<#

.SYNOPSIS

Azure VM Health Check using Excel sheet input

.DESCRIPTION

This runbook reads VM names from an Excel file in Azure Storage,

performs health checks, and sends reports via email and Teams.

.NOTES

Created: 13th May 2025

Author: Akash Chopra

Version: 1.0

#>

param(

[Parameter()]

[string]$ExcelFileName = "VMList.xlsx",

[Parameter()]

[int]$CpuThresholdWarning = 75,

[Parameter()]

[int]$CpuThresholdCritical = 90,

[Parameter()]

[int]$MemoryThresholdWarning = 80,

[Parameter()]

[int]$MemoryThresholdCritical = 90,

[Parameter()]

[int]$DiskThresholdWarning = 85,

[Parameter()]

[int]$DiskThresholdCritical = 95,

[Parameter()]

[int]$TimeRangeInHours = 24,

[Parameter()]

[switch]$SendEmail = $true,

[Parameter()]

[switch]$PostToTeams = $true

)

# Get Automation Variables

$storageAccountName = Get-AutomationVariable -Name "StorageAccountName"

$keyVaultName = Get-AutomationVariable -Name "KeyVaultName"

$clientEmailAddress = Get-AutomationVariable -Name "ClientEmailAddress"

$teamEmailAddress = Get-AutomationVariable -Name "TeamEmailAddress"

$smtpServer = Get-AutomationVariable -Name "SMTPServer"

$smtpPort = Get-AutomationVariable -Name "SMTPPort"

$teamsWebhookUrl = Get-AutomationVariable -Name "TeamsWebhookUrl"

# Get Email Credential

$emailCredential = Get-AutomationPSCredential -Name "EmailCredential"

# Function to write log messages

function Write-Log {

[CmdletBinding()]

param(

[Parameter(Mandatory=$true)]

[string]$Message,

[Parameter()]

[ValidateSet('Info', 'Warning', 'Error')]

[string]$Level = 'Info'

)

# Create timestamp

$timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"

# Format message

$logMessage = "[$timestamp] [$Level] $Message"

# Output to console based on level

switch ($Level) {

'Info' { Write-Output $logMessage }

'Warning' { Write-Warning $logMessage }

'Error' { Write-Error $logMessage }

}

# Also capture in our report data

$script:logMessages += [PSCustomObject]@{

Timestamp = $timestamp

Level = $Level

Message = $Message

}

}

# Initialize log collection array

$script:logMessages = @()

# Connect to Azure with Automation account's managed identity

try {

Write-Log "Connecting to Azure with Managed Identity"

Connect-AzAccount -Identity

# Get Key Vault SAS token

$storageAccountSasToken = Get-AzKeyVaultSecret -VaultName $keyVaultName -Name "storage-sas-token" -AsPlainText

# Get current subscription

$currentSubscription = (Get-AzContext).Subscription.Id

Write-Log "Connected to Azure subscription: $currentSubscription"

}

catch {

Write-Log "Error connecting to Azure: $\_" -Level Error

throw "Failed to connect to Azure: $\_"

}

# Set up storage context

try {

$storageAccountKey = (Get-AzStorageAccountKey -ResourceGroupName (Get-AzResource -Name $storageAccountName).ResourceGroupName -Name $storageAccountName)[0].Value

$storageContext = New-AzStorageContext -StorageAccountName $storageAccountName -StorageAccountKey $storageAccountKey

Write-Log "Created storage context for account: $storageAccountName"

}

catch {

Write-Log "Error setting up storage context: $\_" -Level Error

throw "Failed to set up storage context: $\_"

}

# Create temp folder for downloads and uploads

$tempFolder = Join-Path -Path $env:TEMP -ChildPath "VMHealthCheck\_$(Get-Date -Format 'yyyyMMddHHmmss')"

New-Item -ItemType Directory -Path $tempFolder -Force | Out-Null

Write-Log "Created temporary folder: $tempFolder"

# Download Excel file from blob storage

try {

$excelFilePath = Join-Path -Path $tempFolder -ChildPath $ExcelFileName

Get-AzStorageBlobContent -Container "vm-lists" -Blob $ExcelFileName -Destination $excelFilePath -Context $storageContext -Force | Out-Null

Write-Log "Downloaded Excel file from storage: $ExcelFileName"

}

catch {

Write-Log "Error downloading Excel file: $\_" -Level Error

throw "Failed to download Excel file: $\_"

}

# Read VM list from Excel

try {

$vmList = Import-Excel -Path $excelFilePath

$totalVMs = $vmList.Count

Write-Log "Read $totalVMs VMs from Excel file"

