

Lab Module 3: Intermediate Kubernetes Concepts



Estimated Duration: 60 minutes

Module 3 Table of Contents

[Create a Basic Cluster](#)

[Exercise: Working with Multi-Container Pods, Ephemeral Volumes and ConfigMaps](#)

[Exercise: Working with Persistent Volumes, Persistent Volume Claims and Secrets](#)

[Exercise: Using Ingress Resources and Ingress Controller to Control External Access](#)

[Shutdown or Delete the Cluster](#)

Create a Basic Cluster

For the exercises in this module, you'll need simple AKS cluster.

Task 1 - Create an AKS cluster

1. Select the region closest to your location. Use '**eastus**' for United States workshops, '**westeurope**' for European workshops. Ask your instructor for other options in your region: @lab.DropDownList(region)
[eastus,westus,canadacentral,westeurope,centralindia,australiaeast]
2. Define variables (update as needed)

```
$INSTANCE_ID="@lab.LabInstance.Id"  
$AKS_RESOURCE_GROUP="azure-$($INSTANCE_ID)-rg"  
$LOCATION="@lab.Variable(region)"  
$VM_SKU="Standard_D2as_v5"
```

```
$AKS_NAME="aks-$($INSTANCE_ID)"  
$NODE_COUNT="3"
```

3. Create Resource Group

```
az group create --location $LOCATION  
--resource-group $AKS_RESOURCE_GROUP
```

4. Create Basic cluster.

```
az aks create --node-count $NODE_COUNT  
--generate-ssh-keys  
--node-vm-size $VM_SKU  
--name $AKS_NAME  
--resource-group $AKS_RESOURCE_GROUP
```

5. Connect to local environment

```
az aks get-credentials --name $AKS_NAME  
--resource-group $AKS_RESOURCE_GROUP
```

6. Verify connection

```
kubectl get nodes
```

Exercise: Working with Multi-Container Pods, Ephemeral Volumes and ConfigMaps

This exercise shows how multiple containers within the same pod can use volumes to communicate with each other and how you can use *ConfigMaps* to mount settings files into containers.

The example Deployment below configures a MySQL instance to write its activities to a log file. Another container reads that log file, and in a production setting, would send the contents to an external log aggregator. In this example, the 2nd container just outputs the *tail* of the log file to the console.

Task 1 - Review and deploy ConfigMaps

1. Open a Windows Terminal window (defaults to PowerShell).



2. Change current folder to **Module3**

```
cd C:\k8s\labs\Module3
```

3. Review the **mysql-initdb-cm.yaml** file. This script will run automatically when MySql starts.

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-initdb-cm
data:
  initdb.sql: |
    CREATE DATABASE sample;
    USE sample;
    CREATE TABLE friends (id INT, name VARCHAR(256), age INT, gender VARCHAR(3));
    INSERT INTO friends VALUES (1, 'John Smith', 32, 'm');
    INSERT INTO friends VALUES (2, 'Lilian Worksmith', 29, 'f');
    INSERT INTO friends VALUES (3, 'Michael Rupert', 27, 'm');

```

When the ConfigMap is mounted in a container, it will create a file called **initdb.sql**.

4. Deploy the ConfigMap.

```
kubectl apply -f mysql-initdb-cm.yaml
```

5. Configure MySQL to generate activity logs and save them to a file. Review the contents of **mysql-cnf-cm.yaml**. This file will replace the default **my.cnf** config file in the container.

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-cnf-cm
data:
  my.cnf: |
    !includedir /etc/mysql/conf.d/
    !includedir /etc/mysql/mysql.conf.d/

    [mysqld]
    #Set General Log
    general_log = on
    general_log_file=/usr/log/general.log

```

6. Deploy the ConfigMap.

```
kubectl apply -f mysql-cnf-cm.yaml
```

Task 2 - Review and deploy multi-container Deployment

1. Review the Volumes section of **mysql-dep.yaml**.

```

volumes:
  - name: mysql-initdb
    configMap:
      name: mysql-initdb-cm
  - name: mysql-cnf
    configMap:

```

```
    name: mysql-cnf-cm
  - name: mysql-log
    emptyDir: {}
```

Notice that both ConfigMaps are being mapped to volumes and an empty directory is being created for the logs.

