

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.

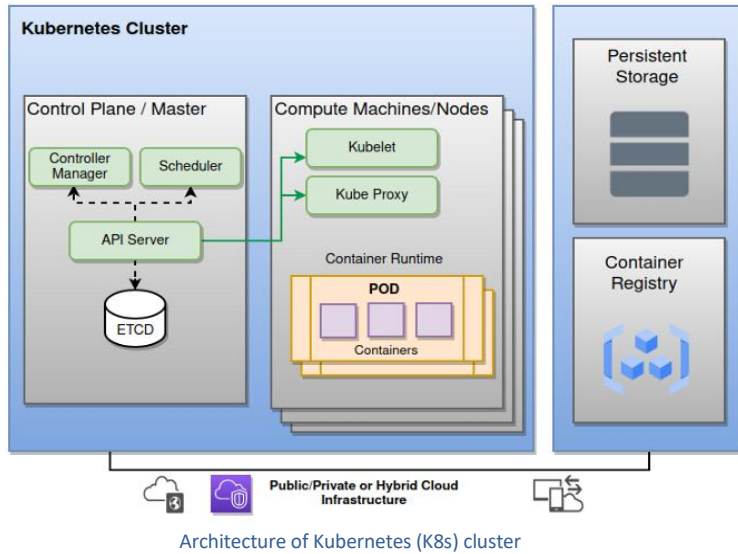
Solution Objectives

- **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.



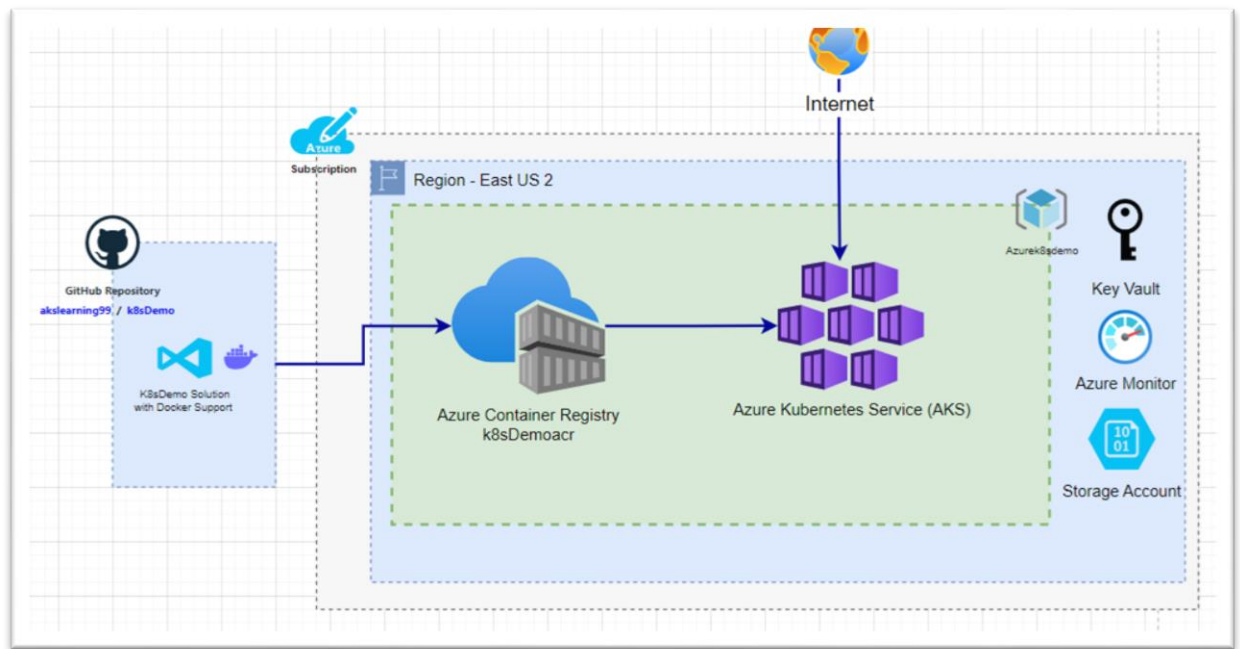
Although **Kubernetes is open source**, many **companies** planning to adopt Kubernetes **do not have the expertise or resources to set up and maintain the cluster** themselves.

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.

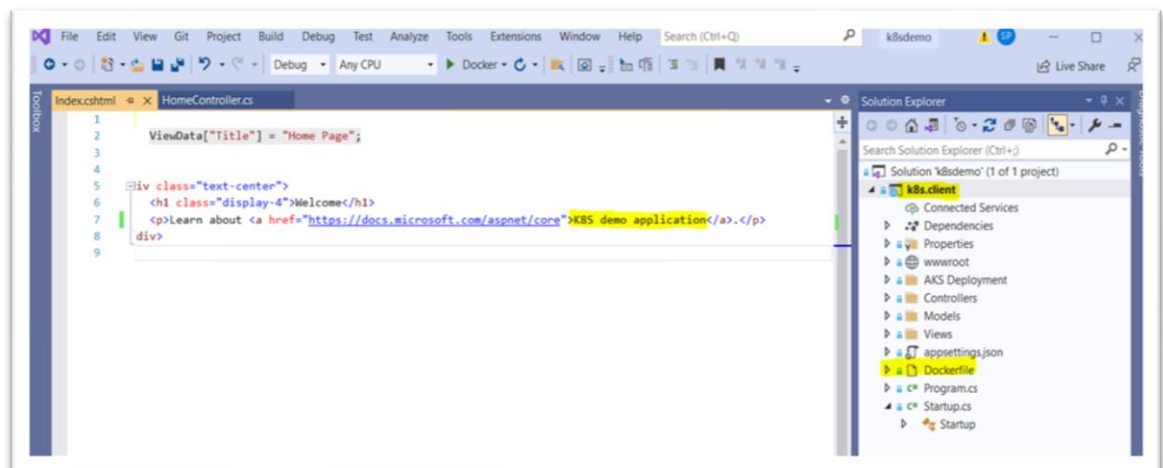
Step 1: Develop micro service with help of .Net core

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

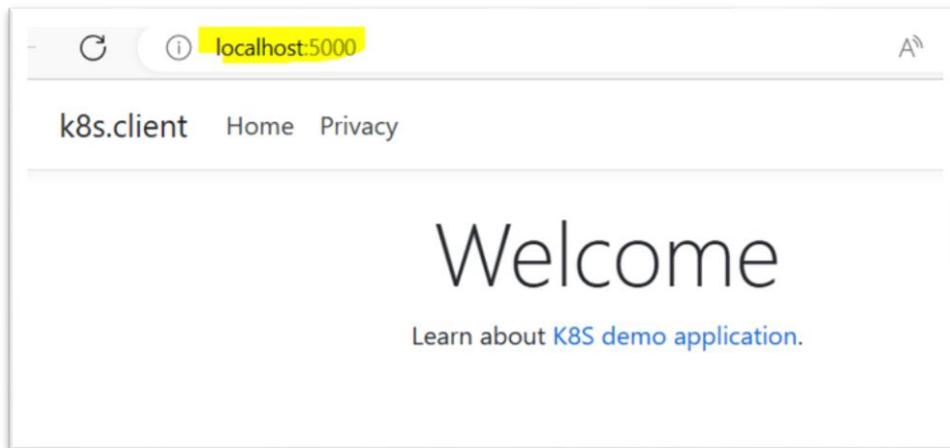
Ready code available here → [Download](#)

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.



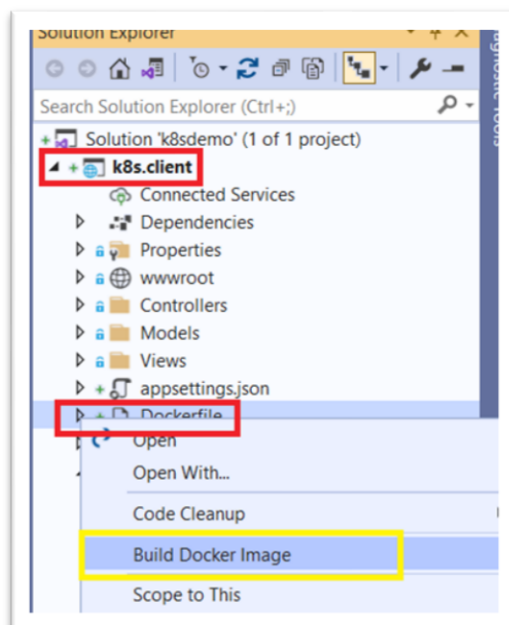
You can see below mentioned Welcome screen as result. It is simple demo app.



