Centralize log collection with a custom REST API

All environments of a larger IT team write several thousand logs and status messages every day. These logs are usually only stored on the respective systems and cannot be centrally administered or managed. Alarms or monitoring on a log basis in a centralised solution is also not available.

This solution is intended to remedy these problems. On the one hand, a complex topic should be broken down and standardised in a simple solution. The installation of the solution should be as simple as possible and the evaluation of the collected data should be structured in an easily understandable dashboard. Standard components should be used whenever possible, based on standard configurations.

Use case

This API is ideal to create a middleware between the logging client and the logging storage. This brings advantages such as having full control over the secrets, access codes and to take the complexity out of the logging solution on the client. This means Azure Runbooks or Intune Remediation Scripts send the corresponding logs directly to a central log storage pot. This storage pot can then be evaluated and the logs stored and evaluated accordingly. There is also the possibility that monitoring can be built on the logs, as all logs can be stored centrally and for a longer period of time.

In an automated process, a log entry should be created for each important event. The log should be transmitted directly at this point in time, as otherwise the status of the process cannot be clearly traced in the event of a next potentially faulty step.

Function code

This code will run in the Azure Function. It receives two input types via an REST API Body. The LogType sets the table, in which the data will be saved to. The second object is dynamic and can contain as much elements as the creator needs.

Important: To get this to work you have to get the customer id and the shared key from the appropriate log analytics workspace.

```
using namespace System.Net

# Input bindings are passed in via param block.
param($Request, $TriggerMetadata)

$logtype = $Request.Body.logtype
$logbody = $Request.Body.logbody

$customerId = ""
$sharedKey = ""

Function Build-Signature ($customerId, $sharedKey, $date, $contentLength, $method, $contentType,
$resource){
    $xHeaders = "x-ms-date:" + $date
    $stringToHash = $method + "`n" + $contentLength + "`n" + $contentType + "`n" + $xHeaders + "`n" +
$resource

    $bytesToHash = [Text.Encoding]::UTF8.GetBytes($stringToHash)
    $keyBytes = [Convert]::FromBase64String($sharedKey)

    $sha256 = New-Object System.Security.Cryptography.HMACSHA256
    $sha256.Key = $keyBytes
    $calculatedHash = $sha256.ComputeHash($bytesToHash)
    $encodedHash = [Convert]::ToBase64String($calculatedHash)
    $authorization = 'SharedKey {0}:{1}' -f $customerId,$encodedHash
    return $authorization
}

Function Post-LogAnalyticsData ($customerId, $sharedKey, $body, $logType){
    $method = "POST"
    $contentType = "application/json"
    $resource = "/api/logs"
    $rfc1123date = ([DateTime]::UtcNow).ToString("r")
    $contentLength = $body.Length
    $signature = Build-Signature -customerId $customerId -sharedKey $sharedKey -date $rfc1123date -
contentLength $contentLength -method $method -contentType $contentType -resource $resource

    $uri = "https://" + $customerId + ".ods.opinsights.azure.com" + $resource + "?api-version=2016-04-01"

    $headers = @{
```

```
        "Authorization" = $signature;
        "Log-Type" = $logType;
        "x-ms-date" = $rfc1123date;
    }

    $response = Invoke-WebRequest -Uri $uri -Method $method -ContentType $contentType -Headers $headers -
Body $body -UseBasicParsing
    return $response.StatusCode
}

$logbodyjson = ConvertTo-JSON $logbody -Depth 10

#Submit the data to the API endpoint
$params = @{
    CustomerId = $customerId
    SharedKey  = $sharedKey
    Body       = ([System.Text.Encoding]::UTF8.GetBytes($logbodyjson))
    LogType    = $LogType
}
$LogResponse = Post-LogAnalyticsData @params
$LogResponse

if($LogResponse -eq 200){
    Push-OutputBinding -Name Response -Value ([HttpResponseContext]@{
        StatusCode = [HttpStatusCode]::OK
        Body = $LogResponse
    })
}
```

This Azure Function returns the Code 200 if the POST Request and the storage in the Azure Log Analytics were successful.

Send logs via PowerShell script

On the client side you can use this PowerShell function to send the logs to the previously created Azure Function. First the Function URL must be inserted into the variable "url". After that you can initialize the function at the very beginning of your PowerShell script. Afterwards you can set the values, that need to be logged, into the Hash Table "Logs". This Hash Table can then be sent via the Function Send-Logs() to the Azure Function API which then stores the code in the Azure Log Analytics Workspace. That it knows which table it should write the data to, you have to specify the

LogType on the Send-Logs() Function.

```
Function Send-Logs(){

    param (
        [String]$LogType,
        [Hashtable]$LogBodyList
    )
    $url="<functionurl>"

    $LogBodyJSON = @"
    {
        "logtype": "$LogType",
        "logbody": {}
    }
"@

    $LogBodyObject = ConvertFrom-JSON $LogBodyJSON
    Foreach($Log in $LogBodyList.GetEnumerator()){
        $LogBodyObject.logbody | Add-Member -MemberType NoteProperty -Name $log.key -Value $log.value
    }
    $Body = ConvertTo-JSON $LogBodyObject

    $Response = Invoke-Restmethod -uri $url -Body $Body -Method POST -ContentType "application/json"
    return $Response
}

$Logs = @{
    "<logmember1>"="<logvalue1>"
    "<logmember2>"="<logvalue2>"
}

Send-Logs -LogType "<logtype>" -LogBodyList $Logs
```

This Function then returns the HTTP status code from the LogCollection API.

Revision #12
Created 6 December 2022 22:34:06
Updated 21 July 2024 15:11:16