RX-M Cloud Native Consulting

Advanced Kubernetes

Lab 8 – 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 Ingres 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:

```
ubuntu@ip-10-0-2-200:~$ kubectl get ns

NAME          STATUS    AGE
default       Active    3d
kube-public   Active    3d
kube-system   Active    3d

ubuntu@ip-10-0-2-200:~$ kubectl create namespace nginx-ingress

namespace/nginx-ingress created

ubuntu@ip-10-0-2-200:~$ kubectl get ns

NAME            STATUS    AGE
default         Active    3d
kube-public     Active    3d
kube-system     Active    3d
nginx-ingress   Active    4s
ubuntu@ip-10-0-2-200:~$
```

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:

```
ubuntu@ip-10-0-2-200:~$ kubectl create serviceaccount nginx-ingress --
namespace=nginx-ingress

serviceaccount/nginx-ingress created

ubuntu@ip-10-0-2-200:~$ kubectl get sa --namespace=nginx-ingress

NAME            SECRETS   AGE
default         1         5m
nginx-ingress   1         22s
ubuntu@ip-10-0-2-200:~$
```

Display the service account details:

```
ubuntu@ip-10-0-2-200:~$ kubectl describe sa nginx-ingress --namespace=nginx-
ingress

Name:                nginx-ingress
Namespace:           nginx-ingress
Labels:              <none>
Annotations:         <none>
Image pull secrets:  <none>
Mountable secrets:   nginx-ingress-token-d56nz
```

```
Tokens:              nginx-ingress-token-d56nz
Events:              <none>
ubuntu@ip-10-0-2-200:~$
```

Examine the token generated for the service account:

```
ubuntu@ip-10-0-2-200:~$ kubectl describe secret nginx-ingress-token-d56nz --
namespace=nginx-ingress

Name:         nginx-ingress-token-d56nz
Namespace:    nginx-ingress
Labels:       <none>
Annotations:  kubernetes.io/service-account.name=nginx-ingress
              kubernetes.io/service-account.uid=e92ab144-b7dc-11e8-971a-
028591ec4e6a

Type:  kubernetes.io/service-account-token

Data
====
ca.crt:     1025 bytes
namespace:  13 bytes
token:
eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwi
a3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJuZ2lueC1pbmdyZXNzIiwia3ViZ
XJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6Im5naW54LWluZ3Jlc3MtdG9rZW4tZD
U2bnoiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC5uYW1lIjoibmd
pbngtaW5ncmVzcyIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50LnVp
ZCI6ImU5MmFiMTQ0LWI3ZGMtMTFlOC05NzFhLTAyODU5MWVjNGU2YSIsInN1YiI6InN5c3RlbTpzZXJ2a
WNlYWNjb3VudDpuZ2lueC1pbmdyZXNzOm5naW54LWluZ3Jlc3MifQ.PpRtOdO0l3sYGjB5p8HdEBtC3BW
LtMaqkEYO3hYLYkPqpfYp9m6QMXIpVowPpC_8Qv5DHdP8Zf_uC54vhRwAJpBJzA2GpOBdZyxk8_5O0MFg
AxOSheSBmiQSvRf__LCOEOglYeSej5EO_kw72hEukbzfdPPKrgxwPtc-
Sj5UvgVvb_2dFVVqmJDd9KXrxhzn_liWUbqAFCTG_wNs24yzkhsx0knQU0jFG2FARsqoPznR6fYqTVV7T
O5PiRir9ZdkCIfK-
Lwb2YZFXFIeJ9WGSwWhRajMmKte4grmIuJ8XIdnJ1Ann7koW1NvP9B55nSJFWyDnncDbgSl4sXV00pUqw
ubuntu@ip-10-0-2-200:~$
```