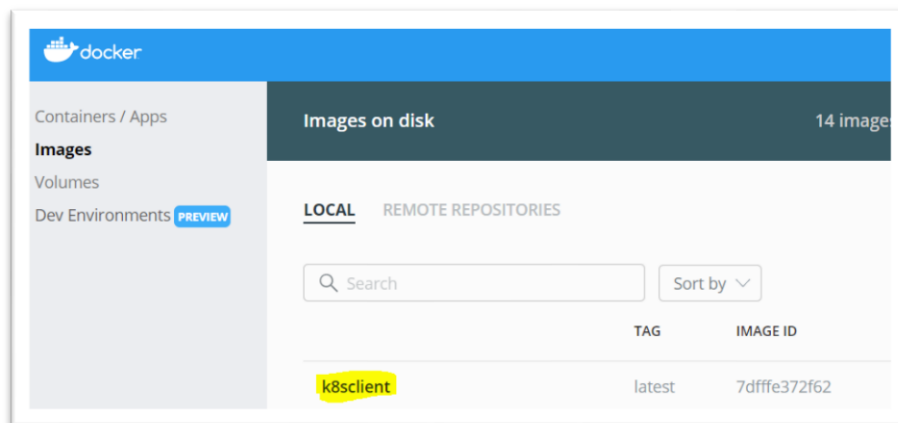
Step 2: Build and Create image of an application

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".

```
Developer PowerShell
+ Developer PowerShell | [?] [?] [?]
*****
* Visual Studio 2019 Developer PowerShell v16.10.2
* Copyright (c) 2021 Microsoft Corporation
*****
S D:\GithubCode\k8sdemo> AZ Login
The default web browser has been opened at https://login.microsoftonline.com/common/oauth2/authorize. Please continue in your browser. If you are behind a proxy, please ensure that the proxy server is available or if the web browser fails to open, use device code flow with `az login --use-device-code`.
You have logged in. Now let us find all the subscriptions to which you have access...

{
  "cloudName": "AzureCloud",
  "homeTenantId": "72f988bf-86f1-41af-b9bc-2e762f88260d",
  "id": "72f988bf-86f1-41af-b9bc-2e762f88260d",
  "isDefault": true,
  "managedByTenants": [],
  "name": "Pay-as-you-go",
  "state": "Enabled",
  "tenantId": "72f988bf-86f1-41af-b9bc-2e762f88260d",
  "....."
}
```

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.

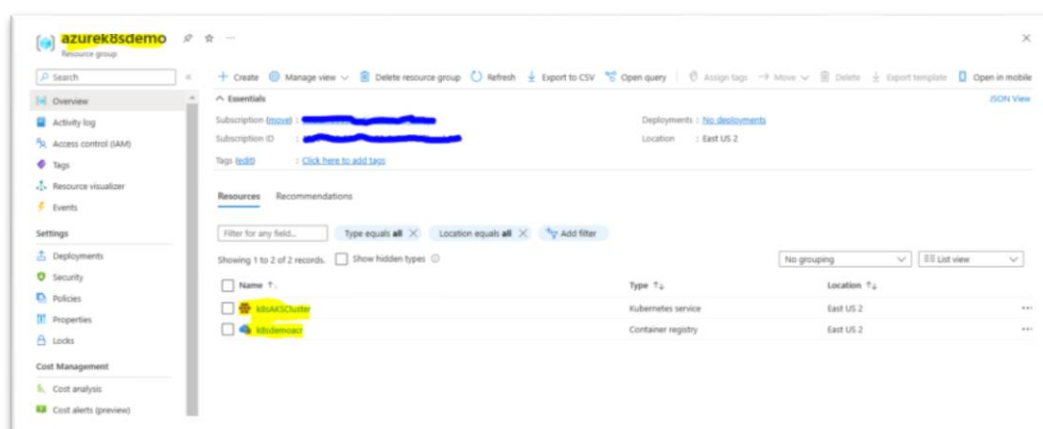
```
az group create --name azurek8sdemo --location EastUS2
az acr create --resource-group azurek8sdemo --name k8sdemoacr --sku Basic
az acr update -n k8sdemoacr --admin-enabled true
az aks create --resource-group azurek8sdemo --name k8sAKSCluster --node-count 1 --generate-ssh-keys --attach-acr k8sdemoacr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

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

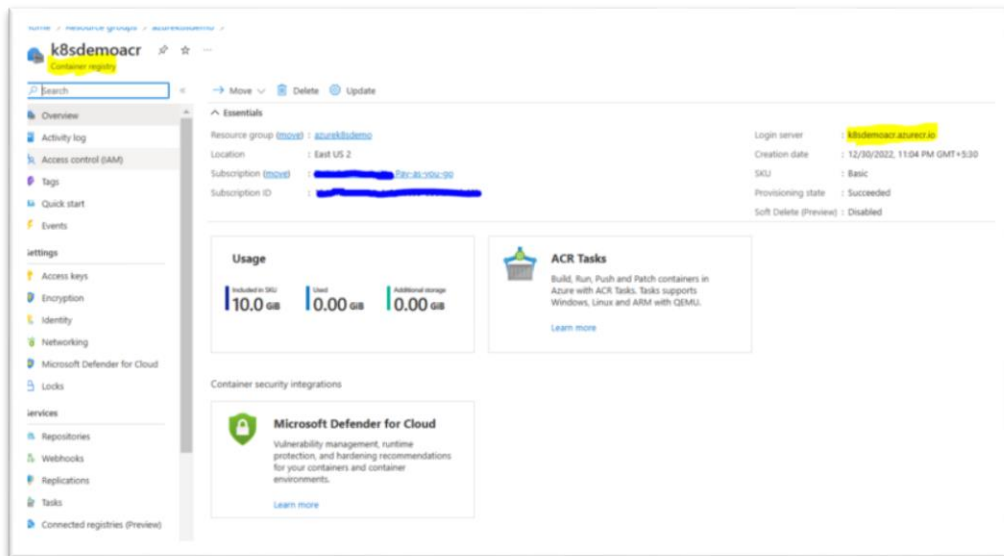
Resource Type	Resource Name
Resource Group	azurek8sdemo
Azure Container registry	k8sdemoacr
Azure Kubernetes Service (AKS)	k8sAKSCluster

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

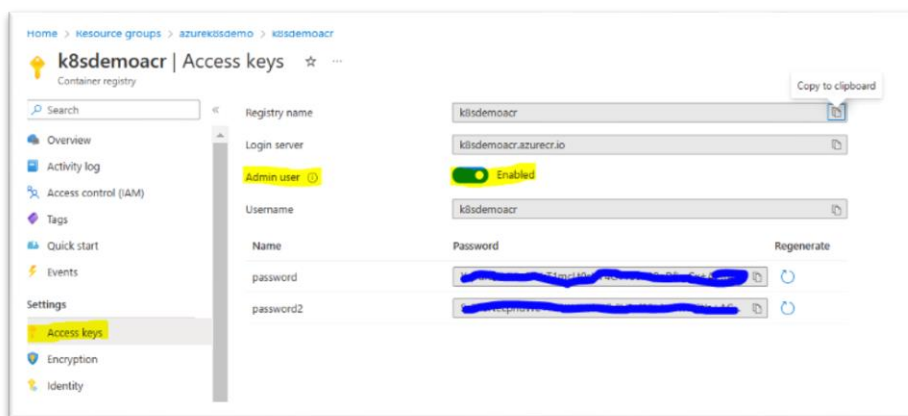
Resource group "azurek8sdemo" is created in EastUS2 region with two resources in it.



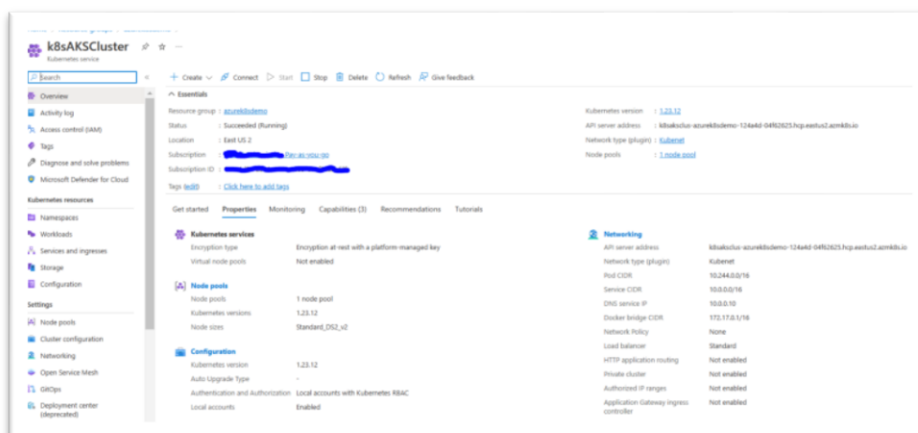
Azure container registry is created, in the resource group. This will be used to store the images. [Private container registry]



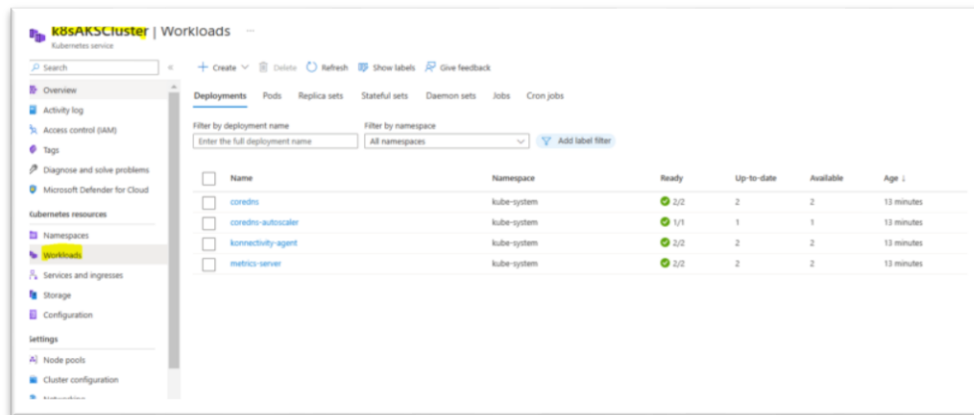
As mentioned in command “Admin user” is enabled for ACR. You can see Username and Password. Plan is to use these credentials in upcoming steps.



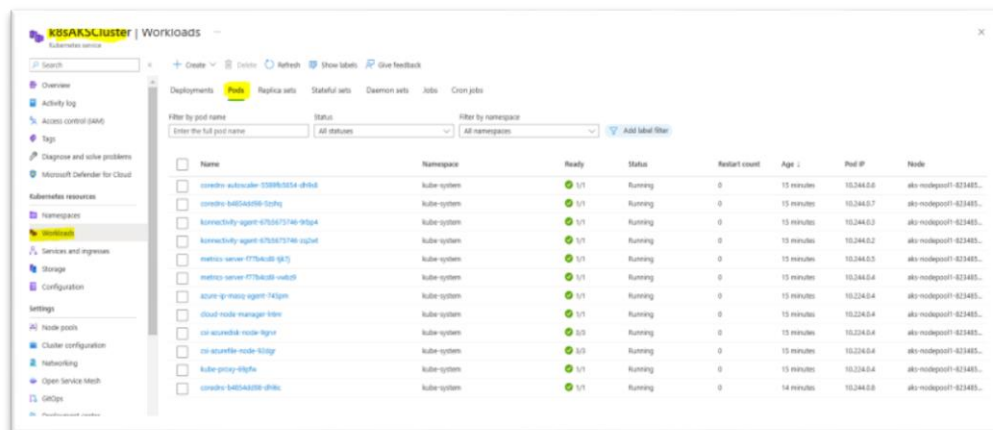
Azure Kubernetes service also called as AKS cluster is created (“k8sAKScluster”).



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

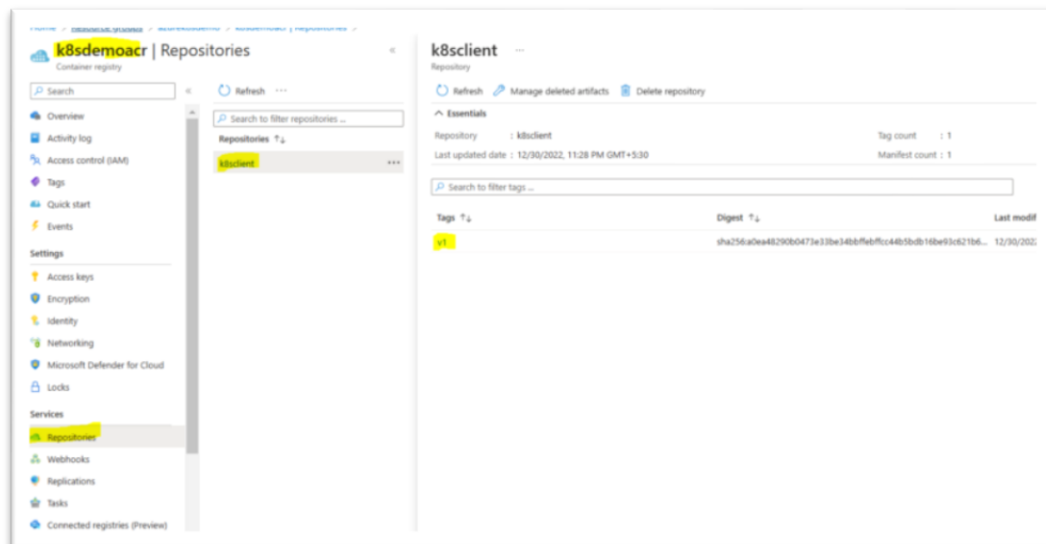
-- Push tagged image in Azure container registry...
```

