Contents

[**Microsoft Azure** 1](file:///C:\Users\gmoha\AppData\Local\Microsoft\Windows\INetCache\IE\4MGBXZ64\MS%20Azure%20Cloud%203.0_Module%208_Task_Containers%20(1)%5b1%5d.docx#_Toc144167022)

[1. agenda 3](file:///C:\Users\gmoha\AppData\Local\Microsoft\Windows\INetCache\IE\4MGBXZ64\MS%20Azure%20Cloud%203.0_Module%208_Task_Containers%20(1)%5b1%5d.docx#_Toc144167023)

[2. ACCEPTANCE CRITERIA 3](file:///C:\Users\gmoha\AppData\Local\Microsoft\Windows\INetCache\IE\4MGBXZ64\MS%20Azure%20Cloud%203.0_Module%208_Task_Containers%20(1)%5b1%5d.docx#_Toc144167024)

[3. Task 3](file:///C:\Users\gmoha\AppData\Local\Microsoft\Windows\INetCache\IE\4MGBXZ64\MS%20Azure%20Cloud%203.0_Module%208_Task_Containers%20(1)%5b1%5d.docx#_Toc144167025)

[4. TASK RESULT 5](file:///C:\Users\gmoha\AppData\Local\Microsoft\Windows\INetCache\IE\4MGBXZ64\MS%20Azure%20Cloud%203.0_Module%208_Task_Containers%20(1)%5b1%5d.docx#_Toc144167026)

[5. USEFUL LINKS 6](file:///C:\Users\gmoha\AppData\Local\Microsoft\Windows\INetCache\IE\4MGBXZ64\MS%20Azure%20Cloud%203.0_Module%208_Task_Containers%20(1)%5b1%5d.docx#_Toc144167027)

# Agenda

The problem of an application failing to run correctly when moved from one environment to another is as old as software development itself. Such problems typically arise due to differences in configuration underlying library requirements and other dependencies.

Containers address this problem by providing a lightweight, immutable infrastructure for application packaging and deployment. An application or service, its dependencies, and its configuration are packaged together as a container image. The containerized application can be tested as a unit and deployed as a container image instance to the host operating system.

There are many options for teams to build and deploy cloud native and containerized applications on Azure. There are many options for teams to build and deploy cloud native and containerized applications on Azure.

In this task, you will get acquainted with the following Azure Container services:

* Azure Container Registry
* Azure Container Instance
* Web App for Containers
* Azure Container Apps
* Azure Kubernetes Service

Every Azure resource has pros and cons. There's no perfect solution for every use case and every team.

In this task we are asking you to build production design and reproduce the typical steps which you will be in touch working some Azure Container resources. In the task you need to automate provisioning for mentioned above azure resources by means of Terraform. You will build and deploy a simple application in these Azure services via Terraform as well.

Please use useful links to do your homework successfully

# ACCEPTANCE CRITERIA

1. Homework corresponds to task details, requirements and attachment recommendations.
2. Terraform configuration has normal view and meets all requirements and best practices studied on previous modules.
3. Terraform configuration for each Azure resources must be as a module.
4. A tutor should be able to execute your Terraform configuration without modifications. Any hardcode that can affect re-run must be fixed. If your code is related some how to OS (or shell), inform about it your tutor and provide description why you did.
5. Minimal Terraform version is 1.5.0.
6. All resource names should use a common name pattern and be aligned with Azure abbreviation examples. No resource names are hardcoded
7. TF configuration has usable outputs, that are used to access to resources, such as storage account name, vault name, cdn endpoint, etc.

# Task

1. Create Terraform configuration that provisions the following resources:
   1. Azure Redis Cache
   2. Azure Container Registry
   3. Azure Key Vault
2. Redis password and Redis Url must be saved in Key Vault as a secrets. Add in Terraform configuration block that builds Docker image using the Dockerfile provided below, and upload built image to Azure Container Registry. Any sensitive values (such as ACR admin key should be provided dynamically)
3. Update your Terraform configuration to create Azure Container Instance which will host a container from the image built earlier. All sensitive variables must be provided to ACI as secure\_environment\_variables (when it is possible).

