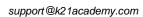




Taint & Tolerations, Ingress-Controller, Persistent Volumes, StatefulSet Resource

[Edition 2]
[Last Update 200920]

For any issues/help contact: support@k21academy.com



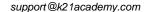




Contents

1 Introduction3

- 2 Documentation4
 - 2.1 Kubernetes Documentation4
 - 2.2 Linux Commands and VIM Commands4
- 3 Previous Guides5
- 4 Advanced Scheduling with Taint and Tolerations6
 - 4.1 Tainting a Node to Simulate Advanced Scheduling6
 - 4.2 Creating Pod without Toleration6
 - 4.3 Creating a Pod with Toleration7
 - 4.4 Simulate eviction of Pod using NoSchedule effect8
- 5 Advanced Routing with Ingress-Controller10
 - 5.1 Deploying NGINX Ingress Controller using helm chart10
 - 5.2 Creating simple demo applications11
 - 5.3 Create Ingress Route to route traffic to both the running applications 12
 - 5.4 Testing the ingress controller routes correctly to both the application 13
 - 5.5 Clean up resources created in this lab exercise 14
- 6 Dynamic Provisioning of Persistent Volumes 15
 - 6.1 Built-in storage classes 15
 - 6.2 Creating Persistent Volume Claim15
 - 6.3 Use PV in a Pod16
 - 6.4 Clean-up resources created in this lab exercise 17
- 7 Deploying and Managing a StatefulSet Resource 18
 - 7.1 Creating Logging namespace 18
 - 7.2 Setting up Elasticsearch application 18
 - 7.3 Pods in a StatefulSet19
 - 7.4 Scaling up and down a Statefulset object20
 - 7.5 Rolling update StatefulSets21
 - 7.6 Clean Up resources created the lab exercise 23
- 8 Summary24







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
 e_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
        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





\$ 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





```
    Skubectl 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 taint node <node_name> 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



5. Delete the pod and taint from the node

\$ kubectl delete -f tt-pod.yaml

Note: Pod is already deleted as soon as we tainted the node. So, you will get an error here if you try deleting the pod manually.

\$ kubectl taint node <node_name> disktype-

\$ kubectl describe nodes <node_name> | grep -i taint







5 ADVANCED ROUTING WITH INGRESS-CONTROLLER

5.1 Deploying NGINX Ingress Controller using helm chart

1. Create a namespace for your ingress resources

\$ kubectl create namespace ingress-basic

\$ kubectl create namespace ingress-basic
namespace/ingress-basic created

2. Add the official stable repository

\$ helm repo add stable https://kubernetes-charts.storage.googleapis.com/

```
$
$ helm repo add stable https://kubernetes-charts.storage.googleapis.com/
"stable" has been added to your repositories
```

3. Use Helm to deploy an NGINX ingress controller

\$ helm install nginx-ingress stable/nginx-ingress \

- --namespace ingress-basic \
- --set controller.replicaCount=2 \
- --set controller.nodeSelector."beta\.kubernetes\.io/os"=linux \
- --set defaultBackend.nodeSelector."beta\.kubernetes\.io/os"=linux

4. Verify the helm chart is installed

\$ helm list --namespace ingress-basic

\$ \$ helm list	namespace ingres	s-basic				j
NAME VERSION	NAMESPACE	REVISION	UPDATED	STATUS	CHART	APP
nginx-ingress 2.0	ingress-basic	1	2020-06-03 07:11:28.315155 +0530 IST	deployed	nginx-ingress-1.39.0	0.3





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 service -l app=nginx-ingress --namespace ingress-basic

5.2 Creating simple demo applications

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

```
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
         - containerPort: 80
        env:
           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
```

2. 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
            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
```



3. 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

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

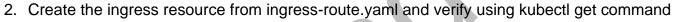
1. View the ingress-route.yaml file and see the rules defined in the file to route the traffic to both the applications

\$ 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: /$2
spec:
 rules:
  - http:
     paths:
      - backend:
          serviceName: aks-helloworld-one
          servicePort: 80
        path: /(.*)
      - backend:
          serviceName: aks-helloworld-two
          servicePort: 80
        path: /hello-world-two(/|$)(.*)
```



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

```
$ kubectl create -f ingress-route.yaml
ingress.extensions/hello-world-ingress created
```

\$ kubectl get ingress -n ingress-basic

