App Modernization Labs
Azure Kubernetes Service

Author: Francis S Nazareth

Cloud Solutions Architect, Azure App Dev, Microsoft Qatar

fnazaret@microsoft.com / +974 33134296

Contents

Pre-Requisites

1. Azure Account with contributor access to the Azure subscription or Resource Group where AKS, Virtual Network, and other resources will be created.
2. GitHub account

1. Create Resources

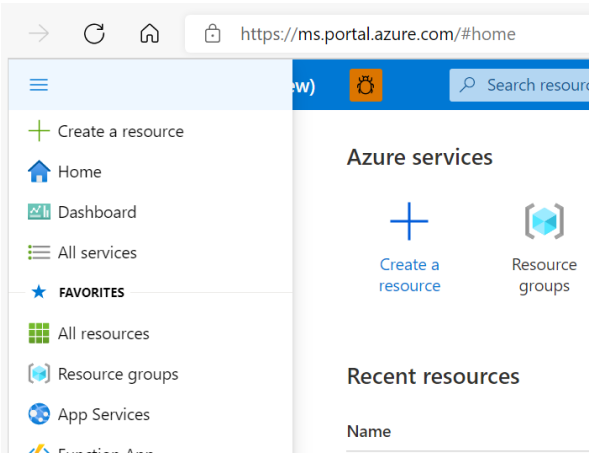
In this lab, you will create a resource group in Azure, virtual network and subnets in Azure, Azure Kubernetes Service and an instance of Azure Container Registry.

Steps:

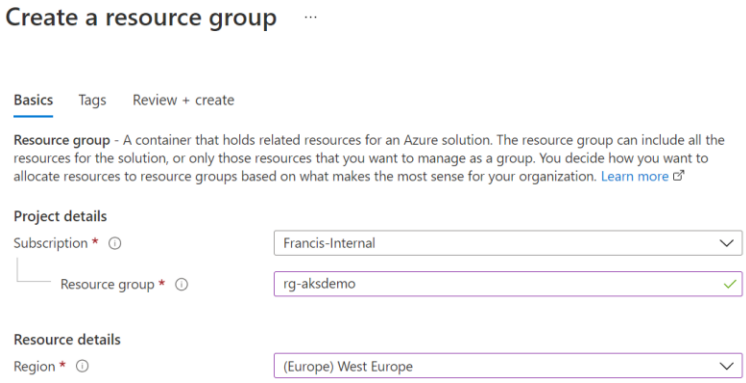
Sign up / Log in to Azure Portal.
http://portal.azure.com

1.1 Create a resource group

From the menu on top left, select "Resource Groups". Click "+Create".



Specify a name for the resource group, and specify a location (for example, West Europe).

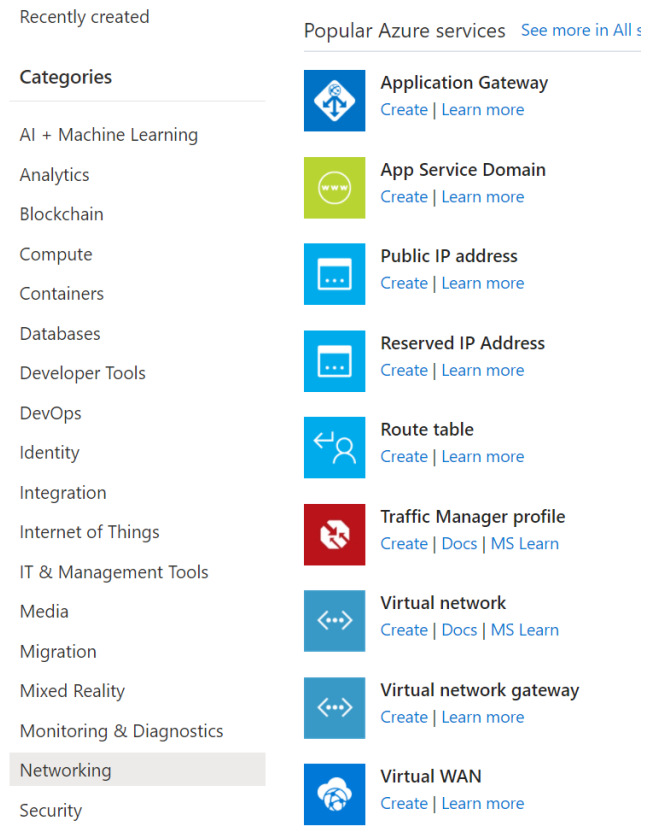


Click "Review & Create", followed by "Create". The resource group should be created after this step.

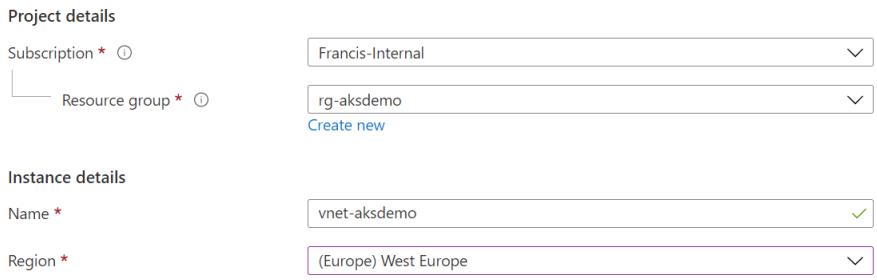
1.2 Create a virtual network and subnets

Go to the resource group (click on the resource group name)
Click on +Create

From the menu, select "Networking"
Under "Virtual Networks", click "Create".



Provide a name for the resource (For example, vnet-aksdemo). Choose the region as same as the resource group's region. And Click "Next: IP Addresses>".

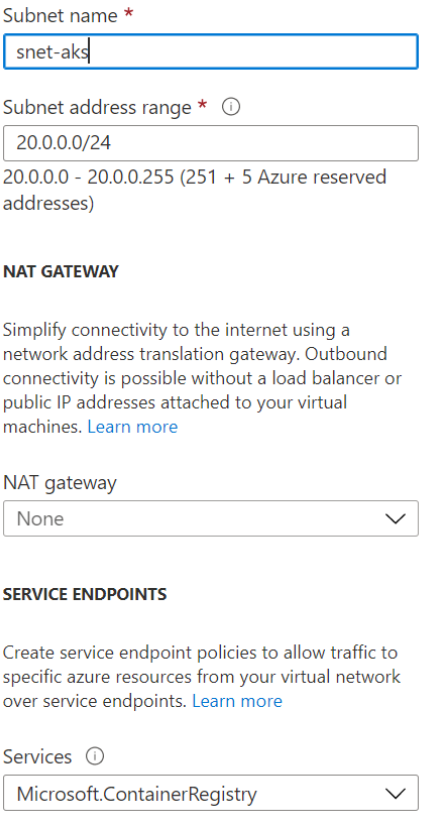


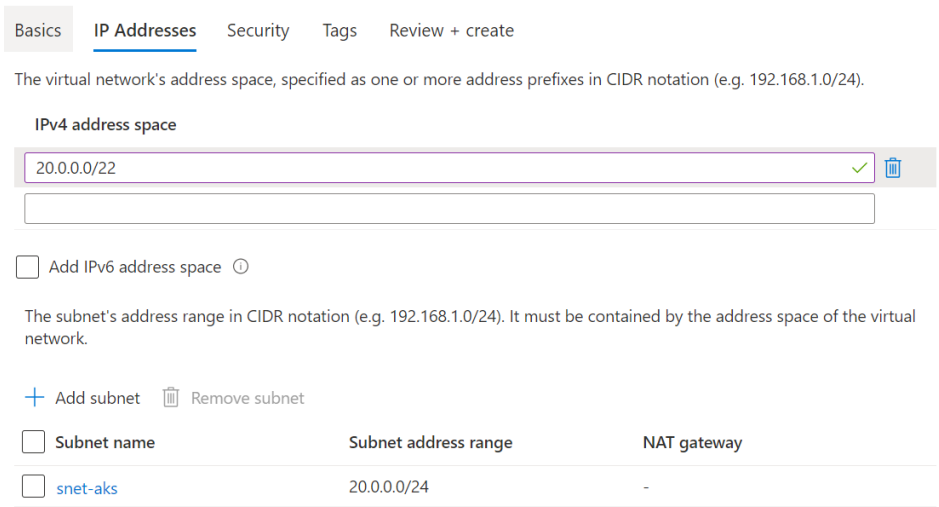
In the IP addresses tab, delete the default IPV4 Address Space.
Add an IPV4 address space: 20.0.0.0/22

click + Add Subnet

- Specify a name for the subnet: snet-aks
- Specify the IP address range: 20.0.0.0/24
- In the service end points, select "Microsoft.ContainerRegistry"

Click "Add" to add the subnet



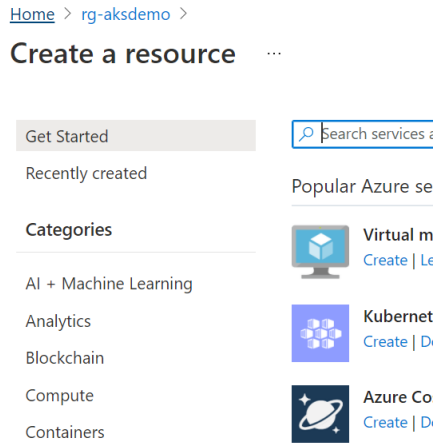


Click “Review & Create”, and “Create” to create the virtual network with subnets.

