Contents

[Demo Guide: Module 5 - Container Orchestrators 2](#_Toc51057578)

[**Demo 1: Create and Scale Azure Kubernetes Service cluster** 2](#_Toc51057579)

[**Demo 2: Deploy a Windows Container to AKS** 7](#_Toc51057580)

[**Demo 3: Working with Kubernetes MiniKube** 10](#_Toc51057581)

Demo Guide: Module 5 - Container Orchestrators

**Demo 1: Create and Scale Azure Kubernetes Service cluster**

The goal of this demo is to show attendees how to create an AKS cluster using the az CLI, as well as to deploy a public image in it. Make sure the cluster’s creation is done before the demo since it takes long to complete.

Tasks

1. **Create Windows Kubernetes Cluster on Azure Kubernetes Services – Complete this task before your demo**
2. Sign onto your LOD Windows VM and open Chrome on the VM. Navigate to portal.azure.com and sign in with your Azure Pass account.
3. On Windows server, open PowerShell.
4. Login to Azure (the command line will give you a URL and a code, put those into your browser to login. Once you are logged in, then close the browser and go back to command line, click on the command line window, hit enter, and you will see your subscriptions).

az login

1. You should see a list of subscriptions once you are logged in. Set the correct subscription, put the name of the subscription in quotes.

az account set --subscription “*subscription\_name*”

1. Create a resource group named (replace initials with your initials so the resource group name is unique) *k8s-win-cluster-rg-INITALS*

az group create --name=k8s-aks-cluster-rg-INITALS --location=eastus

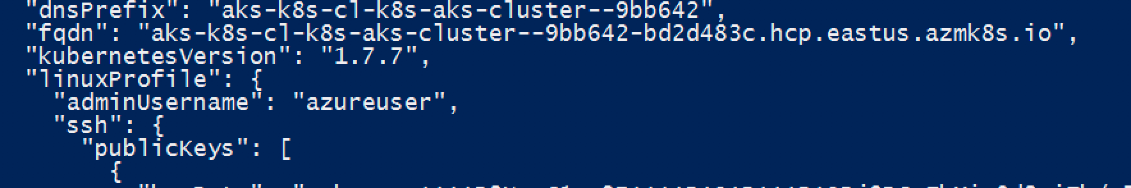
1. Run the following command to deploy Kubernetes cluster with AKS (replace initials with your initials).

az aks create --resource-group k8s-aks-cluster-rg-INITALS --name aks-k8s-cluster --disable-rbac --node-count 2 --node-vm-size "Standard\_D2\_v3" --generate-ssh-keys

Note that this command will generate public/private ssh keys in c:\users\super\.ssh folder.

It may take approximately 10 minutes for the cluster to provision successfully.

1. After the cluster is provisioned successfully you will be shown JSON output describing the AKS cluster.
2. Locate and copy the value of “fdqn” attribute from the JSON output. Note: in the “agentPoolProfiles” there is a fqdn that is null, ignore this one please, and look further down and you will see a second fqdn and this in the one you will need. Please note that your fdqn value may differ from the output shown below.



1. Install kubectl tool which allows you to run commands against Kubernetes cluster. Skip this step if it is already installed on your machine (run “kubectl” to see if the command is recognized).

**Windows:**

choco install -y kubernetes-cli

**Linux:**

sudo apt-get -y update && sudo apt-get -y upgrade

curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl

chmod +x ./kubectl

sudo mv ./kubectl /usr/local/bin/kubectl

In Linux, due to current bug with azure-cli you need to revert to 2.0.23-1 version.

sudo apt-get -y purge azure-cli

sudo apt-get -y install azure-cli=2.0.23-1

1. Run the following command to download the Kubernetes cluster configuration to the local config file *C:\users\super\.kube\config*

az aks get-credentials --resource-group k8s-aks-cluster-rg-INITALS --name aks-k8s-cluster

Be aware that you will need the content of *.kube/config* file while creating Kubernetes service endpoint on VSTS in the DevOps module.

