Kubernetes

Understanding Pod Communication and External Access to Services in Kubernetes

In this article, we will explore how containers inside Pods communicate with each other and how external users can access services or applications running within a Kubernetes cluster.

Service in Kubernetes

A Service in Kubernetes acts as an abstraction that defines how to access an application running in a set of Pods. It enables communication between Pods and provides a stable endpoint for accessing applications.

When creating a Service resource, we define the type of Service we want to use, specifying how we intend to access our application. Kubernetes offers four types of Services for accessing applications:

---

1. ClusterIP

- Description: Exposes the Service internally within the cluster. This type is the default and is used for communication between Pods or components inside the cluster.

- Access: Only accessible to users or applications with access to the cluster.

---

2. NodePort

- Description: Exposes the Service on a static port (NodePort) on each Node in the cluster. This allows access from the same network as the Kubernetes nodes.
- Access: Available to users on the same network, such as within an office or a local testing environment.

---

3. LoadBalancer

- Description: Exposes the Service to external users by integrating with a cloud provider's load balancer (e.g., AWS ELB, Azure Load Balancer, GCP Load Balancer).
- Access: Direct external access via a public IP or DNS assigned by the cloud provider.

---

4. ExternalName

- Description: Maps the Service to an external DNS name. Instead of routing traffic to Pods, it redirects traffic to an external service.
- Access: Resolves to the specified DNS name.

Summary

- ClusterIP provides access inside the cluster only.
- NodePort, LoadBalancer, and ExternalName allow access outside the cluster.

Let’s look into the manifest file of service and ingress for better understanding.

```yaml
apiVersion: v1

kind: Service

metadata:

 name: nginx-service

 namespace: nginx

spec:

 selector:

   app: nginx

 ports:

 - port: 8000   # Outside Port where we want to access service

   targetPort: 80  # insider port of Pod

 type: ClusterIP
```

Ingress:

A Service allows us to expose a single application to the outside world. However, if there are multiple services in the cluster, things become more complex.

One option could be to use the **LoadBalancer** service type for each application. However, this approach can become expensive, as cloud providers charge for each LoadBalancer created. Additionally, Services alone cannot provide advanced features like **reverse proxying** or **path-based routing**.

To solve this problem, the concept of **Ingress** was introduced by the CNCF (Cloud Native Computing Foundation) in collaboration with load balancer providers like NGINX.

An **Ingress Controller** acts as a reverse proxy and provides advanced routing capabilities such as:

- Path-based routing (e.g., directing `/api` to one service and `/app` to another).
- TLS/SSL termination.
- Handling multiple services through a single external LoadBalancer.

This makes it easier and more cost-effective to manage multiple applications and routes within a Kubernetes cluster.

Let's look at the ingress manifest file. Here first don't forget to install the ingress controller. Visit this https://kind.sigs.k8s.io/examples/ingress/deploy-ingress-nginx.yaml .

```yaml
kind: Ingress
apiVersion: networking.k8s.io/v1
metadata:
 name: ingress-nginx-app #name of ingress
 namespace: nginx
 labels:
   name: ingress   # label for ingress
spec:
 rules:
 - http:
     paths:
     - pathType: Prefix   # First service which is nginx
       path: "/"
       backend:
         service:
           name: nginx-service
           port:
             number: 80
     - pathType: Prefix   # Second service my app
       path: "/"
       backend:
         service:
           name: myapp-service
           port:
             number: 8080
```

Conclusion 🙂

In today's article, we explored:

- **What is a Service in Kubernetes?** 🛠️
  - How it helps in accessing applications within and outside the cluster.
- **What is Ingress?** 🌐
  - Why Ingress is used and how it simplifies routing traffic to multiple services with advanced features like path-based routing and TLS termination.

We also looked at example manifest files for both Service and Ingress to understand their configurations. 📝
With this knowledge, you're now better equipped to manage application traffic in Kubernetes efficiently. 🚀