

# App Modernization Labs

## Azure Kubernetes Service

Author: Francis S Nazareth

Cloud Solutions Architect, Azure App Dev, Microsoft Qatar

[fnazaret@microsoft.com](mailto:fnazaret@microsoft.com) / +974 33134296

## Contents

Pre-Requisites .....	3
1. Create Resources.....	3
1.1 Create a resource group.....	3
1.2 Create a virtual network and subnets .....	4
1.3 Create Azure Container Registry .....	6
1.4 Create Azure Kubernetes Service instance.....	8
2. Create a container application and deploy to AKS .....	14
3. Monitor the cluster using Azure Monitor.....	19
4. Continuous Deployment using DevOps pipelines (GitHub).....	21
5. Mount Azure File Share as a Kubernetes volume .....	28
5.3 Create a Kubernetes secret. ....	31
5.4 Create a new image and push to Azure Container Registry .....	31
5.6 Create a deployment that references Azure File Share (using the file share name and Kubernetes secret) .....	32
5.6 Expose the deployment as a service. ....	34
Lab 6: Creating an ingress controller .....	35
Lab 7: Publish utility microservices.....	38

## Pre-Requisites

1. You should have owner access to a 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.

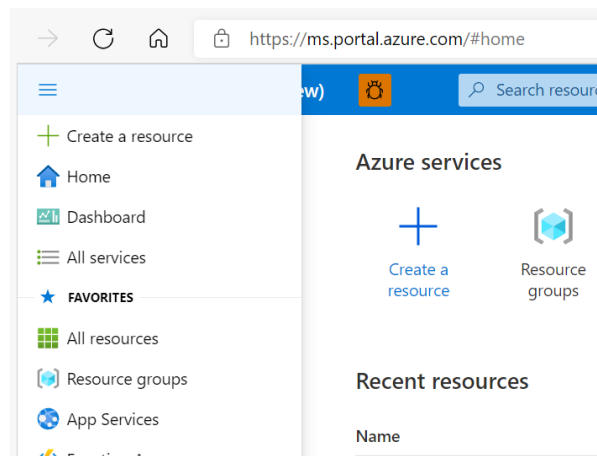
Steps:

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).

## Create a resource group ...

Basics   Tags   Review + create

**Resource group** - A container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group. You decide how you want to allocate resources to resource groups based on what makes the most sense for your organization. [Learn more](#) ⓘ

### Project details

Subscription * ⓘ	Francis-Internal	▼
Resource group * ⓘ	rg-aksdemo	✓

### Resource details

Region * ⓘ	(Europe) 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

(Note: We are creating a virtual network and subnet to use the Azure CNI network plugin in Azure Kubernetes Service. Azure CNI network plugin offers several advantages over Kubenet, the second network plugin option in AKS).

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

Click on +Create

From the menu, select "Networking"

Under "Virtual Networks", click "Create".

Recently created

Categories

AI + Machine Learning

Analytics

Blockchain

Compute

Containers

Databases

Developer Tools

DevOps

Identity

Integration

Internet of Things

IT & Management Tools

Media

Migration


Mixed Reality

Monitoring & Diagnostics

Networking


Security

Popular Azure services [See more in All s](#)




Application Gateway

[Create](#) | [Learn more](#)




App Service Domain

[Create](#) | [Learn more](#)




Public IP address

[Create](#) | [Learn more](#)




Reserved IP Address

[Create](#) | [Learn more](#)




Route table

[Create](#) | [Learn more](#)




Traffic Manager profile

[Create](#) | [Docs](#) | [MS Learn](#)




Virtual network

[Create](#) | [Docs](#) | [MS Learn](#)



Virtual network gateway

[Create](#) | [Learn more](#)



Virtual WAN

[Create](#) | [Learn more](#)

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>".

Project details

Subscription \* ⓘ

Francis-Internal

Resource group \* ⓘ

rg-aksdemo

[Create new](#)

Instance details

Name \*

vnet-aksdemo

Region \*

(Europe) West Europe

In the IP addresses tab, delete the default IPV4 Address Space.  
Add an IPV4 address space. For example: 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

Click "Add" to add the subnet

Subnet name \*

Subnet address range \* ⓘ  
  
20.0.0.0 - 20.0.0.255 (251 + 5 Azure reserved addresses)

**NAT GATEWAY**

Simplify connectivity to the internet using a network address translation gateway. Outbound connectivity is possible without a load balancer or public IP addresses attached to your virtual machines. [Learn more](#)

NAT gateway

**SERVICE ENDPOINTS**

Create service endpoint policies to allow traffic to specific azure resources from your virtual network over service endpoints. [Learn more](#)

Services ⓘ

Basics **IP Addresses** Security Tags Review + create

The virtual network's address space, specified as one or more address prefixes in CIDR notation (e.g. 192.168.1.0/24).

**IPv4 address space**

✓

☐ Add IPv6 address space ⓘ

The subnet's address range in CIDR notation (e.g. 192.168.1.0/24). It must be contained by the address space of the virtual network.

