Cyber Shujaa

Cloud Security Specialist

Assignment 15: Lab 4 - Configuring and Securing ACR and AKS

NAME: WELDON KIPKIRUI KENEI

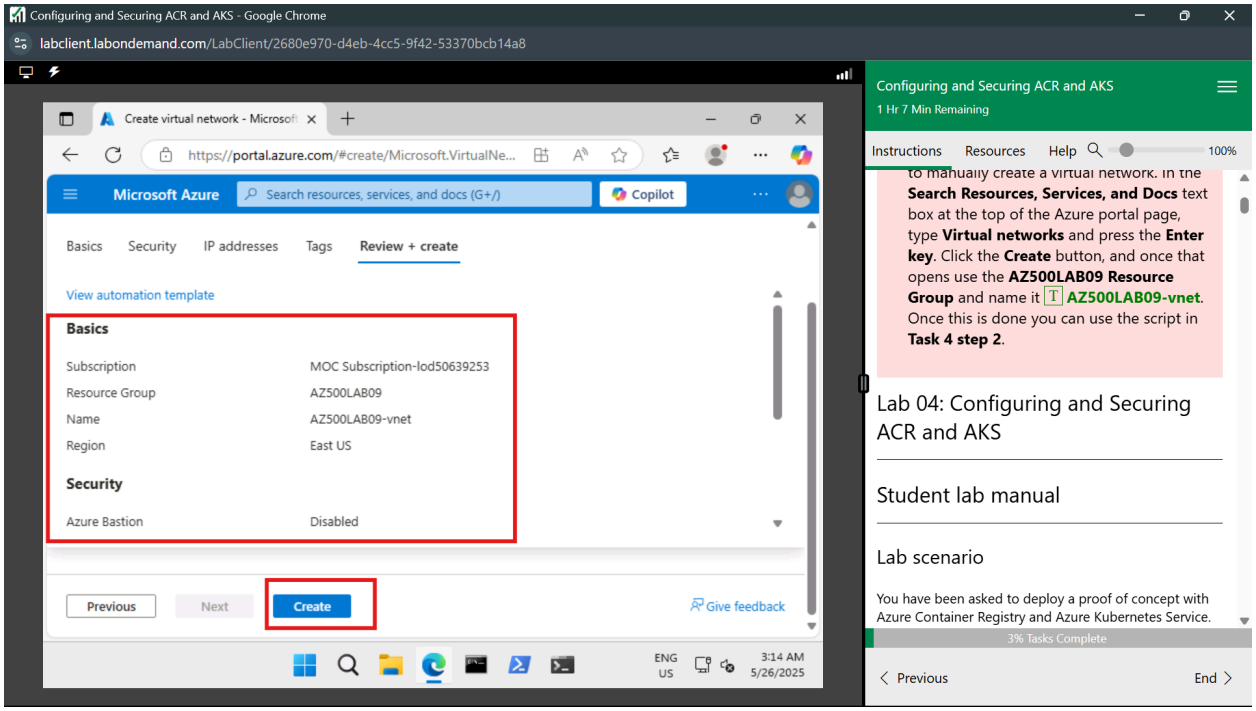
CS NO: ADC-CSS02-25027

Introduction

In this lab scenario, we will configure and secure the Azure Container Registry, which contains container images, and the Azure Kubernetes Service, which manages our container instances. We will follow the steps shown in this report to achieve secure compute services in the Azure cloud environment.

Creating and deploying a virtual network

In this step, we used the Azure portal to provision a virtual network that would be used to host our instances.

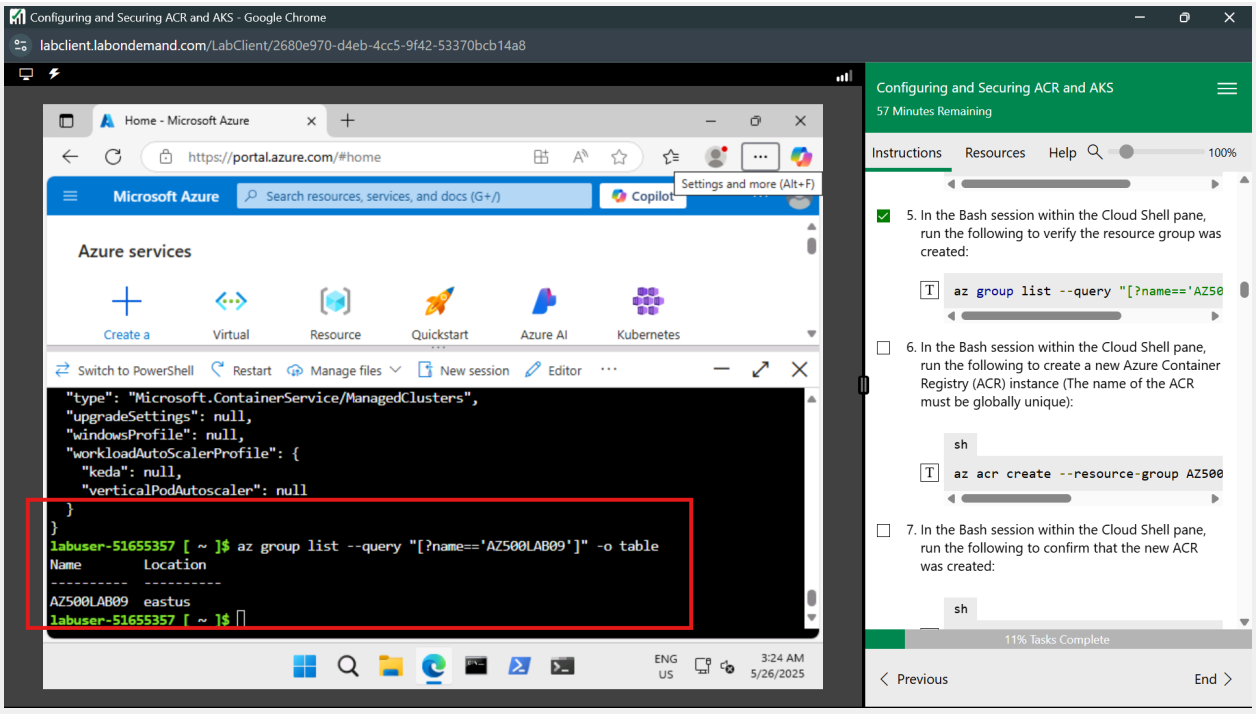


Configuring and Securing ACR and AKS

Now that we have our pre-requisites done with, we can configure and secure our ACR and AKS. These steps will serve as a proof of concept to show how to secure and manage container services, which make up part of of compute resources of the Azure cloud.