1. **Demonstrate how to deploy resources through the kubectl CLI**
2. Run following command to ensure context is set to the correct cluster:

kubectl config set-context aks-k8s-cluster

1. You will now test the cluster by running a nginx container. First create a new deployment using the following command:

kubectl create deployment nginx --image=nginx

1. Next, make sure that you can access the container from the external (public IP). To do that use the expose command to expose port 80 and enable the external IP (type=LoadBalancer).

kubectl expose deployment nginx --port=80 --type=LoadBalancer

1. The above command essentially creates a service with the name nginx. You can view the service by running following command

kubectl get service nginx

A blue and white text

Description automatically generated

Please wait until EXTERNAL-IP for nginx service change from <pending> to a valid IP address. It may take few minutes and you may have to run the command *kubectl get service nginx* few times to probe the status of external IP assignment. Another useful parameter is **-w** that can be added to the command to keep watching the service as it changes. When it is done, it will look like below:

A close up of a sign

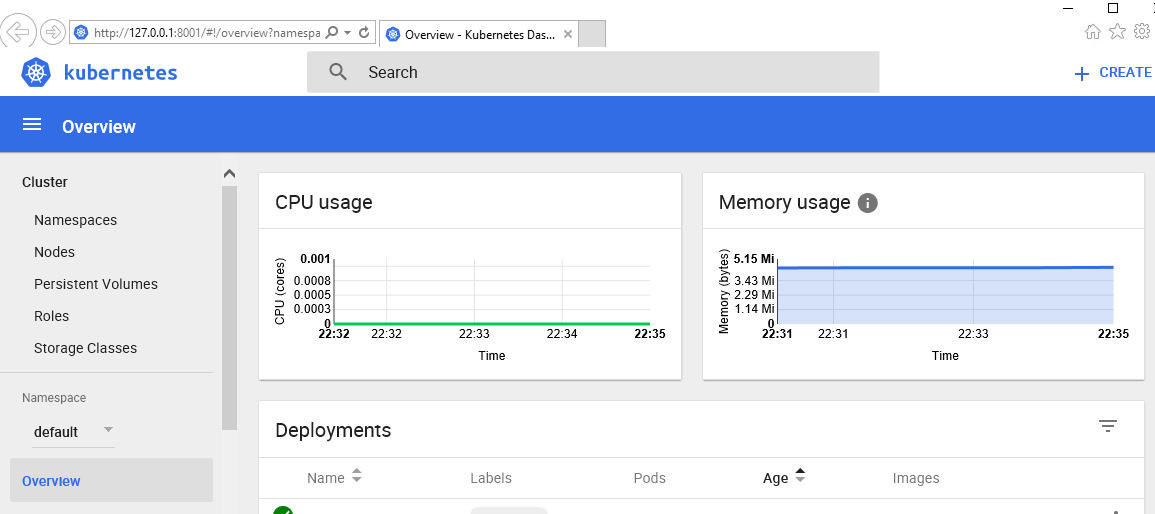
Description automatically generated

1. You can now simply query the content hosted by nginx by using the curl command:

curl http://$( kubectl get service nginx -o=jsonpath='{.status.loadBalancer.ingress[\*].ip}')

1. Kubernetes also provide UI in the form of dashboard. To access the dashboard run the following command. Please refresh the opened browser if needed after a couple of seconds.

az aks browse --resource-group k8s-aks-cluster-rg-INITALS --name aks-k8s-cluster



You can always access the dashboard by browsing to the URL: http://127.0.01:8001. Although this command works with both Windows and Linux you should only run it on Windows as you have access to full browser on Windows running inside the lab virtual machine.

1. Stop running by hitting “Ctrl + C”, then “Y” to terminate the batch job.
2. Let’s check total number of pods running now. You should see two pods ready with a status of running.

kubectl get pods

1. You will now scale the number of pods using replicaset. You will get more details about the replica set using the following command:

kubectl get replicaset

1. To scale, run the command and pass the name of deployment that was created earlier:

kubectl scale --replicas=3 deployment/nginx

1. Now if you run the command to get the number of pods running after scaling it should return three pods (earlier it was one). Note the name of the nodes running the pods as well as pods’ IP.

kubectl get pods -o wide

1. In pervious step you have successfully increase the number of pods by increasing the replica set. However, all the pods are still running on two nodes node since the cluster was initially created with two nodes. You can also check that by running the following command:

kubectl get nodes

In addition to the number of pods, we might want to adjust the compute capacity of the cluster itself. For instance, to add a worker node to the cluster, we can run the following command:

**NOTE:** you might want to only show the following few commands to attendees as it takes a long time to complete.

az aks scale --node-count=3 --resource-group k8s-aks-cluster-rg-INITALS --name aks-k8s-cluster

1. If you check the number of nodes again, you should see three worker nodes instead of two running.

Next time when you scale the pods using replica set, third worker node will also participate. You can also look at the maximum pod capacity of a node by running the command:

*kubectl get node NODE-NAME -o=jsonpath='{.status.capacity.pods}'*

1. To scale back down to a node count of 2, run the following command:

az aks scale --node-count=2 --resource-group k8s-aks-cluster-rg-INITALS --name aks-k8s-cluster

1. Finally, remove the deployment and service using the commands below:

kubectl delete deployment nginx

kubectl delete service nginx

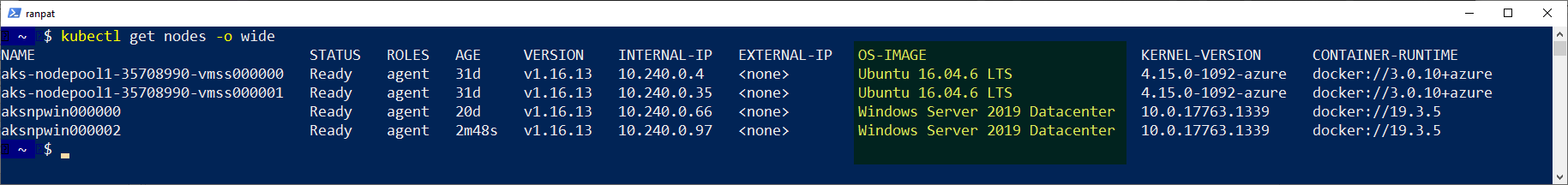
**Demo 2: Deploy a Windows Container to AKS**

In this demo you will deploy a Windows IIS Container to the Azure Kubernetes Cluster

Perform this action before the class begins as it can take awhile

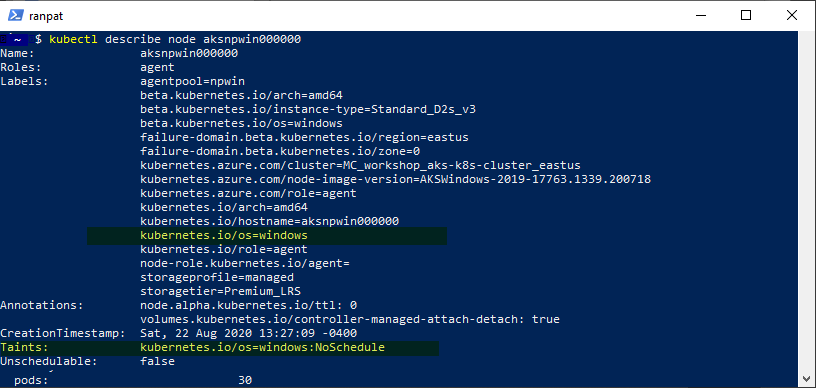
1. az aks nodepool add --resource-group <<resource group>> --cluster-name aks-k8s-cluster --os-type Windows --name npwin --node-count 2 --node-taints kubernetes.io/os=windows:NoSchedule
2. View Node Taints and Labels
   1. Verify your AKS cluster has 2 windows and 2 linux nodes

Kubectl get nodes -o wide



* 1. *Describe* one of the Windows Nodes and show the Taints and Labels. Focus on the automatic node label ‘kubernetes.io/os=windows’ used to differentiate Windows and Linux nodes

kubectl describe node <<node name>>



* 1. Describe a Linux Node and show the ‘kubernetes.io/os=linux’ Label and no Taints

1. Navigate to labs\module5
   1. View the iis-demo YAML file

code .\iis-demo.yaml

Review the **tolerations** and **nodeSelector**

The toleration is used to include the Windows nodes as potential candidates to deploy to and the node selector is used to select the nodes with a Windows operating system.

spec:

replicas: 2

selector:

matchLabels:

app: iis-demo

template:

metadata:

labels:

app: iis-demo

spec:

tolerations:

- key: kubernetes.io/os

operator: Equal

value: windows

effect: NoSchedule

nodeSelector:

kubernetes.io/os: windows

containers:

- name: iis-demo

image: mcr.microsoft.com/windows/servercore/iis

ports:

- containerPort: 80

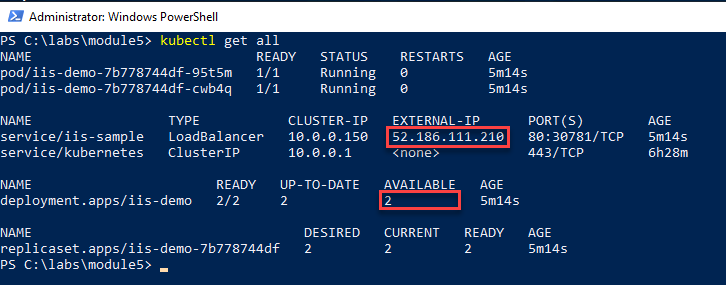
* 1. Deploy the 2 Windows Containers

kubectl apply -f .\iis-demo.yaml

It may take several minutes for the nodes to download the windows image before the pod status changes to running

