# 

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

Contents

[1 Introduction 3](#_Toc63942367)

[2 Documentation 4](#_Toc63942368)

[2.1 Kubernetes Documentation 4](#_Toc63942369)

[2.2 Linux Commands and VIM Commands 4](#_Toc63942370)

[3 Previous Guides 5](#_Toc63942371)

[4 Advanced Scheduling with Taint and Tolerations 6](#_Toc63942372)

[4.1 Tainting a Node to Simulate Advanced Scheduling 6](#_Toc63942373)

[4.2 Creating Pod without Toleration 6](#_Toc63942374)

[4.3 Creating a Pod with Toleration 7](#_Toc63942375)

[4.4 Simulate eviction of Pod using NoSchedule effect 8](#_Toc63942376)

[5 Deploying and Managing a StatefulSet Resource 11](#_Toc63942377)

[5.1 Creating Logging namespace 11](#_Toc63942378)

[5.2 Setting up Elasticsearch application 11](#_Toc63942379)

[5.3 Pods in a StatefulSet 12](#_Toc63942380)

[5.4 Scaling up and down a Statefulset object 13](#_Toc63942381)

[5.5 Rolling update StatefulSets 14](#_Toc63942382)

[5.6 Clean Up resources created the lab exercise 16](#_Toc63942383)

[6 Advanced Routing with Ingress-Controller 17](#_Toc63942384)

[6.1 Deploying NGINX Ingress Controller using helm chart 18](#_Toc63942385)

[6.2 Creating simple demo applications 19](#_Toc63942386)

[6.3 Create Ingress Route to route traffic to both the running applications 21](#_Toc63942387)

[6.4 Testing the ingress controller routes correctly to both the application 22](#_Toc63942388)

[6.5 Clean up resources created in this lab exercise 23](#_Toc63942389)

[7 Dynamic Provisioning of Persistent Volumes 24](#_Toc63942390)

[7.1 Built-in storage classes 25](#_Toc63942391)

[7.2 Creating Persistent Volume Claim 25](#_Toc63942392)

[7.3 Use PV in a Pod 26](#_Toc63942393)

[7.4 Clean-up resources created in this lab exercise 27](#_Toc63942394)

[8 Summary 28](#_Toc63942395)

# 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

# Documentation

## Kubernetes Documentation

1. Taint & Tolerations

<https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/>

1. Ingress Controllers

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

1. Dynamic Volume Provisioning

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

1. StatefulSets

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

* 1. **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>

1. Basic VIM Commands

<https://coderwall.com/p/adv71w/basic-vim-commands-for-getting-started>

1. Popular VIM Commands

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

# 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*

# Advanced Scheduling with Taint and Tolerations

## Tainting a Node to Simulate Advanced Scheduling

1. View all the nodes in the cluster

$ kubectl get nodes



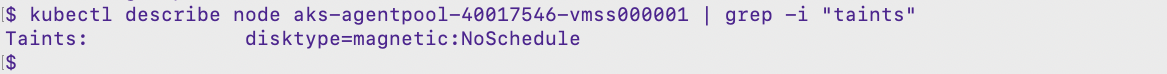
1. Taint one of the nodes by using its name

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



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

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



## Creating Pod without Toleration

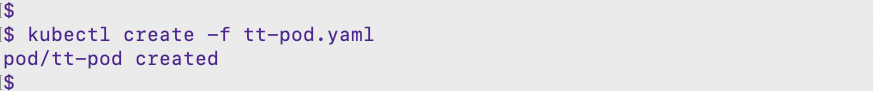
1. View the content of tt-pod.yaml file and create pod using the yaml file

$ vi tt-pod.yaml

A screenshot of a cell phone

Description automatically generated

$ kubectl create -f tt-pod.yaml



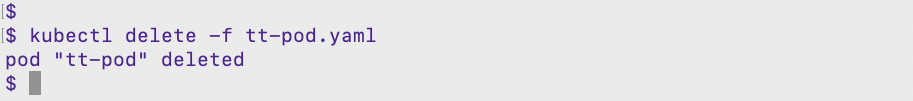
1. Verify the pod status. Notice that it was scheduled on the node which is not tainted

