Taint & Tolerations, Ingress-Controller, Persistent Volumes, StatefulSet Resource

Contents

1 INTRODUCTION

Taint and Tolerations

Node affinity, is a property of Pods that attracts them to a set of nodes (either as a preference or a hard requirement). Taints are the opposite -- they allow a node to repel a set of pods.

Tolerations are applied to pods, and allow (but do not require) the pods to schedule onto nodes with matching taints.

Taints and tolerations work together to ensure that pods are not scheduled onto inappropriate nodes. One or more taints are applied to a node; this marks that the node should not accept any pods that do not tolerate the taints.

Ingress-Controller

In order for the Ingress resource to work, the cluster must have an ingress controller running.

Unlike other types of controllers which run as part of the kube-controller-manager binary, Ingress controllers are not started automatically with a cluster. Use this page to choose the ingress controller implementation that best fits your cluster.

This guide Covers:

- Taint and Tolerations
- Advanced Routing with Ingress-Controller
- Dynamic Provisioning of Persistent Volumes
- Deploying and Managing a StatefulSet Resource

2 DOCUMENTATION

2.1 Kubernetes Documentation

1. Taint & Tolerations

   https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/
2. Ingress Controllers

   https://kubernetes.io/docs/concepts/services-networking/ingress-controllers
3. Dynamic Volume Provisioning

   https://kubernetes.io/docs/concepts/storage/dynamic-provisioning/#:~:text=Dynamic%20volume%20provisioning%20allows%20storage,to%20represent%20them%20in%20Kubernetes.
4. StatefulSets

   https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

2.2 Linux Commands and VIM Commands

1. Basic Linux Commands

   https://maker.pro/linux/tutorial/basic-linux-commands-for-beginners

   https://www.hostinger.in/tutorials/linux-commands
2. Basic VIM Commands

   https://coderwall.com/p/adv71w/basic-vim-commands-for-getting-started
3. Popular VIM Commands

   https://www.keycdn.com/blog/vim-commands

3 PREVIOUS GUIDES

Ensure that you have completed following activity guides:

- ***Note****: Follow Activity Guide* ***AG_Bootstrap_Kubernetes_Cluster_Using_Kubeadm_Guide_ed*** *from portal*
- ***Note****: Follow Activity Guide* ***AG_ Deploy_App_On_Pod_&_Basic_Networking_ed*** *from portal*
- ***Note****: Follow Activity Guide* ***AG_Deploying_Scalable_and_Configuring_Autoscaling_For_Stateless_Application_ed*** *from portal*
- ***Note****: Follow Activity Guide* ***AG_Configuring_NFS_Storage_Persistence_Volume_ed*** *from portal*
- ***Note****: Follow Activity Guide* ***AG_Constraint_Pod_and_Node_Selector_Node_Affinity_&_Anti_Affinity_ed*** *from portal*
- ***Note****: Follow Activity Guide* ***AG_Cluster_Node_Maintenance_Debugging_Application_Failure_Troubleshooting_Cluster_ed*** *from portal*
- ***Note****: Follow Activity Guide* ***AG_Cluster_Security_Working_With_ConfigMap_&_Limiting_Resources_With_Resource_Quota_ed*** *from portal*
- ***Note****: Follow Activity Guide* ***AG_Deploying_PHP_Guestbook_Collect_Logs_With_Elk_Stack_Backup_Restore_ETCD_Cluster_ed*** *from portal*

4 ADVANCED SCHEDULING WITH TAINT AND TOLERATIONS

4.1 Tainting a Node to Simulate Advanced Scheduling

1. View all the nodes in the cluster

```
$ kubectl get nodes
```

```
$ kubectl get nodes
NAME                                STATUS   ROLES   AGE    VERSION
aks-agentpool-40017546-vmss000000   Ready    agent   148m   v1.15.10
aks-agentpool-40017546-vmss000001   Ready    agent   148m   v1.15.10
$
```

2. Taint one of the nodes by using its name

```
$ kubectl taint node aks-agentpool-40017546-vmss000001
disktype=magnetic:NoSchedule
```

```
$
$ kubectl taint node aks-agentpool-40017546-vmss000001 disktype=magnetic:NoSchedule
node/aks-agentpool-40017546-vmss000001 tainted
```

3. Verify that the taint was applied to the desired node

```
$ kubectl describe node aks-agentpool-40017546-vmss000001 | grep -i "taints"
```

```
$ kubectl describe node aks-agentpool-40017546-vmss000001 | grep -i "taints"
Taints:             disktype=magnetic:NoSchedule
$
```

4.2 Creating Pod without Toleration

1. View the content of tt-pod.yaml file and create pod using the yaml file

```
$ vi tt-pod.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: tt-pod
spec:
  containers:
  - name: nginx
    image: nginx
~
~
~
~
~
~
```

```
$ kubectl create -f tt-pod.yaml
```

```
$
$ kubectl create -f tt-pod.yaml
pod/tt-pod created
$
```

