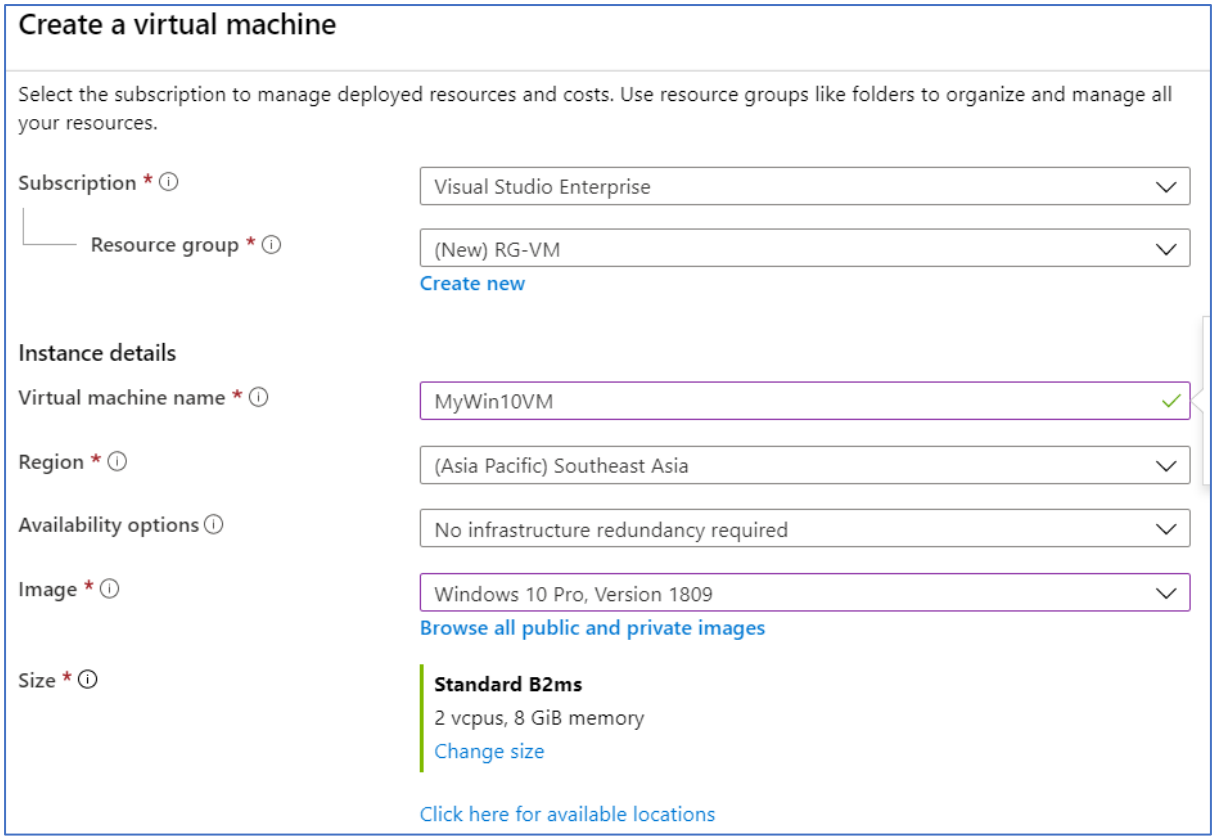
Please use Azure Portal with your personal email and not organization specific email

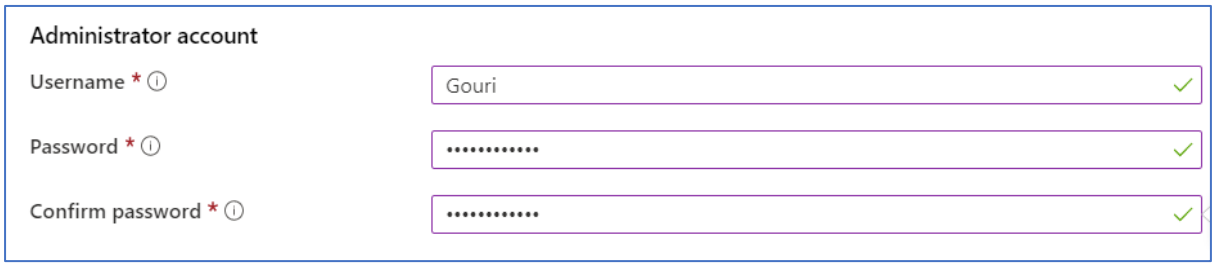
Whenever any string is in <> it means you need to provide the proper value at that location. Exp <VM Name> means enter actual value without <>

Exercise: Azure Virtual Machine Creation - Using Azure Portal

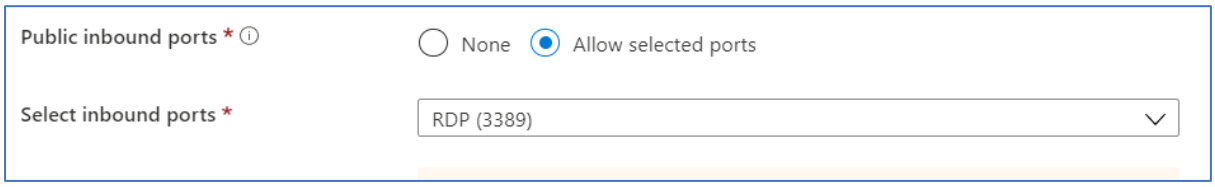
1. Login to Azure Portal, click on Virtual Machines Blade and click on Add
2. Create a new Resource Group, select the subscription and location for VM



3. Create a simple VM with Windows 10 OS with the size B2ms. Provide the user name and password.



4. Make sure that RDP is selected so that we can connect to VM later



5. In Advanced tab find out how extensions can be installed for VM
6. After successful creation of VM, connect using remote desktop

Note: you can create a virtual network, assign nsg to it and use the same when creating a VM. Advantage will be that the VM will automatically have the set rules from nsg.

Exercise: Azure Virtual Machine Creation - Using Azure PowerShell

Use New-AzureRmVm

You can provide parameters when prompted or can use them while using the command (in this case many default settings will be considered)

```
New-AzureRmVm
   -ResourceGroupName <RGName>
   -Name <VM Name>
   -Location "East US"
    -VirtualNetworkName <vNet Name>
    -SubnetName "default
    -OpenPorts 80,3389
```

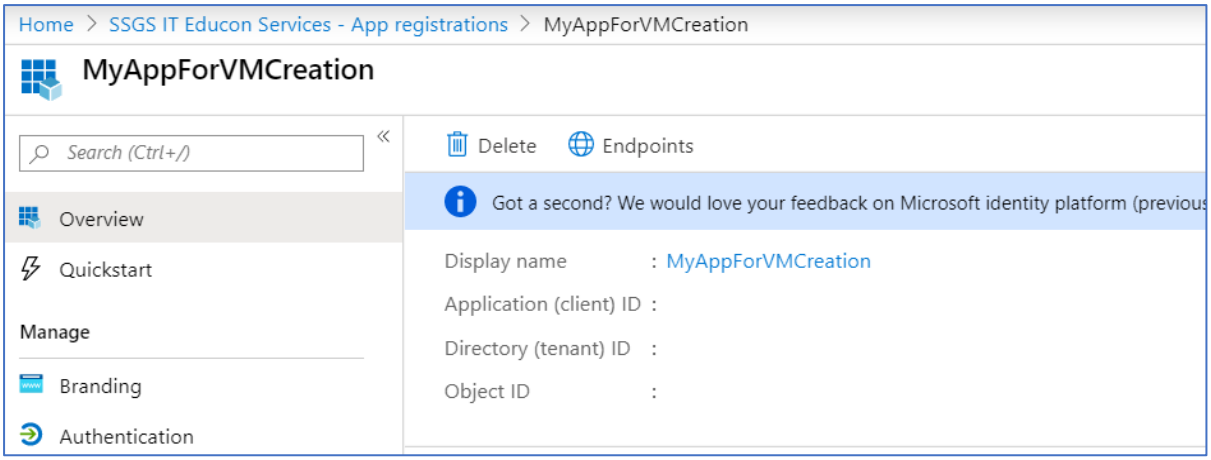
You can also declare the variables first and use them while giving the command

Exercise: Azure Virtual Machine Creation - Using Visual Studio

1. Start Visual Studio and create a console application
2. Add NuGet package to Microsoft.Azure.Management.Fluent
3. Add azureauth.properties with following

   subscription=<subscription-id>
   client=<application-id>
   key=<authentication-key>
   tenant=<tenant-id>
   managementURI=https://management.core.windows.net/
   baseURL=https://management.azure.com/
   authURL=https://login.windows.net/
   graphURL=https://graph.windows.net/

   In order to get values for client create an application with Azure active directory blade and selecting App Registration. You will get client id and tenant id from here.



Create a secrete for the application and copy the value and that becomes your key

4. Add environment variable by using following PowerShell
[Environment]::SetEnvironmentVariable("AZURE_AUTH_LOCATION", "C:\Visual Studio 2019\Projects\myDotnetProject\myDotnetProject\azureauth.properties", "User")

5. You can use following code

```
var loc = Environment.GetEnvironmentVariable("AZURE_AUTH_LOCATION");
var credentials =
SdkContext.AzureCredentialsFactory.FromFile(Environment.GetEnvironmentVariable("AZURE_AUTH_LOCATION"));
```

```
 var azure =
azure.Configure().WithLogLevel(HttpLoggingDelegatingHandler.Level.Basic).Authenticate(credentials).WithDefaultSubscription();
```

```
var groupName = "RG-VMFromCode";
var vmName = "mySimpleVM";
var location = Region.USCentral;

Console.WriteLine("Creating resource group...");
var resourceGroup = azure.ResourceGroups.Define(groupName)
.WithRegion(location).Create();

Console.WriteLine("Creating availability set...");
var availabilitySet = azure.AvailabilitySets.Define("myAVSet")
 .WithRegion(location).WithExistingResourceGroup(groupName)
  .WithSku(AvailabilitySetSkuTypes.Classic).Create();
Console.WriteLine("Creating public IP address...");
var publicIPAddress =
  azure.PublicIPAddresses.Define("myPublicIP").WithRegion(location).
  WithExistingResourceGroup(groupName) .WithDynamicIP().Create();
Console.WriteLine("Creating virtual network...");
var network = azure.Networks.Define("myVNet").
    WithRegion(location).WithExistingResourceGroup(groupName).
    WithAddressSpace("10.0.0.0/16").
    WithSubnet("mySubnet", "10.0.0.0/24").Create();

Console.WriteLine("Creating network interface...");
var networkInterface = azure.NetworkInterfaces.Define("myNIC")
        .WithRegion(location)
```