$ kubectl get pods -o wide



1. Delete the pod created in this task

$ kubectl delete -f tt-pod.yaml

****

## Creating a Pod with Toleration

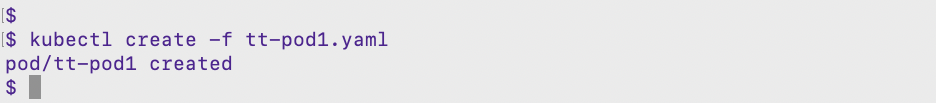
1. View the content of tt-pod1.yaml file and create pod using the yaml file

$ vi tt-pod1.yaml

A screenshot of a cell phone

Description automatically generated

$ kubectl create -f tt-pod1.yaml



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

$ kubectl get pods -o wide



1. Delete the pod created in this task

$ kubectl delete -f tt-pod1.yaml

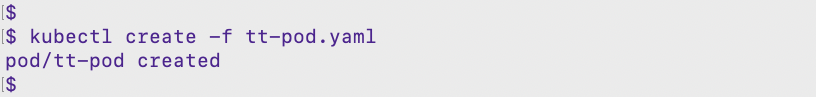
1. Delete the taint from the node

$ kubectl taint node <node\_name> disktype-  
$ kubectl describe nodes <node\_name> | grep -i taint

## Simulate eviction of Pod using NoSchedule effect

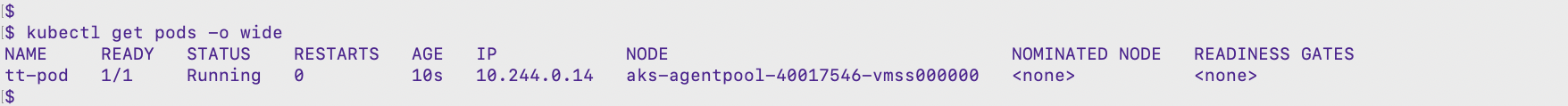
1. Again create a pod using tt-pod.yaml file. It doesn’t have any toleration defined

$ kubectl create -f tt-pod.yaml

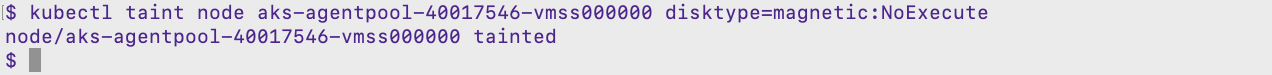


1. Taint the node on which the Pod was scheduled

$ kubectl get pods -o wide



$ kubectl taint node <node\_name> disktype=magnetic:NoExecute





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

$ kubectl get pods -o wide

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

$ kubectl get events

A screenshot of a cell phone

Description automatically generated

1. 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

# Deploying and Managing a StatefulSet Resource

## Creating Logging namespace

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

$ vim namespace.yaml

A screenshot of a cell phone

Description automatically generated

1. Creating namespace from above file

$ kubectl create -f namespace.yaml

A screenshot of a cell phone

Description automatically generated

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

$ kubectl get ns

A close up of a logo

Description automatically generated

## 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

A picture containing knife

Description automatically generated

1. 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

A screenshot of a cell phone

Description automatically generated

## 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](https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands/" \l "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

A screenshot of a cell phone

Description automatically generated

## Scaling up and down a Statefulset object

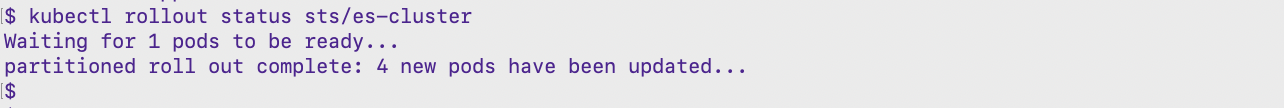
1. Scaling up the replicas from 3 to 4 for sts es-cluster. The StatefulSet controller scales the number of replicas.

