Deployment of Micro services with K8s and Azure Kubernetes Services (AKS)

1. Problem Statement

Modern software is increasingly run as fleets of containers, sometimes called Micro services. A complete application may comprise many containers, all needing to work together in specific ways.

Solution required to automate operational tasks of container management and includes built-in commands for deploying applications, rolling out changes to your applications, scaling your applications up and down to fit changing needs, and more — making it easier to manage applications.

In a production environment, need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start.

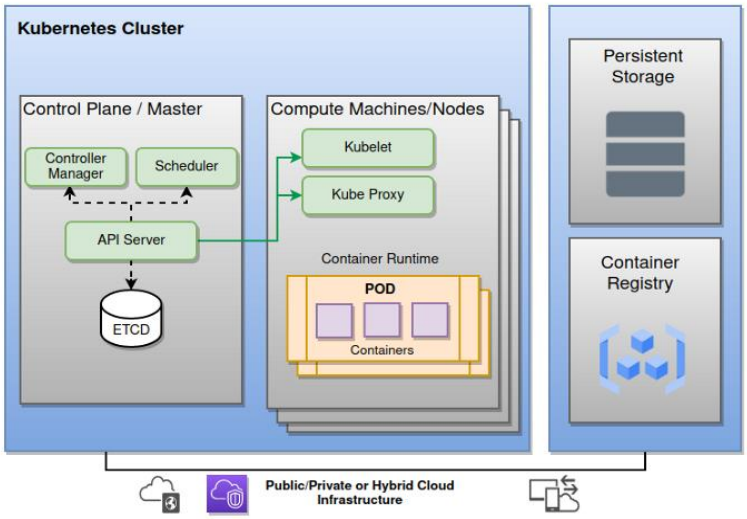
Solution Objectives

- **Automate** and **scale** application/microservices **up and down** based on the need.
- Deploy changes on production environment with **zero downtime**.
- Easy to **manage, monitor and maintain** the containerized application/microservices.

2. Solution / Architecture

Kubernetes (K8s) provides you with a framework to run containerised applications resiliently. It takes care of scaling and failover for your application provides deployment patterns, and more.

A Kubernetes cluster consists of a **set of** worker machines called **compute machines/nodes**, which run containerized applications. The **node(s) host the Pods** that are the components of the application workload. The **control plane** manages the worker nodes and **the Pods** in the cluster.



Architecture of Kubernetes (K8s) cluster

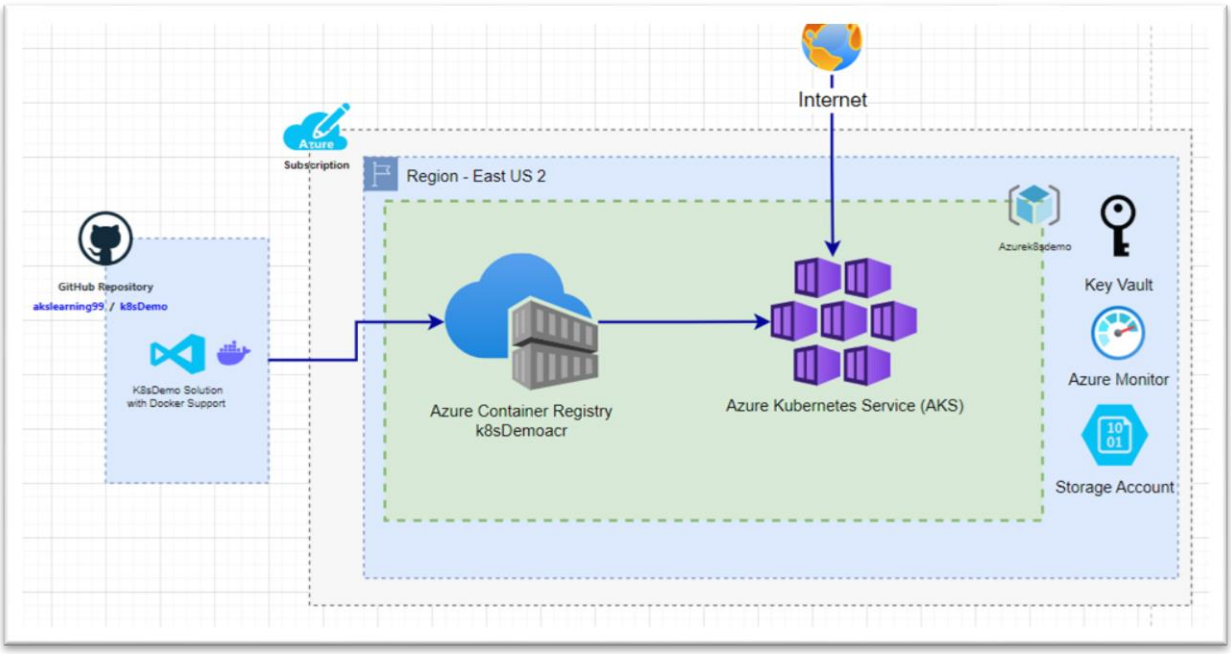
Although **Kubernetes is open source**, many **companies** planning to adopt Kubernetes **do not have the expertise or resources** to **set up and maintain the cluster** themselves.

Managed Kubernetes providers help those who looking to use Kubernetes, by providing them with the necessary support and maintenance of the Kubernetes clusters. A managed Kubernetes deployment should provide users with **a hassle-free control plane**, **easy deployment options, and on-going Kubernetes maintenance, enabling users to focus on their business and bringing their apps to market. Various vendors (Microsoft/ Google/AWS) offers Kubernetes-based platform that deploy Kubernetes.** Here we will check Azure Kubernetes Service (AKS) provided by Microsoft in detail.

Since this is managed service, there is cost associated with the same. Some basics to understand managed Kubernetes price in Azure (Azure Kubernetes Service). With help of below mentioned table we can easily understand what is chargeable in AKS cluster.

	Free	Charged	Optional
Kubernetes Control Plane	✓		
Billing Support	✓		
Basic Load Balancer	✓		
Node-Virtual Machines		✓	
Nodes – Disks		✓	
Bandwidth		✓	
Storage for persistent volumes			✓
Standard Load Balancer			✓
Public IP Addresses			✓
Log Analytics Workspace			✓

Let's see our solution for Micro services deployment with help of Managed Kubernetes i.e. **Azure Kubernetes Service (AKS).** Here we will develop and deploy .Net micro service in Azure Kubernetes service (AKS) and access it from internet. **[More focus is to explain K8s]. Below mentioned is the design of the proposed solution using AKS.**



Solution Architecture of K8sDemo Application

Solution Components	Description
Visual Studio community edition	K8sDemo micro service developed in .Net core. With Docker Support.
GitHub	Source code repository for the code
Azure Container Registry	Private registry service for building, storing, and managing container images.
Azure Kubernetes Service (AKS)	Azure Kubernetes Service (AKS) is a managed Kubernetes service with hardened security and fast delivery

3. Technical Details and Implementation of solution

Prerequisites:

- Visual Studio community edition
- Docker desktop
- Git on local machine to work with Github code
- Azure subscription to create Azure resources.
- Expectation to have basic understanding of Microservices /Docker /Containers /Azure

- Basic understanding of Powershell/.Net/Git

Solution implementation objectives:

- Develop micro service with help of .Net core
- Deployment Azure resources using available subscription
- Build Image and Push it in to Azure contained registry
- Create the YAML configuration file.
- Deploy Microservice in AKS cluster using YAML config and kubectl command.
- Scale up and down Microservice based on the requirement.
- Zero downtime deployment of Microservice in production environment.
- Release Azure resource after completion of the tests.

Note: After execution of the commands, we will see results in Azure portal. Though we can see it using console as well, I preferred to explain it with help of user interface in Azure for easy understanding. Best practice is to check it with commands to save time ☺

Step by step implementation of the solution.