[+ Add subnet](#) [Remove subnet](#)

<input type="checkbox"/> Subnet name	Subnet address range	NAT gateway
<input type="checkbox"/> <a href="#">snet-aks</a>	20.0.0.0/24	-

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"

[Home](#) > [rg-aksdemo](#) >

## Create a resource ...

[Get Started](#)

Recently created

Popular Azure se

**Categories**


AI + Machine Learning


Analytics


Blockchain

Compute

Containers


 **Virtual m**  
[Create](#) | [Le](#)


 **Kubern**  
[Create](#) | [D](#)


 **Azure Co**  
[Create](#) | [D](#)


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


Popular Azure services [See more in All services](#)


 **Kubernetes Service**  
[Create](#) | [Docs](#) | [MS Learn](#)

 **Web App for Containers**  
[Create](#) | [Docs](#) | [MS Learn](#)

 **Batch Service**  
[Create](#) | [Docs](#) | [MS Learn](#)

 **Kubernetes - Azure Arc**  
[Create](#) | [Learn more](#)

 **Container Instances**  
[Create](#) | [Learn more](#)

 **Container Registry**  
[Create](#) | [Docs](#) | [MS Learn](#)

Specify a unique name for the container registry (for example, acrdemo followed by few random numbers). **Please don't use any capital letters in container registry name.**

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

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

**Project details**

Subscription \*

Resource group \*  [Create new](#)

**Instance details**

Registry name \*  ✓  
.azurecr.io

Location \*

Availability zones ⓘ ☐ Enabled

ℹ Availability zones are enabled on premium registries and in regions that support availability zones. [Learn more](#)

SKU \* ⓘ

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.

Registry name  📄

Login server  📄

Admin user ⓘ ☒ Enabled

Username  📄

**Note:** Please explore other options in Azure Container Registry screen, such as geo-replication.

## 1.4 Create Azure Kubernetes Service instance.

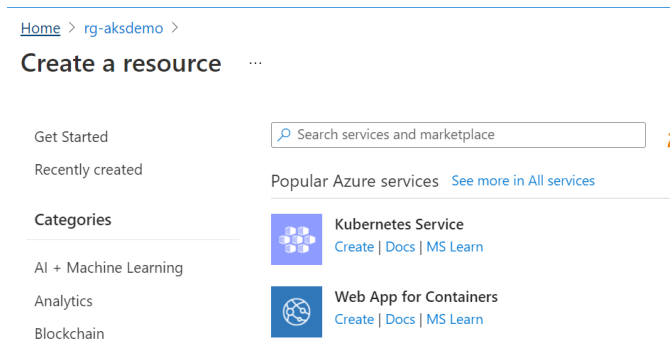
Go to the resource group (click on the resource group name)

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 resource group (that you created in previous step)

In the cluster details, in cluster pre-set configuration select Dev/Test (\$).

Provide a name for the kubernetes cluster (for example, akslab)

Select a region for the cluster (same as virtual network / resource group).

Leave the defaults for availability zone, kubernetes version, and API server availability.

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

## Create Kubernetes cluster ...

Subscription \*

Francis-Internal

Resource group \*

rg.aks

[Create new](#)

Cluster details

Cluster preset configuration

Dev/Test (\$)

To quickly customize your Kubernetes cluster, choose one of the preset configurations above. You can modify these configurations at any time.  
[Learn more and compare presets](#)

Kubernetes cluster name \*

akslab

Region \*

(Europe) West Europe

Availability zones

None

Kubernetes version \*

1.21.9 (default)


API server availability

☐ 99.9%

Optimize for availability. 99.95% is available when at least one availability zone is selected.

☒ 99.5%

Optimize for cost.



99.5% API server availability is recommended for dev/test configuration.

Primary node pool

The number and size of nodes in the primary node pool in your cluster. For production workloads, at least 3 nodes are recommended for resiliency. For development or test workloads, only one node is required. If you would like to add additional node pools or to see additional configuration options for this node pool, go to the 'Node pools' tab above. You will be able to add additional node pools after creating your cluster. [Learn more about node pools in Azure Kubernetes Service](#)

Node size \*

Standard B4ms

4 vcpus, 16 GiB memory

 Standard B4ms is recommended for dev/test configuration.

[Change size](#)

Scale method \*

☐ Manual

☒ Autoscale

Node count range \*

1

2


Click "Next > Node Pools"

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

Basics **Node pools** Authentication Networking Integrations Advanced Tags Review + create

### Node pools

In addition to the required primary node pool configured on the Basics tab, you can also add optional node pools to handle a variety of workloads [Learn more about node pools](#)

+ Add node pool  Delete

Name	Mode	OS type	Node count	Node size
<input type="checkbox"/> agentpool	System	Linux	1	Standard_B4ms


### Enable virtual nodes

Virtual nodes allow burstable scaling backed by serverless Azure Container Instances. [Learn more about virtual nodes](#)

Enable virtual nodes  ☐

### Enable virtual machine scale sets

Enabling virtual machine scale sets will create a cluster that uses virtual machine scale sets instead of individual virtual machines for the cluster nodes. Virtual machine scale sets are required for scenarios including autoscaling, multiple node pools, and Windows support. [Learn more about virtual machine scale sets in AKS](#)

Enable virtual machine scale sets  ☒

Click "Next > Authentication"

Leave the defaults for authentication.