2. Review the volumeMounts section of **mysql-dep.yaml**.

```
volumeMounts:
  - name: mysql-initdb
    mountPath: /docker-entrypoint-initdb.d
  - name: mysql-cnf
    mountPath: /etc/mysql/my.cnf
    subPath: my.cnf
  - name: mysql-log
    mountPath: /usr/log/
```

The init file is mapped to the known MySQL folder. Anything placed in that folder is executed by MySQL during its startup.

The config file is mapped to the MySQL config folder. However, there are other config files already in that folder that MySQL needs to access. As a result, replacing the entire folder with the contents of the ConfigMap will prevent MySQL from working, because the other config files won't be there.

That's where the *subPath* property comes into play. It allows you to mount only a subset of your volume (in this case only a single file), leaving the existing content in place at the destination.

The final mount point points to the empty directory created to hold the logs.

3. Review the second container definition.

```
- name: logreader
  image: busybox
  command:
    - "/bin/sh"
  args:
    - "-c"
    - "tail -f /usr/log/general.log;"
  volumeMounts:
    - name: mysql-log
      mountPath: /usr/log/
```

Notice that the same empty directory is mounted to this container and that the container simply executes a tail command, outputting the last line in the file.

4. Apply the deployment.

```
kubectl apply -f mysql-dep.yaml
```

Task 3 - Confirm communications between containers

1. Get list of Pods.

```
kubectl get pods
```

2. Once the Pod is running, look at the log of the second container.

```
kubectl logs -c logreader -f "name of mysql-dep-xxxxx pod"
```

3. Open another shell window and execute into the first container

```
kubectl exec -it -c mysql "name of mysql-dep-xxxxx pod" -- bash
```

4. In the shell, type **mysql** to enter the MySQL console.

```
mysql
```

5. Type the following commands in the **mysql>** prompt:

```
use sample;  
select * from friends;
```

6. Switch to the other window and verify that the second container is showing the log.

```
mysqld, Version: 5.7.35-log (MySQL Community Server (GPL)). started with:  
Tcp port: 0 Unix socket: /var/run/mysqld/mysqld.sock  
Time           Id  Command   Argument  
2021-08-05T03:57:25.617816Z      2 Connect   root@localhost on  using Socket  
2021-08-05T03:57:25.618005Z      2 Query     select @@version_comment limit 1  
2021-08-05T03:57:43.601557Z      2 Query     SELECT DATABASE()  
2021-08-05T03:57:43.601797Z      2 Init DB   sample  
2021-08-05T03:57:43.602822Z      2 Query     show databases  
2021-08-05T03:57:43.603177Z      2 Query     show tables  
2021-08-05T03:57:43.603323Z      2 Field List  friends  
2021-08-05T03:57:47.077235Z      2 Query     select * from friends
```

7. Exit out of the **MySQL** console.

```
exit;
```

8. Exit out of the container.

```
exit
```

[Module 3 Table of Contents](#)

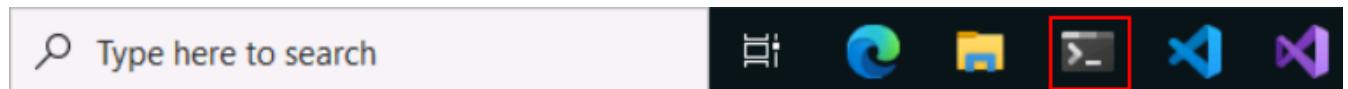
[List of Modules](#)

Exercise: Working with Persistent Volumes, Persistent Volume Claims and Secrets

This exercise shows an example of using a secret to store the password needed by an Azure File Share. You will first create the Azure File Share, get its Access Key and then configure a secret to use that access key to connect a volume to a Pod.