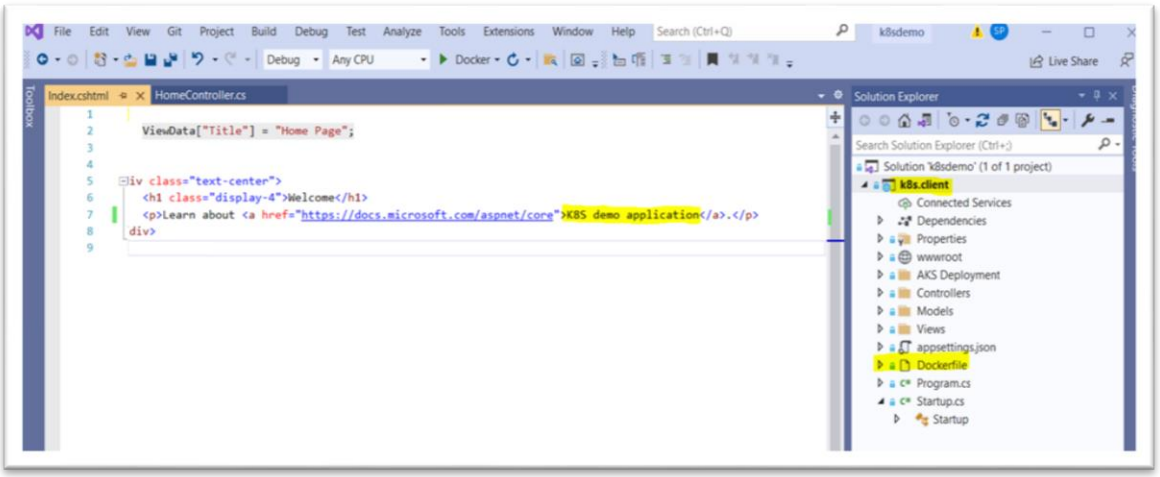
Step 1: Develop micro service with help of .Net core

Create simple .Net core web application with Docker support, which will display simple web page. In our example we will call it as Microservice.

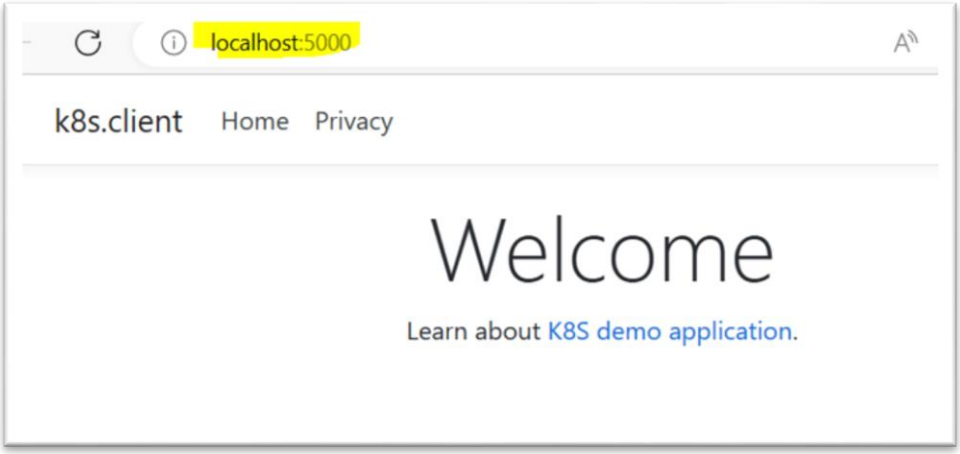
Ready code available here → Download

Or use git clone with link - https://github.com/akslearning99/k8sDemo.git

Once you download an application then open it in visual studio and execute it.



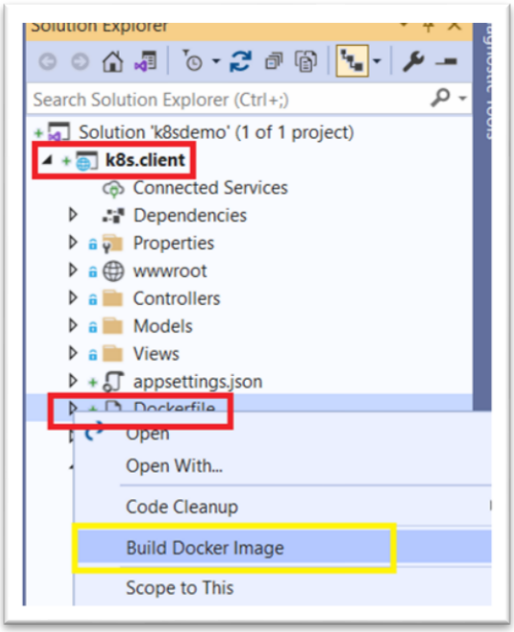
You can see below mentioned Welcome screen as result. It is simple demo app.



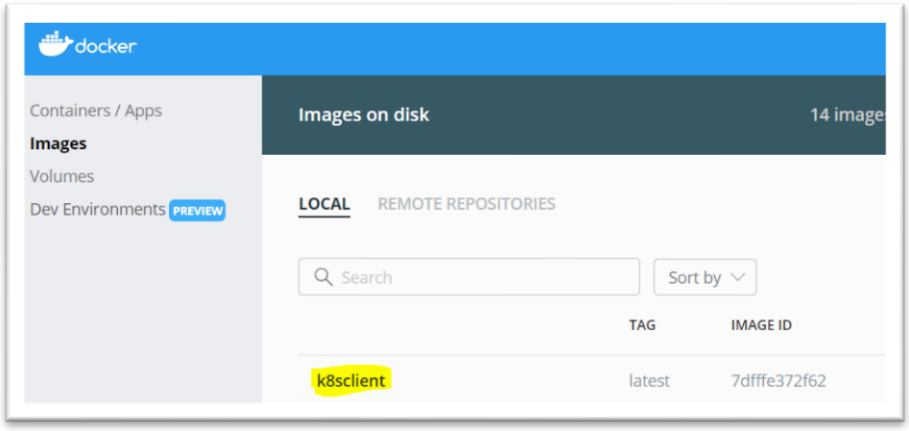
Step 2: Build and Create image of an application

Note: Please make sure that you have Docker desktop is installed on your machine.

To create image of an application. Right click on DockerFile in k8s.client project and click on “Build Docker Image”. Image will get created on your local machine.



Check image in Docker Desktop application.



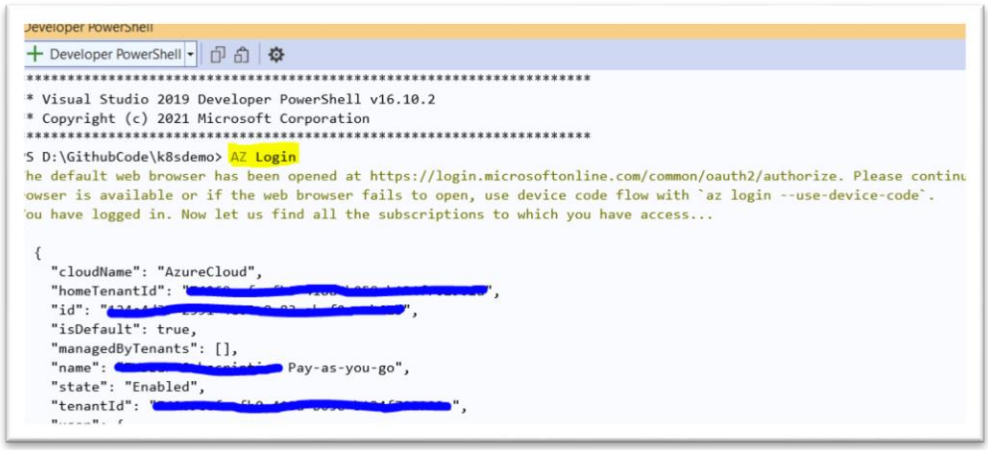
Alternative option is to create publish profile and push our image in Azure container registry as well.