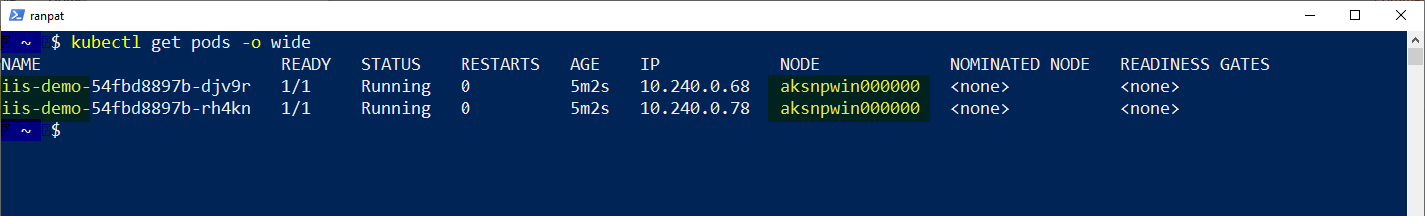
* 1. Wait for the application to deploy. Verify that EXTERNAL-IP is available for **service/iis-sample** and **deployment.apps/iis-demo** has **2** pods available

kubectl get all



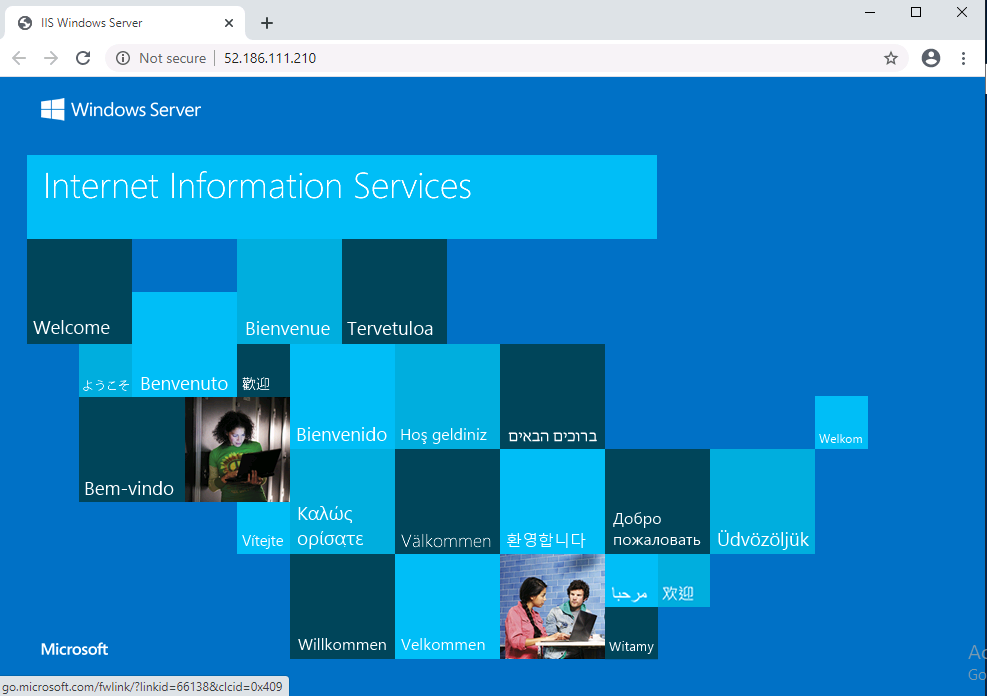
* 1. Verify Windows pods are deployed to Windows Nodes

kubectl get pods -o wide



* 1. launch the browser and verify that the default IIS landing page is shown

start http://$( kubectl get service iis-sample -o=jsonpath='{.status.loadBalancer.ingress[\*].ip}')



**Demo 3: Working with Kubernetes MiniKube**

In this exercise, you will create a Kubernetes cluster on Azure Container Service (managed) AKS service and then deploy a NGINX container image to that cluster. The container image will be fetched from Docker Hub public registry. You will also learn how the application can be scaled using command line or dashboard.

