Kubernetes

Lab 9 – Ingress

In Kubernetes clusters, Services have IP addresses which are only routable within the cluster. Kubernetes Services provide an excellent abstraction for microservices interacting within the cluster, however many applications need to expose a subset of Services to clients outside the cluster.

An Ingress is a collection of rules that allow traffic from outside the cluster to reach the services inside the cluster. Ingresses give services externally-reachable URLs and can load balance traffic, terminate SSL and more. Users request ingress by creating an Ingress resource.

Ingress controllers implement the Ingress specified by the Ingress resource (which is after all just a json document). An ingress controller can configure load balancers, edge routers and/or other frontends to process inbound traffic.

Ingress resources are in Beta and were introduced in Kubernetes 1.1. Because Ingress resources are implemented by Ingress Controllers you must run an Ingress Controller within the cluster (often as a deployment) to implement the Ingress resource, Kubernetes does not provide a default Ingress controller. Creating an Ingress resource without a running Ingress Controller will have no affect. This is unlike other types of controllers, which typically run as part of the kube-controller-manager binary, and which are typically started automatically as part of cluster creation.

The Kubernetes community supports and maintains two Ingress Controllers:

- GCE
- Nginx

Google Kubernetes Engine (GKE) deploys the GCE ingress controller on the master nodes of clusters. The GKE Ingress Controller configures a GCE loadbalancer in response to the creation of Ingress resources.

Several other companies also support and maintain Ingress Controllers:

- F5 Networks provides an F5 BIG-IP Ingress Controller
- Kong offers community or commercial support for the Kong Ingress Controller
- Containous offers support for the Traefik Ingress Controller
- NGINX, Inc. offers support for the NGINX Ingress Controller

Custom Ingress Controllers can written by users with special needs.

Ingress resources can select from multiple Ingress Controllers, allowing a cluster to run many Ingress Controllers concurrently.

In this lab you will implement and test an Nginx Ingress Controller (IC).

Step 1 - Setup a Namespace for the IC

The Nginx Ingress Controller is often configured to run in its own namespace. Create an nginx namespace for your IC:

```
user@ubuntu:~$ kubectl get ns

NAME              STATUS   AGE
default           Active   22h
kube-node-lease   Active   22h
kube-public       Active   22h
kube-system       Active   22h

user@ubuntu:~$ kubectl create namespace nginx-ingress

namespace/nginx-ingress created

user@ubuntu:~$ kubectl get ns

NAME              STATUS   AGE
default           Active   22h
```

```
kube-node-lease   Active   22h
kube-public       Active   22h
kube-system       Active   22h
nginx-ingress     Active   3s

user@ubuntu:~$
```

Step 2 - Create a Service Account

An Ingress Controller needs to monitor the api-server for new Ingress resources. For this to work we need to create a service account for the Ingress Controller (IC) and then grant that account permissions to the Ingress Resource object type.

Create the service account:

```
user@ubuntu:~$ kubectl create serviceaccount nginx-ingress -n nginx-ingress

serviceaccount/nginx-ingress created

user@ubuntu:~$ kubectl get sa -n nginx-ingress

NAME            SECRETS   AGE
default         1         24s
nginx-ingress   1         3s

user@ubuntu:~$
```

Display the service account details:

```
user@ubuntu:~$ kubectl describe sa nginx-ingress -n nginx-ingress

Name:                nginx-ingress
Namespace:           nginx-ingress
Labels:              <none>
Annotations:         <none>
Image pull secrets:  <none>
Mountable secrets:   nginx-ingress-token-jgdg9
Tokens:              nginx-ingress-token-jgdg9
Events:              <none>

user@ubuntu:~$
```