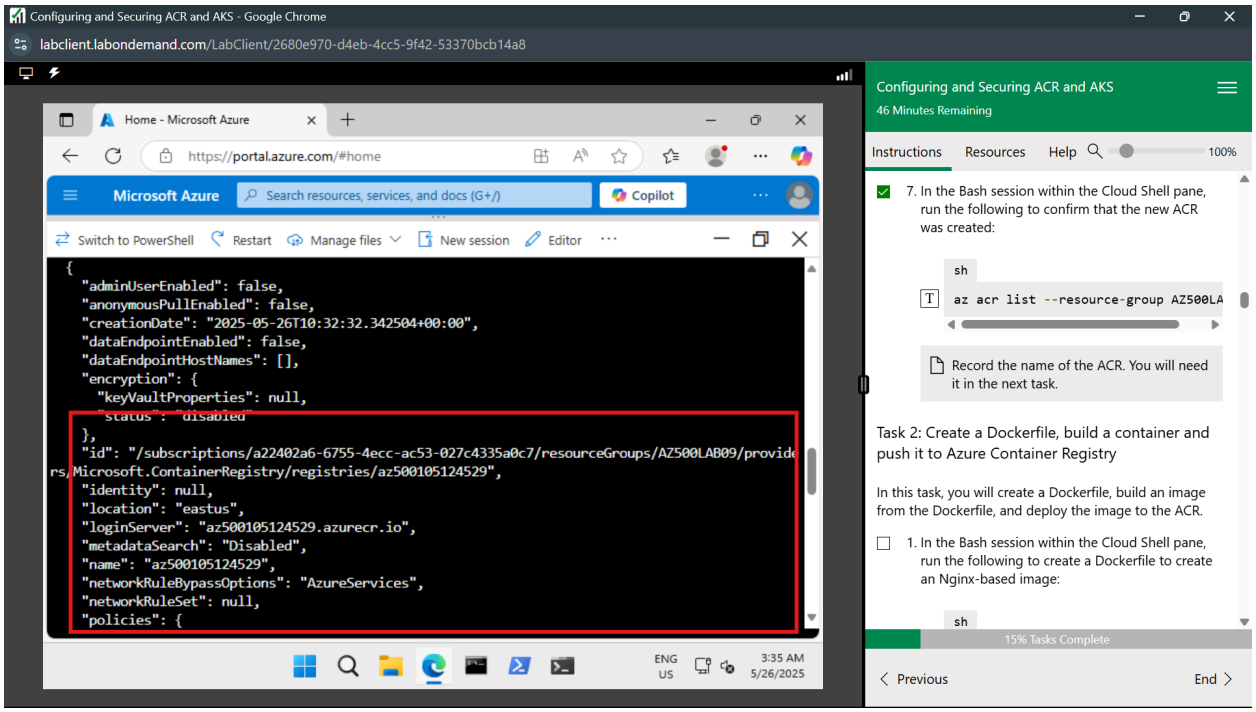
Create an Azure Container Registry

In this task, you will create an Azure Container Registry as shown below. First, we confirm that the AZ500LAB09 resource group exists using the following command. `az group list --query "[?name=='AZ500LAB09']" -o table`



Now we can create our container registry using the bash CLI. The command will be:

```
az acr create --resource-group AZ500LAB09 --name az500$RANDOM$RANDOM --sku Basic
```

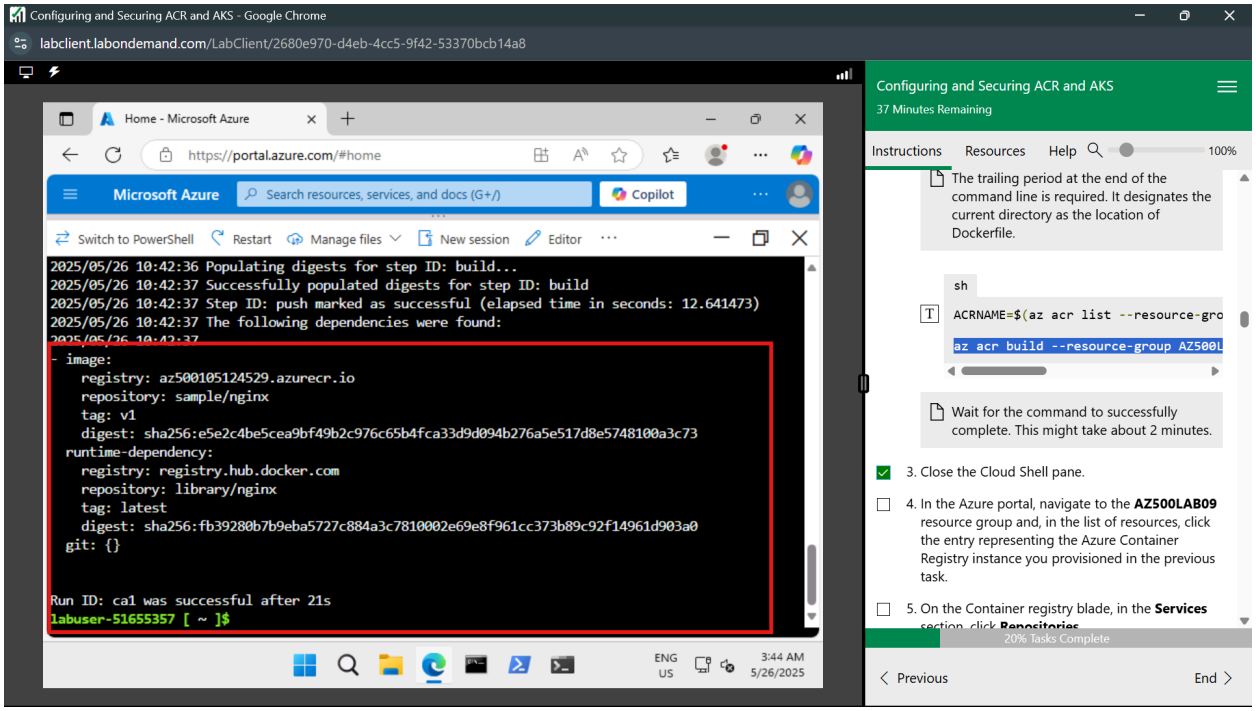


Our registry name is randomly generated as **az500105124529**.

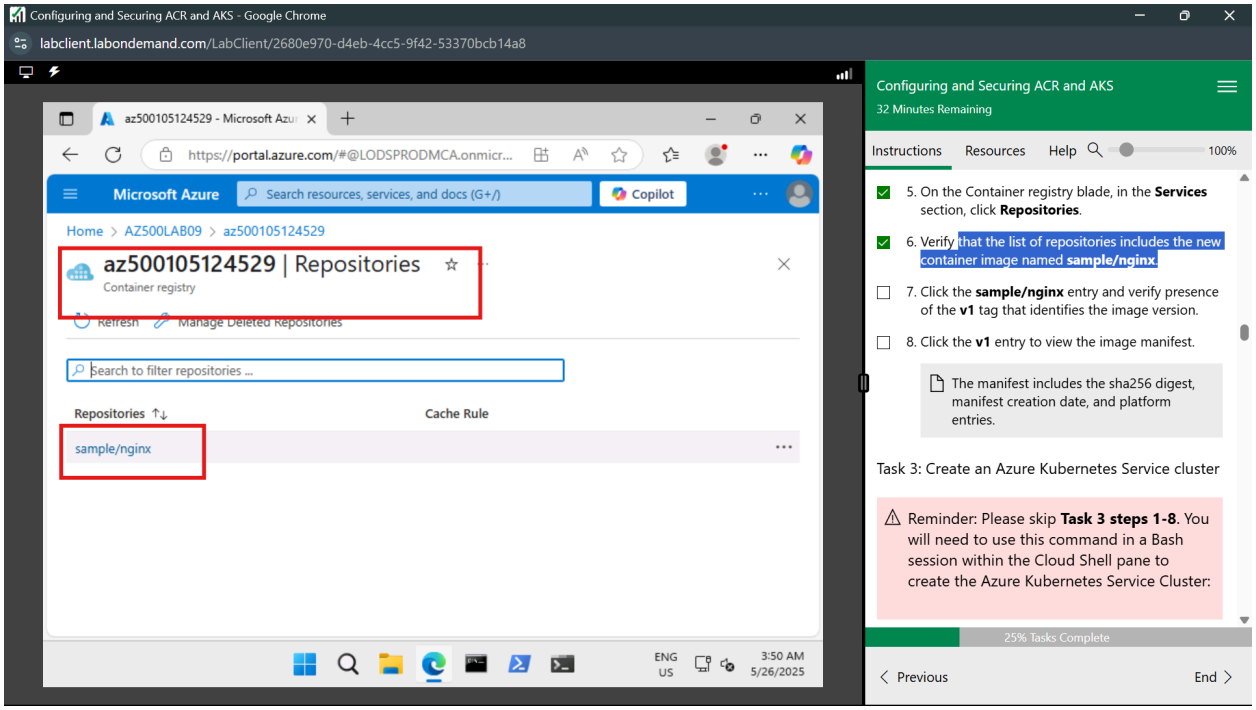
Create a Dockerfile, build a container, and push it to Azure Container Registry

