

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

1 Hr 7 Min Remaining

Instructions Resources Help 100%

to manually create a virtual network. In the **Search Resources, Services, and Docs** text box at the top of the Azure portal page, type **Virtual networks** and press the **Enter** key. Click the **Create** button, and once that opens use the **AZ500LAB09 Resource Group** and name it **AZ500LAB09-vnet**. Once this is done you can use the script in **Task 4 step 2**.

Lab 04: Configuring and Securing ACR and AKS

Student lab manual

Lab scenario

You have been asked to deploy a proof of concept with Azure Container Registry and Azure Kubernetes Service.

3% Tasks Complete

Previous Next Create Give feedback

ENG US 3:14 AM 5/26/2025

## 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
```

```

"type": "Microsoft.ContainerService/ManagedClusters",
"upgradeSettings": null,
"windowsProfile": null,
"workloadAutoScalerProfile": {
  "keda": null,
  "verticalPodAutoscaler": null
}
}
labuser-51655357 [ ~ ]$ az group list --query "[?name=='AZ500LAB09']" -o table
Name          Location
-----
AZ500LAB09    eastus
labuser-51655357 [ ~ ]$ 

```

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
```

```

{
  "adminUserEnabled": false,
  "anonymousPullEnabled": false,
  "creationDate": "2025-05-26T10:32:32.342504+00:00",
  "dataEndpointEnabled": false,
  "dataEndpointHostNames": [],
  "encryption": {
    "keyVaultProperties": null,
    "status": "disabled"
  },
  "id": "/subscriptions/a22402a6-6755-4ecc-ac53-027c4335a0c7/resourceGroups/AZ500LAB09/providers/Microsoft.ContainerRegistry/registries/az500105124529",
  "identity": null,
  "location": "eastus",
  "loginServer": "az500105124529.azurecr.io",
  "metadataSearch": "Disabled",
  "name": "az500105124529",
  "networkRuleBypassOptions": "AzureServices",
  "networkRuleSet": null,
  "policies": {}
},

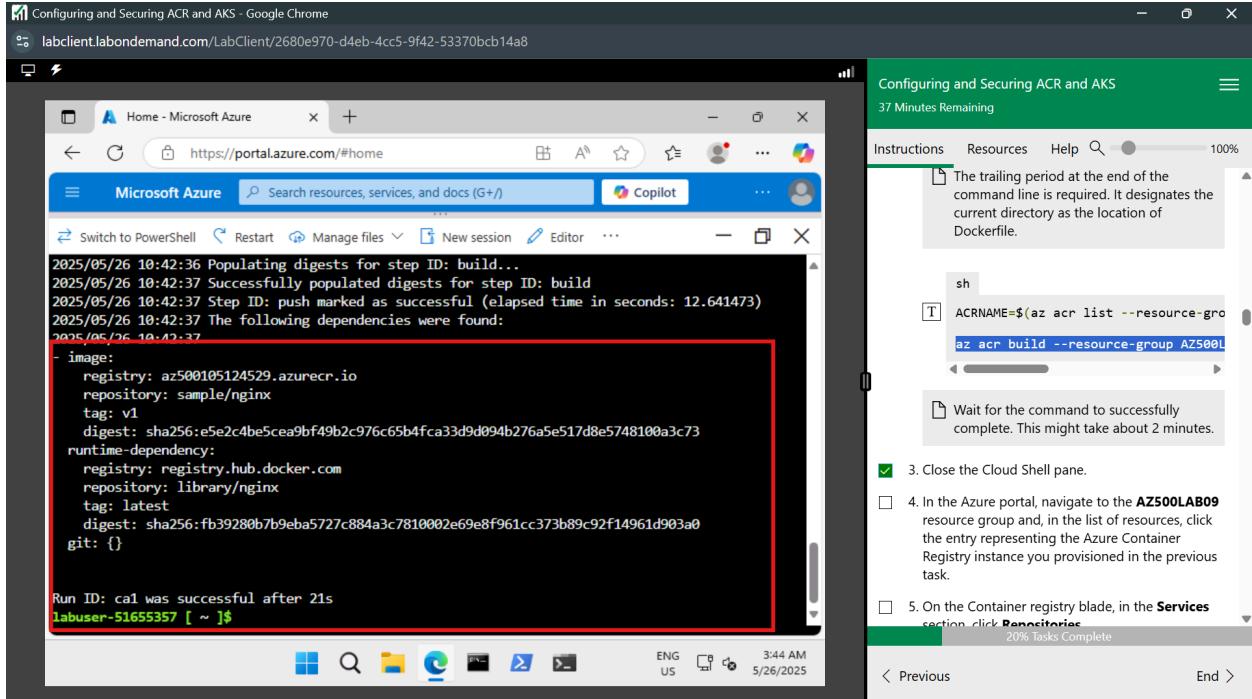
```

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**.