Examine the token generated for the service account:

```
user@ubuntu:~$ kubectl describe secret nginx-ingress-token-jgdg9 -n nginx-ingress

Name:         nginx-ingress-token-jgdg9
Namespace:    nginx-ingress
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: nginx-ingress
              kubernetes.io/service-account.uid: bf127455-34c8-48d7-9e38-f7c19ac1824f

Type:  kubernetes.io/service-account-token

Data
====
ca.crt:     1025 bytes
namespace:  13 bytes
token:
eyJhbGciOiJSUzI1NiIsImtpZCI6IjM0bGpPY0hIZ3h3S3JDWXRKdzVYTmhOYU04NzZkUWUxaTF5VHlZamdJTVUifQ.eyJpc
3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiO
iJuZ2lueC1pbmdyZXNzIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6Im5naW54LWluZ3Jlc
3MtdG9rZW4tamdkZzkiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC5uYW1lIjoibmdpb
ngtaW5ncmVzcyIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6ImJmMTI3NDU1L
TM0YzgtNDhkNy05ZTM4LWY3YzE5YWMxODI0ZiIsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDpuZ2lueC1pbmdyZXNzO
m5naW54LWluZ3Jlc3MifQ.Bbma1QXGdX6sjNecUvgqvr4_k1GpFBoOC3_o6uN4tBcWvpKlsCb0grKdJ45gnbP7k2hjV13BHu
```

```
RDwgTpzY2P9ZsUm1xM56pAeNtyw_PvasT6EZrk-
rsWYYW7raecHXNI8BcpPaxU0GRI_I6hbxJHZd9EltplPMQFF4Mu7kVoTYIzeOGBeKU3Nu1KnaUuf0l8W_2-
_6kiKfh2H5NwpVs_kTsIobcidXH8NfJHDOj1jrd1p91YWgMiqadB7Xd4aDEw4NRQAF3aksuyX5sxnmEGu_zSnKnEZRaOuNg_
S3BBpvfWVhoyvfz9HT5X3RFhJGw_FLQ12W26i8pwKDCMylrwrg

user@ubuntu:~$
```