# Validate Excel data

if ($totalVMs -eq 0) {

throw "No VMs found in Excel file"

}

# Check required columns

$requiredColumns = @("SubscriptionId", "ResourceGroupName", "VMName")

$missingColumns = $requiredColumns | Where-Object { $vmList[0].PSObject.Properties.Name -notcontains $\_ }

if ($missingColumns.Count -gt 0) {

throw "Missing required columns in Excel file: $($missingColumns -join ', ')"

}

}

catch {

Write-Log "Error reading VM list from Excel: $\_" -Level Error

throw "Failed to read VM list from Excel: $\_"

}

# Function to get color code based on threshold values

function Get-ThresholdColor {

param (

[double]$Value,

[double]$WarningThreshold,

[double]$CriticalThreshold

)

if ($Value -ge $CriticalThreshold) {

return "#FF4136" # Red

} elseif ($Value -ge $WarningThreshold) {

return "#FF851B" # Orange

} else {

return "#2ECC40" # Green

}

}

# Function to collect VM metrics using Azure Monitor

function Get-VMMetrics {

param (

[Microsoft.Azure.Commands.Compute.Models.PSVirtualMachine]$VM,

[string]$MetricName,

[int]$TimeRangeInHours,

[string]$Aggregation = "Average"

)

$endTime = Get-Date

$startTime = $endTime.AddHours(-$TimeRangeInHours)

try {

$metrics = Get-AzMetric -ResourceId $VM.Id -MetricName $MetricName -StartTime $startTime -EndTime $endTime -TimeGrain 01:00:00 -AggregationType $Aggregation

$averageValue = ($metrics.Data | Where-Object { $\_.Average -ne $null } | Measure-Object -Property Average -Average).Average

# Return 0 if no data found

if ($null -eq $averageValue) {

return 0

}

return $averageValue

}

catch {

Write-Log "Error collecting $MetricName metrics for VM $($VM.Name): $\_" -Level Error

return 0

}

}

# Function to get VM detailed information using Azure Guest Agent

function Get-VMGuestDetails {

param (

[Microsoft.Azure.Commands.Compute.Models.PSVirtualMachine]$VM

)

try {

# Get VM status

$vmStatus = Get-AzVM -ResourceGroupName $VM.ResourceGroupName -Name $VM.Name -Status

# Initialize result object

$vmDetails = [PSCustomObject]@{

OSType = $VM.StorageProfile.OsDisk.OsType

OSName = "Unknown"

OSVersion = "Unknown"

TotalPhysicalMemoryMB = 0

AvailableMemoryMB = 0

MemoryUsagePercent = 0

Disks = @()

HasGuestAgent = $false

IsGuestAgentResponsive = $false

}

# Check if VM has guest agent

$vmDetails.HasGuestAgent = ($vmStatus.VMAgent.Statuses.DisplayStatus -eq "Ready")

$vmDetails.IsGuestAgentResponsive = ($vmStatus.VMAgent.Statuses.DisplayStatus -eq "Ready")

# If guest agent is available, try to collect guest metrics via Log Analytics if available

if ($vmDetails.HasGuestAgent) {

# For demonstration purposes, using example disk data

# In a real scenario, you would query Log Analytics for this data or use Run Command

$vmDetails.Disks = @(

[PSCustomObject]@{

DriveLetter = "C:"

TotalSpaceGB = 127

FreeSpaceGB = 80

UsedPercent = 37

}

)

# For memory, in a real scenario, query Log Analytics or use Run Command

$vmDetails.TotalPhysicalMemoryMB = 16384 # Example value

$vmDetails.AvailableMemoryMB = 8192 # Example value

$vmDetails.MemoryUsagePercent = 50 # Example value

}

return $vmDetails

}

catch {

Write-Log "Error collecting guest details for VM $($VM.Name): $\_" -Level Error

return $null

}

}

# Function to get VM network diagnostics

function Get-VMNetworkDiagnostics {

param (

[Microsoft.Azure.Commands.Compute.Models.PSVirtualMachine]$VM

)

try {

$networkInterfaces = @()

foreach ($nicRef in $VM.NetworkProfile.NetworkInterfaces) {

$nicId = $nicRef.Id

$nic = Get-AzNetworkInterface -ResourceId $nicId

$nicStats = [PSCustomObject]@{

Name = $nic.Name

PrivateIP = $nic.IpConfigurations[0].PrivateIpAddress

PublicIP = $null

MacAddress = $nic.MacAddress

NetworkSecurityGroup = $null

IsIPForwarding = $nic.EnableIPForwarding

InboundBandwidthMbps = Get-VMMetrics -VM $VM -MetricName "Network In" -TimeRangeInHours $TimeRangeInHours -Aggregation "Average"

