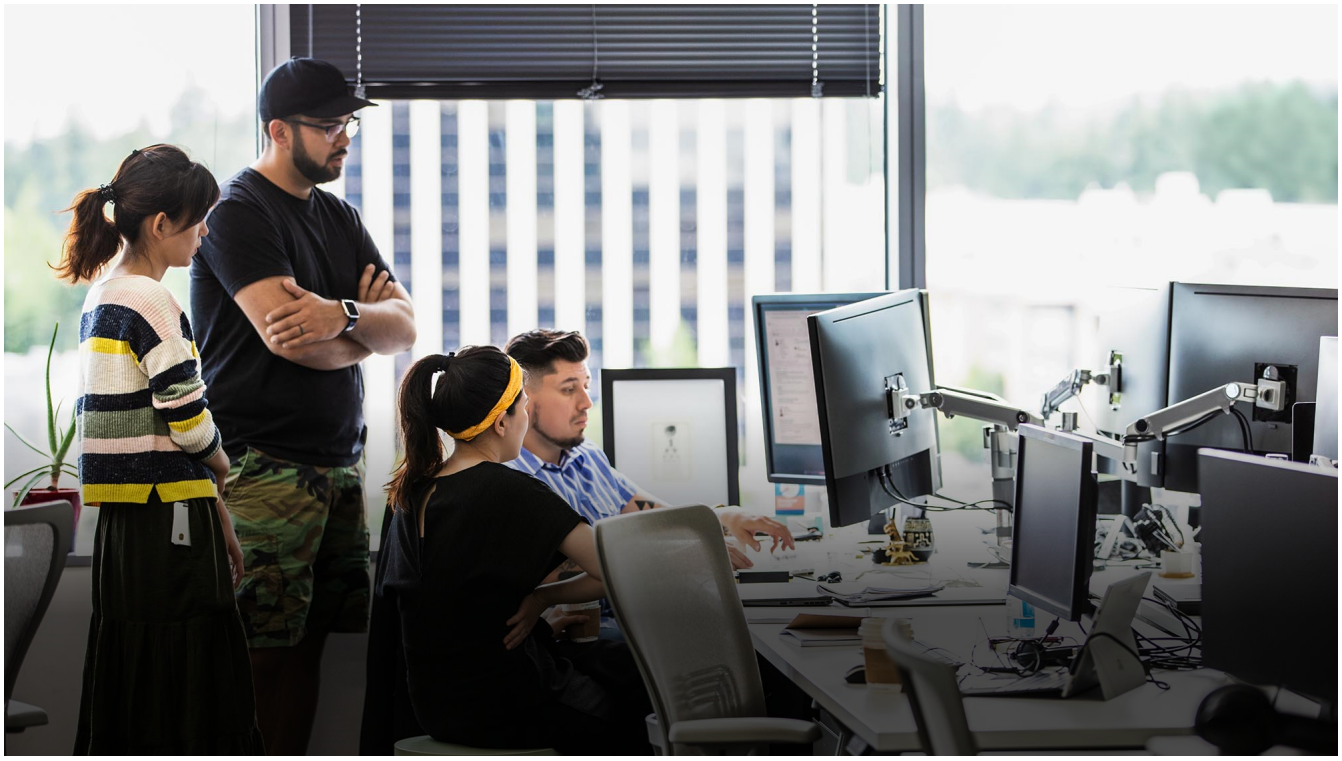
Lab Module 1: Core Kubernetes Concepts



> *Estimated Duration: 60 minutes*

Module 1 Table of Contents

Exercise: Create a Basic Azure Kubernetes Service (AKS) Cluster

In this exercise you will create a simple AKS cluster. In the next module, you'll create a more complete one.

