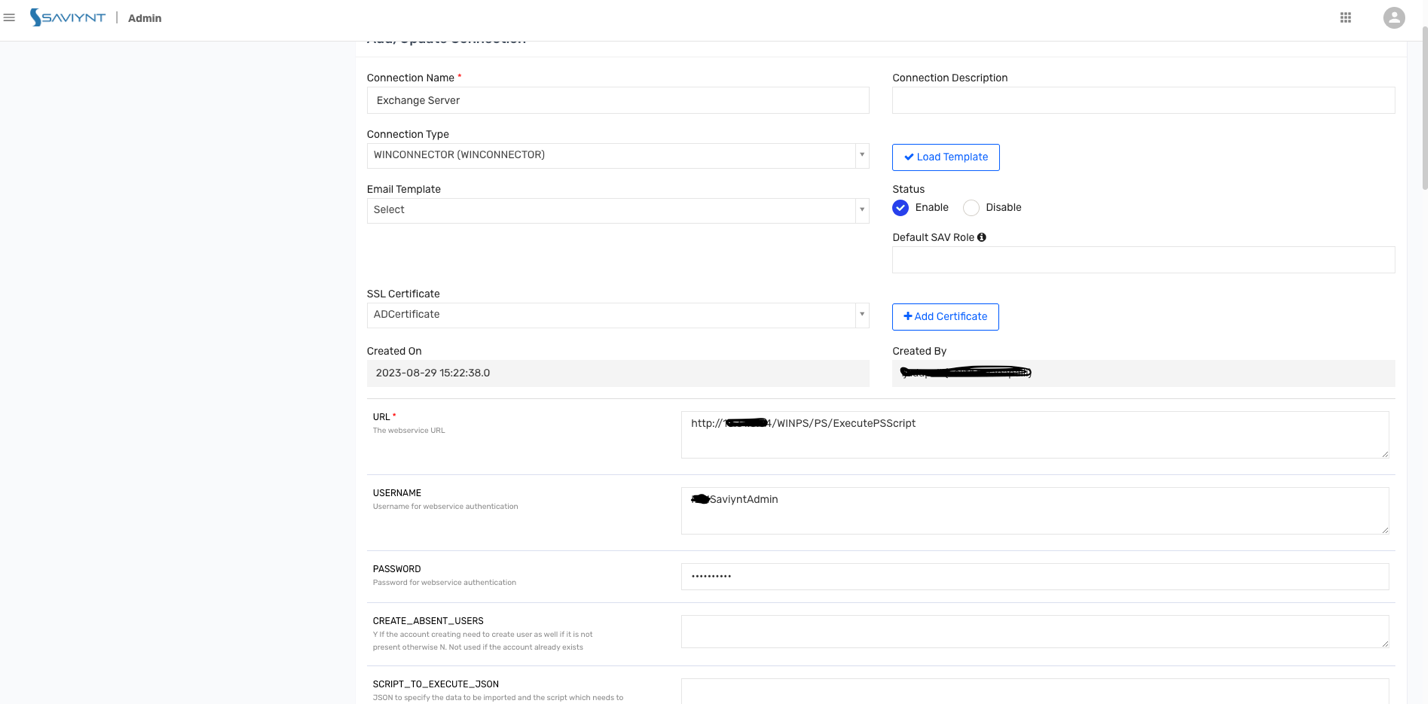
WinConnector (Ps script execution)

Introduction:  
Saviynt EIC offers capability to provision mailboxes provision mailboxes for on-premise Active Directory (AD) users on Microsoft Exchange Server or provision license for Azure AD users on Office 365  
  
  
How it works:  
  
Saviynt offers a connector module which can be configured to make a connection with the application.  
  
  
Prerequisites:   
  
Network connectivity should be established before we can begin operations like recon and provisioning. The network needs to be whitelisted, and firewall should allow the network to flow from Saviynt to EIC and vice versa. The Credentials for the Rest API are valid.  
  
Integration Steps:  
  
  
1) Create a connection for connector type as WinConnector .It will load the template.  
  


* **2**) Provide CreateAccountJson: it will be the command to create an object.  
    
  {"CREATEACC":["Script=\$Ruser = 'tdm\\\\IDMSaviyntAdmin';\$Rpasswd = ConvertTo-SecureString 'T3sT@d!2023#' -AsPlainText -Force;\$Rcred = New-Object System.Management.Automation.PSCredential (\$Ruser, \$Rpasswd);try { \$RSession = New-PSSession -ConfigurationName Microsoft.Exchange -ConnectionUri https://domain.com/PowerShell/ -Authentication Basic -SessionOption (New-PSSessionOption -SkipCACheck -SkipCNCheck -SkipRevocationCheck) -Credential \$Rcred; \$global:output = ''; Invoke-Command -session \$RSession -ScriptBlock { Enable-Mailbox '${task.accountName}' -Alias '${if (user.email != null) {user.email.substring(0,user.email.indexOf('@'))} else {'null-email@testdomain.com'} }' }; Remove-PSSession \$RSession; } catch { Write-Error \$\_; }"],"SuccessResponses": "IsValid:True"}  
    
  Lets understand the json in details  
    
  **PowerShell Script (Script)**: The PowerShell script defines the logic for establishing a remote session, enabling a mailbox, and handling errors.
  + **Credentials Setup**:
    - \$Ruser: The username is set as tdm\\SaviyntAdmin.
    - \$Rpasswd: Password Iamsaviynt is converted to a secure string.
    - \$Rcred: Creates a credential object for authentication.
  + **Remote Session**:
    - **Connection Setup**: A remote PowerShell session (\$RSession) is established with Microsoft Exchange via New-PSSession.
    - **Connection URI**: https://tdm-ex01.testdomain.com/PowerShell/, specifying the endpoint for Exchange.
    - **Session Options**: SkipCACheck, SkipCNCheck, and SkipRevocationCheck are used to bypass certificate checks.
    - **Credential Parameter**: Passes in the credential object \$Rcred for authentication.
  + **Enable Mailbox Command**:
    - **Command Execution**: Invoke-Command enables a mailbox for the specified account using Enable-Mailbox.
    - **Dynamic Alias**: The alias is dynamically generated based on the user's email:
      * If user.email is available, it takes the substring up to the @ symbol as the alias.
      * If user.email is null, a placeholder null-email@testdomain.com is used.
  + **Session Cleanup**: Removes the session with Remove-PSSession to prevent lingering sessions.
  + **Error Handling**: Uses catch to capture and log errors.
* **Success Response**:
  + "IsValid:True" signifies a successful operation when mailbox creation is completed without errors.

3) Provide Grant access Json and Remove Access Json to provision and deprovision  
  
A screenshot of a computer

Description automatically generated

Remove access Json: {"memberOf":["Script=\$pw = convertto-securestring '' -asplaintext -force;\$mycred = new-object -typename System.Management.Automation.PSCredential -argumentlist '\\\\',\$pw;Invoke-Command -ComputerName '' -ScriptBlock {Remove-ADGroupMember -Identity '${task.entitlement\_valueKey.entitlementID}' -Members '${task.accountName}' -Confirm:\$false}"]}  
  
  
Note : use the Username , password and computername accordingly  
  
  
  
**PowerShell Script (Script)**: The script uses the Remove-ADGroupMember cmdlet within Invoke-Command to remove a specified user from a specified AD group.

* **Credential Setup**:
  + \$pw defines an empty password as a secure string, which is typically not used for secure operations unless the environment assumes implicit trust.
  + \$mycred: Sets up a credential object using the empty password, but without a username (denoted as \\). This could imply that credentials are handled outside this script, perhaps through a secure session or implicit trust settings.
* **Command Execution**:
  + Invoke-Command initiates a remote session to execute the Remove-ADGroupMember command on a target AD server.
  + **Computer Name**: The ComputerName '' field is currently empty and would need to be populated with the actual server name.
  + **Remove-ADGroupMember Cmdlet**: This cmdlet removes a user from a specific AD group.
    - **Group Identity**: The group is identified by ${task.entitlement\_valueKey.entitlementID}, dynamically retrieving the group ID.
    - **Member to Remove**: The ${task.accountName} inserts the account name of the user to be removed from the group.
    - **Confirmation Flag**: -Confirm:\$false skips confirmation prompts, making the removal automatic.

Add access json:   
  
{"memberOf":["Script=\$pw = convertto-securestring '' -asplaintext -force;\$mycred = new-object -typename System.Management.Automation.PSCredential -argumentlist '\\\\',\$pw;Invoke-Command -ComputerName '' -ScriptBlock {Add-ADGroupMember -Identity '${task.entitlement\_valueKey.entitlementID}' -Members '${task.accountName}'}"]}  
  
**PowerShell Script (Script)**: This part of the JSON includes a script that remotely adds a specified user to a specified AD group.

* **Credential Setup**:
  + \$pw defines an empty password as a secure string. This placeholder might imply that credentials are managed outside the script or that the environment relies on implicit trust.
  + \$mycred: Creates a credential object using the empty password, which is unusual in typical setups but might work in trusted domains.
* **Command Execution**:
  + Invoke-Command: Initiates a remote PowerShell session to execute the Add-ADGroupMember command on the specified AD server.
  + **Computer Name**: The placeholder ComputerName '' is left empty here and needs to be replaced with the actual name of the AD server.
  + **Add-ADGroupMember Cmdlet**:
    - **Group Identity**: The group is identified dynamically by ${task.entitlement\_valueKey.entitlementID}, allowing for the selection of specific groups at runtime.
    - **Member to Add**: ${task.accountName} dynamically inserts the account name of the user who needs to be added to the group.

**Troubleshooting:  
One of the common Issues with WINPS connector is during the configuration settings on the App side  
  
Resolution: Follow the documentation and provide all the permissions accordingly mentioned in the documentation.**