2. Verify the pod status. Notice that it was scheduled on the node which is not tainted

```
$ kubectl get pods -o wide
```

```
$
$ kubectl get pods -o wide
NAME     READY   STATUS    RESTARTS   AGE   IP           NODE                                NOMINATED NODE   READINESS GATES
tt-pod   1/1     Running   0          83s   10.244.0.13  aks-agentpool-40017546-vmss000000   <none>           <none>
$
```

3. Delete the pod created in this task

```
$ kubectl delete -f tt-pod.yaml
```

```
$
$ kubectl delete -f tt-pod.yaml
pod "tt-pod" deleted
$
```

4.3 Creating a Pod with Toleration

1. View the content of tt-pod1.yaml file and create pod using the yaml file

```
$ vi tt-pod1.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: tt-pod1
spec:
  containers:
   - name: nginx
     image: nginx
  tolerations:
  - key: "disktype"
    operator: "Equal"
    value: "magnetic"
    effect: "NoSchedule"
~
~
~
~
```

```
$ kubectl create -f tt-pod1.yaml
```

```
$
$ kubectl create -f tt-pod1.yaml
pod/tt-pod1 created
$
```

2. Verify the pod status. Notice that it was scheduled on the tainted node

```
$ kubectl get pods -o wide
```

```
$ kubectl get pods -o wide
NAME      READY   STATUS    RESTARTS   AGE   IP           NODE                                NOMINATED NODE   READINESS GATES
tt-pod1   1/1     Running   0          8s    10.244.1.16  aks-agentpool-40017546-vmss000001   <none>           <none>
$
```

3. Delete the pod created in this task

```
$ kubectl delete -f tt-pod1.yaml
```

4. Delete the taint from the node

```
$ kubectl taint node <node_name> disktype-
$ kubectl describe nodes <node_name> | grep -i taint
```

4.4 Simulate eviction of Pod using NoSchedule effect

1. Again create a pod using tt-pod.yaml file. It doesn't have any toleration defined

```
$ kubectl create -f tt-pod.yaml
```

```
$
$ kubectl create -f tt-pod.yaml
pod/tt-pod created
$
```

2. Taint the node on which the Pod was scheduled

```
$ kubectl get pods -o wide
```

```
$
$ kubectl get pods -o wide
NAME     READY   STATUS    RESTARTS   AGE   IP            NODE                                NOMINATED NODE   READINESS GATES
tt-pod   1/1     Running   0          10s   10.244.0.14   aks-agentpool-40017546-vmss000000   <none>           <none>
$
```

```
$ kubectl taint node <node_name> disktype=magnetic:NoExecute
```

```
$ kubectl taint node aks-agentpool-40017546-vmss000000 disktype=magnetic:NoExecute
node/aks-agentpool-40017546-vmss000000 tainted
$
```

```
$
$ kubectl describe node aks-agentpool-40017546-vmss000000 | grep -i "taints"
Taints:             disktype=magnetic:NoExecute
$
```

3. Verify the pods status again and see that the pod is evicted

```
$ kubectl get pods -o wide
```

4. View recent events to see that the pod was evicted due to the taint

```
$ kubectl get events
```

```
$ kubectl get events | grep tt-pod
45m         Normal    Scheduled             pod/tt-pod    Successfully assigned default/tt-pod to aks-agentpool-40017546-vmss000001
45m         Normal    Pulling               pod/tt-pod    Pulling image "nginx"
45m         Normal    Pulled                pod/tt-pod    Successfully pulled image "nginx"
45m         Normal    Created               pod/tt-pod    Created container nginx
45m         Normal    Started               pod/tt-pod    Started container nginx
44m         Normal    Killing               pod/tt-pod    Stopping container nginx
41m         Normal    Scheduled             pod/tt-pod    Successfully assigned default/tt-pod to aks-agentpool-40017546-vmss000001
41m         Normal    Pulling               pod/tt-pod    Pulling image "nginx"
41m         Normal    Pulled                pod/tt-pod    Successfully pulled image "nginx"
41m         Normal    Created               pod/tt-pod    Created container nginx
41m         Normal    Started               pod/tt-pod    Started container nginx
36m         Normal    Killing               pod/tt-pod    Stopping container nginx
35m         Normal    Scheduled             pod/tt-pod    Successfully assigned default/tt-pod to aks-agentpool-40017546-vmss000001
35m         Normal    Pulling               pod/tt-pod    Pulling image "nginx"
34m         Normal    Pulled                pod/tt-pod    Successfully pulled image "nginx"
34m         Normal    Created               pod/tt-pod    Created container nginx
34m         Normal    Started               pod/tt-pod    Started container nginx
2m13s       Normal    TaintManagerEviction  pod/tt-pod    Marking for deletion Pod default/tt-pod
33m         Normal    Killing               pod/tt-pod    Stopping container nginx
```

5. Delete the pod and taint from the node

```
$ kubectl delete -f tt-pod.yaml

$ kubectl taint node <node_name> disktype-
$ kubectl describe nodes <node_name> | grep -i taint
```