In this step, we will create a Docker file, build a container with it, and push it to our previously provisioned registry. The command to create a Docker file is: `echo FROM nginx > Dockerfile`

We then build a container using the following command: `az acr build --resource-group AZ500LAB09 --image sample/nginx:v1 --registry $ACRNAME --file Dockerfile .`



From our portal, we can verify that the list of repositories includes the new container image named **sample/nginx**.

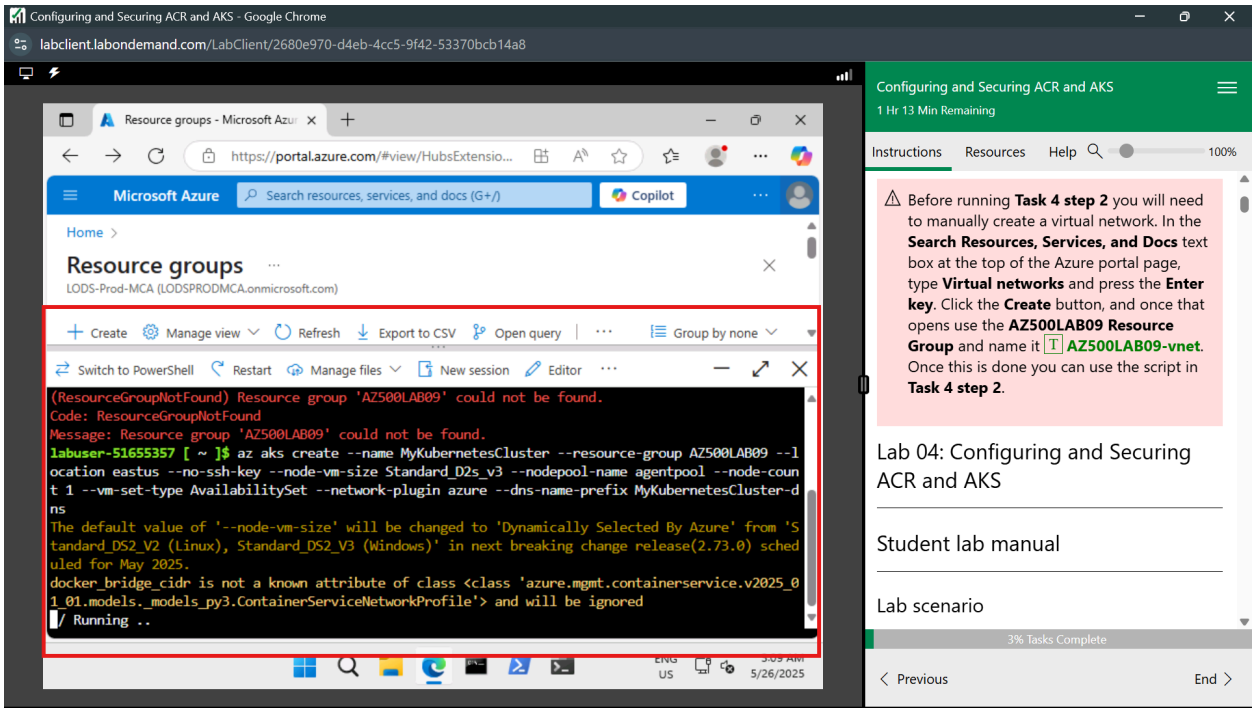


Creating a Kubernetes Cluster using Bash commands

In this step, we will deploy our Kube cluster using this bash command:

```
az aks create --name MyKubernetesCluster --resource-group AZ500LAB09 --location eastus --no-ssh-key --node-vm-size Standard_D2s_v3 --nodepool-name agentpool --node-count 1 --vm-set-type AvailabilitySet --network-plugin azure --dns-name-prefix MyKubernetesCluster-dns
```

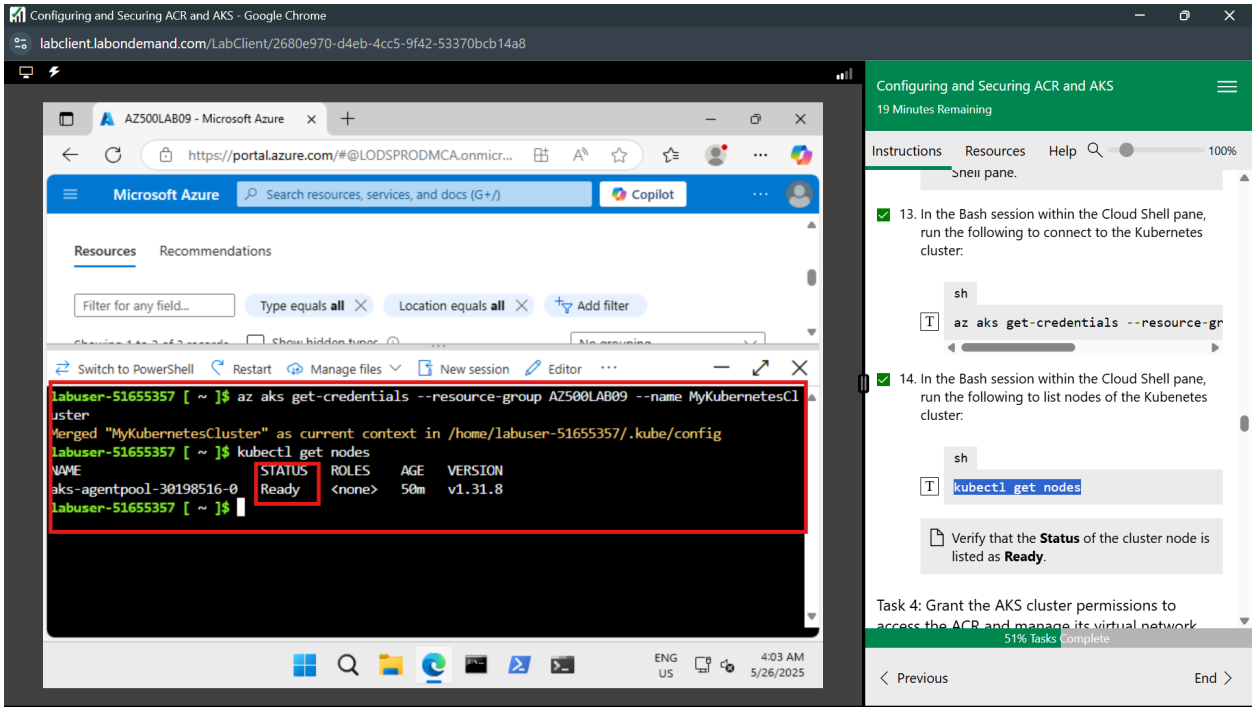
We first created a resource group, AZ500LAB09, in order for our command to run.