**Tasks**

1. Install Minikube – Complete this task and minikube install before your demo
2. Sign into your LOD Ubuntu VM.
3. Open a terminal by right clicking anywhere on the Desktop.
4. Go into root and type in your password when prompted. The Password for the VM is: P@ssw0rd123!

sudo -i

1. Run the following commands:

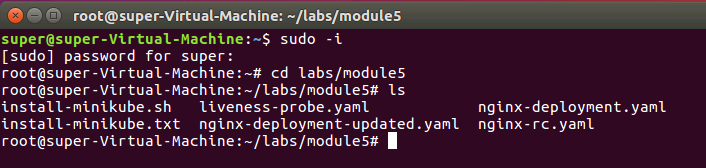
rm /var/lib/apt/lists/lock

rm /var/cache/apt/archives/lock

rm /var/lib/dpkg/lock

1. Navigate to the lab files by running:

cd labs/module5



1. You are now to install Minikube. Run the following two commands (the first command gives execute permission to your script, the second script runs it):

chmod +x /path/to/install-minikube.sh

./install-minikube.sh



1. Deploy Nginx
2. Create a deployment and name it “nginx”

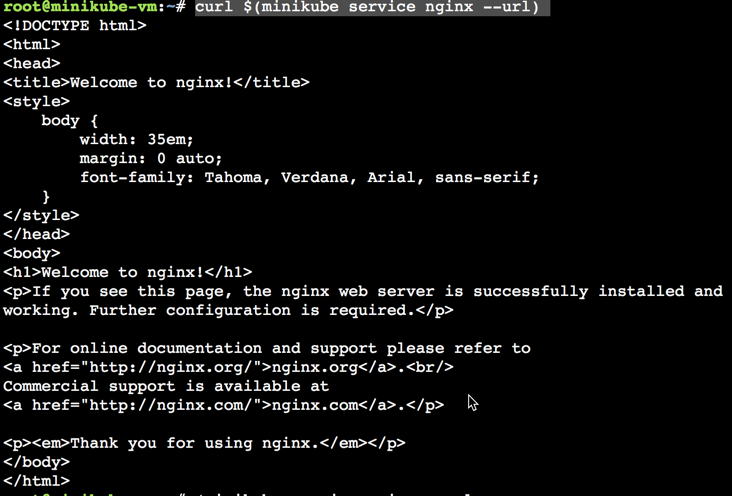
kubectl run nginx --image=nginx --port=80

1. Expose the deployment using a service

kubectl expose deployment nginx --type=NodePort

1. Access the nginx default web page using the curl command.

curl $(minikube service nginx --url)



1. **Working with the health probe**

In this task you will create a new pod and enable a health probe. To test the probe, pod will run a single container that is going to explicitly fail the health probe request after every 5 probes.

1. You should still be in the /labs/module5 folder, if not, navigate to there.
2. Create the pod using the yaml file:

kubectl apply -f liveness-probe.yaml

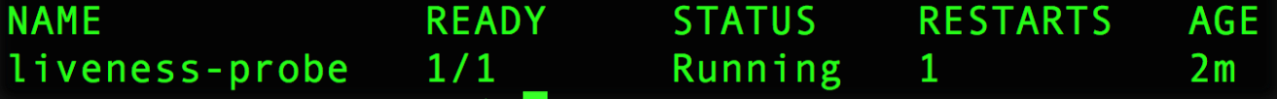
1. Check the status of the newly created pod. It may take few seconds for the container to be up and running.

kubectl get pods

Notice the STATUS column shows Running and RESTARTS column have the value zero. That’s expected because container is just started, and the health probe has not failed yet.



1. After 3-4 minutes if you view the status of pods again you should see the RESTARTS column with the value 1 (or higher depending on how long you have waited to check the status of the pod)