Define “CREATOR” variable for container with value “Azure\_Container\_Instance”

Keep in mind that the Docker container works in pair with Azure Redis, and required parameters must be provided to container during the initialization (see app.py)

Application must be accessible via HTTP/HTTPs from the internet.

1. Update your Terraform configuration to create Azure WebApp for Containers Instance which will host a container from the image built earlier. Sensitive variables for this WebApp must be stored in Azure KeyVault and provided to WebAPP during the container initialization (see useful links).

Define “CREATOR” variable for container with value “Azure\_Web\_APP”.

Application must be accessible via HTTP/HTTPs from the internet.

1. Update your Terraform configuration to create Azure Container APP Instance which will host a container from the image built earlier.

Define “CREATOR” variable for container with value “Azure\_Container\_App”.

Application must be accessible via HTTP/HTTPs from the internet.

1. Update your Terraform configuration to create Azure Kubernetes Service integrated with Azure KeyVault created above, and which will host a container from the image built earlier.

Define “CREATOR” variable for container with value “K8S”.

Pods in k8s should obtain secrets form KeyVault during the initialization. Use “Access with a user-assigned managed identity” approach. Optionally you can use Azure AD workload identity to get access to KeyVault instead of UMI.

1. Deploy the image built earlier to k8s. You can update your Terraform configuration to deploy Docker image(optionally) or just connect to k8s and deploy using k8s manifests. Sensitive variable must be stored in KeyVault in any case. Application must available from the internet at least via IP address.

# TASK RESULT

A result of this task is a running applications on different Azure services such as ACI, WebAPP, Container APP and AKS that are available by IP address or URLs.

## USEFUL LINKS

[Dockerfile reference | Docker Docs](https://docs.docker.com/engine/reference/builder/)

[Provisioner: local-exec | Terraform | HashiCorp Developer](https://developer.hashicorp.com/terraform/language/resources/provisioners/local-exec)

[Quickstart - Create registry in portal - Azure Container Registry | Microsoft Learn](https://learn.microsoft.com/en-us/azure/container-registry/container-registry-get-started-portal?tabs=azure-cli)

[Welcome to Flask — Flask Documentation (2.3.x) (palletsprojects.com)](https://flask.palletsprojects.com/en/2.3.x/)

[What is Azure Key Vault? | Microsoft Learn](https://learn.microsoft.com/en-us/azure/key-vault/general/basic-concepts)

[What is Azure Cache for Redis? | Microsoft Learn](https://learn.microsoft.com/en-us/azure/azure-cache-for-redis/cache-overview)

[Quickstart: Create an Azure Container Instance with a public IP address using Terraform - Azure Container Instances | Microsoft Learn](https://learn.microsoft.com/en-us/azure/container-instances/container-instances-quickstart-terraform)

[Deploy and run a containerized web app with Azure App Service - Training | Microsoft Learn](https://learn.microsoft.com/en-us/training/modules/deploy-run-container-app-service/)

[Azure Container Apps overview | Microsoft Learn](https://learn.microsoft.com/en-us/azure/container-apps/overview)

[Quickstart: Create an Azure Kubernetes Service (AKS) cluster by using Terraform - Azure Kubernetes Service | Microsoft Learn](https://learn.microsoft.com/en-us/azure/aks/learn/quick-kubernetes-deploy-terraform?tabs=azure-cli)

[null\_resource | Resources | hashicorp/null | Terraform | Terraform Registry](https://registry.terraform.io/providers/hashicorp/null/latest/docs/resources/resource)

[azurerm\_container\_app - Cannot deploy container with ingress enabled · Issue #20435 · hashicorp/terraform-provider-azurerm · GitHub](https://github.com/hashicorp/terraform-provider-azurerm/issues/20435)

[Use Key Vault references - Azure App Service | Microsoft Learn](https://learn.microsoft.com/en-us/azure/app-service/app-service-key-vault-references?tabs=azure-cli)