Step 3: Login to Azure using command line (Power shell)

Please note, in this step we have to create resources in Azure cloud. Need valid subscription and login details to proceed further. [You can use Azure free subscription]

In visual studio open powershell terminal (View -> Terminal) and do login in Azure.

Use command “AZ Login”.



Step 4: Deploy and validate Azure resources

As described in solution architecture we need below mentioned public cloud resources in Azure to deploy Microservice in Kubernetes cluster.

Azure Container Registry	Private registry service for building, storing, and managing container images.
Azure Kubernetes Service (AKS)	Azure Kubernetes Service (AKS) is a managed Kubernetes service with hardened security and fast delivery

Execute commands one by one to create Azure resources.

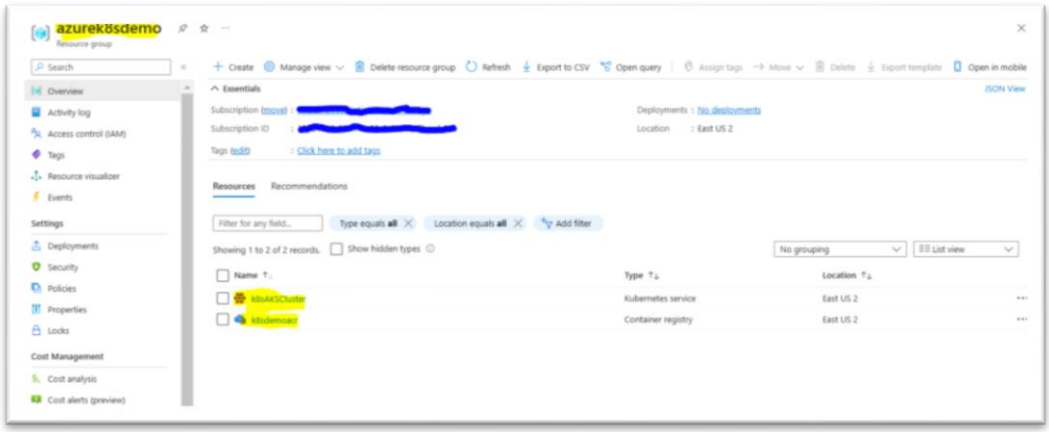
```
az group create --name azurek8sdemo --location EastUS2
az acr create --resource-group azurek8sdemo --name k8sdemoacr --sku Basic
az acr update -n k8sdemoacr --admin-enabled true
az aks create --resource-group azurek8sdemo --name k8sAKSCluster --node-count 1 --generate-ssh-keys --attach-acr k8sdemoacr
```

List of resource created with help commands. It will take around 10-12 minutes to create all resources.

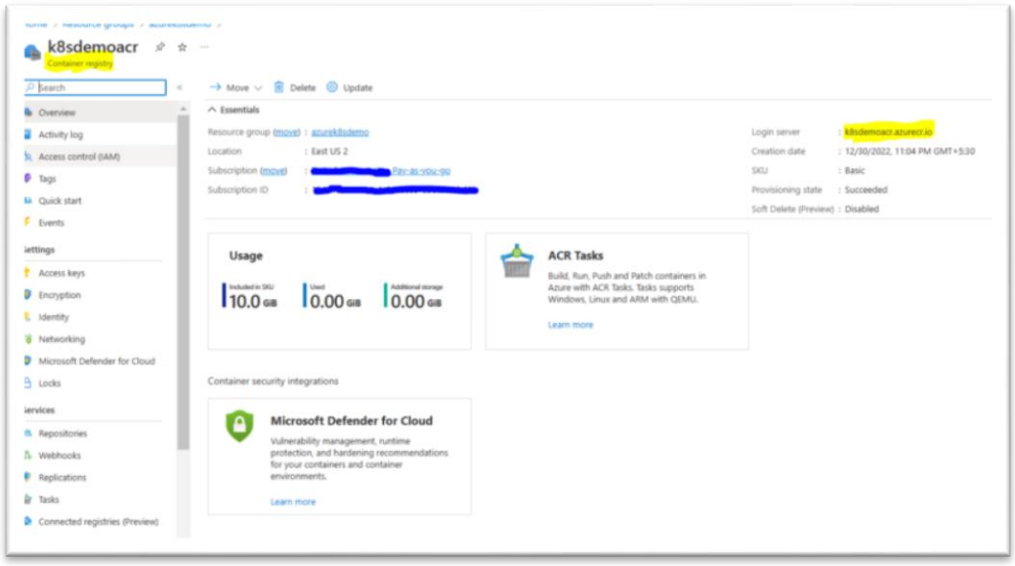
Resource Type	Resource Name
Resource Group	azurek8sdemo
Azure Container registry	k8sdemoacr
Azure Kubernetes Service (AKS)	k8sAKSCluster

Let's validate the resources in Azure portal. You can validate resources with help of command line as well.

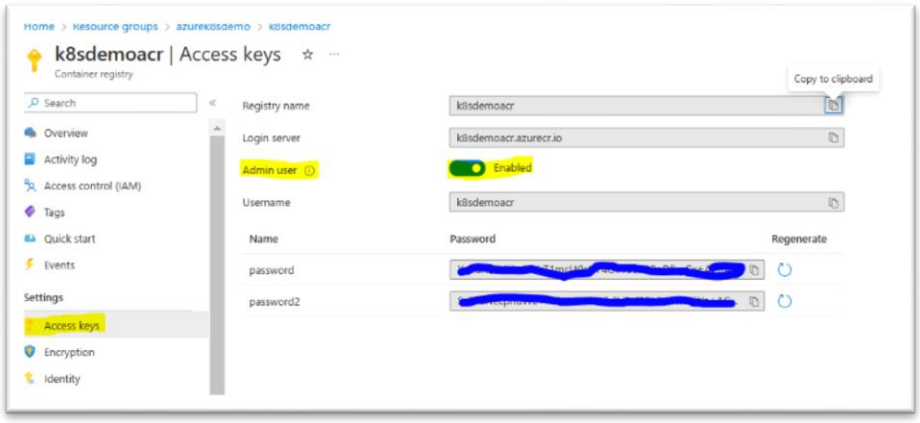
Resource group "azurek8sdemo" is created in EastUS2 region with two resources in it.



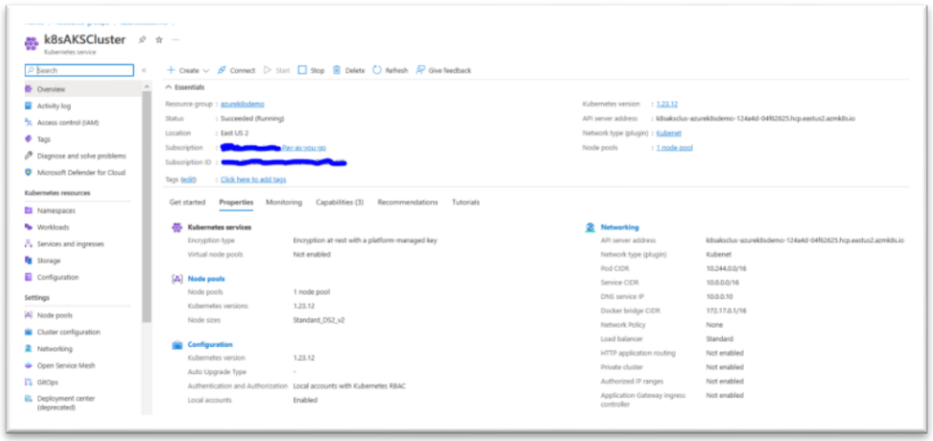
Azure container registry is created, in the resource group. This will be used to store the images. [Private container registry]



