PowerShell Script – CloudFilterFromBackup.ps1

# Description

This PowerShell script creates filters in Jira Cloud based on provided CSV files.

# Script

Save the following script as CloudFilterFromBackup.ps1.

|  |
| --- |
| #Requires -version 7  <#  .SYNOPSIS  Filter reconstruction from Jira Cloud backup.    .PARAMETER Domain  Jira cloud domain, e.g. kcwong.atlassian.net    .PARAMETER Email  Email address.    .PARAMETER Token  API token.    .PARAMETER Protocol  https or http. Default https.    .PARAMETER FilterCsv  CSV file containing filters to reconstruct.  Must contain the following columns:  1. id  A reference ID between CSV files, not filter Id in Jira.  2. name  Name of filter.  3. jql  JQL.  4. owner  Account ID of filter owner.  .PARAMETER PermissionCsv  CSV file containing filter permissions.  Must contain the following columns:  1. id  A reference ID between CSV files, not filter Id in Jira.  2. type  One of the following: loggedin, project, group, user  3. rights  One of the following: 1 (View), 2 (Edit), 3 (View and Edit). 2 is never used in Jira.  4. param1  When type is:  project - project id. Use /rest/api/latest/project/[ProjectKey] to retrieve project id.  group - group NAME. Group names can be found in https://admin.atlassian.com/  user - account id. Account ids can be found in https://admin.atlassian.com/  5. param2  Project role id when type is project. Null for all roles.  Use /rest/api/latest/role to get full list of project roles.    This CSV file is in a many-to-one relationship with FilterCsv.    Note that if you specify loggedin for a rights, you cannot have other types in the same rights.  i.e. If you specify loggedin for view, you cannot have project/user/group for view. But you can have project/user/group for Edit.    .PARAMETER PauseAction  Switch. If specified, pause after each modification action.    .PARAMETER PauseFilter  Switch. If specified, pause after processing each filter CSV record.  #>  <#  Algorithm    1. Take backup CSVs for filter and share permission input (preprocess filter reference id to name).  2. Take list of filter ids (in backup) to be reconstructed.  3. For each filter in current batch:  a. Check if filter id exists (shouldn't be, as supplied ids should be missing).  i. If exist, log as "already exist".  ii. If option allows update, update JQL.  b. If filter reference exists, for each reference, create dummy filter.  c. Create filter based on backup data (name, JQL, share permissions).  d. Change owner to target user.  e. Delete dummy filter created.  #>  Param(  [Parameter(Mandatory)]  [string] $Domain,    [string] $Protocol = 'https',    [Parameter(Mandatory)]  [string] $Email,    [string] $Token = '',    [Parameter(Mandatory)]  [string] $FilterCsv,    [Parameter(Mandatory)]  [string] $PermissionCsv,    [Parameter()]  [string] $DataCsv,    [Parameter()]  [switch] $PauseAction,    [Parameter()]  [switch] $PauseFilter  )  class RestException : Exception {  RestException($Message) : base($Message) {  }  }  function GetAuthHeader {  Param (  [string] $Email,  [string] $Token  )  [hashtable] $Headers = @{  "Content-Type" = "application/json";  "Accept" = "application/json";  }  $Auth = [Convert]::ToBase64String([Text.Encoding]::ASCII.GetBytes($Email + ":" + $Token))  $Headers.Authorization = "Basic " + $Auth  $Headers  }  # Call Invoke-WebRequest without throwing exception on 4xx/5xx  function WebRequest {  Param (  [string] $Uri,  [string] $Method,  [hashtable] $Headers,  [object] $Body  )  $Response = $null  try {  $script:ProgressPreference = 'SilentlyContinue' # Subsequent calls do not display UI.  $Response = Invoke-WebRequest -SkipHttpErrorCheck -Method $Method -Header $Headers -Uri $Uri -Body $Body  } finally {  $script:ProgressPreference = 'Continue' # Subsequent calls do display UI.  }  $Response  }  function GetFilterId {  Param (  [hashtable] $Headers,  [boolean] $OverrideSharePermissions,  [string] $Name,  [string] $Owner  )  $Result = $Null  $Json = $null  $Body = @{  'overrideSharePermissions' = $OverrideSharePermissions;  'expand' = 'owner,jql,sharePermissions,editPermissions';  'filterName' = '"' + $Name + '"';  'accountId' = $Owner;  'startAt' = 0;  }  $Uri = $Protocol + '://' + $Domain + '/rest/api/latest/filter/search'  $Response = WebRequest $Uri 'GET' $Headers $Body  $Json = $Response.Content | ConvertFrom-Json  if ($Json.values.Count -eq 1) {  $Result = $Json.values[0].id  }  $Result  }  function GetFilterDependencies {  Param(  [string] $Jql  )  $Result = [System.Collections.ArrayList]::new()  $MatchInfo = Select-String '\s\*filter\s\*=\s\*"([^"]+)"\s\*' -Input $Jql -AllMatches  foreach ($Match in $MatchInfo.Matches) {  [void] $Result.Add($Match.Groups[1])  }  $Result  }  function CreateFilter {  Param (  [hashtable] $Header,  [string] $Name,  [string] $Jql,  [PSObject] $Permissions  )  $Result = $Null  $Json = $null  $Body = @{  'name' = $Name;  'jql' = $Jql;  }  if ($Permissions) {  $Body['sharePermissions'] = $Permissions.sharePermissions  $Body['editPermissions'] = $Permissions.editPermissions  }  $Uri = $Protocol + '://' + $Domain + '/rest/api/latest/filter'  $Response = WebRequest $Uri 'POST' $Header ($Body | ConvertTo-Json -Depth 100)  if ($Response.StatusCode -eq 200) {  $Json = $Response.Content | ConvertFrom-Json  $Result = $Json.id  } else {  throw $Response.Content  }  $Result  }  function DeleteFilter {  Param (  [hashtable] $Header,  [string] $Id  )  $Body = @{  }  $Uri = $Protocol + '://' + $Domain + "/rest/api/latest/filter/${Id}"  $Response = WebRequest $Uri 'DELETE' $Header $Body  if ($Response.StatusCode -ne 204) {  throw $Response.Content  }  }  function ChangeFilterOwner {  Param (  [hashtable] $Header,  [string] $Id,  [string] $Owner  )  $Result = $False  $Body = @{  'accountId' = $Owner;  }  $Uri = $Protocol + '://' + $Domain + "/rest/api/latest/filter/${Id}/owner"  $Response = WebRequest $Uri 'PUT' $Header ($Body | ConvertTo-Json -Depth 100)  if ($Response.StatusCode -ne 204) {  throw $Response.Content  }  }  # Convert SearchRequest into a map. Key is filter id, value is object providing name, jql and owner  function ReadSearchRequest {  Param (  [string] $Path  )  $Result = [ordered]@{}  $List = Import-Csv -Path $Path  foreach ($Item in $List) {  $Result[$Item.id] = $Item  }  $Result  }  # Convert SharePermission data into a map. Key is filter id, value is filter update payload for the filter  function ReadSharePermission {  Param (  [string] $Path  )  $Result = @{}  $List = Import-Csv -Path $Path  foreach ($Item in $List) {  $id = $Item.id  $Payload = $null  if (-not $Result[$id]) {  $Payload = @{  sharePermissions = [System.Collections.ArrayList]::new()  editPermissions = [System.Collections.ArrayList]::new()  }  } else {  $Payload = $Result[$id]  }  # Add item's data to payload  $Data = @{}  switch ($Item.type) {  'group' {  $Data.type = 'group'  $Data.group = @{  'name' = $Item.param1  }  break  }  'project' {  $Data.type = 'project'  $Data.project = @{  'id' = $Item.param1  }  if ($Item.param2) {  $Data.type = 'projectRole'  $Data.role = @{  'id' = $Item.param2  }  }  break  }  'user' {  $Data.type = 'user'  $Data.user = @{  'accountId' = $Item.param1  }  break  }  'loggedin' {  $Data.type = 'authenticated' # Must be authenticated in request, not loggedin as they give you  break  }  'global' {  # Global no longer supported, permission dropped  break  }  }  if ($Data.Count -ne 0) {  switch ($Item.Rights) {  '1' {  [void] $Payload.sharePermissions.Add($Data)  break;  }  '2' {  # Fall-through  }  '3' {  [void] $Payload.editPermissions.Add($Data)  break;  }  }  }  $Result[$Id] = $Payload  }  $Result  }  # Helper. Append message to $Data['Messages'] (with newline) and write to console (with indent).  