TICKET REQUEST ADDIN SET UP

DEPENDENCY CONFIGURATION: SHAREPOINT ONLINE AND AZURE FUNCTION

Publisher: AVALORA

Document: AVALORA-TicketRequestManual

Date: 20/11/2025

Offices

Av. de Manoteras, 20 Edificio Nueva York - Tercera Planta - 28050 Madrid SPAIN

Phone & Fax

Tel.:(+34) 913 484 848
Fax: (+34) 913 484 607

Online

organizacion@avalora.comwww.avalora.com

Editor:

Mario Jiménez González

Number of pages: 21 Date of presentation: 20/11/2025

Description:

This document details the steps required to configure the dependencies of the Ticket Request add-in, including creating and preparing the site and list in SharePoint Online, as well as deploying and configuring the Azure Function service that acts as a secure intermediary between the Add-in and external services.

Control Version:

Version	Date	Modified section	Content of the changes made
1	20/11/2025	N/A	Initial version

Content

1. Purpose

This document aims to clearly and concisely describe the steps required to configure the dependencies that the Ticket Request add-in requires for its proper functioning.

It includes the preparation of the environment in SharePoint Online, where the configuration values used by the add-in are stored, as well as the definition, deployment, and configuration of the Azure Function, which is responsible for processing the add-in's requests and communicating with the necessary external services.

The purpose of this document is to serve as a technical guide for administrators and support teams, ensuring that the services involved (SPO and Azure Function) are correctly configured and operational, thus allowing the Ticket Request add-in to work in a secure, stable and according to the intended design.

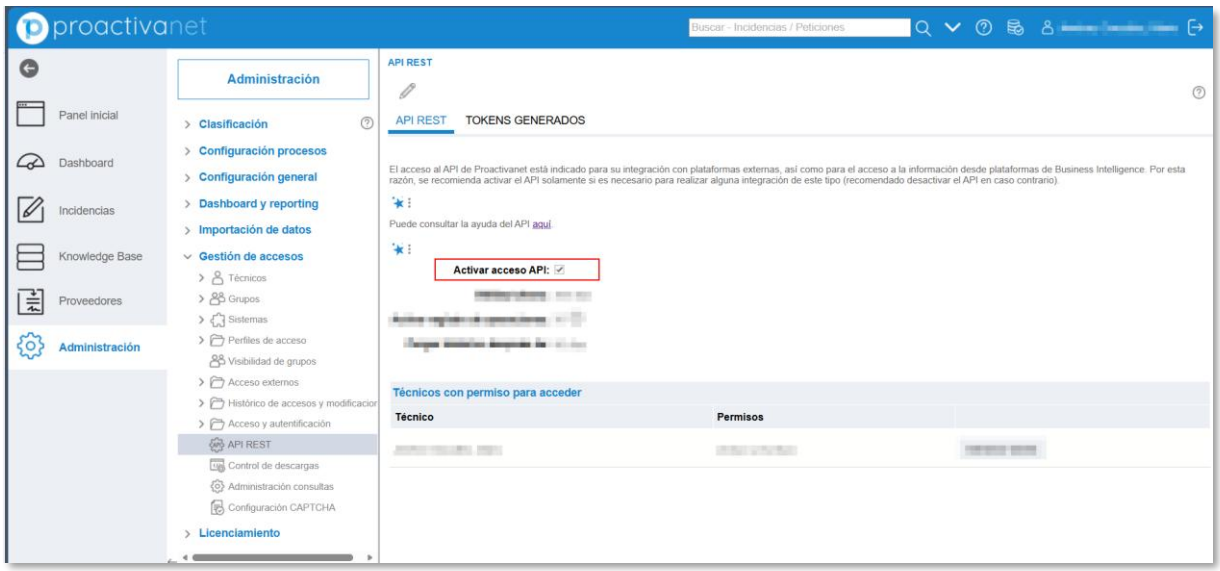
2. Initial requirements

To complete the configuration of the Ticket Request add-in, you need to have the following prerequisites:

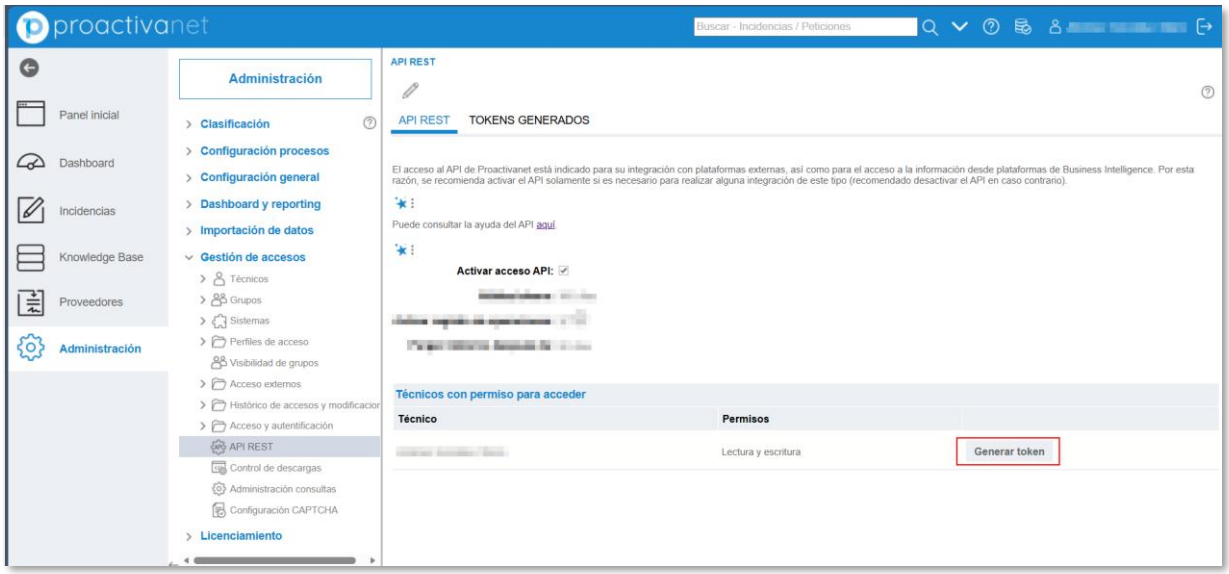
- Access to Proactivanet instance management.
- Active Azure subscription.
- Enterprise application in Microsoft Entra ID for integration with Graph.
- Permission to create and manage Azure Functions.
- Access to the SharePoint Online management portal.

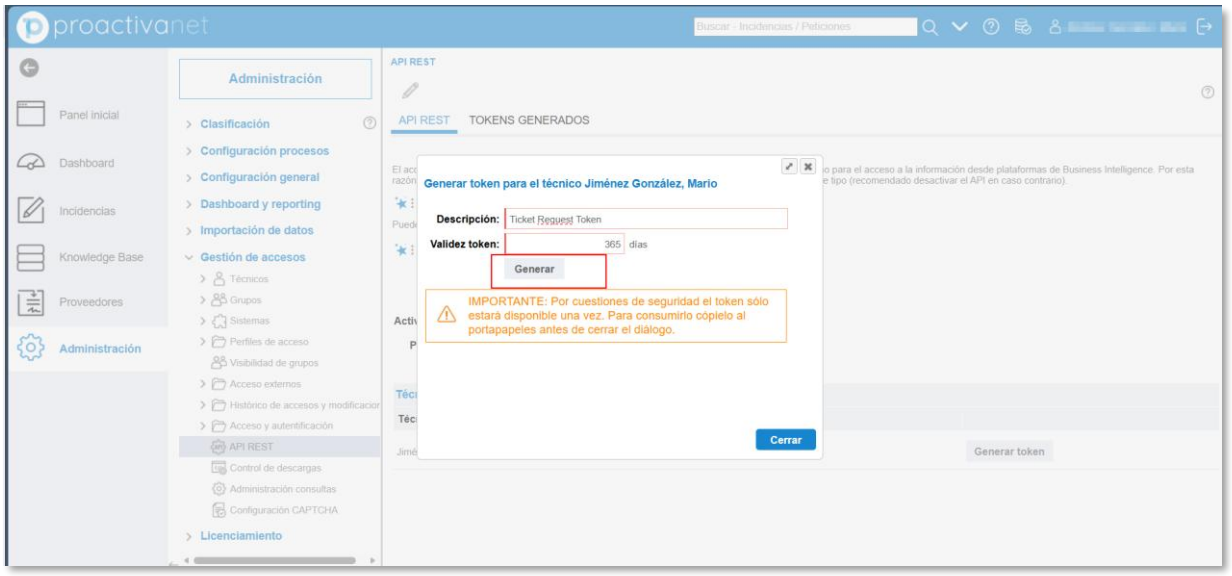
3. Proactivanet Administration

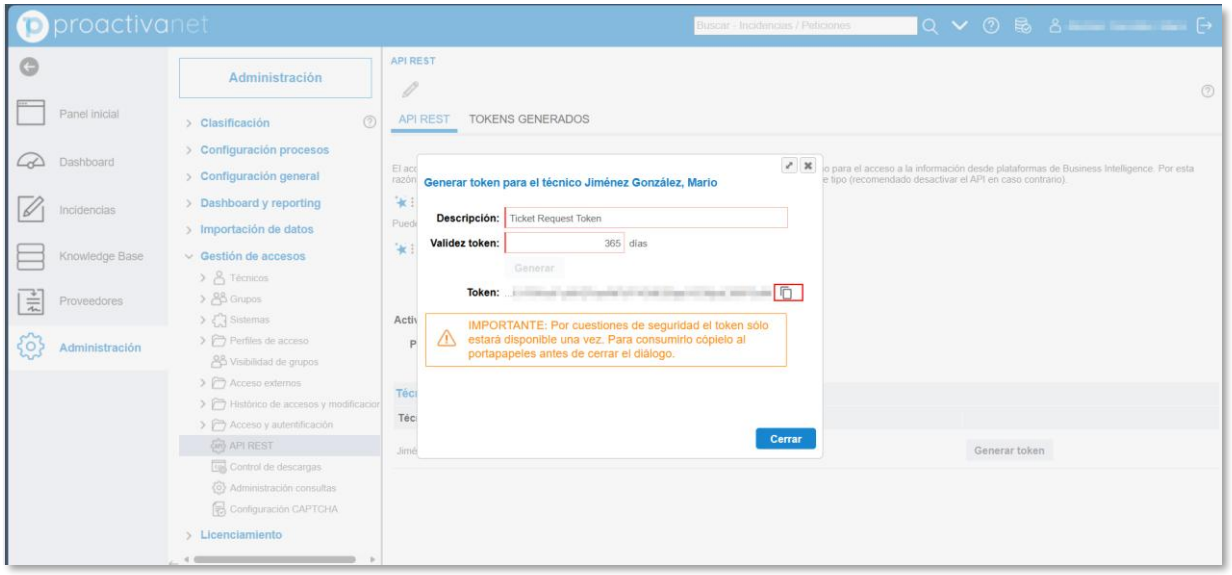
In the admin panel, Administration / Access Management / REST API to activate API access.



Generate a token with Read and Write permissions.





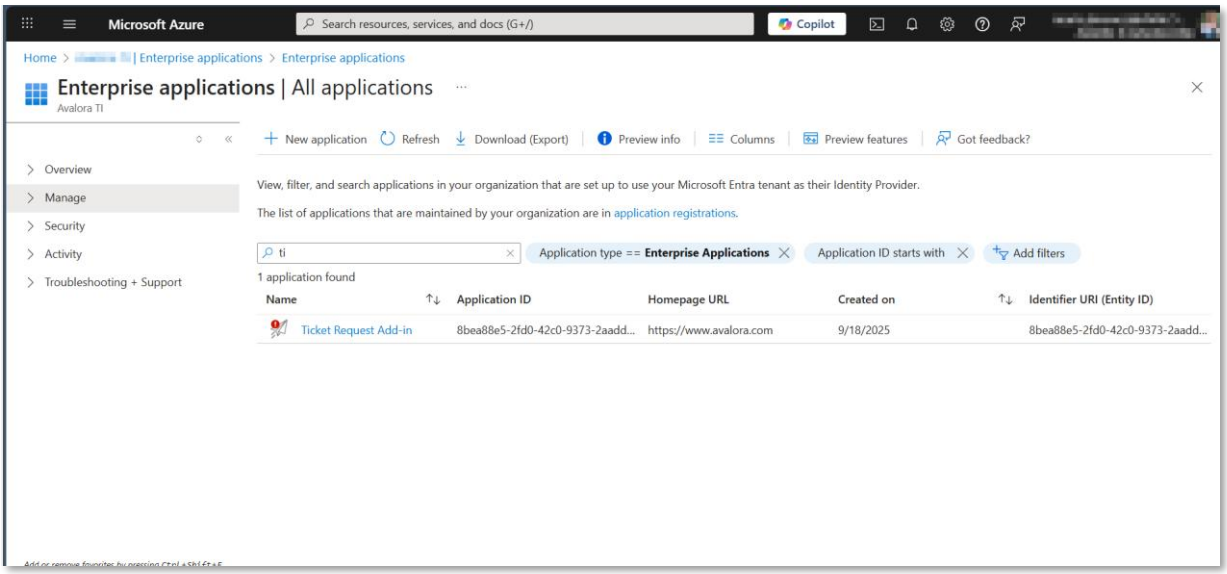


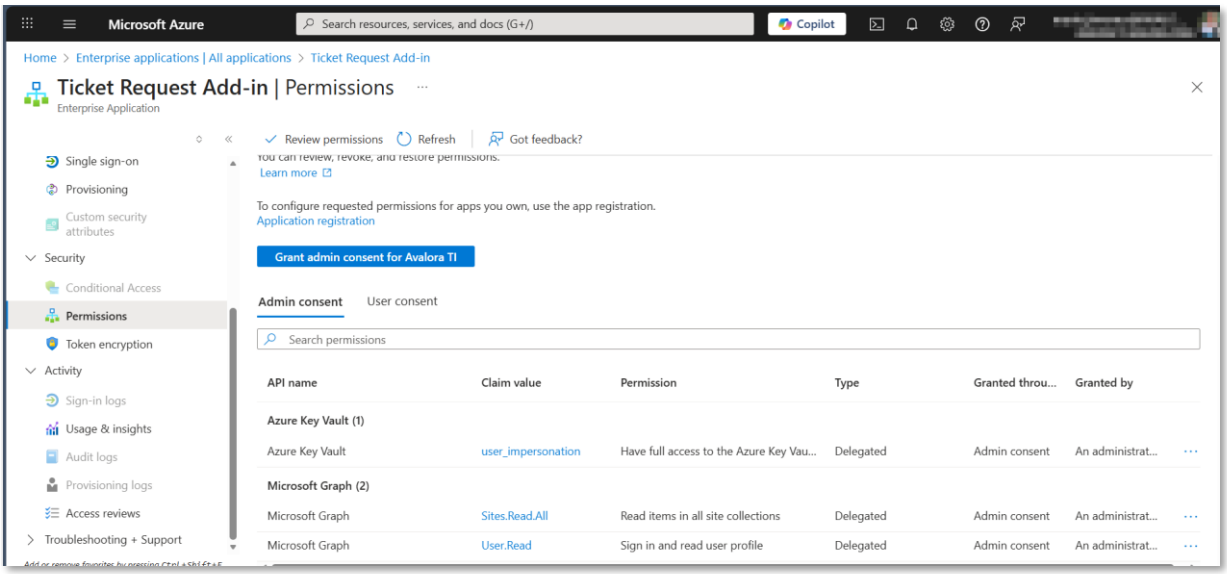
Copy the generated token to include it in the SharePoint Online list in section 6.

4. Microsoft Entra ID enterprise applications

This step is done automatically when a user installs the add-in on their tenant.

The delegated permissions granted to the application are access to user information and read at the SharePoint Online site level.





5. Azure Function

The Azure Function acts as a secure intermediary between the **Ticket Request add-in** and the Proactivanet API. Its function is to receive the request for the add-in and forward the request to the final endpoint.

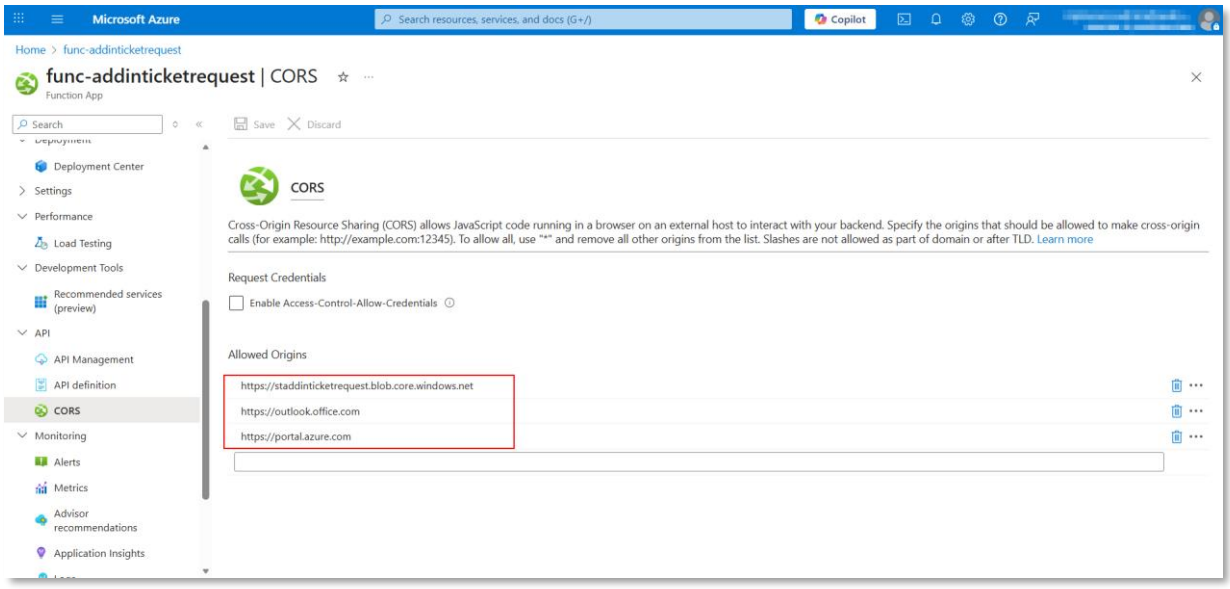
5.1 Creating the Resource

1. Access the **Azure portal**.
2. Create a **Function App** resource with the following recommended features:
    - **Runtime stack:** PowerShell Core.
    - **Version: 7.4**
    - **Plan:** Flex Consumption or Premium, depending on your needs.
3. The function name is customizable; it has no dependency on the plugin.

5.2 CORS Configuration

To bypass browser blocks and allow the Function to accept secure calls from specific domains, the following allowed domains are configured:

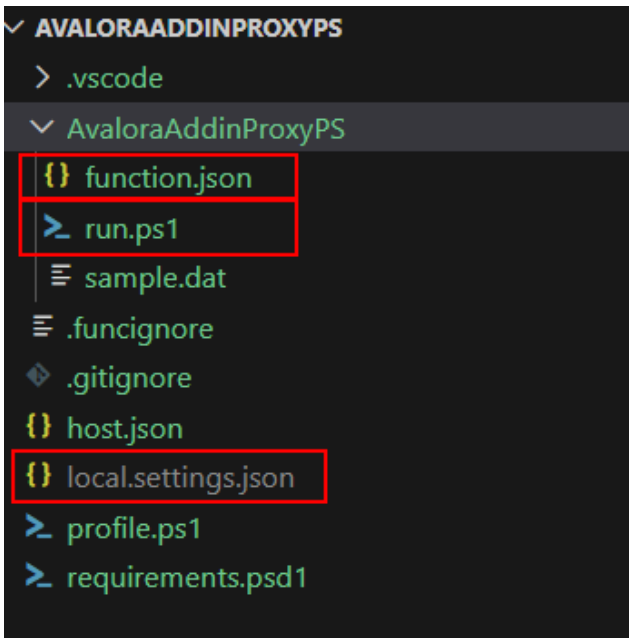
- https://staddinticketrequest.blob.core.windows.net
- https://outlook.office.com
- https://portal.azure.com



5.3 Script.

From a code editor such as Visual Studio Code, an Azure Function-like project is created to deploy the script.

5.3.1 Project scaffolding



5.3.2 File content *function.json*

```
{
  "bindings": [
{
      "authLevel": "function",
      "type": "httpTrigger",
      "direction": "in",
      "name": "Request",
      "methods": [
        "post"
]
},
{
      "type": "http",
      "direction": "out",
      "name": "Response"
}
]
}
```

5.3.3 File content local.settings.json

```
{
  "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "UseDevelopmentStorage=true",
    "FUNCTIONS_WORKER_RUNTIME_VERSION": "7.4",
    "FUNCTIONS_WORKER_RUNTIME": "powershell",
    "ALLOWED_ORIGINS":
"https://staddinticketrequest.blob.core.windows.net,https://outlook.office.com,
https://portal.azure.com"
}
}
```

5.3.4 Contents run.ps1 *file*

```
using namespace System.Net

param($Request, $TriggerMetadata)

# ============================================================
#   Attachment Flag
# ============================================================
function ParseAttachmentBody {
    param($bodyRaw)

    if ($bodyRaw -is [hashtable] -or
        $bodyRaw -is [System.Collections.Specialized.OrderedDictionary]) {
        return $bodyRaw
    }

    if ($bodyRaw -is [string]) {
        $jsonString = $bodyRaw.Trim()

        try {
            $parsed = $jsonString | ConvertFrom-Json -ErrorAction Stop
            return $parsed
        }
        catch {
            Write-Host "ParseAttachmentBody: ERROR → body enviado no es JSON
válido"
            return $null
        }
    }

    Write-Host "ParseAttachmentBody: Tipo no soportado:
$($bodyRaw.GetType().FullName)"
    return $null
}
```

```
# ============================================================
#   PRE-FLIGHT OPTIONS
# ============================================================
if ($Request.Method -eq "OPTIONS") {
    $response = @{
        StatusCode = [HttpStatusCode]::NoContent
        Headers    = @{
            "Access-Control-Allow-Methods" = "GET,POST,OPTIONS"
            "Access-Control-Allow-Headers" = "Authorization,Content-Type"
        }
    }
    Push-OutputBinding -Name Response -Value $response
    return
}

# ============================================================
#   JSON BODY READER
# ============================================================
$inputJson = $Request.Body

$targetUrl = $inputJson.endpoint
$method = $inputJson.verb
$payload = $inputJson.body
$attachentFlag = $inputJson.attachment

Write-Host "REQUEST: verb=$method endpoint=$targetUrl
attachment=$($inputJson.body.Attachment)"

if (-not $targetUrl) {
    Write-Host "ERROR: Missing endpoint"
    $res = @{
        StatusCode = [HttpStatusCode]::BadRequest
        Body       = "Missing endpoint"
        Headers    = @{}
    }

    Push-OutputBinding -Name Response -Value $res
    return
}

# ============================================================
#   AUTH: SENDER BEARER FROM ADDINN
# ============================================================
$authHeader = $Request.Headers["Authorization"]

if (-not $authHeader) {
```

```
    Write-Host "ERROR: Missing Authorization header"

    $res = @{
        StatusCode = [HttpStatusCode]::BadRequest
        Body       = "Missing Authorization header"
        Headers    = @{}
    }

    Push-OutputBinding -Name Response -Value $res
    return
}

$httpClient = [System.Net.Http.HttpClient]::new()

try {

    # ========================================================
    #  REQUESTS WITH ATTACHMENT → MULTIPART/FORMDATA
    # ========================================================

    if ($attachentFlag) {
        Write-Host "ATTACHMENT DETECTED"

        $parsed = ParseAttachmentBody $payload
        if (-not $parsed) {
            Push-OutputBinding -Name Response -Value @{
                StatusCode = 400
                Body       = "Invalid attachment body"
            }
            return
        }

        $cleanBase64 = ([string]$parsed.FileContent) -replace "\s", ""

        $cleanBase64 = [string]$parsed.FileContent
        $cleanBase64 = $cleanBase64.Trim() -replace '\s', ''

        try {
            $tmpBytes = [System.Convert]::FromBase64String($cleanBase64)

            if ($tmpBytes -isnot [byte[]]) {
                Write-Host "ERROR: Decoded Base64 is not byte[]"
                throw "Decoded object is not a byte array"
            }

            [byte[]]$fileBytes = $tmpBytes
        }
        catch {
```

```
            Write-Host "ERROR DECODIFICANDO BASE64 → $($_.Exception.Message)"
            throw $_
        }

        Write-Host "ATTACHMENT: file=$($parsed.FileName)
bytes=$($fileBytes.Length)"

        $multipart = New-Object System.Net.Http.MultipartFormDataContent

        $formJson = @{ PanUsers_id = $parsed.PanUsers_id } | ConvertTo-Json -
Depth 5
        $formPart = New-Object System.Net.Http.StringContent($formJson,
[System.Text.Encoding]::UTF8, "application/json")
        $formPart.Headers.ContentDisposition =
[System.Net.Http.Headers.ContentDispositionHeaderValue]::new("form-data")
        $formPart.Headers.ContentDisposition.Name = '"FormData"'
        $multipart.Add($formPart)

        $filePart = New-Object System.Net.Http.ByteArrayContent( , $fileBytes )

        $ext = [IO.Path]::GetExtension($parsed.FileName).ToLower()
        $mime = switch ($ext) {
            ".png" { "image/png" }
            ".jpg" { "image/jpeg" }
            ".jpeg" { "image/jpeg" }
            ".pdf" { "application/pdf" }
            default { "application/octet-stream" }
        }

        $filePart.Headers.ContentType = $mime
        $filePart.Headers.ContentDisposition =
[System.Net.Http.Headers.ContentDispositionHeaderValue]::new("form-data")
        $filePart.Headers.ContentDisposition.Name = '"File"'
        $filePart.Headers.ContentDisposition.FileName = '"{0}"' -f
$parsed.FileName

        $multipart.Add($filePart)

        $httpMethod = [System.Net.Http.HttpMethod]::new($method)
        $httpRequest = New-Object
System.Net.Http.HttpRequestMessage($httpMethod, $targetUrl)
        $httpRequest.Headers.TryAddWithoutValidation("Authorization",
$authHeader)
        $httpRequest.Content = $multipart

        Write-Host "CALL → Proactivanet $method $targetUrl (multipart)"

        $httpResponse = $httpClient.SendAsync($httpRequest).Result
```

```
        $bodyText = $httpResponse.Content.ReadAsStringAsync().Result

        Push-OutputBinding -Name Response -Value @{
            StatusCode = [int]$httpResponse.StatusCode
            Body       = $bodyText
            Headers    = @{ "Content-Type" = "application/json" }
        }
        return
    }

    # ========================================================
    #   REQUESTS WITH NO ATTACHMENT
    # ========================================================
    $httpRequest = New-Object System.Net.Http.HttpRequestMessage($method,
$targetUrl)
    $httpRequest.Headers.Add("Authorization", $authHeader)
    $httpRequest.Headers.Add("Accept", "application/json")

    if ($payload) {
        $jsonContent = New-Object System.Net.Http.StringContent($payload,
[System.Text.Encoding]::UTF8, "application/json")
        $httpRequest.Content = $jsonContent
    }

    $response = $httpClient.SendAsync($httpRequest).Result
    $bodyText = $response.Content.ReadAsStringAsync().Result

    $contentType = $response.Content.Headers.ContentType
    if (-not $contentType) { $contentType = "application/json" }

    Write-Host "PROACTIVANET STATUS = $($response.StatusCode)"

    $res = @{
        StatusCode = [int]$response.StatusCode
        Body       = $bodyText
        Headers    = @{ "Content-Type" = $contentType.ToString() }
    }

    Push-OutputBinding -Name Response -Value $res
    return
}
catch {
    $err = $_.Exception.Message

    Write-Error "EXCEPTION: $err"

    $res = @{
        StatusCode = 502
```

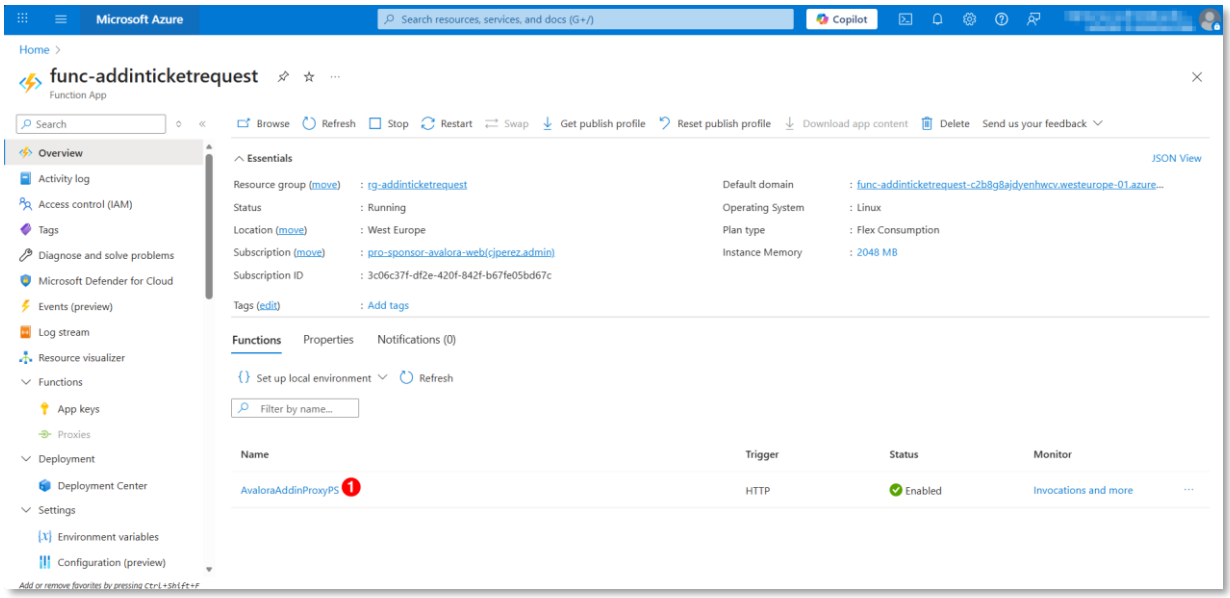
```
        Body        = "Proxy error: $err"
        Headers     = @{}
    }

    Push-OutputBinding -Name Response -Value $res
    return
}
```

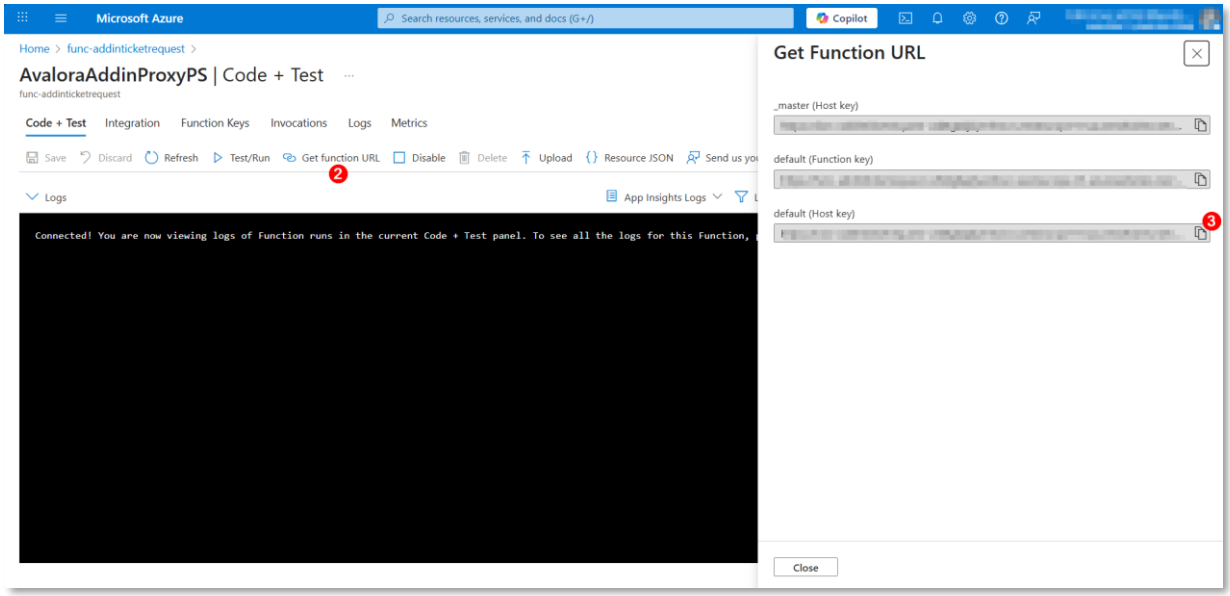
5.4 Getting url function

Once the function is deployed, it is necessary to obtain the endpoint where the requests for the add-in will be sent.

1. In the service overview, click on the function name



2. Click on the Get function URL option and copy the endpoint that includes the function key.



The endpoint will be used in the SharePoint Online list in Section 6.

6. SharePoint Online Management

The Ticket Request add-in obtains from SharePoint Online the configuration values required for its operation. This uses a specific site and a list where the connection keys and endpoints that the add-in will consume are stored.

6.1 Site creation

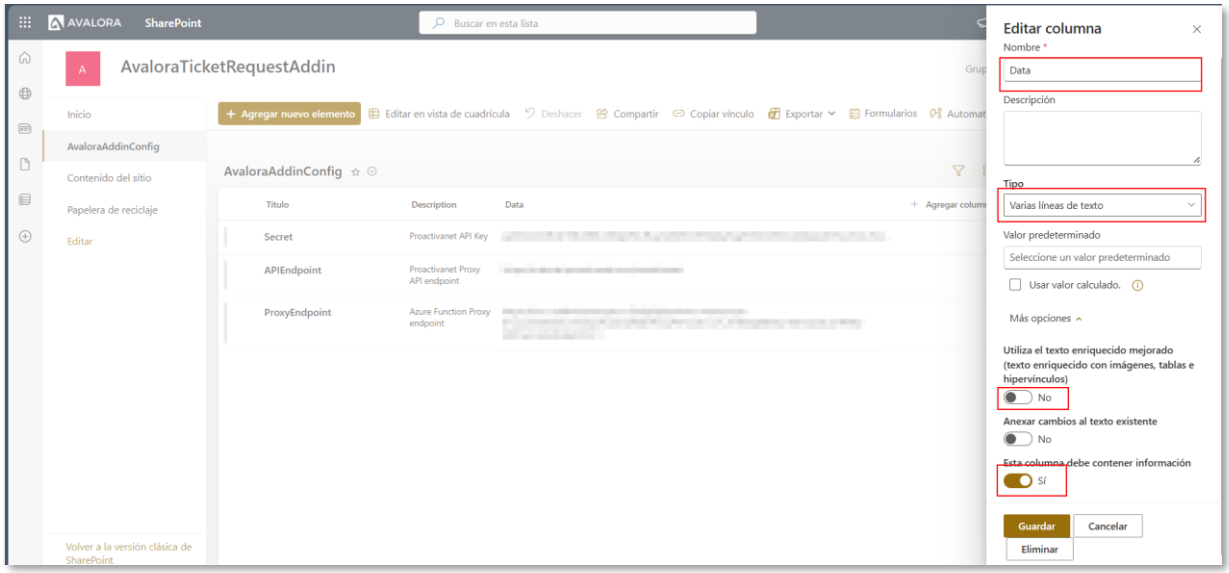
From the SharePoint admin center.

- Create a Microsoft 365 team site type without a group.
- Name: AvaloraTicketRequestAddin

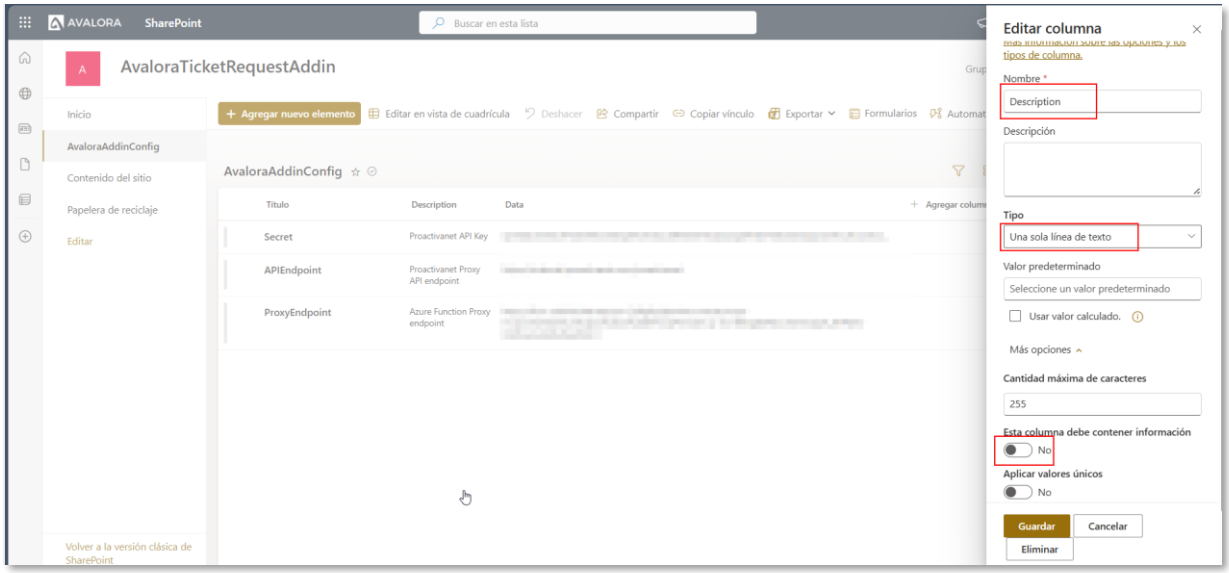
6.2 Creating the Configuration List

Within the AvaloraTicketRequestAddin site, create a new list called AvaloraAddinConfig

1. Add a custom column, named Data with the following settings:



2. Optional. Add a custom column, named **Description** with the following settings:



The list should contain **three items**, each in the following format:

Title	Description	Date
APIEndpoint	Proactivanet Proxy API endpoint	Url instance Proactivanet. Example: https://contoso.proactivanet.com/proactivanet
Secret	Proactivanet API Key	Token generated in section 3
ProxyEndpoint	Azure Function Proxy endpoint	Azure Function URL obtained in section 5.4

6.3 Access Validation

Verify that the users who will use the add-in have read access to the list.

7. Recommended Maintenance

1. Periodically review the configuration list.
    - Verify that URLs and secrets are still valid.
    - Update the external backend endpoint if it changes.
2. Proactivanet token rotation
    - Update the token in the SharePoint Online list if it expires.
3. Service Supervision Function.
    - Review logs in Application Insights or Monitor for errors or spikes in usage.
    - Check that there are no recurring failures when forwarding requests.
4. Permission control.
    - Validate that broader permissions than strictly necessary are not granted.
5. Plugin update.
    - If a new version of the add-in is deployed, validate that it does not require new keys, columns, or endpoints.

8. References

- About support or customizations in Azure services, SharePoint Online, and/or the add-in, please contact contacto@avalora.com or + 34 913 484 848.
- About support at Proactivanet, please contact the service provider.