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

```
ubuntu@ip-10-0-2-200:~$ vim ic-tls.yaml

ubuntu@ip-10-0-2-200:~$ cat ic-tls.yaml
```

```
apiVersion: v1
kind: Secret
metadata:
  name: default-server-secret
  namespace: nginx-ingress
type: Opaque
data:
  tls.crt:
```

```
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUN2akNDQWFZQ0NRREFPRjl0THNhWFhEQU5CZ2txaGtpR
zl3MEJBUXNGQURBaE1SOHdIUVlEVlFRRERCWk8KUjBsT1dFbHVaM0psYzNORGIyNTBjbTlzYkdWeU1CNFhEVE
U0TURreE1qRTRNRE16TlZvWERUSXpNRGt4TVRFNApNRE16TlZvd0lURWZNQjBHQTFVRUF3d1dUa2RKVGxoSmJ
tZHlaWE56UTI5dWRISnZiR3hsY2pDQ0FTSXdEUVlKCktvWklodmNOQVFFQkJRQURnZ0VQQURDQ0FRb0NnZ0VC
QUwvN2hIUEtFWGRMdjNyaUM3QlBrMTNpWkt5eTlyQ08KR2xZUXYyK2EzUDF0azIrS3YwVGF5aGRCbDRrcnNUc
TZzZm8vWUk1Y2Vhbkw4WGM3U1pyQkVRYm9EN2REbWs1Qgo4eDZLS2xHWU5IWlg0Rm5UZ0VPaStlM2ptTFFxRl
BSY1kzVnNPazFFeUZBL0JnWlJVbkNHZUtGeERSN0tQdGhyCmtqSXVuektURXUyaDU4Tlp0S21ScUJHdDEwcTN
RYzhZT3ExM2FnbmovUWRjc0ZYYTJnMjB1K1lYZDdoZ3krZksKWk4vVUkxQUQ0YzZyM1lma1ZWUmVHd1lxQVp1
WXN2V0RKbW1GNWRwdEMzN011cDBPRUxVTExSakZJOTZXNXIwSAo1TmdPc25NWFJNV1hYVlpiNWRxT3R0SmRtS
3FhZ25TZ1JQQVpQN2MwQjFQU2FqYzZjNGZRVXpNQ0F3RUFBVEFOCkJna3Foa2lHOXcwQkFRc0ZBQU9DQVFFQW
pLb2tRdGRPcEsrTzhibWVPc3lySmdJSXJycVFVY2ZOUitjb0hZVUoKdGhrYnhITFMzR3VBTWI5dm15VExPY2x
xeC9aYzJPblEwMEJCLzlTb0swcitFZ1U2UlVrRWtWcitTTFA3NTdUWgozZWI4dmdPdEduMS9ienM3bzNBaS9k
clkrcUI5Q2k1S3lPc3FHTG1US2xFaUtOYkcyR1ZyTWxjS0ZYQU80YTY3Cklnc1hzYktNbTQwV1U3cG9mcGltU
1ZmaXFSdkV5YmN3N0NYODF6cFErUyt1eHRYK2VBZ3V0NHh3VlI5d2IyVXYKelhuZk9HbWhWNThDd1dIQnNKa0
kxNXhaa2VUWXdSN0diaEFMSkZUUkk3dkhvQXprTWIzbjAxQjQyWjNrN3RXNQpJUDFmTlpIOFUvOWxiUHNoT21
FRFZkdjF5ZytVRVJxbStGSis2R0oxeFJGcGZnPT0KLS0tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo=
  tls.key:
LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcEFJQkFBS0NBUUVBdi91RWM4b1JkMHUvZXVJT
HNFK1RYZUprckxMMnNJNGFWaEMvYjVyYy9XMlRiNHEvClJOcktGMEdYaVN1eE9ycXgrajlnamx4NXFjdnhken
RKbXNFUkJ1Z1B0ME9hVGtIekhvb3FVWmcwZGxmZ1dkT0EKUTZMNTdlT1l0Q29VOUZ4amRXdzZUVVRJVUQ4R0J
sRlNjSVo0b1hFTkhzbysyR3VTTWk2Zk1wTVM3YUhudzFtMApxWkdvRWEzWFNyZEJ6eGc2clhkcUNlUDlCMXl3