We will use two commands in Cloud Shell to connect to our provisioned cluster and verify the status of our nodes. These commands are as follows.

- `az aks get-credentials --resource-group AZ500LAB09 --name MyKubernetesCluster`

  `kubectl get nodes`



Grant the AKS cluster permissions to access the ACR and manage its virtual network

In this step, we will grant the AKS cluster permission to access the ACR and manage its virtual network.

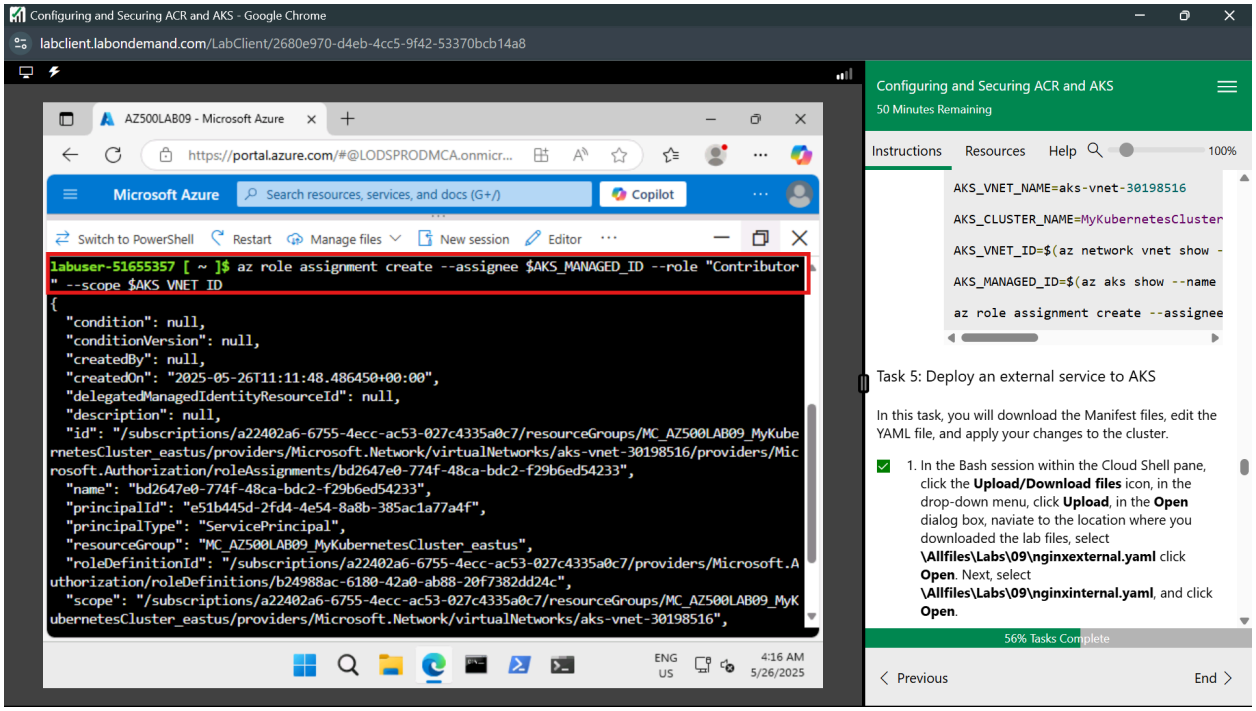
First, we will use commands to configure the AKS cluster to use the Azure Container Registry instance you created earlier in this lab.

```
ACRNAME=$(az acr list --resource-group AZ500LAB09 --query '[].{Name:name}' --output tsv)

az aks update -n MyKubernetesCluster -g AZ500LAB09 --attach-acr $ACRNAME
```

This command grants the 'acrpull' role assignment to the ACR.

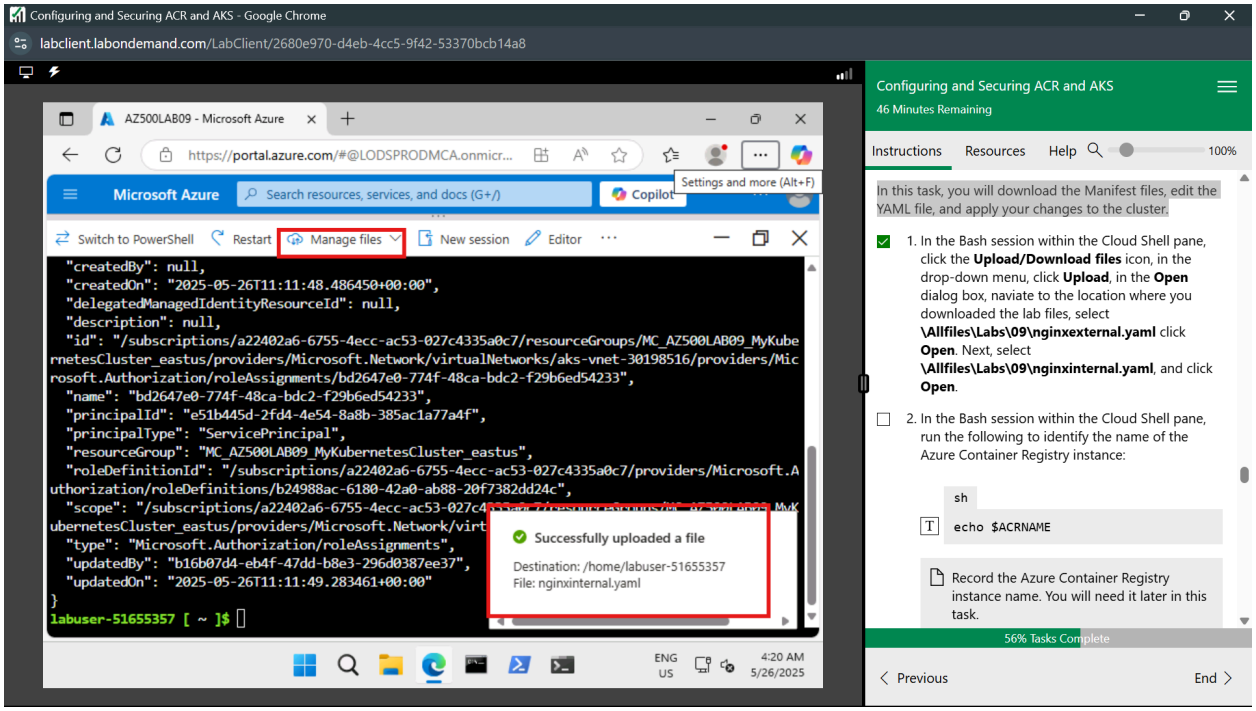
In the next step, we will configure AKS to have a contributor role assignment. The command provided below will have variables that will be used to run the final command.

```
RG_AKS=AZ500LAB09

RG_VNET=MC_AZ500LAB09_MyKubernetesCluster_eastus

AKS_VNET_NAME=aks-vnet-30198516

AKS_CLUSTER_NAME=MyKubernetesCluster

AKS_VNET_ID=$(az network vnet show --name $AKS_VNET_NAME --resource-group $RG_VNET --query id -o tsv)

AKS_MANAGED_ID=$(az aks show --name $AKS_CLUSTER_NAME --resource-group $RG_AKS --query identity.principalId -o tsv)

az role assignment create --assignee $AKS_MANAGED_ID --role "Contributor" --scope $AKS_VNET_ID
```