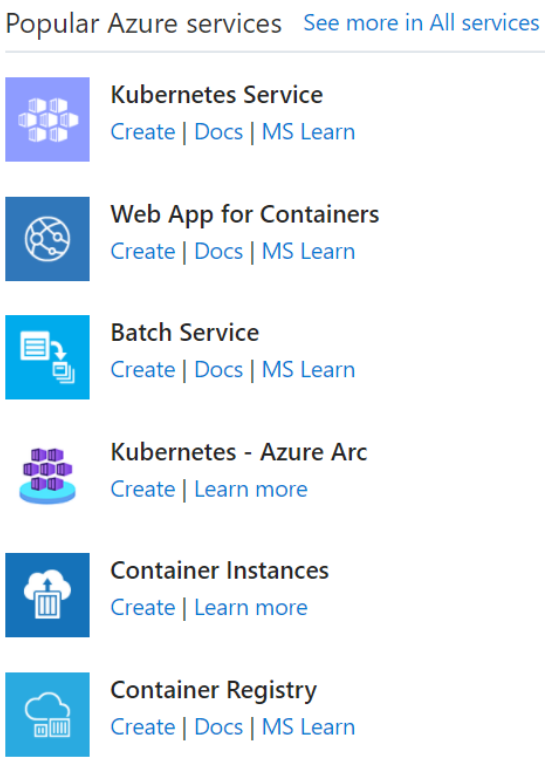
1.3 Create Azure Container Registry

Go to the resource group (click on the resource group name)

Click on +Create
From the menu, select "Containers"



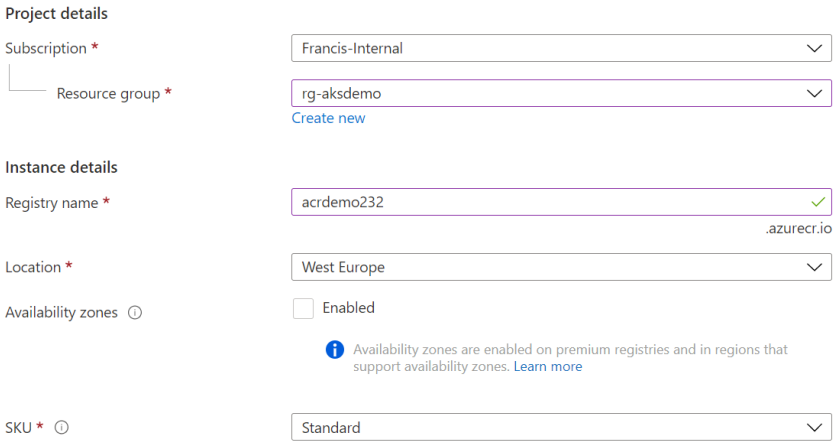
In the list of items, under "Container Registry", click on Create



Specify a unique name for the container registry (for example, acrdemo followed by few random numbers).

Select the region (select the same region of resource group).

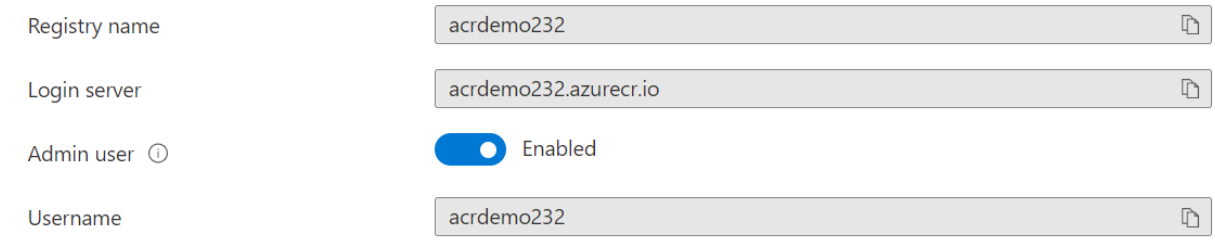
Leave everything else to defaults, and click "Review & Create" followed by "Create".



Enable admin access to the container registry:

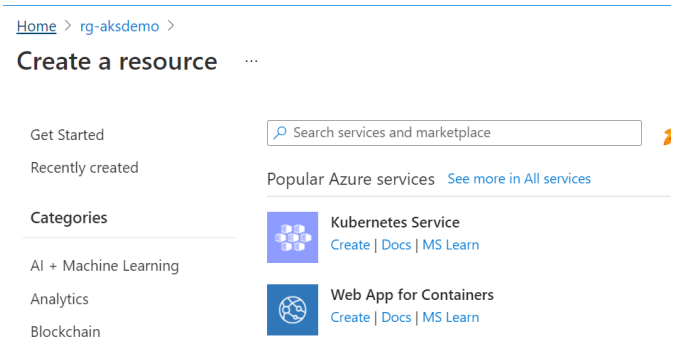
Open the container registry that got created, and in the left menu, select Settings -> Access Keys.

Toggle "Admin User" to Enabled.



1.4 Create Azure Kubernetes Service instance.

Go to the resource group (click on the resource group name)
Click on +Create
From the menu, select "Containers"
Under Kubernetes Service, click "Create"



Provide a name for the cluster (for example, aks-demo)

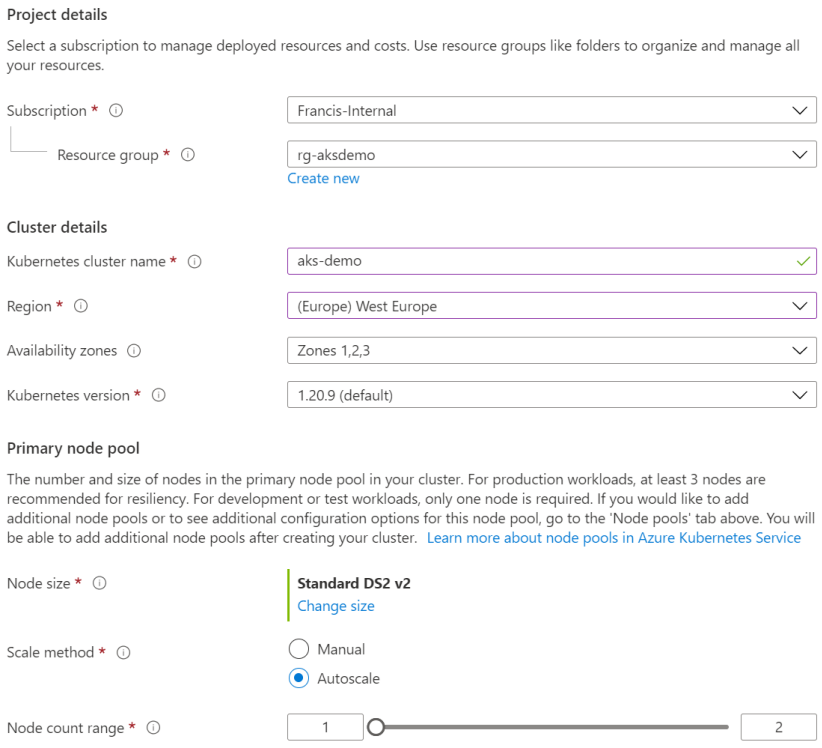
Select the region (same as the resource group's region)

Leave the defaults for availability zones

Leave the defaults for Kubernetes version.

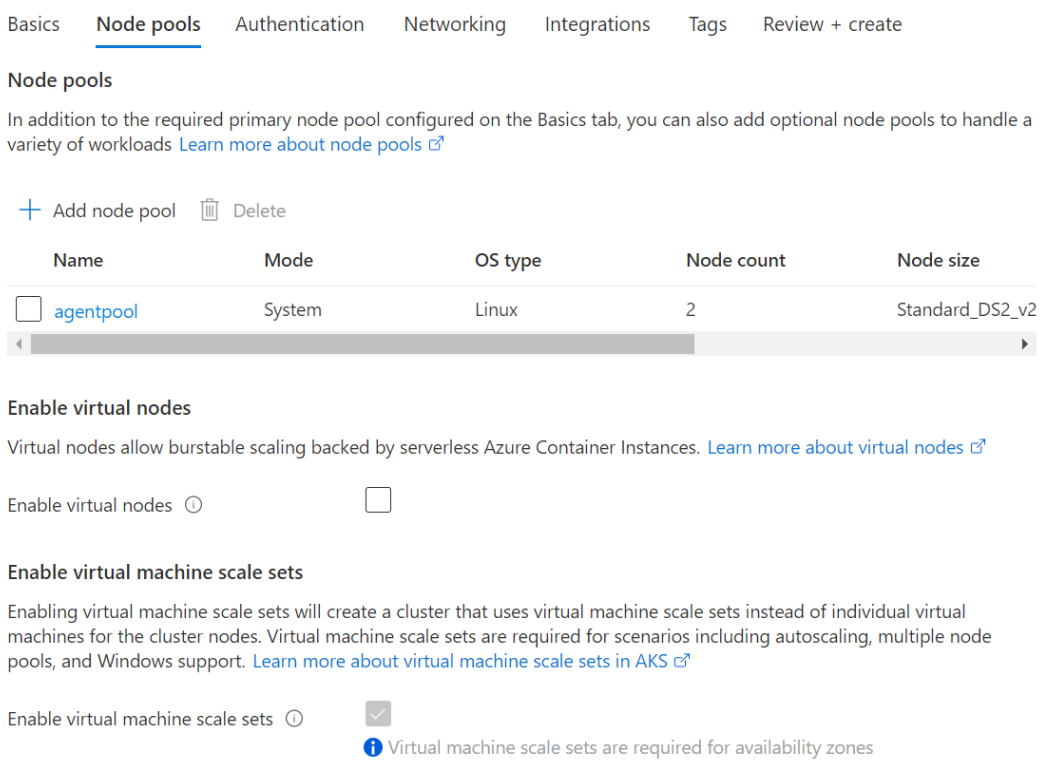
Leave the defaults for default node size.