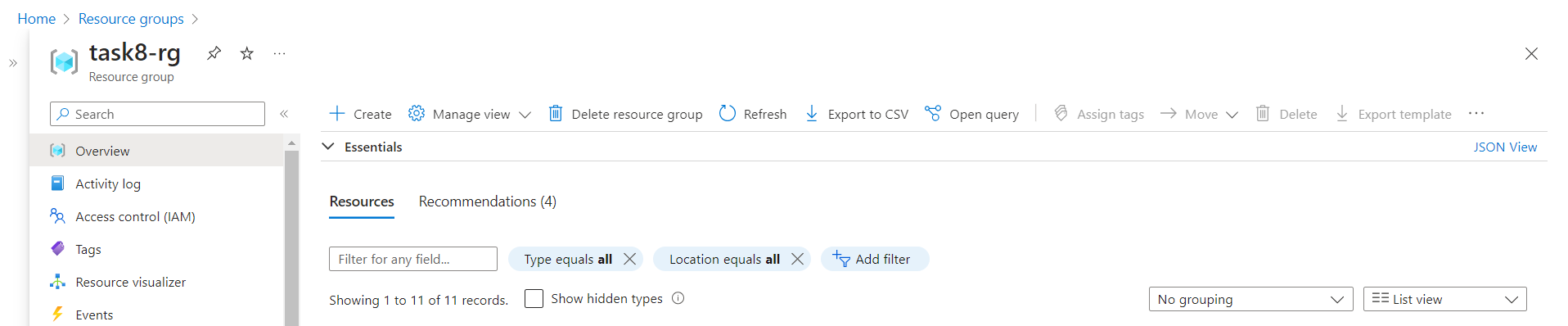
[Use the Azure Key Vault Provider for Secrets Store CSI Driver for Azure Kubernetes Service (AKS) secrets - Azure Kubernetes Service | Microsoft Learn](https://learn.microsoft.com/en-us/azure/aks/csi-secrets-store-driver)

[Provide an access identity to the Azure Key Vault Provider for Secrets Store CSI Driver for Azure Kubernetes Service (AKS) secrets - Azure Kubernetes Service | Microsoft Learn](https://learn.microsoft.com/en-us/azure/aks/csi-secrets-store-identity-access#access-with-a-user-assigned-managed-identity)

[Provide an access identity to the Azure Key Vault Provider for Secrets Store CSI Driver for Azure Kubernetes Service (AKS) secrets - Azure Kubernetes Service | Microsoft Learn](https://learn.microsoft.com/en-us/azure/aks/csi-secrets-store-identity-access#access-with-an-azure-ad-workload-identity)

**Resource Group:**

In Azure Cloud Resource Group is the primary thing we need to create to store all the resources . Create the Resource Group using Terraform modules. Click on GitHub Url to check the Module Source code: https://github.com/gmk1995/azure-terraform-modules



**Key Vault and Access Policies:**

As Per Task we need to create the Key Vault to store Redis Cache Hostname and Access Keys. We need to create Access Policy also to access the key vault secrets. Click on below GitHub Url to access the Terraform Module source code: https://github.com/gmk1995/azure-terraform-modules  
  
A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Azure Redis Cache:**

We need to create the Azure redis cache for store the cache memory of number of time we have visited the web app. Click on the below link to get the Terraform source code module.

https://github.com/gmk1995/azure-terraform-modules

**Note**: We have stored the Redis Hostname and Access Key on the Key vault Secrets using terraform code.

A screenshot of a computer

Description automatically generated

**Azure Redis Cache Hostname and Access Key Stored in the Azure Key Vault Secret:**

A screenshot of a computer

Description automatically generated

**Azure Container Registry:**

To store the images we need to create the Azure container registry using Terraform code. Click on the link to access the Terraform module source code:

https://github.com/gmk1995/azure-terraform-modules

A screenshot of a computer

Description automatically generated

**Docker builds Image and Pushes it to the Azure container Registry:**

We need to build the images from the Dockerfile using terraform code and then we need to push the image to the previously created Azure Container Registry. Click here to access the Terraform Module Source Code https://github.com/gmk1995/azure-terraform-modules  
  