Task 1 - Create an Azure Storage Account and an Azure File Share

1. Open a Windows Terminal window (defaults to PowerShell).



2. Login to Azure.

```
az login  
az account set --subscription "Azure Pass - Sponsorship"
```

3. Define variables.

```
$INSTANCE_ID="@lab.LabInstance.Id"  
$AKS_RESOURCE_GROUP="azure-$($INSTANCE_ID)-rg"  
$LOCATION="@lab.Variable(region)"  
$STORAGE_ACCOUNT_NAME="sa$($INSTANCE_ID)"  
$SHARE_NAME="share$($INSTANCE_ID)"
```

4. Create the Azure Storage Account.

```
az group create --location $LOCATION --resource-group $AKS_RESOURCE_GROUP  
  
az storage account create --name $STORAGE_ACCOUNT_NAME `  
    --resource-group $AKS_RESOURCE_GROUP `  
    --sku Standard_LRS
```

5. Create the Azure File Share.

```
az storage share create --name $SHARE_NAME `  
    --connection-string `  
    $(az storage account show-connection-string `  
        --name $STORAGE_ACCOUNT_NAME `  
        --resource-group $AKS_RESOURCE_GROUP -o tsv)
```

6. The **Account Name** and **Account Key** will be echoed to the screen for reference.

```
$STORAGE_KEY=$(az storage account keys list `  
    --resource-group $AKS_RESOURCE_GROUP `  
    --account-name $STORAGE_ACCOUNT_NAME `  
    --query "[0].value" -o tsv)  
  
$STORAGE_ACCOUNT_NAME  
$STORAGE_KEY
```

Task 2 - Create a Namespace for this lab

All the objects related to this lab will be self-contained in a Namespace. When you're done, delete the Namespace and all the objects it contains will also be deleted. The the Persistent Volume, which is scoped across the entire cluster, will remain.

1. Create a new Namespace.

```
kubectl create ns lab3volume
```

2. Set the new Namespace as the current namespace, so it doesn't have to be specified in future commands.

```
kubectl config set-context --current --namespace lab3volume
```

Task 3 - Create a Secret imperatively

1. The **Account Name** and **Account Key** need to be saved into a Kubernetes Secret object. The best way to do this is to create the Secret imperatively:

```
kubectl create secret generic azure-secret \
  --from-literal=azurestorageaccountname=$STORAGE_ACCOUNT_NAME \
  --from-literal=azurestorageaccountkey=$STORAGE_KEY \
  --dry-run=client -o yaml > azure-secret.yaml
```

2. A YAML file is generated with all the information needed to create the secret. Review the new file.

```
cat azure-secret.yaml
```

NOTE: It's best to create the file first and then apply it so you can repeat the process if needed.

```
apiVersion: v1
data:
  azurestorageaccountkey: NlhCeHlMTUZyS2NKN0xlNDZFc
  EpnV3lSUEFmR2diZFdxGh2d0E9PQ==
  azurestorageaccountname: a2l6c2EyMzQ10Q==
kind: Secret
metadata:
  creationTimestamp: null
  name: azure-secret
```

3. Apply the secret in the **default** namespace. Since the Persistent Volume is not a namespaced resource, it will expect to find the secret in the **default** namespace.

```
kubectl apply -f azure-secret.yaml -n default
```

Task 4 - Create a Persistent Volume and a Persistent Volume Claim

The first step to accessing an Azure File Share is creating a *PersistentVolume* object that connects to that share.

1. Review the value of **SHARE_NAME** created in Task 1 above.

```
echo $SHARE_NAME
```

CRITICAL STEP:

2. Review the contents of **azurefile-pv.yaml**. Make sure the **shareName** property matches the value of **SHARE_NAME**.

```
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  azureFile:
    secretName: azure-secret
    shareName: aksshare
    readOnly: false
  mountOptions:
    - dirMode=0777
    - fileMode=0666
```

3. Also notice there's a label associated with the persistent volume:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: azurefile-pv
  labels:
    storage-provider: azurefileshare
```

4. When the file is ready, apply it.