We also need to provide the IC with a TLS cert. Create a secret to house a generic TLS cert:

```
user@ubuntu:~$ mkdir ~/ingress && cd ~/ingress

user@ubuntu:~/ingress$ nano ic-tls.yaml

user@ubuntu:~/ingress$ cat ic-tls.yaml
```

```
apiVersion: v1
kind: Secret
metadata:
  name: default-server-secret
  namespace: nginx-ingress
type: Opaque
data:
  tls.crt:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUN2akNDQWFZQ0NRREFPRjl0THNhWFhEQU5CZ2txaGtpRzl3MEJBUXNGQU
RBaE1SOHdIUVlEVlFRRERCWk8KUjBsT1dFbHVaM0psYzNORGIyNTBjbTlzYkdWeU1CNFhEVEU0TURreE1qRTRNRE16TlZvWERU
SXpNRGt4TVRFNApNRE16TlZvd0lURWZNQjBHQTFVRUF3d1dUa2RKVGxoSmJtZHlaWE56UTI5dWRISnZiR3hsY2pDQ0FTSXdEUV
lKCktvWklodmNOQVFFQkJRQURnZ0VQQURDQ0FRb0NnZ0VCQUwvN2hIUEtFWGRMdjNyaUM3QlBrMTNpWkt5eTlyQ08KR2xZUXYy
K2EzUDF0azIrS3YwVGF5aGRCbDRrcnNUcTZzZm8vWUk1Y2Vhbkw4WGM3U1pyQkVRYm9EN2REbWs1Qgo4eDZLS2xHWU5IWlg0Rm
5UZ0VPaStlM2ptTFFxRlBSY1kzVnNPazFFeUZBL0JnWlJVbkNHZUtGeERSN0tQdGhyCmtqSXVuektURXUyaDU4Tlp0S21ScUJH
dDEwcTNRYzhZT3ExM2FnbmovUWRjc0ZYYTJnMjB1K1lYZDdoZ3krZksKWk4vVUkxQUQ0YzZyM1lma1ZWUmVHd1lxQVp1WXN2V0
RKbW1GNWRwdEMzN011cDBPRUxVTExSakZJOTZXNXIwSAo1TmdPc25NWFJNV1hYVlpiNWRxT3R0SmRtS3FhZ25TZ1JQQVpQN2Mw
QjFQU2FqYzZjNGZRVXpNQ0F3RUFBVEFOCkJna3Foa2lHOXcwQkFRc0ZBQU9DQVFFQWpLb2tRdGRPcEsrTzhibWVPc3lySmdJSX
JycVFVY2ZOUitjb0hZVUoKdGhrYnhITFMzR3VBTWI5dm15VExPY2xxeC9aYzJPblEwMEJCLzlTb0swcitFZ1U2UlVrRWtWcitT
TFA3NTdUWgozZWI4dmdPdEduMS9ienM3bzNBaS9kclkrcUI5Q2k1S3lPc3FHTG1US2xFaUtOYkcyR1ZyTWxjS0ZYQU80YTY3Ck
lnc1hzYktNbTQwV1U3cG9mcGltU1ZmaXFSdkV5YmN3N0NYODF6cFErUyt1eHRYK2VBZ3V0NHh3VlI5d2IyVXYKelhuZk9HbWhW
NThDd1dIQnNKa0kxNXhaa2VUWXdSN0diaEFMSkZUUkk3dkhvQXprTWIzbjAxQjQyWjNrN3RXNQpJUDFmTlpIOFUvOWxiUHNoT2
1FRFZkdjF5ZytVRVJxbStGSis2R0oxeFJGcGZnPT0KLS0tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo=
  tls.key:
LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcEFJQkFBS0NBUUVBdi91RWM4b1JkMHUvZXVJTHNFK1RYZUprck
xMMnNJNGFWaEMvYjVyYy9XMlRiNHEvClJOcktGMEdYaVN1eE9ycXgrajlnamx4NXFjdnhkenRKbXNFUkJ1Z1B0ME9hVGtIekhv
b3FVWmcwZGxmZ1dkT0EKUTZMNTdlT1l0Q29VOUZ4amRXdzZUVVRJVUQ4R0JsRlNjSVo0b1hFTkhzbysyR3VTTWk2Zk1wTVM3YU
hudzFtMApxWkdvRWEzWFNyZEJ6eGc2clhkcUNlUDlCMXl3VmRyYURiUzc1aGQzdUdETDU4cGszOVFqVUFQaHpxdmRoK1JWClZG
NGJCaW9CbTVpeTlZTW1hWVhsMm0wTGZzeTZuUTRRdFFzdEdNVWozcGJtdlFmazJBNnljeGRFeFpkZFZsdmwKMm82MjBsMllxcH
FDZEtCRThCay90elFIVTlKcU56cHpoOUJUTXdJREFRQUJBb0lCQVFDZklHbXowOHhRVmorNwpLZnZJUXQwQ0YzR2MxNld6eDhV
Nml4MHg4Mm15d1kxUUNlL3BzWE9LZlRxT1h1SENyUlp5TnUvZ2IvUUQ4bUFOCmxOMjRZTWl0TWRJODg5TEZoTkp3QU5OODJDeT
czckM5bzVvUDlkazAvYzRIbjAzSkVYNzZ5QjgzQm9rR1FvYksKMjhMNk0rdHUzUmFqNjd6Vmc2d2szaEhrU0pXSzBwV1YrSjdr
UkRWYmhDYUZhNk5nMUZNRWxhTlozVDhhUUtyQgpDUDNDeEFTdjYxWTk5TEI4KzNXWVFIK3NYaTVGM01pYVNBZ1BkQUk3WEh1dX
FET1lvMU5PL0JoSGt1aVg2QnRtCnorNTZud2pZMy8yUytSRmNBc3JMTnIwMDJZZi9oY0IraVlDNzVWYmcydVd6WTY3TWdOTGQ5
VW9RU3BDRkYrVm4KM0cyUnhybnhBb0dCQU40U3M0ZVlPU2huMVpQQjdhTUZsY0k2RHR2S2ErTGZTTXFyY2pOZjJlSEpZNnhubm
xKdgpGenpGL2RiVWVTbWxSekR0WkdlcXZXaHFISy9iTjIyeWJhOU1WMDlRQ0JFTk5jNmtWajJTVHpUWkJVbEx4QzYrCk93Z0wy
ZHhKendWelU0VC84ajdHalRUN05BZVpFS2FvRHFyRG5BYWkyaW5oZU1JVWZHRXFGKzJyQW9HQkFOMVAKK0tZL0lsS3RWRzRKSk
lQNzBjUis3RmpyeXJpY05iWCtQVzUvOXFHaWxnY2grZ3l4b25BWlBpd2NpeDN3QVpGdwpaZC96ZFB2aTBkWEppc1BSZjRMazg5
b2pCUmpiRmRmc2l5UmJYbyt3TFU4NUhRU2NGMnN5aUFPaTVBRHdVU0FkCm45YWFweUNweEFkREtERHdObit3ZFhtaTZ0OHRpSF
RkK3RoVDhkaVpBb0dCQUt6Wis1bG9OOTBtYlF4VVh5YUwKMjFSUm9tMGJjcndsTmVCaWNFSmlzaEhYa2xpSVVxZ3hSZklNM2hh
UVRUcklKZENFaHFsV01aV0xPb2I2NTNyZgo3aFlMSXM1ZUtka3o0aFRVdnpldm9TMHVXcm9CV2xOVHlGanIrSWhKZnZUc0hpOG
dsU3FkbXgySkJhZUFVWUNXCndNdlQ4NmNLclNyNkQrZG8wS05FZzFsL0FvR0FlMkFVdHVFbFNqLzBmRzgrV3hHc1RFV1JqclRN
UzRSUjhRWXQKeXdjdFA4aDZxTGxKUTRCWGxQU05rMXZLTmtOUkxIb2pZT2pCQTViYjhibXNVU1BlV09NNENoaFJ4QnlHbmR2eA
phYkJDRkFwY0IvbEg4d1R0alVZYlN5T294ZGt5OEp0ek90ajJhS0FiZHd6NlArWDZDODhjZmxYVFo5MWpYL3RMCjF3TmRKS2tD
Z11Cbyt0UzB5TzJ2SWFmK2UwSkN5TGhzVDQ5cTN3Zis2QWVqWGx2WDJ1VnRYejN5QTZnbXo5aCsKcDNlK2JMRUxwb3B0WFhNdU
FRR0xhUkcrYlNNcjR5dERYbE5ZSndUeThXczNKY3dlSTdqZVp2b0ZpbmNvVlVIMwphdmxoTUVCRGYxSjltSDB5cDBwWUNaS2RO
dHNvZEZtQktzVEtQMjJhTmtsVVhCS3gyZzR6cFE9PQotLS0tLUVORCBSU0EgUFJJVkFURSBLRVktLS0tLQo=
```

```
user@ubuntu:~/ingress$ kubectl apply -f ic-tls.yaml

secret/default-server-secret created
```

```
user@ubuntu:~/ingress$
```