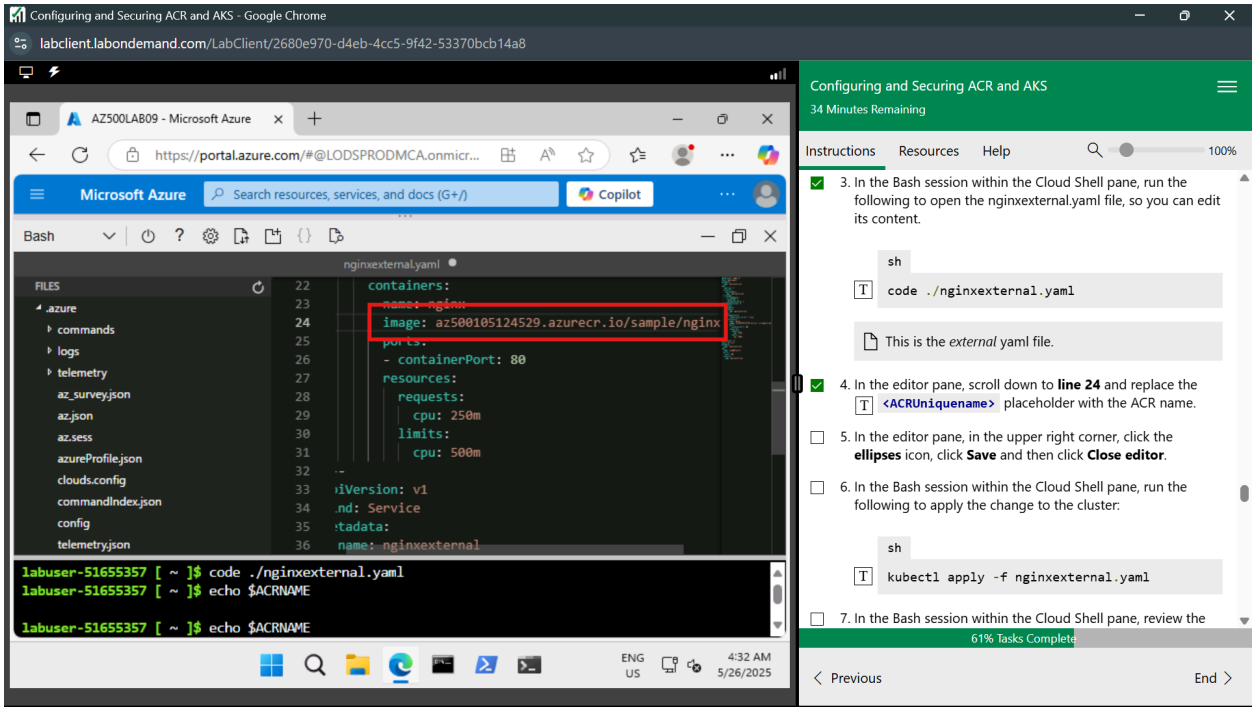


Deploy an external service to AKS

In this task, we will download the Manifest files, edit the YAML file, and apply our changes to the cluster. First, we upload the .yml files to our Cloud Shell as shown;

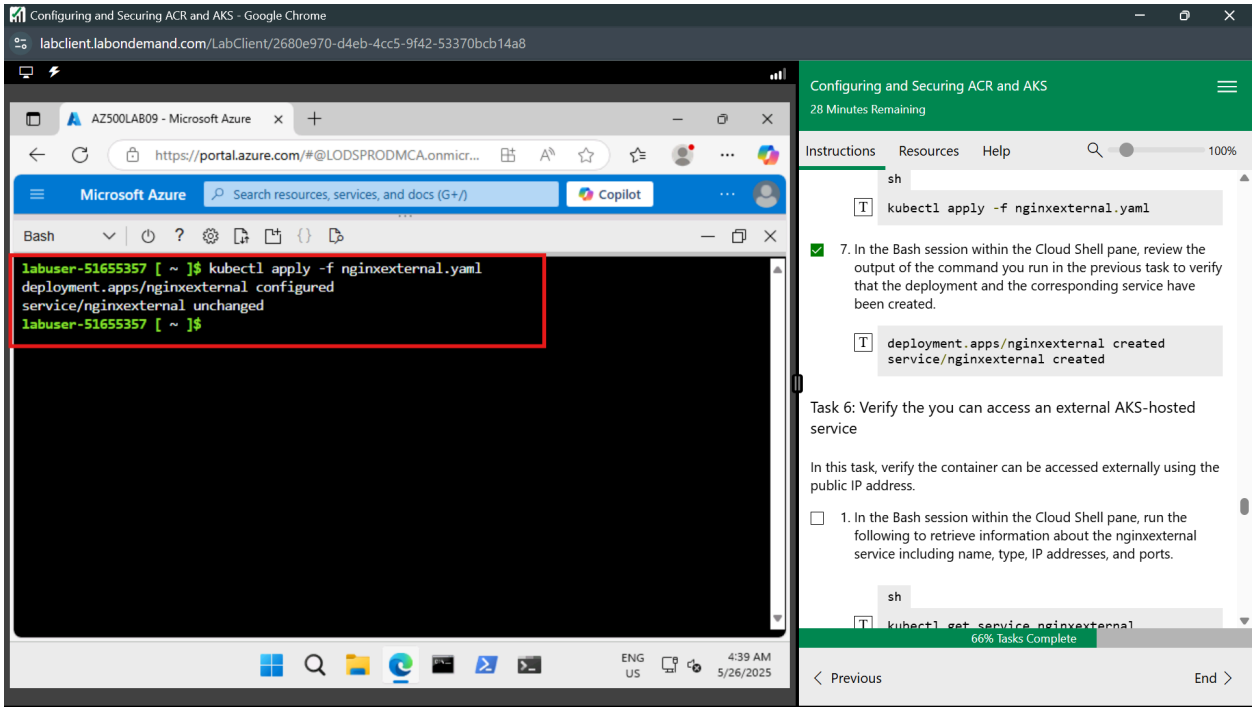


Next, we open an nginxexternal.yml file so that we can edit it in our Cloud Shell.



We changed the name of our registry on line 24 of the file. Now we can run the following to apply changes to our cluster.