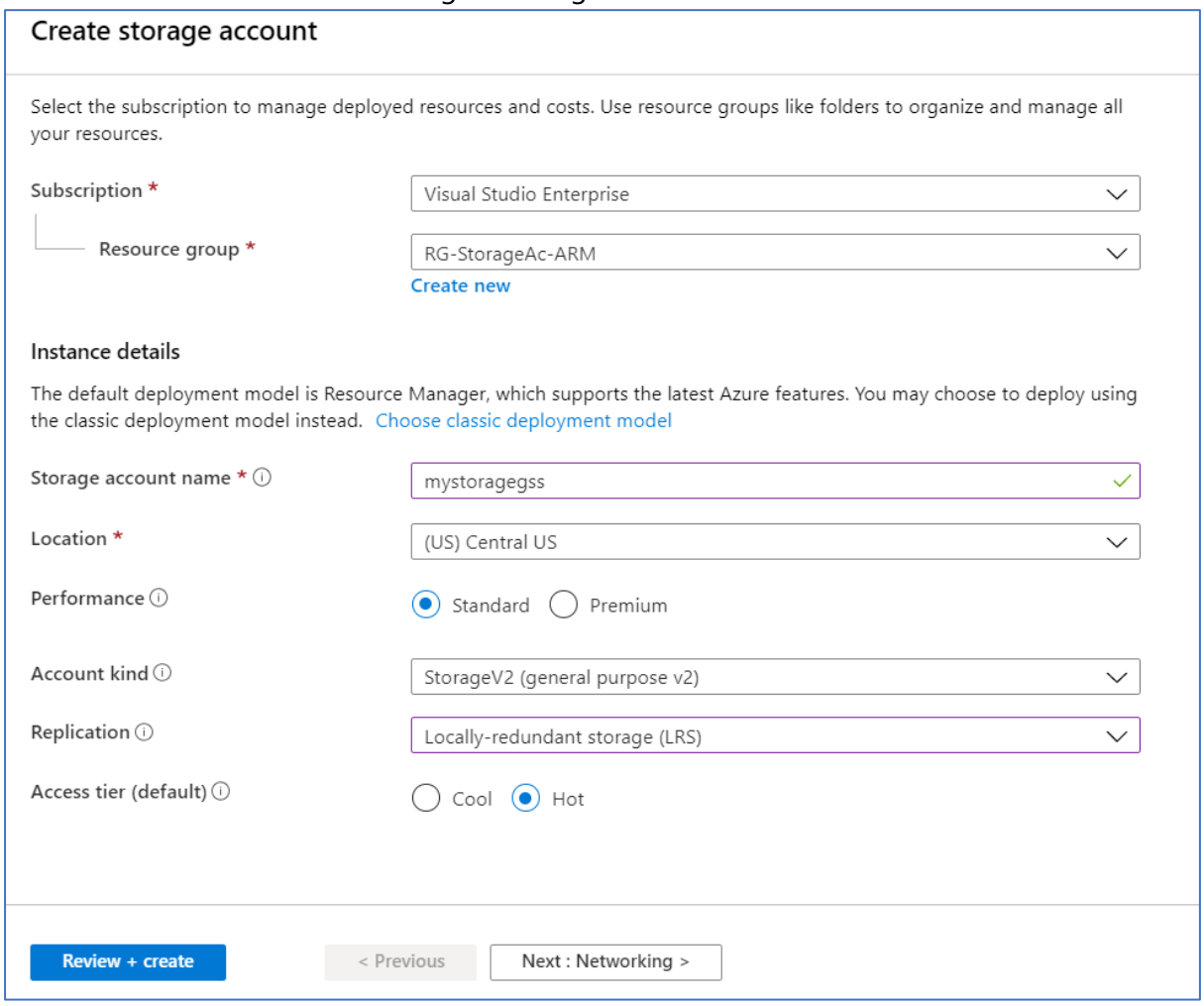
```
            .WithExistingResourceGroup(groupName)
            .WithExistingPrimaryNetwork(network)
            .WithSubnet("mySubnet")
            .WithPrimaryPrivateIPAddressDynamic()
            .WithExistingPrimaryPublicIPAddress(publicIPAddress)
            .Create();

Console.WriteLine("Creating virtual machine...");
azure.VirtualMachines.Define(vmName)
            .WithRegion(location)
            .WithExistingResourceGroup(groupName)
            .WithExistingPrimaryNetworkInterface(networkInterface)
            .WithLatestWindowsImage("MicrosoftWindowsServer", "WindowsServer", "2012-R2-Datacenter")
            .WithAdminUsername("<username>").WithAdminPassword("<password>")
            .WithSize(VirtualMachineSizeTypes.StandardDS1).Create();
```

6. If you find that the file cannot be read with the help of environmental variable use actual path to read the file
7. If you get authority related error use another method for authentication as follows
   var credentials = SdkContext.AzureCredentialsFactory.FromServicePrincipal(clientId, clientSecrete, tenantId, AzureEnvironment.AzureGlobalCloud);
   Provide appropriate values for parameters
8. It will take quite some time to create the Virtual Machine. Connect to VM and see that its working properly. In order to avoid using resources ensure that you delete the whole Resource Groups at the end

Exercise: ARM Template Creation - Using Azure Portal

1. Sign in to Azure Portal

2. Click on Create a Resource – Storage – Storage Account



Provide resource group name, storage account name and other details.

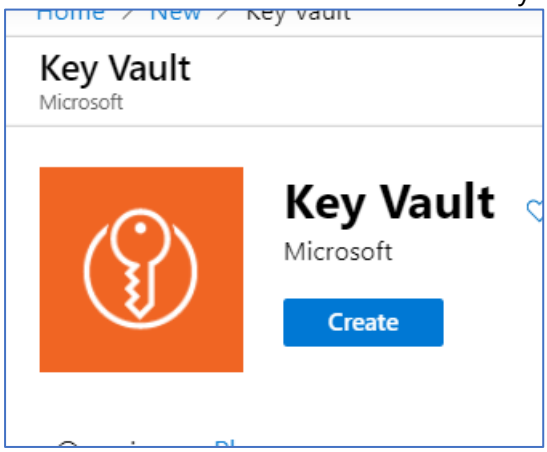
3. Click on Review + Create, click on Create after successful validation
4. Click on Download template from Export Template Blade
5. Look at the parameter created
6. Extract json files at some location
7. Create another resource for template deployment and select build your own template in editor
8. Let us change the parameter and create another resource
9. Click on Load file and load the template downloaded earlier
10. We will pick up the value for the storage account using a variable. In order to do that remove the parameter declared and add a variable called storageAccountName
11. Change the name for resource of account name as "storageAccountName": "[concat(uniquestring(resourceGroup().id), 'standardsa')]" change the value for parameter wherever it appears
12. Save the template, create resource after successful validation
13. Click on Purchase after entering values for resource group and location
14. Ensure that the resource gets created
15. Remove unwanted resource groups and clean up resources

Exercise: ARM Template Creation - Using Visual Studio Code

1. Open Visual Studio Code
2. Open Extensions Pane and search for azure manager resource tools and Install
3. File - Open File - azureDeploy.json (this file is available in lab documents folder)
4. Deploy using Azure Cloud Shell
5. https://shell.azure.com
6. Open Bash - Upload
7. Run following commands
8. echo "Enter the Resource Group name:" &&
   read resourceGroupName &&
   echo "Enter the name for this deployment:" &&
   read deploymentName &&
   echo "Enter the location (i.e. centralus):" &&
   read location &&
   az group create --name $resourceGroupName --location $location && az group deployment create --name $deploymentName --resource-group $resourceGroupName --template-file "azureDeploy.json"
9. Delete resource group after ensuring that the resource got created

Exercise: Create Azure Key Vault

1. Login to Azure Portal
2. Click on Create a resource and select Key Vault



3. Provide details for resource group name, name for key vault, location and click on Review + Create. After successful validation click on Create
4. Once the resource is created add a key to it Select Keys – Generate/Import, provide name and click on enter.
5. We will be using this key vault with key in next exercise

Exercise: Azure Virtual Machine Decryption

Pre-requisite: existing Virtual Machine, key vault with 1 key in it(depending on if your VM and Key Vault are in same resource group or not you need to provide names in the commands)

1. Login to Azure Portal

2. Start Cloud Shell
3. $keyVaultName = "<key vault name>"
   $rgName="<Resource Group Name>"
   $keyVault = Get-AzKeyVault -VaultName $keyVaultName -ResourceGroupName $rgName;
   $diskEncryptionKeyVaultUrl = $keyVault.VaultUri;
   $keyVaultResourceId = $keyVault.ResourceId;
   $keyEncryptionKeyUrl = (Get-AzKeyVaultKey -VaultName $keyVaultName -Name <key name>).Key.kid;

4. Write-Host $keyVault
   Set-AzVMDiskEncryptionExtension -ResourceGroupName $rgName `
   -VMName "<VM Name>" `
   -DiskEncryptionKeyVaultUrl $diskEncryptionKeyVaultUrl `
   -DiskEncryptionKeyVaultId $keyVaultResourceId `
   -KeyEncryptionKeyUrl $keyEncryptionKeyUrl `
   -KeyEncryptionKeyVaultId $keyVaultResourceId

5. This is a time-consuming process and will take at least 10-15 minutes to complete

```
RequestId IsSuccessStatusCode StatusCode ReasonPhrase
--------- ------------------- ---------- ------------
                         True         OK OK
```

6. You can use
   az vm encryption show --resource-group "<RG name>" --name "<VM name>"
   OR
   Get-AzureRmVMDiskEncryptionStatus -ResourceGroupName <RG Name> -VMName <VM Name>
   to find
7. The encryption can be reversed by following command
   Disable-AzureRmVMDiskEncryption -ResourceGroupName <resource-group> -VMName <VM name>

Exercise: Azure Batch Job Creation - Using Azure CLI

1. Create resource group
   az group create
   --name myResourceGroup
   --location eastus2
2. Create storage account
   az storage account create \
   --resource-group myResourceGroup \

