Ingress Controllers and Resources

Ingress

Everything in Kubernetes is an object. Every object has a controller that watches it for changes and acts accordingly.

Ingress is an API object that manages external access to services in a cluster. Typically, it exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is defined by rules on the ingress resource.

An ingress does not expose arbitrary ports and protocols. Exposing anything other than HTTP or HTTPS to the internet uses a service type such as NodePort or LoadBalancer.

Ingress Controllers

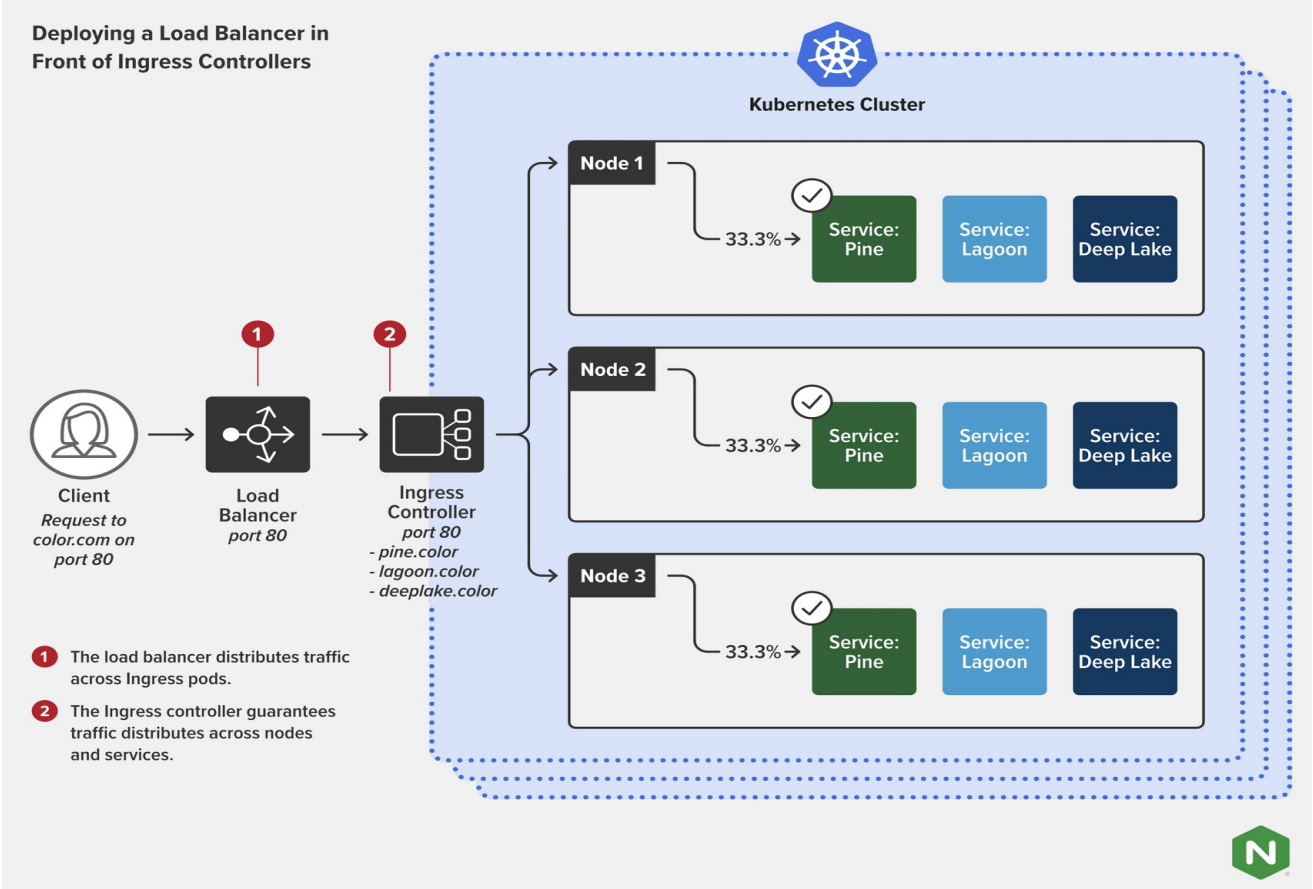
An ingress controller watches for ingress objects, which accept traffic at some host or forward it to a service object such as ClusterIP. Essentially ingress controller is a reverse proxy app that lists on ports 80 and 443 as it monitors the server API server for specific DNS addresses. It is responsible for fulfilling Ingress requests.

For a full list of different types of ingress controller click the following:

https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/

The following example is of an Ingress Controller with a LoadBalancer service type, deployed in front of our cluster. The client will request web traffic on port 80 and according to their DNS name, will be directed to one of three different nodes and 9 different pods. This is the typical setup of an Nginx Ingress Controller. The key

advantage to ingress controllers is seamless integration with cloud loadbalancers such as GCP GLBC.



Ingress Resource

This is a minimal ingress resource:

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx-example
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80
```

Hands On – Setup Ingress with the Nginx Ingress Controllers

I. Enable the Ingress controller

To enable the nginx Ingress controller, we need to enable it on the node.