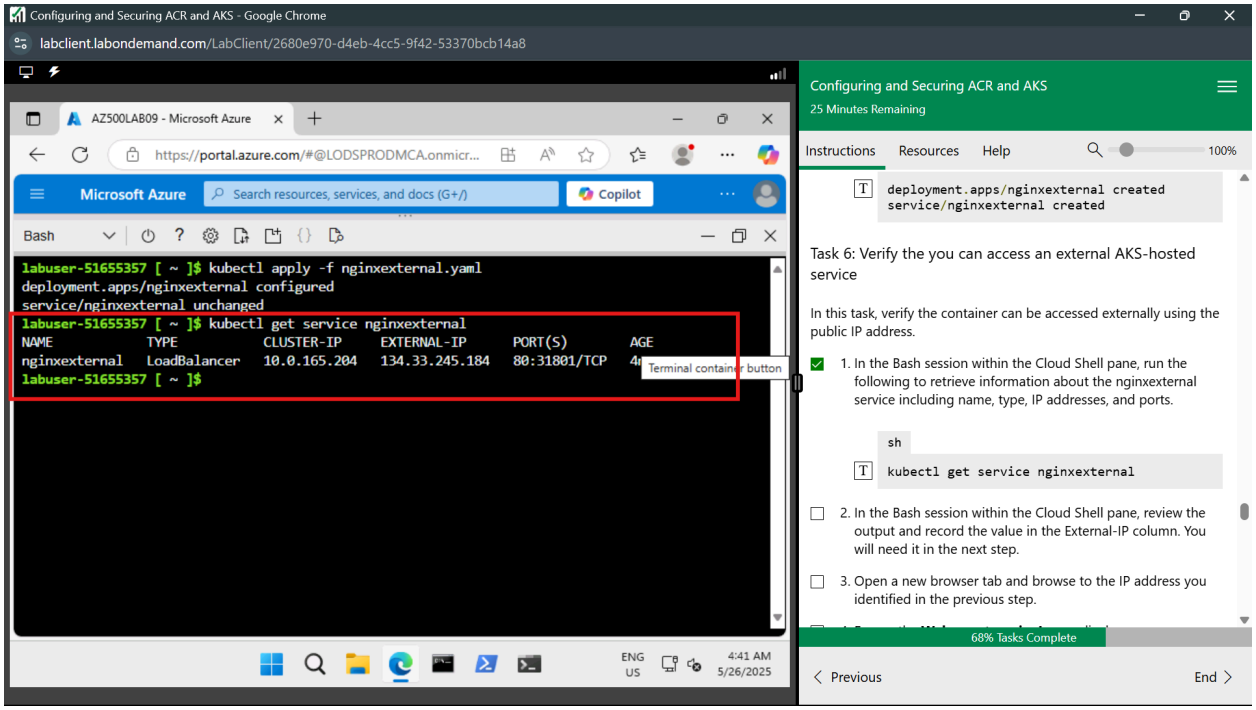
`kubectl apply -f nginxexternal.yaml`



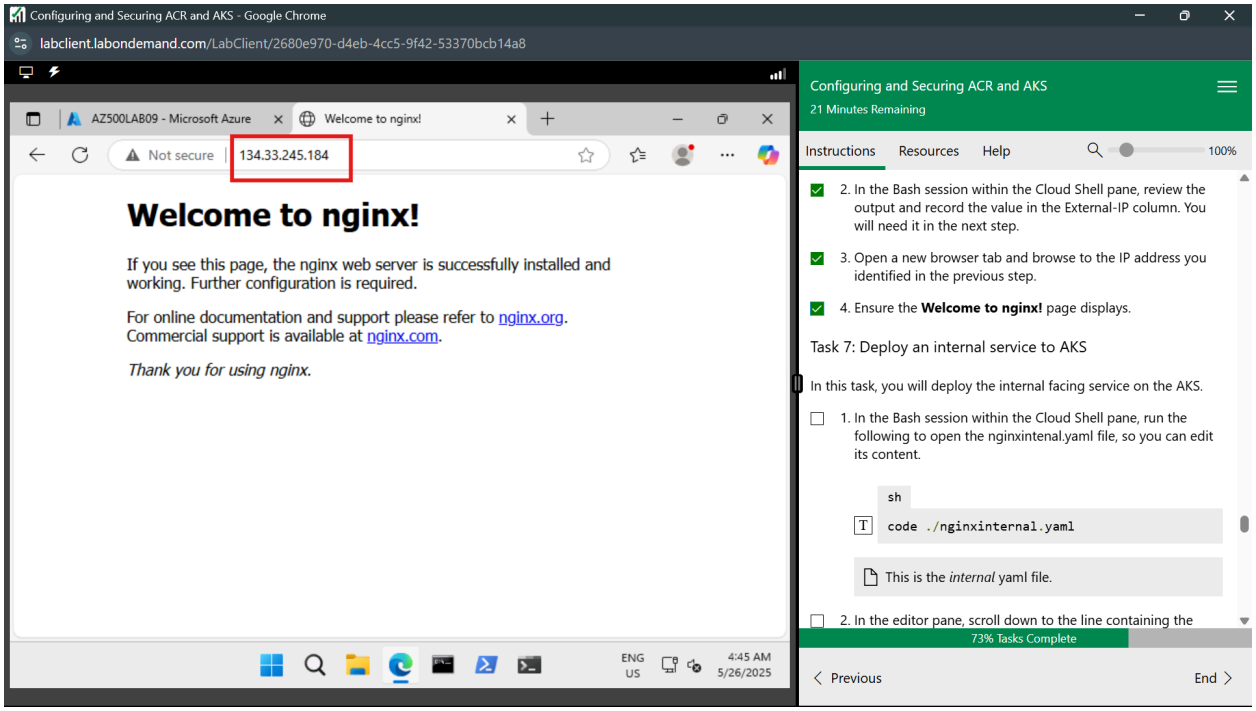
Verify that we can access an external AKS-hosted service

In this task, we verify that the container can be accessed externally using the public IP address. We will run the following to retrieve information about the nginxexternal service, including name, type, IP addresses, and ports.