You can confirm that the secret was created under the nginx-ingress namespace by using `kubectl get secret -n nginx-ingress` command:

```
user@ubuntu:~/ingress$ kubectl get secret -n nginx-ingress

NAME                          TYPE                                  DATA   AGE
default-server-secret         Opaque                                2      4s
default-token-h2zpm           kubernetes.io/service-account-token   3      98s
nginx-ingress-token-jgdg9     kubernetes.io/service-account-token   3      77s

user@ubuntu:~/ingress$
```

Step 3 - Configuring RBAC

The IC will need permissions on several types of resources:

- services - targets for IC traffic
- endpoints - service target IPs
- secrets - for TLS
- configmaps - for IC configuration
- pods - workload targets
- events - to report activity
- ingresses - monitored for new ingress creation
- ingresses/status - where IC writes ingress status

The IC can be deployed for a specific namespace or globally for use by the entire cluster. We deploy our IC globally. This means we will need a ClusterRole to provide the appropriate permissions. Create the ClusterRole:

```
user@ubuntu:~/ingress$ nano ic-cr.yaml

user@ubuntu:~/ingress$ cat ic-cr.yaml
```

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: nginx-ingress
rules:
- apiGroups:
  - ""
  resources:
  - services
  - endpoints
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - secrets
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - get
  - list
```

```
  - watch
  - update
  - create
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - list
- apiGroups:
  - ""
  resources:
  - events
  verbs:
  - create
  - patch
- apiGroups:
  - extensions
  resources:
  - ingresses
  verbs:
  - list
  - watch
  - get
- apiGroups:
  - "extensions"
  resources:
  - ingresses/status
  verbs:
  - update
```

```
user@ubuntu:~/ingress$ kubectl apply -f ic-cr.yaml

clusterrole.rbac.authorization.k8s.io/nginx-ingress created

user@ubuntu:~/ingress$
```

Now bind the ClusterRole to the nginx-ingress SA:

```
user@ubuntu:~/ingress$ nano ic-crb.yaml

user@ubuntu:~/ingress$ cat ic-crb.yaml
```

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: nginx-ingress
subjects:
- kind: ServiceAccount
  name: nginx-ingress
  namespace: nginx-ingress
roleRef:
  kind: ClusterRole
  name: nginx-ingress
  apiGroup: rbac.authorization.k8s.io
```

```
user@ubuntu:~/ingress$ kubectl apply -f ic-crb.yaml

clusterrolebinding.rbac.authorization.k8s.io/nginx-ingress created

user@ubuntu:~/ingress$
```

Step 4 - Run the Ingress Controller

Ingress Controllers can be run as deployments, daemonsets, as kubelet manifest pods among other options. We'll run our controller as a deployment.