5 DEPLOYING AND MANAGING A STATEFULSET RESOURCE

5.1 Creating Logging namespace

1. Viewing the contents of namespace.yaml file to create kube-logging namespace

```
$ vim namespace.yaml
```

```
kind: Namespace
apiVersion: v1
metadata:
  name: kube-logging
~
~
~
~
~
~
~
```

2. Creating namespace from above file

```
$ kubectl create -f namespace.yaml
```

```
$
$ kubectl create -f namespace.yaml
namespace/kube-logging created
$
```

3. Confirm that the Namespace was successfully created by listing all the namespace present in the cluster

```
$ kubectl get ns
```

```
$ kubectl get ns
NAME              STATUS   AGE
default           Active   16h
kube-logging      Active   13s
kube-node-lease   Active   16h
kube-public       Active   16h
kube-system       Active   16h
$
```

5.2 Setting up Elasticsearch application

1. Create the Elasticsearch StatefulSet using elasticsearch-stfullset.yaml file. Run through the content and create the resource

```
$ vim elasticsearch-stfullset.yaml
$ kubectl create -f elasticsearch-stfullset.yaml
```

```
$
$
$ kubectl create -f elasticsearch-svc.yaml
service/elasticsearch created
$
$
```

2. Verify the creation of StatefulSet Elasticsearch pods. monitor the StatefulSet as it is rolled out using kubectl rollout status

```
$ kubectl rollout status sts/es-cluster --namespace=kube-logging
$ kubectl get sts --namespace=kube-logging
$ kubectl get pods --namespace=kube-logging
```

```
$ kubectl rollout status sts/es-cluster --namespace=kube-logging
partitioned roll out complete: 3 new pods have been updated...
$
$ kubectl get sts --namespace=kube-logging
NAME         READY   AGE
es-cluster   3/3     25m
$
$ kubectl get pods --namespace=kube-logging
NAME                    READY   STATUS    RESTARTS   AGE
es-cluster-0            1/1     Running   0          25m
es-cluster-1            1/1     Running   0          6m27s
es-cluster-2            1/1     Running   0          4m51s
```

5.3 Pods in a StatefulSet

1. Pods in a StatefulSet have a unique ordinal index and a stable network identity.

   Each Pod has a stable hostname based on its ordinal index. Use kubectl exec to execute the hostname command in each Pod. Let's examine the pods

```
$ kubectl config set-context --current --namespace=kube-logging
$ kubectl get pods
```

```
for i in 0 1 2; do kubectl exec es-cluster-$i -- sh -c 'hostname'; done
```

```
$ kubectl config set-context --current --namespace=kube-logging
Context "k8s-demo" modified.
$ kubectl get pods
NAME                    READY   STATUS    RESTARTS   AGE
es-cluster-0            1/1     Running   0          3h14m
es-cluster-1            1/1     Running   0          174m
es-cluster-2            1/1     Running   0          172m
fluentd-2vw2j           1/1     Running   0          162m
fluentd-9f298           1/1     Running   0          162m
fluentd-m9hxb           1/1     Running   0          162m
kibana-cd68dcfb-pjnhc   1/1     Running   6          3h8m
$ for i in 0 1; do kubectl exec es-cluster-$i -- sh -c 'hostname' -n kube-logging; done
es-cluster-0
es-cluster-1
```

5.4 Scaling up and down a Statefulset object

1. Scaling up the replicas from 3 to 4 for sts es-cluster. The StatefulSet controller scales the number of replicas.

```
$ kubectl scale sts es-cluster --replicas=4
```

```
$ kubectl scale sts es-cluster --replicas=4
statefulset.apps/es-cluster scaled
```

2. The StatefulSet controller creates each Pod sequentially with respect to its ordinal index, and it waits for each Pod's predecessor to be Running and Ready before launching the subsequent Pod

```
$ kubectl rollout status sts/es-cluster
```

```
$ kubectl rollout status sts/es-cluster
Waiting for 1 pods to be ready...
partitioned roll out complete: 4 new pods have been updated...
$
```

```
$ kubectl get pods
```

```
$
$ kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
es-cluster-0   1/1     Running   0          9m40s
es-cluster-1   1/1     Running   0          8m54s
es-cluster-2   1/1     Running   0          8m8s
es-cluster-3   1/1     Running   0          66s
$
```

3. Scaling down the replicas from 4 to 2 for sts es-cluster. The StatefulSet controller scales the number of replicas.

```
$ kubectl scale sts es-cluster --replicas=2
```

```
$
$ kubectl scale sts es-cluster --replicas=2
statefulset.apps/es-cluster scaled
$
```

4. The controller deletes one Pod at a time, in reverse order with respect to its ordinal index, and it waits for each to completely shut down before deleting the next.

```
$ kubectl rollout status sts/es-cluster
```

```
$
$ kubectl rollout status sts/es-cluster
partitioned roll out complete: 2 new pods have been updated...
$
```

```
$ kubectl get pods
```

```
$
$ kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
es-cluster-0   1/1     Running   0          16m
es-cluster-1   1/1     Running   0          15m
$
```

5.5 Rolling update StatefulSets

1. The RollingUpdate update strategy will update all Pods in a StatefulSet, in reverse ordinal order, while respecting the StatefulSet guarantees.