OutboundBandwidthMbps = Get-VMMetrics -VM $VM -MetricName "Network Out" -TimeRangeInHours $TimeRangeInHours -Aggregation "Average"

}

# Get public IP if available

if ($nic.IpConfigurations[0].PublicIpAddress) {

$publicIpId = $nic.IpConfigurations[0].PublicIpAddress.Id

$publicIp = Get-AzPublicIpAddress -ResourceId $publicIpId

$nicStats.PublicIP = $publicIp.IpAddress

}

# Get NSG if available

if ($nic.NetworkSecurityGroup) {

$nsgName = ($nic.NetworkSecurityGroup.Id -split '/')[-1]

$nicStats.NetworkSecurityGroup = $nsgName

}

$networkInterfaces += $nicStats

}

return $networkInterfaces

}

catch {

Write-Log "Error collecting network diagnostics for VM $($VM.Name): $\_" -Level Error

return @()

}

}

# Function to check VM Performance and Health

function Get-VMHealthCheck {

param (

[Microsoft.Azure.Commands.Compute.Models.PSVirtualMachine]$VM,

[string]$Environment = "Production",

[string]$Priority = "Medium"

)

Write-Log "Starting health check for VM: $($VM.Name)"

# Initialize VM health object

$vmHealth = [PSCustomObject]@{

Name = $VM.Name

ResourceGroup = $VM.ResourceGroupName

Location = $VM.Location

VMSize = $VM.HardwareProfile.VmSize

OSType = $VM.StorageProfile.OsDisk.OsType

ProvisioningState = $VM.ProvisioningState

PowerState = "Unknown"

Status = "Unknown"

Environment = $Environment

Priority = $Priority

AvailabilitySet = $(if ($VM.AvailabilitySetReference) { ($VM.AvailabilitySetReference.Id -split '/')[-1] } else { "None" })

Tags = $(if ($VM.Tags) { $VM.Tags | ConvertTo-Json -Compress } else { "None" })

GuestAgentStatus = "Unknown"

ExtensionStatus = @()

CPUUsage = 0

MemoryUsage = 0

DiskDetails = @()

NetworkInterfaces = @()

SecurityStatus = "Unknown"

RecommendedActions = @()

}

# Get VM status and power state

try {

$vmStatus = Get-AzVM -ResourceGroupName $VM.ResourceGroupName -Name $VM.Name -Status

$vmHealth.PowerState = ($vmStatus.Statuses | Where-Object { $\_.Code -match 'PowerState' }).DisplayStatus

$vmHealth.Status = ($vmStatus.Statuses | Where-Object { $\_.Code -match 'ProvisioningState' }).DisplayStatus

$vmHealth.GuestAgentStatus = ($vmStatus.VMAgent.Statuses).DisplayStatus

}

catch {

Write-Log "Error getting status for VM $($VM.Name): $\_" -Level Error

$vmHealth.Status = "Error getting status"

}

# Check VM Extensions

try {

$extensions = Get-AzVMExtension -ResourceGroupName $VM.ResourceGroupName -VMName $VM.Name -ErrorAction SilentlyContinue

if ($extensions) {

foreach ($extension in $extensions) {

$vmHealth.ExtensionStatus += [PSCustomObject]@{

Name = $extension.Name

Type = $extension.ExtensionType

Publisher = $extension.Publisher

ProvisioningState = $extension.ProvisioningState

StatusMessage = $extension.StatusMessage

}

}

}

}

catch {

Write-Log "Error checking extensions for VM $($VM.Name): $\_" -Level Error

}

# Get CPU metrics

$vmHealth.CPUUsage = Get-VMMetrics -VM $VM -MetricName "Percentage CPU" -TimeRangeInHours $TimeRangeInHours

# Get Guest Details including memory and disk

$guestDetails = Get-VMGuestDetails -VM $VM

if ($guestDetails) {

$vmHealth.MemoryUsage = $guestDetails.MemoryUsagePercent

$vmHealth.DiskDetails = $guestDetails.Disks

}

# Otherwise, use basic disk information from Azure

if (-not $guestDetails -or $guestDetails.Disks.Count -eq 0) {

try {

# Get OS disk

$osDisk = Get-AzDisk -ResourceGroupName $VM.ResourceGroupName -DiskName $VM.StorageProfile.OsDisk.Name

$vmHealth.DiskDetails += [PSCustomObject]@{

DriveLetter = "OS Disk"

Name = $osDisk.Name

Type = $osDisk.Sku.Name

SizeGB = [math]::Round($osDisk.DiskSizeGB, 2)