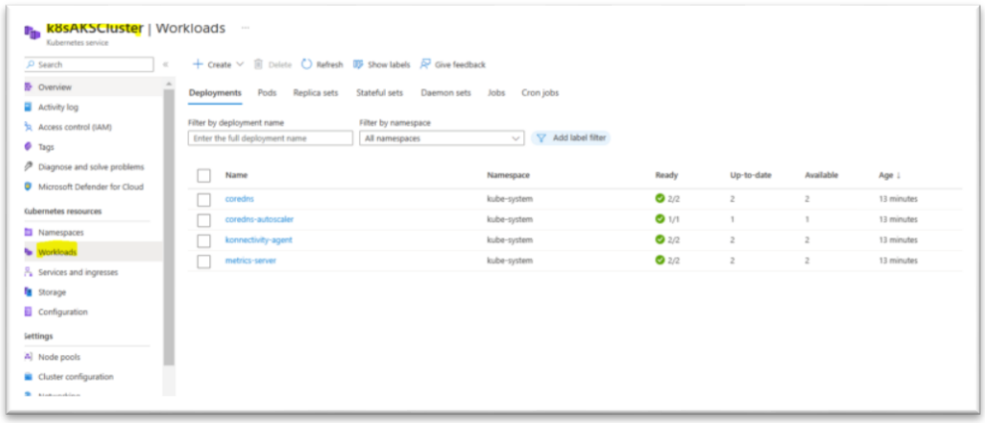
As mentioned in command “Admin user” is enabled for ACR. You can see Username and Password. Plan is to use these credentials in upcoming steps.



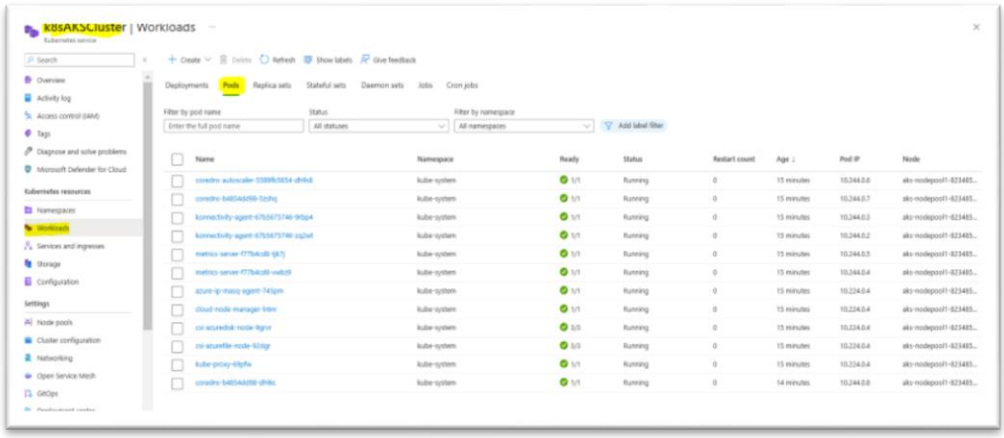
Azure Kubernetes service also called as AKS cluster is created (“k8sAKSCluster”).



Workload blade will show details about the application we deployed. By default there are few services which help to manage and maintain the cluster.



Workloads -> Pods section have detailed about the number of PODS deployed.



After deployment of our application, we can see added entries in POD's and services section.

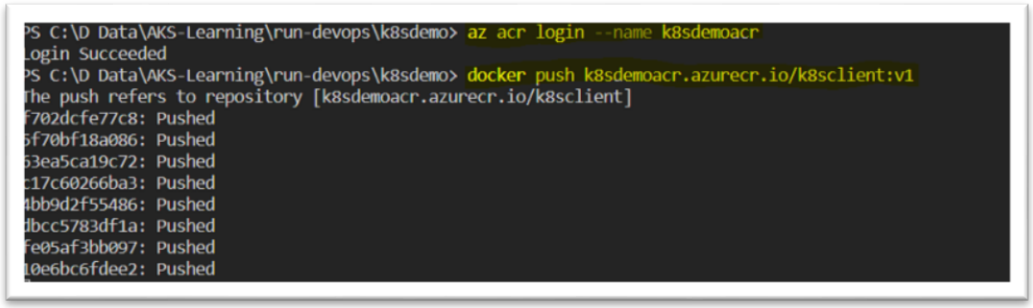
Step 5: Tag to local image and push in Azure container registry

Local image was created in Step2, let's push it in Azure container registry (Private registry) and maintain versions. Images stores in container registry can be used at the time of container creations in AKS cluster. We can manage versioning of the deployments we are doing using ACR.

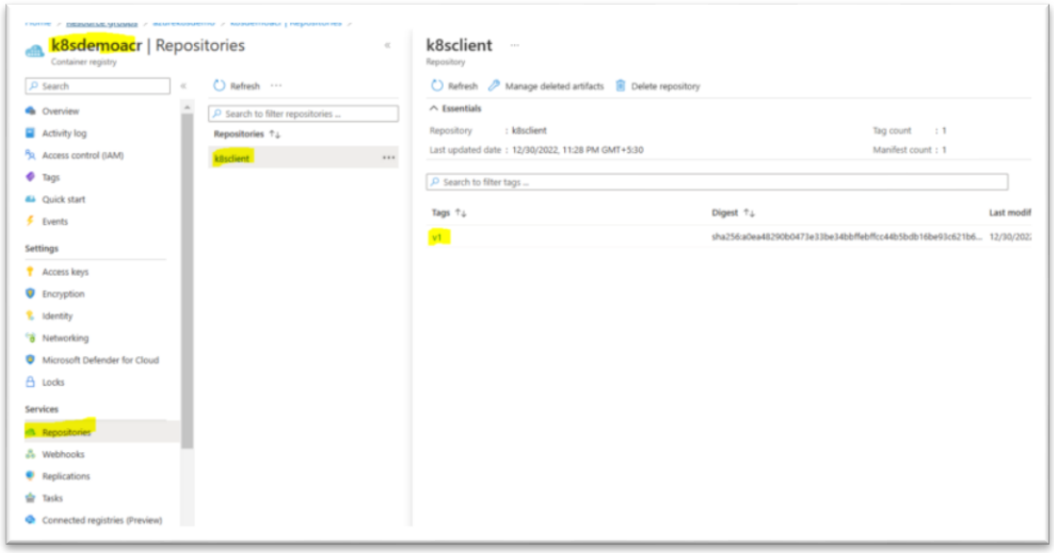
```
-- Tag local image with specific version.
docker tag k8sclient:latest k8sdemoacr.azurecr.io/k8sclient:v1

-- Login to Azure container registry...
az acr login --name k8sdemoacr

-- Push tagged image in Azure container registry...
```

```
docker push k8sdemoacr.azurecr.io/k8sclient:v1
```

Push command result is as below.



Validate image pushed in Azure Container registry [k8sdemoacr]. As we can see in screenshot below, image k8sclient:v1 is available in ACR repositories.



Step 6: Create YAML file for POD deployment in Azure Kubernetes service

Deployment of the container in is based on YAML config file. This is basically used for automated deployments.

