Kubernetes Ingress Controllers

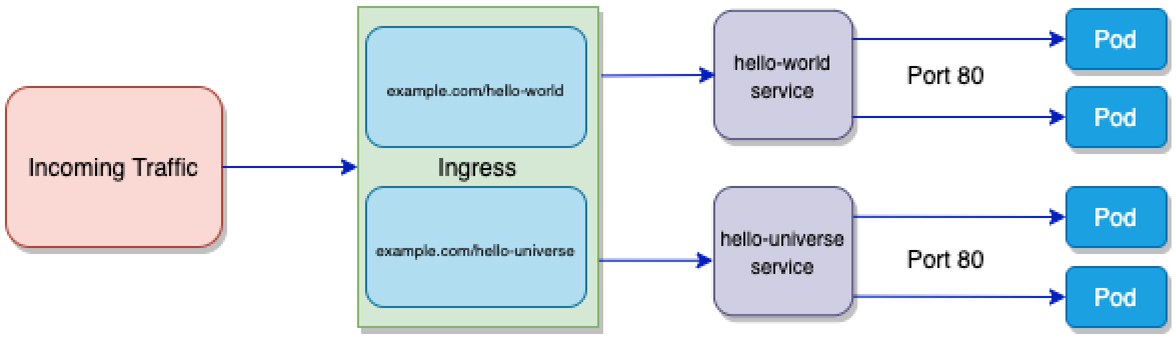
Overview

In this article you'll learn about:

- What is Kubernetes Ingress
- Why Kubernetes Ingress was created
- Differences between Kubernetes Ingress, NodePort, and Load Balancers
- Installing an Ingress Controller
- Using Kubernetes Ingress to expose Services via externally reachable URLs
- Using Kubernetes Ingress to load balance traffic
- Using Kubernetes Ingress to terminate SSL / TLS
- Using Kubernetes Ingress to route traffic to multiple hostnames at the same IP

What is Kubernetes Ingress

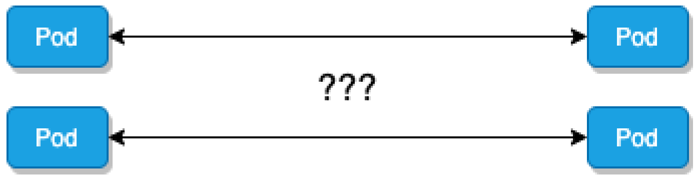
Kubernetes Ingress is an API object that describes the routing rules for traffic (typically HTTP or HTTPS) into an application running within a Kubernetes cluster. Ingress was created to solve a routing problem in Kubernetes: an application service runs within a cluster on one or more container instances, which use virtual IP addresses for "cluster internal" communication. However, external clients that reside outside the cluster need a stable and routable IP endpoint to communicate with the service. Kubernetes Ingress was created to make this easy: it is a Kubernetes API object that maps an application to a stable, externally addressable HTTP or HTTPS URL.



Why Kubernetes Ingress was created: The routing problem explained

Pods are the smallest unit of application computing in Kubernetes: they represent a small, highly cohesive unit of an application. Pods are also dynamic: they are created and destroyed to reflect the state of the cluster and workload requirements. So, the set of Pod container instances running at one moment in time can be different in the next moment.

So, this leads to a problem: if one set of Pods (imagine a frontend Pod) needs to communicate with another set of Pods (image a backend Pod), how do the frontends find out and keep track of which IP address to connect to as container instances come and go?



To solve this, Kubernetes created the "Service" abstraction: a Service is an external interface to a logical set of Pods. Within a cluster, Kubernetes uses IP proxy techniques to ensure that traffic intended for a Service is routed to one of the concerned Pod container instances. However, because this uses a "virtual IP address" internal to the cluster, external services need a different way to access the Service endpoint. Kubernetes Ingress was created to solve this problem.

