

How to authenticate MS Azure and Azure AD /ARM using Graph API call for Bearer Token

EXAMPLES
& SOLUTIONS

Table of Contents

1. Introduction
2. Prerequisites
3. Proper Configuration Within ServiceNow Instance
 - a. HTTPS Request
 - b. Flow Action
4. Security
5. Summary

Introduction

ServiceNow has created basic SAML authentication for Azure AD users to allow specific users to login using their Azure credentials, but unfortunately in & out data flow configuration through API is scattered across different tables and not intuitive enough to quickly setup API integration for example to update CMDB entries. Installation of proper spoke in the IntegrationHub kind of solves the problem, however some data cannot be gathered from Azure spokes, or type of authentication is not granting proper access to particular resources. That is why, to workaround such a case, we need to go through oauth with additional bearer token authentication. Another advantage of the solution is that any resources can be accessible this way, even without installing additional spokes, but it still requires proper permissions and roles from the Azure side.

CHAPTER 2

Prerequisites

First, and the most important thing to start with when creating this solution on ServiceNow level, is to make sure that the azure admin is properly set up with all permissions and roles required to read or manipulate specific data. It's very important because we, and azure admins, have to differentiate between delegated and application permissions. The first one requires initiating a session by properly granted users. For the application permission however, any user can access data, but the API requests must be sent from the one of the granted applications - in our case from the ServiceNow instance. Before you start doing proper configuration, you need to understand the requirements and agree with your team which way is actually better for your setup.

Another thing that is not “must have”, but really helpful for dealing with API calls, is to have an IntegrationHub license which is not a starter.

This will make the whole process much easier to develop instead of using normal and free actions. OOTB starter subscription doesn't process REST calls, but you can do via scripted restmessagev2 call using script step within the action.

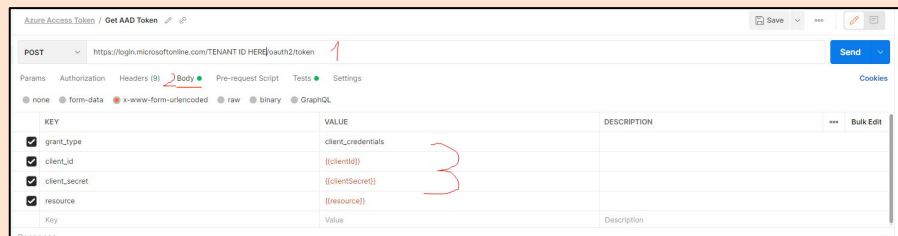
Another smart move is to get any API builder platform such as Postman, which will help you test the endpoints and API calls.

And last, but not least, are the attributes by which you will authenticate with azure - accesses are the most important part. But if we make API calls without the proper body, it will return with nothing because we didn't pass information about resource ID from where we want to fetch data for example. Below is the minimum that you have to get from azure admin to get in successfully:

1. grant_type - always has to be "client_credentials" to properly format the rest of the request body and headers
2. client_id - it's the sys_id of our application in azure
3. client_secret - it's a password by other words. Usually it's randomly generated but also can be encoded.
4. resource - simply it's the constant url after which we will be adding queries. Usually it's <https://graph.microsoft.com/> but for some resources it can differ.
5. tenantID - it's the ID of the environment (database) from where we want to gather data

To start fetching or updating resources you have to authenticate using the schema:

https://login.microsoftonline.com/{{TENANT_ID_HERE}}/oauth2/token and rest of the above parameters pass as a body of your query.



CHAPTER 3

Proper Configuration Within ServiceNow Instance

If all Prerequisites have been done and set up properly, you should receive the bearer token without any issues. However, the whole solution still needs to be moved to the client ServiceNow instance to have it fully automated and integrated with further integration tasks and actions.

Depending on how the configuration is built, there are two ways of how we can actually receive and later store bearer token before it expires - **HTTP Request** from **Rest Messages** table and by using **flow action**.

A) HTTPS Request

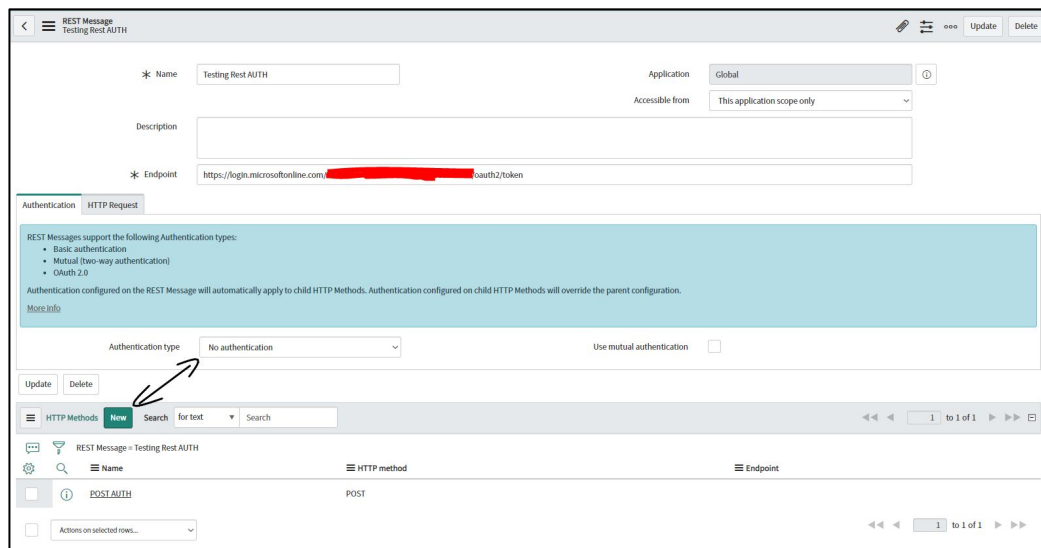
Open Rest Messages table and create new record

