Beginner to Skilled

Docker, Kubernetes, Azure Container Registry (ACR) and Azure Kubernetes Service (AKS)

October 2019

Table of Contents

Introduction

This step by step guide follows an ASP.NET Core application that I've published on GitHub, and takes you through all the steps of creating a Docker image locally, deploying it locally as a Docker container, then using a local AKS cluster to deploy a resilient 3-pod replica set, to finally creating an Azure Container Registry to store the image and using it on Azure Kubernetes

Service. By the end of this guide, you'll have a good understanding of Docker and AKS, and the Kubectl CLI tool that you use to interact with Kubernetes.

GitHub Repository: https://github.com/marlinspike/Learn-Net-Core-ASP

Pre-requisites

You'll need the following tools installed:

- Git
- Docker (for whatever platform you're on)
- SQL Server installed locally or elsewhere you can connect to with a connection string

Create a an ASP.NET Core Web App

Clone the Repository

Before starting on this demo, you'll need to clone the Git repository listed above. You do not need Dotnet Core installed, as you won't need to code or compile the app yourself!

```
git clone https://github.com/marlinspike/Learn-Net-Core-ASP.git

/*
This clones the demo repository with an ASP.NET Core web app. You will need to edit
the appsettings.json file to include the connection string to the SQL Server to
connect to. The connection string goes inside the empty quotes (“”) after “MoviesDB”
in the “ConnectionStrings” section.
*/
```

The rest of this guide assumes that you're on a command prompt at the root of the cloned repository.

Create a Dockerfile for the app

You already have a Dockerfile pre-created in the cloned repository. The contents of the file are shown below.

```
FROM mcr.microsoft.com/dotnet/core/aspnet:3.0-buster-slim AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/core/sdk:3.0-buster AS build
WORKDIR /src
COPY ["Learn-Net-Core-ASP.csproj", ""]
RUN dotnet restore "./Learn-Net-Core-ASP.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "Learn-Net-Core-ASP.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "Learn-Net-Core-ASP.csproj" -c Release -o /app/publish
```

```
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "Learn-Net-Core-ASP.dll"]
```

Create a Docker container

```
//Install Docker Desktop, and from the command line in the app folder:

docker build -t coreasp -f Dockerfile .
```

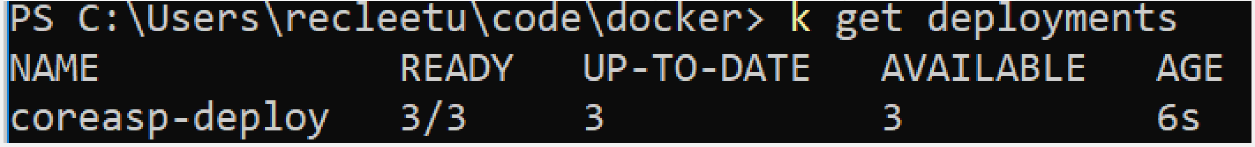
Run the container locally

```
docker container run -p 8080:80 coreasp

/*
Then on your browser, navigate to localhost:8080
-p maps local port 8080 to the container port 80
*/
```

Deploying an app Manually to a Local Kubernetes Cluster

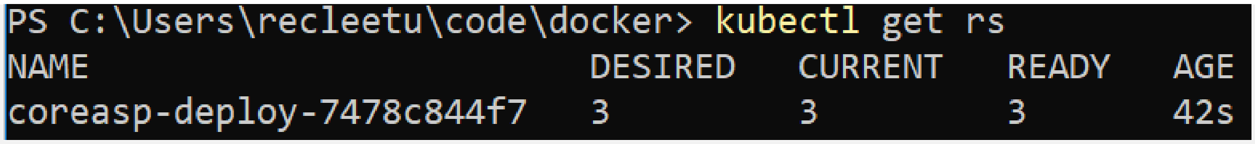
You can use either an interactive (manual) way to deploy an app to Kubernetes, or a Declarative (automated) manner, using a YAML file. I'll demonstrate the Interactive method first briefly, and then proceed to the preferred, Declarative way. You can skip this section entirely and jump to the section named *Create a Container Registry* if you prefer.

```
kubectl run coreasp-deploy --image=coreasp --port=80 --replicas=3

/*
Parameters:
- name of the deployment
- name of the image
- port that the app uses
- Number of replicas to create
*/

Kubectl get deployments
//Gets the deployments, and here it's showing the one we just created
```