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

Push command result is as below.

```
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:v1
The push refers to repository [k8sdemoacr.azurecr.io/k8sclient]
f702dcfe77c8: Pushed
5f70bf18a086: Pushed
63ea5ca19c72: Pushed
c17c60266ba3: Pushed
4bb9d2f55486: Pushed
dbcc5783df1a: Pushed
fe05af3bb097: Pushed
10e6bc6fdee2: Pushed
```

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.

```
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).

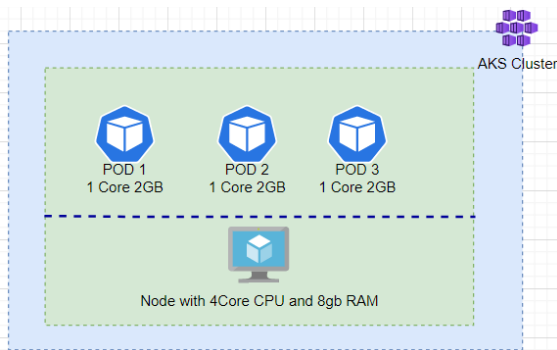
Container is available on port number 80.

```
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 then 100% 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>>
```

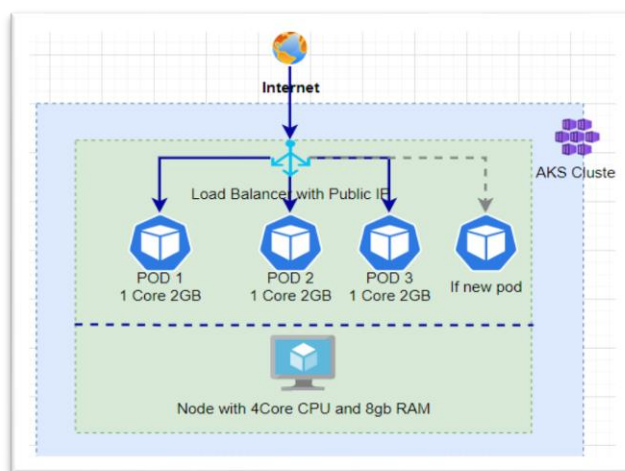
Last but not the least, how we are accessing an application deployed in AKS. Service endpoint is very important.

```
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
```

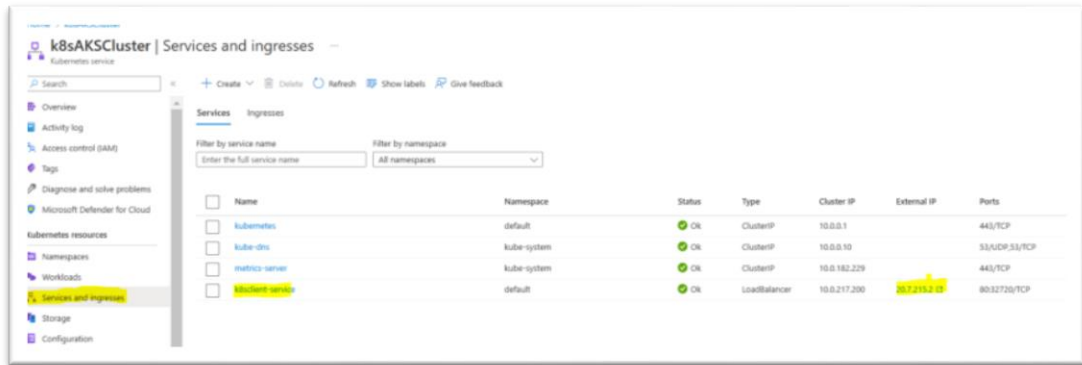
Select the location of k8sclient.yaml file and execute kubectl apply command.

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

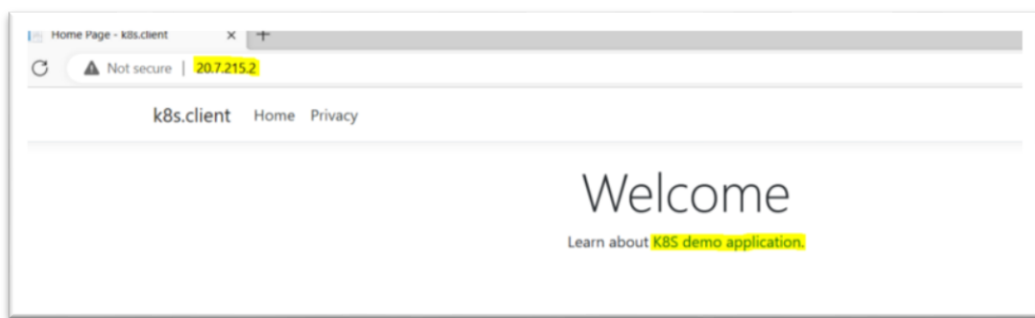
We are done. Containers are deployed in AKS cluster.