The screenshot displays a software interface with a dark sidebar on the left and a light-colored main content area. The sidebar contains the following menu items: 'Orchestration', '▼ Activity Dependencies', 'REST Messages' (highlighted with a black arrow), 'System Web Services', '▼ Outbound', and 'REST Message'. The main content area shows a table with columns: 'Name' (with a dropdown arrow and a black arrow pointing to it), 'Description', and 'Endpoint'. Each column has a search bar below its header. The table lists five records, each with an information icon, a checkbox, and a link:

	Name	Description	Endpoint
<input type="checkbox"/>	Accurate Background Check API	Accurate Background Check API	https://api.accuratebackground.com/v3
<input type="checkbox"/>	AzureCog	Connect with Cognizant to get Azure serv...	https://appmgmtweuonboarding-apis-fa.a...
<input type="checkbox"/>	CTS_Send_Attachments		https://integrator-dev.cognizantgoc.com/...
<input type="checkbox"/>	Firebase Cloud Messaging Send		https://fcm.googleapis.com/fcm/send
<input type="checkbox"/>	ServiceNowMobileApp Push		https://{pushHost}/api/now/v1/push/\${ap...

Name newly created Rest Message record and copy endpoint from the postman URL and create new **HTTP POST Method**. For both, in the authentication field use **No Authentication**. For the HTTP Method endpoint, it is not needed as it will be inherited from the parent by default in this case.

Now the most important part where you set up the whole body and header is needed. To make sure all parameters will be passed properly, I strongly recommend to use Postman Code Snippet functionality from where it can be set up in which form the API call is created. In our case it's **HTTP**.



Now it's the time to pass all gathered details directly to the REST Message. Because previously on the postman, the POST call was made in **x-www-form-urlencoded**, the same format must be sent from ServiceNow. To achieve the proper header to be provided, you need to copy the body of our message from the Code snippet to the **content** field and save it. Then you can check if everything has been set up correctly by doing a simple test call, and see if we receive a bearer token as a response.

The screenshot shows the Postman interface for a REST client request named "Get AAD Token". The request is a POST to the URL `https://login.microsoftonline.com/TENANT ID/oauth2/token`. The request body is configured as `x-www-form-urlencoded` and contains the following parameters:

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> grant_type	client_credentials	
<input checked="" type="checkbox"/> client_id	{{clientId}}	
<input checked="" type="checkbox"/> client_secret	{{clientSecret}}	
<input checked="" type="checkbox"/> resource	{{resource}}	
Key	Value	Description

The right-hand pane shows the "Code snippet" for the request, which is an HTTP POST:

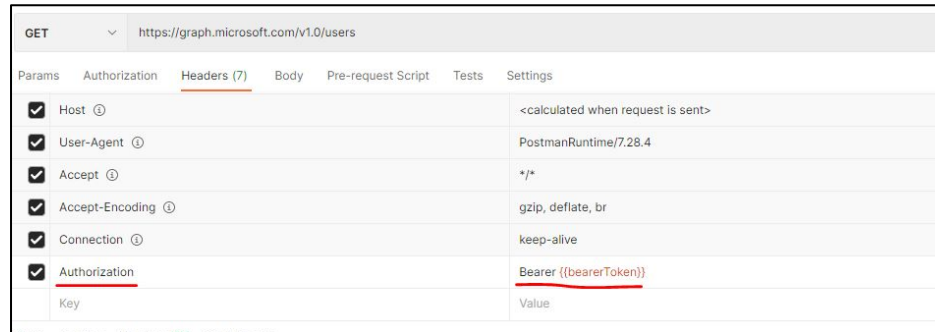
```

1 POST /TENANT ID/oauth2/token HTTP/1.1
2 Host: login.microsoftonline.com
3 Content-Type: application/
  x-www-form-urlencoded
4 Cookie:
  fpc=AvYrwutNJwtPqQZbxbBNpfT-Twa_AQAAAL
  bQEdkOAAAA; stsservicecookie=estsid;
  x-ms-gateway-slice=estsid
5 Content-Length: 171
6
7 grant_type=client_credentials&
  client_id={{clientId}}&
  client_secret={{clientSecret}}&
  resource=https%3A%2F%2Fgraph.
  microsoft.com%2F
  