The screenshot shows a Microsoft Azure browser-based lab client. On the left, a Microsoft Edge window displays the Azure portal with the URL <https://portal.azure.com/#@LODSPRODMCA.onmicrosoft.com>. The page shows the 'az500105124529 | Repositories' section under 'Container registry'. A red box highlights the repository name 'sample/nginx'. On the right, a sidebar titled 'Configuring and Securing ACR and AKS' lists tasks numbered 5 through 8. Task 5 says 'On the Container registry blade, in the Services section, click Repositories.' Task 6 says 'Verify that the list of repositories includes the new container image named sample/nginx.' Task 7 says 'Click the sample/nginx entry and verify presence of the v1 tag that identifies the image version.' Task 8 says 'Click the v1 entry to view the image manifest.' A callout box for task 8 notes that the manifest includes sha256 digest, manifest creation date, and platform entries. Below the tasks, a pink box contains a reminder: 'Reminder: Please skip Task 3 steps 1-8. You will need to use this command in a Bash session within the Cloud Shell pane to create the Azure Kubernetes Service Cluster.' At the bottom of the sidebar, it says '25% Tasks Complete'.

## 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.

The screenshot shows a Microsoft Azure browser-based lab client. On the left, a Microsoft Edge window displays the Azure portal with the URL <https://portal.azure.com/#view/HubsExtension/>. The page shows the 'Resource groups' section. A red box highlights the terminal window where a Bash script is being run. The terminal output shows an error message: '(ResourceGroupNotFound) Resource group 'AZ500LAB09' could not be found. Code: ResourceGroupNotFound'. To the right, a sidebar titled 'Configuring and Securing ACR and AKS' with '1 Hr 13 Min Remaining' lists a task: 'Before running Task 4 step 2 you will need to manually create a virtual network. In the Search Resources, Services, and Docs text box at the top of the Azure portal page, type Virtual networks and press the Enter key. Click the Create button, and once that opens use the AZ500LAB09 Resource Group and name it [ ] AZ500LAB09-vnet. Once this is done you can use the script in Task 4 step 2.' Below the tasks, it says '3% Tasks Complete'.

## Lab 04: Configuring and Securing ACR and AKS

Student lab manual

Lab scenario

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
```

The screenshot shows the Microsoft Azure Cloud Shell interface and a task pane. The Cloud Shell window displays a terminal session with the following commands and output:

```
labuser-51655357 [ ~ ]$ az aks get-credentials --resource-group AZ500LAB09 --name MyKubernetesCluster
Merged "MyKubernetesCluster" as current context in /home/labuser-51655357/.kube/config
labuser-51655357 [ ~ ]$ kubectl get nodes
NAME           STATUS   ROLES      AGE   VERSION
aks-agentpool-30198516-0   Ready    <none>   50m   v1.31.8
labuser-51655357 [ ~ ]$
```

The 'kubectl get nodes' command output is highlighted with a red box. The task pane on the right shows steps 13 and 14, which correspond to the commands shown in the terminal.

## 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
```

Configuring and Securing ACR and AKS - Google Chrome  
labclient.labondemand.com/LabClient/2680e970-d4eb-4cc5-9f42-53370bcb14a8

AZ500LAB09 - Microsoft Azure https://portal.azure.com/#@LODSPRODMCA.onmicrosoft.com

labuser-51655357 [ ~ ]\$ az role assignment create --assignee \$AKS\_MANAGED\_ID --role "Contributor" --scope \$AKS\_VNET\_ID

```
{  
    "condition": null,  
    "conditionVersion": null,  
    "createdBy": null,  
    "createdOn": "2025-05-26T11:11:48.486450+00:00",  
    "delegatedManagedIdentityResourceId": null,  
    "description": null,  
    "id": "/subscriptions/a22402a6-6755-4ecc-ac53-027c4335a0c7/resourceGroups/MC_AZ500LAB09_MyKubernetesCluster_eastus/providers/Microsoft.Network/virtualNetworks/aks-vnet-30198516/providers/Microsoft.Authorization/roleAssignments/bd2647e0-774f-48ca-bdc2-f29b6ed54233",  
    "name": "bd2647e0-774f-48ca-bdc2-f29b6ed54233",  
    "principalId": "e51b445d-2fd4-4e54-8a8b-385ac1a77a4f",  
    "principalType": "ServicePrincipal",  
    "resourceGroup": "MC_AZ500LAB09_MyKubernetesCluster_eastus",  
    "roleDefinitionId": "/subscriptions/a22402a6-6755-4ecc-ac53-027c4335a0c7/providers/Microsoft.Authorization/roleDefinitions/b24988ac-6180-42a0-ab88-20f7382dd24c",  
    "scope": "/subscriptions/a22402a6-6755-4ecc-ac53-027c4335a0c7/resourceGroups/MC_AZ500LAB09_MyKubernetesCluster_eastus/providers/Microsoft.Network/virtualNetworks/aks-vnet-30198516",  
}
```

Configuring and Securing ACR and AKS  
50 Minutes Remaining

Instructions Resources Help 100%

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

Task 5: Deploy an external service to AKS

In this task, you will download the Manifest files, edit the YAML file, and apply your changes to the cluster.