`kubectl get service nginxexternal`

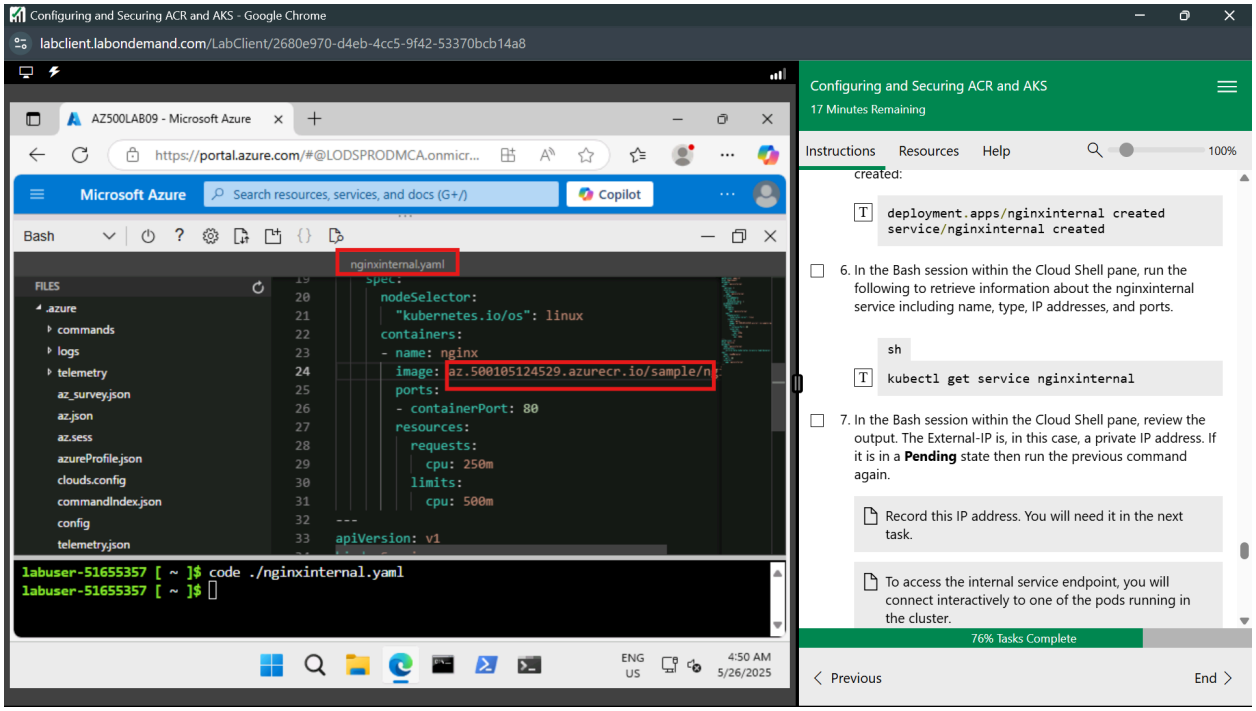


From the information, we can verify the external access through the web browser using the public IP.

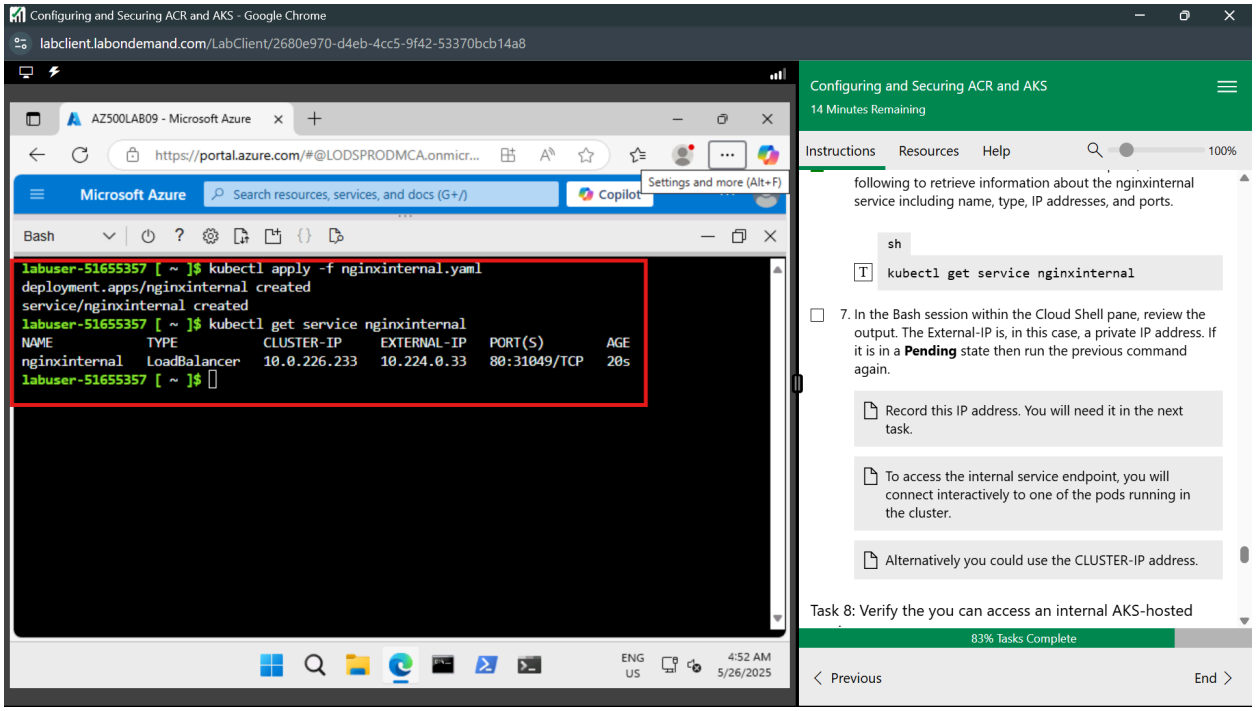


Deploy an internal service to AKS

In this task, we will deploy the internal-facing service on the AKS. First, we open the nginxinternal.yml file to edit the line containing our registry name as shown.



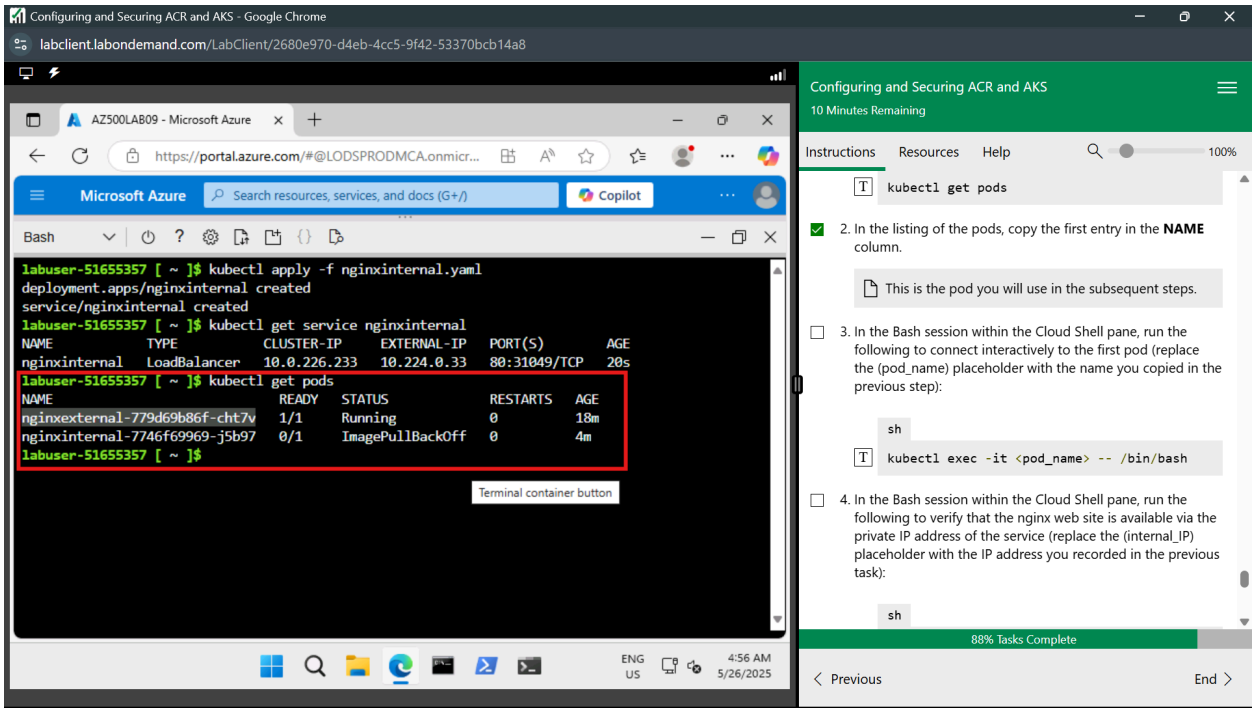
We can then apply the changes to our cluster and view its output as shown.