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



1. 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 get pods

A screenshot of a cell phone

Description automatically generated

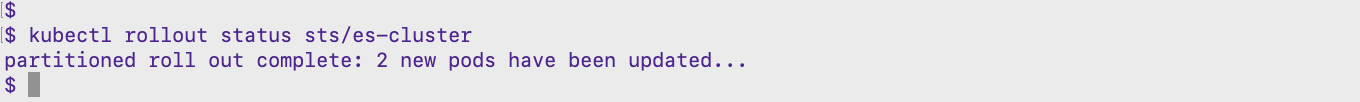
1. Scaling down the replicas from 4 to 2 for sts es-cluster. The StatefulSet controller scales the number of replicas.

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

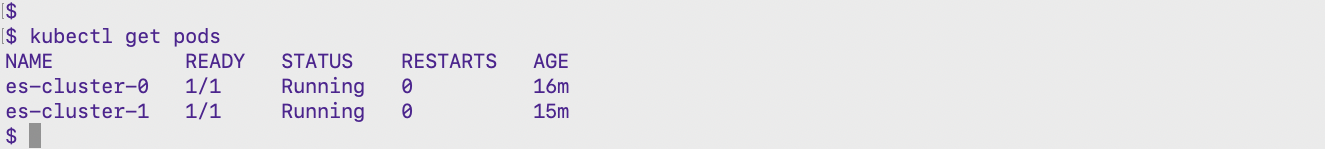


1. The controller deletes one Pod at a time, in reverse order with respect to its ordinal index, and it waits for each to completely shut down before deleting the next.

$ kubectl rollout status sts/es-cluster



$ kubectl get pods



## 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

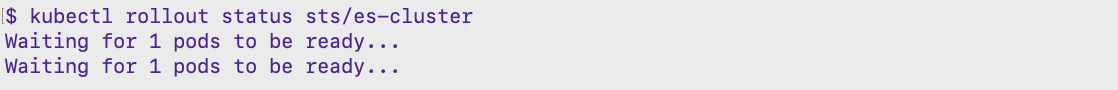
$ kubectl edit sts es-cluster

A screenshot of a social media post

Description automatically generated

1. 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 get pods -w

A screenshot of a cell phone

Description automatically generated

1. Verify the image version with describe command

$ kubectl describe sts es-cluster | grep Image

A picture containing knife

Description automatically generated

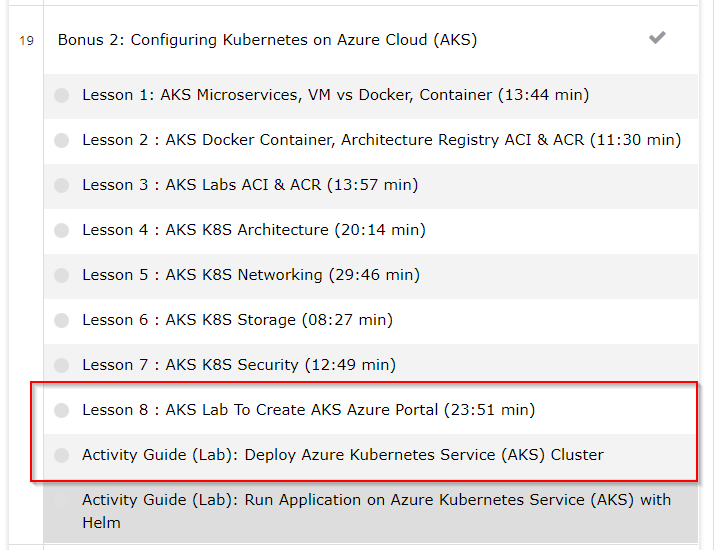
## Clean Up resources created the lab exercise