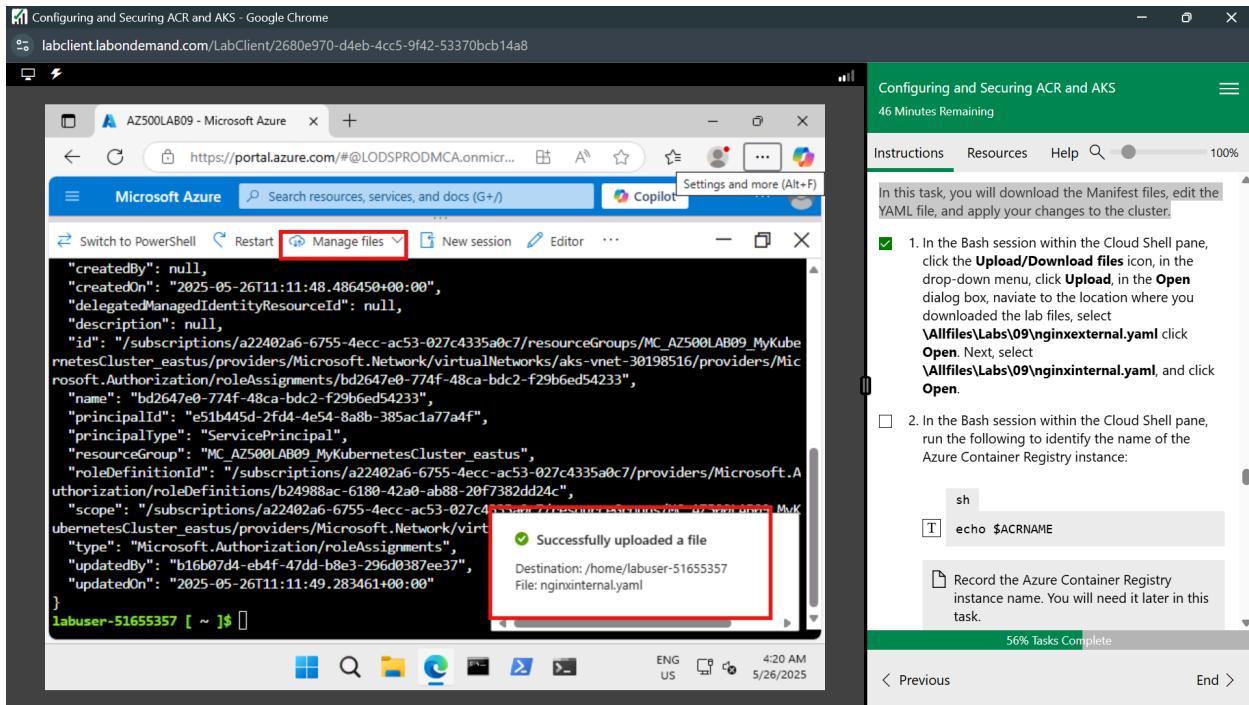
1. In the Bash session within the Cloud Shell pane, click the **Upload/Download files** icon, in the drop-down menu, click **Upload**, in the **Open** dialog box, navigate to the location where you downloaded the lab files, select **\Allfiles\Labs\09\nginxexternal.yaml** click **Open**. Next, select **\Allfiles\Labs\09\nginxinternal.yaml**, and click **Open**.