Basics Node pools **Authentication** Networking Integrations Advanced Tags Review + create

### Cluster infrastructure

The cluster infrastructure authentication specified is used by Azure Kubernetes Service to manage cloud resources attached to the cluster. This can be either a [service principal](#) or a [system-assigned managed identity](#).

Authentication method ☐ Service principal ☒ System-assigned managed identity

### Kubernetes authentication and authorization

Authentication and authorization are used by the Kubernetes cluster to control user access to the cluster as well as what the user may do once authenticated. [Learn more about Kubernetes authentication](#)

Role-based access control (RBAC)  ☒ Enabled ☐ Disabled

AKS-managed Azure Active Directory  ☐

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".


## Create Kubernetes cluster ...

[Learn more about networking in Azure Kubernetes Service](#)

Network configuration ⓘ

☐ Kubenet

☒ Azure CNI

**i** The Azure CNI plugin requires an IP address from the subnet below for each pod on a node, which can more quickly exhaust available IP addresses if a high value is set for pods per node. Consider modifying the default values for pods per node for each node pool on the "Node pools" tab. [Learn more](#) 

Virtual network \* ⓘ

vnet-aksdemo ▼

[Create new](#)

Cluster subnet \* ⓘ

snet-aks (20.0.0.0/24) ▼

[Manage subnet configuration](#)

Kubernetes service address range \* ⓘ

10.0.0.0/16 ✓

Kubernetes DNS service IP address \* ⓘ

10.0.0.10

Docker Bridge address \* ⓘ

172.17.0.1/16 ✓

DNS name prefix \* ⓘ

aks-demo-dns ✓

### Traffic routing

Load balancer ⓘ

Standard

Enable HTTP application routing ⓘ

☐

### Security

Enable private cluster ⓘ

☐

Set authorized IP ranges ⓘ

☐

Network policy ⓘ

☐ None

☒ Calico

☐ Azure

Select "Next: Integrations>"

Select the container registry you created earlier.

Leave container monitoring as enabled.


Select Azure Policy to Enabled.

Basics   Node pools   Authentication   Networking   **Integrations**   Tags   Review + create

Connect your AKS cluster with additional services.

#### Azure Container Registry

Connect your cluster to an Azure Container Registry to enable seamless deployments from a private image registry. You can create a new registry or choose one you already have. [Learn more about Azure Container Registry](#)

Container registry    
[Create new](#)

#### Azure Monitor

In addition to the CPU and memory metrics included in AKS by default, you can enable Container Insights for more comprehensive data on the overall performance and health of your cluster. Billing is based on data ingestion and retention settings.

[Learn more about container performance and health monitoring](#)

[Learn more about pricing](#)

Container monitoring ☒ Enabled ☐ Disabled

Log Analytics workspace ⓘ    
[Create new](#)

#### Azure Policy

Apply at-scale enforcements and safeguards for AKS clusters in a centralized, consistent manner through Azure Policy.

[Learn more about Azure Policy for AKS](#)

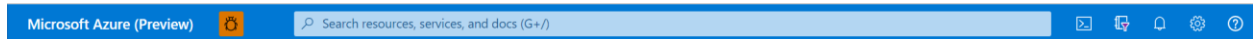
Azure Policy ☒ Enabled ☐ Disabled

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
```

**Note:** Containers follow a layered file system. Each command in the Dockerfile creates a layer. For example, the first line (FROM nginx) starts with the base layer – nginx. If this base image is not available on the local environment, the docker daemon looks for an image named nginx in the public registry (hub.docker.com) and pulls the base image to your environment. The second

line copies the file index.html from local folder to the folder within the container /usr/share/nginx/html/ .. This creates a second layer on top of the base layer (nginx), and builds a new image. (NGINX is a web server, and /usr/share/nginx/html/ is the default directory where NGINX web server looks for HTML files to serve). This image can be named / tagged, and can become the base layer for another image.

- 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 .**

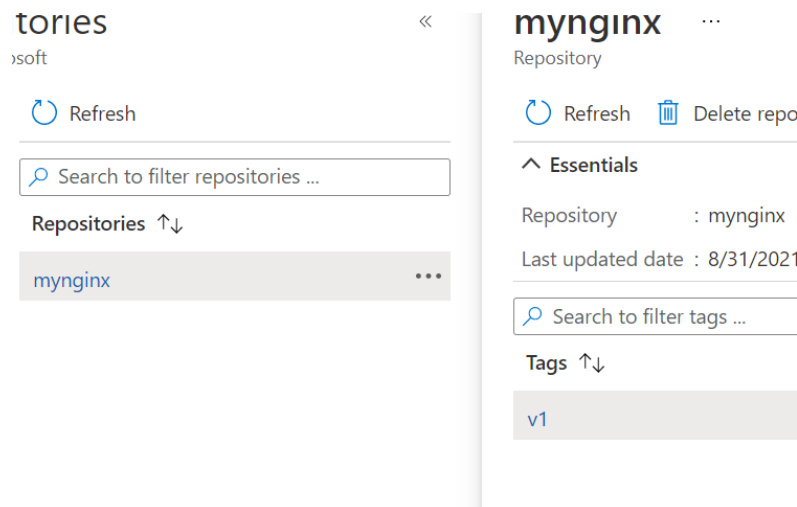
For example,

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

**Note:** The last parameter (.) denotes the current directory. Please remember to copy the dot (.) as well.

The command az acr build does four things: It looks for a file named Dockerfile in the directory specified (by dot). It builds a container based on the Dockerfile. It tags the Dockerfile with the registry name/image name and tag name (in this case acrdemo232.azurecr.io/mynginx and v1), and it uploads the newly built and tagged image to azure container registry you specified (with the --registry parameter).

- 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 set the current Kubernetes cluster).

***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***