VmRyYURiUzc1aGQzdUdETDU4cGszOVFqVUFQaHpxdmRoK1JWClZGNGJCaW9CbTVpeTlZTW1hWVhsMm0wTGZze
TZuUTRRdFFzdEdNVWozcGJtdlFmazJBNnljeGRFeFpkZFZsdmwKMm82MjBsMllxcHFDZEtCRThCay90elFIVT
lKcU56cHpoOUJUTXdJREFRQUJBb0lCQVFDZklHbXowOHhRVmorNwpLZnZJUXQwQ0YzR2MxNld6eDhVNml4MHg
4Mm15d1kxUUNlL3BzWE9LZlRxT1h1SENyUlp5TnUvZ2IvUUQ4bUFOCmxOMjRZTWl0TWRJODg5TEZoTkp3QU5O
ODJDeTczckM5bzVvUDlkazAvYzRIbjAzSkVYNzZ5QjgzQm9rR1FvYksKMjhMNk0rdHUzUmFqNjd6Vmc2d2szaa
EhrU0pXSzBwV1YrSjdrUkRWYmhDYUZhNk5nMUZNRWxhTlozVDhhUUtyQgpDUDNDeEFTdjYxWTk5TEI4KzNXWV
FIK3NYaTVGM01pYVNBZ1BkQUk3WEh1dXFET1lvMU5PL0JoSGt1aVg2QnRtCnorNTZud2pZMy8yUytSRmNBc3J
MTnIwMDJZZi9oY0IraVlDNzVWYmcydVd6WTY3TWdOTGQ5VW9RU3BDRkYrVm4KM0cyUnhybnhBb0dCQU40U3M0
ZVlPU2huMVpQQjdhTUZsY0k2RHR2S2ErTGZTTXFyY2pOZjJlSEpZNnhubmxKdgpGenpGL2RiVWVTbWxSekR0W
kdlcXZXaHFISy9iTjIyeWJhOU1WMDlRQ0JFTk5jNmtWajJTVHpUWkJVbEx4QzYrCk93Z0wyZHhKendWelU0VC
84ajdHalRUN05BZVpFS2FvRHFyRG5BYWkyaW5oZU1JVWZHRXFGKzJyQW9HQkFOMVAKK0tZL0lsS3RWRzRKSkl
QNzBjUis3RmpyeXJpY05iWCtQVzUvOXFHaWxnY2grZ3l4b25BWlBpd2NpeDN3QVpGdwpaZC96ZFB2aTBkWEpp
c1BSZjRMazg5b2pCUmpiRmRmc2l5UmJYbyt3TFU4NUhRU2NGMnN5aUFPaTVBRHdVU0FkCm45YWFweUNweEFkR
EtERHdObit3ZFhtaTZ0OHRpSFRkK3RoVDhkaVpBb0dCQUt6Wis1bG9OOTBtYlF4VVh5YUwKMjFSUm9tMGJjcn
dsTmVCaWNFSmlzaEhYa2xpSVVxZ3hSZklNM2hhUVRUcklKZENFaHFsV01aV0xPb2I2NTNyZgo3aFlMSXM1ZUt
ka3o0aFRVdnpldm9TMHVXcm9CV2xOVHlGanIrSWhKZnZUc0hpOGdsU3FkbXgySkJhZUFVWUNXCndNdlQ4NmNL
clNyNkQrZG8wS05FZzFsL0FvR0FlMkFVdHVFbFNqLzBmRzgrV3hHc1RFV1JqclRNUzRSUjhRWXQKeXdjdFA4a
DZxTGxKUTRCWGxQU05rMXZLTmtOUkxIb2pZT2pCQTViYjhibXNVU1BlV09NNENoaFJ4QnlHbmR2eAphYkJDRk
FwY0IvbEg4d1R0alVZYlN5T294ZGt5OEp0ek90ajJhS0FiZHd6NlArWDZDODhjZmxYVFo5MWpYL3RMCjF3TmR
KS2tDZ1lCbyt0UzB5TzJ2SWFmK2UwSkN5TGhzVDQ5cTN3Zis2QWVqWGx2WDJ1VnRYejN5QTZnbXo5aCsKcDNl
K2JMRUxwb3B0WFhNdUFRR0xhUkcrYlNNcjR5dERYbE5ZSndUeThXczNKY3dlSTdqZVp2b0ZpbmNvVlVIMwphd
mxoTUVCRGYxSjltSDB5cDBwWUNaS2ROdHNvZEZtQktzVEtQMjJhTmtsVVhCS3gyZzR6cFE9PQotLS0tLUVORC
BSU0EgUFJJVkFURSBLRVktLS0tLQo=
```

```
ubuntu@ip-10-0-2-200:~$ kubectl apply -f ic-tls.yaml

secret/default-server-secret created
ubuntu@ip-10-0-2-200:~$
```