Task 1 - Login into your subscription and register providers

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

> *Windows Terminal allows you to open tabbed command terminals.*

2. Login to Azure.

```
az login
```

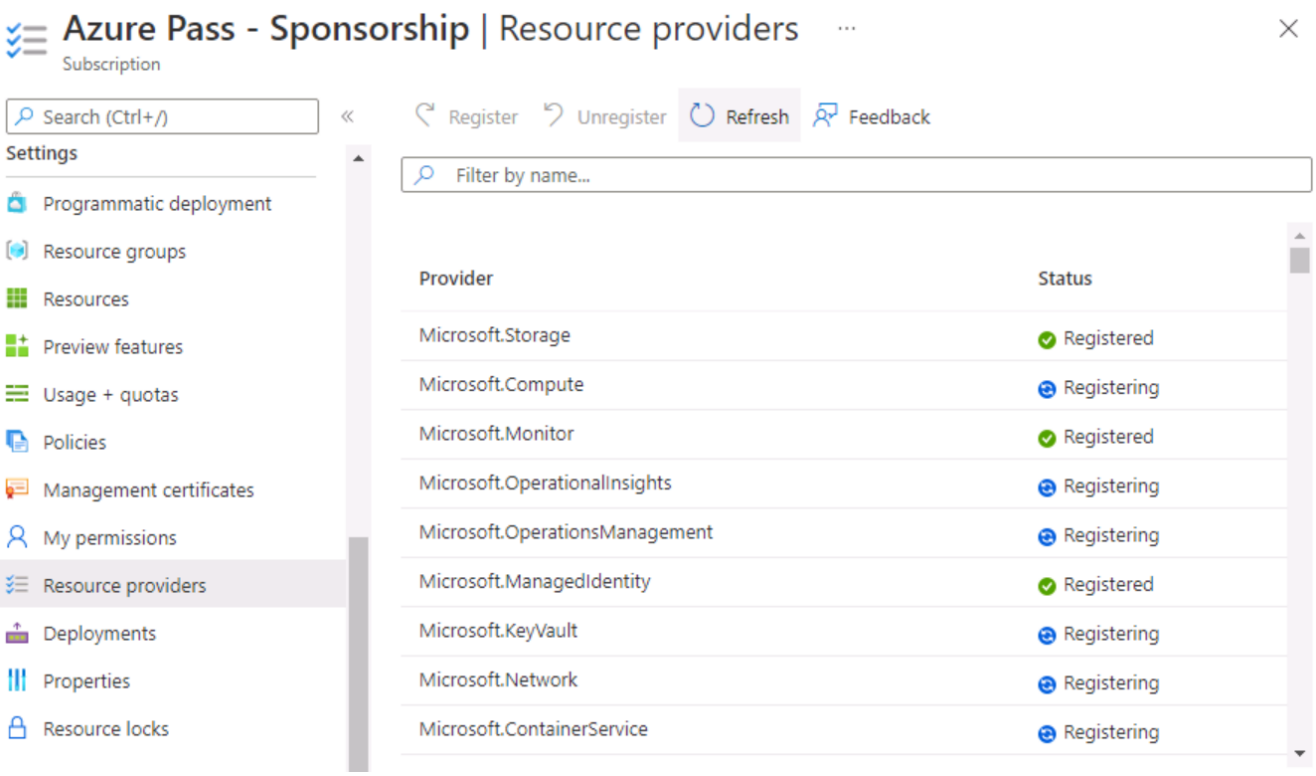
3. Set the current subscription.

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

4. Register needed providers.

```
az provider register --namespace Microsoft.Storage
az provider register --namespace Microsoft.Compute
az provider register --namespace Microsoft.Network
az provider register --namespace Microsoft.Monitor
az provider register --namespace Microsoft.ManagedIdentity
az provider register --namespace Microsoft.OperationalInsights
az provider register --namespace Microsoft.OperationsManagement
az provider register --namespace Microsoft.KeyVault
az provider register --namespace Microsoft.ContainerService
az provider register --namespace Microsoft.Kubernetes
```

5. Open a browser and navigate to the Azure Portal: ***portal.azure.com***
6. Search for and open the **Subscriptions** blade. Select your **Azure Pass - Sponsorship** subscription.
7. Scroll down and select **Resource providers**.