For scale method, select "Auto Scale". Select minimum as 1 and maximum as 2.



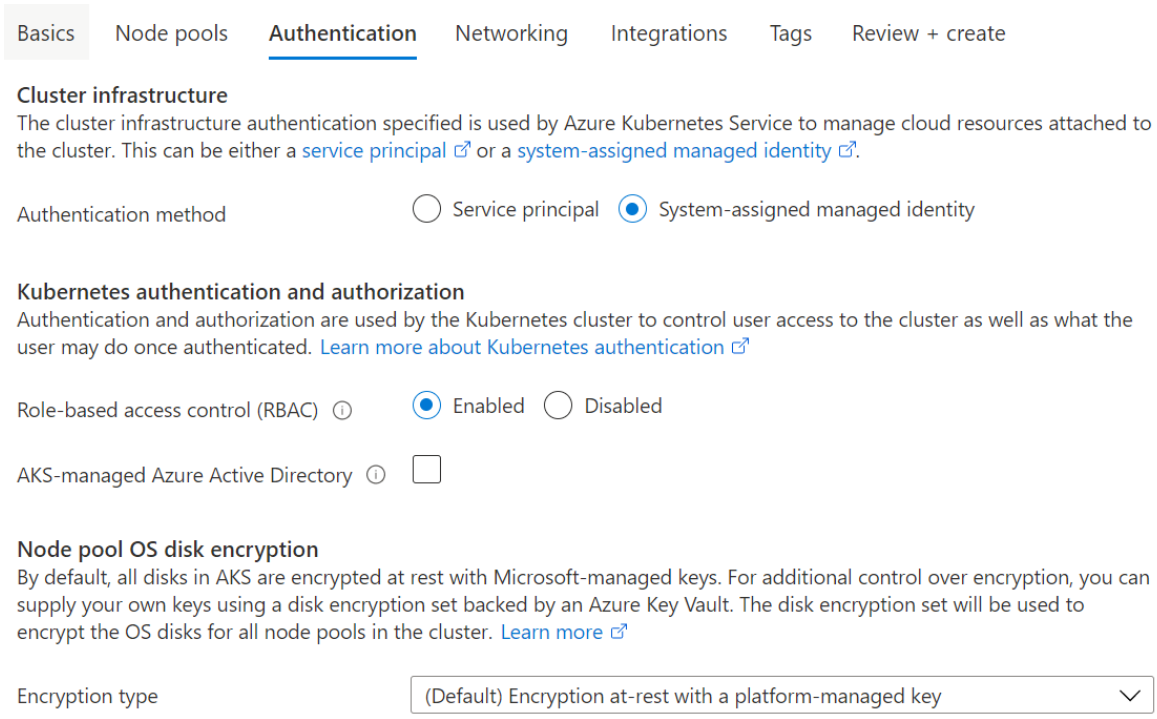
Click "Next > Node Pools"

In the Node Pools screen, leave the defaults for node pools.



Click “Next > Authentication”

Leave the defaults for authentication.



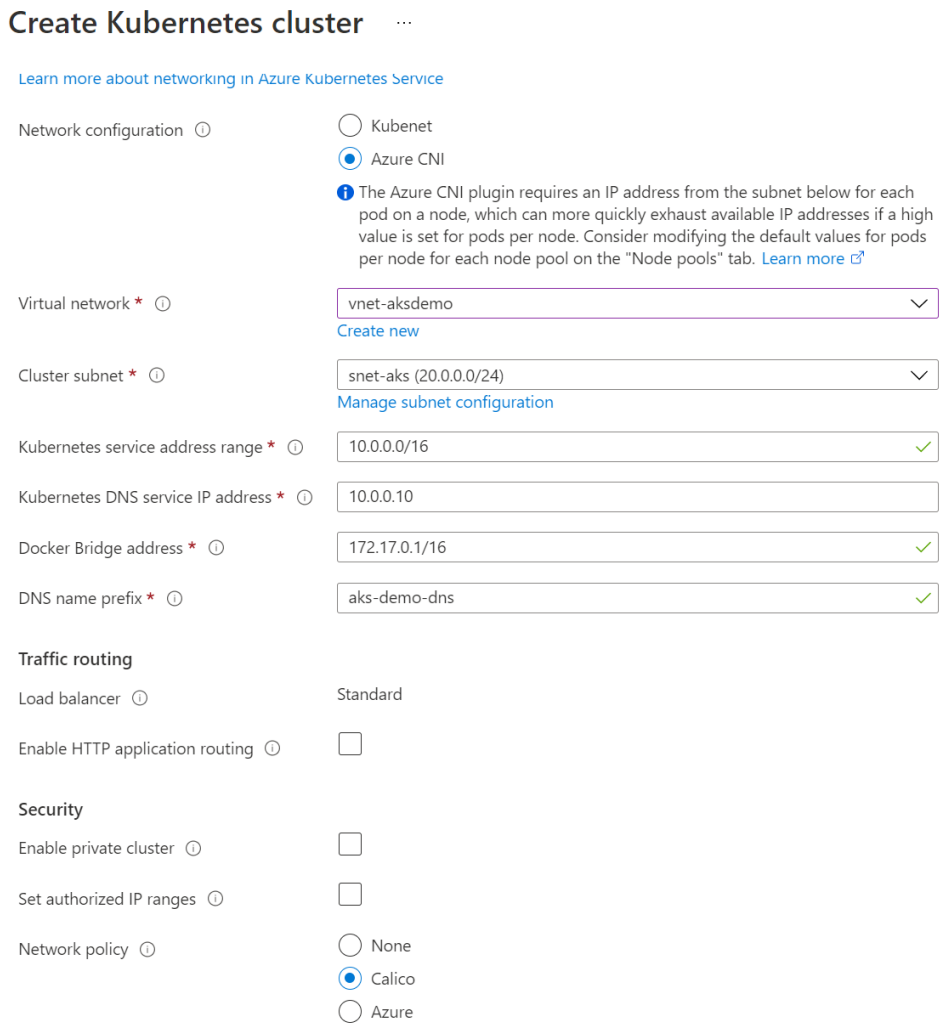
Click “Next: Networking>”

Change the network plugin from kubenet to Azure CNI.

Select the VNET you created earlier.

Select the Subnet you created earlier.

For network policies, select “Calico”.

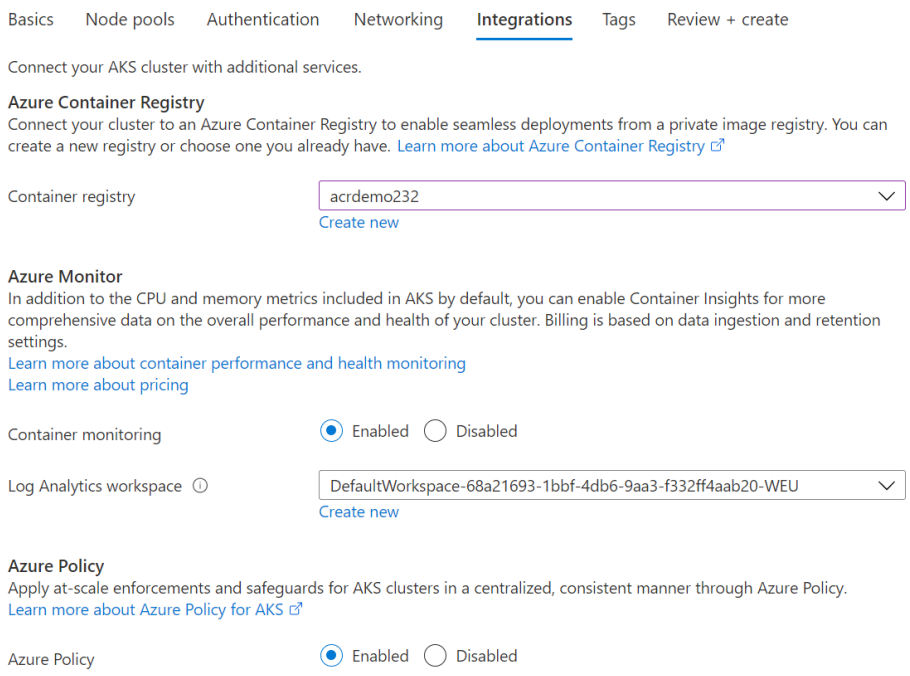


Select “Next: Integrations>”

Select the container registry you created earlier.