We'll deploy a single nginx IC pod using the nginx/nginx-ingress:edge image. Create the following deployment to launch the IC in the nginx-ingress namespace listening on port 80:

```
user@ubuntu:~/ingress$ nano ic-dep.yaml

user@ubuntu:~/ingress$ cat ic-dep.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-ingress
  namespace: nginx-ingress
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-ingress
  template:
    metadata:
      labels:
        app: nginx-ingress
    spec:
      serviceAccountName: nginx-ingress
      containers:
      - image: nginx/nginx-ingress:edge
        imagePullPolicy: Always
        name: nginx-ingress
        ports:
        - name: http
          containerPort: 80
        - name: https
          containerPort: 443
        env:
        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
        - name: POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        args:
          - -default-server-tls-secret=$(POD_NAMESPACE)/default-server-secret
```

```
user@ubuntu:~/ingress$ kubectl apply -f ic-dep.yaml

deployment.apps/nginx-ingress created

user@ubuntu:~/ingress$
```

Check your IC:

```
user@ubuntu:~/ingress$ kubectl get deploy,rs,pod -n nginx-ingress

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-ingress   1/1     1            1           8s

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-ingress-6d79b65f6c   1         1         1       8s

NAME                                READY   STATUS    RESTARTS   AGE
pod/nginx-ingress-6d79b65f6c-4pmkq   1/1     Running   0          8s
```

```
user@ubuntu:~/ingress$
```

Check the logs of the IC pod:

```
user@ubuntu:~/ingress$ kubectl logs nginx-ingress-6d79b65f6c-4pmkq -n nginx-ingress

I0109 17:50:03.429296       1 main.go:169] Starting NGINX Ingress controller Version=edge
GitCommit=94c326b9
2020/01/09 17:50:03 [notice] 12#12: using the "epoll" event method
2020/01/09 17:50:03 [notice] 12#12: nginx/1.17.7
2020/01/09 17:50:03 [notice] 12#12: built by gcc 8.3.0 (Debian 8.3.0-6)
2020/01/09 17:50:03 [notice] 12#12: OS: Linux 4.4.0-171-generic
2020/01/09 17:50:03 [notice] 12#12: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2020/01/09 17:50:03 [notice] 12#12: start worker processes
2020/01/09 17:50:03 [notice] 12#12: start worker process 13
2020/01/09 17:50:03 [notice] 12#12: start worker process 14
E0109 17:50:03.510635       1 reflector.go:123] /home/ec2-user/workspace/PI_IC_kubernetes-
ingress_master/internal/k8s/controller.go:340: Failed to list *v1.VirtualServer:
virtualservers.k8s.nginx.org is forbidden: User "system:serviceaccount:nginx-ingress:nginx-
ingress" cannot list resource "virtualservers" in API group "k8s.nginx.org" at the cluster scope
E0109 17:50:03.511305       1 reflector.go:123] /home/ec2-user/workspace/PI_IC_kubernetes-
ingress_master/internal/k8s/controller.go:341: Failed to list *v1.VirtualServerRoute:
virtualserverroutes.k8s.nginx.org is forbidden: User "system:serviceaccount:nginx-ingress:nginx-
ingress" cannot list resource "virtualserverroutes" in API group "k8s.nginx.org" at the cluster
scope
E0109 17:50:03.527343       1 reflector.go:280] /home/ec2-user/workspace/PI_IC_kubernetes-
ingress_master/internal/k8s/controller.go:332: Failed to watch *v1.Pod: unknown (get pods)
2020/01/09 17:50:03 [notice] 20#20: signal process started
2020/01/09 17:50:03 [notice] 12#12: signal 1 (SIGHUP) received from 20, reconfiguring
2020/01/09 17:50:03 [notice] 12#12: reconfiguring
2020/01/09 17:50:03 [notice] 12#12: using the "epoll" event method
2020/01/09 17:50:03 [notice] 12#12: start worker processes
2020/01/09 17:50:03 [notice] 12#12: start worker process 21
2020/01/09 17:50:03 [notice] 12#12: start worker process 22
2020/01/09 17:50:03 [notice] 14#14: gracefully shutting down
2020/01/09 17:50:03 [notice] 14#14: exiting
2020/01/09 17:50:03 [notice] 13#13: gracefully shutting down
2020/01/09 17:50:03 [notice] 13#13: exiting
2020/01/09 17:50:03 [notice] 13#13: exit
2020/01/09 17:50:03 [notice] 14#14: exit
I0109 17:50:03.635146       1 event.go:255] Event(v1.ObjectReference{Kind:"Secret",
Namespace:"nginx-ingress", Name:"default-server-secret", UID:"b8c02245-0819-4ded-b05e-
c8ff01be001b", APIVersion:"v1", ResourceVersion:"21587", FieldPath:""}): type: 'Normal' reason:
'Updated' the special Secret nginx-ingress/default-server-secret was updated
2020/01/09 17:50:03 [notice] 12#12: signal 17 (SIGCHLD) received from 13
2020/01/09 17:50:03 [notice] 12#12: worker process 13 exited with code 0
2020/01/09 17:50:03 [notice] 12#12: worker process 14 exited with code 0
2020/01/09 17:50:03 [notice] 12#12: signal 29 (SIGIO) received

...

user@ubuntu:~/ingress$
```