```
Bash
Requesting a Cloud Shell.Succeeded.
Connecting terminal...

Welcome to Azure Cloud Shell

Type "az" to use Azure CLI
Type "help" to learn about Cloud Shell

francis@Azure:~$
francis@Azure:~$
francis@Azure:~$
francis@Azure:~$ az aks get-credentials --resource-group rg-aksdemo --name aks-demo
Merged "aks-demo" as current context in /home/francis/.kube/config
francis@Azure:~$
```

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***

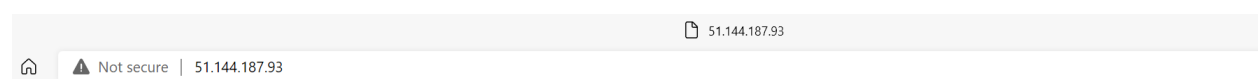
- k. Check the IP address of the service that got created, using the command:

***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.



Welcome!

Note: We are creating a Kubernetes deployment and a Kubernetes service. The following links may be helpful to understand what are pods, deployments and services.

**Pods** – pods are the smallest deployable units of computing that you can create and manage in Kubernetes. [Pods | Kubernetes](#)

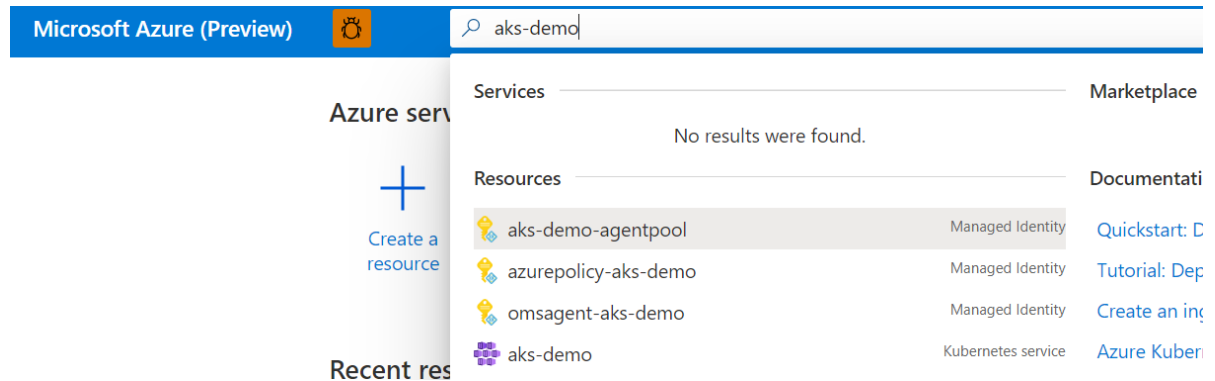
**Deployment** – Deployment is a collection of identical pods. You can specify the number of replicas pods in the deployment (using the replicas parameter). This is the desired state (desired number of pods) and Kubernetes will try to maintain this pod count within a deployment. [Deployments | Kubernetes](#)

**Service** – Service is way to expose your pods as a network resource. In other words, you can think of the service as a load balancer in front of your deployment (pods) with an internal DNS name for the collection of pods. (The service front ends a deployment. I.e., the service load balances the traffic to the group of pods in a deployment). The service could have an internal or external IP address, and the traffic received on the service will be load balanced to the pods behind the service. [Service | Kubernetes](#)

In our case, we created a deployment with three pods. The service we created (using the expose command) creates an Azure Standard Load Balancer, which load balances any requests to the service to the three pods in our deployment.