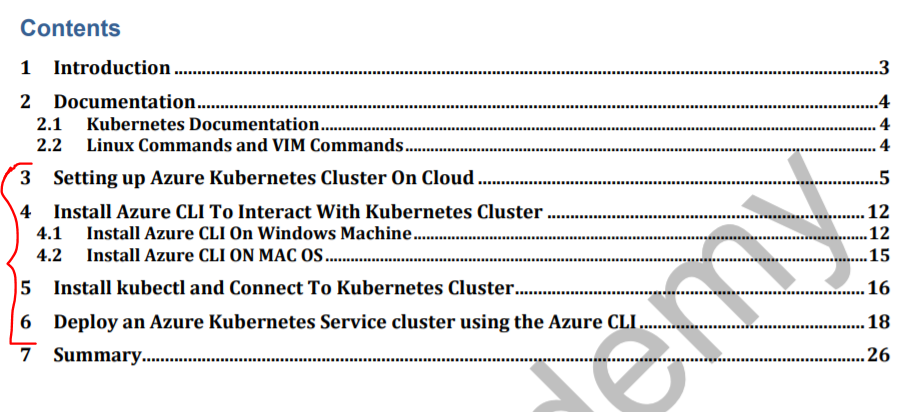
$ kubectl delete ns kube-logging

$ kubectl config set-context --current --namespace=default

# 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.

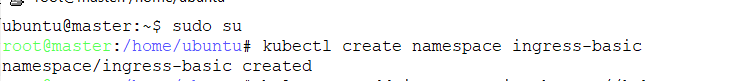




## Deploying NGINX Ingress Controller using helm chart

1. Create a namespace for your ingress resources