Let's check **k8sclient.yaml** file in detail. You can find this file in downloaded source code as well.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8sclient-deployment
  labels:
    app: k8sclient
spec:
```

```
  replicas: 1
  selector:
    matchLabels:
      app: k8sclient
  template:
    metadata:
      labels:
        app: k8sclient
    spec:
      containers:
        - name: k8sclient
          image: k8sdemoacr.azurecr.io/k8sclient:v1
          ports:
            - containerPort: 80
          env:
            - name: ASPNETCORE_ENVIRONMENT
              value: Development
          resources:
            requests:
              memory: "64Mi"
              cpu: "0.1"
            limits:
              memory: "128Mi"
              cpu: "0.3"
      imagePullSecrets:
        - name: acr-secret
---
apiVersion: v1
kind: Service
metadata:
  name: k8sclient-service
spec:
  type: LoadBalancer
  selector:
    app: k8sclient
  ports:
    - protocol: TCP
      port: 80
```

This file has detailed information about what is configuration we need in order to execute application in container. E.g. Image Name, CPU and RAM, replicas etc.

Application name details

```
metadata:
  name: k8sclient-deployment
  labels:
    app: k8sclient
```

Number of instances of this container/POD required is mentioned in setting Replicas. Current it is configured as 1.

```
spec:
  replicas: 1
```

For containers need to specify image, it's version and port details from where specific application/service can be accessed. In our case,

Image: k8sclient:v1 image from Azure container registry (k8sdemoacr).

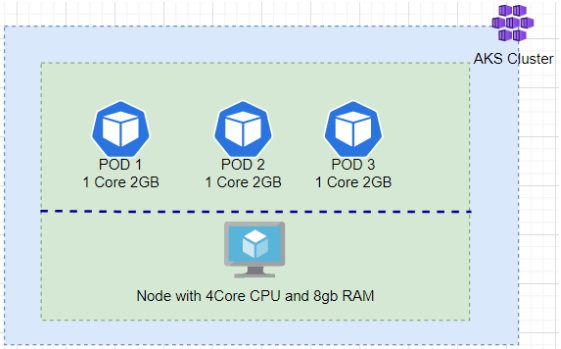
Container is available on port number 80.

```
containers:
        - name: k8sclient
          image: k8sdemoacr.azurecr.io/k8sclient:v1
          ports:
            - containerPort: 80
```

In resources section of container configured CPU and Memory required executing the container/POD. Also configured the MAX limit for resource utilization. It will help us to manage the resources in the cluster.

```
resources:
            requests:
              memory: "64Mi"
              cpu: "0.1"
            limits:
              memory: "128Mi"
              cpu: "0.3"
```

e.g. If you have node in cluster with 4 core CPU and 8gb RAM. We have container/pod which will need 1 core CPU and 2gb RAM for execution. How many pods we can create?



	CPU (core)	RAM (gb)
NODE 1	4	8

Case 1: deployment of 3 pods		
	CPU (core)	RAM (gb)
POD1	1	2
POD2	1	2
POD3	1	2
	3	**6**

In this case we created 3 containers/pods. In case we try to create another container/pod we might face insufficient CPU/memory issue. Because to create new container/pod we need minimum 1 core CPU and 2 GB RAM. If we do this then100% utilization of the node and in AKS cluster in order to manage cluster activities from every node some CPU and Memory is utilised.

We can define max limit as well. Single POD can use MAX 2CPU and 3 GB RAM etc. So Microsoft managed AKS will take care of creation of PODs based on configuration and available resources in the nodepool. If resource are available, it will create the container/pod otherwise it will not create because of insufficient resources.

Image is in Private registry. To pull image during deployment in cluster AKS need credentials to access to private registry. We need imagePullSecrets configuration. It allows AKS to pull the image from Azure container registry.

```
imagePullSecrets:
        - name: acr-secret
```

How to create image pull secret? In step 4 you can find ACR password. Use the same in below mentioned commands and create the secret.

```
-- Login to Azure container registry...
az acr login --name k8sdemoacr
-- Image pull secret...
kubectl create secret docker-registry acr-secret --docker-server=k8sdemoacr.azurecr.io --
docker-username=k8sdemoacr --docker-password=<<Password from ACR>> --docker-email=<<Your Azure
login>>
```

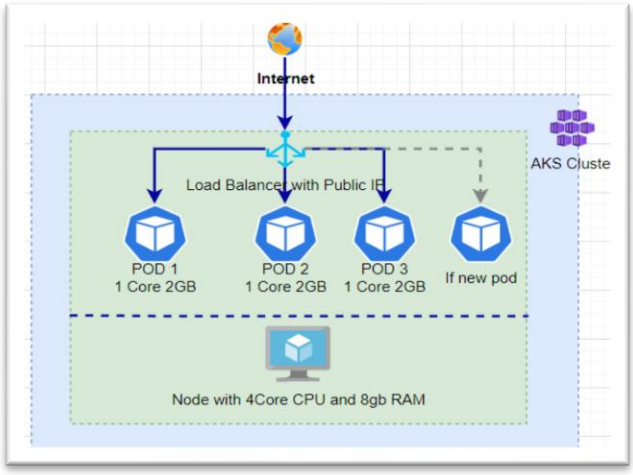
Last but not the least, how we are accessing an application deployed in AKS. Service endpoint is very important.

```
apiVersion: v1
kind: Service
metadata:
  name: k8sclient-service
```

```
spec:
  type: LoadBalancer
  selector:
    app: k8sclient
  ports:
    - protocol: TCP
      port: 80
```

There are two types: NodePort and LoadBalancer

Load Balancer type allows us to access the PODS from public network. It will help to share traffic load with multiple containers/pods.



If we add or remove any container/POD the load balancer will take care of traffic across the containers and using public IP address of the load balancer we can access the application/microservice deployed.

Step 7: Deploy YAML file in AKS using kubectl commands and validate in Azure portal

Note: Please make sure you have kubectl installed to execute commands. You can find installation steps here.

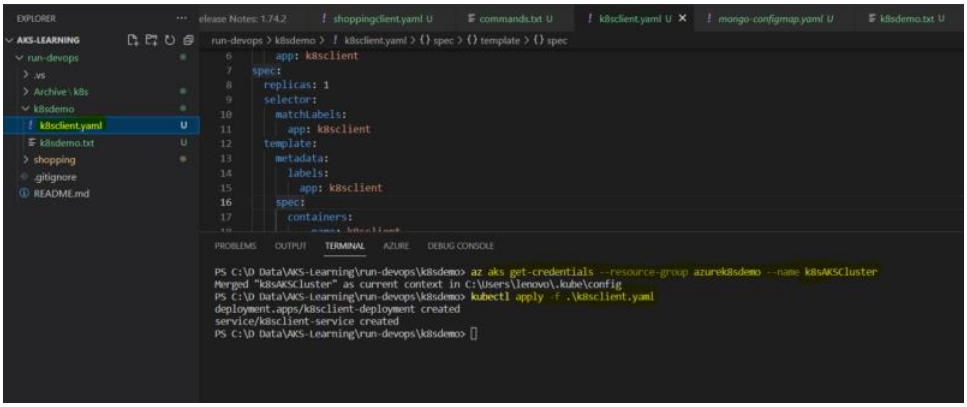
K8sclient.yaml file ready for the deployment of containers/pods. Connect to AKS cluster (k8sAKSCluster) we created in Azure [Refer Step 4].

```
-- Connect to AKS cluster...
az aks get-credentials --resource-group azurek8sdemo --name k8sAKSCluster
```

Select the location of k8sclient.yaml file and execute kubectl apply command.