2. Edit the StatefulSet to update the new image version of Elasticsearch elasticsearch:7.5.0

```
$ kubectl edit sts es-cluster
```

```
# reopened with the relevant failures.
#
apiVersion: apps/v1
kind: StatefulSet
metadata:
  creationTimestamp: "2020-06-03T13:28:20Z"
  generation: 3
  name: es-cluster
  namespace: kube-logging
  resourceVersion: "117909"
  selfLink: /apis/apps/v1/namespaces/kube-logging/statefulsets/es-cluster
  uid: 2e6a26e5-4af2-4b44-84af-1c4b3b5da978
spec:
  podManagementPolicy: OrderedReady
  replicas: 2
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: elasticsearch
  serviceName: elasticsearch
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: elasticsearch
    spec:
      containers:
      - env:
        - name: cluster.name
          value: k8s-logs
        - name: node.name
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: metadata.name
        - name: discovery.seed_hosts
          value: es-cluster-0.elasticsearch,es-cluster-1.elasticsearch,es-cluster-2.elasticsearch
        - name: cluster.initial_master_nodes
          value: es-cluster-0,es-cluster-1,es-cluster-2
        - name: ES_JAVA_OPTS
          value: -Xms512m -Xmx512m
        image: docker.elastic.co/elasticsearch/elasticsearch:7.5.0
/.4
```

3. Verify the updation of StatefulSet Elasticsearch pods. Monitor the StatefulSet as it is rolled out using kubectl rollout status

```
$ kubectl rollout status sts/es-cluster
```

```
$ kubectl rollout status sts/es-cluster
Waiting for 1 pods to be ready...
Waiting for 1 pods to be ready...
```

```
$ kubectl get pods -w
```

```
$ kubectl get pods -w
NAME           READY   STATUS            RESTARTS   AGE
es-cluster-0   1/1     Running           0          28m
es-cluster-1   0/1     PodInitializing   0          2m1s
es-cluster-1   1/1     Running           0          2m4s
es-cluster-0   1/1     Terminating       0          28m
es-cluster-0   0/1     Terminating       0          28m
es-cluster-0   0/1     Terminating       0          28m
es-cluster-0   0/1     Terminating       0          28m
es-cluster-0   0/1     Pending           0          0s
es-cluster-0   0/1     Pending           0          0s
es-cluster-0   0/1     Init:0/3          0          0s
es-cluster-0   0/1     Init:1/3          0          15s
es-cluster-0   0/1     Init:2/3          0          17s
es-cluster-0   0/1     PodInitializing   0          18s
es-cluster-0   1/1     Running           0          65s
```

4. Verify the image version with describe command

```
$ kubectl describe sts es-cluster | grep Image
```

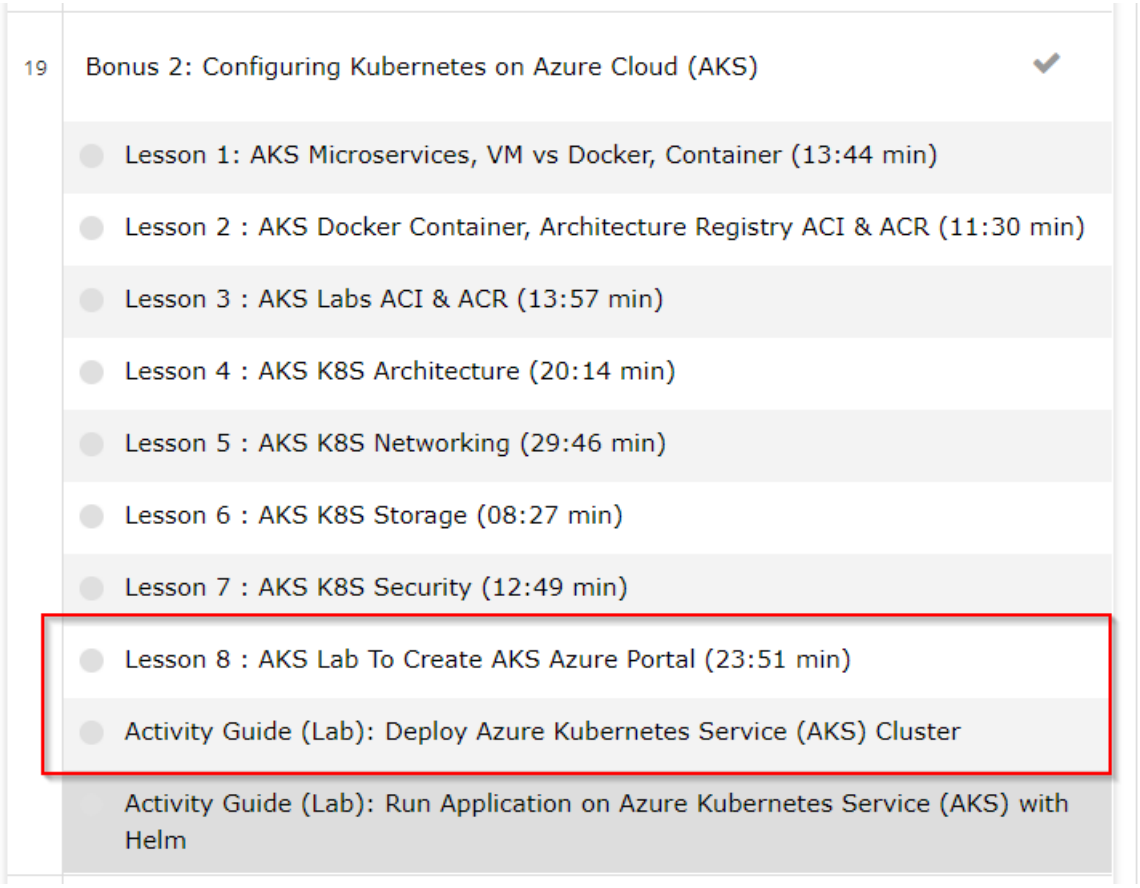
```
$
$ kubectl describe sts es-cluster | grep Image
    Image:      busybox
    Image:      busybox
    Image:      busybox
    Image:       docker.elastic.co/elasticsearch/elasticsearch:7.5.0
$
```