### 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.

(Please remember that we specified the desired number of pods through --replicas parameter when we created the deployment. If one of the pods gets killed / deleted, Kubernetes will automatically clone the pods to maintain the replica count we specified. In other words, Kubernetes is self-healing).

- 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

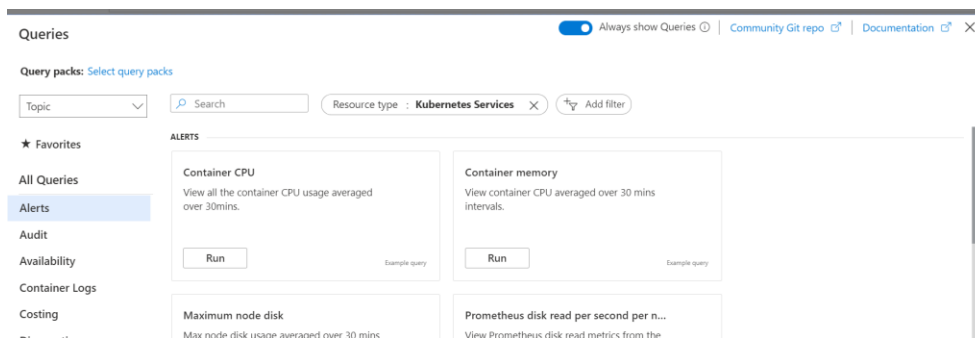
- Reports -> Resource Monitoring -> Deployments
- Nodes
- Controllers (observe the hello-deploy controller)
- Containers (observe the mynginx containers)
- n. Go back to the cluster view, and select "Advisor Recommendations"
- o. Go back to cluster view, and select Monitoring -> Metrics

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.

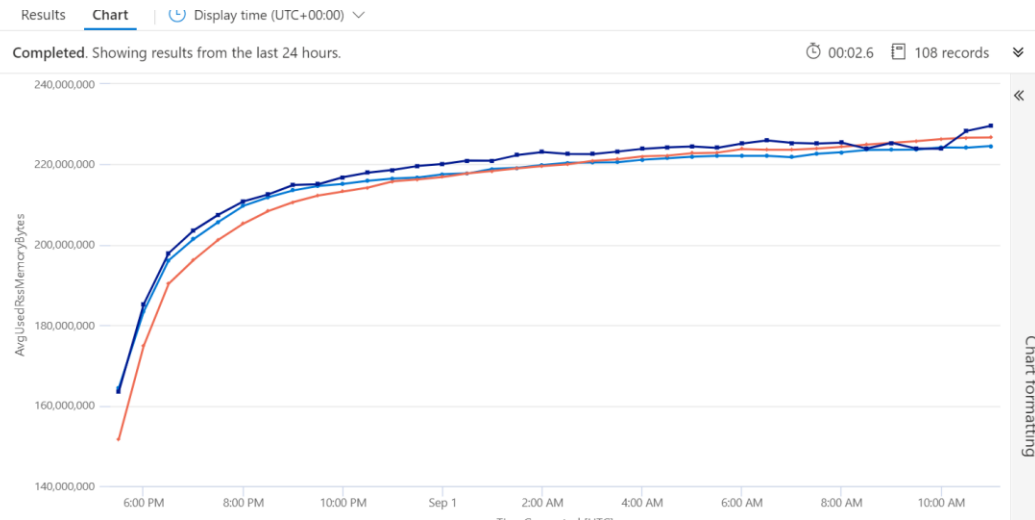
- q. Go back to cluster view, and select Monitoring -> Logs

Click on Container Memory -> Run



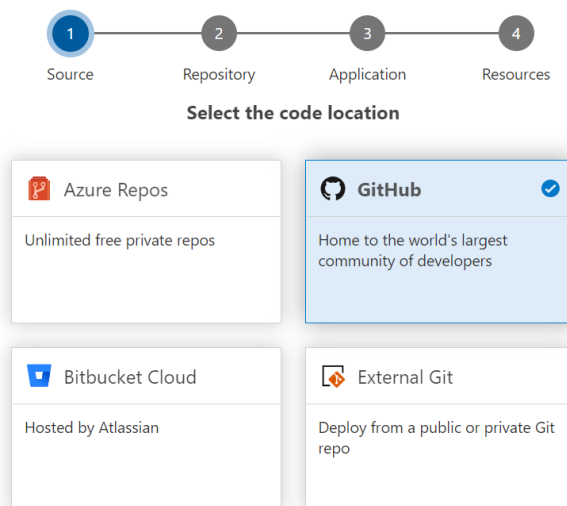
For the query results, select "Chart".

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](https://www.github.com))
2. Fork the following repository -> [francisnazareth/hello](https://github.com/francisnazareth/hello) ([github.com](https://github.com))
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.



### Select a repository

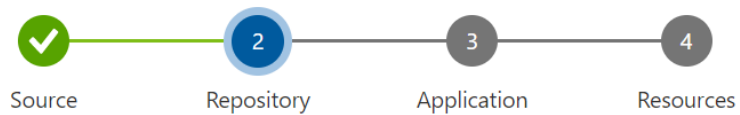


Azure needs permission to access your GitHub account to create the workflow

Authorize

Authorize

- Once authorization is done, select the repository you forked



### Select a repository

My repositories All repositories

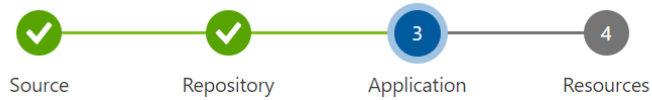
Repository \*

francisnazareth/hello

Branch \*

master

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



### Confirm application settings

#### Detected the following Dockerfile

Dockerfile path \*

/Dockerfile

[/Dockerfile](#)