```

EXPLORER  ... elias Notes: 1.74.2  | shoppingclient.yaml U  | commands.txt U  | k8sclient.yaml U  | mongo-configmap.yaml U  | k8sdemo.txt U
AKS-LEARNING
├── run-devops
│   ├── .vs
│   ├── Archive - k8s
│   ├── k8sdemo
│   │   ├── k8sclient.yaml U
│   │   ├── k8sdemo.txt U
│   │   ├── shopping
│   │   ├── gitignore
│   │   └── README.md
└── ...
run-devops > k8sdemo > | k8sclient.yaml > | spec > | template > | spec
6  app: k8sclient
7  spec:
8    replicas: 1
9    selector:
10   matchLabels:
11     app: k8sclient
12   template:
13     metadata:
14       labels:
15         app: k8sclient
16     spec:
17       containers:
18         - name: k8sclient
19           image: k8sclient
20           ports:
21             - containerPort: 80
22           resources:
23             requests:
24               memory: 1Gi
25             limits:
26               memory: 1Gi
27           volumeMounts:
28             - name: mongo-configmap
29               mountPath: /etc/mongo
30           livenessProbe:
31             httpGet:
32               path: /
33             port: 80
34             initialDelaySeconds: 10
35             periodSeconds: 10
36           readinessProbe:
37             httpGet:
38               path: /
39             port: 80
40             initialDelaySeconds: 10
41             periodSeconds: 10
42           securityContext:
43             runAsUser: 1000
44             runAsGroup: 1000
45             fsGroup: 1000
46           volumeMounts:
47             - name: mongo-configmap
48               mountPath: /etc/mongo
49           livenessProbe:
50             httpGet:
51               path: /
52             port: 80
53             initialDelaySeconds: 10
54             periodSeconds: 10
55           readinessProbe:
56             httpGet:
57               path: /
58             port: 80
59             initialDelaySeconds: 10
60             periodSeconds: 10
61           securityContext:
62             runAsUser: 1000
63             runAsGroup: 1000
64             fsGroup: 1000
65           volumeMounts:
66             - name: mongo-configmap
67               mountPath: /etc/mongo
68           livenessProbe:
69             httpGet:
70               path: /
71             port: 80
72             initialDelaySeconds: 10
73             periodSeconds: 10
74           readinessProbe:
75             httpGet:
76               path: /
77             port: 80
78             initialDelaySeconds: 10
79             periodSeconds: 10
80           securityContext:
81             runAsUser: 1000
82             runAsGroup: 1000
83             fsGroup: 1000
84           volumeMounts:
85             - name: mongo-configmap
86               mountPath: /etc/mongo
87           livenessProbe:
88             httpGet:
89               path: /
90             port: 80
91             initialDelaySeconds: 10
92             periodSeconds: 10
93           readinessProbe:
94             httpGet:
95               path: /
96             port: 80
97             initialDelaySeconds: 10
98             periodSeconds: 10
99           securityContext:
100            runAsUser: 1000
101            runAsGroup: 1000
102            fsGroup: 1000
103            volumeMounts:
104              - name: mongo-configmap
105                mountPath: /etc/mongo
106            livenessProbe:
107              httpGet:
108                path: /
109                port: 80
110                initialDelaySeconds: 10
111                periodSeconds: 10
112              readinessProbe:
113                httpGet:
114                  path: /
115                  port: 80
116                  initialDelaySeconds: 10
117                  periodSeconds: 10
118                securityContext:
119                  runAsUser: 1000
120                  runAsGroup: 1000
121                  fsGroup: 1000
122                volumeMounts:
123                  - name: mongo-configmap
124                    mountPath: /etc/mongo
125                livenessProbe:
126                  httpGet:
127                    path: /
128                    port: 80
129                    initialDelaySeconds: 10
130                    periodSeconds: 10
131                readinessProbe:
132                  httpGet:
133                    path: /
134                    port: 80
135                    initialDelaySeconds: 10
136                    periodSeconds: 10
137                securityContext:
138                  runAsUser: 1000
139                  runAsGroup: 1000
140                  fsGroup: 1000
141                volumeMounts:
142                  - name: mongo-configmap
143                    mountPath: /etc/mongo
144                livenessProbe:
145                  httpGet:
146                    path: /
147                    port: 80
148                    initialDelaySeconds: 10
149                    periodSeconds: 10
150                readinessProbe:
151                  httpGet:
152                    path: /
153                    port: 80
154                    initialDelaySeconds: 10
155                    periodSeconds: 10
156                securityContext:
157                  runAsUser: 1000
158                  runAsGroup: 1000
159                  fsGroup: 1000
160                volumeMounts:
161                  - name: mongo-configmap
162                    mountPath: /etc/mongo
163                livenessProbe:
164                  httpGet:
165                    path: /
166                    port: 80
167                    initialDelaySeconds: 10
168                    periodSeconds: 10
169                readinessProbe:
170                  httpGet:
171                    path: /
172                    port: 80
173                    initialDelaySeconds: 10
174                    periodSeconds: 10
175                securityContext:
176                  runAsUser: 1000
177                  runAsGroup: 1000
178                  fsGroup: 1000
179                volumeMounts:
180                  - name: mongo-configmap
181                    mountPath: /etc/mongo
182                livenessProbe:
183                  httpGet:
184                    path: /
185                    port: 80
186                    initialDelaySeconds: 10
187                    periodSeconds: 10
188                readinessProbe:
189                  httpGet:
190                    path: /
191                    port: 80
192                    initialDelaySeconds: 10
193                    periodSeconds: 10
194                securityContext:
195                  runAsUser: 1000
196                  runAsGroup: 1000
197                  fsGroup: 1000
198                volumeMounts:
199                  - name: mongo-configmap
200                    mountPath: /etc/mongo
201                livenessProbe:
202                  httpGet:
203                    path: /
204                    port: 80
205                    initialDelaySeconds: 10
206                    periodSeconds: 10
207                readinessProbe:
208                  httpGet:
209                    path: /
210                    port: 80
211                    initialDelaySeconds: 10
212                    periodSeconds: 10
213                securityContext:
214                  runAsUser: 1000
215                  runAsGroup: 1000
216                  fsGroup: 1000
217                volumeMounts:
218                  - name: mongo-configmap
219                    mountPath: /etc/mongo
220                livenessProbe:
221                  httpGet:
222                    path: /
223                    port: 80
224                    initialDelaySeconds: 10
225                    periodSeconds: 10
226                readinessProbe:
227                  httpGet:
228                    path: /
229                    port: 80
230                    initialDelaySeconds: 10
231                    periodSeconds: 10
232                securityContext:
233                  runAsUser: 1000
234                  runAsGroup: 1000
235                  fsGroup: 1000
236                volumeMounts:
237                  - name: mongo-configmap
238                    mountPath: /etc/mongo
239                livenessProbe:
240                  httpGet:
241                    path: /
242                    port: 80
243                    initialDelaySeconds: 10
244                    periodSeconds: 10
245                readinessProbe:
246                  httpGet:
247                    path: /
248                    port: 80
249                    initialDelaySeconds: 10
250                    periodSeconds: 10
251                securityContext:
252                  runAsUser: 1000
253                  runAsGroup: 1000
254                  fsGroup: 1000
255                volumeMounts:
256                  - name: mongo-configmap
257                    mountPath: /etc/mongo
258                livenessProbe:
259                  httpGet:
260                    path: /
261                    port: 80
262                    initialDelaySeconds: 10
263                    periodSeconds: 10
264                readinessProbe:
265                  httpGet:
266                    path: /
267                    port: 80
268                    initialDelaySeconds: 10
269                    periodSeconds: 10
270                securityContext:
271                  runAsUser: 1000
272                  runAsGroup: 1000
273                  fsGroup: 1000
274                volumeMounts:
275                  - name: mongo-configmap
276                    mountPath: /etc/mongo
277                livenessProbe:
278                  httpGet:
279                    path: /
280                    port: 80
281                    initialDelaySeconds: 10
282                    periodSeconds: 10
283                readinessProbe:
284                  httpGet:
285                    path: /
286                    port: 80
287                    initialDelaySeconds: 10
288                    periodSeconds: 10
289                securityContext:
290                  runAsUser: 1000
291                  runAsGroup: 1000
292                  fsGroup: 1000
293                volumeMounts:
294                  - name: mongo-configmap
295                    mountPath: /etc/mongo
296                livenessProbe:
297                  httpGet:
298                    path: /
299                    port: 80
300                    initialDelaySeconds: 10
301                    periodSeconds: 10
302                readinessProbe:
303                  httpGet:
304                    path: /
305                    port: 80
306                    initialDelaySeconds: 10
307                    periodSeconds: 10
308                securityContext:
309                  runAsUser: 1000
310                  runAsGroup: 1000
311                  fsGroup: 1000
312                volumeMounts:
313                  - name: mongo-configmap
314                    mountPath: /etc/mongo
315                livenessProbe:
316                  httpGet:
317                    path: /
318                    port: 80
319                    initialDelaySeconds: 10
320                    periodSeconds: 10
321                readinessProbe:
322                  httpGet:
323                    path: /
324                    port: 80
325                    initialDelaySeconds: 10
326                    periodSeconds: 10
327                securityContext:
328                  runAsUser: 1000
329                  runAsGroup: 1000
330                  fsGroup: 1000
331                volumeMounts:
332                  - name: mongo-configmap
333                    mountPath: /etc/mongo
334                livenessProbe:
335                  httpGet:
336                    path: /
337                    port: 80
338                    initialDelaySeconds: 10
339                    periodSeconds: 10
340                readinessProbe:
341                  httpGet:
342                    path: /
343                    port: 80
344                    initialDelaySeconds: 10
345                    periodSeconds: 10
346                securityContext:
347                  runAsUser: 1000
348                  runAsGroup: 1000
349                  fsGroup: 1000
350                volumeMounts:
351                  - name: mongo-configmap
352                    mountPath: /etc/mongo
353                livenessProbe:
354                  httpGet:
355                    path: /
356                    port: 80
357                    initialDelaySeconds: 10
358                    periodSeconds: 10
359                readinessProbe:
360                  httpGet:
361                    path: /
362                    port: 80
363                    initialDelaySeconds: 10
364                    periodSeconds: 10
365                securityContext:
366                  runAsUser: 1000
367                  runAsGroup: 1000
368                  fsGroup: 1000
369                volumeMounts:
370                  - name: mongo-configmap
371                    mountPath: /etc/mongo
372                livenessProbe:
373                  httpGet:
374                    path: /
375                    port: 80
376                    initialDelaySeconds: 10
377                    periodSeconds: 10
378                readinessProbe:
379                  httpGet:
380                    path: /
381                    port: 80
382                    initialDelaySeconds: 10
383                    periodSeconds: 10
384                securityContext:
385                  runAsUser: 1000
386                  runAsGroup: 1000
387                  fsGroup: 1000
388                volumeMounts:
389                  - name: mongo-configmap
390                    mountPath: /etc/mongo
391                livenessProbe:
392                  httpGet:
393                    path: /
394                    port: 80
395                    initialDelaySeconds: 10
396                    periodSeconds: 10
397                readinessProbe:
398                  httpGet:
399                    path: /
400                    port: 80
401                    initialDelaySeconds: 10
402                    periodSeconds: 10
403                securityContext:
404                  runAsUser: 1000
405                  runAsGroup: 1000
406                  fsGroup: 1000
407                volumeMounts:
408                  - name: mongo-configmap
409                    mountPath: /etc/mongo
410                livenessProbe:
411                  httpGet:
412                    path: /
413                    port: 80
414                    initialDelaySeconds: 10
415                    periodSeconds: 10
416                readinessProbe:
417                  httpGet:
418                    path: /
419                    port: 80
420                    initialDelaySeconds: 10
421                    periodSeconds: 10
422                securityContext:
423                  runAsUser: 1000
424                  runAsGroup: 1000
425                  fsGroup: 1000
426                volumeMounts:
427                  - name: mongo-configmap
428                    mountPath: /etc/mongo
429                livenessProbe:
430                  httpGet:
431                    path: /
432                    port: 80
433                    initialDelaySeconds: 10
434                    periodSeconds: 10
435                readinessProbe:
436                  httpGet:
437                    path: /
438                    port: 80
439                    initialDelaySeconds: 10
440                    periodSeconds: 10
441                securityContext:
442                  runAsUser: 1000
443                  runAsGroup: 1000
444                  fsGroup: 1000
445                volumeMounts:
446                  - name: mongo-configmap
447                    mountPath: /etc/mongo
448                livenessProbe:
449                  httpGet:
450                    path: /
451                    port: 80
452                    initialDelaySeconds: 10
453                    periodSeconds: 10
454                readinessProbe:
455                  httpGet:
456                    path: /
457                    port: 80
458                    initialDelaySeconds: 10
459                    periodSeconds: 10
460                securityContext:
461                  runAsUser: 1000
462                  runAsGroup: 1000
463                  fsGroup: 1000
464                volumeMounts:
465                  - name: mongo-configmap
466                    mountPath: /etc/mongo
467                livenessProbe:
468                  httpGet:
469                    path: /
470                    port: 80
471                    initialDelaySeconds: 10
472                    periodSeconds: 10
473                readinessProbe:
474                  httpGet:
475                    path: /
476                    port: 80
477                    initialDelaySeconds: 10
478                    periodSeconds: 10
479                securityContext:
480                  runAsUser: 1000
481                  runAsGroup: 1000
482                  fsGroup: 1000
483                volumeMounts:
484                  - name: mongo-configmap
485                    mountPath: /etc/mongo
486                livenessProbe:
487                  httpGet:
488                    path: /
489                    port: 80
490                    initialDelaySeconds: 10
491                    periodSeconds: 10
492                readinessProbe:
493                  httpGet:
494                    path: /
495                    port: 80
496                    initialDelaySeconds: 10
497                    periodSeconds: 10
498                securityContext:
499                  runAsUser: 1000
500                  runAsGroup: 1000
501                  fsGroup: 1000
502                volumeMounts:
503                  - name: mongo-configmap
504                    mountPath: /etc/mongo
505                livenessProbe:
506                  httpGet:
507                    path: /
508                    port: 80
509                    initialDelaySeconds: 10
510                    periodSeconds: 10
511                readinessProbe:
512                  httpGet:
513                    path: /
514                    port: 80
515                    initialDelaySeconds: 10
516                    periodSeconds: 10
517                securityContext:
518                  runAsUser: 1000
519                  runAsGroup: 1000
520                  fsGroup: 1000
521                volumeMounts:
522                  - name: mongo-configmap
523                    mountPath: /etc/mongo
524                livenessProbe:
525                  httpGet:
526                    path: /
527                    port: 80
528                    initialDelaySeconds: 10
529                    periodSeconds: 10
530                readinessProbe:
531                  httpGet:
532                    path: /
533                    port: 80
534                    initialDelaySeconds: 10
535                    periodSeconds: 10
536                securityContext:
537                  runAsUser: 1000
538                  runAsGroup: 1000
539                  fsGroup: 1000
540                volumeMounts:
541                  - name: mongo-configmap
542                    mountPath: /etc/mongo
543                livenessProbe:
544                  httpGet:
545                    path: /
546                    port: 80
547                    initialDelaySeconds: 10
548                    periodSeconds: 10
549                readinessProbe:
550                  httpGet:
551                    path: /
552                    port: 80
553                    initialDelaySeconds: 10
554                    periodSeconds: 10
555                securityContext:
556                  runAsUser: 1000
557                  runAsGroup: 1000
558                  fsGroup: 1000
559                volumeMounts:
560                  - name: mongo-configmap
561                    mountPath: /etc/mongo
562                livenessProbe:
563                  httpGet:
564                    path: /
565                    port: 80
566                    initialDelaySeconds: 10
567                    periodSeconds: 10
568                readinessProbe:
569                  httpGet:
570                    path: /
571                    port: 80
572                    initialDelaySeconds: 10
573                    periodSeconds: 10
574                securityContext:
575                  runAsUser: 1000
576                  runAsGroup: 1000
577                  fsGroup: 1000
578                volumeMounts:
579                  - name: mongo-configmap
580                    mountPath: /etc/mongo
581                livenessProbe:
582                  httpGet:
583                    path: /
584                    port: 80
585                    initialDelaySeconds: 10
586                    periodSeconds: 10
587                readinessProbe:
588                  httpGet:
589                    path: /
590                    port: 80
591                    initialDelaySeconds: 10
592                    periodSeconds: 10
593                securityContext:
594                  runAsUser: 1000
595                  runAsGroup: 1000
596                  fsGroup: 1000
597                volumeMounts:
598                  - name: mongo-configmap
599                    mountPath: /etc/mongo
600                livenessProbe:
601                  httpGet:
602                    path: /
603                    port: 80
604                    initialDelaySeconds: 10
605                    periodSeconds: 10
606                readinessProbe:
607                  httpGet:
608                    path: /
609                    port: 80
610                    initialDelaySeconds: 10
611                    periodSeconds: 10
612                securityContext:
613                  runAsUser: 1000
614                  runAsGroup: 1000
615                  fsGroup: 1000
616                volumeMounts:
617                  - name: mongo-configmap
618                    mountPath: /etc/mongo
619                livenessProbe:
620                  httpGet:
621                    path: /
622                    port: 80
623                    initialDelaySeconds: 10
624                    periodSeconds: 10
625                readinessProbe:
626                  httpGet:
627                    path: /
628                    port: 80
629                    initialDelaySeconds: 10
630                    periodSeconds: 10
631                securityContext:
632                  runAsUser: 1000
633                  runAsGroup: 1000
634                  fsGroup: 1000
635                volumeMounts:
636                  - name: mongo-configmap
637                    mountPath: /etc/mongo
638                livenessProbe:
639                  httpGet:
640                    path: /
641                    port: 80
642                    initialDelaySeconds: 10
643                    periodSeconds: 10
644                readinessProbe:
645                  httpGet:
646                    path: /
647                    port: 80
648                    initialDelaySeconds: 10
649                    periodSeconds: 10
650                securityContext:
651                  runAsUser: 1000
652                  runAsGroup: 1000
653                  fsGroup: 1000
654                volumeMounts:
655                  - name: mongo-configmap
656                    mountPath: /etc/mongo
657                livenessProbe:
658                  httpGet:
659                    path: /
660                    port: 80
661                    initialDelaySeconds: 10
662                    periodSeconds: 10
663                readinessProbe:
664                  httpGet:
665                    path: /
666                    port: 80
667                    initialDelaySeconds: 10
668                    periodSeconds: 10
669                securityContext:
670                  runAsUser: 1000
671                  runAsGroup: 1000
672                  fsGroup: 1000
673                volumeMounts:
674                  - name: mongo-configmap
675                    mountPath: /etc/mongo
676                livenessProbe:
677                  httpGet:
678                    path: /
679                    port: 80
680                    initialDelaySeconds: 10
681                    periodSeconds: 10
682                readinessProbe:
683                  httpGet:
684                    path: /
685                    port: 80
686                    initialDelaySeconds: 10
687                    periodSeconds: 10
688                securityContext:
689                  runAsUser: 1000
690                  runAsGroup: 1000
691                  fsGroup: 1000
692                volumeMounts:
693                  - name: mongo-configmap
694                    mountPath: /etc/mongo
695                livenessProbe:
696                  httpGet:
697                    path: /
698                    port: 80
699                    initialDelaySeconds: 10
700                    periodSeconds: 10
701                readinessProbe:
702                  httpGet:
703                    path: /
704                    port: 80
705                    initialDelaySeconds: 10
706                    periodSeconds: 10
707                securityContext:
708                  runAsUser: 1000
709                  runAsGroup: 1000
710                  fsGroup: 1000
711                volumeMounts:
712                  - name: mongo-configmap
713                    mountPath: /etc/mongo
714                livenessProbe:
715                  httpGet:
716                    path: /
717                    port: 80
718                    initialDelaySeconds: 10
719                    periodSeconds: 10
720                readinessProbe:
721                  httpGet:
722                    path: /
723                    port: 80
724                    initialDelaySeconds: 10
725                    periodSeconds: 10
726                securityContext:
727                  runAsUser: 1000
728                  runAsGroup: 1000
729                  fsGroup: 1000
730                volumeMounts:
731                  - name: mongo-configmap
732                    mountPath: /etc/mongo
733                livenessProbe:
734                  httpGet:
735                    path: /
736                    port: 80
737                    initialDelaySeconds: 10
738                    periodSeconds: 10
739                readinessProbe:
740                  httpGet:
741                    path: /
742                    port: 80
743                    initialDelaySeconds: 10
744                    periodSeconds: 10
745                securityContext:
746                  runAsUser: 1000
747                  runAsGroup: 1000
748                  fsGroup: 1000
749                volumeMounts:
750                  - name: mongo-configmap
751                    mountPath: /etc/mongo
752                livenessProbe:
753                  httpGet:
754                    path: /
755                    port: 80
756                    initialDelaySeconds: 10
757                    periodSeconds: 10
758                readinessProbe:
759                  httpGet:
760                    path: /
761                    port: 80
762                    initialDelaySeconds: 10
763                    periodSeconds: 10
764                securityContext:
765                  runAsUser: 1000
766                  runAsGroup: 1000
767                  fsGroup: 1000
768                volumeMounts:
769                  - name: mongo-configmap
770                    mountPath: /etc/mongo
771                livenessProbe:
772                  httpGet:
773                    path: /
774                    port: 80
775                    initialDelaySeconds: 10
776                    periodSeconds: 10
777                readinessProbe:
778                  httpGet:
779                    path: /
780                    port: 80
781                    initialDelaySeconds: 10
782                    periodSeconds: 10
783                securityContext:
784                  runAsUser: 1000
785                  runAsGroup: 1000
786                  fsGroup: 1000
787                volumeMounts:
788                  - name: mongo-configmap
789                    mountPath: /etc/mongo
790                livenessProbe:
791                  httpGet:
792                    path: /
793                    port: 80
794                    initialDelaySeconds: 10
795                    periodSeconds: 10
796                readinessProbe:
797                  httpGet:
798                    path: /
799                    port: 80
800                    initialDelaySeconds: 10
801                    periodSeconds: 10
802                securityContext:
803                  runAsUser: 1000
804                  runAsGroup: 1000