$ kubectl create namespace ingress-basic



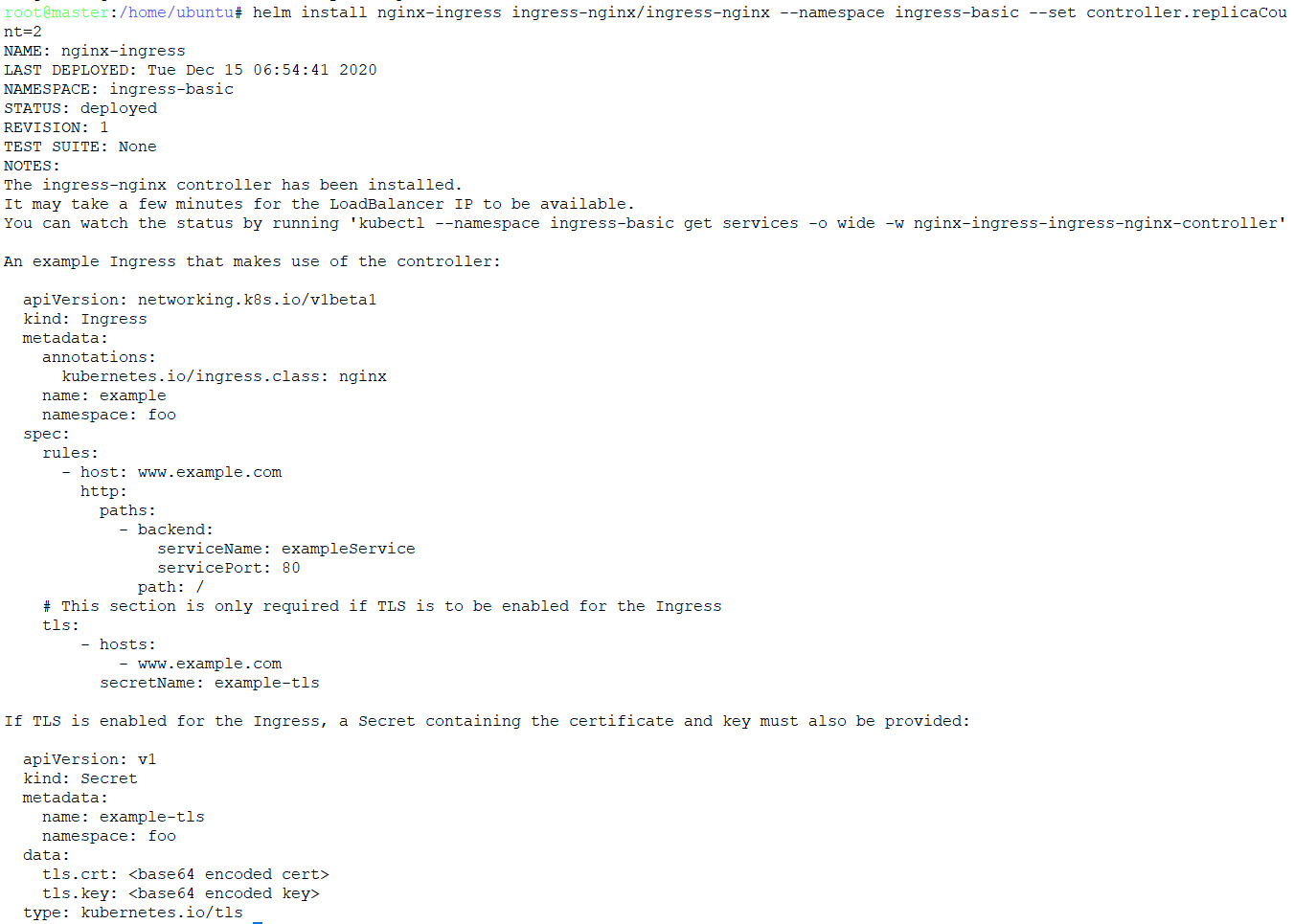
1. Add the official stable repository

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



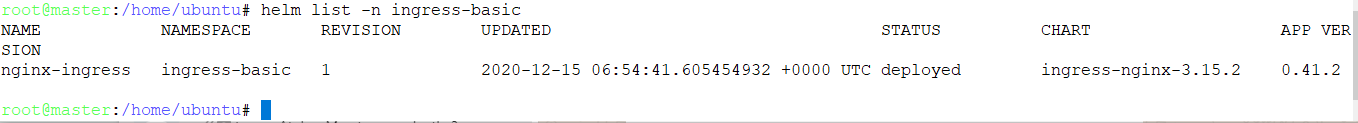
1. Use Helm to deploy an NGINX ingress controller