```
1 FROM nginx
2 COPY index.html /usr/share/nginx/html
3 COPY cloud_skills.jpg /usr/share/nginx/html
```

We have detected the following values from your Dockerfile. Please update if required.

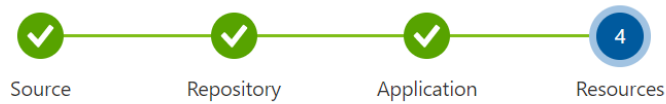
Port \*

80

Docker build context \* ⓘ

/

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



**Almost there!**

A pipeline is chosen automatically. You can change pipeline settings [here](#).

### Namespace

Namespace \*

Create new Use existing

Name \*

aks-demo ✓

### Container Registry

Container Registry \*

Create new Use existing

Name \*

acrdemo232 ✓

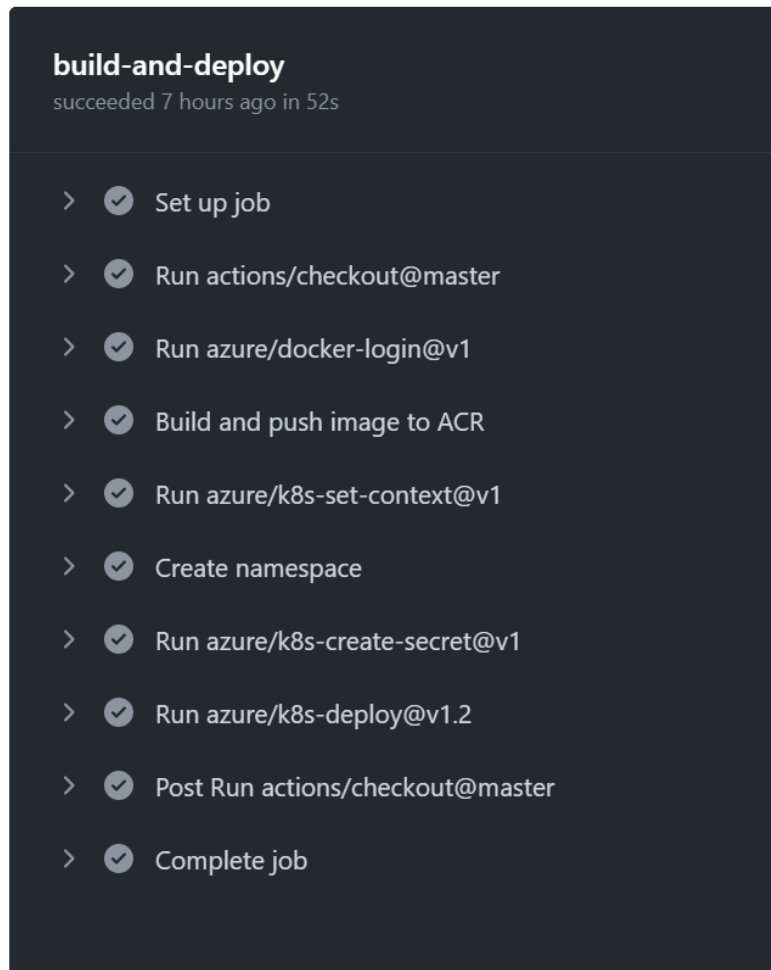
10. In the project in deployment center, click on "View all runs"
11. This will take you to GitHub page. Click on the "Build & Deploy" GitHub action.

**deploytoAksCluster.yml**  
on: push

✓ build-and-deploy 52s

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.

Namespace	Dockerfile	Manifest	Pipeline	Repository	Updated
aks-demob7f1	Dockerfile fnacr100.azurecr.io	deployment.yml service.yml	deploytoAksCluster.yml	hello  master	6 minutes ago (View all runs)

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.

[+ Add](#)
[Delete](#)
[Refresh](#)
[Show labels](#)


[Services](#)
[Ingresses](#)


Filter by service name: 
 Filter by namespace:


<input type="checkbox"/>	Name	Namespace	Status	Type	Cluster IP	External IP	Ports	Age ↓
<input type="checkbox"/>	aksdemo-b06d	aks-demob7f1	✔ Ok	LoadBalancer	10.0.63.123	20.76.166.162 <a href="#">🌐</a>	80:30127/TCP	7 hours

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"

 master ▾


 1 branch






 0 tags

Go to file

Add file ▾

Code ▾




 francisnazareth Adding workflow file ✖ 7e57ad0 12 minutes ago 🕒 9 commits

 .github/workflows	Adding workflow file	12 minutes ago
 manifests	Workflow asset	12 minutes ago
 Dockerfile	caps	3 days ago
 cloud_skills.jpg	first commit	3 days ago
 index.html	first commit	3 days ago

17. Edit the file (using the pencil icon)

9 lines (9 sloc) | 200 Bytes

RawBlame




```
1 <html>
2   <body>
3     <div align="center">
4       <font size="20px">Welcome!</font>
5     <p/>
6     
7   </div>
8 </body>
9 </html>
```

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

hello /  in master

<> Edit file Preview changes