8. Watch the progress of the registration process until all the providers listed above have been registered. Click the *Refresh* button every few minutes to update the progess. Once everything has been registered, continue with the tasks in this lab.

Task 2 - Define variables and create resource group

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.

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

3. Get list of available VM sizes with 2 cores in your region.

```
az vm list-sizes --location $LOCATION `
                 --query "[?numberOfCores == ``2``].{Name:name}" -o table
```

4. Set the VM SKU to one of the available values or use the default below.

```
$VM_SKU="Standard_D2as_v5"
```

5. Create Resource Group.

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

Task 3 - Create a basic cluster using Azure CLI

1. Define variables for AKS cluster.

```
$AKS_NAME="aks-$($INSTANCE_ID)"
Write-Host "AKS Cluster Name: $AKS_NAME"
```

2. Create a simple AKS cluster.

```
az aks create --node-count 2 `
              --generate-ssh-keys `
              --node-vm-size $VM_SKU `
              --name $AKS_NAME  `
              --resource-group $AKS_RESOURCE_GROUP
```

> *The creation process will take able 5-10 minutes.*

3. Once complete, connect the cluster to your local client machine.

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

4. Confirm the connection to the cluster.

```
kubectl get nodes
```

This should return a list of nodes similar to the one below:

```
NAME                                STATUS   ROLES   AGE     VERSION
aks-nodepool1-20466695-vmss000000   Ready    agent   3m17s   v1.19.11
aks-nodepool1-20466695-vmss000001   Ready    agent   3m18s   v1.19.11
```

Module 1 Table of Contents

List of Modules

Exercise: Creating a Pod Declaratively

This Exercise demonstrates the use of a YAML file to create a pod declaratively.

Task 1 - Create a Pod declaratively

1. Change into the **Module1** folder

```
cd C:\k8s\labs\Module1
```

2. Use the YAML file provided to create a Pod. You may want to open the **simple-pod.yaml** file and review its contents.

The pod definition contains the **Nginx** container that listens to port 80.

```
kubectl apply -f simple-pod.yaml
```

3. Now, make sure pod is up and running.

```
kubectl get pods
```

You should see a pod named **nginx-pod**

```
NAME        READY   STATUS    RESTARTS   AGE
nginx-pod   1/1     Running   0          10s
```

4. Add a second pod, then check the list again.

```
kubectl apply -f simple-pod2.yaml
kubectl get pods
```

Task 2 - Filter pods based on a label

1. Show all the labels in the pods

```
kubectl get pods --show-labels
```

```
NAME        READY   STATUS    RESTARTS   AGE     LABELS
nginx-pod   1/1     Running   0          6m49s   kind=web,target=dev
```

2. Let's say you want to list pods that have a label named **kind=web** associated with them. You can use **-l** switch to apply filter based on labels.

```
kubectl get pod -l kind=web
```

3. To prove that this works as expected, run the command again but change the value of label **kind** to **db**. Notice, this time *kubectl* doesn't return any pods because there are no pods that match the label **kind** and a value of **db**.

```
kubectl get pod -l kind=db
```

Task 3 - View complete definition of the Pod

1. Query Kubernetes to return the complete definition of a Pod from its internal database by exporting the output (**-o**) to **YAML**. Then pipe the result to a file.

```
kubectl get pods nginx-pod -o yaml > mypod.yaml
```

> *To view the JSON version, use the **-o json** flag instead.*

2. View the contents of the generated file in VS Code (or an editor of your choice).

```
code mypod.yaml
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","kind":"Pod","metadata":{"annotations
  creationTimestamp: "2021-07-28T21:17:03Z"
  labels:
    kind: web
    target: dev
  name: nginx-pod
  namespace: default
  resourceVersion: "5044"
  selfLink: /api/v1/namespaces/default/pods/nginx-pod
  uid: d2dcfd0d-9968-410f-87a8-6033024834d8
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: nginx
```

NOTE: Observe all the properties that Kubernetes populated with default values when it saved the Pod definition to its database.