Differences between Kubernetes Ingress, NodePort and Load Balancers

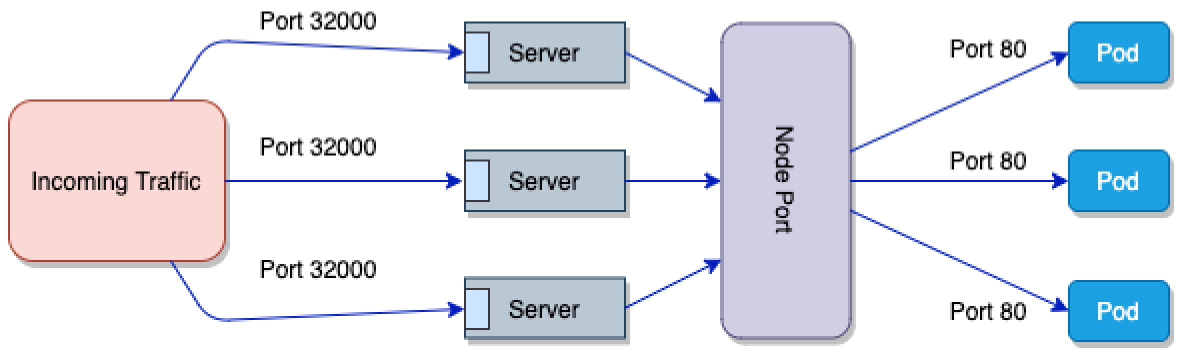
When the notion of a "Service" was first added to Kubernetes, two early mechanisms were incorporated to enable external access to the Service: NodePort, and Load Balancers. Ingress was added in a later revision, for reasons we will describe below.

NodePort was the first ingress. It works by exposing a Service to external clients on each node's IP, at a statically defined port – the NodePort. External clients can reach the service at <NodeIP>:NodePort. Within the cluster, the Kubernetes control plane allocates a port from a configurable range (typically 30000-32767) and each node proxies that port into the Service.

NodePorts are a convenient addressing mechanism, especially in dev/test; but have significant downsides in comparison to ingress:

1. Clients are expected to know node IP addresses to reach the service. In contrast, Ingress makes it easier to use a DNS entry, and hiding cluster internals such as the number of nodes and their IP addresses
2. NodePort requires opening up external port access to every node in the cluster for each service, which increases the complexity of securely managing the cluster – a cluster with more than a handful of services will look like a block of swiss cheese in terms of the number of holes into that cluster
3. Troubleshooting and reasoning about network access quickly becomes very complex with a proliferation of services, clusters and NodePort

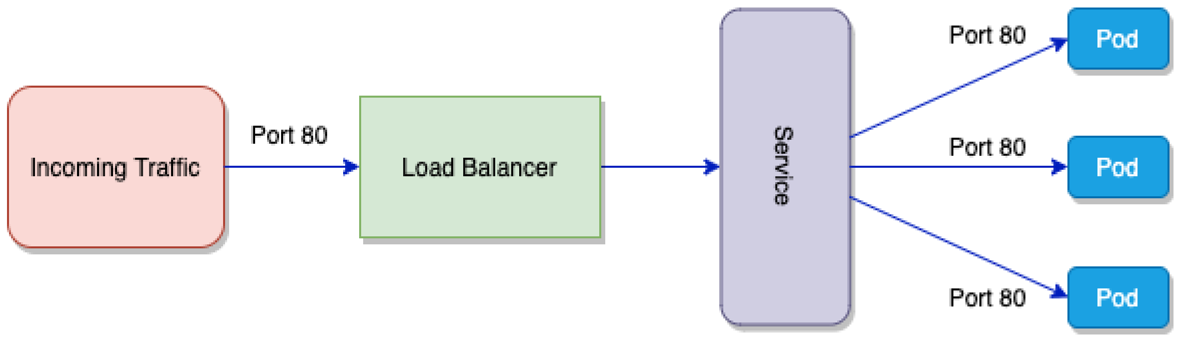
Here is how a Nodeport works:



To overcome these NodePort limitations, Kubernetes also provides an option to rely on a cloud provider's Load Balancer service.

A load balancer allows us to use a single IP address and more familiar ports. Instead of keeping a list of the nodes we need to hit, as well as the port associated with the NodePort, we can use the IP of the load balancer.

Here is how a Load Balancer works:



For large production environments, we need something that will allow us to implement more complicated use cases. We may need to set up routing based on specific content, or features offered by an application, which will route to different sets of pods and services. A more complicated configuration could end up costing a lot, especially if we are using a load balancer per application.

Ingress Controllers provides this feature set.