UsedPercent = 50 # Placeholder since we can't get actual usage from Azure

}

# Get data disks

foreach ($dataDisk in $VM.StorageProfile.DataDisks) {

$disk = Get-AzDisk -ResourceGroupName $VM.ResourceGroupName -DiskName $dataDisk.Name

$vmHealth.DiskDetails += [PSCustomObject]@{

DriveLetter = "Data Disk"

Name = $disk.Name

Type = $disk.Sku.Name

SizeGB = [math]::Round($disk.DiskSizeGB, 2)

UsedPercent = 50 # Placeholder since we can't get actual usage from Azure

}

}

}

catch {

Write-Log "Error getting disk details for VM $($VM.Name): $\_" -Level Error

}

}

# Get Network Interfaces

$vmHealth.NetworkInterfaces = Get-VMNetworkDiagnostics -VM $VM

# Check Security Status

try {

# Check if Azure Security Center is available

$securityStatus = Get-AzSecurityTask -ResourceId $VM.Id -ErrorAction SilentlyContinue

if ($securityStatus) {

$vmHealth.SecurityStatus = "Security tasks: $($securityStatus.Count)"

}

else {

$vmHealth.SecurityStatus = "No security tasks found"

}

}

catch {

Write-Log "Error checking security status for VM $($VM.Name): $\_" -Level Warning

$vmHealth.SecurityStatus = "Unable to check security status"

}

# Generate recommendations based on findings

if ($vmHealth.CPUUsage -ge $CpuThresholdCritical) {

$vmHealth.RecommendedActions += "CRITICAL: CPU usage is very high ($($vmHealth.CPUUsage)%). Consider scaling up the VM or investigating high CPU processes."

}

elseif ($vmHealth.CPUUsage -ge $CpuThresholdWarning) {

$vmHealth.RecommendedActions += "WARNING: CPU usage is high ($($vmHealth.CPUUsage)%). Monitor closely and consider scaling if this persists."

}

if ($vmHealth.MemoryUsage -ge $MemoryThresholdCritical) {

$vmHealth.RecommendedActions += "CRITICAL: Memory usage is very high ($($vmHealth.MemoryUsage)%). Consider adding more memory or investigating memory leaks."

}

elseif ($vmHealth.MemoryUsage -ge $MemoryThresholdWarning) {

$vmHealth.RecommendedActions += "WARNING: Memory usage is high ($($vmHealth.MemoryUsage)%). Monitor for potential memory constraints."

}

foreach ($disk in $vmHealth.DiskDetails) {

if ($disk.UsedPercent -ge $DiskThresholdCritical) {

$vmHealth.RecommendedActions += "CRITICAL: Disk $($disk.Name) is nearly full ($($disk.UsedPercent)% used). Increase disk size or clean up files."

}

elseif ($disk.UsedPercent -ge $DiskThresholdWarning) {

$vmHealth.RecommendedActions += "WARNING: Disk $($disk.Name) is filling up ($($disk.UsedPercent)% used). Plan for cleanup or expansion."

}

}

if ($vmHealth.PowerState -ne "VM running") {

$vmHealth.RecommendedActions += "WARNING: VM is not running. Current state: $($vmHealth.PowerState)"

}

if ($vmHealth.GuestAgentStatus -ne "Ready") {

$vmHealth.RecommendedActions += "WARNING: Guest agent is not ready. Status: $($vmHealth.GuestAgentStatus)"

}

foreach ($extension in $vmHealth.ExtensionStatus) {

if ($extension.ProvisioningState -ne "Succeeded") {

$vmHealth.RecommendedActions += "WARNING: Extension $($extension.Name) has issues. State: $($extension.ProvisioningState)"

}

}

Write-Log "Completed health check for VM: $($VM.Name)"

return $vmHealth

}

# Function to generate HTML report

function Generate-HTMLReport {

param (

[array]$VMHealthResults,

[string]$OutputPath

)

$reportDate = Get-Date -Format "yyyy-MM-dd HH:mm:ss"

$reportFile = Join-Path -Path $OutputPath -ChildPath "AzureVMHealthReport\_$(Get-Date -Format 'yyyyMMdd\_HHmmss').html"

# Create the HTML header with CSS styles

$htmlHeader = @"

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Azure VM Health Check Report - $reportDate</title>

<style>

body {

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

margin: 0;

padding: 20px;

color: #333;

background-color: #f9f9f9;

}

.container {

max-width: 1200px;

margin: 0 auto;

background-color: #fff;

padding: 20px;

box-shadow: 0 2px 10px rgba(0,0,0,0.1);

border-radius: 5px;