Leave container monitoring as enabled.

Select Azure Policy to Enabled.



Click “Review & Create”, and “Create”.

2. Create a container application and deploy to AKS

In the following lab, we are going to create a HTML based container application (with NGINX container image as the base layer).

a. On the top blue ribbon, click on "Cloud Shell" button.

Select "Bash" as the environment.

b. Create an HTML file index.html by using the following command:

code index.html

- Copy the following contents to the file

```
<html>
  <body>
    <div align="center">
      <font size="20px">Welcome!</font>
    </div>
  </body>
</html>
```

- Save the file (CTRL + S)
- Exit out of code (CTRL + Q)

c. Create Dockerfile using the following command:

code Dockerfile

- Copy the following contents to Dockerfile

```
FROM nginx

COPY index.html /usr/share/nginx/html
```

- Save the file (CTRL + S)
- Exit out of code (CTRL + Q)

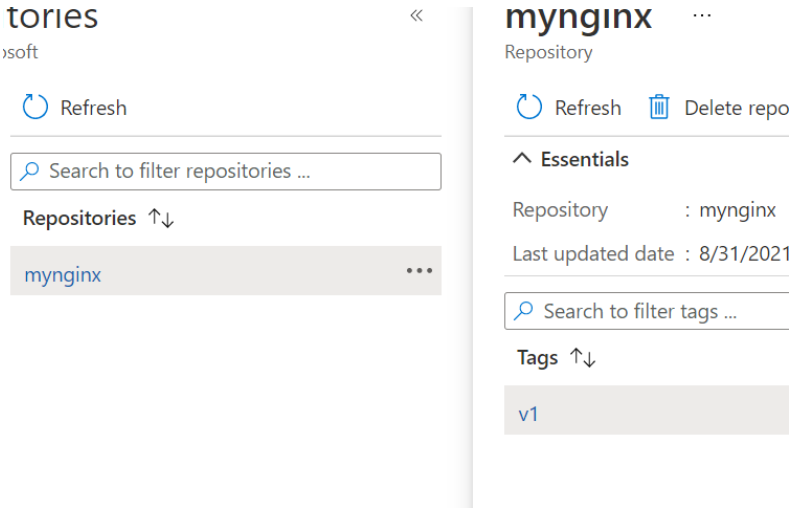
d. Build the docker image and push the docker image to the container registry using the following command

az acr build --registry <<Azure Container Registry Name>> --image mynginx:v1 .

For example,

az acr build --registry acrdemo232.azurecr.io --image mynginx:v1 .

e. In the container registry, select "Services -> Repositories". View the container image that was uploaded.

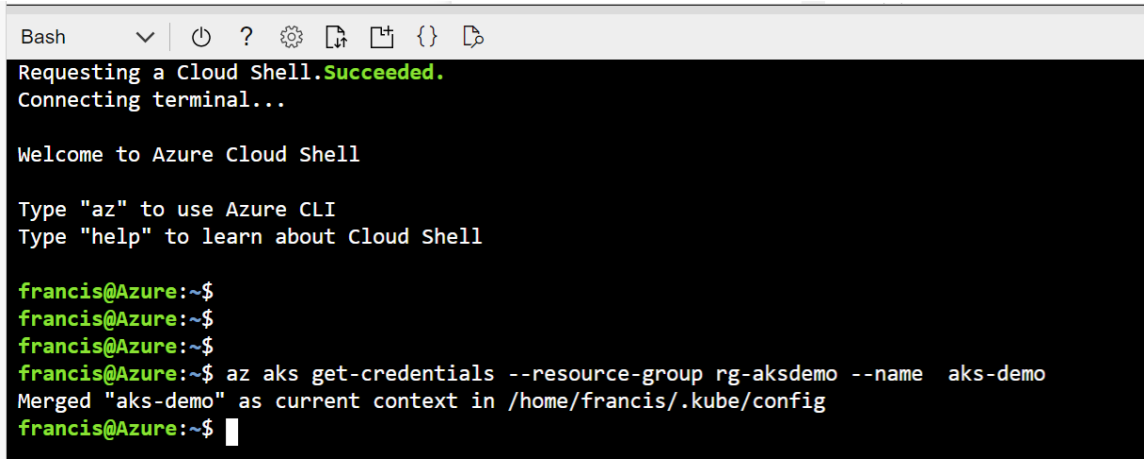


f. In the cloud shell, run the following command (to merge Kubernetes context).

az aks get-credentials --resource-group <<resource group name>> --name <<AKS cluster name>>

For example,

az aks get-credentials --resource-group rg-aksdemo --name aks-demo



g. Check the Kubernetes cluster status

kubectl get nodes -o wide

h. Create a deployment using the container image we uploaded

kubectl create deploy hello-deploy --image <repository name>/<image>:<tag> --replicas=3

For example,

kubectl create deploy hello-deploy --image acrdemo232.azurecr.io/mynginx:v1 --replicas=3

i. Check the pods deployed

kubectl get pods

You should see 3 pods in the running state.

```
francis@Azure:~$ kubectl create deploy hello-deploy --image acrdemo232.azurecr.io/mynginx:v1 --replicas=3
deployment.apps/hello-deploy created
francis@Azure:~$ kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
hello-deploy-f6bff5f76-2d9kv   1/1     Running   0          82s
hello-deploy-f6bff5f76-sd5hp   1/1     Running   0          82s
hello-deploy-f6bff5f76-sxm46   1/1     Running   0          82s
```

j. Expose the deployment as a Kubernetes service (a load balancer with a public IP, which distributes the load to the three pods).

kubectl expose deploy hello-deploy --port=80 --target-port=80 --type=LoadBalancer

k. Check the IP address of the service that got created, using the command:

kubectl get services

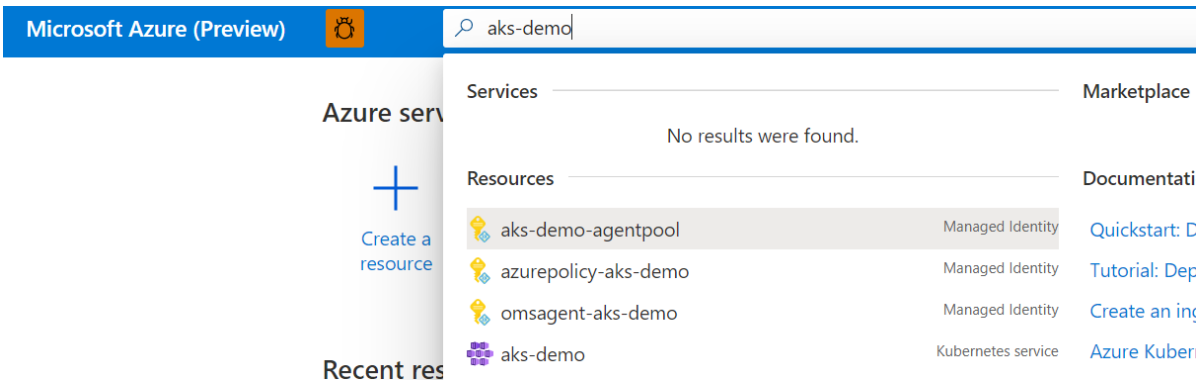
```
^Cfrancis@Azure:~$ kubectl get services
NAME           TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)        AGE
hello-deploy   LoadBalancer   10.0.124.128   51.144.187.93   80:32331/TCP   47s
kubernetes     ClusterIP      10.0.0.1       <none>          443/TCP        16h
```

l. Access the public IP address in a browser

You should see the welcome page rendered in the browser.