Installing an Ingress Controller

To utilize Kubernetes Ingress you will need to deploy an Ingress Controller. You have a vast choice when it comes to choosing an Ingress Controller. Here is a list of all the ones that Kubernetes supports. The three most popular Ingress Controllers deployed on Kubernetes are:

- Nginx
- Traefik
- HAProxy

Using Kubernetes Ingress to expose Services via externally reachable URLs

Now that an Ingress Controller has been set up, we can start using Ingress Resources which are another resource type defined by Kubernetes. Ingress Resources are used to update the configuration within the Ingress Controller. Using Nginx as an example, there will be a set of pods running an Nginx instance that is monitoring Ingress Resources for changes so that it can update and then reload the configuration.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: hello-world
 annotations:
   nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
  - host: test.example.com
    http:
      paths:
      - path: /hello-world
        pathType: Prefix
        backend:
          service:
            name: hello-world
            port:
              number: 8080
```

This is an example of a basic Ingress Resource. Within the metadata, we are selecting which ingress class we want to use, in our case ingress-nginx, which will let the Nginx Ingress controller know that it needs to monitor and make updates. The ingress. the class option is also used to specify which Ingress Controller should monitor the resource if we are using multiple controllers.

The spec defines what we expect to be configured. Within this section we are going to define a rule or set of rules, the host the rules will be applied to, whether or not it is HTTP or HTTPS traffic as well as the path we are watching, and the internal service and port where the traffic will be sent.

Traffic that is sent to test.example.com/hello-world will be directed to the service hello-world on port 8080. This is a basic setup that provides layer 7 functionality. Now we can use a single DNS entry for a host, or set of hosts, and provide path-based load balancing. Next, we will expand on this by adding another path for routing traffic.

Using Kubernetes Ingress to load balance traffic

Now we can add another path. This allows us to load balance between different backends for our application. We can split up traffic and send it to different service endpoints and deployments based on the path. This can be beneficial for endpoints that receive more traffic, as we can scale a single deployment for /hello-world without having to scale up /hello-universe.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hello-world-and-universe
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target:
spec:
  ingressClassName: nginx
  rules:
  - host: test.example.com
    http:
      paths:
      - path: /hello-world
        pathType: Prefix
        backend:
          service:
            name: hello-world
            port:
              number: 8080
      - path: /hello-universe
        pathType: Prefix
        backend:
          service:
            name: hello-universe
            port:
              number: 8080
```

Our example is similar to the basic setup except for the fact that we have added a second path. Behind the second path are another deployment and service.

Using Kubernetes Ingress to terminate SSL / TLS

Setting up services using HTTP is great but what we really need is HTTPS. To do this we need to install a Kubernetes Custom Resource Definition called Cert-Manager. Cert-Manager automates TLS certificate management using multiple providers.

Once we have installed Cert-Manager we can create certificates with annotations in the Ingress Resource, or manually by creating our own Certificates. In our example, we are going to use annotations as it simplifies the configuration.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: hello-world
 annotations:
   nginx.ingress.kubernetes.io/rewrite-target: /
   nginx.ingress.kubernetes.io/force-ssl-redirect: "true"
   cert-manager.io/cluster-issuer: letsencrypt-prod
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - test.example.com
    secretName: test-example-tls
  rules:
  - host: test.example.com
    http:
      paths:
      - path: /hello-world
        pathType: Prefix
        backend:
          service:
            name: hello-world
            port:
              number: 8080
```

This configuration takes advantage of a couple of annotations. Nginx provides options that can be configured using annotations. We are using force-ssl-redirect to redirect HTTP traffic to HTTPS. The Cert-Manager annotation cluster-issuer lets us select which issuer we want to use for our certificate. In the host's section, we are specifying the host we want to route traffic for, a subdomain of http://example.com called test. A secret name is selected to store the certificate information which is automatically configured since we are using a Cert-Manager annotation.

Using Kubernetes Ingress to route HTTP traffic to multiple hostnames at the same IP address

Multiple DNS records can be pointed at the same public IP address used for our LoadBalancer service. We can use a different ingress resource per host, which allows us to control traffic with multiple hostnames while using the same external IP for A Record creation.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
name: hello-world-prod
annotations:
  nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
  - host: test.example.com
   http:
     paths:
     - path: /hello-world
       pathType: Prefix
       backend:
         service:
           name: hello-world-test
           port:
             number: 8080
  - host: prod.example.com
    http:
      paths:
      - path: /hello-world
        pathType: Prefix
        backend:
          service:
            name: hello-world-prod
            port:
              number: 8080
```

In our example, we have added a prod version of hello-world. We are using the hostname prod.example.com and are pointing traffic to a new service called hello-world-prod. Traffic is going to come in through the same Load Balancer IP address and will be routed based on the hostname used and the path.

Conclusion

Using an Ingress Controller will enable you to easily and securely route traffic to your applications running as Kubernetes services. In this article, we went over some of the different Kubernetes service types that can be used for routing and benefits. We also provided a few examples for getting started.