```
1 <html>
2   <body>
3     <div align="center">
4       <font size="20px">Hello World!</font>
5     <p/>
6     
7   </div>
8 </body>
9 </html>
```



### Commit changes



Update index.html

Add an optional extended description...

☒ Commit directly to the master branch.  
☐ Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)


Commit changes Cancel

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

Triggered via push 1 minute ago	Status	Total duration	Artifacts
 francisnazareth pushed  e5655b9 <span>master</span>	<b>Success</b>	<b>46s</b>	—

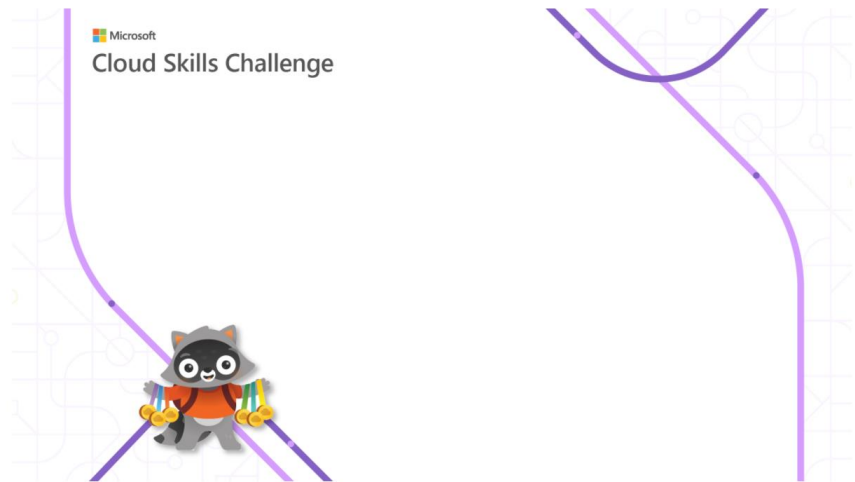
#### deploytoAksCluster.yml

on: push

 build-and-deploy 33s

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).

# Hello World!



## 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)

#### Project details

Select the subscription in which to create the new storage account. Choose a new or existing resource group to organize and manage your storage account together with other resources.

Subscription \*

Resource group \*   
[Create new](#)

#### Instance details

If you need to create a legacy storage account type, please click [here](#).

Storage account name ⓘ \*

Region ⓘ \*

Performance ⓘ \* ☒ **Standard:** Recommended for most scenarios (general-purpose v2 account)  
☐ **Premium:** Recommended for scenarios that require low latency.

Redundancy ⓘ \*

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 \*

Tier ⓘ

### 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 (right click, save picture..).



- 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:

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

For example,

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

**Note:** Secret is a Kubernetes object that is used to store sensitive data. [Secrets | Kubernetes](#)

### 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/>
```

```

</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 --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:

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

For example,

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

**Note:** The `--dry-run` instructs the Kubernetes API server to not to create the objects. `-o` parameter specifies that the output should be in YAML format. We are redirecting the yaml format output to a file (using pipe `>`)

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: world-deploy

name: world-deploy

spec:

replicas: 3

selector:

matchLabels:

app: world-deploy

strategy: {}

template:

metadata:

creationTimestamp: null

labels:

app: world-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.

- a. Expose the deployment as a Kubernetes service:

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

- b. View the public IP address for the service:

kubectl get svc

- c. 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`.

Please note that we are referring to the storage account and file share in the YAML using the secret name (which contains the storage account name and storage account key) and the file share name, in the volumes section of the YAML. The volume name has to match with the volume mount name. We specify the mount path in the volume mount section.

## 6: Create an ingress controller and routes

Note that each of our services had a public IP address. This is not ideal – we should ideally have only one access IP address for the Kubernetes cluster, and should have multiple deployments routes through this common IP address. For example, <ip address>/hello should go to our first deployment, and <ip address>/world should go to our second deployment.

Let us create an (NGINX) ingress controller and route the traffic through the ingress controller.

### 6.1 Install the nginx ingress controller using helm

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo update
kubectl create namespace ingress-nginx
helm install ingress-one ingress-nginx/ingress-nginx --set
controller.ingressClass="ingress-one" --namespace ingress-nginx --version 3.39.0
```

### 6.2 Delete the existing services

```
kubectl delete service hello-deploy
kubectl delete service world-deploy
```

### 6.3 Recreate the services again, this time with ClusterIP as the type.

```
kubectl expose deploy hello-deploy --port=80 --target-port=80 --type=ClusterIP
kubectl expose deploy world-deploy --port=80 --target-port=80 --type=ClusterIP
```

**Note:** The services that we created earlier were of type LoadBalancer, and hence the service got a public IP address. Now, we are creating the service to be of type ClusterIP, which doesn't provide a public IP address. Services of the type ClusterIP can be accessed by objects within the Kubernetes cluster but not from outside.

### 6.4 Check the IP addresses of the services

```
kubectl get services
```

(You will see that the services do not have a public IP anymore).