5.6 Clean Up resources created the lab exercise

```
$ kubectl delete ns kube-logging
$ kubectl config set-context --current --namespace=default
```

6 ADVANCED ROUTING WITH INGRESS-CONTROLLER

Note: Section 6 & 7 **Ingress-Controller** and **Dynamic Provisioning of Persistent Volumes** you need to perform in AKS Cluster, not in your regular kubeadm cluster so before performing these sections first please follow Guide Deploy **Azure Kubernetes Service(AKS)** cluster guide from the portal.



Contents

6.1 Deploying NGINX Ingress Controller using helm chart

1. Create a namespace for your ingress resources

```
$ kubectl create namespace ingress-basic
```

```
ubuntu@master:~$ sudo su
root@master:/home/ubuntu# kubectl create namespace ingress-basic
namespace/ingress-basic created
```

2. Add the official stable repository

```
$ helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
```

```
root@master:/home/ubuntu# helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
"ingress-nginx" has been added to your repositories
```

3. Use Helm to deploy an NGINX ingress controller

```
$ helm install nginx-ingress ingress-nginx/ingress-nginx --namespace ingress-basic --set controller.replicaCount=2
```

```
root@master:/home/ubuntu# helm install nginx-ingress ingress-nginx/ingress-nginx --namespace ingress-basic --set controller.replicaCou
nt=2
NAME: nginx-ingress
LAST DEPLOYED: Tue Dec 15 06:54:41 2020
NAMESPACE: ingress-basic
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The ingress-nginx controller has been installed.
It may take a few minutes for the LoadBalancer IP to be available.
You can watch the status by running 'kubectl --namespace ingress-basic get services -o wide -w nginx-ingress-ingress-nginx-controller'

An example Ingress that makes use of the controller:

  apiVersion: networking.k8s.io/v1beta1
  kind: Ingress
  metadata:
    annotations:
      kubernetes.io/ingress.class: nginx
    name: example
    namespace: foo
  spec:
    rules:
      - host: www.example.com
        http:
          paths:
            - backend:
                serviceName: exampleService
                servicePort: 80
              path: /
    # This section is only required if TLS is to be enabled for the Ingress
    tls:
        - hosts:
            - www.example.com
          secretName: example-tls

If TLS is enabled for the Ingress, a Secret containing the certificate and key must also be provided:

  apiVersion: v1
  kind: Secret
  metadata:
    name: example-tls
    namespace: foo
  data:
    tls.crt: <base64 encoded cert>
    tls.key: <base64 encoded key>
  type: kubernetes.io/tls
```

4. Verify the helm chart is installed

```
$ helm list --namespace ingress-basic
```

```
root@master:/home/ubuntu# helm list -n ingress-basic
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS          CHART                   APP VER
SION
nginx-ingress   ingress-basic   1               2020-12-15 06:54:41.605454932 +0000 UTC deployed        ingress-nginx-3.15.2    0.41.2

root@master:/home/ubuntu#
```