3. Monitor the cluster using Azure Monitor.

a. Search for the cluster name in the Azure portal's search bar.



b. Click on the Kubernetes service name to go to the cluster view.

c. In the left menu, select Kubernetes resources -> Namespaces

d. Select (click on) the default namespace, and select "View Events".

e. Go back to the cluster view, and select Kubernetes resources -> Workloads

f. Click on the deployment we created (hello-deploy)

g. Select one of the pods, and try deleting the pod.

h. While on the deployments screen, on the left menu, view events
i. While on the deployments screen, on the left menu, select "Insights".
j. Expand the controller name to see the container details.

k. Go back to the cluster view, and select Kubernetes resources -> Services and ingresses

l. View the details of the service we created (hello-deploy)
Please observe the cluster IP, external IP, and the ports / pods

m. Go back to the cluster view, and select Monitoring -> Insights
Observe the following tabs:

- Cluster
- Reports -> Resource Monitoring -> Deployments
- Nodes
- Controllers (observe the hello-deploy controller)
- Containers (observe the mynginx containers)

n. Go back to the cluster view, and select "Advisor Recommendations"

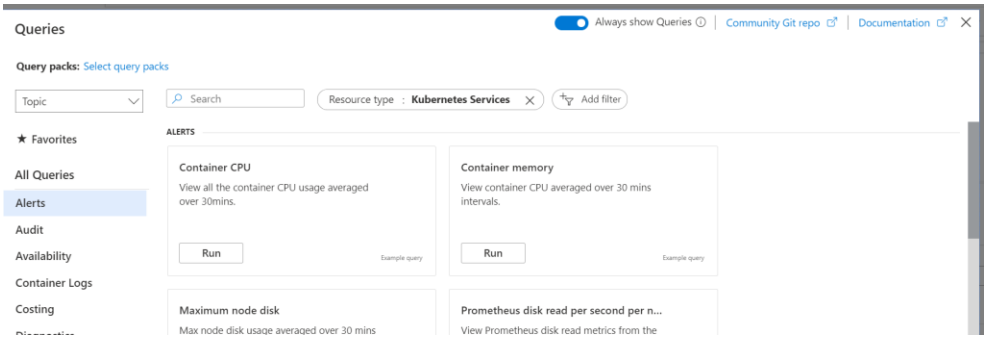
o. Go back to cluster view, and select Monitoring -> Metrics

For the metric, select "CPU usage millicores", and observe the graph.

p. Go back to the cluster view, and select Monitoring -> Alerts
   - Click "+New Alert Rule"
   - For the condition, click "Add condition" and select "CPU Usage Percentage"
   - On the alert logic, select the threshold value of 70
   - Explore the action group, etc. But don't create the alert rule.

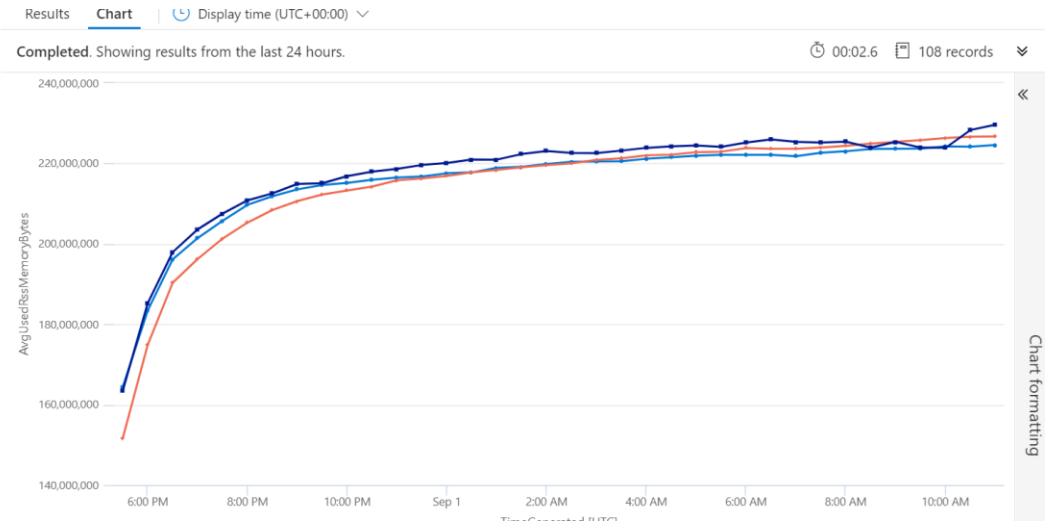
q. Go back to cluster view, and select Monitoring -> Logs

Click on Container Memory -> Run



For the query results, select "Chart".

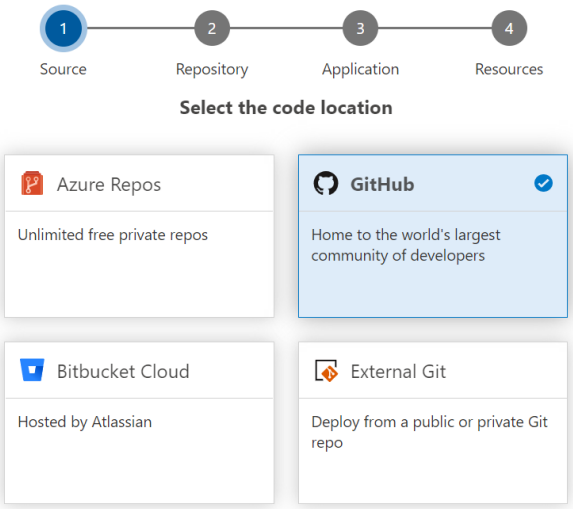
You should see a graph of container memory usage



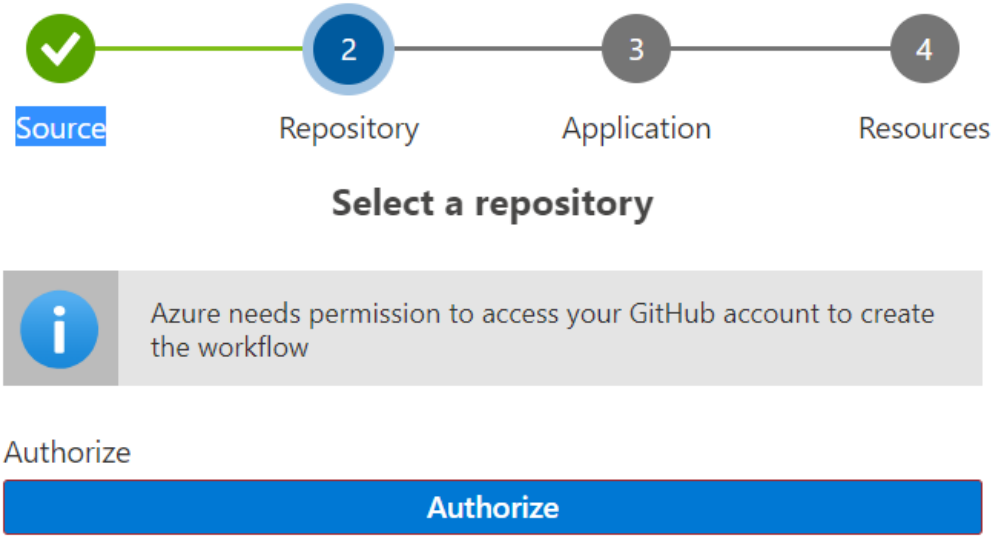
4. Continuous Deployment using DevOps pipelines (GitHub)

1. Signup / login to your GitHub account (www.github.com)

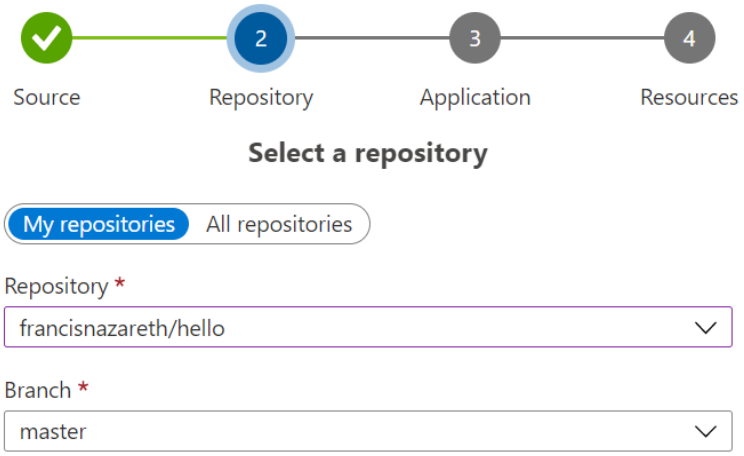
2. Fork the following repository -> [francisnazareth/hello (github.com)](https://github.com/francisnazareth/hello)
3. In the Azure Portal, search for the Kubernetes cluster you crated and go to the settings -> Deployment Center
4. Click on (+Add Projects)
5. In the source, select "GitHub" and click Next



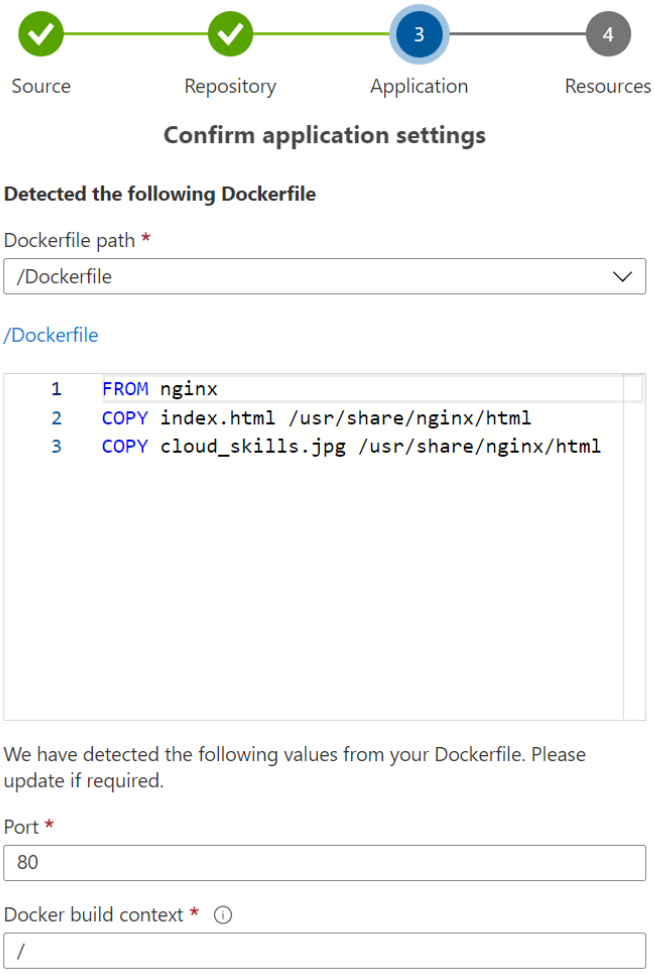
6. In the repository screen, click on "Authorize" to authorize azure to access GitHub.