}

h1, h2, h3 {

color: #0078d4;

}

h1 {

border-bottom: 2px solid #0078d4;

padding-bottom: 10px;

}

.header-info {

display: flex;

justify-content: space-between;

color: #666;

margin-bottom: 20px;

}

.summary {

background-color: #f0f8ff;

padding: 15px;

border-left: 5px solid #0078d4;

margin-bottom: 20px;

}

table {

width: 100%;

border-collapse: collapse;

margin-bottom: 20px;

}

th {

background-color: #0078d4;

color: white;

text-align: left;

padding: 10px;

}

td {

padding: 8px 10px;

border-bottom: 1px solid #ddd;

}

tr:nth-child(even) {

background-color: #f2f2f2;

}

.vm-card {

border: 1px solid #ddd;

border-radius: 5px;

margin-bottom: 20px;

overflow: hidden;

}

.vm-header {

padding: 15px;

background-color: #f5f5f5;

border-bottom: 1px solid #ddd;

display: flex;

justify-content: space-between;

align-items: center;

}

.vm-name {

font-size: 18px;

font-weight: bold;

color: #0078d4;

margin: 0;

}

.vm-status {

font-weight: bold;

padding: 5px 10px;

border-radius: 3px;

color: white;

}

.vm-body {

padding: 15px;

}

.metric-tile {

display: inline-block;

width: 150px;

margin: 10px;

padding: 15px;

text-align: center;

border-radius: 5px;

background-color: #f5f5f5;

box-shadow: 0 2px 5px rgba(0,0,0,0.1);

}

.metric-value {

font-size: 24px;

font-weight: bold;

margin: 5px 0;

}

.metric-label {

font-size: 14px;

color: #666;

}

.section {

margin-bottom: 20px;

}

.section-title {

font-size: 16px;

font-weight: bold;

margin-bottom: 10px;

padding-bottom: 5px;

border-bottom: 1px solid #ddd;

color: #0078d4;

}

.critical {

background-color: #FF4136;

}

.warning {

background-color: #FF851B;

}

.healthy {

background-color: #2ECC40;

}

.recommendation {

border-left: 4px solid #ff9800;

padding: 10px;

margin: 10px 0;

background-color: #fff3e0;

}

.critical-rec {

border-left-color: #FF4136;

background-color: #ffebee;

}

.warning-rec {

border-left-color: #FF851B;

background-color: #fff3e0;

}

.info-rec {

border-left-color: #0078d4;

background-color: #e3f2fd;

}

.metric-chart {

width: 100%;

height: 10px;

background-color: #eee;

border-radius: 5px;

margin-top: 5px;

}

.metric-fill {

height: 100%;

border-radius: 5px;

}

.progress-bar {

background-color: #eee;

height: 10px;

border-radius: 10px;

margin-top: 5px;

}

.progress-fill {

height: 100%;

border-radius: 10px;

}

.footer {

text-align: center;

margin-top: 30px;

color: #666;

}

</style>

</head>

<body>

<div class="container">

<h1>Azure VM Health Check Report</h1>

<div class="header-info">

<span>Generated: $reportDate</span>

<span>Environment: Azure</span>

</div>

"@

# Create the summary section

$healthyVMs = ($VMHealthResults | Where-Object { $\_.RecommendedActions.Count -eq 0 }).Count

$warningVMs = ($VMHealthResults | Where-Object { $\_.RecommendedActions | Where-Object { $\_ -match "WARNING" } }).Count

$criticalVMs = ($VMHealthResults | Where-Object { $\_.RecommendedActions | Where-Object { $\_ -match "CRITICAL" } }).Count

$htmlSummary = @"

<div class="summary">

<h2>Executive Summary</h2>

<p>This report provides a comprehensive health check of your Azure Virtual Machines.</p>

<table>

<tr>

<td><strong>Total VMs Analyzed:</strong></td>

<td>$($VMHealthResults.Count)</td>

</tr>

<tr>

<td><strong>Healthy VMs:</strong></td>

<td>$healthyVMs</td>

</tr>

<tr>

<td><strong>VMs with Warnings:</strong></td>

<td>$warningVMs</td>

</tr>

<tr>

<td><strong>VMs with Critical Issues:</strong></td>

<td>$criticalVMs</td>

</tr>

<tr>

<td><strong>Report Date:</strong></td>

<td>$reportDate</td>

</tr>

</table>

</div>

"@

# Create detailed VM sections

$vmDetailsSections = ""

foreach ($vmHealth in $VMHealthResults) {

# Determine VM status color

$statusClass = "healthy"

if ($vmHealth.RecommendedActions | Where-Object { $\_ -match "CRITICAL" }) {

$statusClass = "critical"