5. Verify that the load balancer service is created for the NGINX ingress controller and a dynamic public IP address is assigned to it

```
$ kubectl get all -n ingress-basic
```

```
root@master1:/home/ubuntu# kubectl get all -n ingress-basic
NAME                                                        READY   STATUS    RESTARTS   AGE
pod/nginx-ingress-ingress-nginx-controller-6d7cd9854f-cms2j   1/1     Running   0          34s
pod/nginx-ingress-ingress-nginx-controller-6d7cd9854f-ljvgq   1/1     Running   0          34s

NAME                                                 TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)
        AGE
service/nginx-ingress-ingress-nginx-controller             LoadBalancer   10.0.99.212    20.62.158.84   80:30621/TCP,443:3079
9/TCP   34s
service/nginx-ingress-ingress-nginx-controller-admission   ClusterIP      10.0.218.186   <none>         443/TCP
        34s

NAME                                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-ingress-ingress-nginx-controller   2/2     2            2           34s

NAME                                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-ingress-ingress-nginx-controller-6d7cd9854f   2         2         2       34s
root@master1:/home/ubuntu#
```

6.2 Creating simple demo applications

1. cd to the directory

```
$ cd kubernetes
$ ls ingress-
```

```
root@master1:/home/ubuntu# git clone https://github.com/mamtajha-ts/Kubernetes.git
Cloning into 'Kubernetes'...
remote: Enumerating objects: 222, done.
remote: Counting objects: 100% (222/222), done.
remote: Compressing objects: 100% (160/160), done.
remote: Total 222 (delta 77), reused 200 (delta 57), pack-reused 0
Receiving objects: 100% (222/222), 17.09 MiB | 30.65 MiB/s, done.
Resolving deltas: 100% (77/77), done.
root@master1:/home/ubuntu# cd Kubernetes/
root@master1:/home/ubuntu/Kubernetes# cd ingress-
ingress-app1.yaml   ingress-app2.yaml   ingress-route.yaml
root@master1:/home/ubuntu/Kubernetes# cd ingress-
```

2. View the content of ingress-app1.yaml file and see the definition of first application and its service in the file

```
$ vim ingress-app1.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: aks-helloworld-one
spec:
  replicas: 1
  selector:
    matchLabels:
      app: aks-helloworld-one
  template:
    metadata:
      labels:
        app: aks-helloworld-one
    spec:
      containers:
      - name: aks-helloworld-one
        image: neilpeterson/aks-helloworld:v1
        ports:
        - containerPort: 80
        env:
        - name: TITLE
          value: "Welcome to Azure Kubernetes Service (AKS)"
---
apiVersion: v1
kind: Service
metadata:
  name: aks-helloworld-one
spec:
  type: ClusterIP
  ports:
  - port: 80
  selector:
    app: aks-helloworld-one
~
~
```

3. View the content of ingress-app2.yaml file and see the definition of second application and its service in the file

```
$ vim ingress-app2.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: aks-helloworld-two
spec:
  replicas: 1
  selector:
    matchLabels:
      app: aks-helloworld-two
  template:
    metadata:
      labels:
        app: aks-helloworld-two
    spec:
      containers:
      - name: aks-helloworld-two
        image: neilpeterson/aks-helloworld:v1
        ports:
        - containerPort: 80
        env:
        - name: TITLE
          value: "AKS Ingress Demo"
---
apiVersion: v1
kind: Service
metadata:
  name: aks-helloworld-two
spec:
  type: ClusterIP
  ports:
  - port: 80
  selector:
    app: aks-helloworld-two
~
~
```

4. Create the deployment and services resources from both the files created above:

```
$ kubectl create -f ingress-app1.yaml -n ingress-basic
$ kubectl create -f ingress-app2.yaml -n ingress-basic
```

```
root@master1:/home/ubuntu/Kubernetes# kubectl create -f ingress-app1.yaml -n ingress-basic
deployment.apps/aks-helloworld-one created
service/aks-helloworld-one created
root@master1:/home/ubuntu/Kubernetes# kubectl create -f ingress-app2.yaml -n ingress-basic
deployment.apps/aks-helloworld-two created
service/aks-helloworld-two created
root@master1:/home/ubuntu/Kubernetes#
```

6.3 Create Ingress Route to route traffic to both the running applications

1. View the ingress-route.yaml file and see the rules defined in the file to route the traffic to both the applications

$ vim ingress-route.yaml

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: hello-world-ingress
  namespace: ingress-basic
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
    nginx.ingress.kubernetes.io/rewrite-target: /$2
spec:
  rules:
  - http:
      paths:
      - backend:
          serviceName: aks-helloworld-one
          servicePort: 80
        path: /(.*)
      - backend:
          serviceName: aks-helloworld-two
          servicePort: 80
        path: /hello-world-two(/|$)(.*)
~
~
~
~
~
```

2. Create the ingress resource from ingress-route.yaml and verify using kubectl get command

$ kubectl create -f ingress-route.yaml -n ingress-basic

```
root@master1:/home/ubuntu/Kubernetes# kubectl create -f  ingress-route.yaml -n ingress-basic
ingress.extensions/hello-world-ingress created
root@master1:/home/ubuntu/Kubernetes#
```

$ kubectl get ingress -n ingress-basic