```
kubectl apply -f azurefile-pv.yaml
```

5. Make sure the *PersistentVolume* was created and has a status of *Available*.

```
kubectl get pv
```

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM	STORAGECLASS	REASON	AGE
azurefile-pv	5Gi	RWX	Retain	Available				48s

6. Review the contents of **azurefile-pvc.yaml**. Notice that the label selector is set to look for the persistent volume:

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: azurefile-pvc
spec:
  selector:
    matchLabels:
      storage-provider: azure-file-share
  storageClassName: azurefile-csi
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi

```

Also the label selector is set to find the PV.

7. Apply the Persistent Volume Claim.

```
kubectl apply -f azurefile-pvc.yaml
```

8. Verify that the Persistent Volume Claim is *bound* to the Persistent Volume.

```
kubectl get pvc
```

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	AGE
azurefile-pvc	Bound	azurefile-pv	5Gi	RwX		3s

9. Review the PersistentVolume again

```
kubectl get pv
```

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM	STORAGECLASS
azurefile-pv	5Gi	RwX	Retain	Bound	lab3volume/azurefile-pvc	azurefile-csi

Notice that the PVC claimed the PV:

Task 5 - Use the Persistent Volume Claim in a Pod

Now that the connection is configured to the Azure File Share, create a Pod to use the Persistent Volume Claim.

1. Review the contents of **test-pvc-pod.yaml**. Notice how the *persistentVolumeClaim* is being used and *mountPath* of the *volumeMount*.

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pvc-pod
spec:
  containers:
    - image: nginx
      name: nginx
      volumeMounts:
        - mountPath: /sharedfolder
          name: azure-file-volume
  volumes:
    - name: azure-file-volume
      persistentVolumeClaim:
        claimName: azurefile-pvc
```

2. Apply the Pod.

```
kubectl apply -f test-pvc-pod.yaml
```

3. Confirm the Pod is running. It might take a few minutes for the Pod to connect to the PVC.

```
kubectl get pod test-pvc-pod
```

4. Once the Pod is running, shell into the Pod.

```
kubectl exec -it test-pvc-pod -- bash
```

5. Change into the shared folder, create a file and confirm it's there.

```
cd /sharedfolder
touch abc123.txt
ls -l
```

6. Open the Azure Portal in a browser.

7. Navigate to the storage account created earlier (as defined in **STORAGE_ACCOUNT_NAME**).

8. Click on the File shares selection.

kizsa23459 | File shares

Storage account | Directory: Microsoft

Search (Ctrl+ /) < + File share Refresh

Overview Activity log Tags Diagnose and solve problems Access Control (IAM) Data migration Events Storage browser (preview)

Data storage

Containers File shares

The screenshot shows the 'File shares' section of the Azure Storage account settings. On the left, there's a sidebar with various navigation links like Overview, Activity log, Tags, etc. The 'File shares' link is highlighted. The main area has a search bar at the top. Below it, it says 'File share settings' and 'Active Directory: Not configured'. There's a search bar for 'Search file shares by prefix (c...)' and a list of file shares. One share is listed: 'Name' followed by 'aksshare' with a small icon next to it.

9. Click on the file share name.

aksshare ...

File share

Search (Ctrl+ /) < Connect Upload

Overview Diagnose and solve problems Access Control (IAM) ...

Search files by prefix

Name abc123.txt

The screenshot shows the details for the 'aksshare' file share. It has a similar header with a search bar and navigation links. The main area shows a list of files. One file is highlighted: 'Name' followed by 'abc123.txt' with a small icon next to it.

10. You should see the file you created in the container. Click the **Upload** button and upload a file from your hard drive to the share.

 Connect  Upload 

 Search files by prefix

Name

 abc123.txt

 simple-pod.yaml

10. Get a list of files in the shared folder again.

