

Run Automation with Azure Kubernetes Service

This page will describe the steps we have followed to run the GHQ Automation Script within a Docker Container deployed in Azure Kubernetes Cluster. After going through this page you can understand the different commands that we have used for Azure Kubernetes Service (AKS).

The Problem Statement:

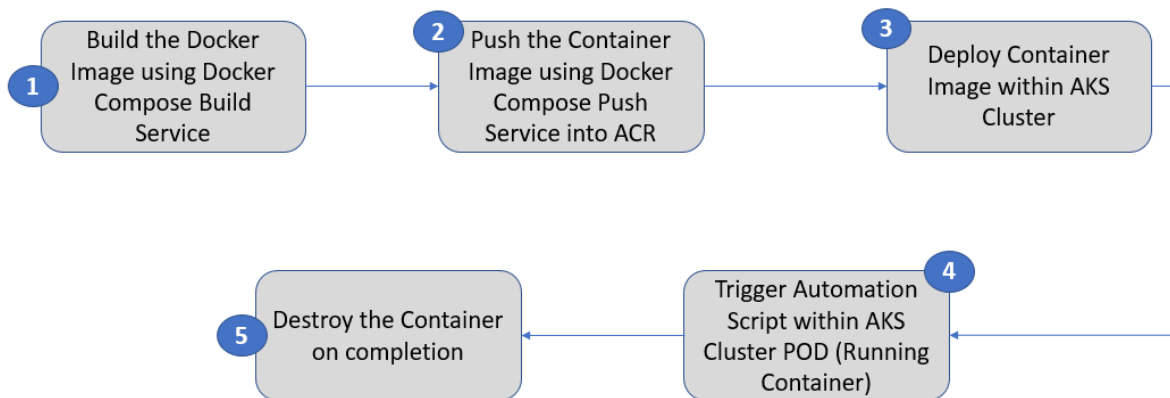
For GHQ Automation we have a limitation to bypass the single sign on step within UAT when it runs through Azure Build client. As a result it is not possible to simulate the exact end user environment within UAT as it will always ask for Single sign on details.

Solution Approach:

To overcome this challenge we have decided to run the GHQ Automation within a Docker Container which has a public IP associated and run the GHQ Test without any Single sign on dependency. As a pre-requisite we have created a containerized image explained at another page. After building and pushing the image within Azure Container Registry (ACR) we need to deploy the image at some place and run the desired automation script. TAL has decided to leverage AKS to deploy a container and run the test within the running containers.

The below diagram shows a very high level diagram of the overall workflow that we are going to achieve using AKS.