```
root@master1:/home/ubuntu/Kubernetes# kubectl get ingress -n ingress-basic
NAME                 CLASS    HOSTS   ADDRESS        PORTS   AGE
hello-world-ingress  <none>   *       20.62.158.84   80      33s
root@master1:/home/ubuntu/Kubernetes#
```

6.4 Testing the ingress controller routes correctly to both the application

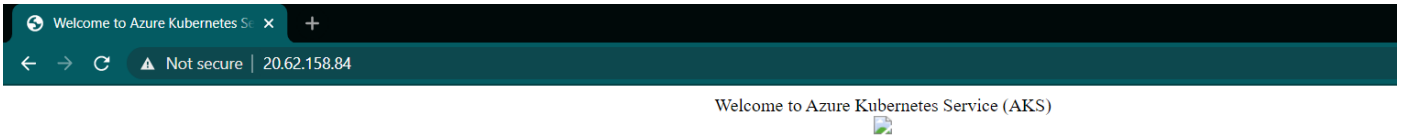
1. Open a web browser to the IP address of your NGINX ingress controller, *EXTERNAL_IP*. The first demo application should be displayed in the web browser,

```
root@master1:/home/ubuntu# kubectl get all -n ingress-basic
NAME                                                     READY   STATUS    RESTARTS   AGE
pod/nginx-ingress-ingress-nginx-controller-6d7cd9854f-cms2j   1/1     Running   0          34s
pod/nginx-ingress-ingress-nginx-controller-6d7cd9854f-ljvgq   1/1     Running   0          34s

NAME                                                TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)
       AGE
service/nginx-ingress-ingress-nginx-controller             LoadBalancer   10.0.99.212    20.62.158.84   80:30621/TCP,443:3079
9/TCP   34s
service/nginx-ingress-ingress-nginx-controller-admission   ClusterIP      10.0.218.186   <none>         443/TCP
       34s

NAME                                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-ingress-ingress-nginx-controller   2/2     2            2           34s

NAME                                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-ingress-ingress-nginx-controller-6d7cd9854f   2         2         2       34s
root@master1:/home/ubuntu# clear
```



Welcome to Azure Kubernetes Service (AKS)

2. Open a web browser to the IP address of your NGINX ingress controller with /hello-world-two path, EXTERNAL_IP /hello-world-two path. The second demo application should be displayed in the web browser,

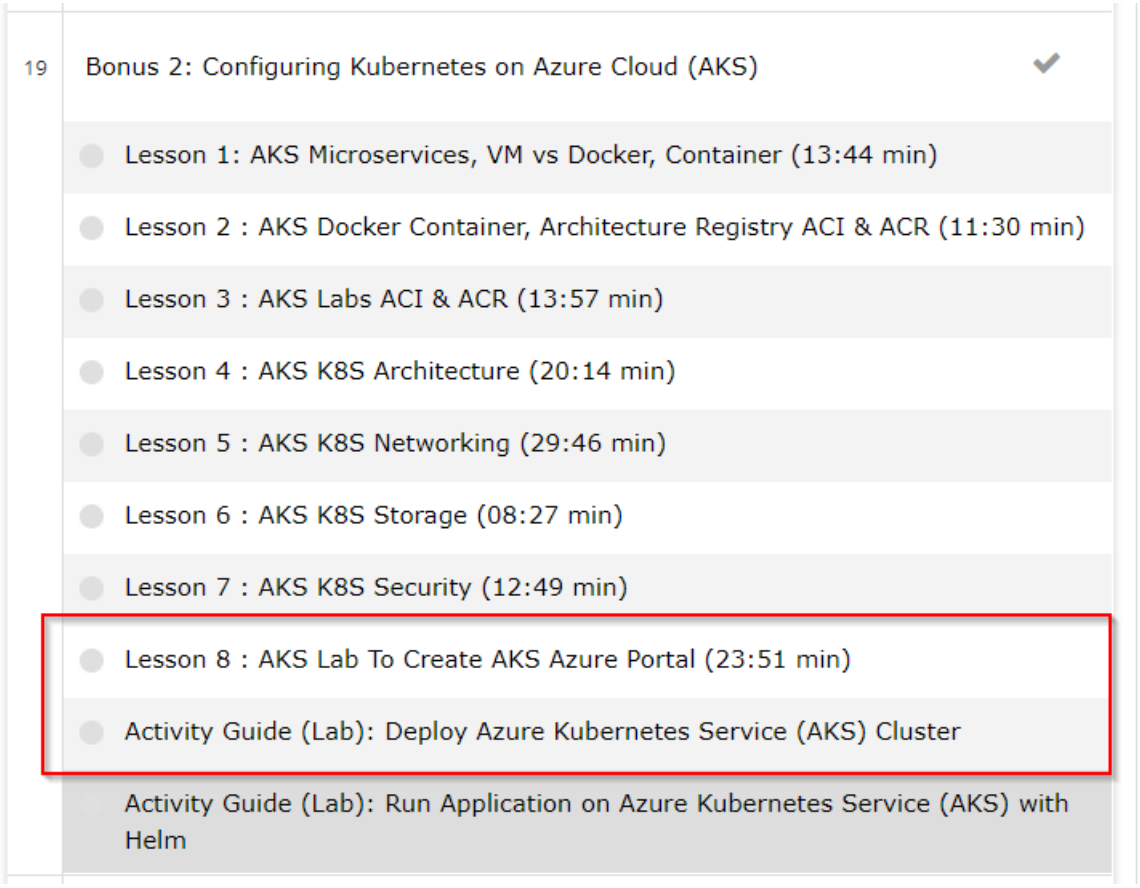
AKS Ingress Demo

6.5 Clean up resources created in this lab exercise