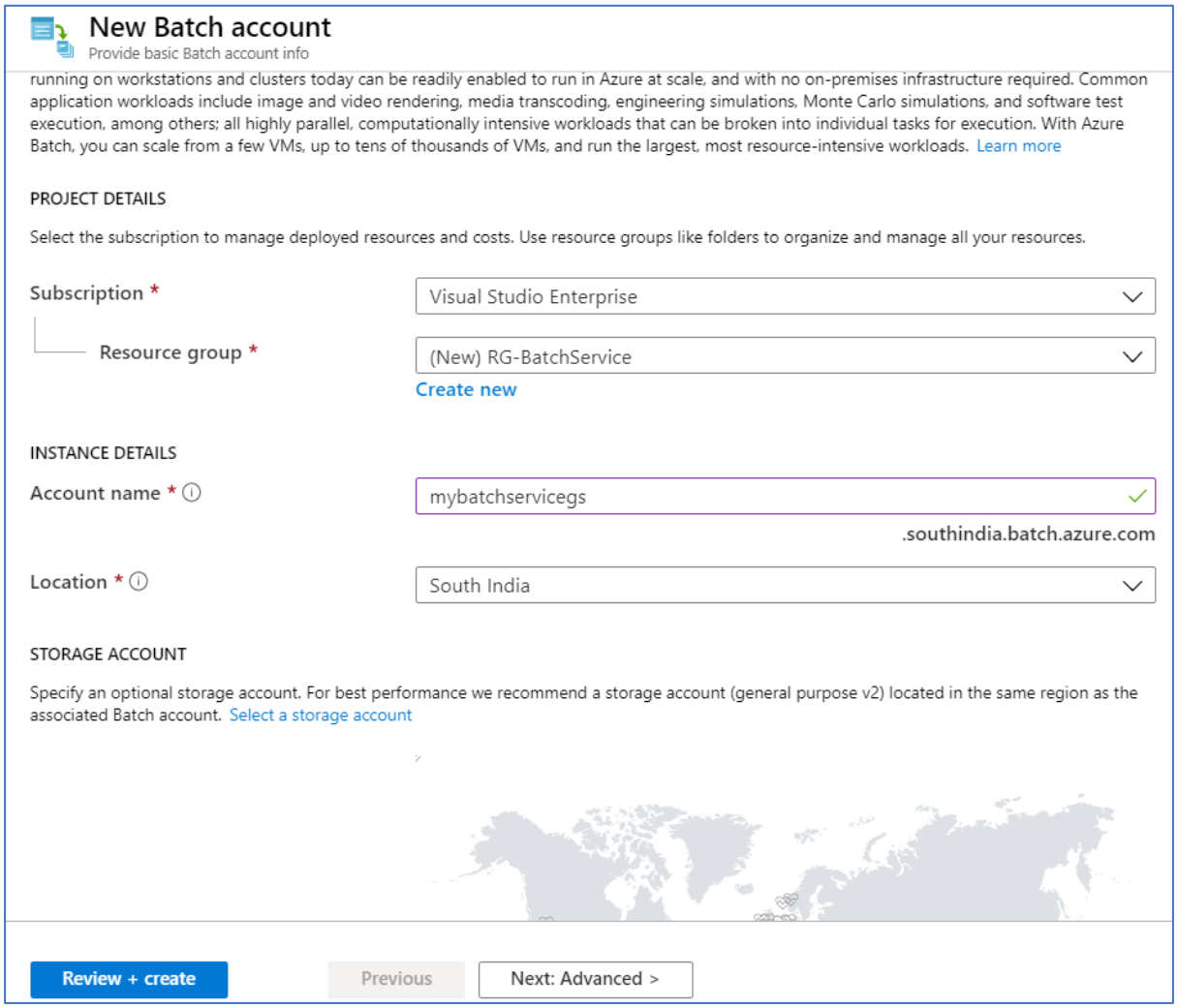
```
 --name mystorageaccount \
--location eastus2 \
--sku Standard_LRS
```

3. Create batch account
```
az batch account create \
--name mybatchaccount \
--storage-account mystorageaccount \
--resource-group myResourceGroup \
--location eastus2
```
4. Login to batch account
```
az batch account login \
--name mybatchaccount \
--resource-group myResourceGroup \
--shared-key-auth
```
5. Create pool of compute nodes
```
az batch pool create \
--id mypool --vm-size Standard_A1_v2 \
--target-dedicated-nodes 2 \
--image canonical:ubuntuserver:16.04-LTS \
--node-agent-sku-id "batch.node.ubuntu 16.04"
```
6. Find allocation state of pool
```
az batch pool show --pool-id mypool \
--query "allocationState"
```
7. Create job
```
az batch job create \
--id myjob \
--pool-id mypool
```
8. Create tasks
```
for ($i=1 ;$i -le 5; $i++) {az batch task create --task-id mytask$i --job-id myjob --
command-line "/bin/bash -c 'printenv | grep AZ_BATCH; sleep 90s'" }
```
9. View task status
```
az batch task show \
--job-id myjob \
--task-id mytask1
```
10. View task output
```
az batch task file list \
--job-id myjob \
--task-id mytask1 \
 --output table
```
11. Download one of the output files
```
az batch task file download \
--job-id myjob \
--task-id mytask1 \
```

--file-path stdout.txt \
--destination ./stdout.txt

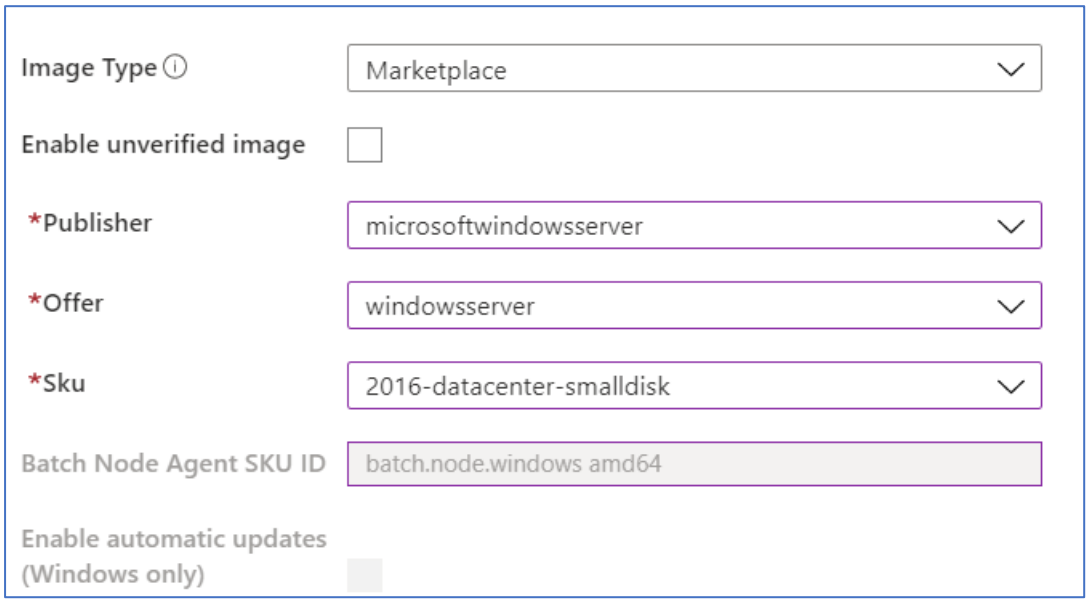
Exercise: Azure Batch Job Creation - Using Azure Portal

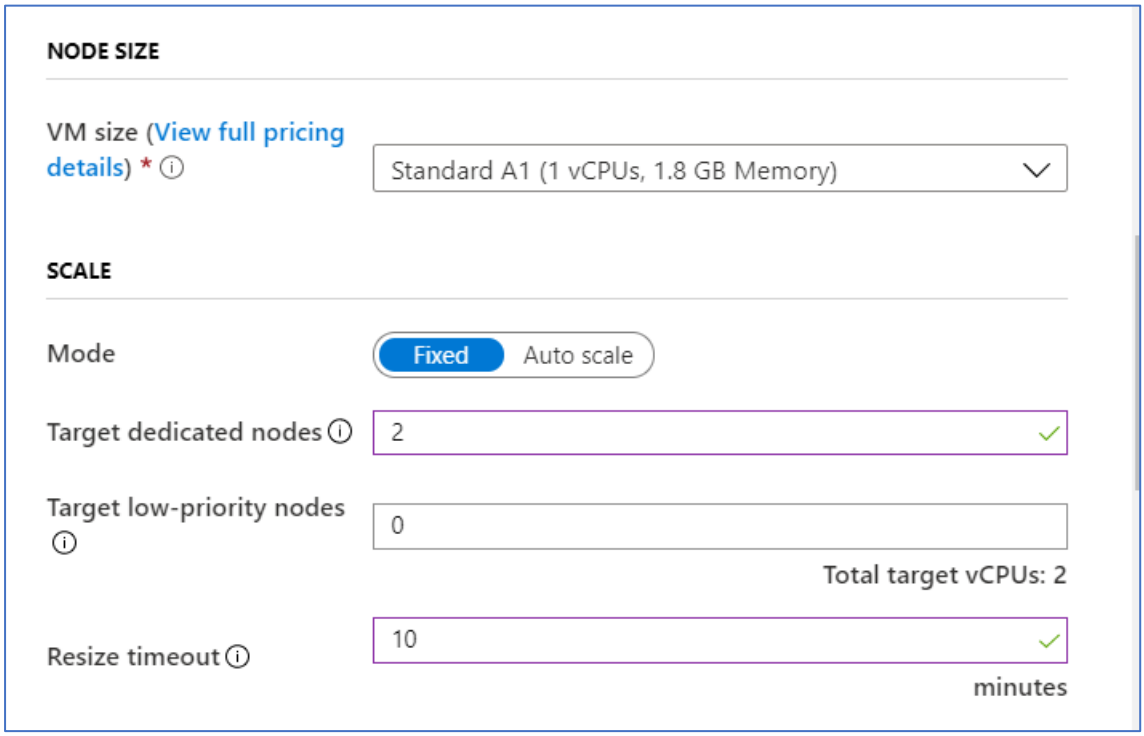
1. Login to Azure Portal
2. Select Create a resource – Compute – Batch Service. Provide subscription, resource group, name and location