Overall Pipeline Workflow

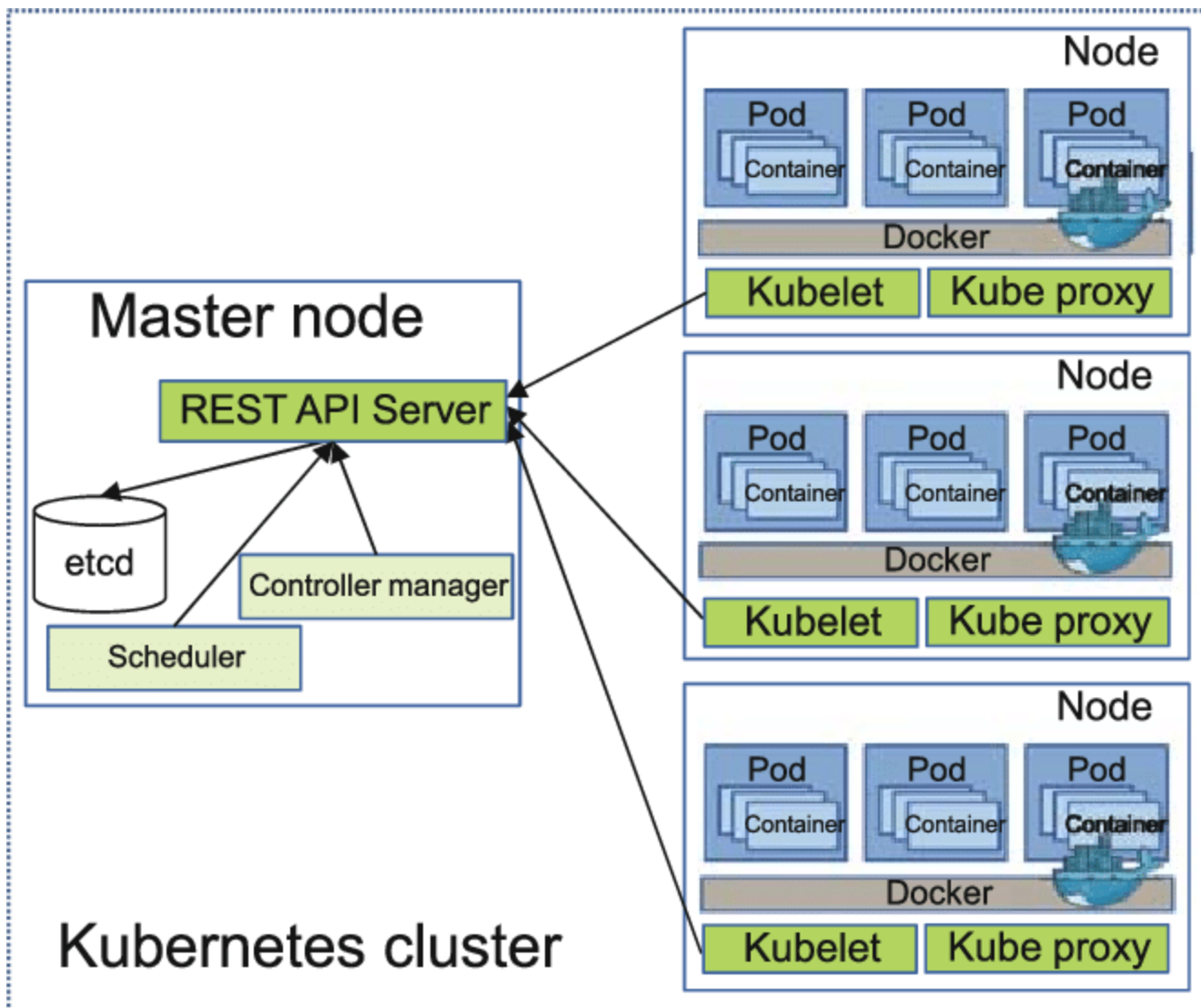


Azure Kubernetes Service (AKS):

Kubernetes is a container management and orchestration tool using which you can achieve majorly 3 most important requirement for any application deployment. Whenever you need to deal with multiple container with same configuration or different configuration and want to establish a link across all containers then Kubernetes comes into picture.

1. **Higher Availability:** Kubernetes will always ensure that your applications are running and available all the time
2. **Higher Scalability:** You can always scale up your environment based on the need you have. The scaling can be done in both manual way and automated way
3. **Higher Disaster Recovery:** Whenever there will be any issue with running POD Kubernetes will ensure to restart the container automatically and make the application running properly without any manual interval.

The overall Kubernetes architecture can be divided into 2 major blocks. The below diagram gives an overview of the overall Kubernetes Architecture.



- **Master Node**

Master Node is the main controller block which will allow user to deal with different container and application service related request with Kubernetes. The **etcd** component is responsible for storing all the status information related to master and nodes. The **Rest API component** is responsible to communicate with different node to work with containers. The **Scheduler** decides which container needs to be deployed at what node based on the available resource of the node. The **Controller manager** always monitors the health of the node and based on any failure it sends the request to scheduler to execute the task.

- **Node**

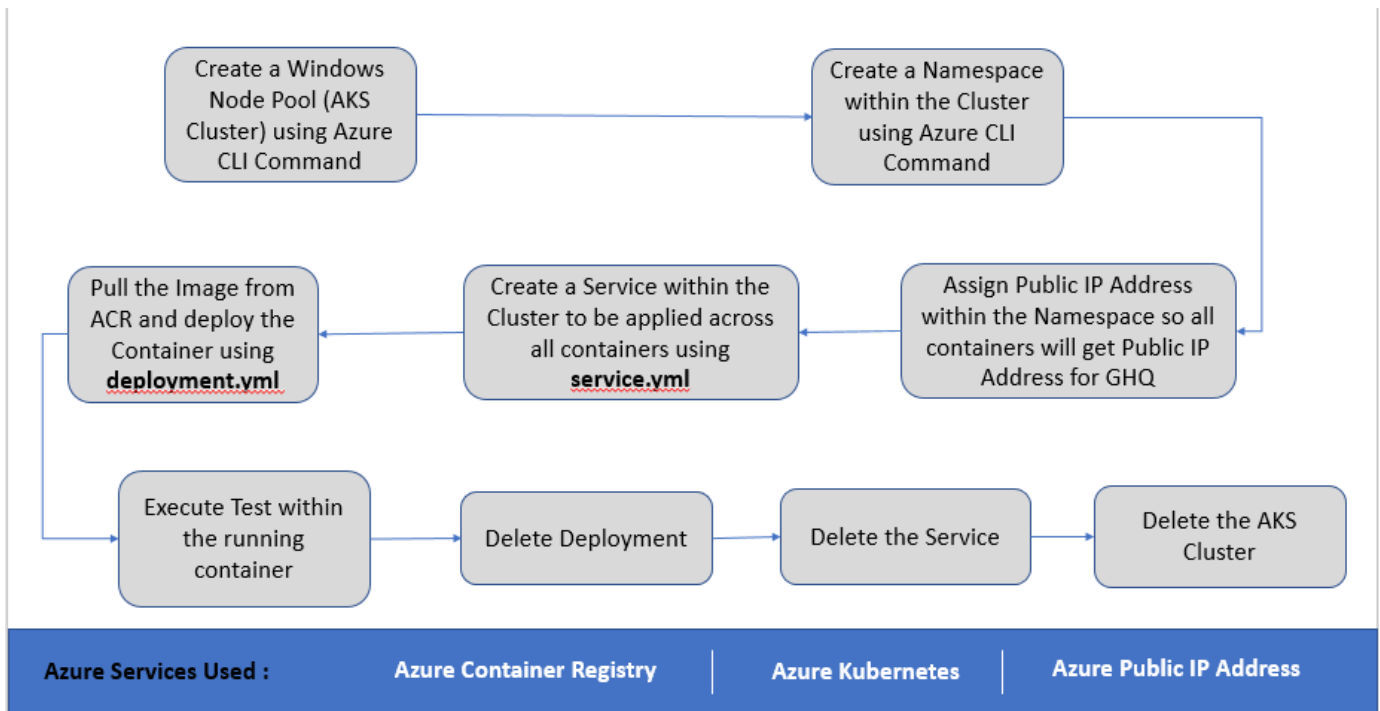
Node Server is the main server where all the container related tasks are getting executed. This is also called as **Working Node**. All the communication that happens from Master Node is received by **Kubelet** component installed within the node. Within Node when the containers are deployed and running then that instance is called as **POD**. Based on the replication factor multiple nodes gets created. Also, within the nodes we can create multiple POD based on the deployment configuration we will be defining. **Kube Proxy** is responsible for dealing with the network you want to assign. It will create a link with the Host Network on top of which the nodes are running. In our case Kube Proxy will enable us to assign Public IP to all the running containers.