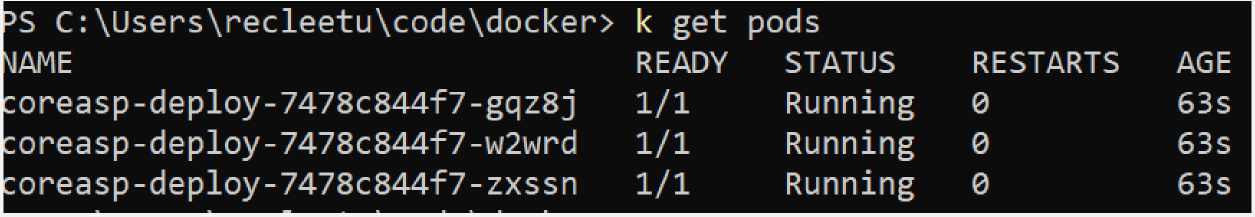
```
PS C:\Users\recleetu\code\docker> k get deployments
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
coreasp-deploy   3/3     3            3           6s
```

```
kubectl get rs
//Gets the replica sets, here showing the one we just requested. Note the name of the
replica set includes the name of the deployment, followed by a GUID
```



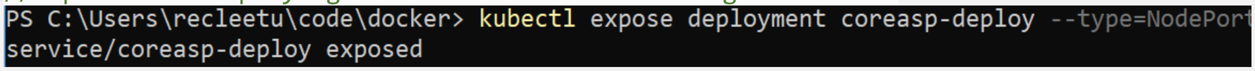
```
PS C:\Users\recleetu\code\docker> kubectl get rs
NAME                       DESIRED   CURRENT   READY   AGE
coreasp-deploy-7478c844f7   3         3         3       42s
```

```
kubectl get pods
//Gets the pods associated with the replica set. Note that the name consists of the deployment name, replica set name, and then the pod name
```



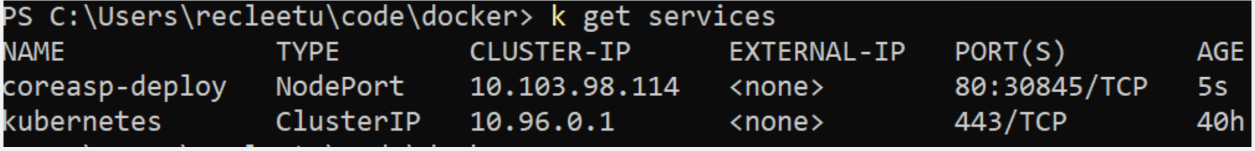
```
PS C:\Users\recleetu\code\docker> k get pods
NAME                             READY   STATUS    RESTARTS   AGE
coreasp-deploy-7478c844f7-gqz8j   1/1     Running   0          63s
coreasp-deploy-7478c844f7-w2wrd   1/1     Running   0          63s
coreasp-deploy-7478c844f7-zxssn   1/1     Running   0          63s
```

```
//Now we need to connect the app to the external world, for which we need a Service. Since this is Docker Desktop, we'll use a NodePort. On AKS, we'll use a NodeType of LoadBalancer.

kubectl expose deployment coreasp-deploy --type=NodePort
//Exposes the deploying to the outside world using a NodePort
```