56% Tasks Complete

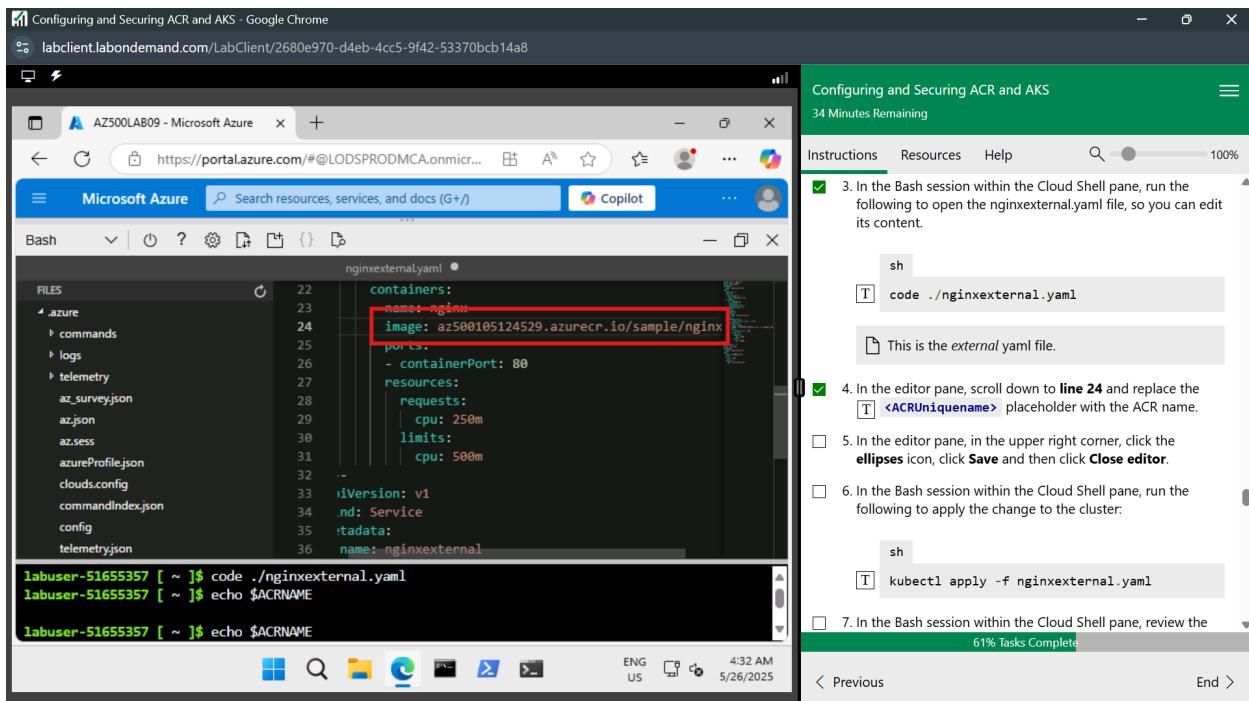
< Previous End >

# 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.yaml 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
```

The screenshot shows the Azure Cloud Shell interface. On the left, there is a terminal window titled 'AZ500LAB09 - Microsoft Azure' displaying the command output:

```
labuser-51655357 [ ~ ]$ kubectl apply -f nginxexternal.yaml
deployment.apps/nginxexternal configured
service/nginxexternal unchanged
labuser-51655357 [ ~ ]$
```

A red box highlights this terminal output. To the right of the terminal is a 'Task' pane for 'Configuring and Securing ACR and AKS'. The task is titled 'Task 6: Verify the you can access an external AKS-hosted service'. It contains a step 7 with a checked checkbox and a command input field:

- 7. In the Bash session within the Cloud Shell pane, review the output of the command you run in the previous task to verify that the deployment and the corresponding service have been created.

```
sh
[T] kubectl apply -f nginxexternal.yaml
```

Below this, another step 1 is listed:

- 1. In the Bash session within the Cloud Shell pane, run the following to retrieve information about the nginxexternal service including name, type, IP addresses, and ports.

```
sh
[T] kubectl get service nginxexternal
```

The task progress bar at the bottom indicates '66% Tasks Complete'.

## 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
```

The screenshot shows a Microsoft Azure Cloud Shell interface. On the left, a terminal window displays the command `kubectl apply -f nginxexternal.yaml` followed by `deployment.apps/nginxexternal configured` and `service/nginxexternal unchanged`. Below this, another command `kubectl get service nginxexternal` is run, showing a table with the following data:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
nginxexternal	LoadBalancer	10.0.165.204	134.33.245.184	80:31801/TCP	4m