6.5 Create an ingress resource using the following file (copy the contents to a file hello-world-ingress.yaml)

code hello-world-ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hello-world-ingress
  annotations:
    kubernetes.io/ingress.class: ingress-one
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  rules:
  - http:
      paths:
      - path: /world(/|$)(.*)
        pathType: Prefix
        backend:
          service:
            name: world-deploy
            port:
              number: 80
      - path: /hello(/|$)(.*)
        pathType: Prefix
        backend:
          service:
            name: hello-deploy
            port:
              number: 80
      - path: /(.*)
        pathType: Prefix
        backend:
          service:
            name: world-deploy
            port:
              number: 80
  ---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hello-world-ingress-static
```

```

annotations:
  kubernetes.io/ingress.class: ingress-one
  nginx.ingress.kubernetes.io/ssl-redirect: "false"
  nginx.ingress.kubernetes.io/rewrite-target: /static/$2
spec:
  rules:
  - http:
      paths:
      - path:
          pathType: Prefix
          backend:
            service:
              name: world-deploy
              port:
                number: 80
          path: /static(/|$)(.*)

```

Create the ingress resource:

```
kubectl apply -f hello-world-ingress.yaml
```

6.6 Get the external IP address of ingress

```
kubectl --namespace ingress-nginx get services
```

6.7 Try to access the website with following URLs

```
http://<ingress external IP>/
```

(this goes to the default path / in our ingress routing rules)

```
http://<ingress external IP>/hello
```

(this goes to the path /hello in our ingress routing rules)

```
http://<ingress external IP>/world
```

(this goes to the path /world in our ingress routing rules)

## 7: Publish utility microservices

Four utility microservices are provided in the following Github location:

Microservice	URL
Water and Electricity Usage Microservice	<a href="https://github.com/francisnazareth/microservices">microservices/WaterAndElectricityUsage at main · francisnazareth/microservices (github.com)</a>
Mobile and Internet Bill Microservice	<a href="https://github.com/francisnazareth/microservices">microservices/MobileAndInternetBill at main · francisnazareth/microservices (github.com)</a>
Loyalty Points Microservice	<a href="https://github.com/francisnazareth/microservices">microservices/LoyaltyPoints at main · francisnazareth/microservices (github.com)</a>
Health Statistics Microservice	<a href="https://github.com/francisnazareth/microservices">microservices/HealthStats at main · francisnazareth/microservices (github.com)</a>

If you browse through the repositories you will see that each of the repository has a Dockerfile and dotnet source code.

Let us build and deploy these microservices

1. Clone the source code to your cloud shell environment

git clone <https://github.com/francisnazareth/microservices.git>

2. Build the container images, and push the images to Azure Container Registry

From Azure Cloud Shell, issue the following commands (replace <<your registry name>> with the name of container registry that you created:

```
RegistryName=<<your registry name>>
```

```
az acr build --registry $RegistryName --image electricitybill:v1  
./microservices/WaterAndElectricityUsage
```

```
az acr build --registry $RegistryName --image mobilebill:v1  
./microservices/MobileAndInternetBill
```

```
az acr build --registry $RegistryName --image loyaltypoints:v1 ./microservices/LoyaltyPoints
```

```
az acr build --registry $RegistryName --image healthstats:v1 ./microservices/HealthStats
```

3. Create an ingress controller for Microservices

```
helm install ingress-microsvc ingress-nginx/ingress-nginx --set  
controller.ingressClass="microsvc" --namespace ingress-nginx --version 3.39.0
```

4. Create the microservice deployments, services, and ingress routes using a YAML file.

- a) Edit the file `microsvc.yml`

***code microservices/microsvc.yml***

- b) Replace the word `YOUR_REGISTRY` with the name of your container registry (CTRL + H). Note that there are four occurrences.
- c) Save the file (CTRL + S) and quit the editor (CTRL + Q)
- d) Create the Kubernetes artifacts using the command

***kubectl apply -f microservices/microsvc.yml***

5. Get the ingress IP address

***kubectl get ingress -n microservices***

6. Try invoking microservices in a browser:

- a) Mobile & Internet Bill microservice
  - Service URL: `http://<<ingress IP>>/mobilebill/api/MobileBill/88888888`
  - OpenAPI definition: `http://<<ingress IP>>/mobilebill/swagger/v1/swagger.json`
- b) Water & Electricity Bill microservice
  - Service URL: `http://<<ingress IP>>/electricitybill/api/UtilityBill/1234`
  - Open API definition: `http://<<ingress IP>>/electricitybill/swagger/v1/swagger.json`
- c) Loyalty Points microservice
  - Service URL: `http://<<ingress IP>>/loyalty/api/Loyalty/1234`
  - Open API definition: `http://<<ingress IP>>/loyalty/swagger/v1/swagger.json`
- d) Health Statistics
  - Service URL: `http://<<ingress IP>>/healthstats/api/HealthStats/1234`
  - Open API definition: `http://<<ingress IP>>/healthstats/swagger/v1/swagger.json`

We may use the microservices you deployed now in API management lab & app service lab.

- End of Lab