3. Click on Review + Create, after successful validation click on Create

4. Go to the resource and select Pools – Add

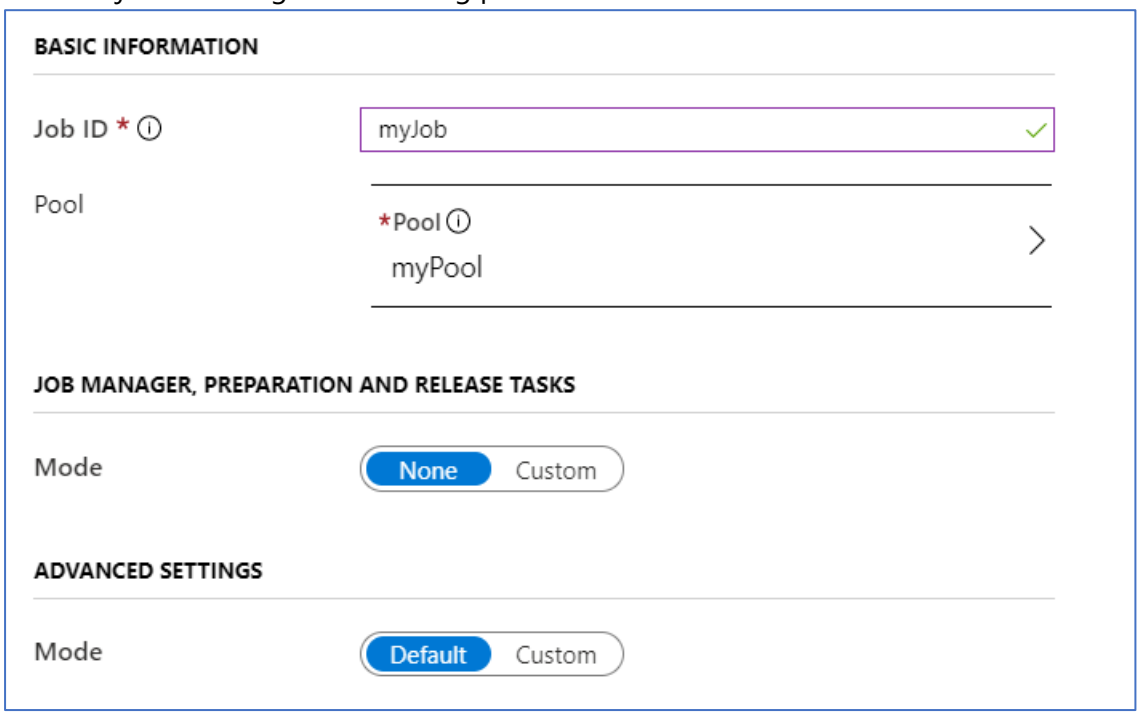




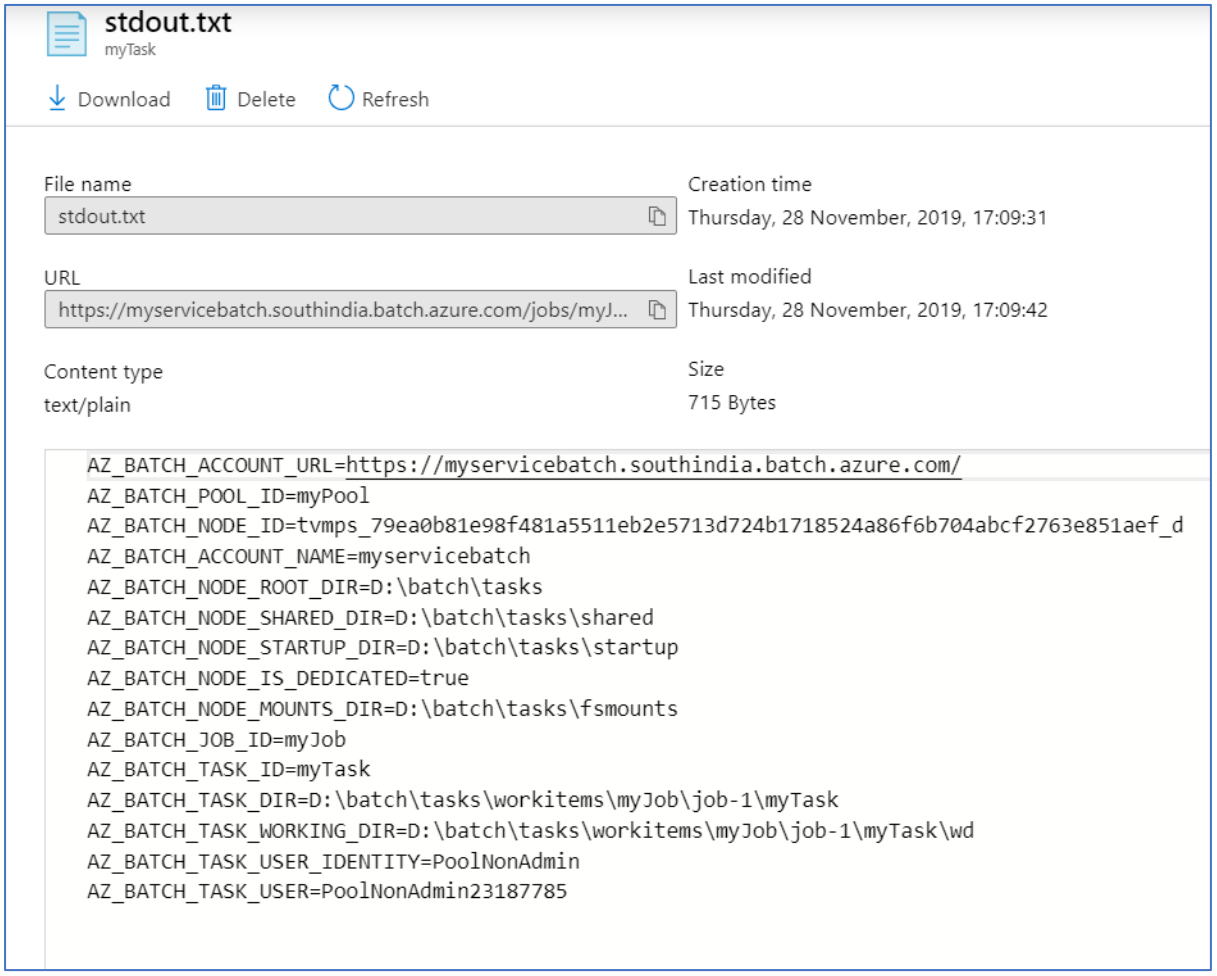
5. After a few minutes state of pool will become steady

6.

7. Create a job and assign it to existing pool

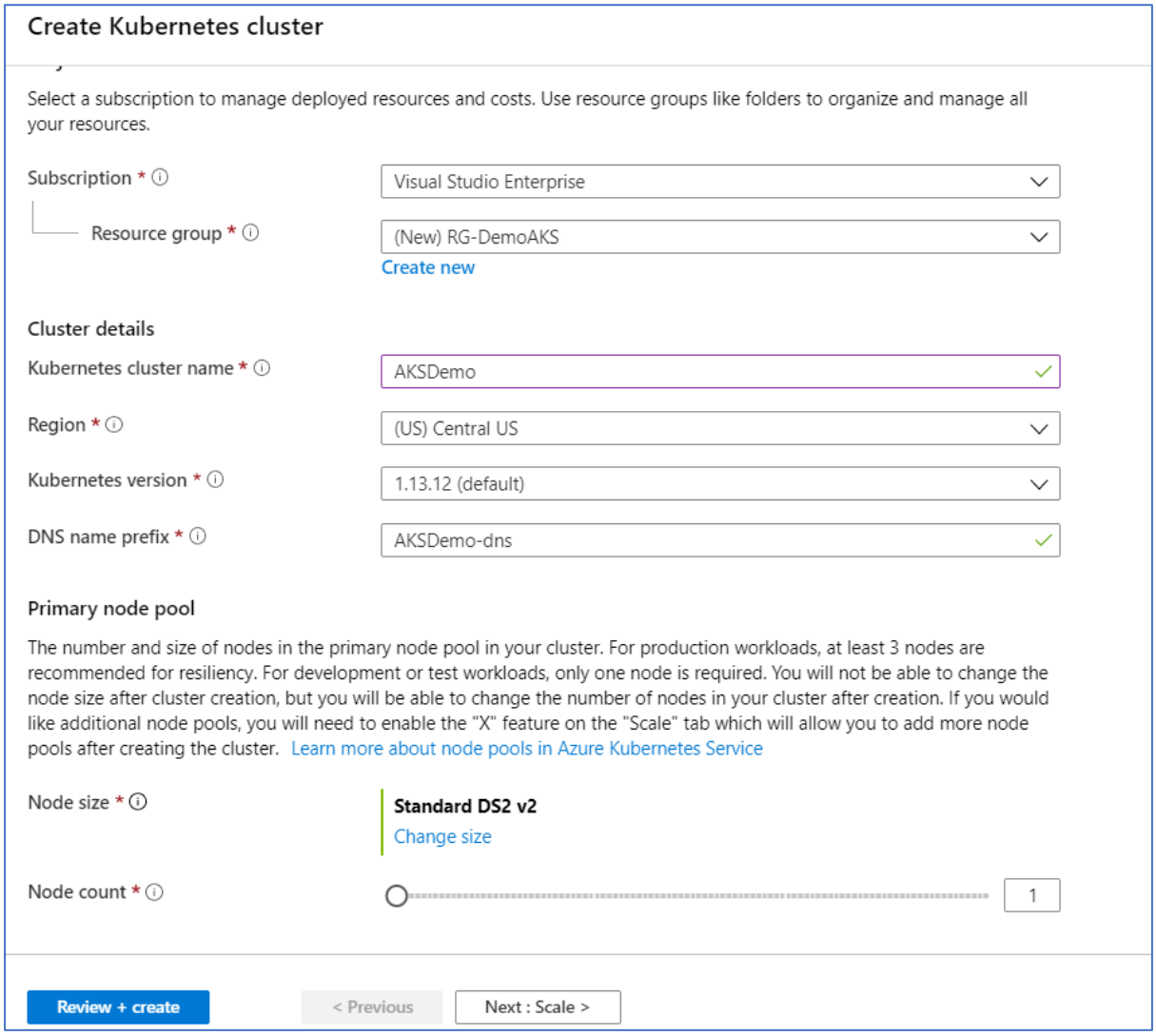


8. Create Task – Add – provide id and enter cmd /c "set AZ_BATCH & timeout /t 90 > NUL" in command line, click on Submit by keeping all default settings
9. Click Files on node and select the file stdout.txt