For more details basic concept of AKS you can refer to <https://docs.microsoft.com/en-us/azure/aks/concepts-clusters-workloads>

Within Azure Kubernetes Service Azure provides the full service of Master Node. So, from AKS you can control all the Nodes and respective POD. Whenever we creates an AKS Cluster within Azure it will automatically create at least 1 Node within the cluster by default.

Our Solution Task:

In our solution approach we will be creating a se of task within Kubernetes Cluster to run the automation script using the container image that we built earlier and pushed to ACR. The below diagram will give you a high level idea about the task we need to execute within Kubernetes cluster.



All the tasks that are shown in the above diagram is achieved using Azure CLI Command and Kubectl command using a PowerShell script. In the below section we will talk about the individual section of the PowerShell script. In TAL we will have a Azure Kubernetes Cluster already created like shown below and the PowerShell script will be using that cluster only to perform all operation on top of the cluster.

Home >

AKSperf ...
Kubernetes service

Search (Ctrl+/) << Connect Delete Refresh

Overview

- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Security

Kubernetes resources

- Namespaces
- Workloads
- Services and ingresses
- Storage
- Configuration

Settings

- Node pools
- Cluster configuration

Essentials

Resource group (change)	: aksperf-uat-rg	Kubernetes version	: 1.20.7
Status	: Succeeded	API server address	: aksperf-aksperf-uat-rg-48d91c-d6ca86e8.hcp.
Location	: Australia East	Network type (plugin)	: Azure CNI
Subscription (change)	: TAL non-prod	Node pools	: 1 node pool
Subscription ID	: 48d91cc7-1330-47d4-8894-2bbdf23e91bf		
Tags (change)	: application-service : system business-service : infrastructure-services data-classification : confidential		

Properties Capabilities

Kubernetes services		Networking	
Encryption type	Encryption at-rest with a platform-managed key	API server address	aksperf-aksperf-uat-rg-48d91c-d6ca86e8.hcp.australiaeast.azmk8s.io
Virtual node pools	Not enabled	Network type (plugin)	Azure CNI
Node pools		Pod CIDR	-
Node pools	1 node pool	Service CIDR	192.168.8.0/22
Kubernetes versions	1.20.7	DNS service IP	192.168.8.10
Node sizes	Standard_DS2_v2	Docker bridge CIDR	172.17.0.1/16

You can find 3 different files within our Solution Package under TAL.GLS.WebAutomation Solution.

- **deployment.yml File Overview:**

Deployment is the blueprint for all the POD that you can create. Using deployment file you can define the image that you want to deploy and also you can set the replication factor for the POD. The below section will describe about the details of the deployment file that we have used.

Deployment file has 3 major sections.

Metadata: In this metadata section it specifies the name of the deployment and also you can define the namespace. Namespace within a Node Server is logical grouping of the resources. So, if there are multiple teams working on the same AKS Cluster they can have their respective namespaces and only they can access the individual resources within cluster.

Spec: In this section you can define the replication factor. By default it is 1. If you make the replication factor as 2 or 3 then when after deployment 2/3 containers will be running with same image configuration within the Node Server.

Template: template talks about the image that you want to use for Container and also other properties and task you want to associate with the container image.

In our case we will be using the deployment file where we need 1 POD to run for the time being and also we will be using the images pulled from ACR. The below section talks about the different section of the deployment yml file.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: <appname>
  namespace: <namespace>
```

At the very first it will define the version of the API Service that will be used. Within **kind** we need to define what kind of activity we are going to work. Here it is deployment which means it will pull some image and run the containers.

Within **metadata** section we have provided a **name** and **namespace** name. Those are variables and it will be passed during pipeline run by updating some runtime value.

```
spec:
  replicas: 1
  selector:
    matchLabels:
      kubernetes.io/os: windows
```

In the **spec** section you can see we have set **replica** as 1. That means there will be only 1 POD which will be running from the image that we will specify. Within spec we can also specify the **selectors**. So, for any case if you have multiple image to deal with you can specify the selector and it will pull that image only for which the selector will **match**. Here we are giving the value of the key as windows. Now within the template section it will find for the container image section with this windows key only and deploy that image.

