**Kubernetes Ingress Controllers and DNS Configuration Documentation**

**Overview**

In this document, we will explain how to set up multiple **Ingress controllers** in Kubernetes and configure **DNS** to resolve a subdomain to the external IPs of those ingress controllers. This will allow you to expose your application securely and route traffic effectively using Ingress resources.

**1. Setting Up Multiple Ingress Controllers in Kubernetes**

**Step 1: Deploying Multiple Ingress Controllers**

If you wish to have multiple ingress controllers in your Kubernetes cluster (for redundancy and high availability), follow these general steps:

1. **Install NGINX Ingress Controller**: You can install the NGINX ingress controller using Helm or kubectl. For example, to install NGINX ingress via Helm:

bash

Copy

helm install ingress-nginx ingress-nginx/ingress-nginx

1. **Create Multiple Ingress Controllers**: To deploy multiple instances of the ingress controller, you can replicate the existing ingress controller configuration or scale it horizontally.

bash

Copy

kubectl scale deployment ingress-nginx-controller --replicas=3 -n ingress-nginx

1. **Expose Each Ingress Controller**: Each ingress controller will need to be exposed via a **LoadBalancer** service. You can define the LoadBalancer service as shown below. For example:

yaml

Copy

apiVersion: v1

kind: Service

metadata:

name: ingress-nginx-controller-1

labels:

app: ingress-nginx

spec:

ports:

- protocol: TCP

port: 80

targetPort: 80

selector:

app: ingress-nginx

type: LoadBalancer

Repeat this service definition for each ingress controller, and Kubernetes will assign an **external IP** to each LoadBalancer service.

**2. Configuring DNS to Resolve Subdomain to Ingress Controller IPs**

**Step 1: Determine External IPs of Ingress Controllers**

After creating **LoadBalancer** services for each ingress controller, Kubernetes will assign external IPs. You can retrieve these IPs using the following command:

bash

Copy

kubectl get services -n ingress-nginx

Example output:

txt

Copy

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

ingress-nginx-controller-1 LoadBalancer 10.96.0.1 203.0.113.10 80:31969/TCP 3h

ingress-nginx-controller-2 LoadBalancer 10.96.0.2 203.0.113.11 80:31969/TCP 3h

ingress-nginx-controller-3 LoadBalancer 10.96.0.3 203.0.113.12 80:31969/TCP 3h

* **Ingress Controller 1 External IP**: 203.0.113.10
* **Ingress Controller 2 External IP**: 203.0.113.11
* **Ingress Controller 3 External IP**: 203.0.113.12

**Step 2: Configure DNS for Subdomain**

In your DNS management console, create **A records** for your subdomain (my-app.example.com) that resolve to each of the ingress controller external IPs.

Example DNS configuration:

txt

Copy

my-app.example.com A 203.0.113.10 # Ingress Controller 1 IP

my-app.example.com A 203.0.113.11 # Ingress Controller 2 IP

my-app.example.com A 203.0.113.12 # Ingress Controller 3 IP

This allows **round-robin DNS** resolution, where the DNS resolver will distribute requests across the available ingress controllers.

**Step 3: Verify DNS Resolution**

After configuring the DNS records, use nslookup or dig to verify that my-app.example.com resolves to the correct IPs:

bash

Copy

nslookup my-app.example.com

This should return one of the IPs configured above (e.g., 203.0.113.10, 203.0.113.11, or 203.0.113.12).

**3. Configuring Ingress Resource**

**Step 1: Define the Ingress Resource**

Create an **Ingress resource** that will route the traffic to the appropriate backend service based on the incoming request's **host** (subdomain).

Here’s an example of the Ingress resource for your app:

yaml

Copy

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: my-app-ingress

spec:

rules:

- host: my-app.example.com # This should match your DNS subdomain

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: my-app-service # The name of your service

port:

number: 80

* **host**: The **subdomain** (my-app.example.com) that will be used by the ingress controller to route traffic.
* **service**: The **backend service** (my-app-service) where the traffic will be forwarded inside the cluster.

**Step 2: Apply the Ingress Resource**

Apply the above Ingress YAML file to your cluster:

bash

Copy

kubectl apply -f my-app-ingress.yaml

**4. Traffic Flow**

1. **User Request**: A user visits http://my-app.example.com.
2. **DNS Resolution**: The DNS resolver resolves my-app.example.com to one of the ingress controller IPs (e.g., 203.0.113.10, 203.0.113.11, or 203.0.113.12) using round-robin DNS or based on caching.
3. **Ingress Controller**: The request reaches one of the ingress controllers (based on the resolved IP). The ingress controller checks the **host** (my-app.example.com) and matches it to the corresponding **Ingress resource**.
4. **Ingress Routing**: The ingress controller routes the traffic to the appropriate **service** (e.g., my-app-service).
5. **Service and Pod**: The service forwards the traffic to the correct **pod** running your application.

**5. High Availability and Fault Tolerance**

If one of the ingress controllers goes down, traffic can still be routed to the remaining ingress controllers because DNS will resolve to one of the healthy IPs.

* **Health Checks**: Make sure you configure health checks for your **LoadBalancer** services so that unhealthy ingress controllers are automatically removed from DNS rotation.
* **DNS Caching**: Keep in mind that **DNS caching** might affect the distribution of traffic. Clients might continue to hit an old IP until the DNS cache expires.

**6. Conclusion**

* **Multiple Ingress Controllers**: You can expose multiple ingress controllers using LoadBalancer services, which each get a unique external IP.
* **DNS Configuration**: Configure your DNS provider to resolve the subdomain (my-app.example.com) to the external IPs of the ingress controllers. This allows round-robin DNS and traffic distribution across multiple controllers.
* **Ingress Resource**: Define an Ingress resource to route incoming requests based on the host (subdomain) and path, forwarding them to the correct backend service.
* **High Availability**: Using multiple ingress controllers increases availability and fault tolerance. DNS ensures that traffic is distributed across multiple ingress controllers.

This document provides a comprehensive overview of how to set up multiple ingress controllers, configure DNS for your subdomain, and route traffic to your app in a Kubernetes environment.

You can save the above content to a file named Kubernetes\_Ingress\_and\_DNS\_Configuration.md or Kubernetes\_Ingress\_and\_DNS\_Configuration.txt for easy reference.