```
ls -l
```

Both files should be in the shared folder:

```
root@test-pvc:/sharedfolder# ls -l
total 1
-rwxrwxrwx 1 1000 1000 0 Jul 29 03:55 abc123.txt
-rwxrwxrwx 1 1000 1000 205 Jul 29 04:02 simple-pod.yaml
```

This confirms that the **sharedfolder** is mapped to the Azure File Share.

11. Exit the container.

```
exit
```

Task 6 - Set the current Namespace back to default

1. Set the current namespace back to default so all of these objects stay separate from the other labs.

```
kubectl config set-context --current --namespace default
```

2. **OPTIONAL:** If you prefer, you can delete the namespace. This will delete all the objects (secrets, persistent volume claim, pods) from the cluster. The only object that will remain will be the Persistent Volume, because it's not tied to any specific namespace.

```
kubectl delete ns lab3volume
```

[Module 3 Table of Contents](#)

[List of Modules](#)

Exercise: Using Ingress Resources and Ingress Controller to control external access

Ingress resources and 3rd-party Ingress Controllers can be used to centrally control external access to the cluster.

Task 1 - Install NGinx Ingress Controller

1. Change current folder to **C:\k8s\labs\Module3**.

```
cd C:\k8s\labs\Module3
```

2. Install **Nginx Ingress Controller** using **Helm**:

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

```
helm install nginx-ingress ingress-nginx/ingress-nginx \
--namespace ingress-nginx --create-namespace \
--set controller.nodeSelector."kubernetes\.io/os"=linux \
--set defaultBackend.nodeSelector."kubernetes\.io/os"=linux \
--repo https://kubernetes.github.io/ingress-nginx
```

3. Wait a few minutes to allow the Nginx Load Balancer service to acquire an external IP address.

4. Query the services in the **ingress-nginx** namespace.

```
kubectl get svc -n ingress-nginx
```

Get the *EXTERNAL IP* of the Load Balancer:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
AGE				
service/ingress-nginx-controller	LoadBalancer	10.0.240.47	52.146.66.160	
80:32616/TCP,443:31149/TCP 108s				
service/ingress-nginx-controller-admission	ClusterIP	10.0.244.155	<none>	443/TCP
108s				

Task 2 - Install Deployments, Services and Ingress Resource in a separate namespace

1. Create the **dev** namespace

```
kubectl create ns dev
```

2. Change current folder to **C:\k8s\labs\Module3**.

```
cd C:\k8s\labs\Module3
```

3. Apply the deployments and services.

```
kubectl apply -f blue-dep.yaml -f blue-svc.yaml -n dev  
kubectl apply -f red-dep.yaml -f red-svc.yaml -n dev
```

4. Review the contents of **C:\k8s\labs\Module3\colors-ingress.yaml** file in an editor. Notice the *path* setting to route traffic to the correct service.

```
apiVersion: networking.k8s.io/v1  
kind: Ingress  
metadata:  
  annotations:  
    kubernetes.io/ingress.class: nginx  
    nginx.ingress.kubernetes.io/rewrite-target: /$1  
  name: colors-ingress  
spec:  
  rules:  
    - http:  
        paths:  
          - path: /blue/(.*)  
            pathType: Prefix  
            backend:  
              service:  
                name: blue-svc  
                port:  
                  number: 8100  
          - path: /red/(.*)  
            pathType: Prefix  
            backend:  
              service:  
                name: red-svc  
                port:  
                  number: 8100
```

5. Apply the Ingress resource.

```
kubectl apply -f colors-ingress.yaml -n dev
```

6. Verify the ingress resource was created.