```
$ helm uninstall nginx-ingress --namespace ingress-basic
$ kubectl delete namespace ingress-basic
```

7 DYNAMIC PROVISIONING OF PERSISTENT VOLUMES

Note: Section 6 & 7 **Ingress-Controller** and **Dynamic Provisioning of Persistent Volumes** you need to perfrom in AKS Cluster not in your regular kubeadm cluster so before perfroming these sections first please follow Guide Deploy **Azure Kubernetes Service(AKS)** cluster guide from the portal.



Contents

7.1 Built-in storage classes

1. List the built-in storage classes in Azure AKS cluster

```
$ kubectl get sc
```

```
$
$  kubectl get sc
NAME                PROVISIONER                AGE
azurefile           kubernetes.io/azure-file   27h
azurefile-premium   kubernetes.io/azure-file   27h
default (default)   kubernetes.io/azure-disk   27h
managed-premium     kubernetes.io/azure-disk   27h
$
```

7.2 Creating Persistent Volume Claim

1. Verify the content of pvc.yaml file. The claim requests a disk named oracle-managed-disk that is 1GB in size with ReadWriteOnce access. The managed-premium storage class is specified as the storage class.

```
$ vim pvc.yaml
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: azure-managed-disk
spec:
  accessModes:
  - ReadWriteOnce
  storageClassName: managed-premium
  resources:
    requests:
      storage: 1Gi
~
~
~
~
~
~
~
~
~
```

```
$ kubectl create -f  pvc.yaml
```

2. Check the status of newly created pvc and see that dynamically a pv is created and bounded

```
$ kubectl get pvc
```

```
$
$ kubectl get pvc
NAME                 STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS      AGE
azure-managed-disk   Bound    pvc-4bb9012b-d917-4441-94e7-51eb74a4547a   1Gi        RWO            managed-premium   11s
$
```

7.3 Use PV in a Pod

1. The persistent volume claim has been created and the disk is successfully provisioned, a pod can be created with access to the disk. Check the content of pod-dynamicpv.yaml file

```
$ vim pod-dynamicpv.yaml
```

```
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
  - name: mypod
    image: nginx:1.15.5
    resources:
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 250m
        memory: 256Mi
    volumeMounts:
    - mountPath: "/mnt/azure"
      name: volume
  volumes:
    - name: volume
      persistentVolumeClaim:
        claimName: azure-managed-disk
~
~
~
~
~
~
```

2. Create the pod using apply command

```
$ kubectl apply -f pod-dynamicpv.yaml
```

```
$
$ kubectl apply -f pod-dynamicpv.yaml
pod/mypod created
$
```

3. Watch the creation of pod with -w option

```
$ kubectl get pods -w
```

```
$ kubectl get pods -w
NAME      READY   STATUS    RESTARTS   AGE
counter   1/1     Running   0          10h
mypod     1/1     Running   0          35s
```

4. Describe the pod and see that the volume details are mentions in pod specification

```
$ kubectl describe pod mypod
```

```
Volumes:
  volume:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  azure-managed-disk
    ReadOnly:   false
  default-token-v7f66:
    Type:       Secret (a volume populated by a Secret)
    SecretName:  default-token-v7f66
    Optional:    false
QoS Class:       Burstable
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason                  Age   From                                        Message
  ----    ------                  ----  ----                                        -------
  Normal  Scheduled               60s   default-scheduler                           Successfully assigned default/mypod to aks-agentpool-40017546-vmss000002
  Normal  SuccessfulAttachVolume  45s   attachdetach-controller                     AttachVolume.Attach succeeded for volume "pvc-4bb9012b-d917-4441-94e7-51eb7
4a4547a"
  Normal  Pulling                 39s   kubelet, aks-agentpool-40017546-vmss000002  Pulling image "nginx:1.15.5"
  Normal  Pulled                  28s   kubelet, aks-agentpool-40017546-vmss000002  Successfully pulled image "nginx:1.15.5"
  Normal  Created                 27s   kubelet, aks-agentpool-40017546-vmss000002  Created container mypod
  Normal  Started                 27s   kubelet, aks-agentpool-40017546-vmss000002  Started container mypod
$
```

7.4 Clean-up resources created in this lab exercise

```
$ kubectl delete -f  pvc.yaml
$ kubectl delete -f pod-dynamicpv.yaml
```

8 SUMMARY

In this guide we Covered:

- Taint and Tolerations
- Advanced Routing with Ingress-Controller
- Dynamic Provisioning of Persistent Volumes
- Deploying and Managing a StatefulSet Resource