```
-- deploy services...
kubectl apply -f .\k8sclient.yaml
```

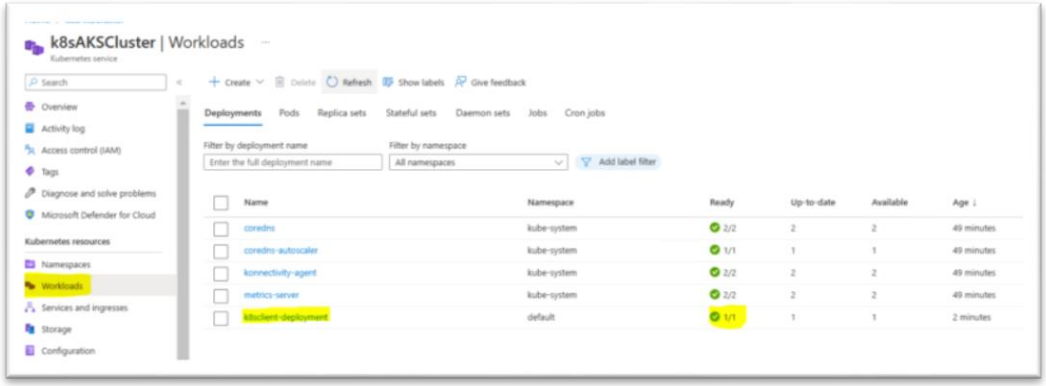
We are done. Containers are deployed in AKS cluster.

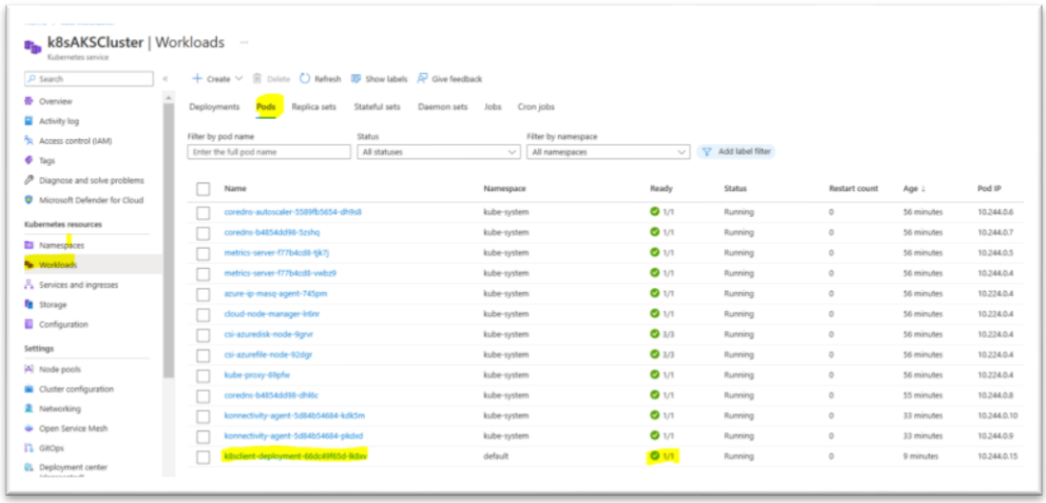


We can check PODS/Containers etc. using command line kubectl commands (Reference Kubectl commands) as well. In our case we will check it in Azure portal.

Let's open k8sAKSCluster in Azure portal and see the newly created POD and service.

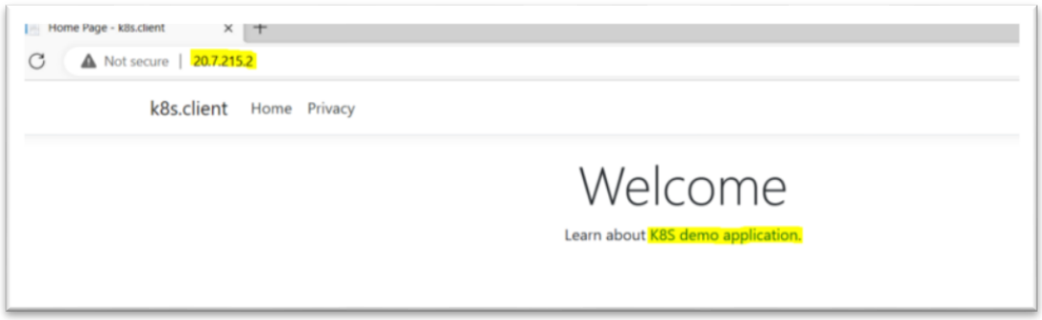
Based on k8sclient.yaml we can see one deployment in cluster. As we configured in Yaml file we have only one instance.





Service type mentioned as load balancer so we can see external IP.

Using this public IP address let's open our deployed containerized application in Managed AKS.

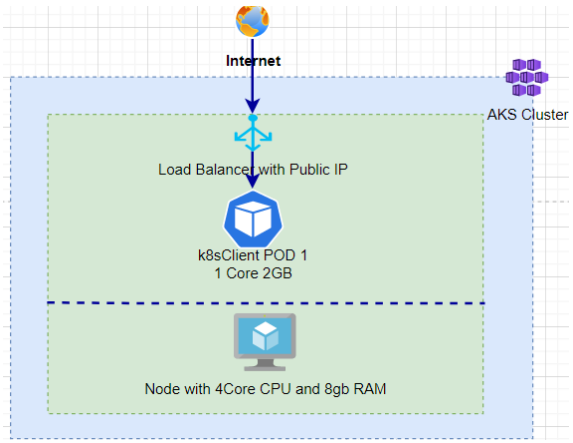


Our application packaged in image "k8sclient:v1" is available online ☺

Step 8: Horizontal scaling of pods and nodes in AKS cluster

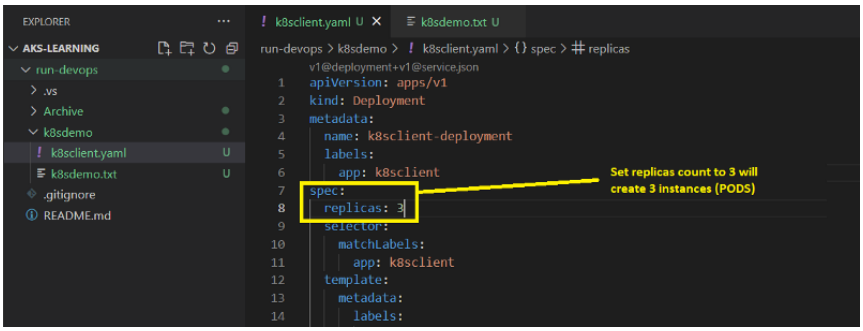
How to scale up application horizontally in case of business need?

Let's see our current deployment. We deployed application with one POD in AKS cluster. It is working well. It looks like below mentioned diagram.



If there is sudden increase in traffic and need multiple instances of deployed micro service/application. What to do in this case? How to increase number of POD instances?

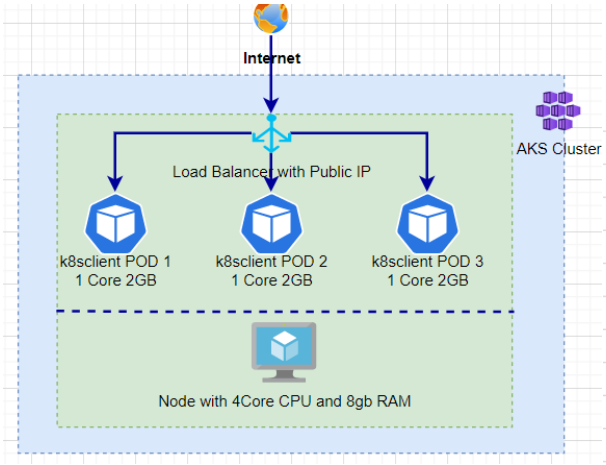
We can achieve it with help of only change in configuration of “Replicas” in k8sclient.yaml file.



Modify “Replicas: **3**” in the k8sclient.yaml and Apply it again in AKS Cluster.