A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Azure Container Instance:**

We need to Create the Azure Container Instance to Create the Container from the Previously build and pushed image from the Azure container using Terraform code. Click here to access the Terraform Module Source Code Module: https://github.com/gmk1995/azure-terraform-modules

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Access the Deployed Container through the web Using the IP address with Port Number of the Container: https://github.com/gmk1995/azure-terraform-modules

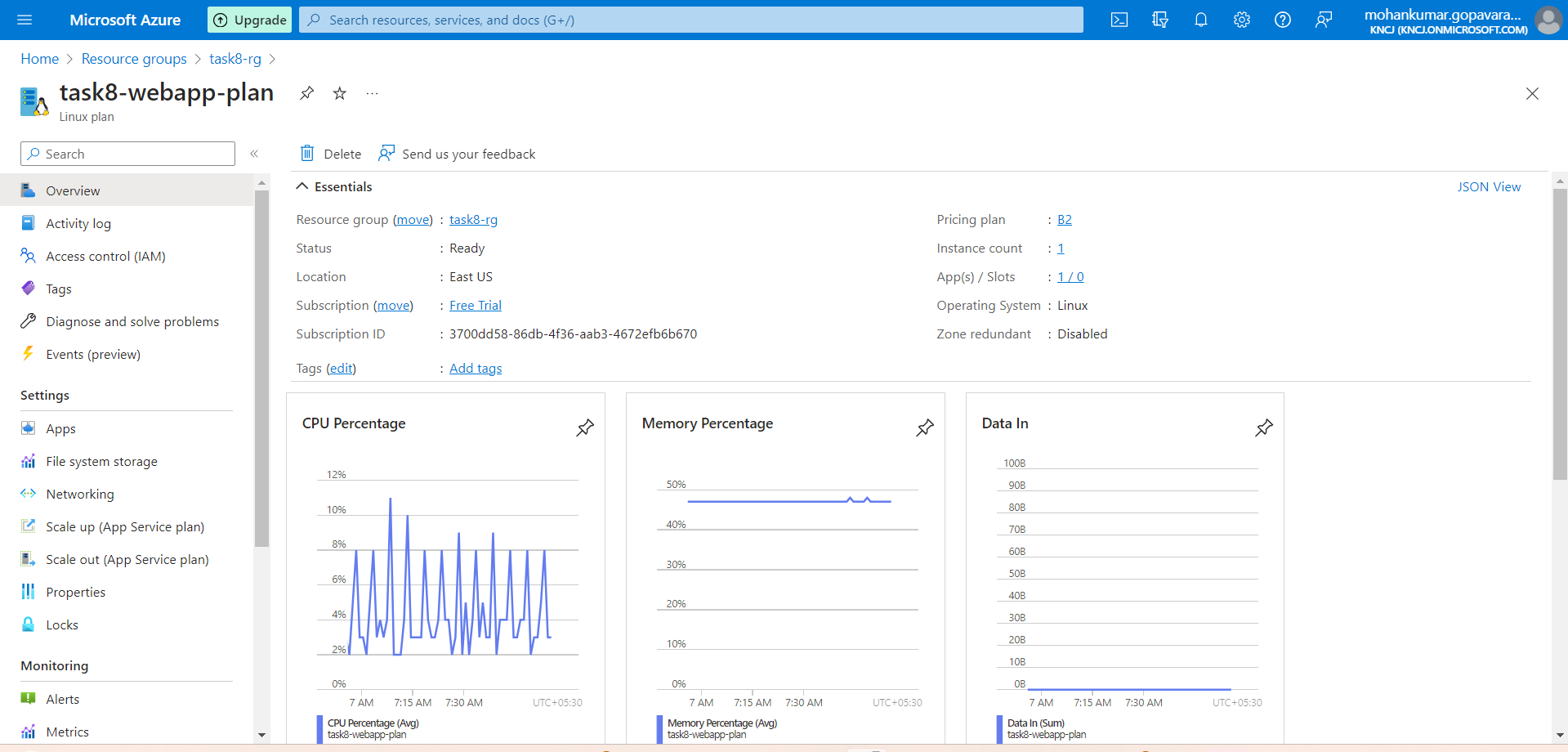
A white rectangular object with a black border