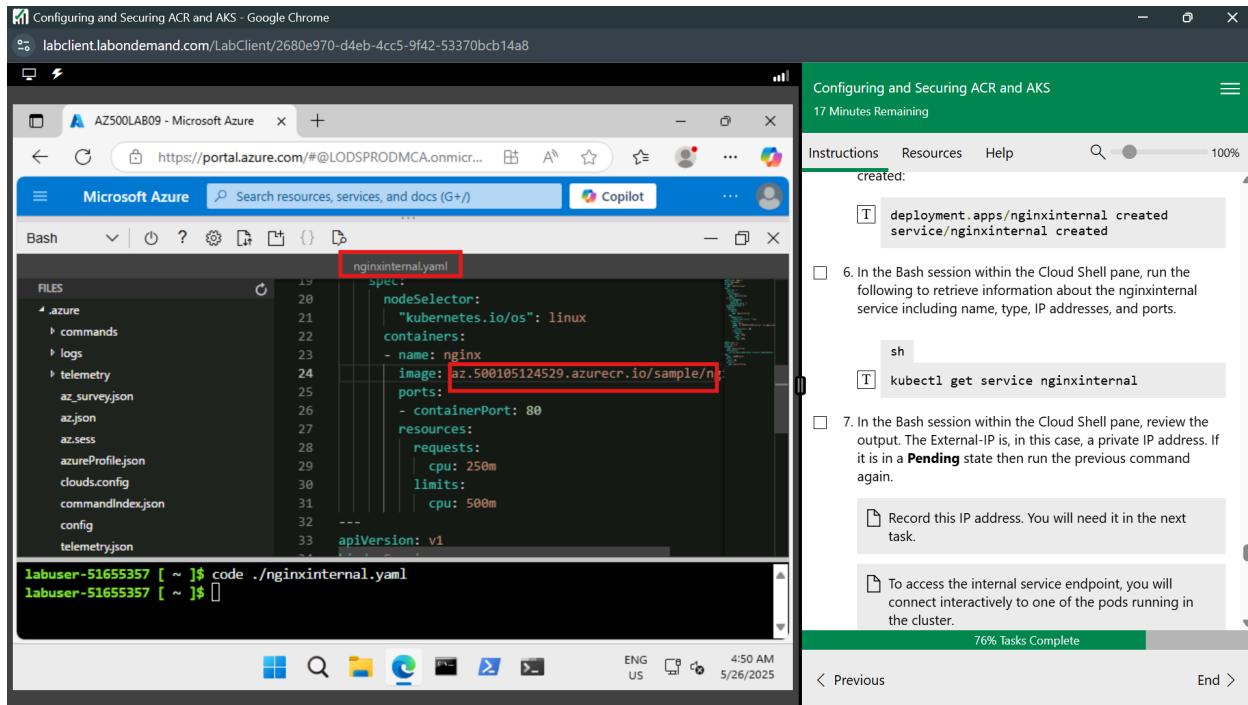
A red box highlights the table output. To the right, a task pane titled "Configuring and Securing ACR and AKS" shows the status "deployment.apps/nginxexternal created" and "service/nginxexternal created". Task 6 is listed: "Verify you can access an external AKS-hosted service". The instructions state: "In this task, verify the container can be accessed externally using the public IP address." Step 1 is checked: "In the Bash session within the Cloud Shell pane, run the following to retrieve information about the nginxexternal service including name, type, IP addresses, and ports." Step 2 is unchecked: "In the Bash session within the Cloud Shell pane, review the output and record the value in the External-IP column. You will need it in the next step." Step 3 is unchecked: "Open a new browser tab and browse to the IP address you identified in the previous step." A progress bar at the bottom indicates "68% Tasks Complete".

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

The screenshot shows a Microsoft Azure Cloud Shell interface. On the left, a browser window displays the URL <http://134.33.245.184>. A red box highlights the URL. The page content is "Welcome to nginx!". Below it, text reads: "If you see this page, the nginx web server is successfully installed and working. Further configuration is required. For online documentation and support please refer to [nginx.org](http://nginx.org). Commercial support is available at [nginx.com](http://nginx.com). Thank you for using nginx." To the right, a task pane titled "Configuring and Securing ACR and AKS" shows the status "21 Minutes Remaining". Task 2 is checked: "In the Bash session within the Cloud Shell pane, review the output and record the value in the External-IP column. You will need it in the next step." Task 3 is checked: "Open a new browser tab and browse to the IP address you identified in the previous step." Task 4 is checked: "Ensure the Welcome to nginx! page displays." Task 7 is listed: "Deploy an internal service to AKS". The instructions state: "In this task, you will deploy the internal facing service on the AKS." Step 1 is unchecked: "In the Bash session within the Cloud Shell pane, run the following to open the nginxinternal.yaml file, so you can edit its content." Step 2 is unchecked: "In the editor pane, scroll down to the line containing the". A progress bar at the bottom indicates "73% Tasks Complete".