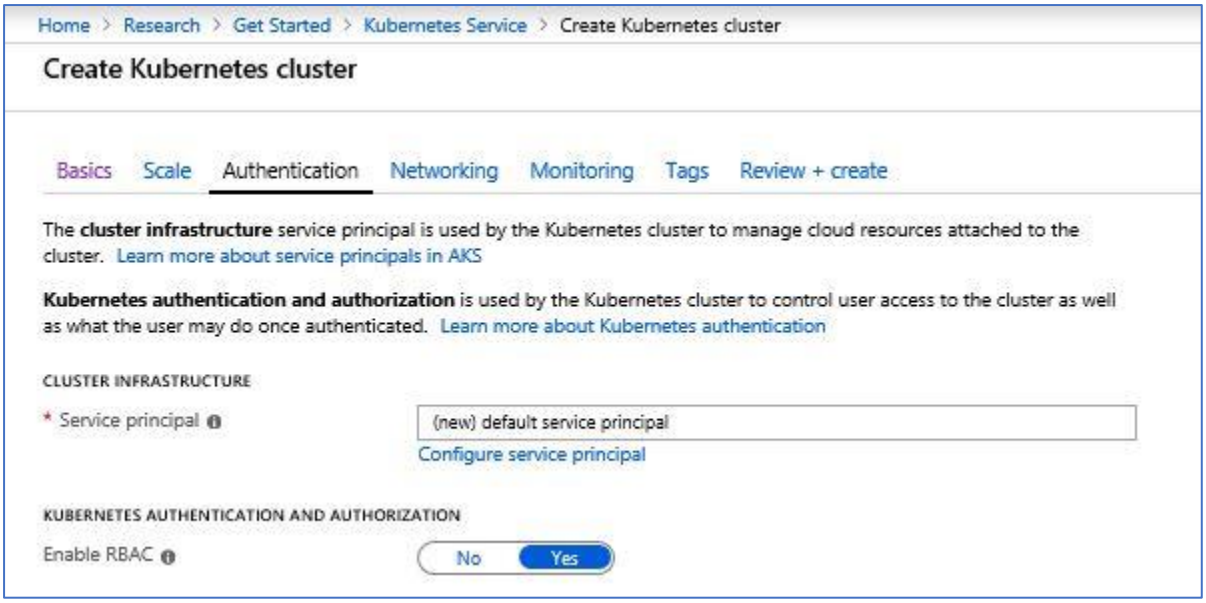


Exercise: Azure Kubernetes Service (AKS) – Creation -Using Azure Portal

1. Login to Azure Portal
2. Create Azure Kubernetes Cluster (AKS Cluster) – this has VMs in the form of nodes
   Click on Create a resource – Kubernetes Service

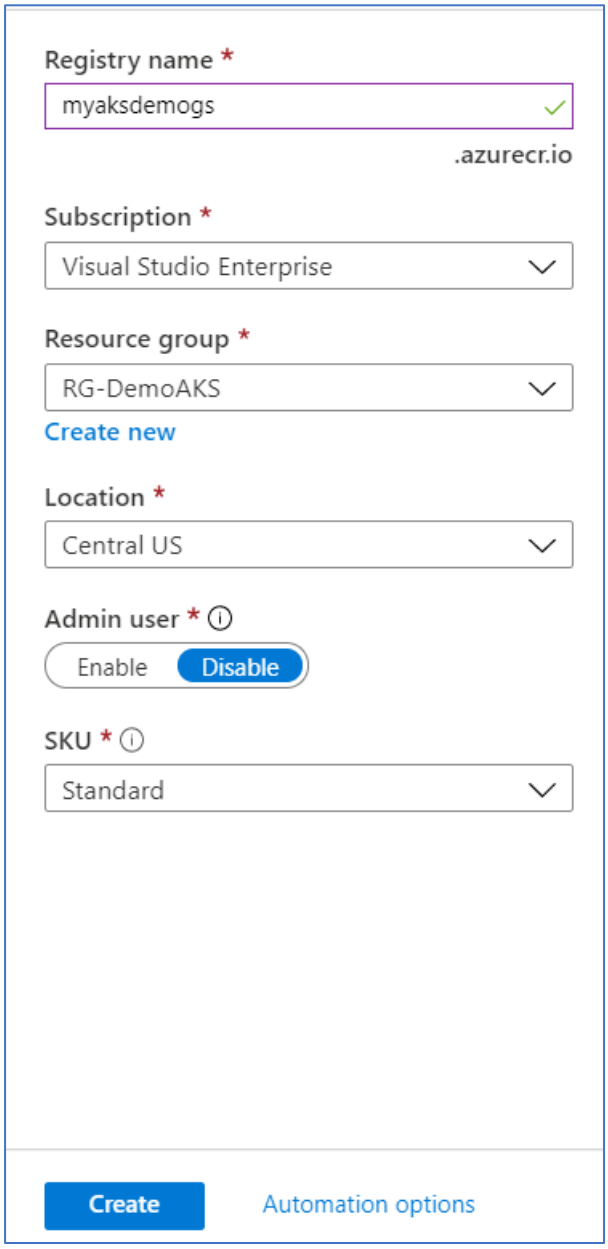


This in turn will create service principal which gets created in Azure AD (used to grant permission to access resources)

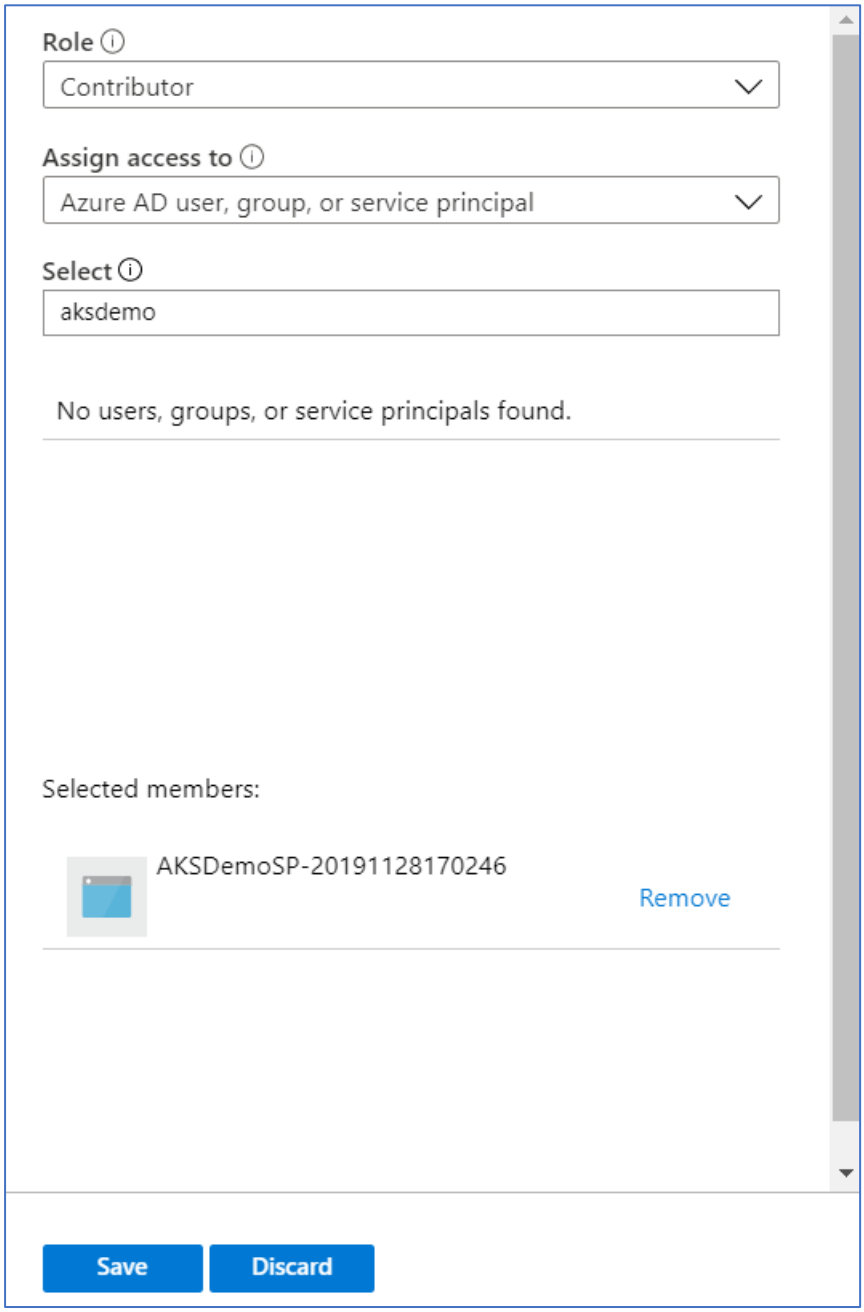


You can check it by going to Azure AD and selecting App registrations blade. I can see mine as follows