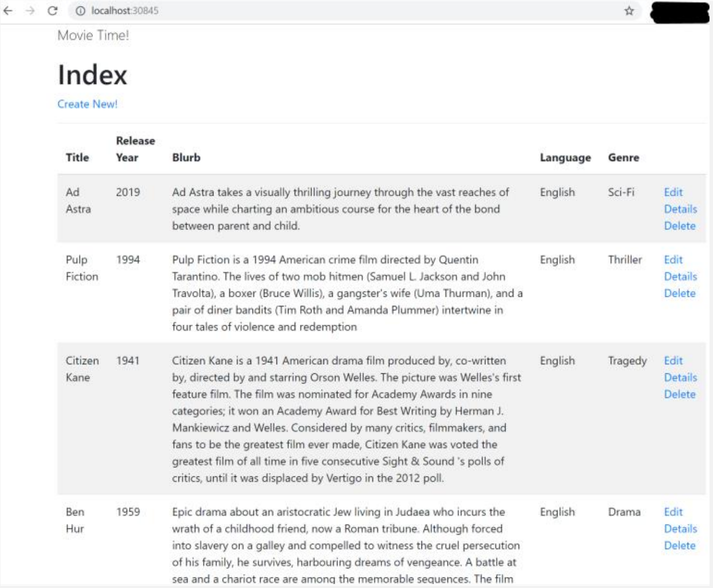
```
PS C:\Users\recleetu\code\docker> kubectl expose deployment coreasp-deploy --type=NodePort
service/coreasp-deploy exposed
```

```
kubectl get services
//Gets the services created. Notice that our NodePort is ready, with the app listenining on Port 80, and the Exposed Port on the Node is 30845
```



```
PS C:\Users\recleetu\code\docker> k get services
NAME             TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
coreasp-deploy   NodePort    10.103.98.114   <none>        80:30845/TCP   5s
kubernetes       ClusterIP   10.96.0.1       <none>        443/TCP        40h
```

//Now browsing to http://localhost:3845, we get the page we're expecting:



localhost:30845

[redacted]

Movie Time!

Index

Create New!

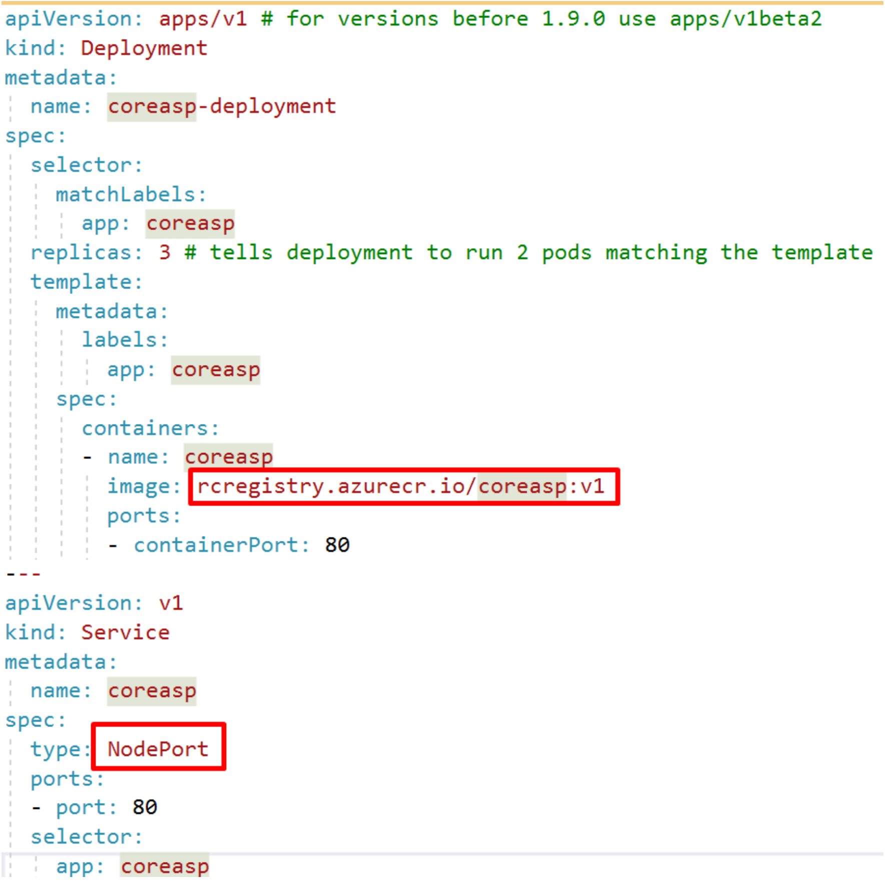
Title	Release Year	Blurb	Language	Genre	
Ad Astra	2019	Ad Astra takes a visually thrilling journey through the vast reaches of space while charting an ambitious course for the heart of the bond between parent and child.	English	Sci-Fi	Edit Details Delete
Pulp Fiction	1994	Pulp Fiction is a 1994 American crime film directed by Quentin Tarantino. The lives of two mob hitmen (Samuel L. Jackson and John Travolta), a boxer (Bruce Willis), a gangster's wife (Uma Thurman), and a pair of diner bandits (Tim Roth and Amanda Plummer) intertwine in four tales of violence and redemption	English	Thriller	Edit Details Delete
Citizen Kane	1941	Citizen Kane is a 1941 American drama film produced by, co-written by, directed by and starring Orson Welles. The picture was Welles's first feature film. The film was nominated for Academy Awards in nine categories; it won an Academy Award for Best Writing by Herman J. Mankiewicz and Welles. Considered by many critics, filmmakers, and fans to be the greatest film ever made, Citizen Kane was voted the greatest film of all time in five consecutive Sight & Sound 's polls of critics, until it was displaced by Vertigo in the 2012 poll.	English	Tragedy	Edit Details Delete
Ben Hur	1959	Epic drama about an aristocratic Jew living in Judaea who incurs the wrath of a childhood friend, now a Roman tribune. Although forced into slavery on a galley and compelled to witness the cruel persecution of his family, he survives, harbouring dreams of vengeance. A battle at sea and a chariot race are among the memorable sequences. The film	English	Drama	Edit Details Delete