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



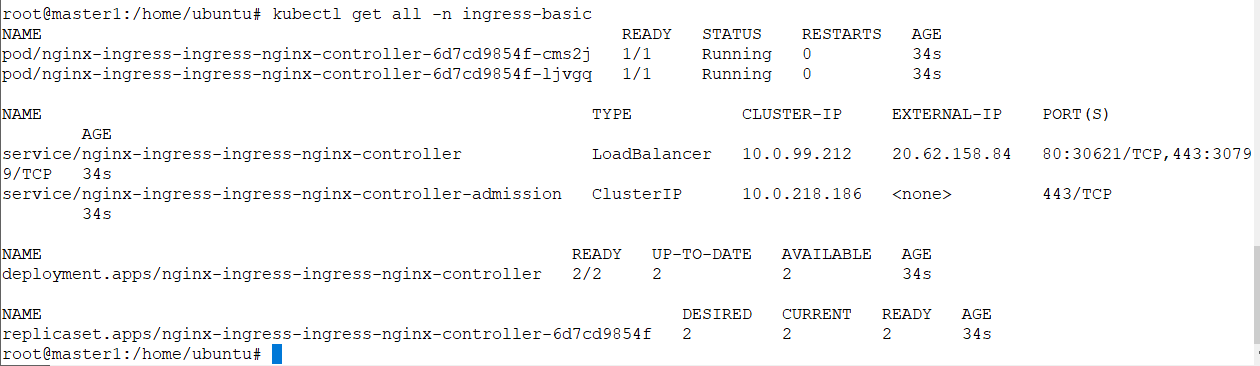
1. Verify the helm chart is installed

$ helm list --namespace ingress-basic



1. 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

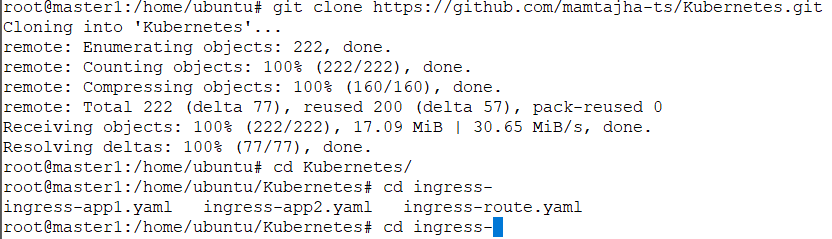


## Creating simple demo applications

1. cd to the directory

$ cd kubernetes

$ ls ingress-



1. 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

A screenshot of a social media post

Description automatically generated

1. 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

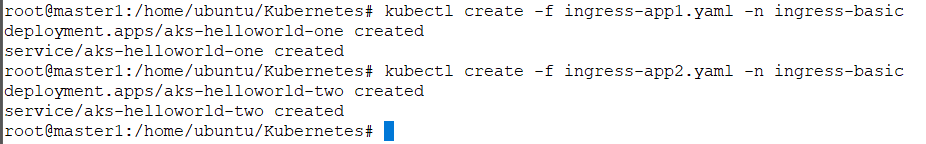
A screenshot of a social media post

Description automatically generated

1. 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