805                  fsGroup: 1000
806                volumeMounts:
807                  - name: mongo-configmap
808                    mountPath: /etc/mongo
809                livenessProbe:
810                  httpGet:
811                    path: /
812                    port: 80
813                    initialDelaySeconds: 10
814                    periodSeconds: 10
815                readinessProbe:
816                  httpGet:
817                    path: /
818                    port: 80
819                    initialDelaySeconds: 10
820                    periodSeconds: 10
821                securityContext:
822                  runAsUser: 1000
823                  runAsGroup: 1000
824                  fsGroup: 1000
825                volumeMounts:
826                  - name: mongo-configmap
827                    mountPath: /etc/mongo
828                livenessProbe:
829                  httpGet:
830                    path: /
831                    port: 80
832                    initialDelaySeconds: 10
833                    periodSeconds: 10
834                readinessProbe:
835                  httpGet:
836                    path: /
837                    port: 80
838                    initialDelaySeconds: 10
839                    periodSeconds: 10
840                securityContext:
841                  runAsUser: 1000
842                  runAsGroup: 1000
843                  fsGroup: 1000
844                volumeMounts:
845                  - name: mongo-configmap
846                    mountPath: /etc/mongo
847                livenessProbe:
848                  httpGet:
849                    path: /
850                    port: 80
851                    initialDelaySeconds: 10
852                    periodSeconds: 10
853                readinessProbe:
854                  httpGet:
855                    path: /
856                    port: 80
857                    initialDelaySeconds: 10
858                    periodSeconds: 10
859                securityContext:
860                  runAsUser: 1000
861                  runAsGroup: 1000
862                  fsGroup: 1000
863                volumeMounts:
864                  - name: mongo-configmap
865                    mountPath: /etc/mongo
866                livenessProbe:
867                  httpGet:
868                    path: /
869                    port: 80
870                    initialDelaySeconds: 10
871                    periodSeconds: 10
872                readinessProbe:
873                  httpGet:
874                    path: /
875                    port: 80
876                    initialDelaySeconds: 10
877                    periodSeconds: 10
878                securityContext:
879                  runAsUser: 1000
880                  runAsGroup: 1000
881                  fsGroup: 1000
882                volumeMounts:
883                  - name: mongo-configmap
884                    mountPath: /etc/mongo
885                livenessProbe:
886                  httpGet:
887                    path: /
888                    port: 80
889                    initialDelaySeconds: 10
890                    periodSeconds: 10
891                readinessProbe:
892                  httpGet:
893                    path: /
894                    port: 80
895                    initialDelaySeconds: 10
896                    periodSeconds: 10
897                securityContext:
898                  runAsUser: 1000
899                  runAsGroup: 1000
900                  fsGroup: 1000
901                volumeMounts:
902                  - name: mongo-configmap
903                    mountPath: /etc/mongo
904                livenessProbe:
905                  httpGet:
906                    path: /
907                    port: 80
908                    initialDelaySeconds: 10
909                    periodSeconds: 10
910                readinessProbe:
911                  httpGet:
912                    path: /
913                    port: 80
914                    initialDelaySeconds: 10
915                    periodSeconds: 10
916                securityContext:
917                  runAsUser: 1000
918                  runAsGroup: 1000
919                  fsGroup: 1000
920                volumeMounts:
921                  - name: mongo-configmap
922                    mountPath: /etc/mongo
923                livenessProbe:
924                  httpGet:
925                    path: /
926                    port: 80
927                    initialDelaySeconds: 10
928                    periodSeconds: 10
929                readinessProbe:
930                  httpGet:
931                    path: /
932                    port: 80
933                    initialDelaySeconds: 10
934                    periodSeconds: 10
935                securityContext:
936                  runAsUser: 1000
937                  runAsGroup: 1000
938                  fsGroup: 1000
939                volumeMounts:
940                  - name: mongo-configmap
941                    mountPath: /etc/mongo
942                livenessProbe:
943                  httpGet:
944                    path: /
945                    port: 80
946                    initialDelaySeconds: 10
947                    periodSeconds: 10
948                readinessProbe:
949                  httpGet:
950                    path: /
951                    port: 80
952                    initialDelaySeconds: 10
953                    periodSeconds: 10
954                securityContext:
955                  runAsUser: 1000
956                  runAsGroup: 1000
957                  fsGroup: 1000
958                volumeMounts:
959                  - name: mongo-configmap
960                    mountPath: /etc/mongo
961                livenessProbe:
962                  httpGet:
963                    path: /
964                    port: 80
965                    initialDelaySeconds: 10
966                    periodSeconds: 10
967                readinessProbe:
968                  httpGet:
969                    path: /
970                    port: 80
971                    initialDelaySeconds: 10
972                    periodSeconds: 10
973                securityContext:
974                  runAsUser: 1000
975                  runAsGroup: 1000
976                  fsGroup: 1000
977                volumeMounts:
978                  - name: mongo-configmap
979                    mountPath: /etc/mongo
980                livenessProbe:
981                  httpGet:
982                    path: /
983                    port: 80
984                    initialDelaySeconds: 10
985                    periodSeconds: 10
986                readinessProbe:
987                  httpGet:
988                    path: /
989                    port: 80
990                    initialDelaySeconds: 10
991                    periodSeconds: 10
992                securityContext:
993                  runAsUser: 1000
994                  runAsGroup: 1000
995                  fsGroup: 1000
996                volumeMounts:
997                  - name: mongo-configmap
998                    mountPath: /etc/mongo
999                livenessProbe:
1000                 httpGet:
1001                   path: /
1002                   port: 80
1003                   initialDelaySeconds: 10
1004                   periodSeconds: 10
1005                 readinessProbe:
1006                   httpGet:
1007                     path: /
1008                     port: 80
1009                     initialDelaySeconds: 10
1010                     periodSeconds: 10
1011                 securityContext:
1012                   runAsUser: 1000
1013                   runAsGroup: 1000
1014                   fsGroup: 1000
1015                 volumeMounts:
1016                   - name: mongo-configmap
1017                     mountPath: /etc/mongo
1018                 livenessProbe:
1019                   httpGet:
1020                     path: /
1021                     port: 80
1022                     initialDelaySeconds: 10
1023                     periodSeconds: 10
1024                 readinessProbe:
1025                   httpGet:
1026                     path: /
1027                     port: 80
1028                     initialDelaySeconds: 10
1029                     periodSeconds: 10
1030                 securityContext:
1031                   runAsUser: 1000
1032                   runAsGroup: 1000
1033                   fsGroup: 1000
1034                 volumeMounts:
1035                   - name: mongo-configmap
1036                     mountPath: /etc/mongo
1037                 livenessProbe:
1038                   httpGet:
1039                     path: /
1040                     port: 80
1041                     initialDelaySeconds: 10
1042                     periodSeconds: 10
1043                 readinessProbe:
1044                   httpGet:
1045                     path: /
1046                     port: 80
1047                     initialDelaySeconds: 10
1048                     periodSeconds: 10
1049                 securityContext:
1050                   runAsUser: 1000
1051                   runAsGroup: 1000
1052                   fsGroup: 1000
1053                 volumeMounts:
1054                   - name: mongo-configmap
1055                     mountPath: /etc/mongo
1056                 livenessProbe:
1057                   httpGet:
1058                     path: /
1059                     port: 80
1060                     initialDelaySeconds: 10
1061                     periodSeconds: 10
1062                 readinessProbe:
1063                   httpGet:
1064                     path: /
1065                     port: 80
1066                     initialDelaySeconds: 10
1067                     periodSeconds: 10
1068                 securityContext:
1069                   runAsUser: 1000
1070                   runAsGroup: 1000
1071                   fsGroup: 1000
1072                 volumeMounts:
1073                   - name: mongo-configmap
1074                     mountPath: /etc/mongo
1075                 livenessProbe:
1076                   httpGet:
1077                     path: /
1078                     port: 80
1079                     initialDelaySeconds: 10
1080                     periodSeconds: 10
1081                 readinessProbe:
1082                   httpGet:
1083                     path: /
1084                     port: 80
1085                     initialDelaySeconds: 10
1086                     periodSeconds: 10
1087                 securityContext:
1088                   runAsUser: 1000
1089                   runAsGroup: 1000
1090                   fsGroup: 1000
1091                 volumeMounts:
1092                   - name: mongo-configmap
1093                     mountPath: /etc/mongo
1094                 livenessProbe:
1095                   httpGet:
1096                     path: /
1097                     port: 80
1098                     initialDelaySeconds: 10
1099                     periodSeconds: 10
1100                 readinessProbe:
1101                   httpGet:
1102                     path: /
1103                     port: 80
1104                     initialDelaySeconds: 10
1105                     periodSeconds: 10
1106                 securityContext:
1107                   runAsUser: 1000
1108                   runAsGroup: 1000
1109                   fsGroup: 1000
1110                 volumeMounts:
1111                   - name: mongo-configmap
1112                     mountPath: /etc/mongo
1113                 livenessProbe:
1114                   httpGet:
1115                     path: /
1116                     port: 80
1117                     initialDelaySeconds: 10
1118                     periodSeconds: 10
1119                 readinessProbe:
1120                   httpGet:
1121                     path: /
1122                     port: 80
1123                     initialDelaySeconds: 10
1124                     periodSeconds: 10
1125                 securityContext:
1126                   runAsUser: 1000
1127                   runAsGroup: 1000
1128                   fsGroup: 1000
1129                 volumeMounts:
1130                   - name: mongo-configmap
1131                     mountPath: /etc/mongo
1132                 livenessProbe:
1133                   httpGet:
1134                     path: /
1135                     port: 80
1136                     initialDelaySeconds: 10
1137                     periodSeconds: 10
1138                 readinessProbe:
1139                   httpGet:
1140                     path: /
1141                     port: 80
1142                     initialDelaySeconds: 10
1143                     periodSeconds: 10
1144                 securityContext:
1145                   runAsUser: 1000
1146                   runAsGroup: 1000
1147                   fsGroup: 1000
1148                 volumeMounts:
1149                   - name: mongo-configmap
1150                     mountPath: /etc/mongo
1151                 livenessProbe:
1152                   httpGet:
1153                     path: /
1154                     port: 80
1155                     initialDelaySeconds: 10
1156                     periodSeconds: 10
1157                 readinessProbe:
1158                   httpGet:
1159                     path: /
1160                     port: 80
1161                     initialDelaySeconds: 10
1162                     periodSeconds: 10
1163                 securityContext:
1164                   runAsUser: 1000
1165                   runAsGroup: 1000
1166                   fsGroup: 1000
1167                 volumeMounts:
1168                   - name: mongo-configmap
1169                     mountPath: /etc/mongo
1170                 livenessProbe:
1171                   httpGet:
1172                     path: /
1173                     port: 80
1174                     initialDelaySeconds: 10
1175                     periodSeconds: 10
1176                 readinessProbe:
1177                   httpGet:
1178                     path: /
1179                     port: 80
1180                     initialDelaySeconds: 10
1181                     periodSeconds: 10
```



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

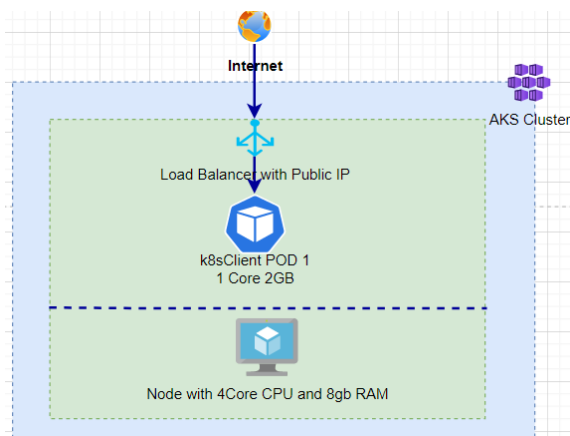


Our application packaged in image "k8sclient:v1" is available online 😊

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.