```

template:
  metadata:
    labels:
      kubernetes.io/os: windows
  spec:
    containers:
      - name: <podname>
        image: aksperfcr.azurecr.io/ghqtestautomation_test:latest
        command:
          - powershell.exe
          - -command
          - 'ping 127.0.0.1 -t'
        imagePullPolicy: Always
        ports:
          - containerPort: 80
            name: http
            protocol: TCP
        imagePullSecrets:
          - name: file-secret

```

Within the template we define the blueprint of the Containers that will be running. So, at the very first section we are defining the **metadata** of the container image. You can see that we have specified the same key value pair that we have mentioned within the selector section of main deployment. The spec section defines the specification of the image that we will be running. The below section is describing the individual key details within **spec** section.

name: It will show as the name of the POD when we will run the POD. We will specify the POD name from PowerShell Script

image: What image it will pull to deploy into cluster node. You can see we are using the ACR Image that we have built earlier.

command: within command section you can specify some command which will get executed after the container will run. In this case I am pinging to localhost so that once POD will be running I can see some activity within the container image. If the Container image has nothing to do then Kubernetes will automatically terminate the POD.

imagePullPolicy: We are telling always it will pull the image from ACR

ports: This is not required for our Automation run. But, if you want any port to be open within the POD then you can specify that port to be open for external communication.

imagePullSecret: To download the image from ACR you need to use credentials. We will be using Kubernetes secret to authenticate the image pull using this file. It will be more explained when we will go over the PowerShell command in later section

- **testservice.yml File Overview:**

In Kubernetes POD we apply the common network policy or port mapping through service file. Our requirement in this case is to assign public IP to all the POD that will get created for Automation Run. So, we will define a Loadbalancer job to the service which will get applied across all the running POD

```

apiVersion: v1
kind: Service
metadata:
  name: test-ghq
  namespace: <namespace>
spec:
  loadBalancerIP: <ipaddress>
  externalIPs: [<externalIP>]
  type: LoadBalancer
  ports:
    - port: 80

```

In the above file you can see we have defined service as **kind** within the yml. So, this will create a service within AKS node. Within the metadata section we have given a service name and mention the namespace too.

Within **spec** section we have defined the service **type** as **LoadBalancer** which will tell the node that the IP will be distributed using LoadBalancer. Also, the Loadbalancer IP will be updated from Powershell script with a public IP. We will also use the same IP as **external IP**. For Automation run the external IP is required as we want to run the test within a public network. We have kept the port 80 as open but it is not required for our case.

- **Test_AKS.ps1 File Overview:**

This is the PowerShell script through which we will be executing all AKS related task. Using this PowerShell script we will be performing the following tasks.

1. *Creation of New Node Pool (Node Pool is same as Node Server)*
2. *Creation of namespace within the node pool*
3. *Create a deployment which will pull the image and run the container*
4. *Create a Load Balancer service using Public IP*
5. *Execute the test within the running container (POD)*
6. *Delete the deployment*
7. *Delete the service*
8. *Delete the namespace*
9. *Delete the Node Pool*

Initial Variable Declaration and Value Assignment within PowerShell Script:

As a initial setup we are defining some variable that we will use for Node Pool Name, Namespace name etc. As of now the Release name is hardcoded. But, later we can replace with Pipeline variable. We are first loading the variable from the file and assign the value of the variable by combining multiple values. Then we are printing all the variables.

```

$TEST = "GHQTest"

$RELEASE_RELEASENAME = "Release-11"
$TestPath = split-path -parent $MyInvocation.MyCommand.Definition
Write-Host "The Path is : " + $TestPath
# Defining the path for Powershell Script variable file
$varfile=".\\TAL.GLS.TestAutomation\\TAL.GLS.WebAutomation\\AKS_Variable.
ps1"

# Checking if the file is present in that path or not
if ( !(Test-Path "$varfile") )
{
    Write-Host "ERROR: AKS_Variable.ps1 file doesn't exist "
    exit 1
}

```

```

}
# Loading all the variable defined within AKS_Variable.ps1 file
. $varfile

# Checking if the TEST Variable is initialized with any value or not
if ($TEST -eq $null)
{
    Write-
Host "ERROR: TEST variable is not set in pipeline. Please check again ."
    exit 1
}

# Getting the Date and Time in Australia Time Zone and convert into specified format
$currentTime=[System.TimeZoneInfo]::ConvertTimeBySystemTimeZoneId
([DateTime]::Now,"AUS Eastern Standard Time")
$testtimestamp = $currentTime.ToString("yyMMdHm")

# Nodepool name and Namespace will be created using TEST Variable and the Release Number
$poolname=$TEST.substring(0,4).tolower() + $RELEASE_RELEASENAME.
substring($RELEASE_RELEASENAME.IndexOf('-')+1,2)
$namespace=$poolname
$resdir="$TEST-$testtimestamp"

# Printing all the variable that has come from the file and defined above
echo "\
***** Test Configs *****
*****   TimsStamp: $testtimestamp
*****   AKS Name: $AKS_NAME
*****   Node Size: $NODESIZE
*****   Number of Nodes: $NODECOUNT
*****   Jmeter Images Tag: $TAG_ID
*****
*****   Namespace: $namespace
*****   Pool Name: $poolname
*****   Result folder: $resdir

*****   Test Item: $TEST"

```

VARIABLE SCRIPT:

Below given the snapshot of variable script.

```
# Script create to define variables for the Member Online First State Super solution
```

```
$AKS_RG="aksperf-uat-rg"
$AKS_NAME="AKSperf"
$NODEPOOL_RG="aksperfnodes-uat-rg"
$LOCATION="australiaeast"
$NODESIZE="Standard_DS2_v2"
$NODECOUNT=1
$OSTYPE="Windows"
$PODNAME="win-testrunner"
$APPNAME="win-webserver"