Module 1 Table of Contents

List of Modules

Exercise: Adding/Updating/Deleting Labels on a Pod

In this Exercise, you will create a pod that has labels associated with it. Labels make it easy to filter the pods later. Labels play a vital role in the Kubernetes ecosystem, so it's important to understand their proper usage.

Task 1 - Assign a new label to a running Pod

1. Assign a new label (key=value) pair to a running pod. This comes in handy when you are troubleshooting an issue and would like to distinguish between different pod(s). Assign a new label **health=fair** to the pod **nginx-pod**, which is already running.

```
kubectl label pod nginx-pod health=fair
```

2. Run the command below to show the pod labels. Notice that now an additional label is shown with the pod.

```
kubectl get pods nginx-pod --show-labels
```

Task 2 - Update an existing label that is assigned to a running pod

1. Update the value of an existing label that is assigned to a running pod. Change the value of the label **kind=web** to **kind=db** of the **nginx-pod** pod.

```
kubectl label pod nginx-pod kind=db --overwrite
```

--overwrite is needed because the pod is running and won't accept changes otherwise.

2. Show the pod labels again. Notice that *kind* has changed from **web** to **db**.

```
kubectl get pods --show-labels
```

Task 3 - Delete a label that is assigned to a running Pod

1. Delete the label **health** from the **nginx-pod** pod.

```
kubectl label pod nginx-pod health-
```

NOTE: Notice the minus (**-**) sign at the end of the command. You can also remove a label from all running pods by using the **--all** flag.

```
kubectl label pod health- --all
```

2. Run the command below to show the pod labels again. Notice that *health* is not part of the list of labels.

```
kubectl get pods --show-labels
```

Task 4 - Delete Pods based on their labels

1. Delete all the Pods that match a specific label.

```
kubectl delete pod -l target=dev
```

Module 1 Table of Contents
List of Modules

Exercise: Working with Deployments

In this Exercise, you will create a Deployment and rollout an application update. Deployments provide a consistent mechanism to upgrade an application to a new version, while keeping the downtime to a minimum. Note that internally, Deployments use *ReplicaSets* for managing Pods. However, you never work directly with *ReplicaSets* since Deployments abstract out that interaction.

Task 1 - Create a new Deployment

The **ng-dep.yaml** file contains a Deployment manifest. The Pod in the *template* contains an *nginx* container with a tag **1.0**. The **1.0** represents the version of this container and hence of the application running inside it.

1. Create a Deployment and a Service to access the Pods of the deployment.

```
kubectl apply -f ng-dep.yaml
kubectl apply -f ng-svc.yaml
```

NOTE: The *--record* flag saves the command you applied in the deployment's ReplicaSet history. This helps in deciding which previous Revision to roll back to if needed.

2. Run the following command to see the Pods, ReplicaSets, Deployments and Services that were created.

```
kubectl get all --show-labels
```

```
NAME                              READY   STATUS    RESTARTS   AGE   LABELS
pod/ng-dep-596c57c89b-b5xp6       1/1     Running   0          98s   pod-template-hash=596c57c89b,target=dev
pod/ng-dep-596c57c89b-wklxl       1/1     Running   0          98s   pod-template-hash=596c57c89b,target=dev

NAME                 TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)   AGE   LABELS
service/kubernetes   ClusterIP   10.0.0.1      <none>        443/TCP   94m   component=apiserver,provider=kube
rnetes

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE   LABELS
deployment.apps/ng-dep       2/2     2            2           98s   <none>

NAME                                    DESIRED   CURRENT   READY   AGE   LABELS
replicaset.apps/ng-dep-596c57c89b       2         2         2       98s   pod-template-hash=596c57c89b,target=dev
```