Description automatically generated with medium confidence

**Azure Web Apps:**

Previously build and pushed image to the Azure Container Registry used to Create the Azure Web App. Here we need to create the Azure Web App Service Plan First and then Azure Web App. We need to Assign a Role to the Web App to Pull the Images from Azure Container Registry. Click on the below link to access Terraform Module Source Code. https://github.com/gmk1995/azure-terraform-modules

Service Plan:



Web App:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Access the Web App from the web browser using the Default Domain Name:



**Azure Container App:**

First, we need to create a Log Analytic Workspace and Azure Container App Environment and then Azure Container App. Previously build and pushed docker image from Azure Container Registry need to take to create a Azure Container App using terraform Code. Click here to access the Terraform Module Source Code: https://github.com/gmk1995/azure-terraform-modules

**Log Analytics workspace:**

A screenshot of a computer

Description automatically generated

Container Apps Environment:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Container App:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Access the container build in the container apps using application url**

A close up of a computer screen

Description automatically generated

**Azure Kubernetes Services:**

To create Azure Kubernetes Service integrated with Azure KeyVault created above, and which will host a container from the image built earlier.

Define “CREATOR” variable for container with value “K8S”.

Pods in k8s should obtain secrets form KeyVault during the initialization. Use “Access with a user-assigned managed identity” approach.

Key Vault Administrator and AcrPull Role Assignments needs to be added to the user-assigned managed identity to pull the images from Azure Container Registry and Access the Key Vault Secrets.

Key Access Policy Also need to add to the user-assigned managed identity to access the secrets.

Click here to access the Terraform Module Source Code. https://github.com/gmk1995/azure-terraform-modules