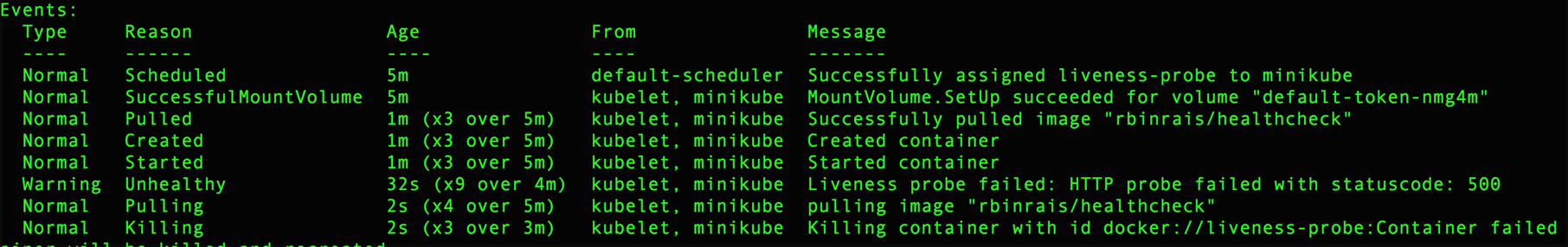


If you wait for few more minutes and check the status of pod again you should see the value of RESTARTS column changes to a higher number.

1. Behind the scenes, every time a container fails the health probe it will be restarted again. To get bit more information about the failing health probe run the following command:

kubectl describe po liveness-probe

This describes the pod in detail along with the events that are happening including the failed health probes



1. Eventually after failing the health probes multiple times in a short interval container will be put under “CrashLoopBackOff” status.



1. You can view the logs from the container that is terminated by using the command:

kubectl logs liveness-probe --previous

The sample docker container application is basic so very limited information is available in logs but typically for production ready applications its recommended to write more detailed messages to the logs.

1. Finally, remove the pod

kubectl delete pod liveness-probe

1. **Working with Replica Set**
2. In this task you will first create a replica with predefined labels assigned to pods. Later you will change the labels for a pod and observer behavior of replica set.

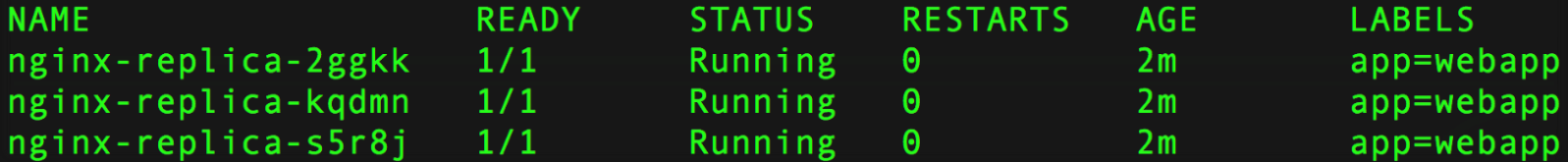
The nginx-rc.yaml file is available inside the /labs/module5 subfolder and contains definition of replica set. If you review the content of file you will notice that it will maintain 3 pods with each running nginx container. Pods are also labeled app=webapp.

To create the replica set and pods run the following command in the **labs/module5 directory**.

kubectl create -f nginx-rc.yaml

1. Let’s look at the pods along with their labels.

kubectl get pods --show-labels



You can also list all the replica sets that are available by using the command:

kubectl get replicaset



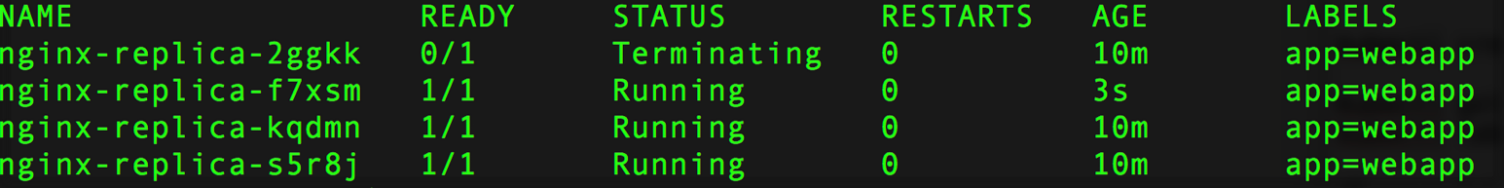
1. Notice we have three pods running. If you delete one of them, replica set will ensure that total pods count remain three and it will do that by creating a new pod.