}

elseif ($vmHealth.RecommendedActions | Where-Object { $\_ -match "WARNING" }) {

$statusClass = "warning"

}

# Create CPU color based on threshold

$cpuColor = Get-ThresholdColor -Value $vmHealth.CPUUsage -WarningThreshold $CpuThresholdWarning -CriticalThreshold $CpuThresholdCritical

$memoryColor = Get-ThresholdColor -Value $vmHealth.MemoryUsage -WarningThreshold $MemoryThresholdWarning -CriticalThreshold $MemoryThresholdCritical

# Create VM section

$vmDetailsSections += @"

<div class="vm-card">

<div class="vm-header">

<h3 class="vm-name">$($vmHealth.Name)</h3>

<span class="vm-status $statusClass">$($vmHealth.PowerState)</span>

</div>

<div class="vm-body">

<div class="section">

<div class="section-title">Basic Information</div>

<table>

<tr>

<td><strong>Resource Group:</strong></td>

<td>$($vmHealth.ResourceGroup)</td>

<td><strong>Location:</strong></td>

<td>$($vmHealth.Location)</td>

</tr>

<tr>

<td><strong>VM Size:</strong></td>

<td>$($vmHealth.VMSize)</td>

<td><strong>OS Type:</strong></td>

<td>$($vmHealth.OSType)</td>

</tr>

<tr>

<td><strong>Environment:</strong></td>

<td>$($vmHealth.Environment)</td>

<td><strong>Priority:</strong></td>

<td>$($vmHealth.Priority)</td>

</tr>

<tr>

<td><strong>Provisioning State:</strong></td>

<td>$($vmHealth.ProvisioningState)</td>

<td><strong>Availability Set:</strong></td>

<td>$($vmHealth.AvailabilitySet)</td>

</tr>

<tr>

<td><strong>Guest Agent Status:</strong></td>

<td>$($vmHealth.GuestAgentStatus)</td>

<td><strong>Security Status:</strong></td>

<td>$($vmHealth.SecurityStatus)</td>

</tr>

</table>

</div>

<div class="section">

<div class="section-title">Performance Metrics</div>

<div class="metric-tile">

<div class="metric-label">CPU Usage</div>

<div class="metric-value" style="color: $cpuColor">$([Math]::Round($vmHealth.CPUUsage, 1))%</div>

<div class="progress-bar">

<div class="progress-fill" style="width: $($vmHealth.CPUUsage)%; background-color: $cpuColor"></div>

</div>

</div>

<div class="metric-tile">

<div class="metric-label">Memory Usage</div>

<div class="metric-value" style="color: $memoryColor">$([Math]::Round($vmHealth.MemoryUsage, 1))%</div>

<div class="progress-bar">

<div class="progress-fill" style="width: $($vmHealth.MemoryUsage)%; background-color: $memoryColor"></div>

</div>

</div>

"@

# Add network metrics if available

if ($vmHealth.NetworkInterfaces.Count -gt 0) {

$vmDetailsSections += @"

<div class="metric-tile">

<div class="metric-label">Network In</div>

<div class="metric-value">$([Math]::Round($vmHealth.NetworkInterfaces[0].InboundBandwidthMbps, 2)) MB/s</div>

</div>

<div class="metric-tile">

<div class="metric-label">Network Out</div>

<div class="metric-value">$([Math]::Round($vmHealth.NetworkInterfaces[0].OutboundBandwidthMbps, 2)) MB/s</div>

</div>

"@

}

$vmDetailsSections += @"

</div>

<div class="section">

<div class="section-title">Disk Usage</div>

<table>

<tr>

<th>Drive</th>

<th>Name</th>

<th>Size</th>

<th>Usage</th>

</tr>

"@

foreach ($disk in $vmHealth.DiskDetails) {

$diskColor = Get-ThresholdColor -Value $disk.UsedPercent -WarningThreshold $DiskThresholdWarning -CriticalThreshold $DiskThresholdCritical

$vmDetailsSections += @"

<tr>

<td>$($disk.DriveLetter)</td>

<td>$($disk.Name)</td>

<td>$($disk.SizeGB) GB</td>

<td>

<div style="display: flex; align-items: center;">

<span style="margin-right: 10px; color: $diskColor">$($disk.UsedPercent)%</span>

<div class="progress-bar" style="flex-grow: 1;">

<div class="progress-fill" style="width: $($disk.UsedPercent)%; background-color: $diskColor"></div>

</div>

</div>

</td>

</tr>

"@

}

$vmDetailsSections += @"