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

```
ubuntu@ip-10-0-2-200:~$ kubectl get secret --namespace=nginx-ingress

NAME                          TYPE                                  DATA   AGE
default-server-secret         Opaque                                2      10s
...

ubuntu@ip-10-0-2-200:~$
```

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:

```
ubuntu@ip-10-0-2-200:~$ vim ic-cr.yaml

ubuntu@ip-10-0-2-200:~$ 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
```

```
ubuntu@ip-10-0-2-200:~$ kubectl apply -f ic-cr.yaml

clusterrole.rbac.authorization.k8s.io/nginx-ingress created
ubuntu@ip-10-0-2-200:~$
```

Now bind the ClusterRole to the nginx-ingress SA:

```
ubuntu@ip-10-0-2-200:~$ vim ic-crb.yaml

ubuntu@ip-10-0-2-200:~$ 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
```

```
ubuntu@ip-10-0-2-200:~$ kubectl apply -f ic-crb.yaml

clusterrolebinding.rbac.authorization.k8s.io/nginx-ingress created
ubuntu@ip-10-0-2-200:~$
```

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:

```
ubuntu@ip-10-0-2-200:~$ vim ic-dep.yaml

ubuntu@ip-10-0-2-200:~$ 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
```

```
ubuntu@ip-10-0-2-200:~$ kubectl apply -f ic-dep.yaml

deployment.apps/nginx-ingress created
ubuntu@ip-10-0-2-200:~$
```

Check your IC:

```
ubuntu@ip-10-0-2-200:~$ kubectl get deploy,rs,pod --namespace=nginx-ingress

NAME                                  DESIRED   CURRENT   UP-TO-DATE   AVAILABLE
AGE
deployment.extensions/nginx-ingress   1         1         1            1
1m

NAME                                             DESIRED   CURRENT   READY
AGE
replicaset.extensions/nginx-ingress-6b49c44f44   1         1         1         1m

NAME                                 READY     STATUS    RESTARTS   AGE
pod/nginx-ingress-6b49c44f44-tgk5k   1/1       Running   0          1m
ubuntu@ip-10-0-2-200:~$
```

Check the logs of the IC pod:

```
ubuntu@ip-10-0-2-200:~$ kubectl logs nginx-ingress-6b49c44f44-tgk5k --
namespace=nginx-ingress

I0327 03:45:35.291103       1 main.go:136] Starting NGINX Ingress controller
Version=edge GitCommit=4d0ffc2e
2019/03/27 03:45:35 [notice] 12#12: using the "epoll" event method
2019/03/27 03:45:35 [notice] 12#12: nginx/1.15.9
2019/03/27 03:45:35 [notice] 12#12: built by gcc 6.3.0 20170516 (Debian 6.3.0-
18+deb9u1)
```

```
2019/03/27 03:45:35 [notice] 12#12: OS: Linux 4.4.0-31-generic
2019/03/27 03:45:35 [notice] 12#12: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2019/03/27 03:45:35 [notice] 12#12: start worker processes
2019/03/27 03:45:35 [notice] 12#12: start worker process 13
2019/03/27 03:45:35 [notice] 12#12: start worker process 14
2019/03/27 03:45:35 [notice] 19#19: signal process started
2019/03/27 03:45:35 [notice] 12#12: signal 1 (SIGHUP) received from 19,
reconfiguring
2019/03/27 03:45:35 [notice] 12#12: reconfiguring
2019/03/27 03:45:35 [notice] 12#12: using the "epoll" event method
2019/03/27 03:45:35 [notice] 12#12: start worker processes
2019/03/27 03:45:35 [notice] 12#12: start worker process 20
2019/03/27 03:45:35 [notice] 12#12: start worker process 21
2019/03/27 03:45:35 [notice] 14#14: gracefully shutting down
2019/03/27 03:45:35 [notice] 14#14: exiting
2019/03/27 03:45:35 [notice] 13#13: gracefully shutting down
2019/03/27 03:45:35 [notice] 13#13: exiting
2019/03/27 03:45:35 [notice] 13#13: exit
2019/03/27 03:45:35 [notice] 14#14: exit
I0327 03:45:35.466392       1 event.go:218]
Event(v1.ObjectReference{Kind:"Secret", Namespace:"nginx-ingress", Name:"default-
server-secret", UID:"2e745a07-b7e3-11e8-971a-028591ec4e6a", APIVersion:"v1",
ResourceVersion:"356308", FieldPath:""}): type: 'Normal' reason: 'Updated' the
default server Secret nginx-ingress/default-server-secret was updated
2019/03/27 03:45:35 [notice] 12#12: signal 17 (SIGCHLD) received from 13
2019/03/27 03:45:35 [notice] 12#12: worker process 13 exited with code 0
2019/03/27 03:45:35 [notice] 12#12: worker process 14 exited with code 0
2019/03/27 03:45:35 [notice] 12#12: signal 29 (SIGIO) received
ubuntu@ip-10-0-2-200:~$
```

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:

```
ubuntu@ip-10-0-2-200:~$ vim ic-svc.yaml

ubuntu@ip-10-0-2-200:~$ 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
```

```
ubuntu@ip-10-0-2-200:~$ kubectl apply -f ic-svc.yaml

service/nginx-ingress created
ubuntu@ip-10-0-2-200:~$
```