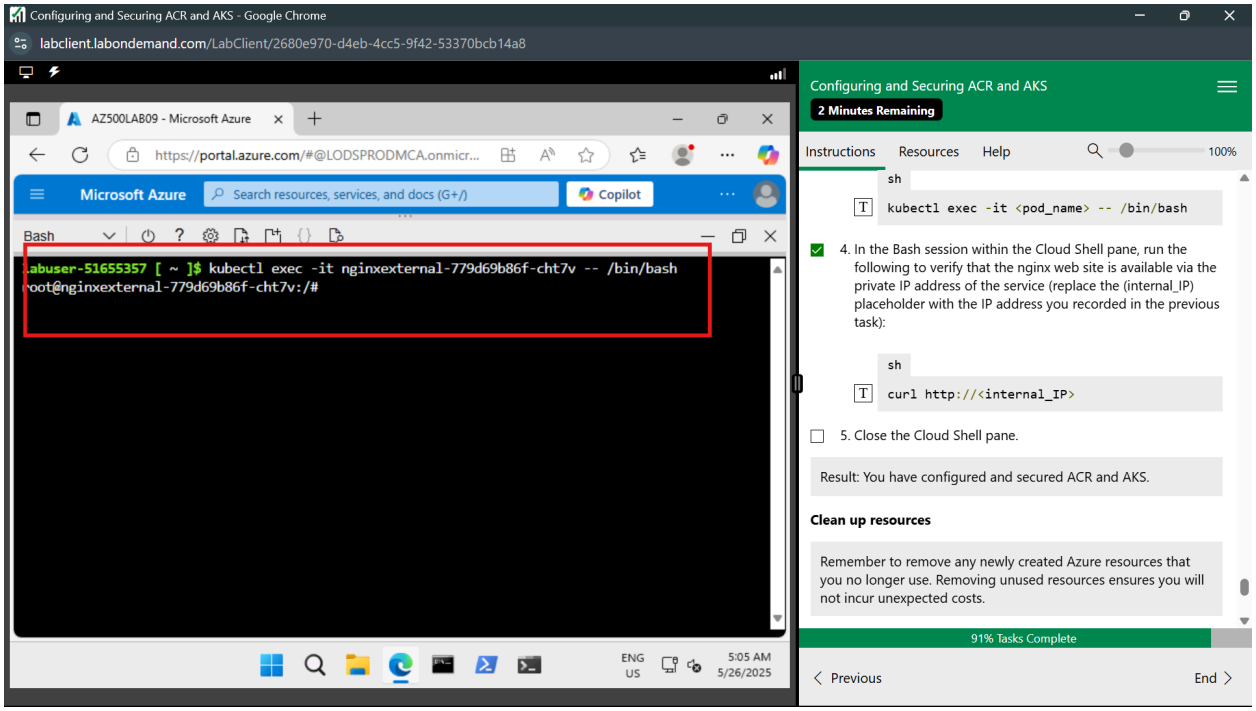
Verify that we can access an internal AKS-hosted service

In this task, we will use one of the pods running on the AKS cluster to access the internal service.

We get the pod information as follows using `kubectl get pods`

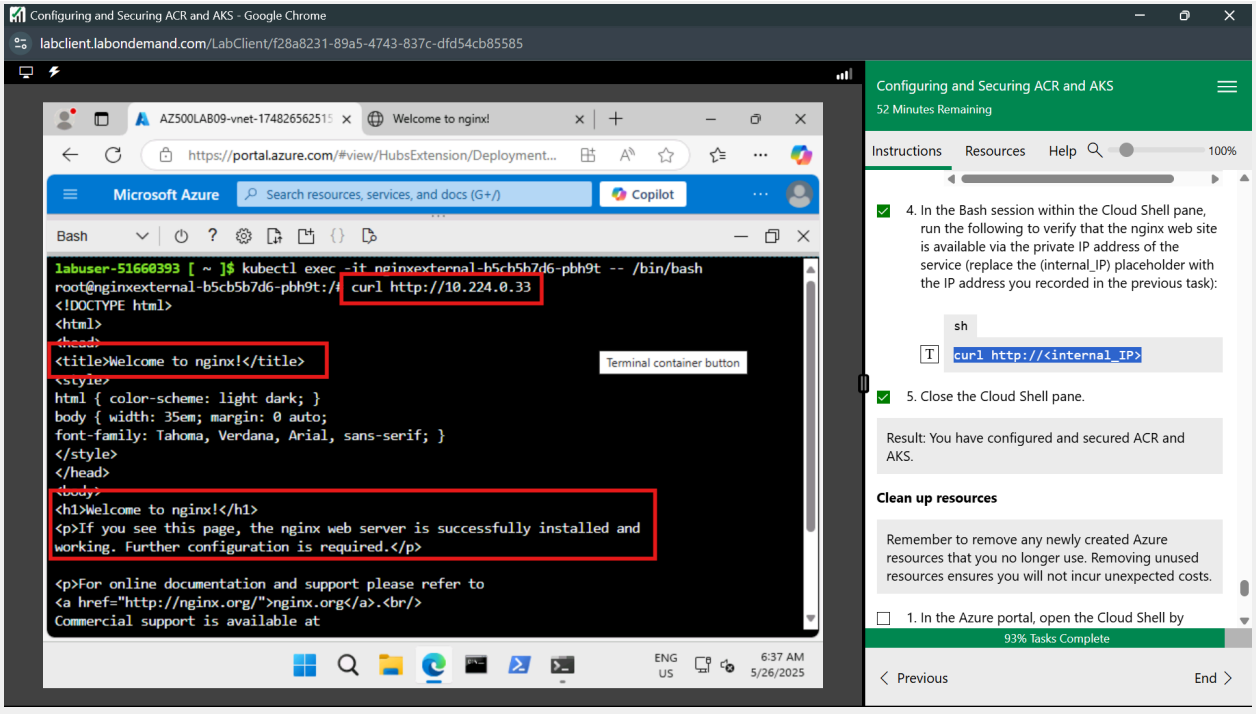


We then connect to the pod using bash as shown; the command is `kubectl exec -it <pod-name> - /bin/bash`



From the CLI inside the pod, we can verify that the nginx server is accessible internally using private IP addresses, as shown below;
The curl command was used to verify its accessibility.

```
curl http://<internal_IP>
```



Conclusion

In conclusion, this lab has equipped us with the skills and knowledge of configuring and securing ACR and AKS. We have deployed a container registry and configured a secure way that AKS can access the container images using role assignments. We have also configured public and privately accessible images.