```

At the bottom, the response status is `200 OK` with a time of `373 ms` and a size of `2.42 KB`. The response is displayed in the "Pretty" format.

```
[{"token_type": "Bearer", "expires_in": "3599", "ext_expires_in": "3599", "expires_on": "1635777400", "not_before": "1635773500", "resource": "https://graph.microsoft.com/", "access_token": "eyJ0eXAiOiJKV1QiOiJub25jZSI6Iml2bmhVPMWhpUpk1R3FkUGlGMWJnJUI1RaU3VBTjYnTBDz09idy1OLWk0ckkiLCJjbGciOiJSUzI1NiIsImlzIjoiCi6l6mwc2LE1NTBjQ0g0eEJWWkxVEd3b3lnSNzY4Ml5lmtpZC16l6mwc2LE1NTBjQ0g0eEJWWkxVEd3b3lnSNzY4Mj9jEjyJhdWQioiJodHRwczovL2dyYXB0Lm1pY3Jvc29mdC5jb20viiaWxzXzIjoiaHR0cHM6Ly9rd2hmud2lwZG93cy5uZXQvNThlMzBkZytZDdmZS00ODVhLWJhMzEtNmRjMzVhZyRhZ2ZlY2l5mlk2Ml6MTY2NzY2Ml6MwCwibmJmXjoiNmJ1NzczNtAwLClleAhoiEj2MZ3n0C0MAD5lmlFpbyl6kUmVmdZr2w3WTVQ0MvdyN0l6mUxCTEhRWkY2UXpBDt09liwYXbW2Xpc3BsYXlWY2l1ljoieU2ydmZjZlY2V5dWByBUUEkUjHJzC3l5mFwCgkljZjQjUwNz44YUWUTY1K0E00QWRkLT5K2MDtMCEJYwWnT6kNmKliwYXbW2R3Y3V0IjdiaWwYXbWj0jiaHR0cHM6Ly9zd2hmud2lwZG93cy5uZXQvNThlMzBkZytZDdmZS00ODVhLWJhMzEtNmRjMzVhZyRhZ2ZlY2l5mlk2Ml6MTY2NzY2Ml6MwCwibmJmXjoiNmJ1NzczNtAwLClleAhoiEj2MZ3n0C0MAD5lmlFpbyl6kUmVmdZr2w3WTVQ0MvdyN0l6mUxCTEhRWkY2UXpBDt09liwYXbW2Xpc3BsYXlWY2l1ljoieU2ydmZjZlY2V5dWByBUUEkUjHJzC3l5mFwCgkljZjQjUwNz44YUWUTY1K0E00QWRkLT5K2MDtMCEJYwWnT6kNmKliwYXbW2R3Y3V0IjdiaWwYXbWj0jiaHR0cHM6Ly9zd2hmud2lwZG93cy5uZXQvNThlMzBkZytZDdmZS00ODVhLWJhMzEtNmRjMzVhZyRhZ2ZlY2l5mlk2Ml6MTY2NzY2Ml6MwCwibmJmXjoiNmJ1NzczNtAwLClleAhoiEj2MZ3n0C0MAD5lmlFpbyl6kUmVmdZr2w3WTVQ0MvdyN0l6mUxCTEhRWkY2UXpBDt09liwYXbW2Xpc3BsYXlWY2l1ljoieU2ydmZjZlY2V5dWByBUUEkUjHJzC3l5mFwCgkljZjQjUwNz44YUWUTY1K0E00QWRkLT5K2MDtMCEJYwWnT6kNmKliwYXbW2R3Y3V0IjdiaWwYXbWj0jiaHR0cHM6Ly9zd2hmud2lwZG93cy5uZXQvNThlMzBkZytZDdmZS00ODVhLWJhMzEtNmRjMzVhZyRhZ2ZlY2l5mlk2Ml6MTY2NzY2Ml6MwCwibmJmXjoiNmJ1NzczNtAwLClleAhoiEj2MZ3n0C0MAD5lmlFpbyl6kUmVmdZr2w3WTVQ0MvdyN0l6mUxCTEhRWkY2UXpBDt09liwYXbW2Xpc3BsYXlWY2l1ljoieU2ydmZjZlY2V5dWByBUUEkUjHJzC3l5mFwCgkljZjQjUwNz44YUWUTY1K0E00QWRkLT5K2MDtMCEJYwWnT6kNmKliwYXbW2R3Y3V0IjdiaWwYXbWj0jiaHR0cHM6Ly9zd2hmud2lwZG93cy5uZXQvNThlMzBkZytZDdmZS00ODVhLWJhMzEtNmRjMzVhZyRhZ2ZlY2l5mlk2Ml6MTY2NzY2Ml6MwCwibmJmXjoiNmJ1NzczNtAwLClleAhoiEj2MZ3n0C0MAD5lmlFpbyl6kUmVmdZr2w3WTVQ0MvdyN0l6mUxCTEhRWkY2UXpBDt09liwYXbW2Xpc3BsYXlWY2l1ljoieU2ydmZjZlY2V5dWByBUUEkUjHJzC3l5mFwCgkljZjQjUwNz44YUWUTY1K0E00QWRkLT5K2MDtMCEJYwWnT6kNmKliwYXbW2R3Y3V0IjdiaWwYXbWj0jiaHR0cHM6Ly9zd2hmud2lwZG93cy5uZXQvNThlMzBkZytZDdmZS00ODVhLWJhMzEtNmRjMzVhZyRhZ2ZlY2l5mlk2Ml6MTY2NzY2Ml6MwCwibmJmXjoiNmJ1NzczNtAwLClleAhoiEj2MZ3n0C0MAD5lmlFpbyl6kUmVmdZr2w3WTVQ0MvdyN0l6mUxCTEhRWkY2UXpBDt09liwYXbW2Xpc3BsYXlWY2l1ljoieU2ydmZjZlY2V5dWByBUUEkUjHJzC3l5mFwCgkljZjQjUwNz44YUWUTY1K0E00QWRkLT5K2MDtMCEJYwWnT6kNmKliwYXbW2R3Y3V0IjdiaWwYXbWj0jiaHR0cHM6Ly9zd2hmud2lwZG93cy5uZXQvNThlMzBkZytZDdmZS00ODVhLWJhMzEtNmRjMzVhZyRhZ2ZlY2l5mlk2Ml6MTY2NzY2Ml6MwCwibmJmXjoiNmJ1NzczNtAwLClleAhoiEj2MZ3n0C0MAD5lmlFpbyl6kUmVmdZr2w3WTVQ0MvdyN0l6mUxCTEhRWkY2UXpBDt09liwYXbW2Xpc3BsYXlWY2l1ljoieU2ydmZjZlY2V5dWByBUUEkUjHJzC3l5mFwCgkljZjQjUwNz44YUWUTY1K0E00QWRkLT5K2MDtMCEJYwWnT6kNmKliwYXbW2R3Y3V0IjdiaWwYXbWj0jiaHR0cHM6Ly9zd2hmud2lwZG93cy5uZXQvNThlMzBkZytZDdmZS00ODVhLWJhMzEtNmRjMzVhZyRhZ2ZlY2l5mlk2Ml6MTY2NzY2Ml6MwCwibmJmXjoiNmJ1NzczNtAwLClleAhoiEj2MZ3n0C0MAD5lmlFpbyl6kUmVmdZr2w3WTVQ0MvdyN0l6mUxCTEhRWkY2UXpBDt09liwYXbW2Xpc3BsYXlWY2l1ljoieU2ydmZjZlY2V5dWByBUUEkUjHJzC3l5mFwCgkljZjQjUwNz44YUWUTY1K0E00QWRkLT5K2MDtMCEJYwWnT6kNmKliwYXbW2R3Y3V0IjdiaWwYXbWj0jiaHR0cHM6Ly9zd2hmud2lwZG93cy5uZXQvNThlMzBkZytZDdmZS00ODVhLWJhMzEtNmRjMzVhZyRhZ2ZlY2l5mlk2Ml6MTY2NzY2Ml6MwCwibmJmXjoiNmJ1NzczNtAwLClleAhoiEj2MZ3n0C0MAD5lmlFpbyl6kUmVmdZr2w3WTVQ0MvdyN0l6mUxCTEhRWkY2UXpBDt09liwYXbW2Xpc3BsYXlWY2l1ljoieU2ydmZjZlY2V5dWByBUUEkUjHJzC3l5mFwCgkljZjQjUwNz44YUWUTY1K0E00QWRkLT5K2MDtMCEJYwWnT6kNmKliwYXbW2R3Y3V0IjdiaWwYXbWj0jiaHR0cHM6Ly9zd2hmud2lwZG93cy5uZXQvNThlMzBkZytZDdmZS00ODVhLWJhMzEtNmRjMzVhZyRhZ2ZlY2l5mlk2Ml6MTY2NzY2Ml6MwCwibmJmXjoiNmJ1NzczNtAwLClleAhoiEj2MZ3n0C0MAD5lmlFpbyl6kUmVmdZr2w3WTVQ0MvdyN0l6mUxCTEhRWkY2UXpBDt09liwYXbW2Xpc3BsYXlWY2l1ljoieU2ydmZjZlY2V5dWByBUUEkUjHJzC3l5mFwCgkljZjQjUwNz44YUWUTY1K0E00QWRkLT5K2MDtMCEJYwWnT6kNmKliwYXbW2R3Y3V0IjdiaWwYXbWj0jiaHR0cHM6Ly9zd2hmud2lwZG93cy5uZXQvNThlMzBkZytZDdmZS00ODVhLWJhMzEtNmRjMzVhZyRhZ2ZlY2l5mlk2Ml6MTY2NzY2Ml6MwCwibmJmXjoiNmJ1NzczNtAwLClleAhoiEj2MZ3n0C0MAD5lmlFpbyl6kUmVmdZr2w3WTVQ0MvdyN0l6mUxCTEhRWkY2UXpBDt09liwYXbW2Xpc3BsYXlWY2l1ljoieU2ydmZjZlY2V5dWByBUUEkUjHJzC3l5mFwCgkljZjQjUwNz44YUWUTY1K0E00QWRkLT5K2MDtMCEJYwWnT6kNmKliwYXbW2R3Y3V0IjdiaWwYXbWj0jiaHR0cHM6Ly9zd2hmud2lwZG93cy5uZXQvNThlMzBkZytZDdmZS00ODVhLWJhMzEtNmRjMzVhZyRhZ2ZlY2l5mlk2Ml6MTY2NzY2Ml6MwCwibmJmXjoiNmJ1NzczNtAwLClleAhoiEj2MZ3n0C0MAD5lmlFpbyl6kUmVmdZr2w3WTVQ0MvdyN0l6mUxCTEhRWkY2UXpBDt09liwYXbW2Xpc3BsYXlWY2l1ljoieU2ydmZjZlY2V5dWByBUUEkUjHJzC3l5mFwCgkljZjQjUwNz44YUWUTY1K0E00QWRkLT5K2MDtMCEJYwWnT6kNmKliwYXbW2R3Y3V0IjdiaWwYXbWj0jiaHR0cHM6Ly9zd2hmud2lwZG93cy5uZXQvNThlMzBkZytZDdmZS00ODVhLWJhMzEtNmRjMzVhZyRhZ2ZlY2l5mlk2Ml6MTY2NzY2Ml6MwCwibmJmXjoiNmJ1NzczNtAwLClleAhoiEj2MZ3n0C0MAD5lmlFpbyl6kUmVmdZr2w3WTVQ0MvdyN0l6mUxCTEhRWkY2UXpBDt09liwYXbW2Xpc3BsYXlWY2l1ljoieU2ydmZjZlY2V5dWByBUUEkUjHJzC3l5mFwCgkljZjQjUwNz44YUWUTY1K0E00QWRkLT5K2MDtMCEJYwWnT6kNmKliwYXbW2R3Y3V0IjdiaWwYXbWj0jiaHR0cHM6Ly9zd2hmud2lwZG93cy5uZXQvNThlMzBkZytZDdmZS00ODVhLWJhMzEtNmRjMzVhZyRhZ2ZlY2l5mlk2Ml6MTY2NzY2M
```