Display the resources:

```
ubuntu@ip-10-0-2-200:~$ kubectl get deploy,rs,po,service --namespace=nginx-
ingress

NAME                                  DESIRED   CURRENT   UP-TO-DATE   AVAILABLE
AGE
deployment.extensions/nginx-ingress   1         1         1            1
17m

NAME                                             DESIRED   CURRENT   READY
AGE
replicaset.extensions/nginx-ingress-6b49c44f44   0         0         0
17m
replicaset.extensions/nginx-ingress-84df7788c8   1         1         1         7m

NAME                                 READY     STATUS    RESTARTS   AGE
pod/nginx-ingress-84df7788c8-cvpdb   1/1       Running   0          7m

NAME                    TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
service/nginx-ingress   NodePort   10.96.228.230   <none>
80:32559/TCP,443:30352/TCP   48s
ubuntu@ip-10-0-2-200:~$
```

In the example above our nodes will forward traffic on 32559 to port 80 and 30352 to port 443.

Try curling the insecure Ingress controller node port:

```
ubuntu@ip-10-0-2-200:~$ curl http://127.0.0.1:32559

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

```
<hr><center>nginx/1.15.9</center>
</body>
</html>
ubuntu@ip-10-0-2-200:~$
```

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

```
ubuntu@ip-10-0-2-200:~$ curl https://127.0.0.1:30352 -k

<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx/1.15.9</center>
</body>
</html>
ubuntu@ip-10-0-2-200:~$
```

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.

Well use kubectl run and expose to create the httpd application quickly:

```
ubuntu@ip-10-0-2-200:~$ kubectl create deployment web-svc --image=httpd

deployment.apps/web-svc created

ubuntu@ip-10-0-2-200:~$ kubectl expose deploy/web-svc --port=80

service/web-svc exposed

ubuntu@ip-10-0-2-200:~$ kubectl get deploy,rs,po,service

NAME                            DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/web-svc   1         1         1            1           47s

NAME                                       DESIRED   CURRENT   READY     AGE
replicaset.extensions/web-svc-7b88547ff5   1         1         1         47s

NAME                           READY     STATUS    RESTARTS   AGE
pod/web-svc-7b88547ff5-jwkzg   1/1       Running   0          47s

NAME                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP   3d
```

```
service/web-svc      ClusterIP   10.102.104.236   <none>        80/TCP    23s
ubuntu@ip-10-0-2-200:~$
```

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:

```
ubuntu@ip-10-0-2-200:~$ vim ing.yaml

ubuntu@ip-10-0-2-200:~$ cat ing.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
```

```
ubuntu@ip-10-0-2-200:~$ kubectl apply -f ing.yaml

ingress.extensions/web-ingress created

ubuntu@ip-10-0-2-200:~$ kubectl get ing

NAME          HOSTS             ADDRESS   PORTS     AGE
web-ingress   www.example.com             80        11s

ubuntu@ip-10-0-2-200:~$ kubectl describe ing

Name:             web-ingress
Namespace:        default
Address:
Default backend:  default-http-backend:80 (<none>)
Rules:
  Host             Path  Backends
  ----             ----  --------
  www.example.com
                   /web   web-svc:80 (<none>)
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":"/web"}]}}]}}
```

```
Events:
  Type    Reason          Age   From                      Message
  ----    ------          ----  ----                      -------
  Normal  AddedOrUpdated  19s   nginx-ingress-controller  Configuration for
default/web-ingress was added or updated
ubuntu@ip-10-0-2-200:~$
```

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:

```
ubuntu@ip-10-0-2-200:~$ ip a show

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:29:8c:ed:9a brd ff:ff:ff:ff:ff:ff
    inet 192.168.225.140/24 brd 192.168.225.255 scope global ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe8c:ed9a/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
group default
    link/ether 02:42:fc:29:0c:e2 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:fcff:fe29:ce2/64 scope link
       valid_lft forever preferred_lft forever

...

ubuntu@ip-10-0-2-200:~$ kubectl get service -n nginx-ingress

NAME            TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
nginx-ingress   NodePort   10.96.228.230   <none>
80:32559/TCP,443:30352/TCP   28m

ubuntu@ip-10-0-2-200:~$ curl --resolve www.example.com:32559:192.168.225.140
http://www.example.com:32559/

<html><body><h1>It works!</h1></body></html>
ubuntu@ip-10-0-2-200:~$
```

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/v1.13/#ingress-v1beta1-extensions

Congratulations, you have completed the lab!