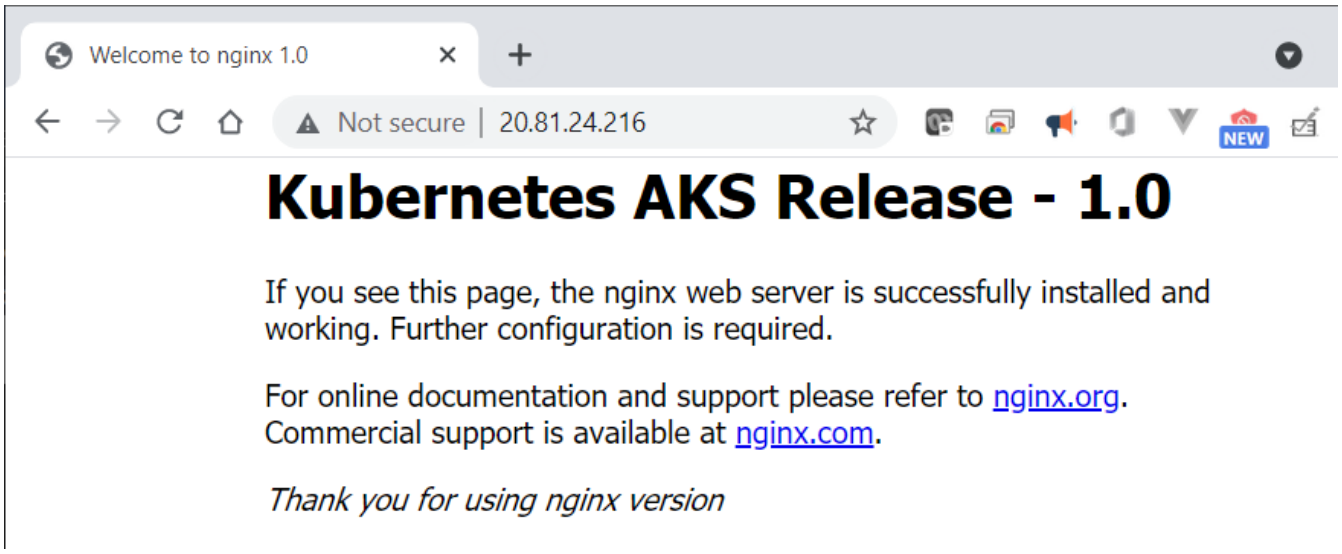
Task 2 - Access version 1.0 of application

1. Wait about 3-4 minutes to allow Azure to create a Public IP address for the service. Check to see if an address has been assigned by getting the list of services.

```
kubectl get svc
```

```
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)        AGE
kubernetes   ClusterIP      10.0.0.1        <none>         443/TCP        6d8h
ng-svc       LoadBalancer   10.0.251.199    20.81.24.216   80:31549/TCP   44s
```

2. When you see an **EXTERNAL-IP** assigned, open a browser with that address. Example: **http://20.81.24.216**



Task 3 - Update the Deployment to version 2.0

You are now going to update the Deployment to use version **2.0** of the container instead of **1.0**. This can be done in one of two ways. One approach is to use *imperative* syntax, which is faster and is often used during the development/testing stage of an application. The alternate method is to update the YAML file and then to reapply it to the cluster.

1. To start rolling out the new update, change the container image tag from **1.0** to **2.0** by running this command:

```
kubectl set image deployment ng-dep nginx=k8slab/nginx:2.0
```

2. In the command above, **ng-dep** is the name of Deployment and **nginx** is the name of the container within the Pod template. The change will force the Deployment to create a new ReplicaSet with an image tagged **2.0**.
3. List all the pods and notice that old pods are terminating and that new Pods have been created.

```
kubectl get pods
```

4. Run the follwing command to review the Deployment definition with the updated value of container image:

```
kubectl describe deployment ng-dep
```

```
Name:                   ng-dep
Namespace:              default
CreationTimestamp:      Wed, 28 Jul 2021 16:18:42 -0600
Labels:                 <none>
Annotations:            deployment.kubernetes.io/revision: 2
                        kubectl.kubernetes.io/last-applied-configuration:
                          {"apiVersion":"apps/v1","kind":"Deployment","metadata":{"anno
"namespace":"default"},"spec":{"replicas":2,"sele...
Selector:               target=dev
Replicas:               2 desired | 2 updated | 2 total | 2 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  target=dev
  Containers:
   nginx:
    Image:        k8slab/nginx:2.0
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
```