</table>

</div>

<div class="section">

<div class="section-title">Network Interfaces</div>

<table>

<tr>

<th>Name</th>

<th>Private IP</th>

<th>Public IP</th>

<th>MAC Address</th>

<th>Network Security Group</th>

</tr>

"@

foreach ($nic in $vmHealth.NetworkInterfaces) {

$vmDetailsSections += @"

<tr>

<td>$($nic.Name)</td>

<td>$($nic.PrivateIP)</td>

<td>$($nic.PublicIP)</td>

<td>$($nic.MacAddress)</td>

<td>$($nic.NetworkSecurityGroup)</td>

</tr>

"@

}

$vmDetailsSections += @"

</table>

</div>

"@

# Add recommendations if any

if ($vmHealth.RecommendedActions.Count -gt 0) {

$vmDetailsSections += @"

<div class="section">

<div class="section-title">Recommendations</div>

"@

foreach ($action in $vmHealth.RecommendedActions) {

$recClass = "info-rec"

if ($action -match "CRITICAL") {

$recClass = "critical-rec"

$action = $action -replace "CRITICAL: ", ""

}

elseif ($action -match "WARNING") {

$recClass = "warning-rec"

$action = $action -replace "WARNING: ", ""

}

$vmDetailsSections += @"

<div class="recommendation $recClass">$action</div>

"@

}

$vmDetailsSections += @"

</div>

"@

}

$vmDetailsSections += @"

</div>

</div>

"@

}

# Create HTML footer

$htmlFooter = @"

<div class="footer">

<p>Generated by Azure VM Health Check Script | Report Date: $reportDate</p>

</div>

</div>

</body>

</html>

"@

# Combine all HTML sections

$htmlReport = $htmlHeader + $htmlSummary + $vmDetailsSections + $htmlFooter

# Create output file

$htmlReport | Out-File -FilePath $reportFile -Force

return $reportFile

}

# Function to send email with the report

function Send-ReportEmail {

param (

[string]$ReportPath,

[string]$EmailTo,

[PSCredential]$Credential,

[string]$SMTPServer,

[int]$SMTPPort

)

try {

$subject = "Azure VM Health Check Report - $(Get-Date -Format 'yyyy-MM-dd')"

$body = @"

<html>

<body>

<h2>Azure VM Health Check Report</h2>

<p>Please find attached the latest Azure VM Health Check Report generated on $(Get-Date -Format 'yyyy-MM-dd HH:mm:ss').</p>

<p>This report provides a comprehensive health assessment of your Azure Virtual Machines, including CPU, memory, disk, and network metrics.</p>

<p>Please review any warnings or critical issues highlighted in the report.</p>

<p><strong>Note:</strong> This is an automated email. Please do not reply.</p>

</body>

</html>

"@

Send-MailMessage -From $Credential.UserName `

-To $EmailTo.Split(';') `

-Subject $subject `

-Body $body `

-BodyAsHtml `

-SmtpServer $SMTPServer `

-Port $SMTPPort `

-UseSsl `

-Credential $Credential `

-Attachments $ReportPath

Write-Log "Email sent successfully to $EmailTo"

return $true

}

catch {

Write-Log "Error sending email: $\_" -Level Error

return $false

}

}

# Function to post summary to Teams

function Send-TeamsNotification {

param (

[string]$WebhookUrl,

[PSCustomObject]$Summary,

[string]$ReportUrl

)

try {

$reportDate = Get-Date -Format "yyyy-MM-dd HH:mm:ss"

$color = "0076D7" # Default blue

if ($Summary.CriticalVMs -gt 0) {

$color = "FF0000" # Red

}

elseif ($Summary.WarningVMs -gt 0) {

$color = "FFA500" # Orange

}

$teamsMessage = @{

"@type" = "MessageCard"

"@context" = "https://schema.org/extensions"

"summary" = "Azure VM Health Check Report"

"themeColor" = $color

"title" = "Azure VM Health Check Report - $reportDate"

"sections" = @(

@{

"facts" = @(

@{

"name" = "Total VMs Analyzed:"

"value" = $Summary.TotalVMs

},

@{

"name" = "Healthy VMs:"

"value" = $Summary.HealthyVMs

},

@{

"name" = "VMs with Warnings:"

"value" = $Summary.WarningVMs

},

@{

"name" = "VMs with Critical Issues:"

"value" = $Summary.CriticalVMs

},

@{

"name" = "Report Generated:"

"value" = $reportDate

}

)

"text" = "Health check completed. Please review the detailed report for more information."