```
dominickhrndz314@cloudshell:~$ minikube addons enable ingress
  - Using image k8s.gcr.io/ingress-nginx/kube-webhook-certgen:v1.1.1
  - Using image k8s.gcr.io/ingress-nginx/controller:v1.0.4
  - Using image k8s.gcr.io/ingress-nginx/kube-webhook-certgen:v1.1.1
* Verifying ingress addon...
* The 'ingress' addon is enabled
dominickhrndz314@cloudshell:~$
```

Verify the services is working – it can take up to a minute for all 3 pods to be up and running.

```
dominickhrndz314@cloudshell:~$ kubectl get pods -n ingress-nginx
NAME                                        READY   STATUS      RESTARTS   AGE
ingress-nginx-admission-create--1-6pg64     0/1     Completed   0          2m25s
ingress-nginx-admission-patch--1-5gcn4      0/1     Completed   1          2m25s
ingress-nginx-controller-5f66978484-ldmmp   1/1     Running     0          2m25s
dominickhrndz314@cloudshell:~$
```

II. Deploy Hello-World app

Instead of writing an app, we're going to deploy an existing hello-world app from Google.

```
dominickhrndz314@cloudshell:~$ kubectl create deployment web --image=gcr.io/google-samples/hello-app:1.0
deployment.apps/web created
dominickhrndz314@cloudshell:~$ kubectl get deployment web
NAME   READY   UP-TO-DATE   AVAILABLE   AGE
web    1/1     1            1           24s
dominickhrndz314@cloudshell:~$
```

Once you've verified it exists, take a look at the deployment in detail. Use the kubectl describe deployment web commands to view information about your deployment. **Notice there is no port or host port.**

After you've described the deployment, check out the pod as well to ensure its in running status.

Now we need to expose the deployment to port 8080 and then we need to verify the service is created. Notice how our service is running NodePort.

```
dominickhrndz314@cloudshell:~$ kubectl expose deployment web --type=NodePort --port=8080
service/web exposed
dominickhrndz314@cloudshell:~$ kubectl get service web
NAME   TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
web    NodePort   10.107.16.63   <none>        8080:31086/TCP   107s
dominickhrndz314@cloudshell:~$
```

Test your new 'web' service by trying to access your app from the Minikube node. To do so execute the minikube service web --url command. If you receive the following output, your web app and Ingress controller are communicating successfully.

```
dominickhrndz314@cloudshell:~$ minikube service web --url
http://192.168.49.2:31086
dominickhrndz314@cloudshell:~$
```

III. Create an Ingress

Now lets create an Ingress resource that sends traffic to our service via the new application.

service/networking/example-ingress.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  rules:
    - host: hello-world.info
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: web
                port:
                  number: 8080
```

Create the deployment, apply it to the cluster, and verify. Notice that in this deployment our IP address is listed as *localhost*. This is exclusive to Minikube, but in a regular environment this IP would be different. To get the IP we need on minikube we can execute the minikube ip command.

```
dominickhrndz314@cloudshell:~$ cat ingress-policy.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  rules:
    - host: hello-world.info
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: web
                port:
                  number: 8080
dominickhrndz314@cloudshell:~$ kubectl apply -f ingress-policy.yaml
ingress.networking.k8s.io/example-ingress unchanged
dominickhrndz314@cloudshell:~$ kubectl get ingress
NAME              CLASS   HOSTS              ADDRESS     PORTS   AGE
example-ingress   nginx   hello-world.info   localhost   80      86s
dominickhrndz314@cloudshell:~$
```

Issue the command and save the IP address.

```
dominickhrndz314@cloudshell:~$ minikube ip
192.168.49.2
```

We now need take our IP address and associated with a hostname to create a DNS entry for our Ingress. In GCP cloudshell, navigate to the file path *etc*/hosts. Add the entry <your minikube IP> hello-world.info

```
dominickhrndz314@cloudshell:/home$ cd ..
dominickhrndz314@cloudshell:/$ ls
bin   get-pip.py           lib      libx32      mysql-apt-config_0.8.17-1_all.deb  root  sys              usr
boot  google               lib32    linux-amd64 opt                                run   tinkey.bat       var
dev   home                 lib64    media       packages-microsoft-prod.deb        sbin  tinkey_deploy.jar
etc   install_kustomize.sh  libgit2  mnt         proc                               srv   tmp
dominickhrndz314@cloudshell:/$ cd etc/
dominickhrndz314@cloudshell:/etc$ nano hosts
dominickhrndz314@cloudshell:/etc$ sudo nano hosts
```

```
  GNU nano 5.4
# Kubernetes-managed hosts file.
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
fe00::0 ip6-mcastprefix
fe00::1 ip6-allnodes
fe00::2 ip6-allrouters
172.17.0.4      cs-357356478907-default
192.168.49.2 hello-world.info
```

Now let’s verify that the Ingress controller is directing traffic. If you receive the following message then you have completed this training.

```
dominickhrndz314@cloudshell:~$ curl hello-world.info
Hello, world!
Version: 1.0.0
Hostname: web-79d88c97d6-bzlzq
dominickhrndz314@cloudshell:~$
```