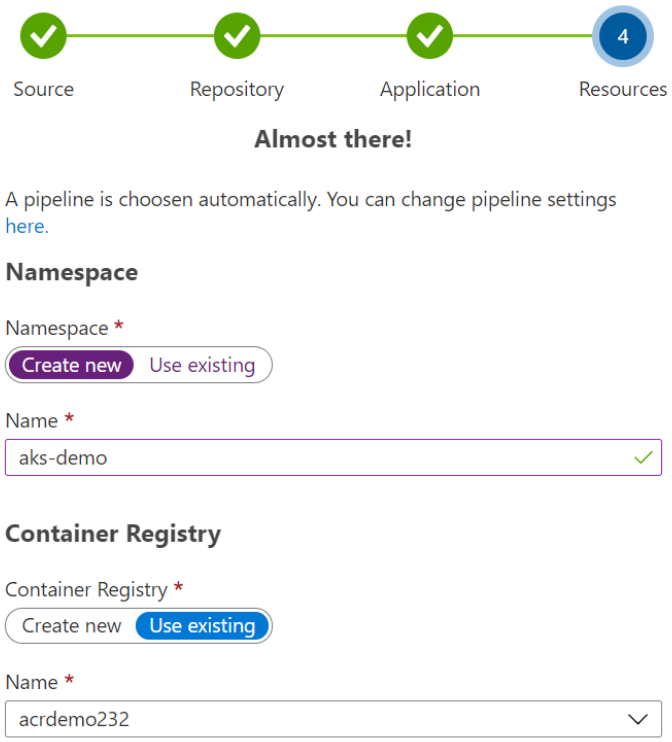
7. Once authorization is done, select the repository you forked



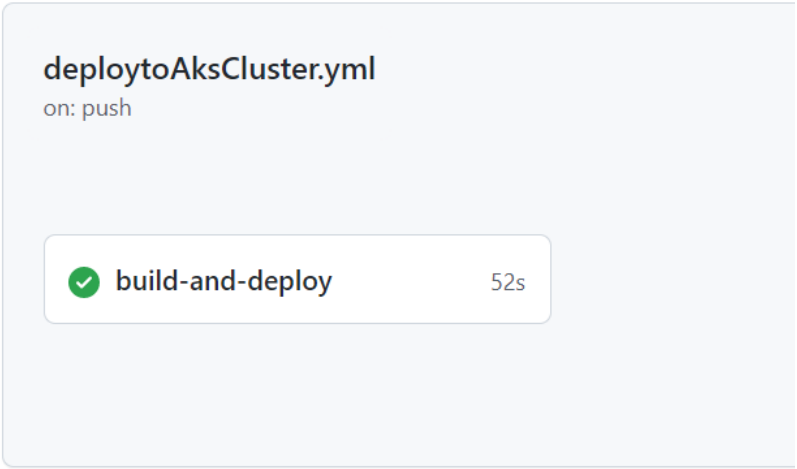
8. In the next screen, accept the defaults (deployment center automatically detects the Dockerfile).



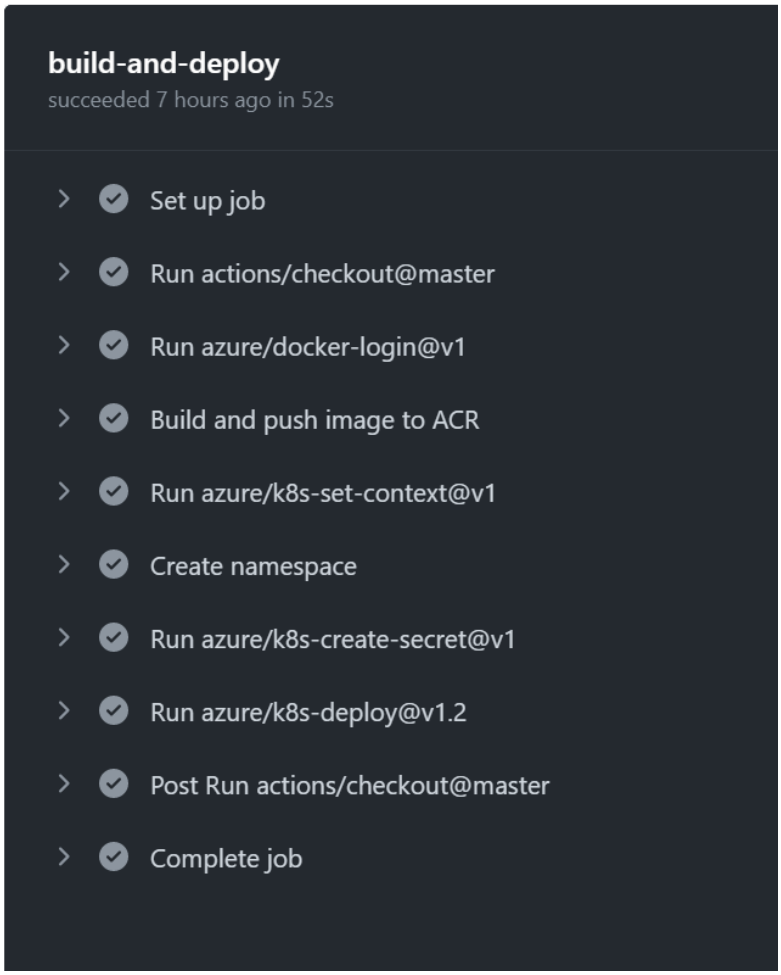
9. Accept the defaults for the namespace, select the Azure Container Registry that you created, and click "Done".



10. In the project in deployment center, click on "View all runs"
11. This will take you to GitHub page. Click on the "Build & Deploy" GitHub action.



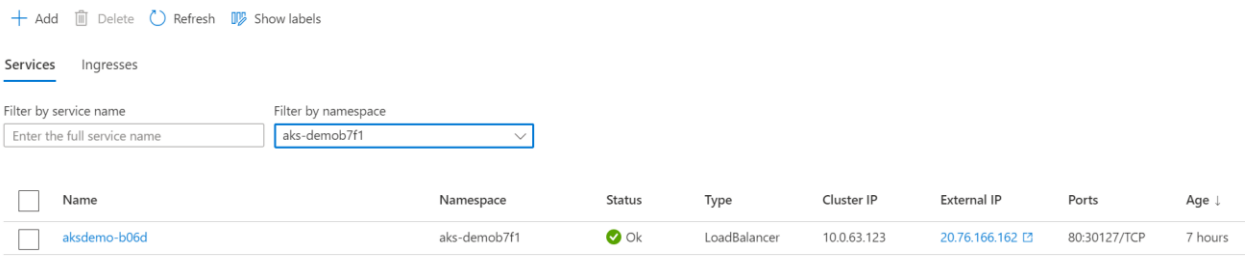
12. You should be able to view the pipeline steps that got executed.



13. In Azure Portal, within Deployment Center, view the manifest files deployment.yml and service.yml that got generated.

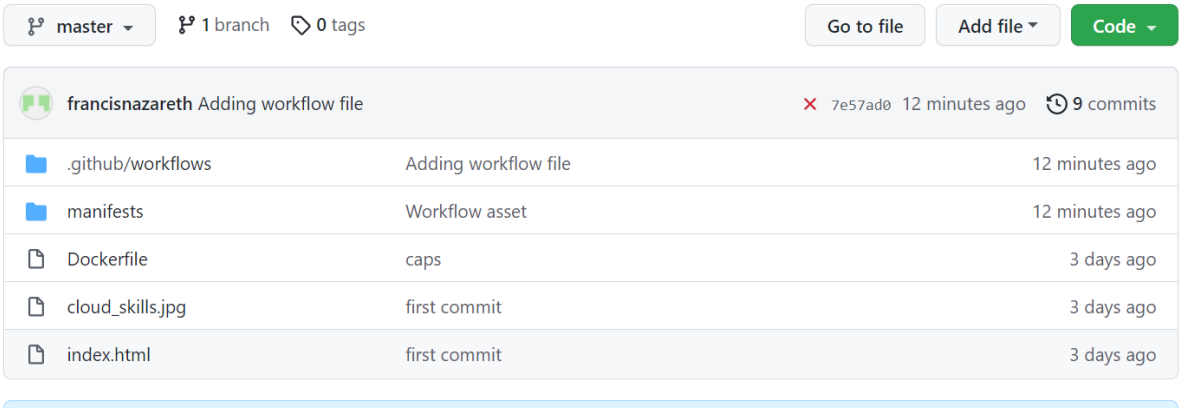
Also, make a note of the namespace that got generated.