}

)

"potentialAction" = @(

@{

"@type" = "OpenUri"

"name" = "View Full Report"

"targets" = @(

@{

"os" = "default"

"uri" = $ReportUrl

}

)

}

)

}

$jsonBody = ConvertTo-Json -InputObject $teamsMessage -Depth 4

Invoke-RestMethod -Uri $WebhookUrl -Method Post -Body $jsonBody -ContentType 'application/json'

Write-Log "Teams notification sent successfully"

return $true

}

catch {

Write-Log "Error sending Teams notification: $\_" -Level Error

return $false

}

}

# Process all VMs from Excel

$vmHealthResults = @()

$processedCount = 0

$totalVMs = $vmList.Count

# Track subscription changes

$currentSubscriptionId = ""

foreach ($vmInfo in $vmList) {

$processedCount++

Write-Log "Processing VM $processedCount of $totalVMs: $($vmInfo.VMName) in resource group $($vmInfo.ResourceGroupName)"

try {

# Switch subscription if needed

if ($currentSubscriptionId -ne $vmInfo.SubscriptionId) {

Write-Log "Switching to subscription: $($vmInfo.SubscriptionId)"

Select-AzSubscription -SubscriptionId $vmInfo.SubscriptionId

$currentSubscriptionId = $vmInfo.SubscriptionId

}

# Get VM

$vm = Get-AzVM -ResourceGroupName $vmInfo.ResourceGroupName -Name $vmInfo.VMName -ErrorAction Stop

# Get environment and priority info from Excel

$environment = if ($vmInfo.Environment) { $vmInfo.Environment } else { "Production" }

$priority = if ($vmInfo.Priority) { $vmInfo.Priority } else { "Medium" }

# Get health data

$vmHealth = Get-VMHealthCheck -VM $vm -Environment $environment -Priority $priority

$vmHealthResults += $vmHealth

}

catch {

Write-Log "Error processing VM $($vmInfo.VMName): $\_" -Level Error

# Add to results as errored VM

$vmHealthResults += [PSCustomObject]@{

Name = $vmInfo.VMName

ResourceGroup = $vmInfo.ResourceGroupName

PowerState = "Error"

Status = "Failed to retrieve"

Environment = if ($vmInfo.Environment) { $vmInfo.Environment } else { "Production" }

Priority = if ($vmInfo.Priority) { $vmInfo.Priority } else { "Medium" }

RecommendedActions = @("ERROR: Failed to retrieve VM details: $\_")

}

}

}

# Generate report

if ($vmHealthResults.Count -gt 0) {

Write-Log "Generating HTML report for $($vmHealthResults.Count) VMs"

$reportPath = Generate-HTMLReport -VMHealthResults $vmHealthResults -OutputPath $tempFolder

Write-Log "Report generated: $reportPath"

# Upload report to blob storage

$reportFileName = Split-Path -Path $reportPath -Leaf

$blobName = "reports/$reportFileName"

Write-Log "Uploading report to blob storage: $blobName"

$blobUpload = Set-AzStorageBlobContent -File $reportPath -Container "health-reports" -Blob $blobName -Context $storageContext -Force

$reportUrl = "$($blobUpload.ICloudBlob.Uri.AbsoluteUri)"

Write-Log "Report uploaded to: $reportUrl"

# Prepare summary for Teams

$reportSummary = [PSCustomObject]@{

TotalVMs = $vmHealthResults.Count

HealthyVMs = ($vmHealthResults | Where-Object { $\_.RecommendedActions.Count -eq 0 }).Count

WarningVMs = ($vmHealthResults | Where-Object { $\_.RecommendedActions | Where-Object { $\_ -match "WARNING" } }).Count

CriticalVMs = ($vmHealthResults | Where-Object { $\_.RecommendedActions | Where-Object { $\_ -match "CRITICAL" } }).Count

}

# Send email if configured

if ($SendEmail) {

Write-Log "Sending email to recipients"

# Send to client

Send-ReportEmail -ReportPath $reportPath -EmailTo $clientEmailAddress -Credential $emailCredential -SMTPServer $smtpServer -SMTPPort $smtpPort

# Send to team

Send-ReportEmail -ReportPath $reportPath -EmailTo $teamEmailAddress -Credential $emailCredential -SMTPServer $smtpServer -SMTPPort $smtpPort

}

# Post to Teams if configured

if ($PostToTeams) {

Write-Log "Posting summary to Teams"

Send-TeamsNotification -WebhookUrl $teamsWebhookUrl -Summary $reportSummary -ReportUrl $reportUrl

}

}

else {

Write-Log "No VM results to report" -Level Warning

}

# Clean up temp folder

Remove-Item -Path $tempFolder -Recurse -Force -ErrorAction SilentlyContinue

Write-Log "Script execution completed successfully"