First delete one of the pods (get name from this command: kubectl get pods --show-labels)

kubectl delete pod <<pod-name>>

Now, check the pods again. Notice you still have three pods running and one of them is terminating.

kubectl get pods --show-labels

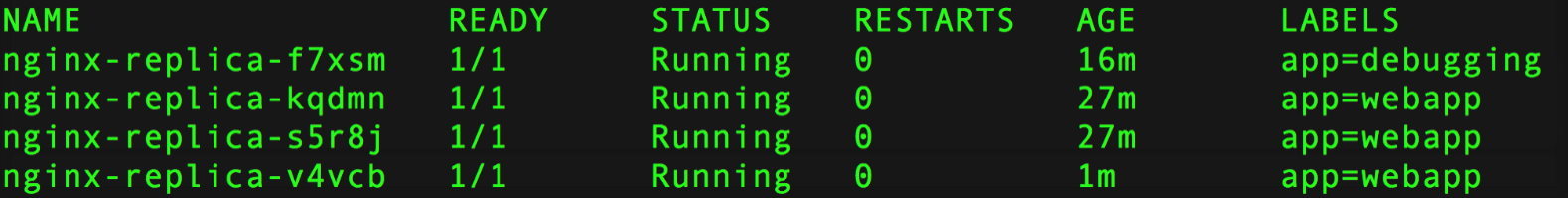


1. Another factor that plays an important role in determining pods relationship with replica set is the labels. Currently app=webapp is the selector used by replica set to determine the pods under its watch. If you change the label of a pod from app=webapp to say app=debugging then replica set will effectively remove it from its watch and create another pod with the label app=webapp. For replica set its job is to maintain the total count of pods to three as per the definition provided in the yaml file.

kubectl label pod <<pod-name>> app=debugging --overwrite=true

1. View the pods again and notice that there are four pods running. Replica set created an additional pod immediately after it noticed pod count was less than three.

kubectl get pods --show-labels



1. Replica set is essentially using selector (defined in the yaml file) to which pods to observe. In this case its label *app* matching value *webapp*. You can also get all the pods with app=webapp label using the following command.

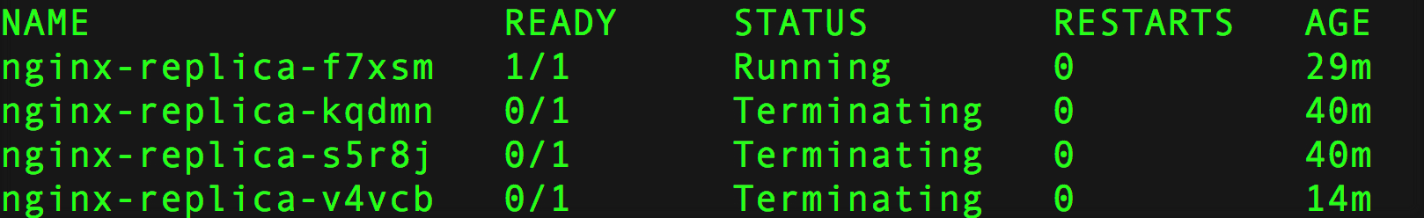
kubectl get pods --show-labels -l app=webapp

1. Finally remove the replica set using the following command.

kubectl delete replicaset nginx-replica

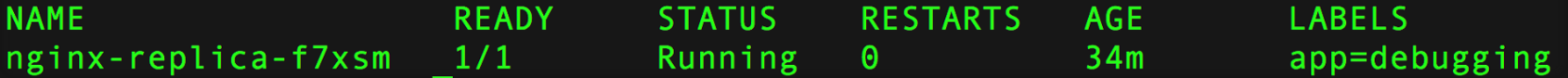
1. As part of the deletion process replica set will remove all the pods that it had created. You can see that by listing the pods and looking at the STATUS column which shows Terminating.

kubectl get pods



Eventually pods will be removed. However, if you list the pods again the pod with label app=debugging is still Running.

kubectl get pods --show-labels



Since you have change the label this pod is no longer manage by the replica set. In cases like these you can bulk remove pods based on labels.

kubectl delete pods -l app=debugging

1. **Working with Deployments**
2. In this task you will begin by performing a deployment based on specific version of nginx container image (v 1.7.9). Later you will leverage RollingUpdate strategy for deployment to update pods running nginx container image from v1.7.9 to container image 1.8.

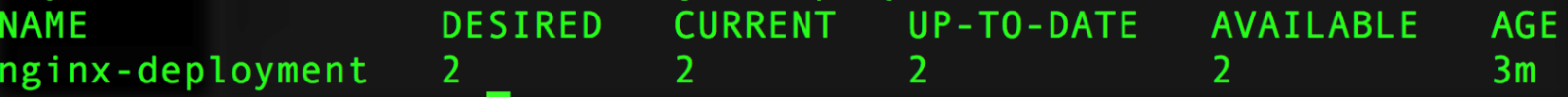
The nginx-deployment.yaml file is available inside the /labs/module5 subfolder and contains definition of deployment. If you review the content of file you will notice that it will maintain 2 pods with each running nginx container image v1.7.9. Pods are also labeled app=nginx.