```
kubectl get ing -n dev
```

Returns:

NAME	CLASS	HOSTS	ADDRESS	PORTS	AGE
colors-ingress	<none>	*		80	16s

Task 3 - Access Pods From a single external IP

1. Open the browser and enter the external IP of the Ingress controller service and append "/blue/" to the path: **http://"external ip"/blue/** (make sure to include the last /).

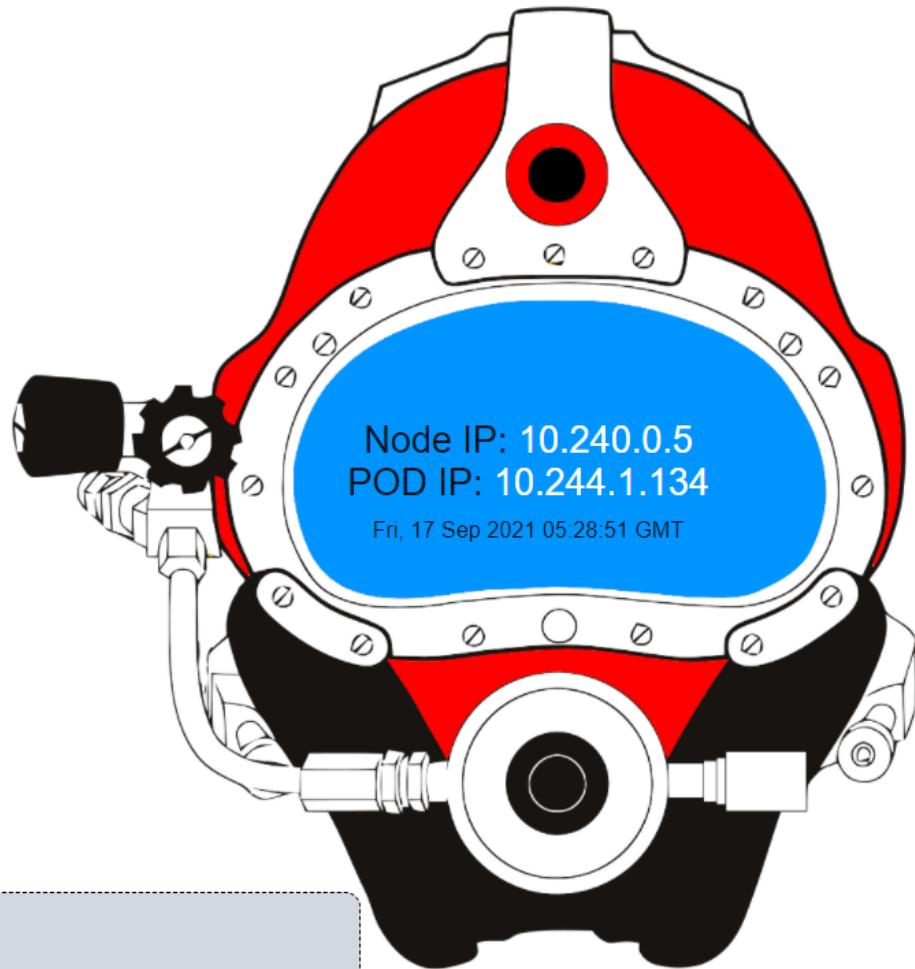
The screenshot shows a web browser window with the URL `20.72.145.246/blue/` highlighted in red. The main content is a blue submarine-themed interface. In the center, it displays the following information:

Node IP: 10.240.0.4
POD IP: 10.244.0.132
Fri, 17 Sep 2021 05:28:13 GMT

Below this, there is a section titled "Additional Info:" enclosed in a dashed box. It contains the following details:

Node Name:	aks-agentpool-35127589-vmss000000
POD Name:	blue-dep-89bcc4ffd-knwcx
Namespace:	dev

2. Open the browser and enter the external IP of the Ingress controller service and append "/red/" to the path: **http://"external ip"/red/** (make sure to include the last /)



Additional Info:

Node Name: aks-agentpool-35127589-vmss000001
POD Name: red-dep-fdbc49b9-sdvf8
Namespace: dev

Task 4 - Define a Default Backend

1. Open the browser and enter the external IP of the Ingress controller by itself: [http://external ip](http://20.72.145.246)

404 Not Found

nginx

This is the default NGinx Ingress Controller page.

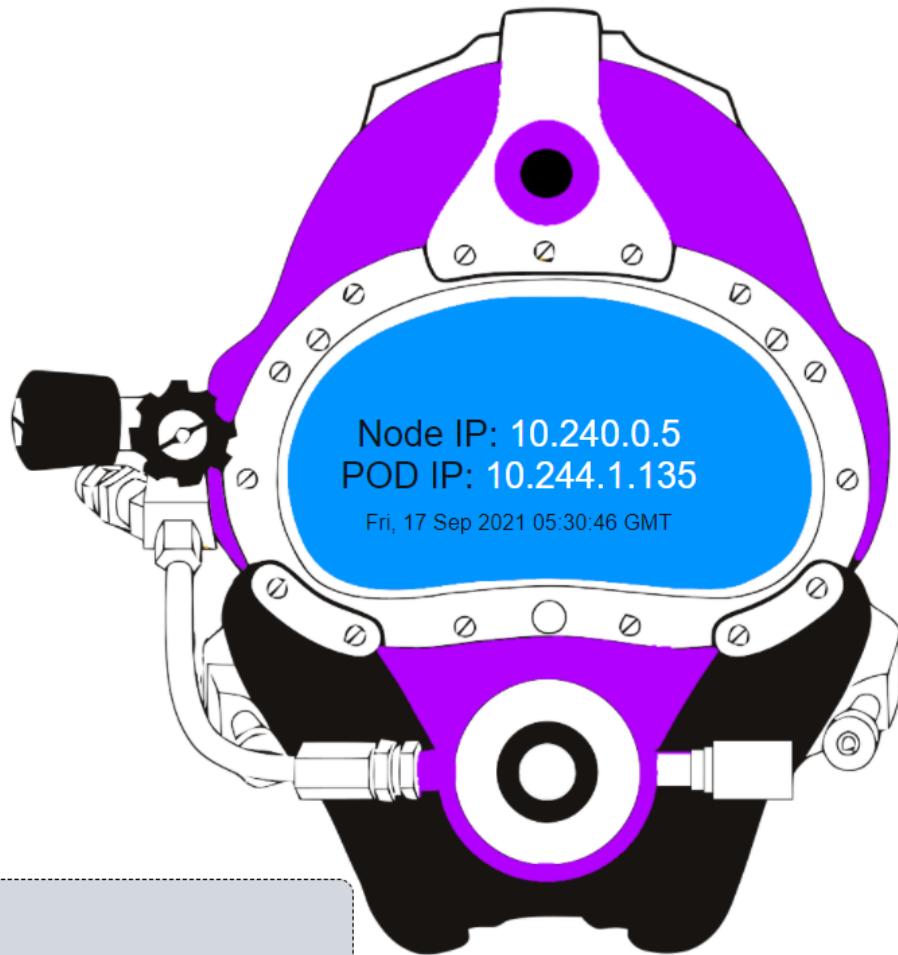