# Deploy an internal service to AKS

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

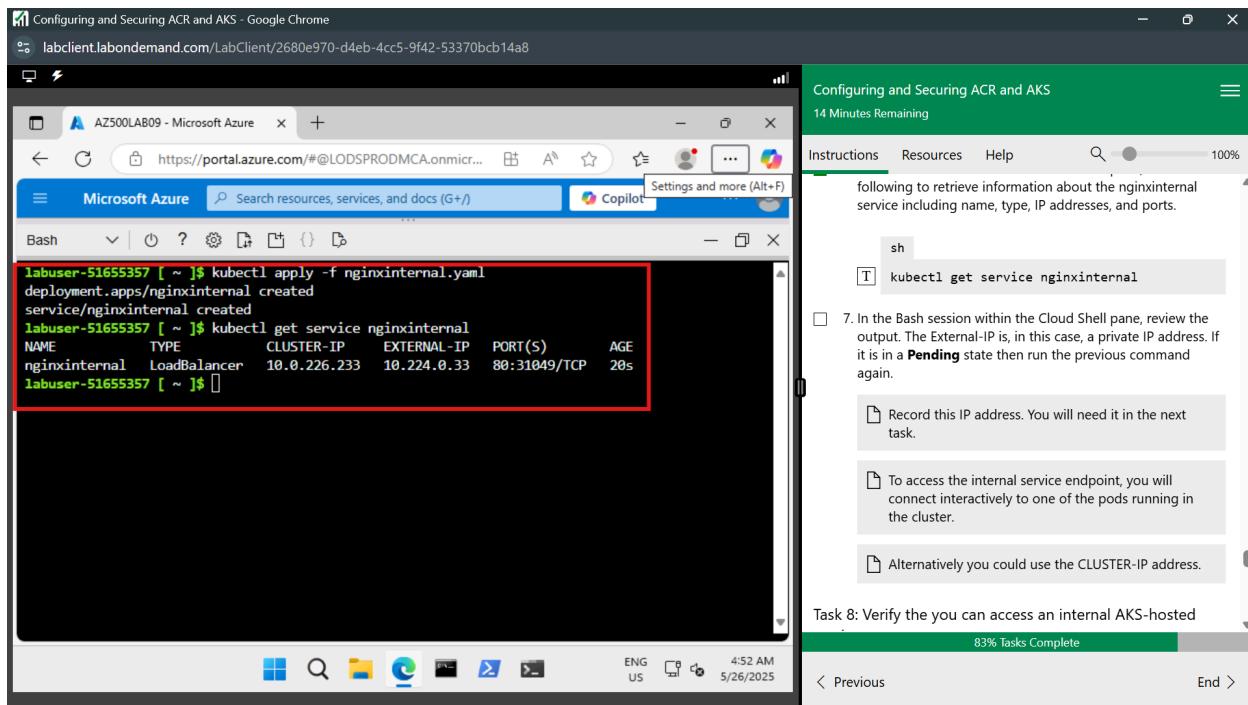


The screenshot shows a Microsoft Azure Cloud Shell interface. On the left, there's a file browser pane showing a directory structure with files like .azure, commands, logs, telemetry, az\_survey.json, az.json, azsесс, azureProfile.json, clouds.config, commandIndex.json, config, and telemetry.json. In the center, a code editor window displays the contents of the nginxinternal.yaml file. The 'image' field is highlighted with a red box. The file content is as follows:

```
spec:
  nodeSelector:
    "kubernetes.io/os": linux
  containers:
  - name: nginx
    image: az.500105124529.azurecr.io/sample/nginxinternal:v1
    ports:
    - containerPort: 80
      resources:
        requests:
          cpu: 250m
        limits:
          cpu: 500m
  apiVersion: v1
```

At the bottom of the code editor, the terminal shows the command: `labuser-51655357 [ ~ $] code ./nginxinternal.yaml`. To the right of the code editor is a task pane titled "Configuring and Securing ACR and AKS". It shows a progress bar at 76% Tasks Complete. Step 6 is listed: "In the Bash session within the Cloud Shell pane, run the following to retrieve information about the nginxinternal service including name, type, IP addresses, and ports." Below it is a command: `sh kubectl get service nginxinternal`.

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



The screenshot shows the same Microsoft Azure Cloud Shell interface. The terminal window now displays the output of the `kubectl apply -f nginxinternal.yaml` command. The output is highlighted with a red box. It shows the deployment and service being created. The service output includes the following table:

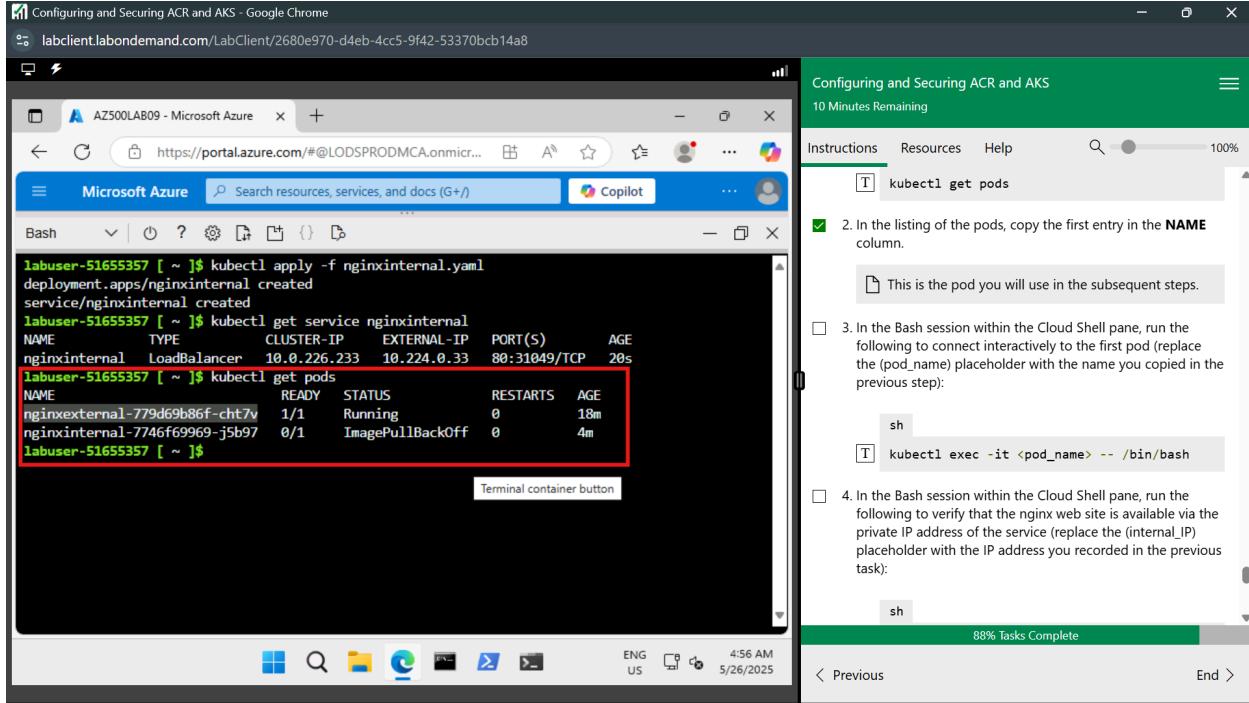
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
nginxinternal	LoadBalancer	10.0.226.233	10.224.0.33	80:31049/TCP	20s