A screenshot of a computer

Description automatically generated

A screenshot of a computer

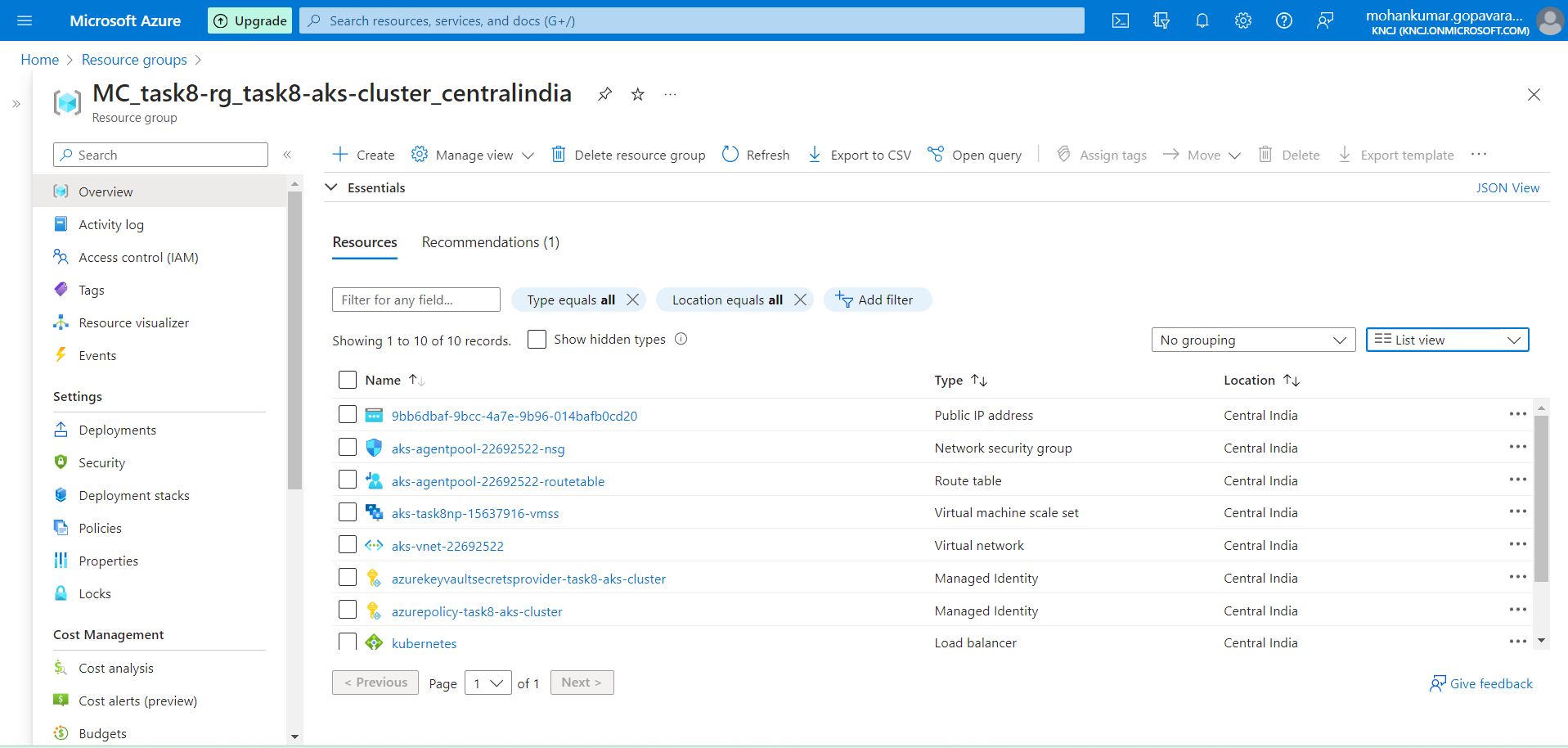
Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Once Azure Kubernetes Cluster is Created Open any Terminal and run the below command to get kubeconfig file to access the AKS Cluster:

**az aks get-credentials --resource-group task8-rg --name task8-aks-cluster**

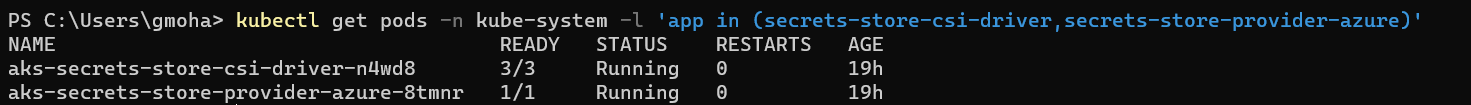
**Above Command Response: WARNING: Merged "task8-aks-cluster" as current context in C:\Users\xxxx\.kube\config**

**Note:** resource-group name and aks cluster name required.

Once the kubeconfig file is downloaded, and stored in locally we can access the aks cluster.

**Verify the Azure Key Vault provider for Secrets Store CSI Driver installation:**

**kubectl get pods -n kube-system -l 'app in (secrets-store-csi-driver,secrets-store-provider-azure)'**



**Create secretprovider class using below yaml file and command kubectl apply -f secretproviderclass.yaml :**

**Note:** All the Required File are in the GitHub Url: https://github.com/gmk1995/azure-terraform-modules

Update the userAssignedIdentityID: # Set the clientID of the user-assigned managed identity to use. Go the Azure Portal and Get the Client ID of user-assigned managed identity

Update the Azure Key Name keyvaultName: # Set to the name of your key vault

In the Object Name update the objectName: CREATOR # secret name which you have created and stored in key vault secret

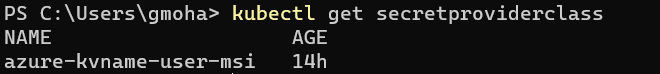
Update the tenantId run the below command to get tenantId of the key vault.