$ACR_NAME="aksperfcr"
```

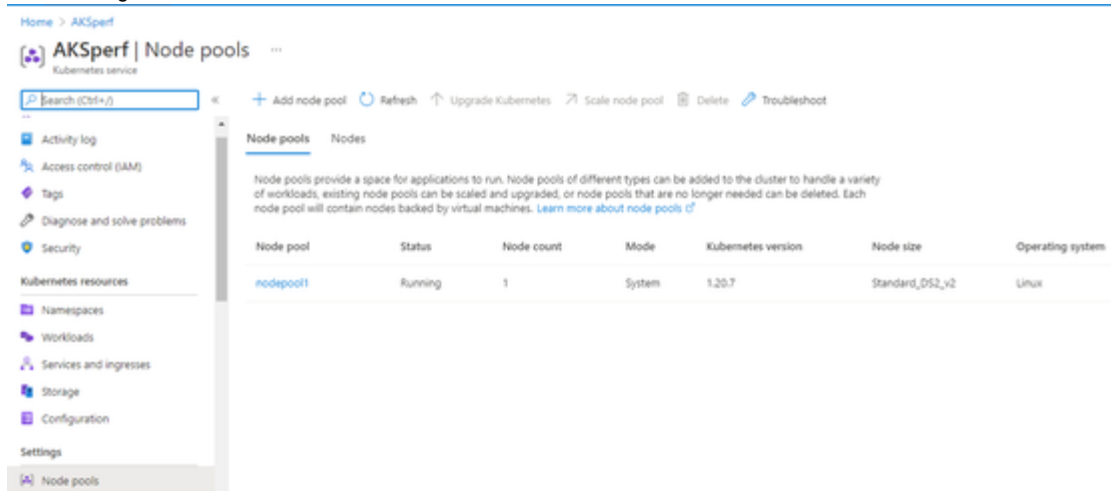
Creating the Node Pool:

Within AKS within each cluster we can create multiple Node Pool. By default you can have one single Node Pool created. Here we are creating a Node Pool within the cluster using Azure CLI command.

```
echo "Create new nodepool"
az extension add --name aks-preview
az aks nodepool add --resource-group $AKS_RG --cluster-name $AKS_NAME --name $poolname --node-count $NODECOUNT --node-vm-size $NODESIZE --os-type $OSTYPE
```

Before you will interact with cluster using Azure CLI command you have to install the **aks-preview** extension. After that using **az aks nodepool add** command we are creating a node pool within the cluster. You can see we are passing the value of the arguments from the variable taken from the Variable file like AKS_RG, AKS_NAME, poolname etc. You need to keep 3 important argument while creating the nodepool. **NODECOUNT** signify the number of parallel node you want to create which will be used for scale up situation. In this case we have used it 1 as mentioned in variable file. **NODESIZE** signify the type of hardware you want to use as Node. **OSTYPE** is the base OS you want to use on top of which you will run the POD. Please make sure for Windows container image we have to have a node of Windows type.

Once you will create a node pool you verify that will get created at the section shown below. Please ensure that the OS type of your nodepool is showing as Windows.



The screenshot shows the Azure portal interface for the AKSperf cluster. The left sidebar contains navigation links for various services. The main content area is titled 'AKSperf | Node pools' and includes a search bar and action buttons like 'Add node pool', 'Refresh', 'Upgrade Kubernetes', 'Scale node pool', 'Delete', and 'Troubleshoot'. Below this, there's a table listing the node pools. The table has columns for Node pool, Status, Node count, Mode, Kubernetes version, Node size, and Operating system. One node pool, 'nodepool1', is listed with a status of 'Running', a node count of 1, mode 'System', Kubernetes version 1.20.7, node size 'Standard_DS2_v2', and operating system 'Linux'.

Node pool	Status	Node count	Mode	Kubernetes version	Node size	Operating system
nodepool1	Running	1	System	1.20.7	Standard_DS2_v2	Linux

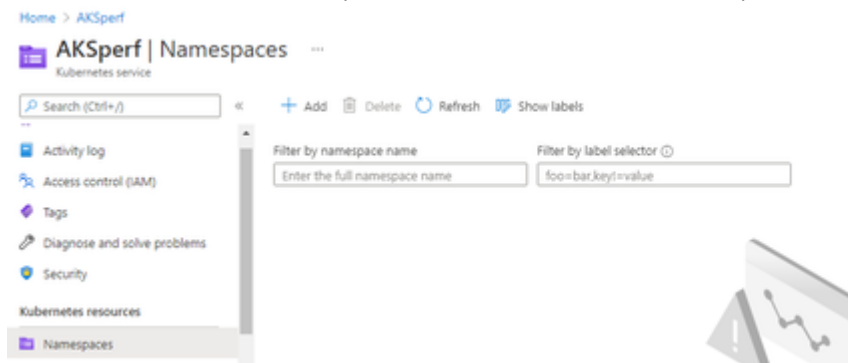
Creating Namespace:

After we create the nodepool we will create the namespace. Keep in mind that after nodepool creation when we will do rest of the step on top of that nodepool those needs to be executed using **kubect**l command. We are no longer using the Azure CLI command.

```
kubectl create namespace $namespace

if(! $? )
{
    Write-Host "[Error:]The namespace has not created successfully"
    exit 1
}
```

You can notice within the script that after executing each command we are checking for the process to execute properly. Else we will exit from the execution. After creation of namespace that can be verified within the namespace section of AKS.



Create the Service:

Before we will create the deployment we need to create the service that will get applied in each POD of the node. The below section describes the command we have used for creating the service. Remember we have already created a service YML file. We need to update the IP address in that file and then we can create the service using that YML file.

