# Input bindings are passed in via param block.

param($Timer)

# Get the current universal time in the default string format.

$currentUTCtime = (Get-Date).ToUniversalTime()

# The 'IsPastDue' property is 'true' when the current function invocation is later than scheduled.

if ($Timer.IsPastDue) {

    Write-Host "PowerShell timer is running late!"

}

$VerbosePreference = "Continue"

$ProgressPreference = "Continue"

# Write an information log with the current time.

Write-Host "PowerShell timer trigger function ran! TIME: $currentUTCtime"

Connect-AzAccount -Identity

Select-AzSubscription -SubscriptionName '<Enter Subscription Name>'

[string] $ResourceGroupName='<Enter Resource Group Name>'

[string] $WorkspaceName='<Enter Synapse Workspace Name>'

[string] $Action='Stop'

[array] $IncludeSqlPoolNames = $null

[array] $ExcludeSqlPoolNames = $null

[boolean] $Wait = $true

$startTime = Get-Date

Write-Host "$Action initiated for $ResourceGroupName/$WorkspaceName"

Import-Module 'Az.Synapse'

# get synapse workspace

try {

    $workspace = Get-AzSynapseWorkspace -ResourceGroupName $ResourceGroupName -Name $WorkspaceName

    if (-not $workspace) {

        Write-Error "Workspace ($ResourceGroupname/$WorkspaceName) not found"

        return

    }

} catch {

    throw "Error getting the Synapse Workspace ($ResourceGroupName/$WorkspaceName)"

}

# get sql pools

try {

    $sqlPools = Get-AzSynapseSqlPool -ResourceGroupName $ResourceGroupName -WorkspaceName $WorkspaceName

    if ($sqlPools -and $SqlPoolName) {

        $sqlPools = $sqlPools | Where-Object { $SqlPoolName -contains $\_.SqlPoolName }

        if (-not $sqlPools) {

            Write-Warning "No SQL Pools found matching $($SqlPoolName)"

            return

        }

    }

    if (-not $sqlPools) {

        Write-Warning 'No SQL Pools found'

        return

    }

} catch {

    throw "Error getting the Synapse Workspace ($ResourceGroupName/$WorkspaceName)"

}

# include only names that match IncludeSqlPoolNames

if ($IncludeSqlPoolNames) {

    $sqlPools = $sqlPools | Where-Object { $IncludeSqlPoolNames -contains $\_.SqlPoolName }

}

# exclude any names that match ExcludeSqlPoolNames

if ($ExcludeSqlPoolNames) {

    $sqlPools = $sqlPools | Where-Object { $ExcludeSqlPoolNames -notcontains $\_.SqlPoolName }

}

Write-Verbose "SQL Pools to $($Action): $($sqlPools.SqlPoolName -join ', ')"

# Loop through the SQL Pools

$jobs = @()

foreach ($sqlPool in $sqlPools) {

    if ($Action -eq 'Stop') {

        if ($sqlPool.Status -eq 'Paused') {

            Write-Host "$($WorkspaceName)/$($sqlPool.SqlPoolName) already $($sqlPool.Status)"

            continue

        }

        Write-Host "Stopping $($WorkspaceName)/$($sqlPool.SqlPoolName)"

        $jobs += Suspend-AzSynapseSqlPool -WorkspaceName $WorkspaceName -Name $sqlPool.SqlPoolName -AsJob

    }

    elseif ($Action -eq 'Start') {

        if ($sqlPool.Status -eq 'Online' -or $sqlPool.Status -eq 'Resuming') {

            Write-Host "$($WorkspaceName)/$($sqlPool.SqlPoolName) already $($sqlPool.Status)"

            continue

        }

        Write-Host "Starting $($WorkspaceName)/$($sqlPool.SqlPoolName)"

        $jobs += Resume-AzSynapseSqlPool -WorkspaceName $WorkspaceName -Name $sqlPool.SqlPoolName -AsJob

    } else {

        throw "Invalid action ($Action)"

    }

}

if ($jobs.Count -eq 0) {

    Write-Host "Script complete. ($("{0:HH:mm:ss}" -f $([datetime] $($(Get-Date) - $startTime).Ticks)) elapsed)"

    return

}

# wait for jobs to complete

if (-not $Wait) {

    Write-Host "Job submitted. Check job status using Get-Job."

    Write-Host "Script complete. ($("{0:HH:mm:ss}" -f $([datetime] $($(Get-Date) - $startTime).Ticks)) elapsed)"

    return

}

$jobIds =  [System.Collections.ArrayList] @($jobs.Id)

do {

    $job = Wait-Job -Id $jobIds[0]

    Write-Host "$($job.Name) $($job.StatusMessage)"

    Remove-Job $job

    $jobIds.Remove($jobIds[0])

} until (-not $jobIds)

Write-Host "Script complete. ($("{0:HH:mm:ss}" -f $([datetime] $($(Get-Date) - $startTime).Ticks)) elapsed)"