> *Notice the Image section (under Containers) shows the value of container image as **2.0**.*

5. Run the command to view the Pods, ReplicaSets and Deployments again.

```
kubectl get all
```

```
NAME                            READY   STATUS    RESTARTS   AGE
pod/ng-dep-6c58bf56cc-9q4fk     1/1     Running   0          5m47s
pod/ng-dep-6c58bf56cc-kpb4n     1/1     Running   0          5m51s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.0.0.1     <none>        443/TCP   107m

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/ng-dep     2/2     2            2           15m

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/ng-dep-596c57c89b     0         0         0       15m
replicaset.apps/ng-dep-6c58bf56cc     2         2         2       5m51s
```

> *Notice that the old replica set still exists, even though it has 0 Desired Pods.*

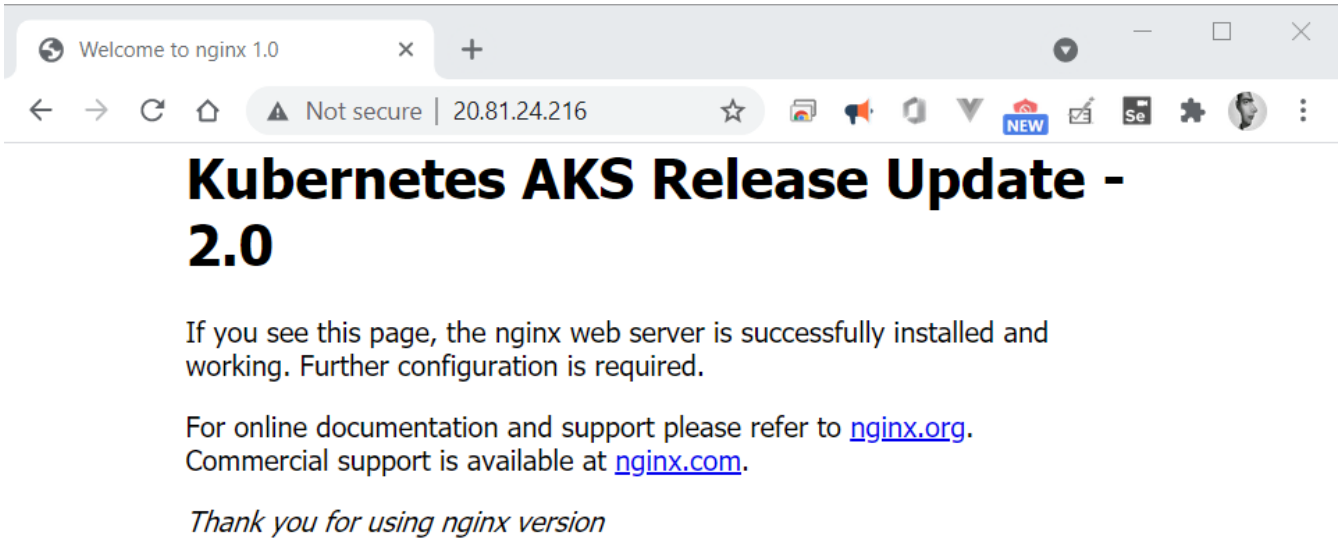
6. Run the *describe* command on that old ReplicaSet.

```
kubectl describe rs <old replicaset name>
```

```
Name:           ng-dep-596c57c89b
Namespace:      default
Selector:       pod-template-hash=596c57c89b,target=dev
Labels:         pod-template-hash=596c57c89b
                target=dev
Annotations:    deployment.kubernetes.io/desired-replicas: 2
                deployment.kubernetes.io/max-replicas: 3
                deployment.kubernetes.io/revision: 1
Controlled By:  Deployment/ng-dep
Replicas:       0 current / 0 desired
Pods Status:    0 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  pod-template-hash=596c57c89b
           target=dev
  Containers:
   nginx:
    Image:        k8slab/nginx:1.0
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  SuccessfulCreate   18m    replicaset-controller  Created po
```

> *Notice that the old definition still has the previous version number. This is maintained so you can roll back the change to that version if you which.*

7. Access the 2.0 version of application by refreshing the browser at the same address as above.



Task 4 - Rollback the Deployment

The purpose of maintaining the previous **ReplicaSet** is to be able to rollback changes to any previous version.