Run the following command from the labs/module5 directory:

kubectl create -f nginx-deployment.yaml

1. Notice the deployment status by running the command:

kubectl get deployment



1. If you list the pods you should see the out similar to following:

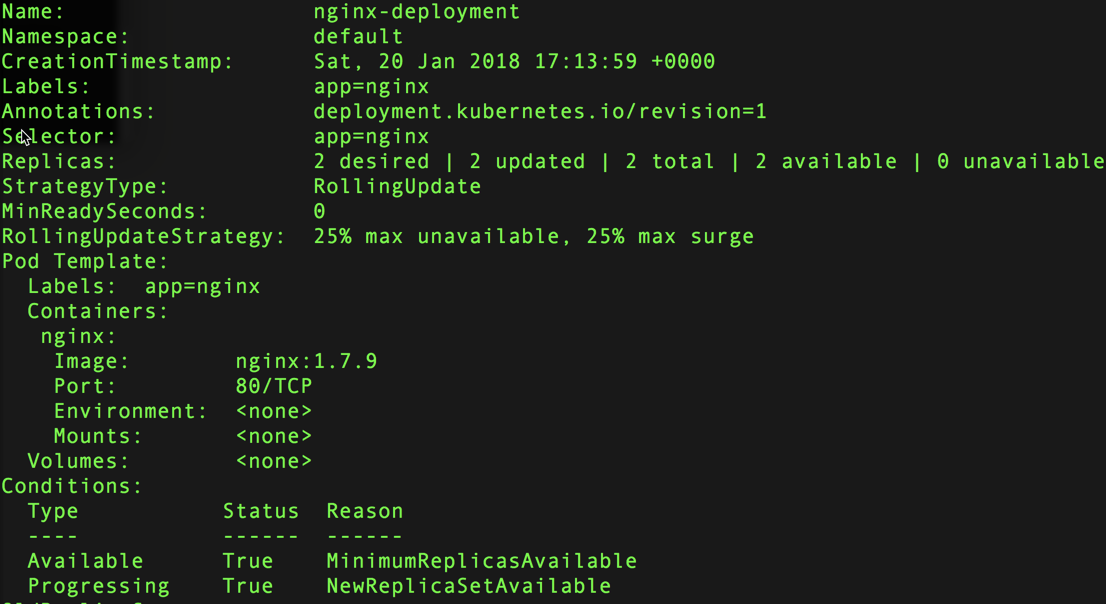
kubectl get pods --show-labels



Notice the LABELS column and presence of pod-template-hash label. This label is used by the deployment during the update process.

1. You are now going to update the deployment. You are going to update nginx container image from v 1.7.9 to v1.8. Before you do that first check the existing definition of the deployment:

kubectl describe deployment nginx-deployment



Notice the line *Image: nginx:1.7.9* which confirms that the current deployment is using 1.7.9 version of nginx image.

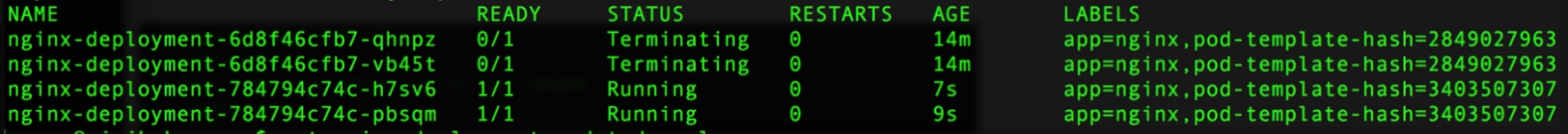
1. Perform the update using the command below.

kubectl apply -f nginx-deployment-updated.yaml

If you review the content of nginx-deployment-updated.yaml file and compare it with original nginx-deployment-updated.yaml the only difference is the image tag which is changed from 1.7.9 to 1.8.

1. If you immediately (after step 5) run the command to list all the pods you should see output like following:

kubectl get pods --show-labels



Notice that the deployment strategy of rolling update ensures that the old pods (nginx v 1.7.9) are terminated only after new pods (nginx image v 1.8) are in a running state. Also notice that the label pod-template-hash values are different for old and new pods. This is because the pod definition (due to change of image tag) is not same for both deployments.

1. You can also look at the new deployment details and make sure that correct nginx image (v 1.8) is used.

kubectl describe deployment nginx-deployment