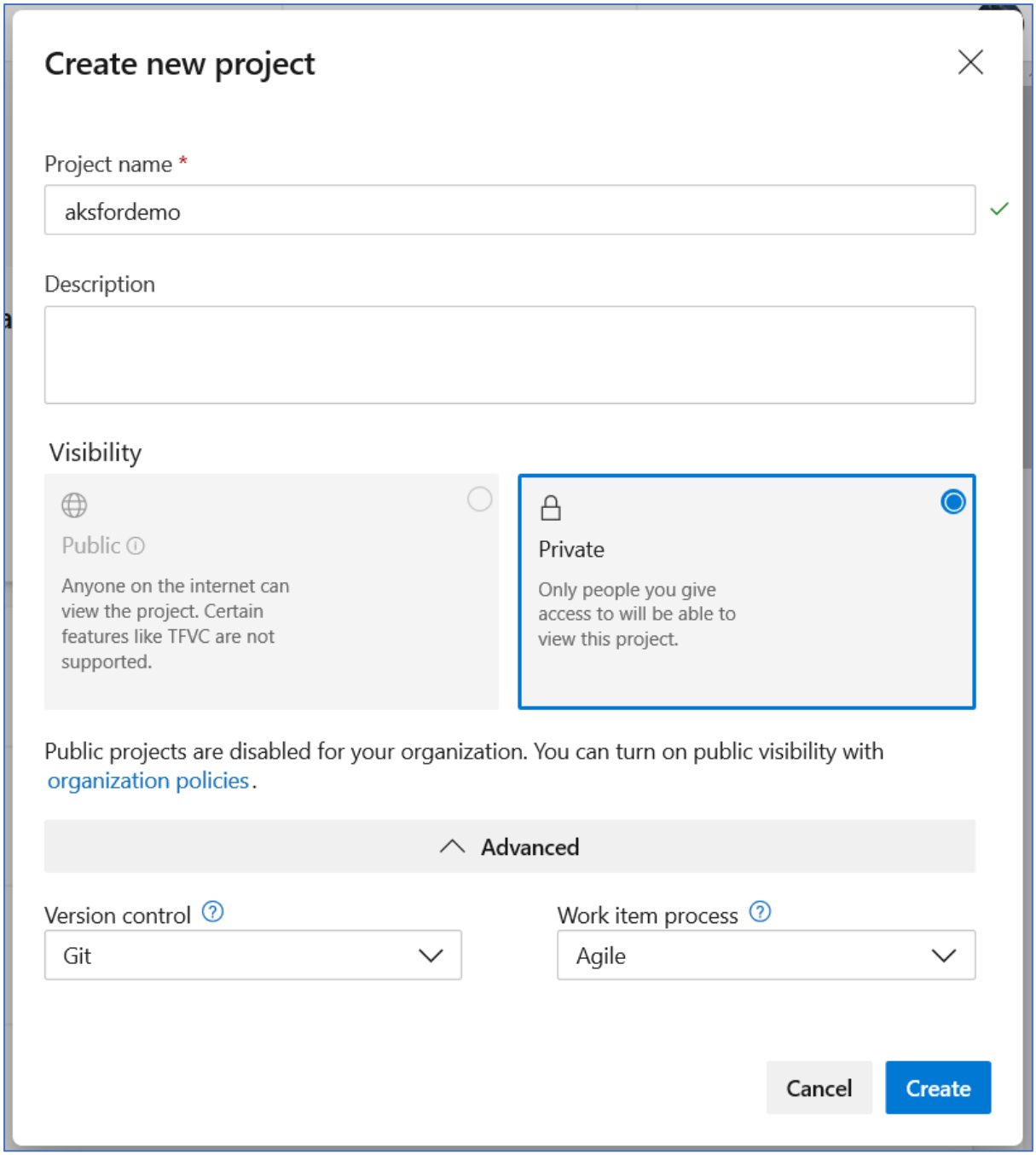
3. Create Azure Container Registry (Container Registry) in the same resource group. Provide unique name and click on Create



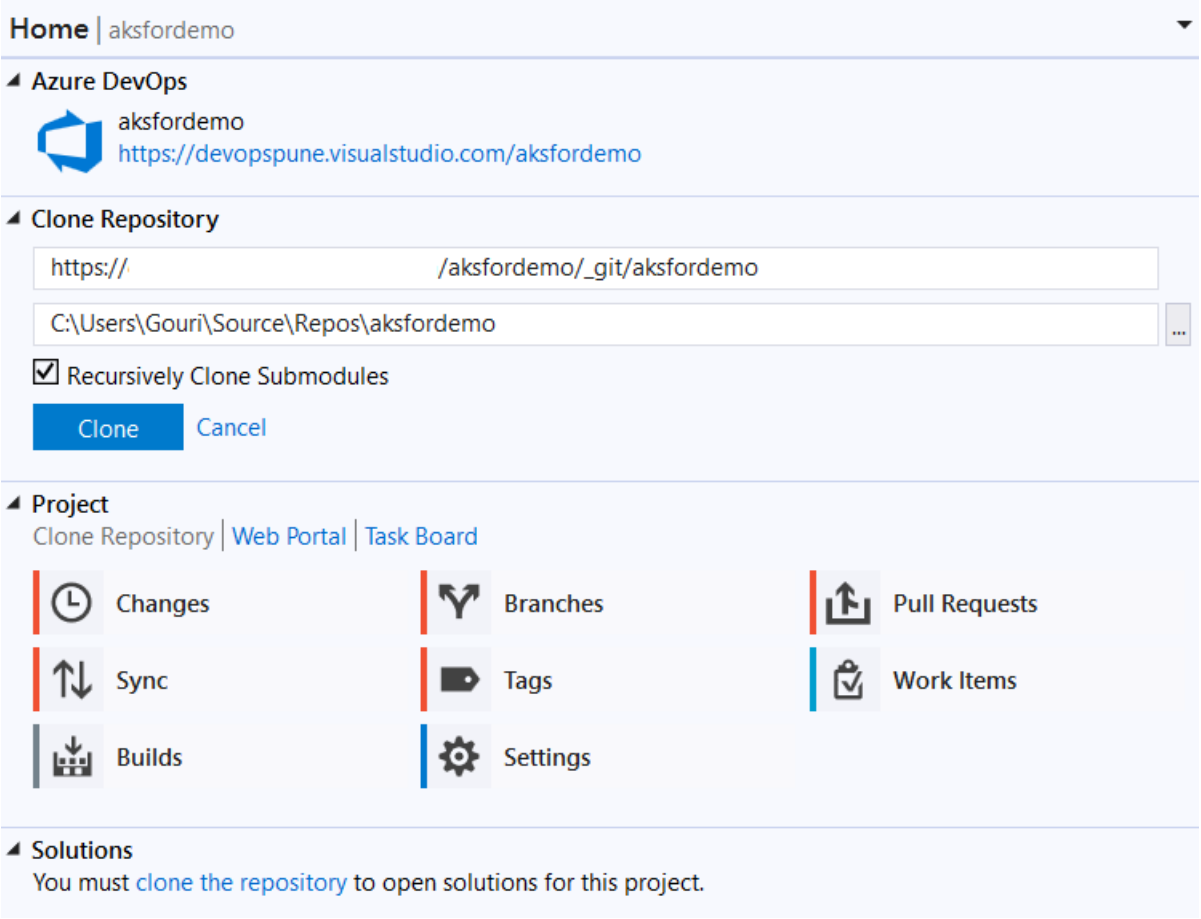
4. We need to edit Access Control of ACR to add Service Principal of AKS cluster with the role of contributor (to push and pull images)



5. Now that Azure resources are ready, let us create a Team Project in Azure DevOps as follows (create a new account using https://dev.azure.com – use same email used to Azure to create this account)

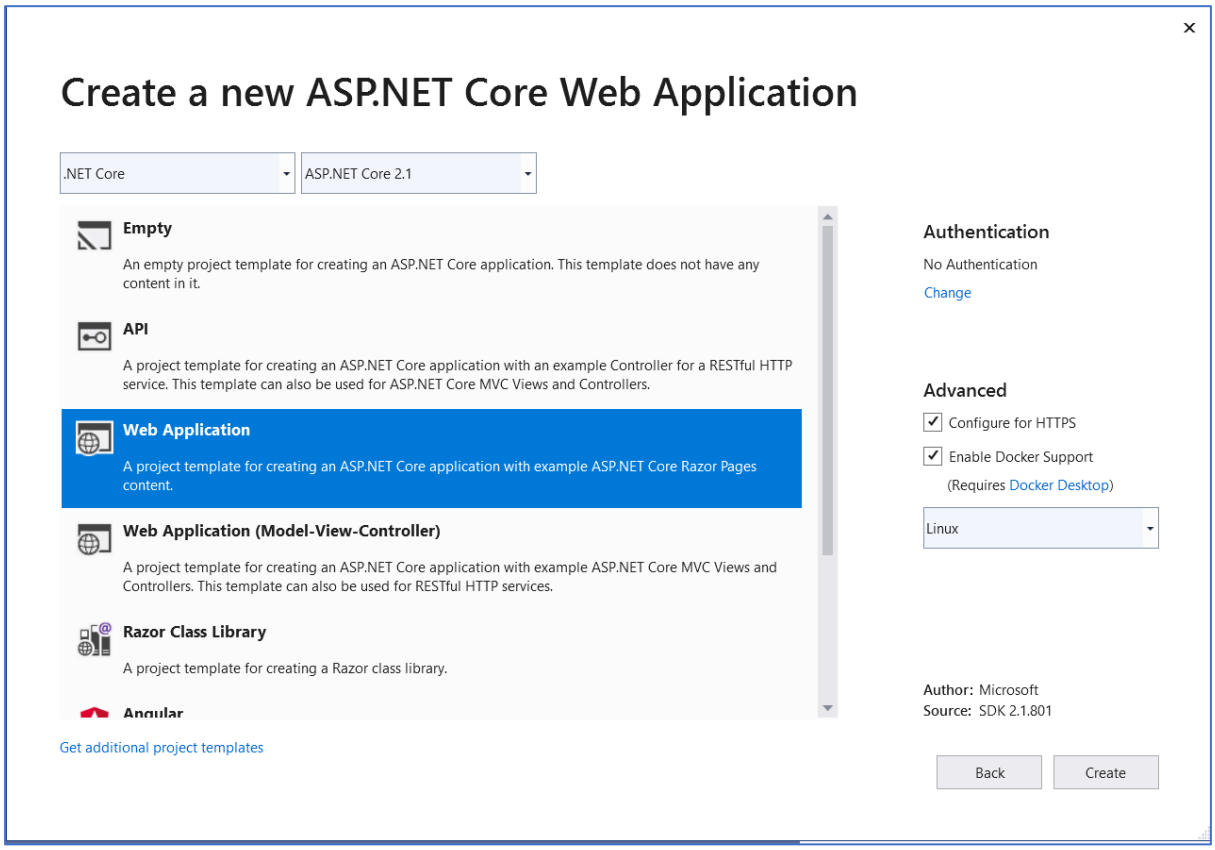

Create new project
Project name *
aksfordemo
Description
Visibility
Public
Anyone on the internet can view the project. Certain features like TFVC are not supported.
Private
Only people you give access to will be able to view this project.
Public projects are disabled for your organization. You can turn on public visibility with organization policies.
Advanced
Version control
Git
Work item process
Agile
Cancel
Create