14. In Azure Portal, within the cluster view, select "Kubernetes Resources -> Services and Ingresses". In Filter by namespace field select the namespace where deployment center created resources.

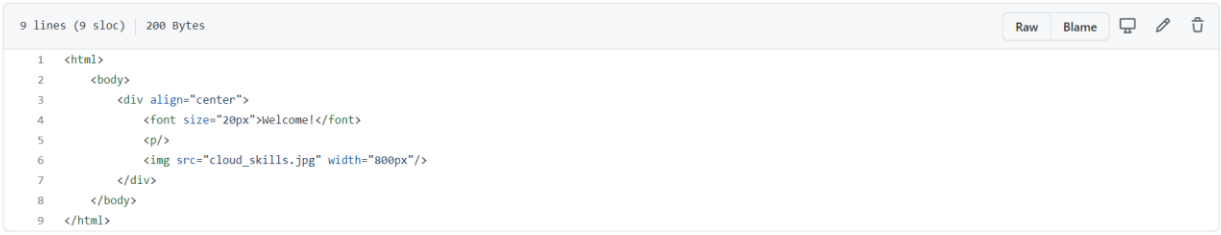


15. Click on the External IP to view the container application.

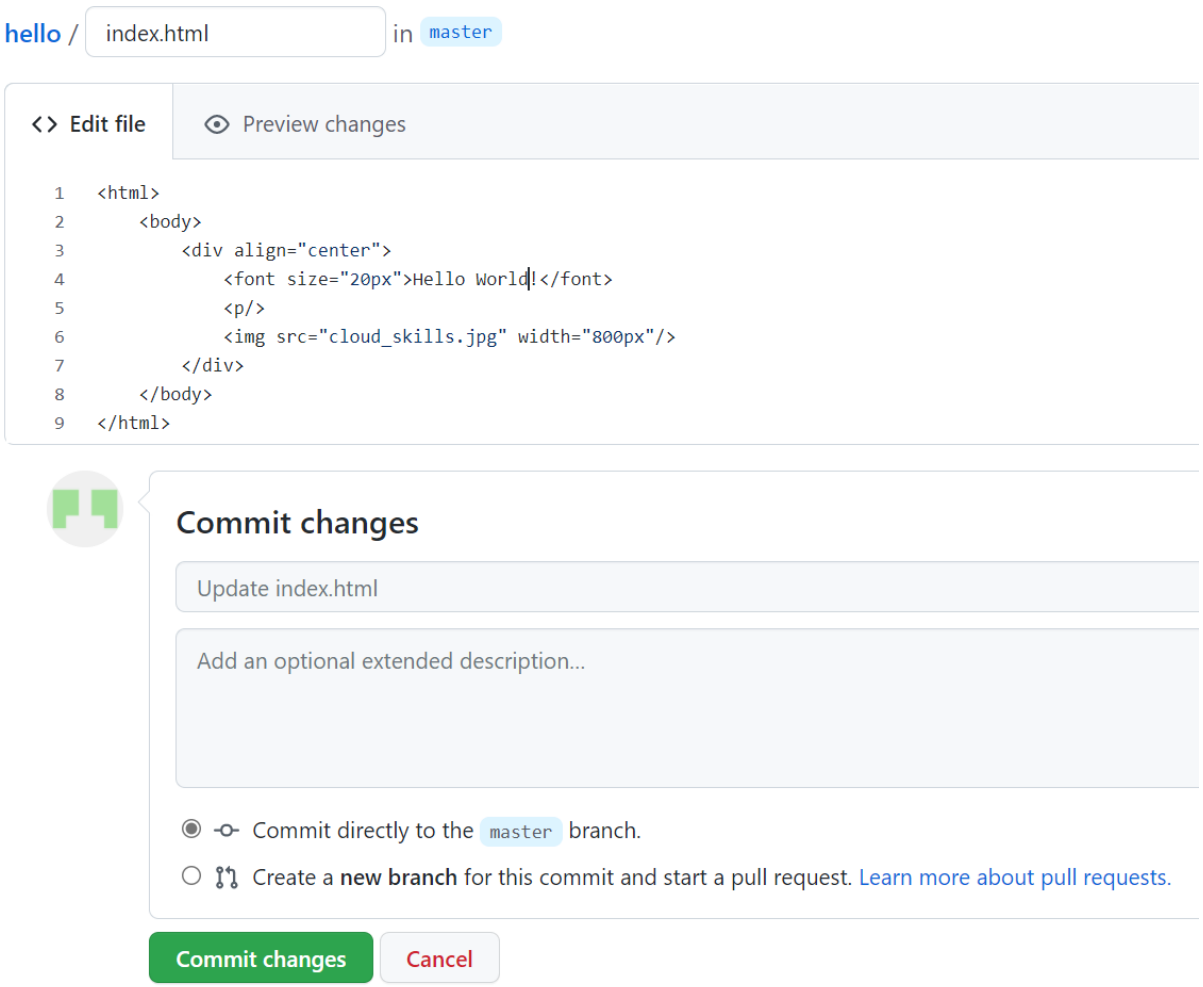
16. Go back to GitHub, go to “Code” tab, and select the file “index.html”



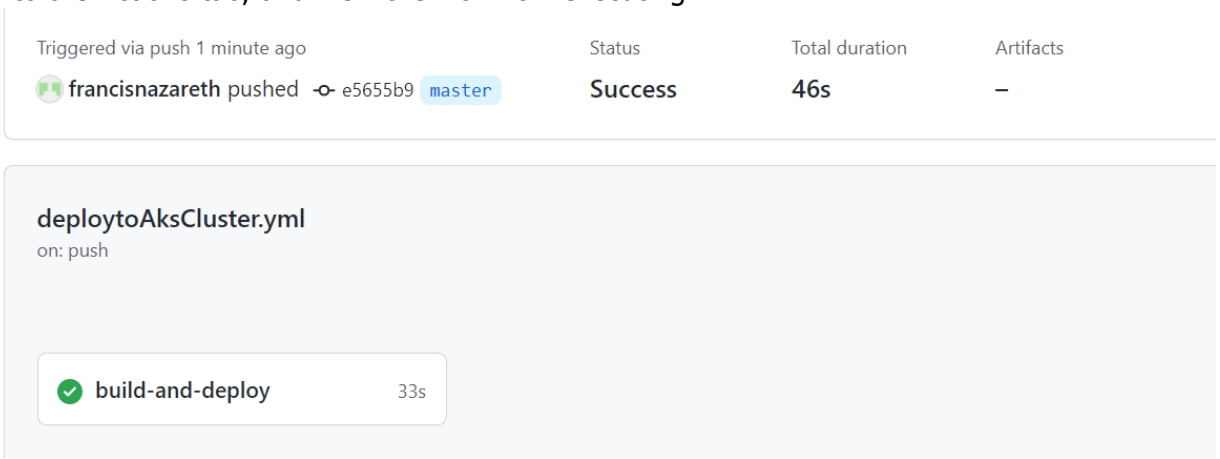
17. Edit the file (using the pencil icon)



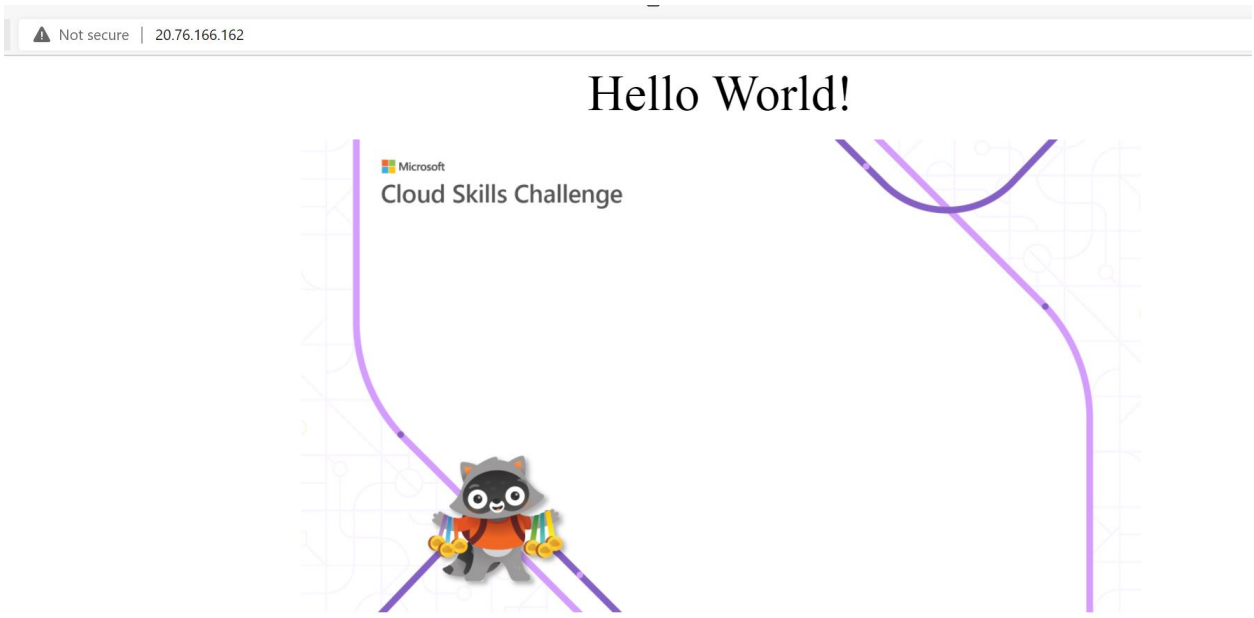
18. Edit the “Welcome!” text (in line 4) to something else, and click on “Commit Changes”.