Deploying an app Declaratively to a Local Kubernetes Cluster

Once you've done this manually, deploying to Kubernetes via a Deployment file is an absolute breeze. You're going to use the Deployment file that's part of the solution, and be careful to update the few settings for things like the name of the image, and most importantly, the type of the Service. We'll discuss that as we go.

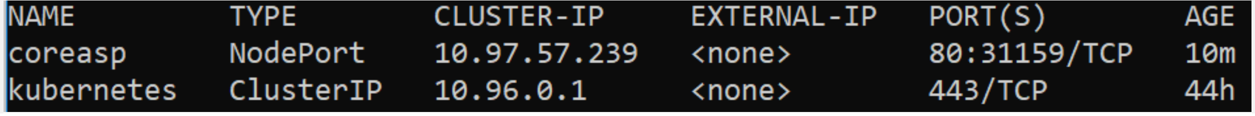
Changing the Service Type

Edit the Deployment.yaml file, and change the **type** and **image** parameters as shown in the screenshot below. The type *must* be NodePort, since we can't create a software load balancer on a local machine, and the image must be whatever you named yours when you created it.



```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: coreasp-deployment
spec:
  selector:
    matchLabels:
      app: coreasp
  replicas: 3 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: coreasp
    spec:
      containers:
      - name: coreasp
        image: rcregistry.azurecr.io/coreasp:v1
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: coreasp
spec:
  type: NodePort
  ports:
  - port: 80
  selector:
    app: coreasp
```

```
kubectl create -f deployment.yaml

/*
This creates the deployment as well as the NodePort service which exposes the
application.
*/

kubectl get services
```



```
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
coreasp      NodePort    10.97.57.239   <none>        80:31159/TCP   10m
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP        44h
```

```
/* This command returns information about the service created as part of the yaml
file. You'll browse to http://localhost:<port>, as defined above. In my case, it's
http://localhost:31159
*/
```

Deploying to a local cluster is exactly how deployment to any Kubernetes cluster happens

This is a really important point – by testing out the Deployment.yaml file here, you've deployed using exactly the same set of steps that you'll use to deploy to any other Kubernetes cluster. There is no difference as far as Kubectl is concerned and for a developer or engineer, that's fantastic! If it works locally, you can pretty much say it's going to work anywhere else now, and you wouldn't be fibbing!