```
PS C:\D Data\AKS-Learning\run-devops\k8sdemo> kubectl apply -f .\k8sclient.yaml
deployment.apps/k8sclient-deployment configured
service/k8sclient-service unchanged
PS C:\D Data\AKS-Learning\run-devops\k8sdemo>
```

As we can see only deployment is configured again and service is as it is. So now two new pods added in AKS cluster.

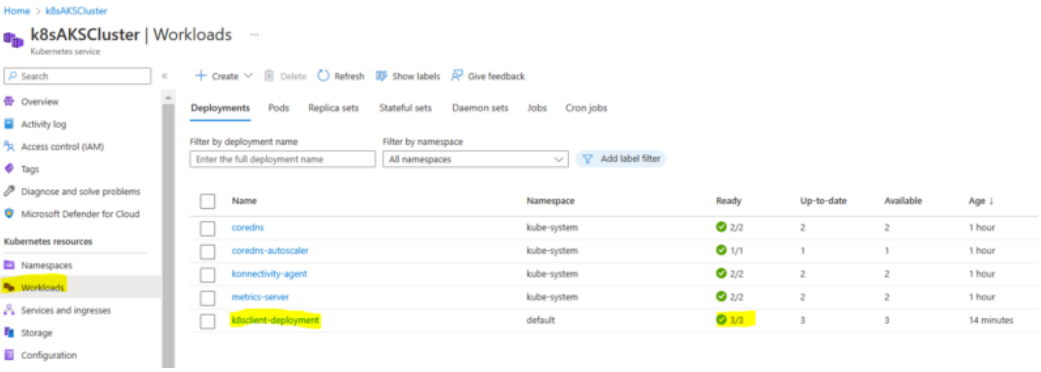


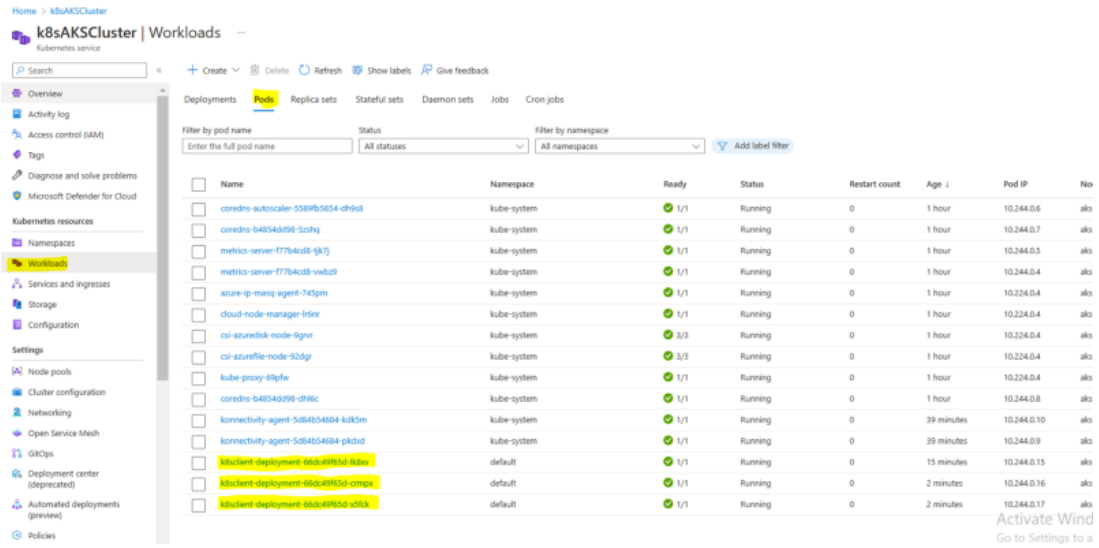
	CPU (core)	RAM (gb)
NODE 1	4	8

Case 1: deployment of 3 pods		
	CPU (core)	RAM (gb)
POD1	1	2
POD2	1	2
POD3	1	2
	3	**6**

Traffic will get distributed amongst the 3 different running POD’s.

See the result of applied configuration in AKS cluster.





Kubernetes support horizontal POD autoscaling to adjust the number of pods in deployment depending on CPU utilization or other metrics.

If number of POD's increased and we not have enough space to execute the POD? What to do in this case?

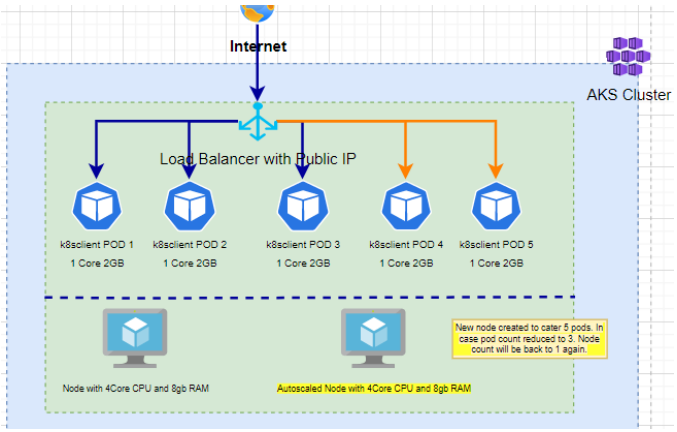
There is **cluster-autoscaler setting available in Azure Kubernetes service**. We have to specify min and max count of nodes.

e.g.

```
-- Cluster Autoscaler setting...
az aks update --resource-group azurek8sdemo --name k8sAKSCluster --
update-cluster-autoscaler --min-count 1 --max-count 3
```

In this command we updated setting with minimum 1 and maximum 3 nodes. So in scaling example if we updated "replicas = 5". It means requirement of 5 pods.

In case we not have enough processing power. With help of autocluster setting new node will be automatically created and new pods will get executed on the new node.



	CPU (core	RAM (gb)
NODE 1	4	8
NODE 2	4	8

Case 2: deployment of 5 pods			
	CPU (core	RAM (gb)	Node
POD1	1	2	NODE 1
POD2	1	2	NODE 1
POD3	1	2	NODE 1
POD4	1	2	NODE 2
POD5	1	2	NODE 2
	5	**10**	

Similarly in case we reduced the pods then nodes as well get reduced automatically. AKS will take care of the same.

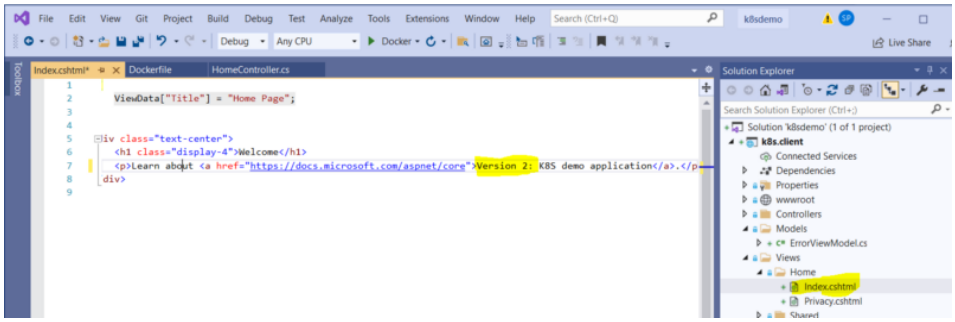
We can create and manage group of nodes using nodepool.

Step 9: Explanation of Zero downtime deployment of microservices

Application/Microservice is deployed in production AKS cluster. It is running in 3 different pods.

There is business situation now and we have code corrections and have to deploy it again on production environment with zero downtime. How to deploy Application/Microservices in production with Zero downtime?

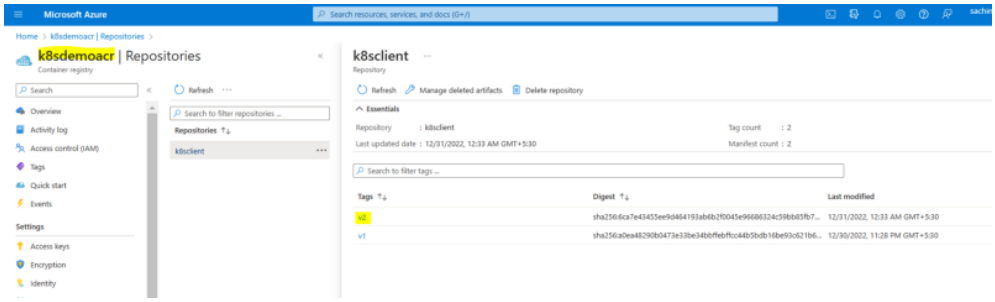
Let's correct the code and generate image.



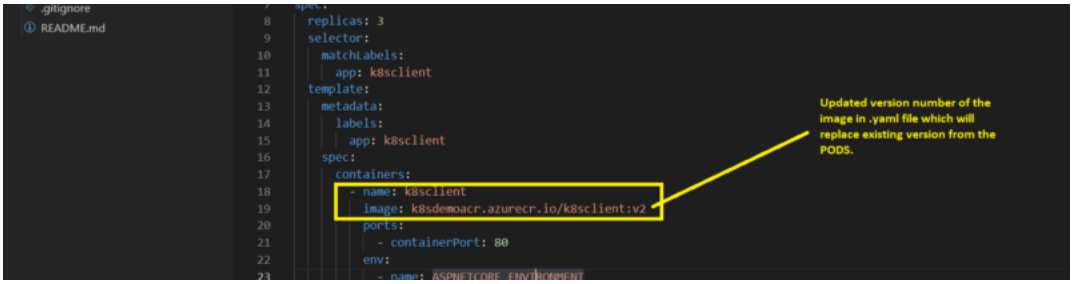
Tag and push image again in azure container registry with version v2.