To the right of the terminal is the same task pane as before. Step 7 is listed: "In the Bash session within the Cloud Shell pane, review the output. The External-IP is, in this case, a private IP address. If it is in a Pending state then run the previous command again." Below it are two tips: "Record this IP address. You will need it in the next task." and "To access the internal service endpoint, you will connect interactively to one of the pods running in the cluster." At the bottom of the task pane, the progress bar shows 83% Tasks Complete.

## 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.

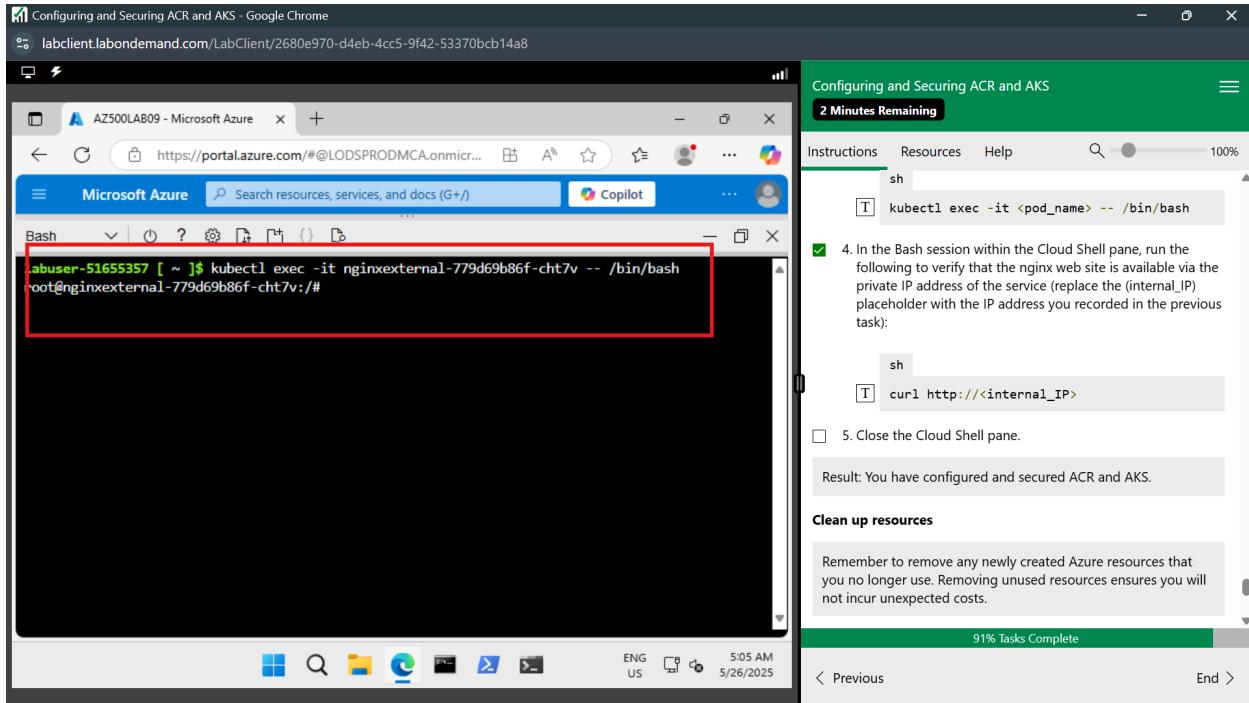
We get the pod information as follows using `kubectl get pods`