```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: k8sclient-deployment
5  labels:
6    app: k8sclient
7  spec:
8    replicas: 3
9  selector:
10   matchLabels:
11     app: k8sclient
12  template:
13   metadata:
14     labels:

```

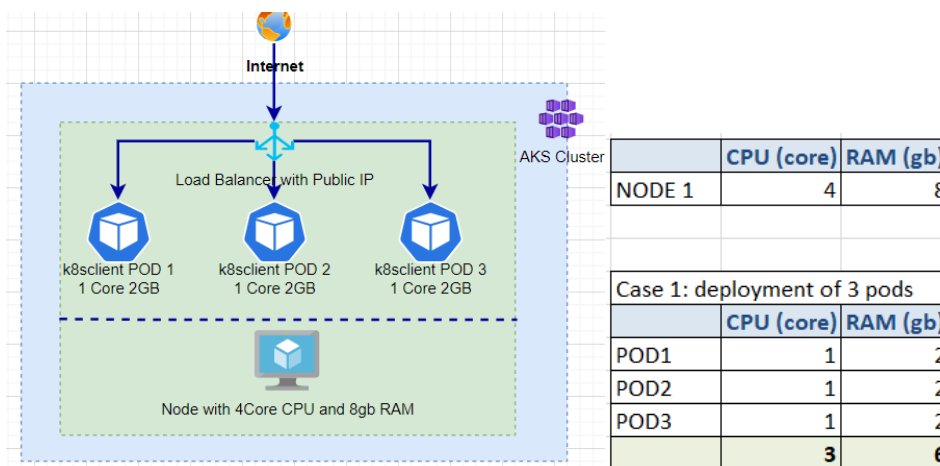
Modify “Replicas: 3” in the k8sclient.yaml and Apply it again in AKS Cluster.