```
$
$ kubectl get ingress -n ingress-basic
NAME HOSTS ADDRESS PORTS AGE
hello-world-ingress * 10.240.0.4 80 25m
$
```

5.4 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,







Welcome to Azure Kubernetes Service (AKS)



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



AKS Ingress Demo



5.5 Clean up resources created in this lab exercise

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

\$ kubectl delete namespace ingress-basic





6 DYNAMIC PROVISIONING OF PERSISTENT VOLUMES

6.1 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
```

6.2 Creating Persistent Volume Claim

 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





\$ 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

```
$ | Skubectl get pvc | STATUS | VOLUME | CAPACITY | ACCESS MODES | STORAGECLASS | AGE | azure-managed-disk | Bound | pvc-4bb9012b-d917-4441-94e7-51eb74a4547a | 16i | RWO | managed-premium | 11s | Status | Statu
```

6.3 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:
               PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
   Type:
   ClaimName: azure-managed-disk
   ReadOnly: false
 default-token-v7f66:
              Secret (a volume populated by a Secret)
   Type:
   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"
                             28s kubelet, aks-agentpool-40017546-vmss000002 Successfully pulled image "nginx:1.15.5"
 Normal Pulled
 Normal Created
                             27s kubelet, aks-agentpool-40017546-vmss000002 Created container mypod
 Normal Started
                               27s kubelet, aks-agentpool-40017546-vmss000002 Started container mypod
```

6.4 Clean-up resources created in this lab exercise

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





7 DEPLOYING AND MANAGING A STATEFULSET RESOURCE

7.1 Creating Logging namespace

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

\$ vim namespace.yaml

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
                            16h
                  Active
kube-logging
                  Active
                            13s
kube-node-lease
                            16h
                  Active
kube-public
                  Active
                            16h
kube-system
                  Active
                            16h
```

7.2 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
            READY AGE
es-cluster 3/3
$ kubectl get pods --namespace=kube-logging
                        READY
                                STATUS
                                          RESTARTS
                                                     AGE
es-cluster-0
                                Running
                        1/1
                                                     25m
                                          0
es-cluster-1
                        1/1
                                Running
                                          0
                                                     6m27s
es-cluster-2
                        1/1
                                Running
                                                     4m51s
```

7.3 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 <u>kubectl exec</u> 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
                                                     3h14m
                        1/1
                                Running
                                          0
es-cluster-1
                        1/1
                                Running
                                          0
                                                     174m
                                          0
                                                     172m
es-cluster-2
                        1/1
                                Running
fluentd-2vw2j
                                          0
                        1/1
                                Running
                                                     162m
fluentd-9f298
                        1/1
                                          0
                                                     162m
                                Running
fluentd-m9hxb
                        1/1
                                Running
                                                     162m
                                Running
kibana-cd68dcfb-pjnhc 1/1
                                         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
```

7.4 Scaling up and down a Statefulset object

 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
                                            9m40s
                                 0
                                 0
                                            8m54s
es-cluster-1
               1/1
                       Running
es-cluster-2
              1/1
                                 0
                                            8m8s
                       Running
es-cluster-3
                       Running
              1/1
                                            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
```





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...
```

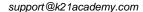
\$ kubectl get pods

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

7.5 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
  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
```

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
es-cluster-0
                          Running
PodInitializing
                 1/1
                                                            28m
es-cluster-1
                 0/1
                                                            2m1s
es-cluster-1
                 1/1
1/1
                          Running
Terminating
                                                            2m4s
                                                            28m
es-cluster-0
es-cluster-0
                          Terminating
es-cluster-0
                 0/1
0/1
                          Terminating
Terminating
                                               0
0
                                                            28m
es-cluster-0
                                                            28m
es-cluster-0
                          Pending
es-cluster-0
                          Pending
                 0/1
                                                            05
                 0/1
es-cluster-0
                          Init:0/3
                                                            0s
es-cluster-0
                 0/1
                                                            15s
es-cluster-0
                 0/1
                          Init:2/3
                                                            17s
es-cluster-0
                 0/1
                          PodInitializing
                                                            18s
es-cluster-0
```

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
$
```

7.6 Clean Up resources created the lab exercise

\$ kubectl delete ns kube-logging

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







8 SUMMARY

In this guide we Covered:

- Taint and Tolerations
- Advanced Routing with Ingress-Controller
- Dynamic Provisioning of Persistent Volumes

