Demystifying Azure AD Service Principals

Published: 2019-07-16 · Archived: 2026-04-05 13:28:55 UTC

Anyone who's worked with Azure for a bit has encountered the need to create a service principal. If you are an IT Ops person, you probably equate an SP with a service account in local Active Directory. If you're more of an application developer, then you may have created an SP as part of your application in Azure, because you want to give that application permissions to Azure resources. The purpose of this post is to tease apart what service principals are, how they interact with application objects, and all the myriad ways to create an SP on Azure.

Azure Active Directory Applications

The first thing you need to understand when it comes to service principals is that they cannot exist without an application object. If that sounds totally odd, you aren't wrong. The service principal construct came from a need to grant an Azure based application permissions in Azure Active Directory. Since access to resources in Azure is governed by Azure Active Directory, creating an SP for an application in Azure also enabled the scenario where the application was granted access to Azure resources at the management level. That is all very useful in the context of creating applications in Azure. If you wanted an account in Azure AD to use for automation or to power a service, then you could use the same construct. Instead of creating a separate object type in Azure AD, Microsoft decided to roll forward with an application object that has a service principal. For the purposes of using an SP like a service account, the application it creates as part of the process sits unused and misunderstood.

To make things even more confusing, a single application object can have multiple service principals across different Azure AD tenants. When an application object is registered with the home tenant, an SP is also created in that Azure AD tenant. If the application being developed is a single-tenant application, that's the only SP needed. But if the application is meant to be multi-tenant - by which I mean multiple Azure AD tenants - then it will have an SP created in each tenant that uses it. The consent process of enabling an application for your Azure AD tenant includes creating and granting permissions to that application object in the form of an SP in your tenant.

Of course, if your whole goal was to use a service principal to do some automation, then you don't care about any of this nonsense. You just want to create an SP and be done with it.

Service Principals

As an IT Ops person trying to get some work done, you don't care about the application object. You probably don't want to deal with the application object. You just want to create an SP. There is NO way to do this without also creating an application object. That's the decision that Microsoft made, and it seems to be sticking with it. What that means is that depending on which tool you use to create a service principal, you may need to create an application object first. For instance, the portal requires that you create the application object first and doesn't even mention the service principal as a construct. If you are using a different tool, it may automatically create that application object for you. For instance, the Azure CLI allows you to directly create an SP, and it will take care of

creating that application object for you in the background. How helpful! The downside is that there are so many different tools to use with Azure, and they ALL seem to have a different workflow. You can create an SP by using:

- The Azure Portal
- The AzureAD PowerShell Module
- The Az PowerShell Module
- The AzureRM PowerShell Module
- The Azure CLI
- Other Azure SDKs
- The Azure AD API
- The Microsoft Graph API

Holy cow! What's a poor IT Ops person to do? Let's break it down with what will likely be the most common ways you will create a Service Principal.

Creating Service Principals

Microsoft Graph API

The one thing that all of these tools have in common is that they are all referencing one of two APIs to perform their operations. Microsoft is in the process of deprecating the Azure AD API in favor of the Microsoft Graph API. All the other methods are using some kind of SDK to interact with one of these two APIs. If you're curious about the Azure AD API, the relevant sections for the application and service principal objects can be found in the entity and complex types area of the docs. The Microsoft Graph API docs seem to be a little better organized, and you can find information on applications and service principals. Funny thing that I noticed, there is no create function for the service principal object. The resource appears to be implicitly created when an application is registered with a tenant.

The Azure Portal

The experience for registering an application and creating a service principal has changed recently. I'd like to say it makes more sense now, but I would be lying. As far as I can tell it's more confusing with check boxes that don't fully explain what they want you to do. Regardless, if you want to create a service principal through the portal, just follow these directions.

The AzureAD PowerShell Module

If you don't already have the AzureAD PowerShell module, you can install it by running `Install-Module AzureAD -Force`. Then run the following commands:

```
 Connect-AzureAD -TenantId "YOUR_TENANT_ID"
$myApp = New-AzureADApplication -DisplayName "AzureAD Module App" -IdentifierUris "https://azureadmoduleapp"
$mySP = New-AzureADServicePrincipal -AppId $myApp.AppId
```

Obviously, the AzureAD module does not take care of creating the application object for you. You have to do that first and then create the SP. The commands above will get you a service principal, but without any type of credentials to login. If you want a password associated with the service principal, then you can run the following:

```
 $spCredParameters = @{
  StartDate = [DateTime]::UtcNow
  EndDate = [DateTime]::UtcNow.AddYears(1)
  Value = 'MySuperAwesomePasswordIs3373'
  ObjectId = $mySP.ObjectID
}

New-AzureADServicePrincipalPasswordCredential @spCredParameters
```

Now you have a service principal that you can assign roles and permissions to.