```

$ipaddress=$(az network public-ip show -g $NODEPOOL_RG --
name egressPublicIP --output tsv --query "ipAddress")

Write-Host "The Public IP Address is: " $ipaddress

(Get-Content TAL.GLS.TestAutomation\TAL.GLS.WebAutomation\testservice.
yaml).replace('<namespace>', $namespace) | Set-Content TAL.GLS.
TestAutomation\TAL.GLS.WebAutomation\testservice.yaml
(Get-Content TAL.GLS.TestAutomation\TAL.GLS.WebAutomation\testservice.
yaml).replace('<ipaddress>', $ipaddress) | Set-Content TAL.GLS.
TestAutomation\TAL.GLS.WebAutomation\testservice.yaml
(Get-Content TAL.GLS.TestAutomation\TAL.GLS.WebAutomation\testservice.
yaml).replace('<externalIP>', $ipaddress) | Set-Content TAL.GLS.
TestAutomation\TAL.GLS.WebAutomation\testservice.yaml

# Creating the Service that will be applicable to all deployed Container
kubectl create -f TAL.GLS.TestAutomation\TAL.GLS.
WebAutomation\testservice.yaml
if(!$?)
{
    Write-Host "[Error:]The Service has not created successfully"
    exit 1
}
Do{
    $serviceStatus = kubectl -n $namespace get services | Select-
String -Pattern test
    $serviceStatus = $serviceStatus.ToString() -split " "
    $status = $serviceStatus[6] | Select-String -Pattern "pending"
    Start-Sleep -s 2
    echo $status
}
While (!$?)

```

Initially we are taking an Public IP Address from the Public IP Pool that is available as part of Azure IP Address Service. TAL has an IP Service called **engressPublicIP** using which you can get an Public IP that can be applied to all the Nodes. The below diagram shows the IP Service.

Home >



Public IP addresses ...

TAL Limited

[+ Create](#)
[Manage view](#)
[Refresh](#)
[Export to CSV](#)
[Open query](#)
[Assign tags](#)
[Delete](#)

Subscription == all
Resource group == all
Location == all
[Add filter](#)

Showing 1 to 2 of 2 records.

<input type="checkbox"/> Name ↑↓	Resource group ↑↓
<input type="checkbox"/>  egressPublicIP	aksperfnodes-uat-rg
<input type="checkbox"/>  egressPublicIPSecondary	aksperfnodes-uat-rg

Once the public IP will get created the script will update the Public IP Address section and the Loadbalancer IP section within the testservice.yml file. After updating the YML file we are creating the service using **kubectl** command. After creation of the service the script is looking for the service status and wait till it will remain in pending state.

Creating the Deployment:

Once the service is up and running then we can go ahead and do the deployment. Deployment is nothing but pulling the image from ACR and create the POD as specified within the deployment YML file. You remember that within deployment file we have some variables to update. We will first do that and then we will create the deployment.

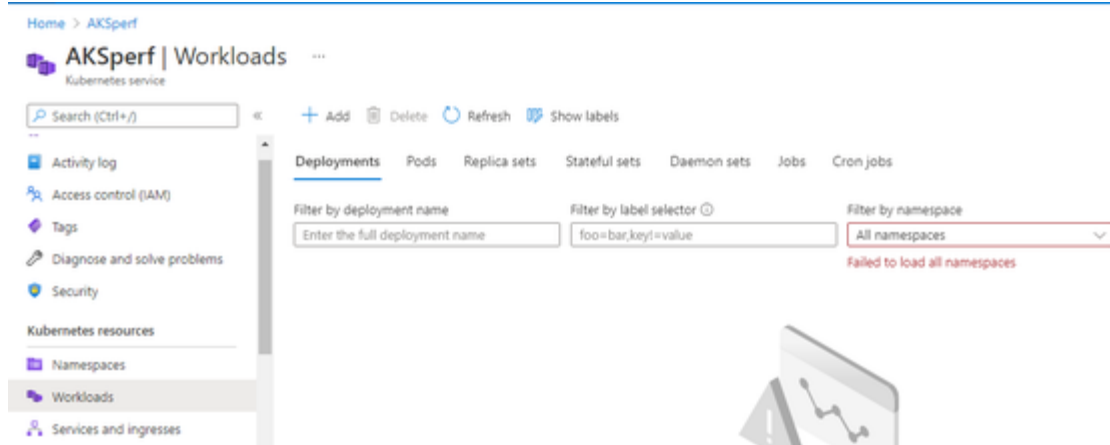
```
kubect1 -n default get secret file-secret -o yaml | % {$_replace
(namespace: default", "namespace: " + $namespace)} | out-file secret.
yaml
kubect1 -n $namespace create -f secret.yaml

(Get-Content TAL.GLS.TestAutomation\TAL.GLS.WebAutomation\deployment.
yaml).replace('<podname>', $PODNAME) | Set-Content TAL.GLS.
TestAutomation\TAL.GLS.WebAutomation\deployment.yaml
(Get-Content TAL.GLS.TestAutomation\TAL.GLS.WebAutomation\deployment.
yaml).replace('<namespace>', $namespace) | Set-Content TAL.GLS.
TestAutomation\TAL.GLS.WebAutomation\deployment.yaml
(Get-Content TAL.GLS.TestAutomation\TAL.GLS.WebAutomation\deployment.
yaml).replace('<appname>', $APPNAME) | Set-Content TAL.GLS.
TestAutomation\TAL.GLS.WebAutomation\deployment.yaml