Create a Container Registry

https://docs.microsoft.com/en-us/azure/container-registry/container-registry-get-started-azure-cli

```
az acr create --resource-group dev --name rcregistry --sku Basic
```

Log into the Registry

```
az acr login --name rcregistry
```

Push an image to the Registry

```
docker tag coreasp rcregistry.azurecr.io/coreasp:v1
docker push rcregistry.azurecr.io/coreasp:v1
```

Remove the image from your local registry

```
docker rmi rcregistry/coreasp:v1
```

List the containers in the ACR

```
az acr repository list --name rcregistry --output table
```

Run the image from the registry

```
docker run rcregistry.azurecr.io/coreasp:v1

//This will download the image to your local docker registry
```

Run image on a Mac or Ubuntu

```
docker run -p 5000:80 --rm --name coreasp rcregistry.azurecr.io/coreasp:v1

//add -d if you want to run it as a daemon
```

```
//stop all containers:
docker kill $(docker ps -q)
//remove all containers
docker rm $(docker ps -a -q)
//remove all docker images
docker rmi $(docker images -q)
```

Deploy an AKS Cluster

https://docs.microsoft.com/en-us/azure/aks/tutorial-kubernetes-deploy-cluster

Create an AKS Cluster

```
az aks create \
    --resource-group kuber \
    --name rcklusterr \
    --node-count 1 \
    --generate-ssh-keys \
    --node-vm-size Standard_B2ms \
    --attach-acr rcregistry

// Note the name of the registry being passed with the --attach-acr param.
/*
{
  "aadProfile": null,
  "addonProfiles": null,
  "agentPoolProfiles": [
    {
      "availabilityZones": null,
      "count": 1,
      "enableAutoScaling": null,
      "enableNodePublicIp": null,
      "maxCount": null,
      "maxPods": 110,
      "minCount": null,
      "name": "nodepool1",
      "nodeTaints": null,
      "orchestratorVersion": "1.13.10",
      "osDiskSizeGb": 100,
      "osType": "Linux",
      "provisioningState": "Succeeded",
      "scaleSetEvictionPolicy": null,
      "scaleSetPriority": null,
      "type": "AvailabilitySet",
      "vmSize": "Standard_B2ms",
      "vnetSubnetId": null
    }
  ],
  "apiServerAccessProfile": null,
  "dnsPrefix": "rckluster-kuber-917416",
  "enablePodSecurityPolicy": null,
  "enableRbac": true,
  "fqdn": "rckluster-kuber-917416-14dc0b05.hcp.eastus2.azmk8s.io",
  "id": "/subscriptions/917416c8-760a-45f2-a774-
671813cee4f9/resourcegroups/kuber/providers/Microsoft.ContainerService/managedCluster
s/rckluster",
  "identity": null,
  "kubernetesVersion": "1.13.10",
  "linuxProfile": {
    "adminUsername": "azureuser",
    "ssh": {
      "publicKeys": [
        {
          "keyData": "ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDUiL/Dhb4cJ+08eFFVaSCoaaclswyYkPYZcMq2obNs3r8xoWPm9NmtB
y2Xjwc28J4rrnjV22fDlGg0u9XDzVcdDwosT7b9pUKGrJJbtpF0KL6LZWsG9WPBa1Ija6WqxJdFP6Ny8sXYi4
yEkgwY7YUM51G7ybZrdxH6ll3rBwoZvjFxrn9M8yyXa1MZG6cRz9J0D3TozQ1Ei0R6ANNHkM83cep+DJkmdw/
KCvhapk3QiEwqZV7h02gXhNoFGkBL3fkzNxjFqHIU7hn8xlXgNBEEfuNkEa2hl1m+4PeiWCq7CESHWMQ34rdo
/iaqPvmHt90Zn8wFvdRO6L08Qu8Sbw3N reuben@rembrandt.fios-router.home\n"
        }
      ]
    }
  },
  "location": "eastus2",
  "maxAgentPools": 1,
  "name": "rckluster",
  "networkProfile": {
    "dnsServiceIp": "10.0.0.10",
    "dockerBridgeCidr": "172.17.0.1/16",
    "loadBalancerProfile": null,
    "loadBalancerSku": "Basic",
    "networkPlugin": "kubenet",
    "networkPolicy": null,
    "podCidr": "10.244.0.0/16",
    "serviceCidr": "10.0.0.0/16"
  },
  "nodeResourceGroup": "MC_kuber_rckluster_eastus2",
  "provisioningState": "Succeeded",
  "resourceGroup": "kuber",
  "servicePrincipalProfile": {
    "clientId": "3a983105-fc9f-4fe1-85bd-13b170f2588f",
    "secret": null
  },
  "tags": null,
  "type": "Microsoft.ContainerService/ManagedClusters",
  "windowsProfile": null
}
*/
```

Install the Kubernetes CLI

To connect to the Kubernetes cluster from your local computer, you use *kubectl*, the Kubernetes command-line client.