Step 5 - Create an Ingress Service

There are multiple ways to access an Ingress Controller from outside of a cluster:

- HostPorts - You can configure each pod to accept traffic on a host port
- NodePorts - You can create a node port service to forward traffic to the IC from the service port on every node
- LoadBalancer - Cloud deployments can use LB services for forward traffic to the IC

For this walk through, create a node port service for the IC:

```
user@ubuntu:~/ingress$ nano ic-svc.yaml

user@ubuntu:~/ingress$ cat ic-svc.yaml
```

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-ingress
  namespace: nginx-ingress
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
    name: http
  - port: 443
    targetPort: 443
    protocol: TCP
    name: https
  selector:
    app: nginx-ingress
```

```
user@ubuntu:~/ingress$ kubectl apply -f ic-svc.yaml

service/nginx-ingress created

user@ubuntu:~/ingress$
```

Display the resources:

```
user@ubuntu:~/ingress$ kubectl get deploy,rs,po,service -n nginx-ingress

NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-ingress   1/1     1            1           66s

NAME                                             DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-ingress-6d79b65f6c   1         1         1       66s

NAME                                 READY   STATUS    RESTARTS   AGE
pod/nginx-ingress-6d79b65f6c-4pmkq   1/1     Running   0          66s

NAME                    TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
service/nginx-ingress   NodePort   10.96.127.42   <none>        80:30820/TCP,443:30211/TCP   5s

user@ubuntu:~/ingress$
```

In the example above our nodes will forward traffic on 30820 to port 80 and 30211 to port 443.

Try curling the insecure Ingress controller node port:

```
user@ubuntu:~/ingress$ curl http://127.0.0.1:30820

<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx/1.17.7</center>
</body>
</html>

user@ubuntu:~/ingress$
```

The IC responds with a 404 not found error because we have no Ingress resources defined. Try the secure port:

```
user@ubuntu:~/ingress$ curl https://127.0.0.1:30211 -k
```

```
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx/1.17.7</center>
</body>
</html>

user@ubuntu:~/ingress$
```

Note that we have to use the https scheme, the correct node port (the one that forwards to 443) and the -k switch to avoid trying to verify the certificate the server provides (the one from the secret we created, which is self signed).

Our Ingress Controller is ready to use!

Step 6 - Create an Application

Imagine we would like to run an Apache web server and make it available through he Ingress Controller. We can create a deployment to run some Apache httpd pods, add a service for the pods and then create an Ingress resource that forwards traffic to the httpd service.

We'll use kubectl create and expose to create the httpd application quickly:

```
user@ubuntu:~/ingress$ kubectl create deployment web-svc --image=httpd

deployment.apps/web-svc created

user@ubuntu:~/ingress$ kubectl expose deploy/web-svc --port=80

service/web-svc exposed

user@ubuntu:~/ingress$ kubectl get deploy,rs,po,service

NAME                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/web-svc   1/1     1            1           10s

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/web-svc-546877f7db   1         1         1       10s

NAME                            READY   STATUS    RESTARTS   AGE
pod/web-svc-546877f7db-jjc8q   1/1     Running   0          10s