function Log {  Param (  [System.Collections.Specialized.OrderedDictionary] $Data,  [string] $Msg  )  Write-Host "`t${Msg}"  $Data['Messages'] += $Msg + "`n"  $Data  }  function PausePrompt {  Param (  [string] $Msg  )  if ($Msg) {  Write-Host "`t${Msg}"  }  Write-Host "`t`tEnter to continue / Ctrl-C to exit" -NoNewline  $Null = $Host.UI.ReadLine()  }  # Main body  if (-not $Token) {  $pwd = Read-Host "Enter API token" -AsSecureString  $Token = [Runtime.InteropServices.Marshal]::PtrToStringAuto([Runtime.InteropServices.Marshal]::SecureStringToBSTR($pwd))  }  $AuthHeader = GetAuthHeader $Email $Token  $DataCsv = 'CloudFilterFromBackup.' + (Get-Date -Format 'yyyyMMddHHmmss') + '.csv'  $FilterData = ReadSearchRequest $FilterCsv  $PermissionData = ReadSharePermission $PermissionCsv  foreach ($Filter in $FilterData.GetEnumerator()) {  $BackupId = $Filter.Value.id  $Name = $Filter.Value.name  $Jql = $Filter.Value.jql  $Owner = $Filter.Value.owner  $Permission = $PermissionData[$BackupId]  $View = 'None'  $Edit = 'None'  if ($Permission) {  $View = ($Permission.sharePermissions | ConvertTo-Json -Compress -Depth 100)  $Edit = ($Permission.editPermissions | ConvertTo-Json -Compress -Depth 100)  }  $Data = [ordered]@{}  $Data['BackupId'] = $BackupId  $Data['CurrentId'] = ''  $Data['Name'] = $Name  $Data['Jql'] = $Jql  $Data['Owner'] = $Owner  $Data['Create'] = ''  $Data['ChangeOwner'] = ''  $Data['Messages'] = ''  Write-Host "Processing filter Id: ${BackupId} Name: ${Name} JQL: ${Jql} Owner: ${Owner} View: ${View} Edit: ${Edit}"  $Id = GetFilterId $AuthHeader $True $Name $Owner  if (-not $Id) {  $Data = Log $Data "Filter does not exist, recreating..."  if ($PauseAction) { PausePrompt }  $Error = $False  $DummyFilterList = [System.Collections.ArrayList]::new()  $DependencyList = GetFilterDependencies $Jql  foreach ($Dependency in $DependencyList) {  $Data = Log $Data "Filter depends on filter ${Dependency}"  if (-not (GetFilterId $AuthHeader $False $Dependency)) {  $Data = Log $Data "Dependency filter ${Dependency} is inaccessible, creating dummy..."  if ($PauseAction) { PausePrompt }  try {  $DummyId = CreateFilter $AuthHeader $Dependency 'order by created asc'  [void] $DummyFilterList.Add($DummyId)  $Data = Log $Data "Dependency filter ${Dependency} created: ${DummyId}"  if ($PauseAction) { PausePrompt }  } catch {  $Data = Log $Data ("Failed to create dummy filter ${Dependency}" + $\_.ToString())  if ($PauseAction) { PausePrompt }  $Error = $True  break  }  } else {  $Data = Log $Data "Dependency filter ${Dependency} exists and is accessible"  if ($PauseAction) { PausePrompt }  }  }  if (-not $Error) {  try {  $Data = Log $Data "Creating filter..."  $Id = CreateFilter $AuthHeader $Name $Jql $Permission  $Data = Log $Data "Created filter: ${Id}"  $Data['CurrentId'] = $Id  $Data['Create'] = 'Success'  if ($PauseAction) { PausePrompt }  try {  $Data = Log $Data "Changing owner..."  ChangeFilterOwner $AuthHeader $Id $Owner  $Data = Log $Data "Owner changed"  $Data['ChangeOwner'] = 'Success'  } catch {  $Data = Log $Data ("Failed to change owner: " + $\_.ToString())  $Data['ChangeOwner'] = 'Failed'  $Data['Messages'] += $\_.ToString() + ';'  }  if ($PauseAction) { PausePrompt }  } catch {  $Data = Log $Data ("Failed to create filter: " + $\_.ToString())  $Data['Create'] = 'Failed'  $Data['Messages'] += $\_.ToString() + ';'  if ($PauseAction) { PausePrompt }  }  }  foreach ($DummyId in $DummyFilterList) {  $Data = Log $Data "Deleting dummy filter: ${DummyId}"  if ($PauseAction) { PausePrompt }  try {  DeleteFilter $AuthHeader $DummyId  $Data = Log $Data "Deleted dummy filter: ${DummyId}"  } catch {  $Data = Log $Data ("Failed to delete dummy filter ${DummyId}: " + $\_.ToString())  }  if ($PauseAction) { PausePrompt }  }  } else {  $Data = Log $Data "Filter already exists"  $Data['CurrentId'] = $Id  $Data['Create'] = 'Filter already exists'  if ($PauseAction) { PausePrompt }  }  $NewRow = New-Object PsObject -Property $Data  Export-Csv -NoTypeInformation -Path $DataCsv -InputObject $NewRow -Append  if ($PauseFilter) {  PausePrompt 'Filter processed'  }  } |