If you use the Azure Cloud Shell, kubectl is already installed. To use it from your local Linux/Mac/Windows workstation, you need to install it locally using the following command:

```
az aks install-cli command
```

Connect to cluster using kubectl

Once the Kubernetes Cluster has been created, it's time to connect to it, and for that you'll need credentials, which are retrieved directly from the Kubernetes cluster. The credentials are merged into your local kube config file.

```
az aks get-credentials --resource-group kuber --name rckluster

// Merged "rcakscluster" as current context in /home/reuben/.kube/config
```

Verify the connection to the cluster by connecting to it and getting node information for the cluster.

```
kubectl get nodes

/*
NAME                           STATUS   ROLES   AGE   VERSION
aks-nodepool1-54814196-0       Ready    agent   55m   v1.13.10
aks-nodepool1-54814196-1       Ready    agent   56m   v1.13.10
*/
```

Run the container in Kubernetes

Create the Deployment manifest and run the app

Save the file below in the root folder for your project, named appropriately (***deployment.yaml***)

```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: coreasp-deployment
spec:
  selector:
    matchLabels:
      app: coreasp
  replicas: 5
  template:
    metadata:
      labels:
        app: coreasp
    spec:
      containers:
      - name: coreasp
        image: rcregistry.azurecr.io/coreasp:v1
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: coreasp
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: coreasp
```

Apply the changes to Kubernetes

Apply the manifest deployment file to Kubernetes by using the *kubectl* command

```
kubectl apply -f deployment.yaml
```

Once applied, *watch* the Loadbalancer service provision a public-ip with the command:

```
kubectl get service coreasp --watch
```

Navigate to the public IP, and you should see your ASP.NET Core app loaded!

Scale a Kubernetes application

Get the number of Pods

In the deployment manifest submitted, we specifically asked for *2 replicas* and *2 nodes*. Kubernetes balanced the load by placing one pod on each node, and we can see that with the following command:

```
kubectl get pods

/*
NAME                                  READY   STATUS    RESTARTS   AGE
coreasp-deployment-66998bb545-k5qvm   1/1     Running   0          16m
coreasp-deployment-66998bb545-ljh6d   1/1     Running   0          16m
*/
```

Set the number of Pods

Scaling out an application by increasing the number of pods is simple. Either modify the deployment descriptor and submit it again (*kubectl apply -f deployment.yaml)*, or simply tell Kubernetes to scale the pods directly via a kubectl command:

```
kubectl scale --replicas=5 deployment/coreasp-deployment

/*
deployment.extensions/coreasp-deployment scaled
*/
```

To see the new pods, run the *kubectl get pods* command again, and you should see a total of 5 pods:

```
kubectl get pods
/*
coreasp-deployment-66998bb545-4w5rr   1/1     Running   0          39s
coreasp-deployment-66998bb545-7pll9   1/1     Running   0          39s
coreasp-deployment-66998bb545-k5qvm   1/1     Running   0          24m
coreasp-deployment-66998bb545-ljh6d   1/1     Running   0          24m
coreasp-deployment-66998bb545-n2b8z   1/1     Running   0          39s
*/
```

Manually scale AKS Nodes

The example we've built create a node cluster consisting of 2 nodes, but you can adjust the number of nodes manually if you plan more or fewer container workloads on your cluster.

The following example increases the number of nodes to 3 in the Kubernetes cluster named rcakscluster. The command takes a couple of minutes to complete.

```
az aks scale --resource-group kuber --name rcakscluster --node-count 3

/*
{
  "aadProfile": null,
  "addonProfiles": null,
  "agentPoolProfiles": [
    {
      "availabilityZones": null,
      "count": 3,
…
*/
```

The command should take a few minutes to execute and running *kubectl get nodes* should now show three nodes present in the cluster. Scaling it back down is just as easy.

```
kubectl get nodes

/*
aks-nodepool1-54814196-0   Ready    agent   3h      v1.13.10
aks-nodepool1-54814196-1   Ready    agent   3h      v1.13.10
aks-nodepool1-54814196-2   Ready    agent   4m39s   v1.13.10
*/
```