az aks show -g < resource group name > -n <aks-cluster> --query addonProfiles.azureKeyvaultSecretsProvider.identity.clientId -o tsv

Update the resource group name and aks-cluster-name

A screenshot of a computer

Description automatically generated

Verify Secret Provider Class using below command:  


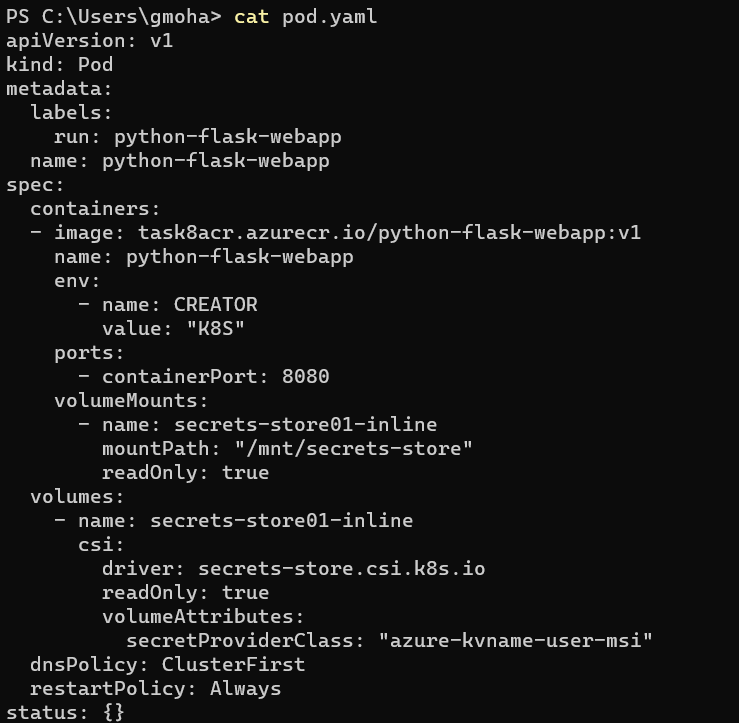
We have Created a secret to get the Secret from Key Vault:

A screenshot of a computer

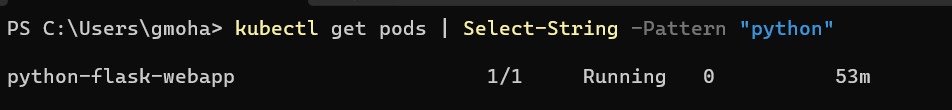
Description automatically generated

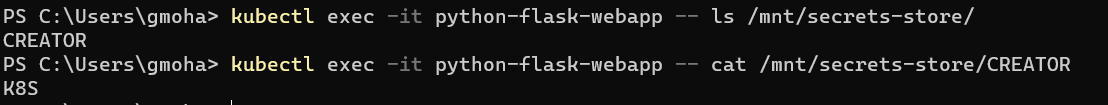
Create a Pod to Mount the Secret as secret volume in the pod using csi driver secetproviderclass and add a environment variable **CREATOR : K8S**

kubectl apply -f pod.yaml



Verify the Pod is running status and check whether the secret is mounted as secret volume or not.





Check whether the Key Vault secret value and secret volume mounted in pod is same or not

A screenshot of a computer

Description automatically generated

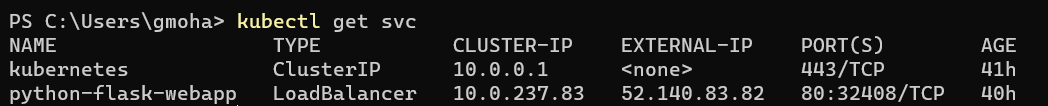
Create a Load Balancer Service to Access the Pod from web browser:

kubectl apply -f service.yaml

A screen shot of a computer

Description automatically generated

Verify whether the service



Access the pod with load balancer IP address with port number 80

A white background with a black and white line

Description automatically generated with medium confidence