# Algorithm

1. Read FilterCsv and PermissionCsv file.
2. For each filter in FilterCsv:
   1. Check if filter name exists under owner.
   2. If filter exists, log as filter already exists.
   3. Otherwise:
      1. Retrieve filter references.
      2. For each filter reference:
         1. Check if filter name is accessible.
         2. If not accessible:
            1. Create dummy filter with same name.
            2. If success, record created dummy filter id.
      3. If no error occurred for filter references:
         1. Create filter.
         2. If created successfully, change filter owner.
      4. Delete created dummy filters.

The algorithm ensures it will not modify existing filters in any way.

# Usage

|  |
| --- |
| > .\CloudFilterFromBackup.ps1 -Domain <name.atlassian.net> -Email <Email> [-Token <API Token>] -FilterCsv <FilterCSV> -PermissionCsv <PermissionCsv> [-PauseAction] [-PauseFilter] [-DataCsv <DataCsv>] |

## Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| Domain | Jira Cloud domain to work on, e.g. kcwong.atlassian.net |
| Email | Email address. |
| Token | API token. |
| FilterCsv | CSV containing filters to be created. |
| PermissionCsv | CSV containing filter permissions. |
| PauseAction | Pause after each action. |
| PauseFilter | Pause after each filter is processed. |
| DataCsv | CSV output file. If not specified, defaults to |

## FilterCsv

FilterCsv contains information about filters to be created. It must have a header row with the following columns:

|  |  |
| --- | --- |
| **Column** | **Description** |
| id | Reference id to link data between FilterCsv and PermissionCsv. This is not the filter’s id in Jira. |
| name | Name of the filter. |
| jql | JQL. Filter references must be in name instead of id. |
| owner | Filter owner’s account id. |

## PermissionCsv

PermissionCsv contains filter permissions. It is in a many-to-one relationship with FilterCsv. It must have a header row with the following columns:

|  |  |  |
| --- | --- | --- |
| **Column** | **Description** | |
| id | Reference id to link data between FilterCsv and PermissionCsv. This is not the filter’s id in Jira. | |
| type | loggedin | Same as “My Organization” in UI. Param1 and param2 will be ignored. |
| project | Param1 is project id.  You can locate project id using REST API:  /rest/api/latest/project/[Project Key]  Param2 is project role id.  It can be null, which indicates all project roles.  You can locate project role id using REST API:  /rest/api/latest/role |
| group | Param1 is group name, e.g. “jira-administrators” |
| user | Param1 is account id. |
| rights | 1 | View. |
| 2 | Edit. This is defined but never used in Jira, as Edit implies having View as well. |
| 3 | View + Edit. |
| param1 | Value depends on type. | |
| param2 | If type is project, this contains project role id. | |

Note: You can have additional columns in the CSV files for readability.

## DataCsv

DataCsv is the output file. It contains the following columns:

|  |  |
| --- | --- |
| **Column** | **Description** |
| BackupId | Reference id to from FilterCsv. |
| CurrentId | