

Run PowerShell Code from frontend on backend using Azure Function

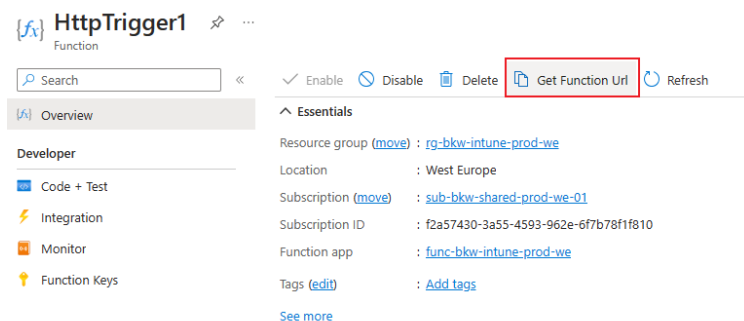
This tutorial shows how to build a REST API that executes PowerShell code using an Azure Function. This code can then return values and objects as JSON responses using "return".

Using Function

To use this function you need the following:

- Azure Function with server-side code (manual below).
- Entra ID App Registration credentials with the permissions you need: [Get app details and gr... | LNC DOCS \(lucanoahcaprez.ch\)](#)
- PowerShell code you want to run on the server (example below).

To execute the code on the server, you need to send a JSON to the function URL via POST request. It is important that the POST call goes to the correct URL. You can get this URL on the function level via this button:



The URL is structured as follows:

```
https://<yourfunctionappname>.azurewebsites.net/api/<yourfunctionname>?code=<yourfunctionkey>
```

The JSON which is needed consists of the main key "parameters", which then must have the following four keys. The keys starting with "microsoft" are needed for the authentication. The "powerShellCode"-key contains the final code which will be executed.

It is also useful to know that all other keys, which are created below "parameters", can be used as variables in the PowerShell script. As an example: If a new key named "UserLanguage" is added, the \$UserLanguage variable can be used in PowerShell.

```
{
  "parameters":{
    "microsoftTenantID":"<yourtenantname>",
    "microsoftClientID":"<yourappregistrationclientid>",
    "microsoftClientSecret":"<yourappregistrationclientsecret>",
    "powerShellCode":"<yourpowershellcode>"
  }
}
```

Example Request

Here is a sample request that allows to fetch all user data from Microsoft Entra ID via Graph API.

```
POST https://<yourfunctionappname>.azurewebsites.net/api/<yourfunctionname>?code=<yourfunctionkey>

{
  "parameters":{
    "microsoftTenantID":"<yourtenantname>",
    "microsoftClientID":"<yourappregistrationclientid>",
    "microsoftClientSecret":"<yourappregistrationclientsecret>",
    "powerShellCode": "$Uri = \"https://graph.microsoft.com/v1.0/users\"
\n$Result = Invoke-RestMethod -Uri $Uri -Body $requestBody -Headers @{Authorization = $Global:AzureADAccessToken; ConsistencyLevel = 'eventual' } -Method GET -ContentType 'application/json'
\n$Members = $Result.value
\nwhile ($Result.'@odata.nextLink') {
\n  $Result = Invoke-RestMethod -Uri $Result.'@odata.nextLink' -Headers $Header
\n  $Members += $Result.value
\n}
\nreturn $Members"
  }
}
```

Setup Function

For this Azure Function you will need an Azure Function App. It must be able to execute PowerShell code. In the Function App you have to create a Function with the HTTP trigger and FUNCTION Authorization level.

Create function

Select development environment

Instructions will vary based on your development environment. [Learn more](#)

Development environ...

Develop in portal

▼

Select a template

Use a template to create a function. Triggers describe the type of events that invoke your functions. [Learn more](#)

Filter

Template	Description
HTTP trigger	A function that will be run whenever it receives an HTTP request, responding based on data in the body or query string
Timer trigger	A function that will be run on a specified schedule
Azure Queue Storage trigger	A function that will be run whenever a message is added to a specified Azure Storage queue
Azure Service Bus Queue trigger	A function that will be run whenever a message is added to a specified Service Bus queue
Azure Service Bus Topic trigger	A function that will be run whenever a message is added to the specified Service Bus topic
Azure Blob Storage trigger	A function that will be run whenever a blob is added to a specified container
Azure Event Hub trigger	A function that will be run whenever an event hub receives a new event

Template details

We need more information to create the HTTP trigger function. [Learn more](#)

New Function *

FUNC-APP-PS1-MicrosoftGraphAPI-P...

Authorization level * ⓘ

FUNCTION

Create

Cancel

This function then acts as the container where the server-side code runs. The Azure Function framework then accepts the REST calls and does the load balancing and handling of the HTTP requests.

Server-side PowerShell Code

This code is used on the Function to handle the requests. It is important to know that this is specifically for handling the incoming JSON and responses.

```
using namespace System.Net

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

# Function for returning Values to Request
function Return-FunctionValue{
    param(
        [boolean] $Error,
        [string] $StatusCodeString,
```

```

    $OutputBody
)
switch -exact ($StatusCodeString.ToUpper()) {
    "OK" {
        $StatusCode = [System.Net.HttpStatusCode]::OK
    }
    "NOTFOUND" {
        $StatusCode = [System.Net.HttpStatusCode]::NotFound
    }
    "NOCONTENT" {
        $StatusCode = [System.Net.HttpStatusCode]::NoContent
    }
    "BADREQUEST" {
        $StatusCode = [System.Net.HttpStatusCode]::BadRequest
    }
    "INTERNALSERVERERROR" {
        $StatusCode = [System.Net.HttpStatusCode]::InternalServerError
    }
    default {
        $StatusCode = $null
    }
}
if($Error){
    $OutputBody = @{
        "statusCode" = $StatusCode
        "errorMessage" = $OutputBody
    }
}
Write-Output $StatusCode
Write-Output $OutputBody
Push-OutputBinding -Name Response -Value ([HttpResponseContext]@{
    StatusCode = $StatusCode
    Body = $OutputBody
})
Exit
}

try{
    $BodyParameters = [PSCustomObject]$Request.Body.parameters
    Foreach($BodyParameter in $Bodyparameters.PSObject.Properties) {

```

```

        New-Variable -Name $BodyParameter.Name -Value $BodyParameter.Value
    }
}
catch{
    Return-FunctionValue -StatusCodeString "InternalServerError" -OutputBody "Parameters provided could not
be converted to variable" -Error $True
}

$powerShellCode = $powerShellCode.Replace("`n", "`n")

# Function for getting Azure AD Access Header
Function Build-MicrosoftEntraIDApplicationAccessHeader(){
    param(
        [Parameter(Mandatory=$true)]
        [string] $TenantID,
        [string] $ClientID,
        [string] $ClientSecret,
        [string] $refreshToken
    )

    $authenticationurl = "https://login.microsoftonline.com/$TenantID/oauth2/v2.0/token"

    if($refreshToken -and $TenantID){
        $tokenBodySource = @{
            grant_type = "refresh_token"
            scope = "https://graph.microsoft.com/.default"
            refresh_token = $refreshToken
        }
    }
    elseif($TenantID -and $ClientID -and $ClientSecret){
        $tokenBodySource = @{
            grant_type = "client_credentials"
            scope = "https://graph.microsoft.com/.default"
            client_id = $ClientID
            client_secret = "$ClientSecret"
        }
    }
    else{
        Return-FunctionValue -StatusCodeString "BadRequest" -OutputBody "Authentication failed: Not all
parameters provided for authentication" -Error $True
    }
}

```

```

}

while ([string]::IsNullOrEmpty($AuthResponse.access_token)) {
    $AuthResponse = try {
        Invoke-RestMethod -Method POST -Uri $authenticationurl -Body $tokenBodySource
    }
    catch {
        $ErrorAuthResponse = $_.ErrorDetails.Message | ConvertFrom-Json
        if ($ErrorAuthResponse.error -ne "authorization_pending") {
            Write-Output "error"
            Return-FunctionValue -StatusCodeString "BadRequest" -OutputBody "Authentication failed: Error while
posting body source: $($ErrorAuthResponse.error)" -Error $True
        }
    }
}

if($AuthResponse.token_type -and $AuthResponse.access_token){
    $global:MicrosoftEntraIDAccessToken = "$($AuthResponse.token_type) $($AuthResponse.access_token)"
    $global:Header = @{
        "Authorization" = "$global:MicrosoftEntraIDAccessToken"
    }
    Write-Output "Authorization successful! Token saved in variable."
}
else{
    Return-FunctionValue -StatusCodeString "BadRequest" -OutputBody "Authentication failed: Not all
parameters provided for authentication" -Error $True
}
}

# Authorization Header with ClientId & ClientSecret
try{
    if(($microsoftTenantID) -and ($microsoftClientID) -and ($microsoftClientSecret)){
        Build-MicrosoftEntraIDApplicationAccessHeader -tenantid $microsoftTenantID -clientid $microsoftClientID -
clientSecret $microsoftClientSecret
    }
    else{
        Return-FunctionValue -StatusCodeString "BadRequest" -OutputBody "Authentication failed: Not all
parameters provided for authentication" -Error $True
    }
}

```

```
} catch{
    Return-FunctionValue -StatusCodeString "InternalServerError" -OutputBody "Bearer token could not be
created with provided parameters" -Error $True
}

# Run PowerShell Code
try{
    if($PowerShellCode){
        $OutputBody = Invoke-Expression $PowerShellCode
        Return-FunctionValue -StatusCodeString "OK" -OutputBody $OutputBody -Error $False
    }
    else{
        Return-FunctionValue -StatusCodeString "BadRequest" -OutputBody "PowerShell Code failed: No PowerShell
Code provided for runtime" -Error $True
    }
}
catch{
    Return-FunctionValue -StatusCodeString "InternalServerError" -OutputBody "PowerShell Code provided did not
run correctly" -Error $True
}

# Return-FunctionValue -StatusCodeString "NOCONTENT" -OutputBody ""
```

Revision #2

Created 7 September 2023 11:16:45

Updated 21 July 2024 15:11:16