## 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

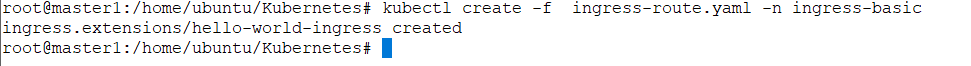
$ vim ingress-route.yaml

A screenshot of a cell phone

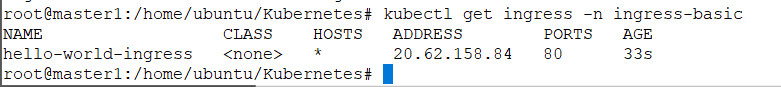
Description automatically generated

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

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

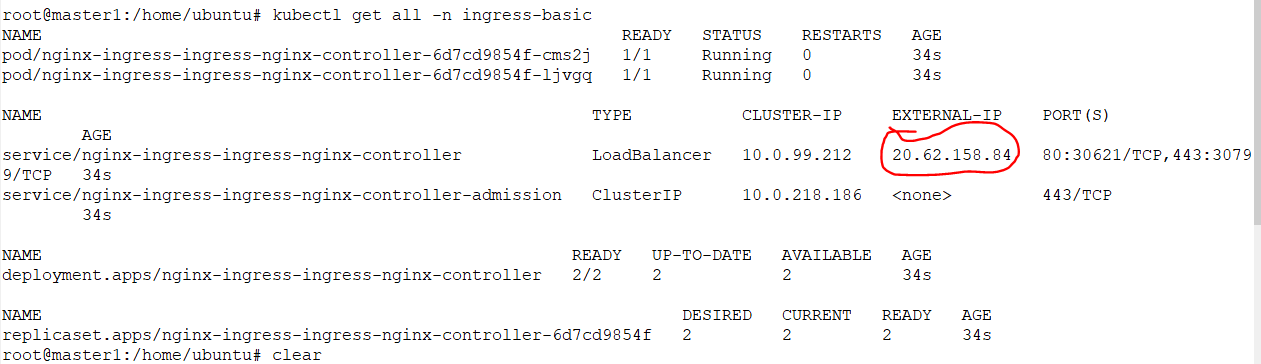


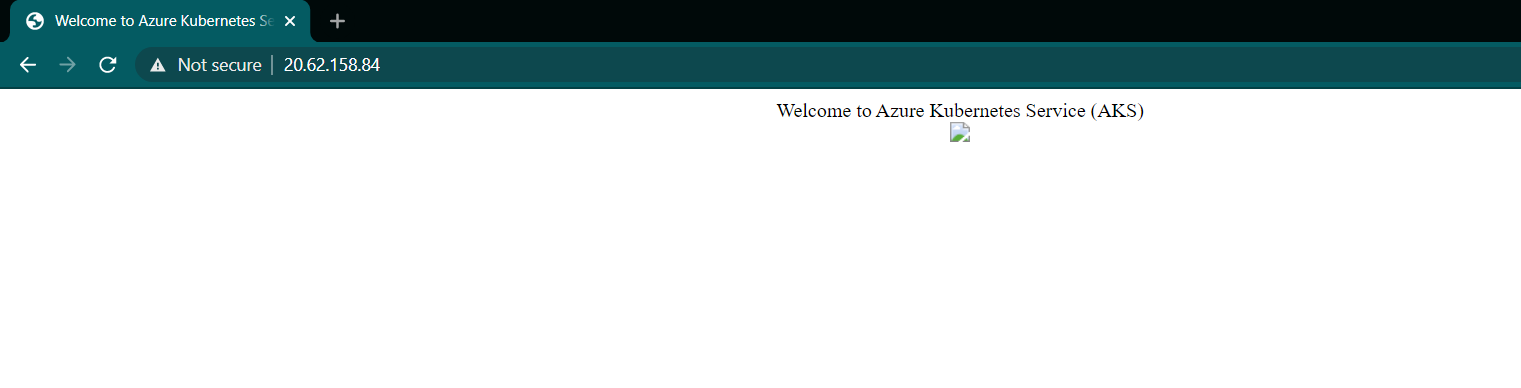
$ kubectl get ingress -n ingress-basic



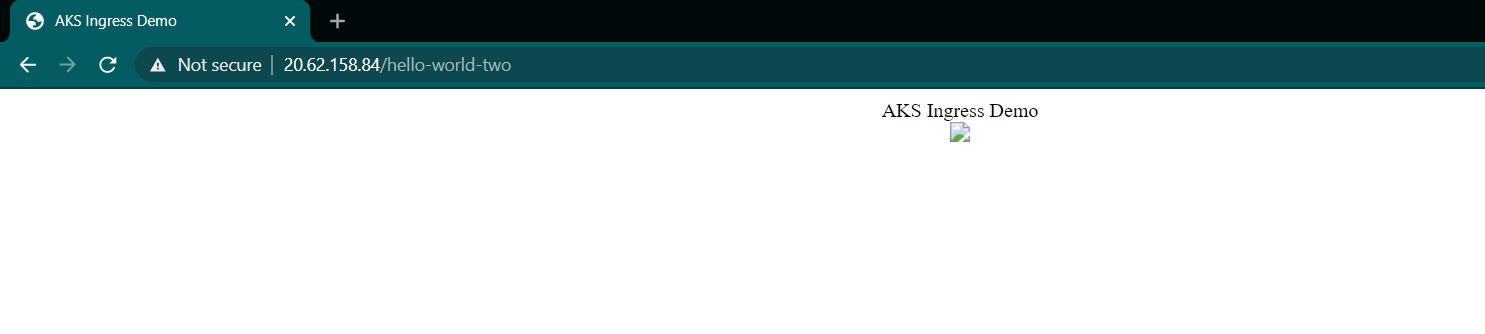
## Testing the ingress controller routes correctly to both the application

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,





1. 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,



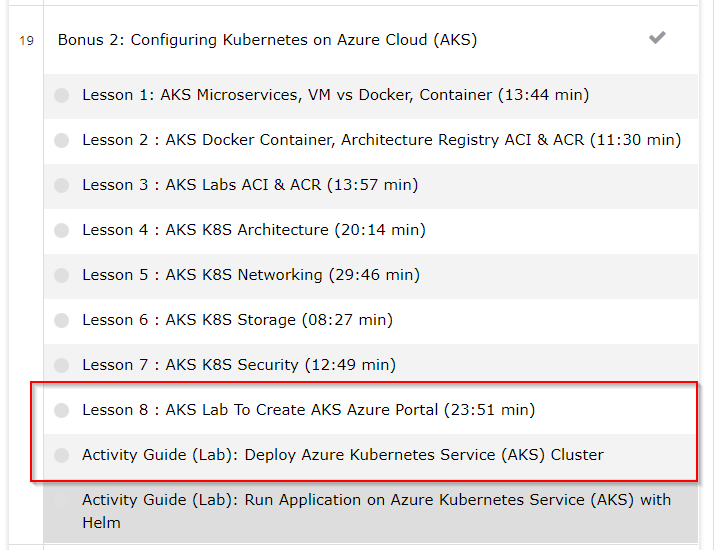
## Clean up resources created in this lab exercise