IMPORTANT: Before you pass the bearer token you have generated in the header you have to add "Bearer" space there!

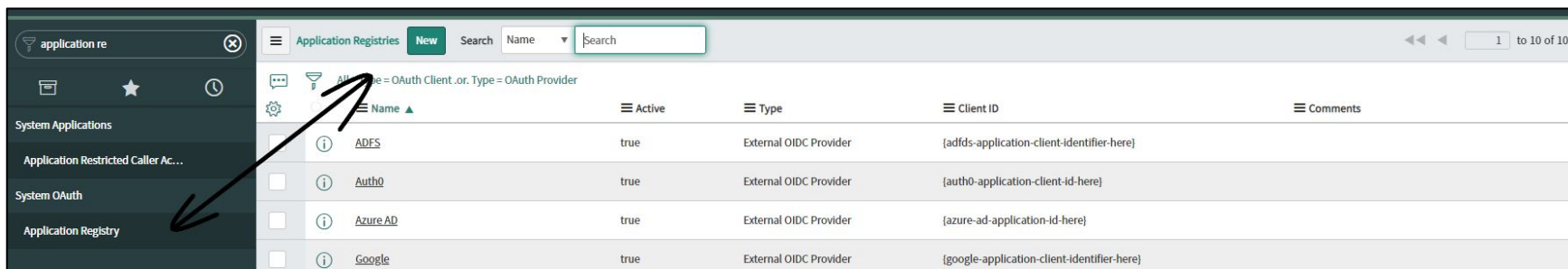


B) Flow Action

Generating bearer token through flow action gives more OOTB possibilities, but also requires a little different configuration, which can confuse at first glance. Some additional steps are required to input/process/output the whole authorization process. If the whole integration goes through flow from A to Z, you have to remember what should come into the flow action and what should come out, to successfully pass a token to the further actions.

To increase security, configuration in our case takes all parameters from the application registry table, where client secret value is encrypted, and by using input type **Password (2 Way Encrypted)** it's impossible to catch client secret password from any logs. On the next page you will find an example of how to do that.

First, open the application registry table (oauth_entity) and create a new record. There are few options to choose from, but the 3rd option fits our needs the most. So I recommend to choose “Connect to a third party OAuth Provider”. Then provide all mandatory fields. Notice that once you set up a client secret it will be encrypted as long as you toggle it to view it’s content, without a specific role to the table, and write access to the records. No one is able to see it’s value.



Name	Active	Type	Client ID	Comments
AD FS	true	External OIDC Provider	{adfs-application-client-identifier-here}	
Auth0	true	External OIDC Provider	{auth0-application-client-id-here}	
Azure AD	true	External OIDC Provider	{azure-ad-application-id-here}	
Google	true	External OIDC Provider	{google-application-client-identifier-here}	

What kind of OAuth application?

- Create an OAuth API endpoint for external clients
- Create an OAuth JWT API endpoint for external clients
- Connect to a third party OAuth Provider
- Configure an OIDC provider to verify ID tokens.
- Connect to an OAuth Provider (simplified)

Application Registries
Test Auth [OAuth Provider view]

OAuth provider details.

- Name: A unique name.
- Client ID: Client ID of application registered in third-party OAuth server.
- Client Secret: Client secret of application registered in third-party OAuth server.
- Refresh Token Lifespan: Time in seconds the Refresh Token will be valid.
- Authorization URL: OAuth Server's auth code flow endpoint. Required only for Authorization Code grant type.
- Token URL: OAuth Server's token endpoint.
- Token Revocation URL: OAuth Server's token revocation endpoint.
- Redirect URL: OAuth callback endpoint. Leave it empty for auto-generation.

* Name: Test Auth

* Client ID: [REDACTED]

* Client Secret: [REDACTED]

OAuth API Script: [REDACTED]

Logo URL: [REDACTED]

* Default Grant type: Authorization Code

* Refresh Token Lifespan: 8,640,000

Public Client: ☐

Comments: [REDACTED]

Application: Global

Accessible from: All application scopes

Active: ☒

Authorization URL: [https://login.microsoftonline.com/\[REDACTED\]/oauth2/authorize](https://login.microsoftonline.com/[REDACTED]/oauth2/authorize)

* Token URL: [https://login.microsoftonline.com/\[REDACTED\]/oauth2/token](https://login.microsoftonline.com/[REDACTED]/oauth2/token)

Token Revocation URL: [REDACTED]

Redirect URL: https://dev83866.service-now.com/oauth_redirect.do

Use mutual authentication: ☐

Send Credentials: In Request Body (Form URL-Encoded)

Now, open flow designer and create a new action, name it and prepare inputs that will be passed to the action. As an input, use strings with all body parameters needed to send a rest message.

Following the security - client secret input variable change to "Password (2 Way Encrypted)". This will make sure that logs will show only asterisks instead of the value. Remember also to look up first in the flow record with all details that were created in the previous step - you can use sys_id for instant matches.

ACTIONS

1 Look Up Application Registries Record

Action: Look Up Record

Table: Application Registries [oauth_entity]

Conditions: All of these conditions must be met

Sys ID is 408926231b8f7810553f65fde54bcb9b OR AND

or

New Criteria

Order by: Select a field a to z

If multiple records are found action: Return only the first record

Don't fail on error ☐

Delete Cancel Done

ACTIONS

1 Look Up Application Registries Record where (Sys ID is 408926231b8f7810553f65fde54bcb9b)

2 Azure Bearer Token

Action: Azure Bearer Token

* Tenant:

* Resource: https://graph.microsoft.com/

* Client_Id: 1 Application Registries Record Client ID

* Secret: 1 Application Registries Record Client Secret

Add a Stage

Delete Cancel Done

Trigger Run Daily

Run Start Time Date/Time

1- Look Up Record

Application Registries Record

Application Registries Ta... Table

Status Choice

Error Message String

2 - Azure Bearer Token

targetObject Object

3 - get users list

Item Count Integer

Page Count Integer

Azure Bearer Token
Properties
Test
Execution

Action Outline

- Inputs
 - 1 {REST} REST step REST
 - 2 {JSON} JSON Parser step JSON Parser
- Outputs

Action Input

Create Input

Label	Name	Type	Mandatory
:: Tenant	tenant	String	<input type="checkbox"/>
:: Resource	resource	String	<input type="checkbox"/>
:: Client_Id	client_id	String	<input type="checkbox"/>
:: Secret	secret	Password (2 Way...)	<input type="checkbox"/>

Once all inputs are ready to be passed on further, you should create a REST step from the action steps left in the menu to process received data.

Here in the REST Step, you do the most important part which is almost a reflection of what has been done before in HTTP Message.

Rest step gives a few more options which will allow you to diversify the solution and automate some data, except manually providing it to the action, for example by using credential & alias record. This example shows manually built rest API call, because we here decided to have this value hardcoded instead of jumping between tables and records.

To build Endpoint URL manually it's enough to choose “*Define connection inline*”, which lets you type base url and doesn't require the use of credential alias. If the integration goes through multiple tenants it's good to put the rest of the endpoint url within the “*Resource path*” where you can pass incoming tenants as a parameter from the data pill. Obviously everything can be done under the base URL field, but in my case some messages were throwing errors. I could not find out why it behaved like that. The HTTP Method is the POST like in the previous example. This is an interesting thing, because in comparison to the previously built Rest Message, Rest step within flow actions successfully process JSON payloads which makes building a Message much more efficient and friendly. If for some reason there needs to be url-specific, there is a possibility to change **request type** from “*Multipart*” (which supports JSON) to “*text*” and use values generated from postman code snippet.

1. REST step

Connection Details ⓘ

Connection: Define Connection Inline

Use MID: ☐

Credential Alias: Select Credential Alias

Base URL: https://login.microsoftonline.com/

Request Details ⓘ

Build Request: Manually

Resource Path: action ▶ Tenant /oauth2/token

HTTP Method: POST

Query Parameters

Name	Value

Headers

Name	Value

Request Content

Request Type: Multipart

Name	Type	Value
1 grant_type	application/json	client_credentials
2 client_id	application/json	action ▶ Client_id ×
3 client_secret	application/json	action ▶ Secret ×
4 resource	application/json	action ▶ Resource ×

Now, to have a well formatted JSON response, additional step called “*JSON Parser Step*” is a must. Otherwise the response will be in unreadable format and the response wouldn’t be passed further. This step is simple, but in my opinion not intuitive enough by first glance.

Remember that a source data always has a **response body** which contains needed values. Paste the whole payload you receive after getting a token and click “*Generate Target*”. This will generate a fully parsed, formatted and structured hierarchy of your JSON response. Thanks to this, the response will be shown on the data pill and can be transferred as an output of the action.

2. JSON Parser step

Source

source data: step ► REST step ► Response Body Generate Target

Structured Payload View ☐

```
1 { "token_type": "Bearer", "expires_in": "3599", "ext_expires_in": "3599", "expires_on": "1625746811", "not_before": "1625746811", "resource": "https://api.azurestack.com/management/2015-06-01/management/operations" }
```

Target

Clear All Exit Edit Mode

Label	Name	Type	Mandatory
root	root	Object	<input type="checkbox"/>
token_type	token_type	String	<input type="checkbox"/>
expires_in	expires_in	String	<input type="checkbox"/>
ext_expires_in	ext_expires_in	String	<input type="checkbox"/>
expires_on	expires_on	String	<input type="checkbox"/>
not_before	not_before	String	<input type="checkbox"/>
resource	resource	String	<input type="checkbox"/>

Data

Input Variables

- Tenant String
- Resource String
- Client_id String
- Secret Password (2 Way...)

REST step

- Error Message String
- Response Body String
- Error Code String
- Response Headers String
- Status Code String

JSON Parser step

- root Object

At last, the only thing left is to create a new object-type variable and pass the generated root target from the previous action step. This will generate a full structure of response and allows passing output to other actions to authenticate next actions.

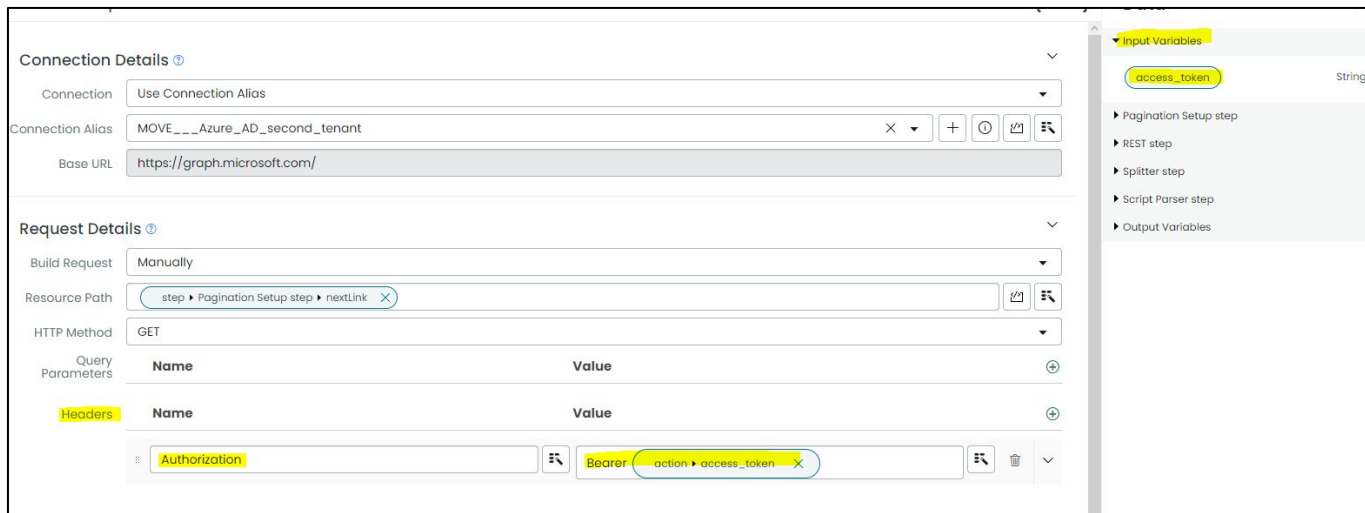
Action Output [Edit Outputs](#)

Label	Value
▼ targetObject	step > JSON Parser step > root
token_type	
expires_in	
ext_expires_in	
expires_on	
not_before	
resource	
access_token	

Data >

- ▼ Input Variables
 - Tenant String
 - Resource String
 - Client_Id String
 - Secret Password (2 Way...
- ▼ REST step
 - Error Message String
 - Response Body String
 - Error Code String
 - Response Headers String
 - Status Code String
- ▼ JSON Parser step
 - root Object
- ▼ Output Variables
 - targetObject Object

How to pass action output further within the flow? Once the output and whole action is saved, published and selected in the flow, it should be visible onto the data pill panel and by drag & drop. Simply pass a token to the other action's input field and then from the input, pass it directly to the Rest message header.



CHAPTER 4

Security

Some people may ask if doing authentication this way is secure enough if you are dealing with internal customer data and the answer to that question is - No, it's not secured **enough**, but there is no security that you cannot break. If we take a look on the presented solution, there are actually some levels of security but mainly done by the Microsoft Azure environment:

1. **Bearer token and the default short lifespan (1h)**
2. **Authentication which requires at least 3 unique parameters to authenticate (client id, tenant id, client secret)**
3. **Encrypted token**
4. **Application and delegated access types which depends of the needs and accessibility can be restricted**

There is a high chance that even if someone gets the whole bearer token key, it can already be expired and it's working only for a specific session. Token changes every time a client secret is an equivalent of giving someone's password - we should let such things be easily accessed.

If I would have to choose a more secure solution from those two, only by trusting OOTB functionalities, I would recommend getting token by using flow. Depending on the requirements, it's possible to build using Rest HTTP Messages. Does this means that we are out of options for this one? Absolutely not. Obviously it's harder to manage, develop and probably it will require some additional scripts. If we as developers want to create a secure solution, we can use for example the well working GlideEncrypter.

https://docs.servicenow.com/bundle/rome-application-development/page/app-store/dev_portal/API_reference/GlideEncrypter/concept/GlideEncrypterAPI.html?cshalt=yes

The advantage of using GlideEncrypter is that instead of hardcoding values within HTTP Messages, there is a way to run API calls directly from the script, and include and pass value to the body of our message in an encrypted way.

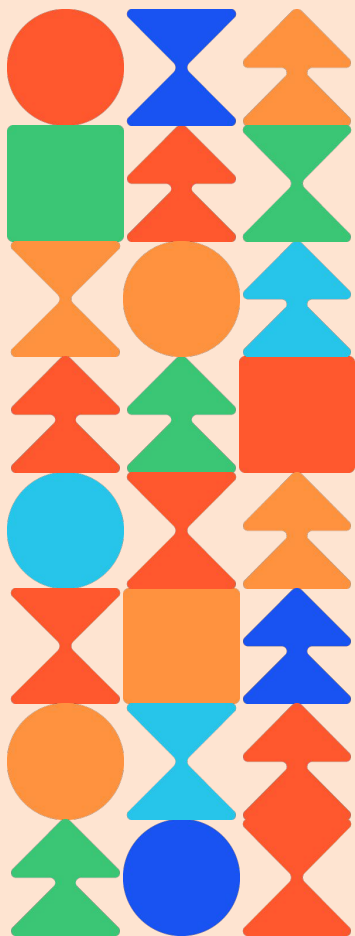
From the ServiceNow perspective, the most susceptible data in such authentication type is to leave parameters unprotected which has the biggest impact on the Azure security. So it's important to additionally protect, hide and encrypt sensitive data. What we can do as developers and from a ServiceNow perspective, is to make access to this data as hard as possible. Not only rely on the access controls and role-based structure, because even without admin role, some tables, logs and flow executions are at least readable. This is causing a high risk of giving access to for example the client secret to undesirable persons. For some it might be gibberish, but for some a potential backdoor to strike.

One of the security improvements has been provided by using application registry to **at least** hide client secret password and then make 2 Way encrypted input within the flow, which is returning encrypted value.

VARIABLE NAME	TYPE	CONFIGURATION	RUNTIME VALUE
Tenant	String	63a49930-d3e8-4d14-9e8c-f016576f5f08	63a49930-d3e8-4d14-9e8c-f016576f5f08
Resource	String	https://graph.microsoft.com/	https://graph.microsoft.com/
Client_Id	String	1 ▶ Application Registries Record ▶ Client ID	437309c6-1c4e-4429-868d-a1f6f23b26c5
Secret	Password (2 Way Encrypted)	1 ▶ Application Registries Record ▶ Client Secret	*****

Summary

As it turns out, Microsoft and ServiceNow can deliver massive and well-working solutions, but even such big players can't predict all possible cases and sometimes some of them are not worth developing. In such cases, we as developers are obligated to solve these challenges and leave official documentation and standards, to rather create our own solutions based on our experience and different best practices built ourselves. Presented solution, if continuously developed, may someday be considered as a real solution, even for bigger players on the market. It is also good to have a backup if traditional solutions are failing. With some practice it can also give as an advantage by delivering your own secured solution, which definitely will make you visible for others in the global market.



Want to learn more?

Contact the Cloud People

info@thecloudpeople.com

www.thecloudpeople.com

The Cloud People is a certified ServiceNow and Google Cloud Platform Partner, and authorized Google Workspace (earlier G Suite) Reseller. We help and guide organizations transform their business to the cloud and gain and utilize the competitive advantages from one of the best cloud platform solutions on the market today.