6. Start Visual Studio and connect to the Team Project from Team Explorer. Clone the repository.



7. Click on New for Solutions from Team Explorer (Please do not use File – New Project to create a new solution- if done the mapping may not be what is required after cloning). Select ASP.NET Core Web Application, select Web Application and ensure that Docker

support is enabled



8. Change existing dockerfile witl following (you can download this from https://github.com/GouriSohoni/Training with file name as dockerrelated.zip)

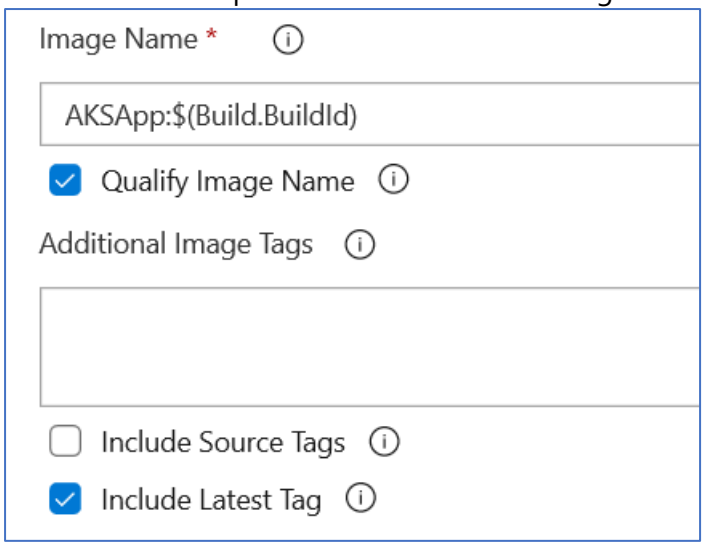
```
FROM microsoft/dotnet:2.2-aspnetcore-runtime AS base

WORKDIR /app
EXPOSE 80

FROM microsoft/dotnet:2.2-sdk AS build
WORKDIR /src
COPY AKSDemo.csproj .
RUN dotnet restore AKSDemo.csproj
COPY . .
RUN dotnet build AKSDemo.csproj -c Release -o /app

FROM build AS publish
RUN dotnet publish AKSDemo.csproj -c Release -o /app

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

9. Add yaml filenamed deployment.yml. Create a folder named manifest and add the file to it (this is also available in zip file. Remember to do the necessary changes to file – in this the image needs to be copied from container registry created earlier)

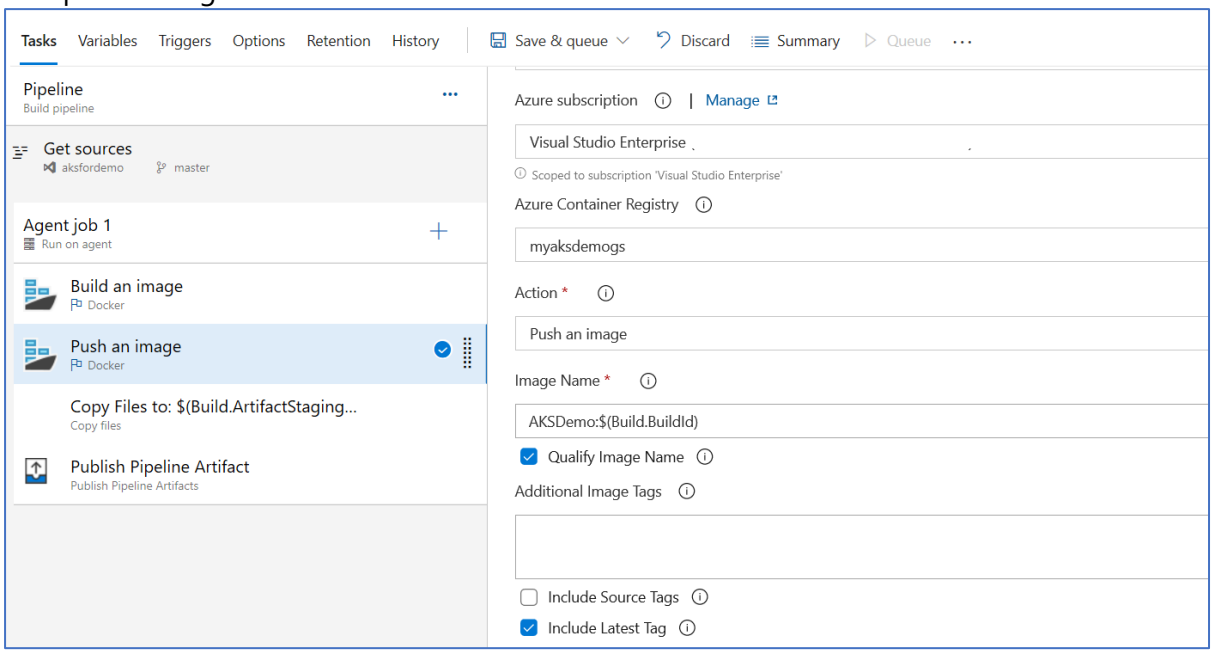
```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
    name: aksdemo
```