# Deploying the Container from ACR into created Nodepool
kubect1 create -f .\TAL.GLS.TestAutomation\TAL.GLS.
WebAutomation\deployment.yaml
```

At very first section we are getting the secret of default namespace to authenticate Azure service. Our goal is to apply the same credential that got created for default namespace in our newly created namespace. Within deployment YML file we have used the secret to pull the image. This credentials will help to download the image. **Default** namespace is the namespace that gets created when we create the AKS cluster in Azure. Once the secret is getting created we are applying that authentication using kubect1 command in our namespace.

After secret will get created we are updating the deployment YML file and creating the deployment using kubect1 command. You can see all of your deployment under **workload** section in AKS.



Checking the POD Status:

Once we create the deployment it will take some time to pull the image from ACR and deploy it into different containers within nodepool. Within the script we are checking for the status and wait till the POD status will be set to **Running**.

```
$podContainer="Test"

While ($true){
    $podCount = kubectl -n $namespace get pods | Select-String -
Pattern $APPNAME | Measure-Object -Line
    if($podCount.ToString() -le 0){
        Start-Sleep -s 10
        continue
    }
    $podDetails = kubectl -n $namespace get pods | Select-String -
Pattern $APPNAME
    $podStatus = $podDetails.ToString() -split " "
    if($podStatus[8] -ne "Running"){
        Start-Sleep -s 10
        Write-Host "The State is "$podStatus[8]" wait...."
        continue
    }
    # Getting the POD Name
    $podContainer=$podStatus[0]
    break
}
```

In the first section we are verifying the POD count. It should not be zero as we are creating 1 POD. Till the count will become 1 we will wait. After the POD count will give the count we will check the status of the POD. The POD will take some time to come into Running state. So, till it will come into running state we will wait in the script. Once it will go into **Running** state we will take the POD name that will be used later to execute automation script.

Execute the Automation Script:

Finally we have come into the state that we thought to achieve. The below command will run the automation script using the category passed within the running container.

```
kubectl --namespace $namespace exec $podContainer -- powershell.exe -command "cd C:
\testProject; vstest.console.exe SeleniumDemoTests.dll /TestCaseFilter:'TestCategory=Smoke'"
```

You can see that within kubectl command we are passing the namespace and POD name where we will be executing our script. Within powershell command we are first going into testProject folder that we have mentioned in Dockerfile workspace directory and execute the VSTestconsole command to trigger the execution.

Cleaning up all Resources:

It is always a good practice to clean up all the resources once you are done with your job to free up the resource from Azure. The following section of the script is deleting all the resources like Deployment, service, namespace and nodepool to make the AKS into earlier state. Keep in mind the the order you maintained to create the resources during deletion you have to follow the reverse order.

```
Write-Host "Cleaning Resources"

#Deleting the deployment
$podDetails = kubectl -n $namespace get pods | Select-String -
Pattern $APPNAME
$podStatus = $podDetails.ToString() -split " "
if($podStatus[8] -eq "Running"){
    kubectl delete -f .\TAL.GLS.TestAutomation\TAL.GLS.
WebAutomation\deployment.yaml
}

#Deleting the service
$serviceDetails = kubectl -n $namespace get services | Select-String -
Pattern test-aks | Measure-Object -Line
if($serviceDetails.ToString() -ne 0){
    kubectl delete -f .\TAL.GLS.TestAutomation\TAL.GLS.
WebAutomation\testservice.yaml
}

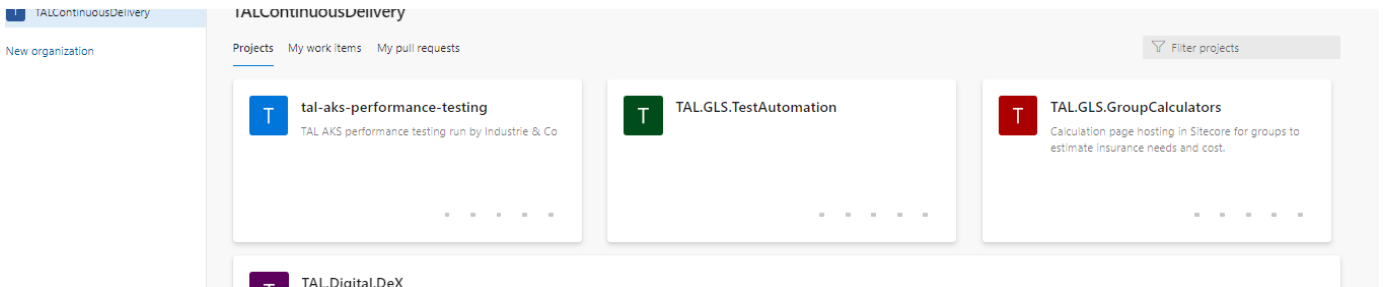
#Deleting the namespace
$namespaceDetails = kubectl get namespaces | Select-String -
Pattern $namespace | Measure-Object -Line
if($namespaceDetails.ToString() -ne 0){
    kubectl delete namespace $namespace
}