2. Review the contents of **C:\k8s\labs\Module3\default-backend.yaml** file in an editor. Notice the special property to specify the default backend. There's no *path* property needed.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: default-ingress-backend
spec:
  defaultBackend:
    service:
      name: default-svc
      port:
        number: 8100
```

3. Apply the *Default Backend* deployment, service and ingress resource to the *default* namespace (not *dev*).

```
kubectl apply -f default-dep.yaml -n default
kubectl apply -f default-svc.yaml -n default
kubectl apply -f default-backend.yaml -n default
```

4. Open the browser and enter the external IP of the Ingress controller by itself: **http://"external ip"**



Additional Info:

Node Name:	aks-agentpool-35127589-vmss00001
POD Name:	default-dep-75b5656d5f-xxrhc
Namespace:	default

[Module 3 Table of Contents](#)

[List of Modules](#)

Shutdown or Delete the Cluster

When you're done for the day, you can shutdown your cluster to avoid incurring charges when you're not using it. This will delete all your nodes, but keep your configuration in tact.

```
az aks stop --name $AKS_NAME  
          --resource-group $AKS_RESOURCE_GROUP
```

Or you can choose to delete the entire resource group instead. You can create a new one prior to the next lab.

```
az group delete --resource-group $AKS_RESOURCE_GROUP
```

[Module 3 Table of Contents](#)

[List of Modules](#)