```
PS C:\D Data\AKS-Learning\run-devops\k8sdemo> docker tag k8sclient:latest k8sdemoacr.azurecr.io/k8sclient:v2
PS C:\D Data\AKS-Learning\run-devops\k8sdemo> az acr login --name k8sdemoacr
Login Succeeded
PS C:\D Data\AKS-Learning\run-devops\k8sdemo> docker push k8sdemoacr.azurecr.io/k8sclient:v2
The push refers to repository [k8sdemoacr.azurecr.io/k8sclient]
f6ff5438e2b0: Pushed
5f70bf18a086: Layer already exists
63ea5ca19c72: Layer already exists
c17c60266ba3: Layer already exists
4bb9d2f55486: Layer already exists
dbcc5783df1a: Layer already exists
fe05af3bb097: Layer already exists
10e6bc6fdee2: Layer already exists
v2: digest: sha256:6ca7e43455ee9d464193ab6b2f0045e96686324c59bb85fb769b51b59a304494 size: 1995
PS C:\D Data\AKS-Learning\run-devops\k8sdemo>
```

In Azure container registry, we can see two different version of the image.



Update the image version in k8sclient.yaml file.



Apply the changes using kubectl command.

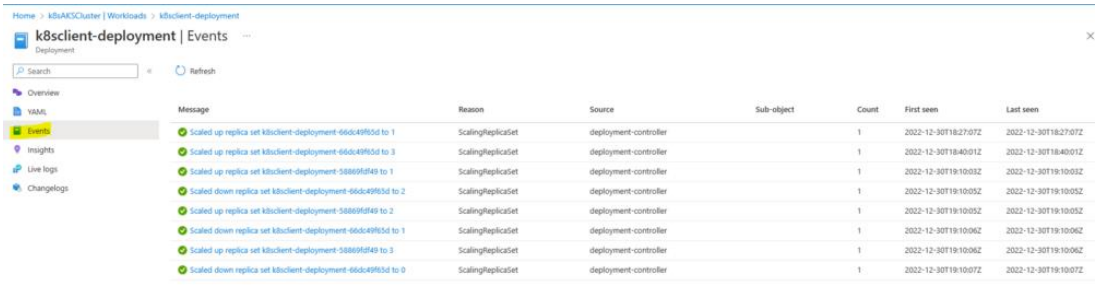
```
PS C:\D Data\AKS-Learning\run-devops\k8sdemo> kubectl apply -f .\k8sclient.yaml
deployment.apps/k8sclient-deployment configured
service/k8sclient-service unchanged
```

This step will terminate pods one by one and start new pods with updated version in .yaml file.

How it works? **[Note: In actual sequence of POD termination might be different since it is managed by AKS]**

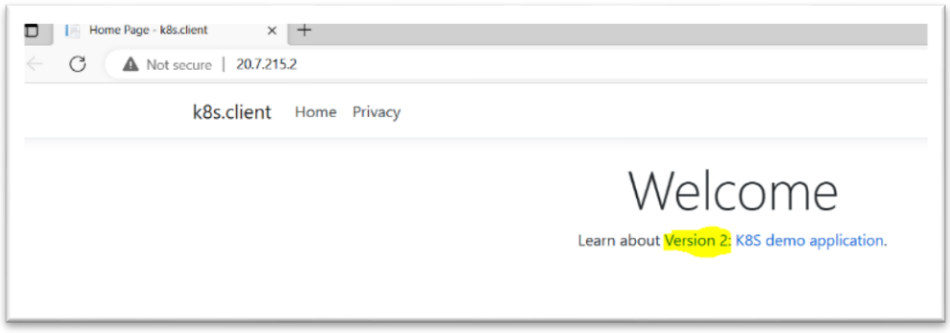
POD Name	App Version	Comment
POD 1	**v1**	Prod running
POD 2	**v1**	Prod running
POD 3	**v1**	Prod running
Started new deployment of version 2		
New POD1	v2	POD creation is in progress
New POD1	v2	POD created
POD 1	v1	POD terminated
New POD2	v2	POD creation is in progress
New POD2	v2	POD created
POD 2	v1	POD terminated
New POD3	v2	POD creation is in progress
New POD3	v2	POD created
POD 3	v1	POD terminated
After successful deployment of version 2		
New POD1	**v2**	Prod running
New POD2	**v2**	Prod running
New POD3	**v2**	Prod running

For reference you can check the events.



Kubernetes will internally take care of all the deployments. In this way we can achieve deployment with zero downtime on business critical environments.

Check newly deployed application using same external public IP address.



What we did so far is all manual execution of commands, we can do it with help of CICD pipelines in order to automate the deployment. So when you change the source code, image will get created automatically and application will get deployed in AKS cluster.

Step 10: Clean resources deployed in Azure environment

Final Step ☺. Clean up activity for the resources created for this demo application.

```
-- delete all resources
az group delete --name azurek8sdemo --yes --no-wait
```

4. Challenges in implementing the solution

Setup of Kubernetes manually is complex task and need experts. To be honest with help of Managed Kubernetes i.e. Azure Kubernetes service (AKS) it is quite simple. However if we are not aware about core concepts of Kubernetes sometime it is very difficult to do the deployments. e.g. we do the deployments and pods are not running etc. We need to use the logs and events information effectively and understand why specific pod was not created. Sometime image not pulled from acr to akscluster because of rights or all well with configuration but pod not created because of sufficient cpu and memory. Basically it is learning experience from application as well as infrastructure point of view.

5. Business Benefits

The Solution Architecture design in this blog can further extend to deploy multiple microservices in the AKS cluster. For every microservice we just have to create respective YAML configuration file. Using CICD pipeline we can automate the manual process of deployment. This will allow faster end to end deployments in production with zero

dowtime. With cluster and pod autoscaling we can use resources effectively. AKS protects your business by enabling administrators to tailor access to Azure Active Directory (AD) and identity and group identities.

6. References

Azure Kubernetes Service (AKS) documentation | Microsoft Learn
https://kubernetes.io/
https://en.wikipedia.org/wiki/Kubernetes

Mr. SACHIN M. POWAR

Cloud Solution Architect (#LinkedinProfile)