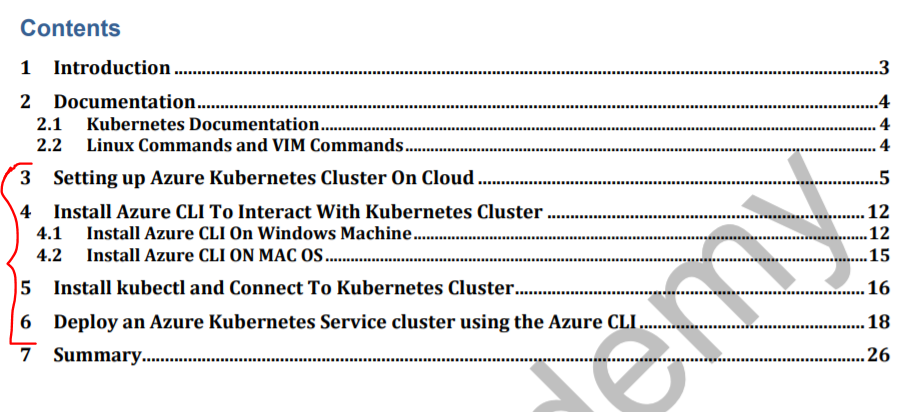
$ helm uninstall nginx-ingress --namespace ingress-basic

$ kubectl delete namespace ingress-basic

# 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.





## Built-in storage classes

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

$ kubectl get sc

A screenshot of a cell phone

Description automatically generated

## 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

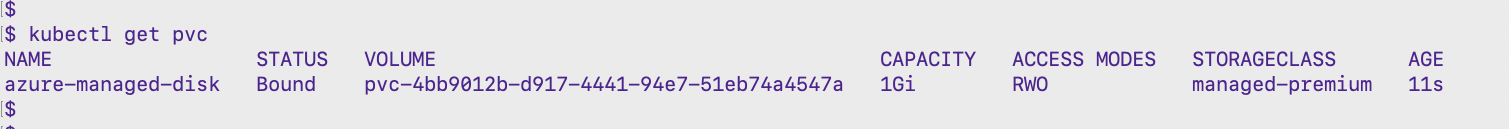
A screenshot of a cell phone

Description automatically generated

$ kubectl create -f pvc.yaml

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

$ kubectl get pvc



## 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

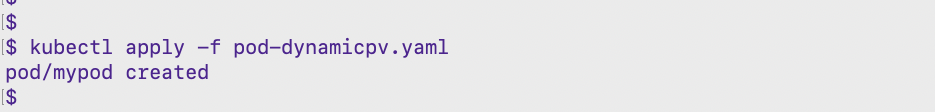
$ vim pod-dynamicpv.yaml

A screenshot of a cell phone

Description automatically generated

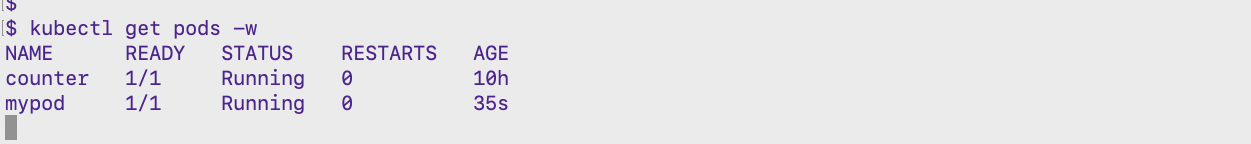
1. Create the pod using apply command

$ kubectl apply -f pod-dynamicpv.yaml



1. Watch the creation of pod with -w option

$ kubectl get pods -w



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

$ kubectl describe pod mypod

A screenshot of a cell phone

Description automatically generated

## Clean-up resources created in this lab exercise

$ kubectl delete -f pvc.yaml

$ kubectl delete -f pod-dynamicpv.yaml

# 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