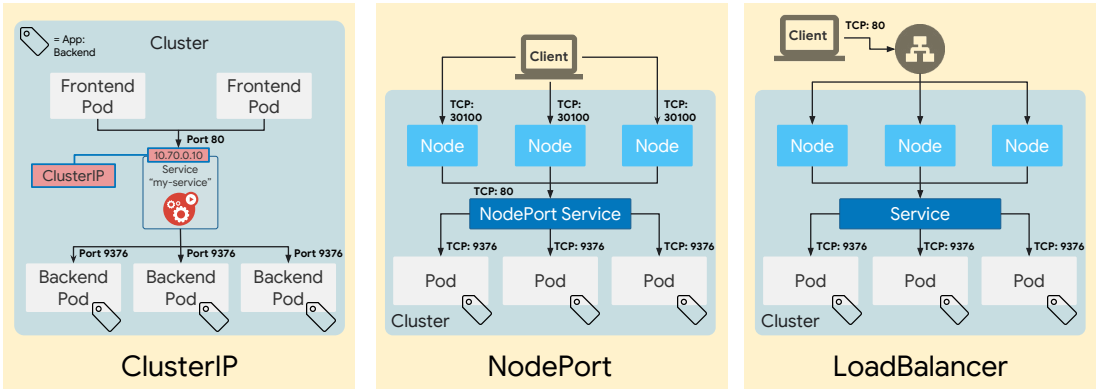
When you create a LoadBalancer Service, GKE automatically provisions a GCP network load balancer for inbound access to the Services from outside the cluster. Traffic will be directed to the IP address of the network load balancer, and the load balancer forwards the traffic on to the nodes for this Service.

Creating a LoadBalancer Service

```
[...]
spec:
  type: LoadBalancer
  selector:
      app: external
    ports:
      - protocol: TCP
      port: 80
        targetPort: 9376
```

Here's how you create a LoadBalancer Service. You only need to specify the type: LoadBalancer. GCP will assign a static load balancer IP address that is accessible from outside your project.

Service types summary



You've seen how ClusterIP Services can be used within the cluster to provide a stable endpoint that allows Pods to connect to other Pods without the risk of the IP address changing.

You learned how NodePort Services extend the ClusterIP Services concept to expose and then bind a port number on each of the nodes in the cluster to the Service. This allows you to access resources within the cluster from external resources such as GCP compute instances.

Finally, you saw how the LoadBalancer Service type implementation in GKE improved on the NodePort Service by using the cloud provider API to create a GCP network load balancer for traffic distribution across the nodes.

Now let's look at a way to group Services together using the Ingress resource.