NAME                 TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1     <none>        443/TCP   22h
service/web-svc      ClusterIP   10.96.91.149   <none>        80/TCP    5s

user@ubuntu:~/ingress$
```

Now create an Ingress Resource for the httpd service. We'll ask the IC to forward traffic with the www.example.com Host header route to the httpd service. Create the ingress:

```
user@ubuntu:~/ingress$ nano ing.yaml

user@ubuntu:~/ingress$ cat ing.yaml
```

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: web-ingress
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      - path:
```

```
        backend:
          serviceName: web-svc
          servicePort: 80
```

```
user@ubuntu:~/ingress$ kubectl apply -f ing.yaml

ingress.extensions/web-ingress created

user@ubuntu:~/ingress$ kubectl get ing

NAME          HOSTS             ADDRESS   PORTS   AGE
web-ingress   www.example.com             80      21s

user@ubuntu:~/ingress$ kubectl describe ing

Name:             web-ingress
Namespace:        default
Address:
Default backend:  default-http-backend:80 (<none>)
Rules:
  Host             Path  Backends
  ----             ----  --------
  www.example.com
                      web-svc:80 (10.32.0.6:80)
Annotations:
  kubectl.kubernetes.io/last-applied-configuration:
{"apiVersion":"extensions/v1beta1","kind":"Ingress","metadata":{"annotations":{},"name":"web-
ingress","namespace":"default"},"spec":{"rules":[{"host":"www.example.com","http":{"paths":
[{"backend":{"serviceName":"web-svc","servicePort":80},"path":null}]}}]}}

Events:
  Type    Reason          Age   From                      Message
  ----    ------          ----  ----                      -------
  Normal  AddedOrUpdated  17s   nginx-ingress-controller  Configuration for default/web-ingress
was added or updated

user@ubuntu:~/ingress$
```

To test the ingress we will use the curl `--resolve` switch to force www.example.com to resolve to our ingress node port. In the real world you would setup your public DNS service to resolve www.example.com to the ingress node port. Try it:

```
user@ubuntu:~/ingress$ ip a show | head

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default
qlen 1000
    link/ether 00:0c:29:ca:be:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.228.157/24 brd 192.168.229.255 scope global ens33
       valid_lft forever preferred_lft forever

user@ubuntu:~/ingress$ kubectl get service -n nginx-ingress

NAME            TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
nginx-ingress   NodePort   10.96.127.42    <none>        80:30820/TCP,443:30211/TCP   3m1s

user@ubuntu:~/ingress$ curl --resolve www.example.com:30820:192.168.228.157
http://www.example.com:30820/

<html><body><h1>It works!</h1></body></html>
user@ubuntu:~$
```

We can create multiple ingresses and give a single ingress many rules. The example above routes based on the host name header. The

nginx ingress controller can also route by path, sending /web traffic to one services and /login traffic to another service for example.

For more information on ingress resources refer to the Kubernetes Ingress reference:
https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.15/#ingress-v1beta1-extensions

cleanup

When you are finished exploring your elements for the lab, begin deleting all resources created by commands:

```
user@ubuntu:~/ingress$ kubectl delete svc/web-svc deploy/web-svc

service "web-svc" deleted
deployment.extensions "web-svc" deleted

user@ubuntu:~/ingress$
```

Use `kubectl delete -f` in the ingress directory to delete all resources created by files:

```
user@ubuntu:~/ingress$ kubectl delete -f ~/ingress/.

clusterrole.rbac.authorization.k8s.io "nginx-ingress" deleted
clusterrolebinding.rbac.authorization.k8s.io "nginx-ingress" deleted
deployment.apps "nginx-ingress" deleted
service "nginx-ingress" deleted
secret "default-server-secret" deleted
ingress.extensions "web-ingress" deleted

user@ubuntu:~/ingress$
```

Then delete the `nginx-ingress` namespace:

```
user@ubuntu:~/ingress$ kubectl delete ns nginx-ingress

namespace "nginx-ingress" deleted

user@ubuntu:~/ingress$
```

And check to make sure you have removed all custom namespaces and resources:

```
user@ubuntu:~/ingress$ kubectl get all,ns

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   5h7m

NAME                          STATUS   AGE
namespace/default             Active   5h7m
namespace/kube-node-lease     Active   5h7m
namespace/kube-public         Active   5h7m
namespace/kube-system         Active   5h7m

user@ubuntu:~/ingress$ cd ~

user@ubuntu:~$
```

Congratulations, you have completed the lab!