```

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>

```

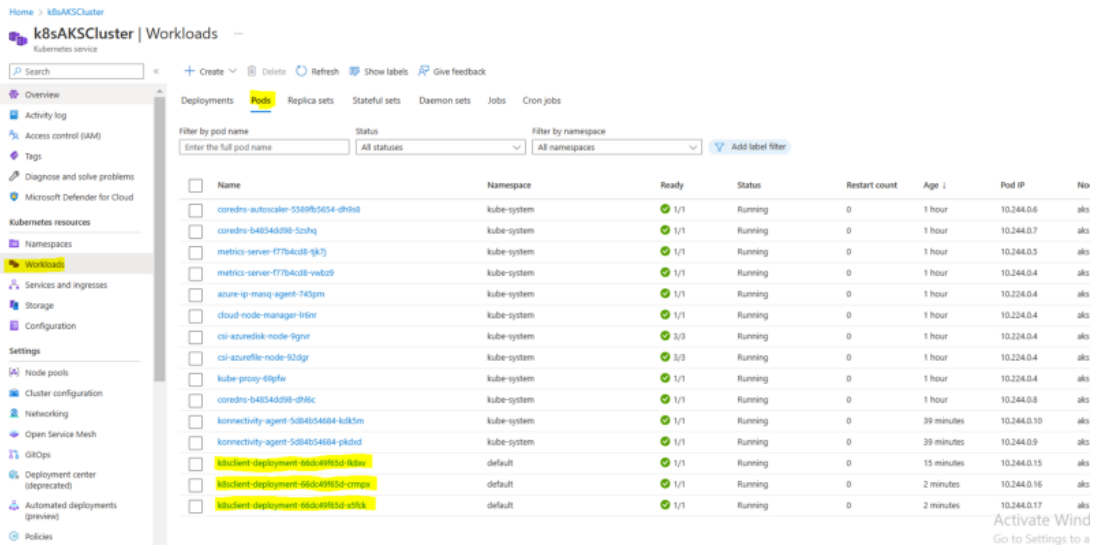
As we can see only deployment is configured again and service is as it is. So now two new pods added in AKS cluster.



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

See the result of applied configuration in AKS cluster.

Name	Namespace	Ready	Up-to-date	Available	Age
coredns	kube-system	2/2	2	2	1 hour
coredns-autoscaler	kube-system	1/1	1	1	1 hour
connectivity-agent	kube-system	2/2	2	2	1 hour
metrics-server	kube-system	2/2	2	2	1 hour
k8sclient-deployment	default	3/3	3	3	14 minutes



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?

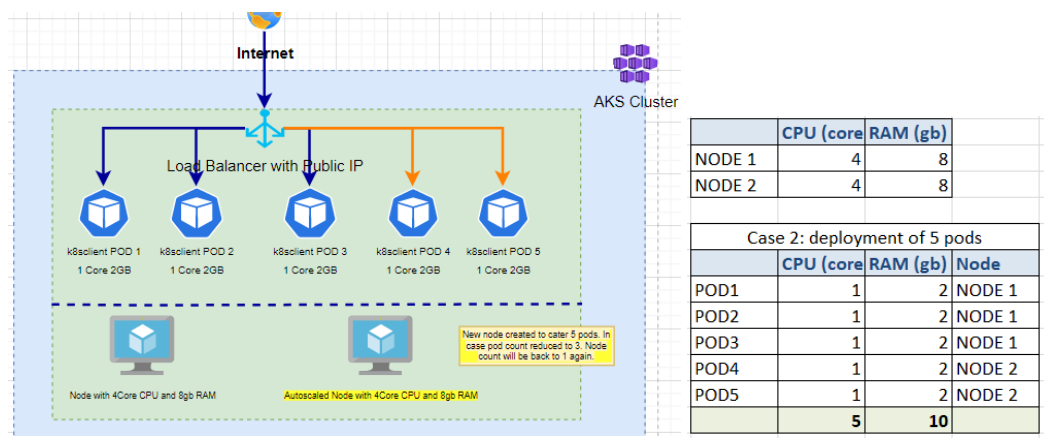
There is **cluster-autoscaler** setting available in **Azure Kubernetes service**. We have to specify min and max count of nodes.

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.



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

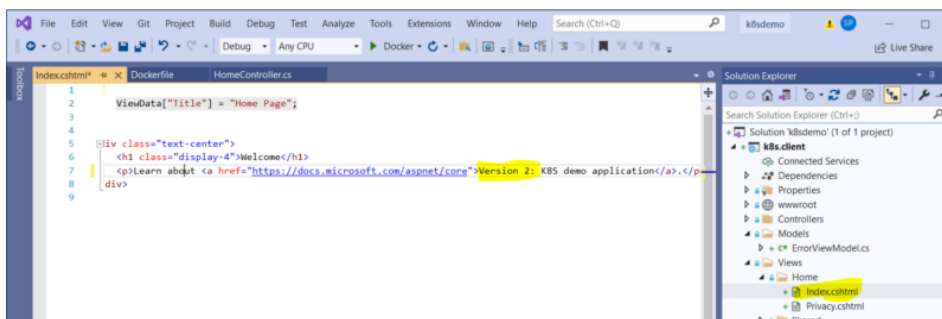
We can create and manage group of nodes using nodepool.

Step 9: Explanation of Zero downtime deployment of microservices

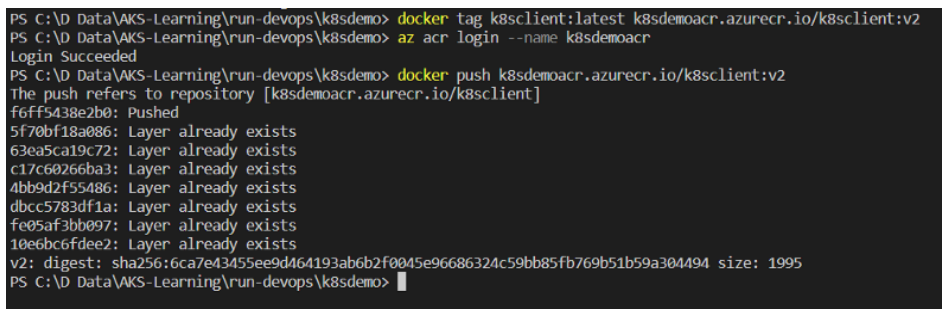
Application/Microservice is deployed in production AKS cluster. It is running in 3 different pods.

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?

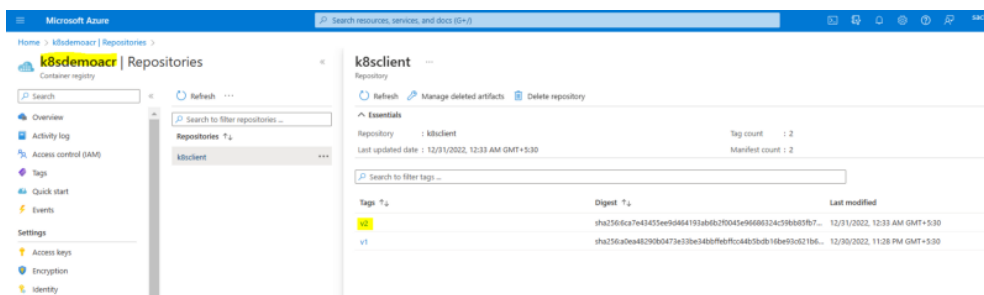
Let's correct the code and generate image.



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



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



Update the image version in k8sclient.yaml file.

```

7 spec:
8   replicas: 3
9   selector:
10    matchLabels:
11     app: k8sclient
12   template:
13     metadata:
14     labels:
15      app: k8sclient
16     spec:
17     containers:
18     - name: k8sclient
19       image: k8sdemoacr.azurecr.io/k8sclient:v2
20     ports:
21     - containerPort: 80
22     env:
23     - name: ASPNETCORE_ENVIRONMENT

```

Updated version number of the image in .yaml file which will replace existing version from the PODs.

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

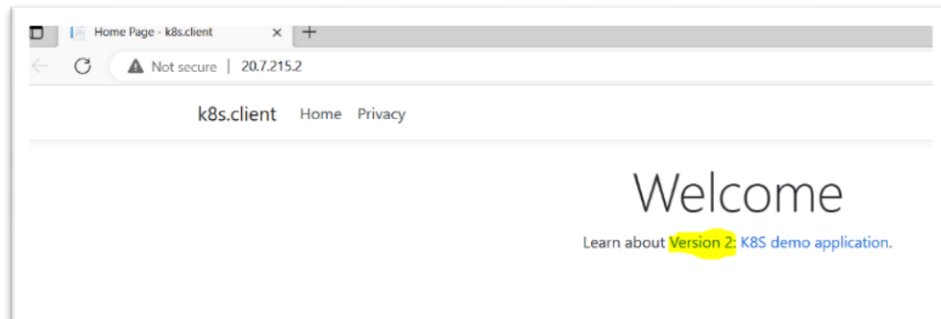
For reference you can check the events.

The screenshot shows the 'Events' page for the 'k8sclient-deployment' in the 'k8sAKScluster' namespace. The events table shows a sequence of operations performed by the 'deployment-controller' source:

Message	Reason	Source	Sub-object	Count	First seen	Last seen
Scaled up replica set k8sclient-deployment-66d49f5d to 1	ScalingReplicaSet	deployment-controller		1	2022-12-30T18:27:07Z	2022-12-30T18:27:07Z
Scaled up replica set k8sclient-deployment-66d49f5d to 3	ScalingReplicaSet	deployment-controller		1	2022-12-30T18:40:12Z	2022-12-30T18:40:12Z
Scaled up replica set k8sclient-deployment-58889f2f9 to 1	ScalingReplicaSet	deployment-controller		1	2022-12-30T19:10:03Z	2022-12-30T19:10:03Z
Scaled down replica set k8sclient-deployment-66d49f5d to 2	ScalingReplicaSet	deployment-controller		1	2022-12-30T19:10:05Z	2022-12-30T19:10:05Z
Scaled up replica set k8sclient-deployment-58889f2f9 to 2	ScalingReplicaSet	deployment-controller		1	2022-12-30T19:10:05Z	2022-12-30T19:10:05Z
Scaled down replica set k8sclient-deployment-66d49f5d to 1	ScalingReplicaSet	deployment-controller		1	2022-12-30T19:10:06Z	2022-12-30T19:10:06Z
Scaled up replica set k8sclient-deployment-58889f2f9 to 3	ScalingReplicaSet	deployment-controller		1	2022-12-30T19:10:06Z	2022-12-30T19:10:06Z
Scaled down replica set k8sclient-deployment-66d49f5d to 0	ScalingReplicaSet	deployment-controller		1	2022-12-30T19:10:07Z	2022-12-30T19:10:07Z

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 CI/CD 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 CI/CD pipeline we can automate the manual process of deployment. This will allow faster end to end deployments in production with zero

downtime. 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](#))