19. Go to the Actions tab, and view the workflow executing.



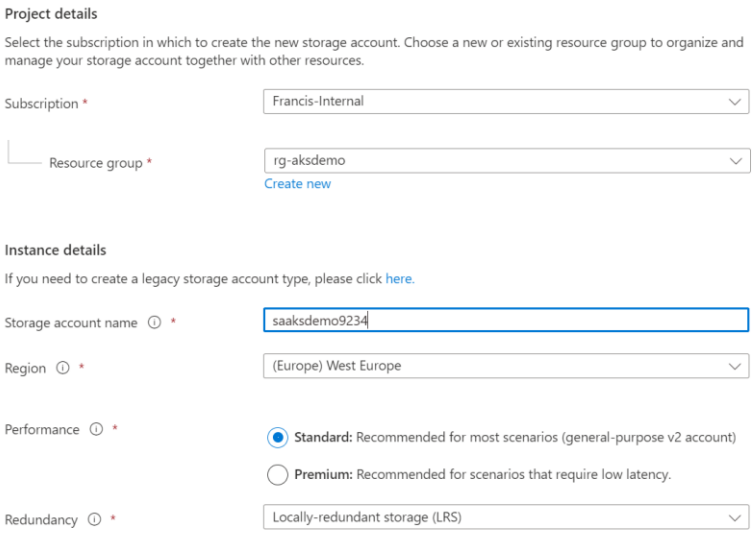
20. Refresh the web browser page with the Kubernetes service IP (from step 15). You should see the Kubernetes service referring to new container (built from the modified HTML file).



5. Mount Azure File Share as a Kubernetes volume

5.1 Create a storage account

1. Create an azure storage account (search for storage accounts), click on +Create.
2. Provide a unique name for the storage account.
3. Select the resource group, and select the same region as that of resource group.
4. Select the performance as standard
5. Select the redundancy as Locally Redundant Storage (LRS)



Click on "Review & Create", and Create.

5.2 Create a file share and upload an image to file share.

Once the storage account is provisioned, click on "Go to resource" to view properties of the storage account.

Create a file share within the storage account.

- In Data storage -> file shares, click (+FileShare) to create a file share.
- Provide a name (images) to the file share. And click Create.

New file share

Name *

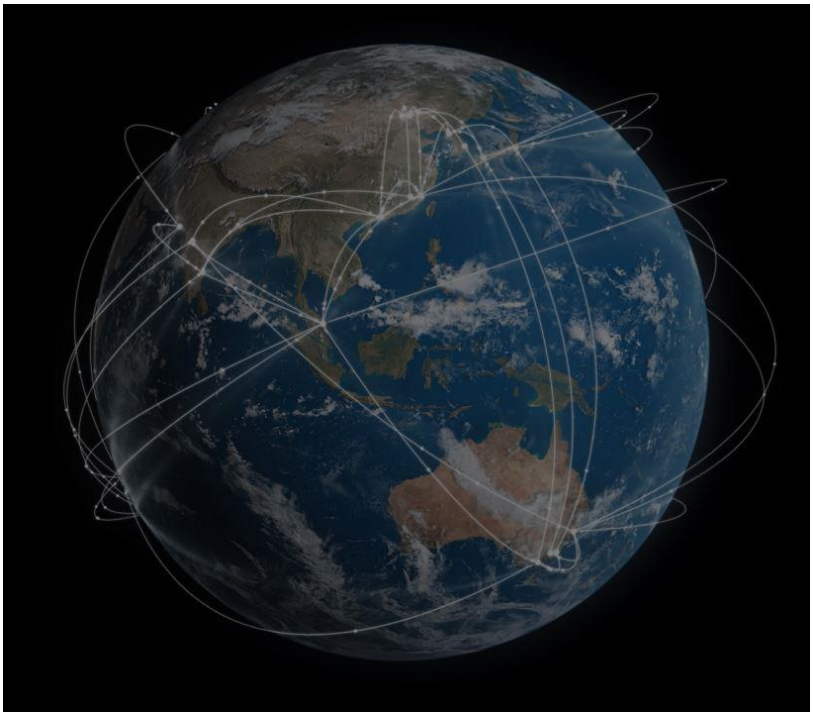
images

Tier ⓘ

Transaction optimized

Performance	
Maximum IO/s ⓘ	1000
Egress Rate ⓘ	60 MiBytes / s
Ingress Rate ⓘ	60 MiBytes / s
Maximum capacity	5 TiB
Large file shares	Disabled

Save the following picture as file "world.jpg" on your local machine.



- In Azure Portal, once the Azure file share is created, click on "Upload" to upload an image.
- Upload the image that you just saved to Azure File Share.

Copy the access key of storage account

- In the storage account, Go to "Security + Networking" -> Access Keys, and click on "Show Keys". Copy the value of key1.

Make a note of the following values:

1. Storage Account name
2. File Share name
3. Access Key of the storage account.

5.3 Create a Kubernetes secret.

In the cloud shell, create a Kubernetes secret using the following command:

kubectl create secret generic azure-file-secret --from-literal=azurestorageaccountname=<<storage account name>> --from-literal=azurestorageaccountkey=<<storage account key>>

For example,

kubectl create secret generic azure-file-secret --from-literal=azurestorageaccountname=saaksdemo9234 --from-literal=azurestorageaccountkey=NI7AJNUcL7Jwl59fMFYbMcR+fejoBIyjV2q0sgL8WLozjTDlyz2W9vifrmJti025pF3r7Ya3NubUuuFeg97FRA==

5.4 Create a new image and push to Azure Container Registry

Modify the HTML file (index.html) as follows:

```
<html>
  <body>
    <div align="center">
      <font size="20px">Welcome!</font>
      <p/>
      <img src="/images/world.jpg" width="600px"/>
    </div>
  </body>
```

```
</html>
```

Build a new image and push to the Azure Container Registry

az acr build --registry <<registry name>> --image <<image name>>:<<tag>>

For example,

az acr build --registry acrdemo232.azurecr.io --image nginx-mount:v1 .

5.6 Create a deployment that references Azure File Share (using the file share name and Kubernetes secret)

Create a deployment YAML file using the following command:

kubectl create deploy world-deploy --image <<registry URL>>/<<image>>:<<tag>> --replicas=3 --dry-run=client -o yaml > world-deploy.yaml

For example,

kubectl create deploy world-deploy --image acrdemo232.azurecr.io/nginx-mount:v1 --replicas=3 --dry-run=client -o yaml > world-deploy.yaml

Edit the file

code world-deploy.yaml

Add the sections volumeMounts and volumes as follows:

(Note: correct spaces – indentation – is important in YAML files).

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: hello-deploy
  name: hello-deploy
spec:
```

```
  replicas: 3
  selector:
    matchLabels:
      app: hello-deploy
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: hello-deploy
    spec:
      containers:
      - image: acrdemo232.azurecr.io/nginx-mount:v1
        name: mynginx
        resources: {}
        volumeMounts:
        - name: my-images
          mountPath: /usr/share/nginx/html/images
      volumes:
      - name: my-images
        azureFile:
          secretName: azure-file-secret
          shareName: images
          readOnly: false
status: {}
```

Save & Quit (CTRL + S, followed by CTRL + Q)

Apply the YAML file

kubectl apply -f world-deploy.yaml

5.6 Expose the deployment as a service.

Expose the deployment as a Kubernetes service:

kubectl expose deploy world-deploy --port=80 --type=LoadBalancer

View the public IP address for the service:

kubectl get svc

Access the IP address in a browser, you should be able to see the web site with image loaded from Azure Storage account.

(Note: We mounted the azure file share to container in the path /usr/share/nginx/html/images. Hence an HTML file in the path /usr/share/nginx/html/ is able to access the image as /images/world.jpg).

- End of Lab