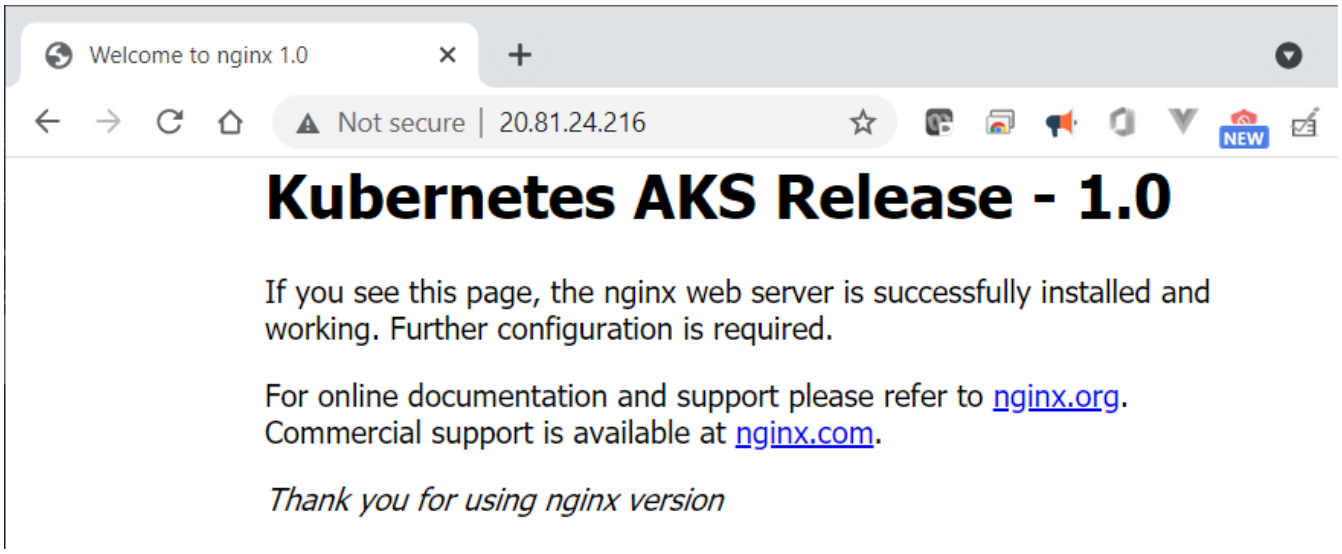
1. Review the deployment history.

```
kubectl rollout history deploy/ng-dep
```

2. Rollback the Deployment to the previous version.

```
kubectl rollout undo deploy/ng-dep
```

3. Wait a few seconds and refresh the browser again.



> *Notice the site is back to the previous version.*

Task 5 - Delete the Deployment and Service

1. Delete the Deployment and Service.

```
kubectl delete deployment ng-dep
kubectl delete service ng-svc
```

NOTE: It may take a few minutes to delete the service because has to delete the Public IP resource in Azure.

Module 1 Table of Contents

List of Modules

Exercise: Working with Services

In this Exercise you will create a simple Service. Services help you expose Pods externally using label selectors.