```
        spec:
        replicas: 1
        template:
        metadata:
  labels:
    app: aksdemo-app
  spec:
  containers:
    - name: aksdemo-services-app
      image: myaksdemogs.azurecr.io/aksdemo:latest
      ports:
        - containerPort: 80
        ---
        apiVersion: v1
        kind: Service
        metadata:
name: aksdemo-app
spec:
         ports:
        - name: http-port
         port: 80
        targetPort: 80
        selector:
          app: aksdemo-app
        type: LoadBalancer
```

10. Do some small change to any of the pages. Commit and push all code. If you are using Visual Studi0 2019 you need to take care of adding ignore file
11. Let us create a build definition. Select Pipelines – create pipelines- use the classic editor and select the template of Docker Container. Provide azure subscription and authorize. Make sure that your browser supports popups. Now the name of container registry can be selected. Provide the Image name as <app name>:$(Build.BuildId) and select the dockerfile from repo. Click on include latest tag

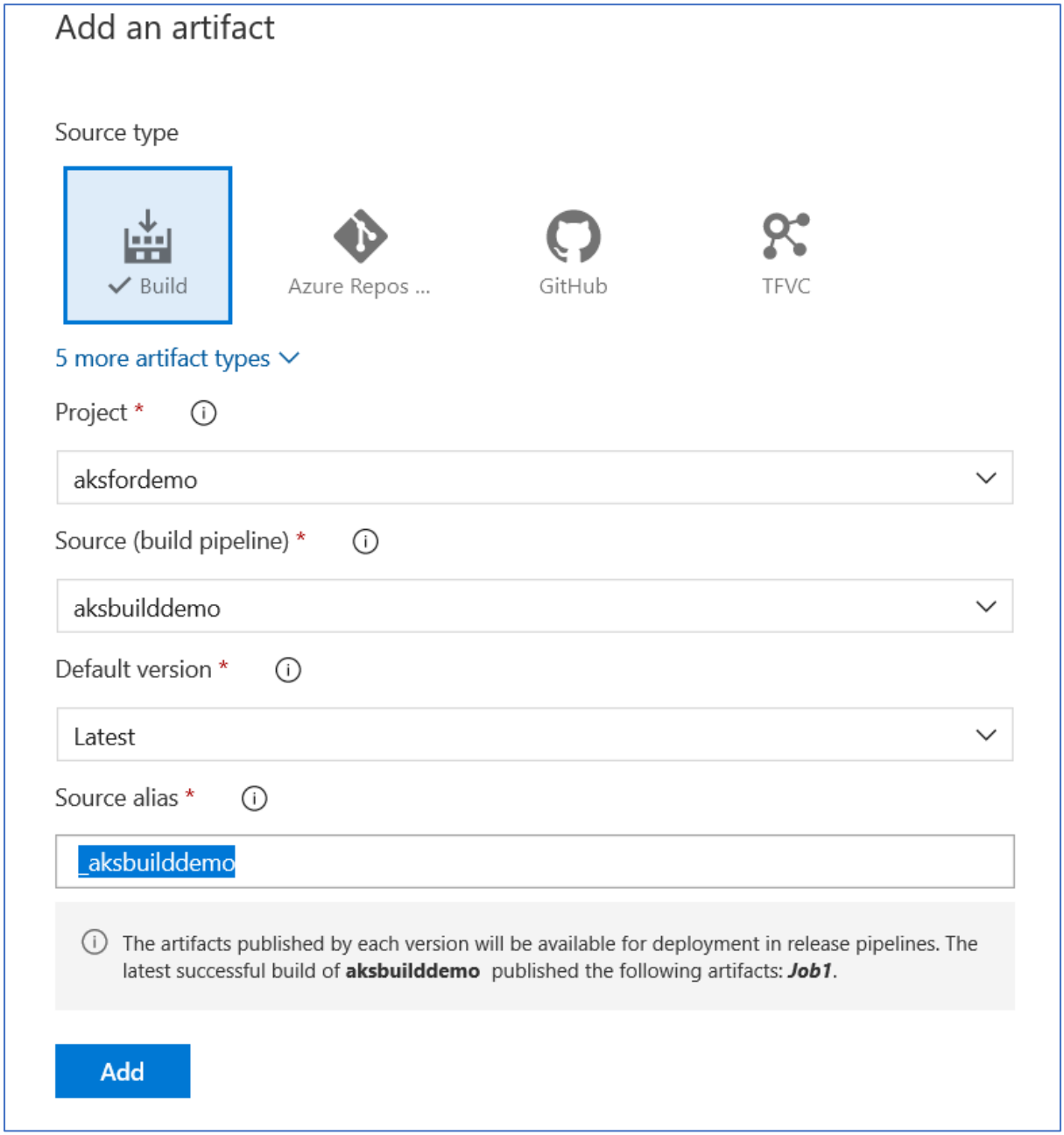


Complete configuration looks as follows

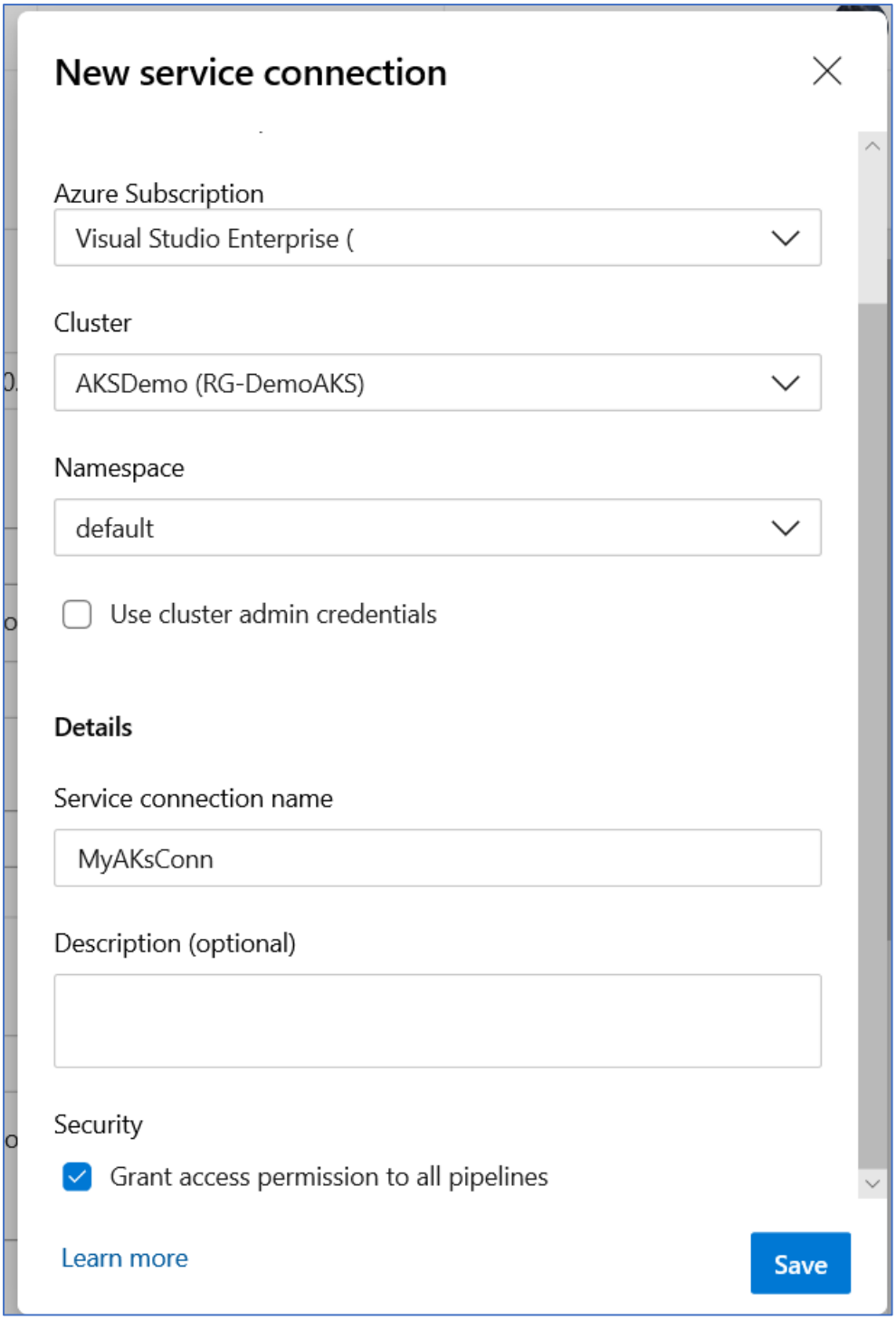


12. Provide similar configuration for push image
13. Make sure that the build name is small and without any spaces
14. Add 2 tasks for copy and publish. For copy file select source as manifest folder and destination as $(Build.ArtifactStagingDirectory). Publish artifact picks up from $(Build.ArtifactStagingDirectory), so you do not have to do anything (we need this file to deploy which will be taken care of in next stage)
    Note: Make sure you are not selecting Publish Pipeline artifact
15. Save and queue the build, you will observe that the agent has already picked up as ubuntu
16. After successful build you can find that the repository is shown in the Container Registry
17. Now we need to deploy the application we created using Visual Studio, create a release definition for it. Select Release blade – New Pipeline – Select template for Deploy to Kubernetes Cluster.

18. Add the artifact to build we created earlier, observe the source alias created

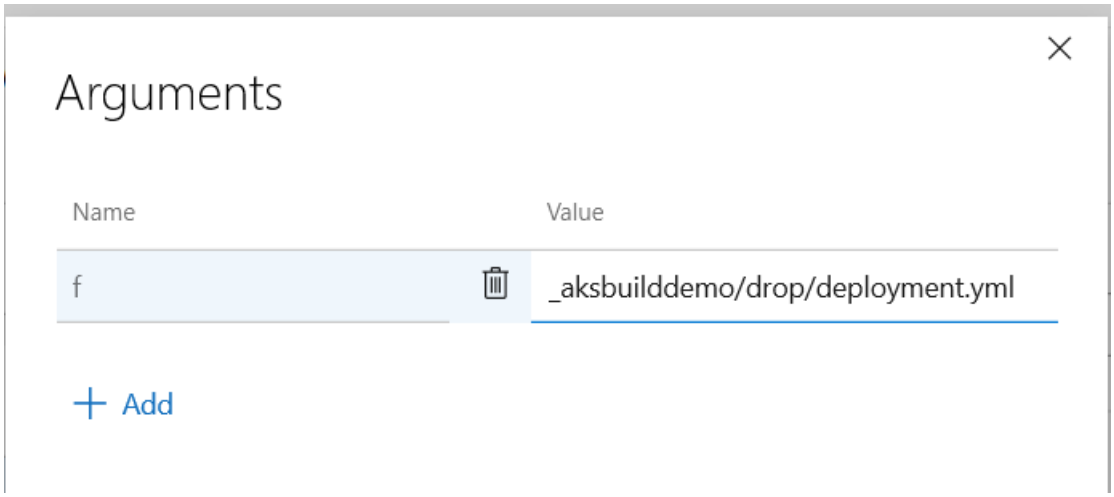


19. Configure kubectl task which got added, create connection to Kubernetes service by using New- select Azure Subscription and provide the required details for authentication. Select the cluster name and namespace as default

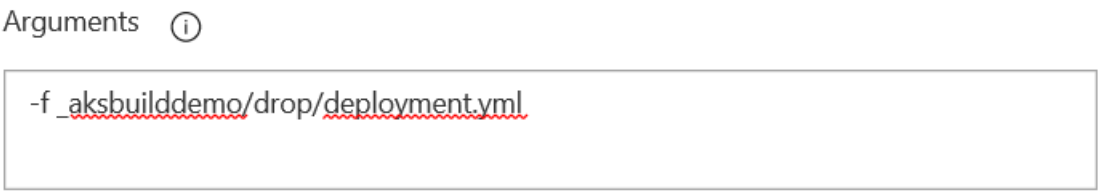


You can decide if you want to grant access to all pipelines or not.

20. Do not select any namespace for kubectl task, select command as apply
21. Add an argument by clicking on ellipse, enter f for file and provide value as <source alias for build>/drop/deployment.yml



It will look as fllows



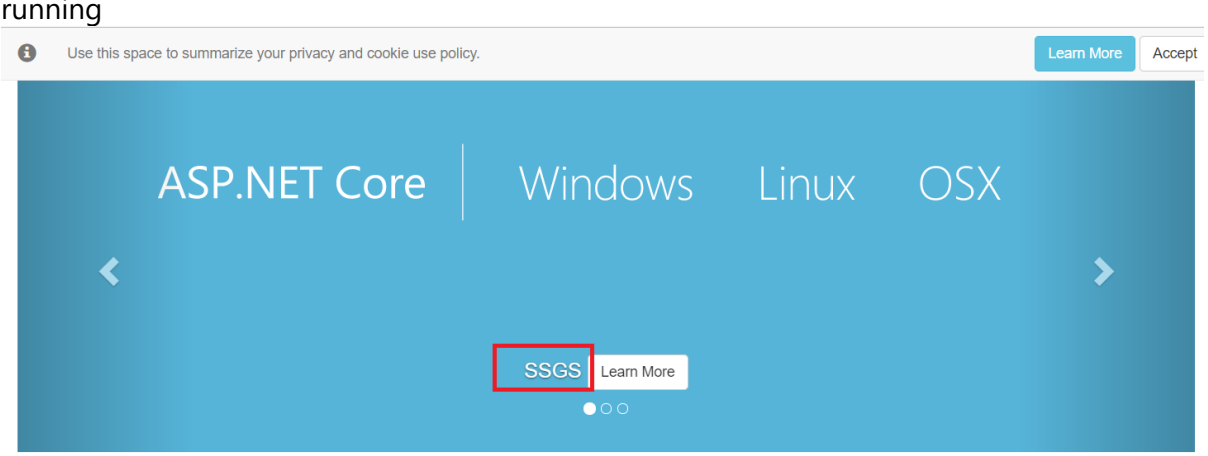
22. Save the release definition with proper name (ensure that the agent selected is ubuntu) and create release
23. After release succeeds, start cloud shell and provide credentials by using az aks get-credentials --resource-group <RG name> --name <cluster name> You should get

```
Merged "AKSDemo" as current context in /home/gouri/.kube/config
```

24. Enter commands kubectl get pods and kubectl get services

```
PS Azure:\> kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
aksdemo-b758d4d97-kk4wn     1/1     Running   0          4m3s
Azure:/
PS Azure:\> kubectl get services
NAME          TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)        AGE
aksdemo-app   LoadBalancer   10.0.176.182   52.191.222.5   80:32401/TCP   4m14s
kubernetes    ClusterIP      10.0.0.1       <none>         443/TCP        56m
```

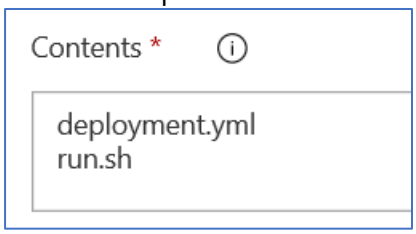
25. Select the external IP address and enter in browser, you should see the application running



26. I had made one change in index.cshtml which is being shown in the above diagram

    Remaining steps can be done later if time permits – to create complete CI CD for AKS

27. Add a file named run.sh to manifest folder using Visual Studio and enter
    sed -i "s/latest/$1/1" _aksbuilddemo/drop/deployment.yml
28. We need to publish this file to drop folder so change the copy file as

    Change the build trigger to CI and just save the build definition
29. Commit and push the changes so that build will be automatically rejiggered.
30. Let us take care of having latest tag in release definition, add bash script task to release definition before kubectl apply. (this can be configured only after build is successful so as to have run.sh in drop folder).