The Az PowerShell Module

There's a new Azure PowerShell module on the block. The Az module is replacing the original AzureRM module. The reason? Partly, Microsoft just wanted to shorten the commands by five letters. They also wanted to rewrite the module to take advantage of new functionality in PowerShell and in Azure and get rid of some of the old commands that maybe weren't following best practices. Regardless, this is the module you'll be using to do things with Azure going forward. If you're currently running AzureRM, **beware** here there be dragons. You need to completely remove AzureRM first, or install PowerShell 6 and run the Az module in PowerShell 6 context instead. To create a service principal with the Az module, run the following commands:

```
 Connect-AzAccount
$mySP = New-AzADServicePrincipal
```

That's it. The good news is that the command creates the application in the background for you. And that is pretty much where the good news ends. In this case, the command creates a service principal with a display name that starts *azure-powershell-* and appends the current date and time. It also gives it a secret of the type *System.Security.SecureString* which is not particularly useful. It is possible to decrypt it, but I would recommend setting a password credential manually like we did in the AzureAD module example. You might think that there is a command like `New-AzureADServicePrincipalPasswordCredential` in the Az module, and you would be partly correct. There is the `New-AzADSpCredential` command, but that only allows you to add a certificate type and not a password. No idea why that choice was made. Maybe because Microsoft hates passwords?

Before we get into the process for creating a password based credential, which I assure you is non-intuitive and annoying, I would first like to point out something that really annoys me. The service principal object from the AzureAD module isn't the same type as the service principal object from the Az module. If you run `Get-Member` on the SP object from the AzureAD module you get the *TypeName* `Microsoft.Open.AzureAD.Model.ServicePrincipal`, whereas with the Az module you get the *TypeName* `Microsoft.Azure.Commands.Resources.Models.Authorization.PSADServicePrincipalWrapper`. The properties

exposed in each object type also differ. The AzureAD module exposes 25 different properties, and the Az module exposes only 7. If I might present a table for comparison:

Az	**AzureAD**
ApplicationId	AppId
DisplayName	DisplayName
Id	ObjectId
ObjectType	ObjectType
Secret	N/A
ServicePrincipalNames	ServicePrincipalNames
Type	N/A

Right off the bat, the *ApplicationId* is named differently across the two objects. So is the *ObjectId*. And it's not even consistent in its inconsistency. The Az modules uses the longer *ApplicationId* property and the shorter *Id* property. Then there is the *Secret* property, which is really just the value stored in one of the keys in the *PasswordCredential* property. A service principal can have multiple passwords - aka secrets - which are held in an array in the *PasswordCredential* property. The *PasswordCredential* property is an object type of `Microsoft.Open.AzureAD.Model.PasswordCredential`. There is a separate *KeyCredentials* property and object type that houses certificate based authentication. The `New-AzADSpCredential` command takes a cert value and adds it to the service principal in a property that is not even exposed by the Az implementation of the service principal type. That means you need to run the Get-AzADSpCredential command to get the value back. In order to create a password based credential, you have to create *PasswordCredential* object directly like this:

```
 $credProps = @{
   StartDate = Get-Date
   EndDate = (Get-Date -Year 2024)
   KeyId = (New-Guid).ToString()
   Value = 'MySuperAwesomePasswordIs3373'
}
$credentials = New-Object Microsoft.Azure.Graph.RBAC.Models.PasswordCredential -Property $credProps

Set-AzADServicePrincipal -ObjectId $mysp.Id -PasswordCredential $credentials
```

That's if you want to configure a password after creation of the SP. If you wanted to set the password while creating the service principal, you have to create a **completely different object type**. When you run `New-AzAdServicePrincipal` with the *PasswordCredential* parameter, the command is expecting an object of type `Microsoft.Azure.Commands.ActiveDirectory.PSADPasswordCredential`. When you run `Set-AzAdServicePrincipal` with the *PasswordCredential* parameter, the command is expecting an object of type

`Microsoft.Azure.Graph.RBAC.Models.PasswordCredential`. And it will not do an implicit conversion for you! Instead you have to do the following:

```
 $credProps = @{
    StartDate = Get-Date
    EndDate = (Get-Date -Year 2024)
    Password = 'MySuperAwesomePasswordIs3373'
}
$credentials = New-Object Microsoft.Azure.Commands.ActiveDirectory.PSADPasswordCredential -Property $credProps

$sp = New-AzAdServicePrincipal -DisplayName "WhateverItDoesntMatter" -PasswordCredential $credentials
```

You may have also noticed that the two different object types have different arguments. One expects a *KeyId* and *Value* and the other expects a *Password* argument only.

My advice is this. **Don't use the Az module for managing Azure AD resources.** It's a hot mess.

The Azure CLI

Unlike the PowerShell modules, the Azure CLI is written in Python. The process for creating a service principal is simple. Run the following command:

```
 az ad sp create-for-rbac -n "MySpCLI"
```

The command will create the application object in the background for you. And the output will include all the information you need to use the service principal, including the password in clear text.

```
 {
  "appId": "d9bdde72-73eb-4bc2-8dbf-0855c0d19820",
  "displayName": "MySpCLI",
  "name": "http://MySpCLI",
  "password": "7778ef85-2c5a-4f8d-81d5-88389c9a4feb",
  "tenant": "f06624a8-558d-45ab-8a87-a88094a3995d"
}
```

That's all there is to it. Super easy and simple.

Conclusion

I started this post hoping to demystify the application and service principal relationship and shed some light on how to use different tools to accomplish the same goal. At the end, I may have made things a little more confusing. But that simply reflects the confusing nature of service principal kludge. My advice would be to use the Azure CLI to create a service principal. The command is simple. You can run it from the Cloud Shell if you don't have the Azure CLI locally. It is faster than using the portal, and easier than using PowerShell. Additionally,

many resources in Azure now have the ability to use Managed Service Idenities (MSIs) to access other Azure resources. You still need service principals for some use cases, but I would highly recommend checking to see if an MSI can meet your requirements.