Task 1 - Create a new Service

1. Create a deployment.

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

2. The **sample-svc.yaml** file contains a Service manifest. Services use label selectors to determine which Pods it needs to track and forward the traffic to.
3. Review running Pods and their labels.

```
kubectl get pods --show-labels
```

> *Notice the label **sample=color** that is associated with the Pods.*

2. Open the **sample-svc.yaml** file and examine the **selector** attribute. Notice the **sample: color** selector. This Service will track all Pods that have a label **sample=color** and load balance traffic between them.
3. Create the Service.

```
kubectl apply -f sample-svc.yaml
```

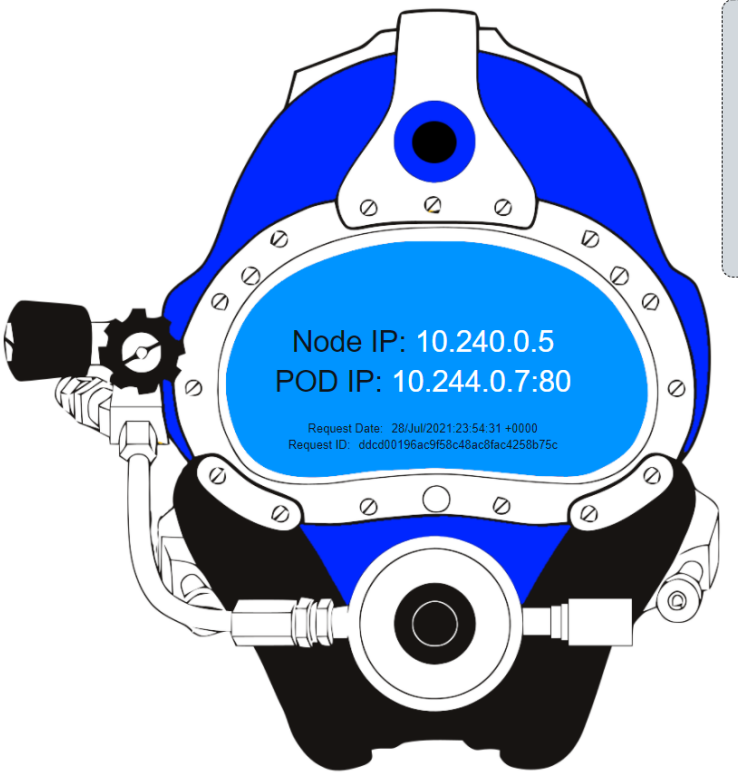
4. Check the of newly created service.

```
kubectl get svc -o wide
```

The command above will display the details of all available services along with their label selectors. You should see the **sample-svc** Service with **PORTS 80:30101/TCP** and **SELECTOR sample=color**.

Task 2 - Access the sample-svc Service

1. Open a browser and navigate to the IP address shown in the output of the previous command.



Additional Info:

Node Name:	**aks-nodepool1-20466695-vmss000001**
POD Name:	**sample-dep-5957b459c-7gx2c**
Namespace:	**default**
Service Account:	**default**
CPU Request:	**0**
CPU Limit:	**4**
MEM Request:	**0**
MEM Limit:	**13213429760**

References:

Kubernetes Downward API

NGINX Demo Environment

2. The website displays the Node IP/Pod IP address of the pod currently receiving the traffic through the service's load balancer. The page refreshes every 3 seconds and each request may be directed to a different pod, with a different IP address. This is the service's internal load balancer at work.

Task 3 - Delete the Deployment and Service

Deleting any Pod will simply tell Kubernetes that the Deployment is not in its *desired* state and it will create a replacement. You can only delete Pods by deleting the Deployment.

1. Delete the Deployment.

```
kubectl delete deployment sample-dep
```

2. The Service is independent of the Pods it services, so it's not affected when the Deployment is deleted. Anyone trying to access the service's address will simply get a 404 error. If the Deployment is ever re-created, the Service will automatically start sending traffic to the new Pods.
3. Delete the Service.

```
kubectl delete service sample-svc
```

Module 1 Table of Contents

List of Modules

Exercise: Cleanup

Task 1 - Delete the cluster

When you're done working with the cluster, you can delete it if you wish. This will ensure you don't incur any costs on your sponsorship subscription when you're not working on the labs.

1. Deleting the cluster is as easy as creating it.

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

List of Modules