#Deleting the nodepool
$nodepoolDetails = az aks nodepool list --cluster-name $AKS_NAME -
g $AKS_RG --query "[].name" | Select-String -
Pattern $poolname | Measure-Object -Line
if($nodepoolDetails.ToString() -ne 0){
    az aks nodepool delete --resource-group $AKS_RG --cluster-
name $AKS_NAME --name $poolname --no-wait
}
```

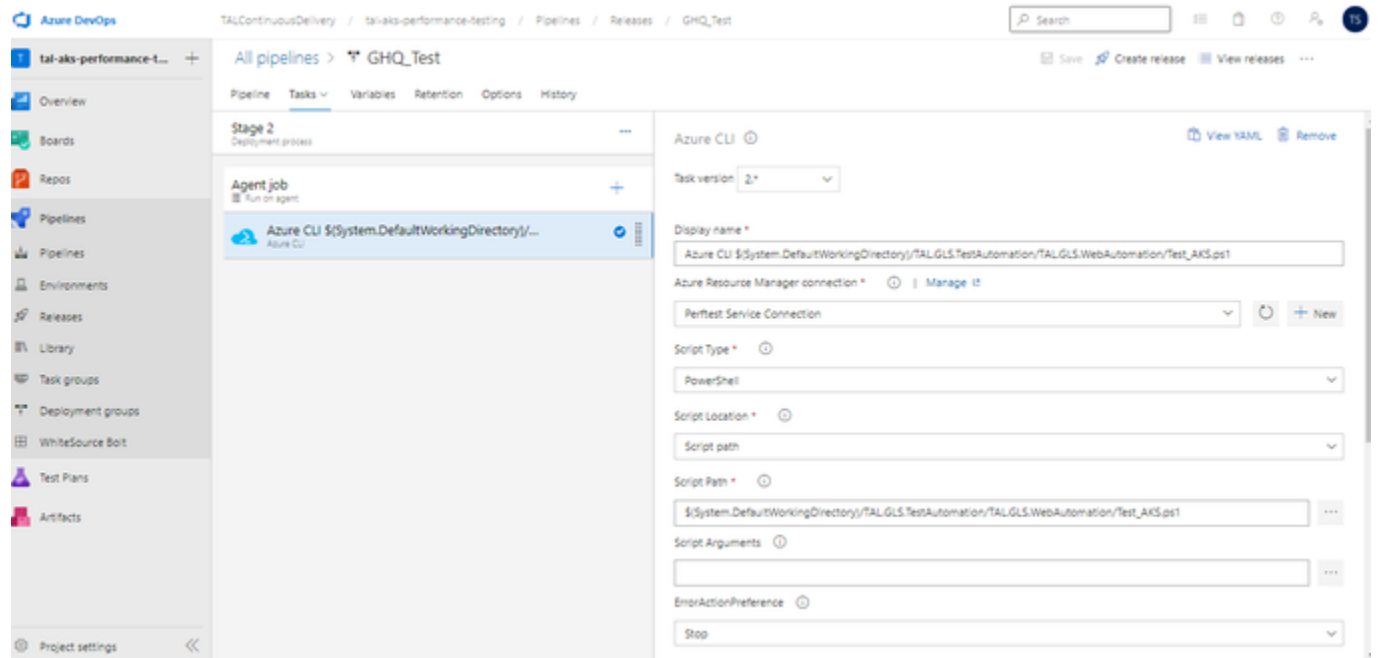
Creating a Release Pipeline to Automate the Entire Process:

Once we are done with all of these activity we can create a Release Pipeline which will automate the entire process using the PowerShell script we have developed.

Keep in mind that TAL.GLS.Automation project doesn't have access to AKS. The Performance Test project has the access to AKS. So, during POC we have created pipeline within the Performance Project.



Within Release pipeline you can add Azure CLI Task and point to the PowerShell script to execute all the task. Using this way you can automate the entire process within Release pipeline.



Some AKS Commands:

You can create a AKS Cluster to deploy Windows Container using Windows Admin Credentials

```
az aks create -g MYNewResourceGroup -n akstsengupt --load-balancer-sku Standard --network-plugin azure --windows-admin-username tsengupt82 --windows-admin-password Tathagata1982@h --generate-ssh-keys
```

Attach an ACR with AKS

```
az aks update -n akstsengupt -g MYNewResourceGroup --attach-acr tsenguptacr
```

Check the COnnectivity with attached ACR

```
az aks check-acr --name akstsengupt --resource-group MYNewResourceGroup --acr tsenguptacr.azurecr.io
```

Delete the Deployment

```
kubectl delete -f .\deployment.yaml
```

Delete the Service

```
kubectl delete -f .\testservice.yaml
```

Delete the namespace

kubectl delete namespace test76

##Delete the pool

az aks nodepool delete --resource-group MYNewResourceGroup --cluster-name akstsengupt --name test76 --no-wait

To see the deployment details

kubectl --namespace test76 describe deployments

To see the POD along with status

kubectl --namespace test76 get pods

It will return the POD matching the pattern

kubectl -n test76 get pods | Select-String -Pattern win-testrunner

Copy from POD to Local System

kubectl --namespace test76 cp win-webserver-587ffd5969-lgxl2:/testProject/installchrome.ps1 ./abcd.ps1

To execute command in Powershell

kubectl --namespace test76 exec win-webserver-587ffd5969-lgxl2 -- powershell.exe -command "cd testProject; vstest.console.exe SeleniumDemoTests.dll /TestCaseFilter:'TestCategory=Smoke'"

GHQ container Handover session Walkthrough by Tatha: <https://web.microsoftstream.com/video/c957c245-6b76-4c90-ac6b-714bfad21dc2>