The screenshot shows a Microsoft Azure Cloud Shell interface. On the left, a terminal window displays the output of the `kubectl get pods` command. The output shows two pods: `nginxexternal-779d69b86f-cht7v` and `nginxinternal-7746f69969-j5b97`. The second pod, `nginxinternal-7746f69969-j5b97`, is highlighted with a red box. On the right, a task pane titled "Configuring and Securing ACR and AKS" shows step 2: "In the listing of the pods, copy the first entry in the NAME column." Below it, a note says "This is the pod you will use in the subsequent steps."

```
labuser-51655357 [ ~ ]$ kubectl apply -f nginxinternal.yaml
deployment.apps/nginxinternal created
service/nginxinternal created
labuser-51655357 [ ~ ]$ kubectl get service nginxinternal
NAME          TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
nginxinternal LoadBalancer  10.0.226.233  10.224.0.33  80:31049/TCP  20s
labuser-51655357 [ ~ ]$ kubectl get pods
NAME                  READY   STATUS      RESTARTS   AGE
nginxexternal-779d69b86f-cht7v  1/1     Running    0          18m
nginxinternal-7746f69969-j5b97  0/1     ImagePullBackOff  0          4m
labuser-51655357 [ ~ ]$
```

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



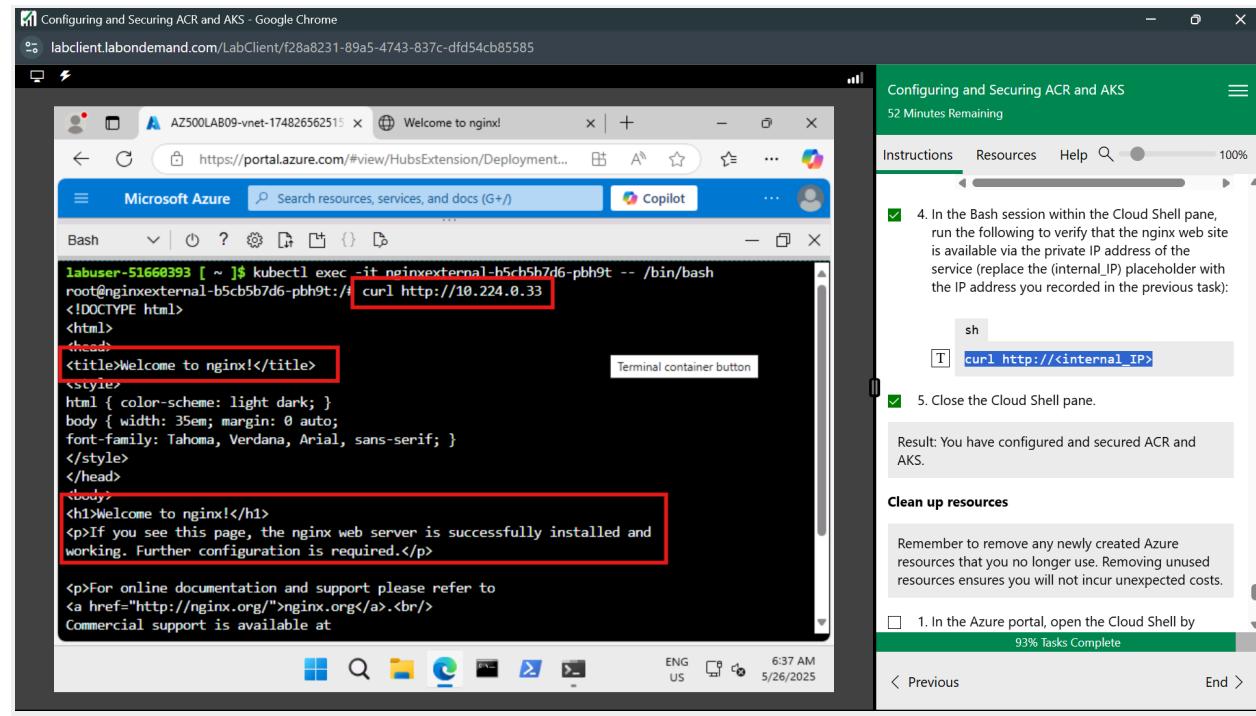
The screenshot shows a Microsoft Azure Cloud Shell interface. On the left, a terminal window displays the output of the `kubectl exec -it nginxexternal-779d69b86f-cht7v -- /bin/bash` command. The entire terminal output is highlighted with a red box. On the right, a task pane titled "Configuring and Securing ACR and AKS" shows step 4: "In the Bash session within the Cloud Shell pane, run the following to verify that the nginx web site is available via the private IP address of the service (replace the (internal\_IP) placeholder with the IP address you recorded in the previous task)." Below it, a note says "curl http://<internal\_IP>" and "Result: You have configured and secured ACR and AKS." A "Clean up resources" section at the bottom provides instructions to remove unused resources.

```
labuser-51655357 [ ~ ]$ kubectl exec -it nginxexternal-779d69b86f-cht7v -- /bin/bash
root@nginxexternal-779d69b86f-cht7v:~#
```

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.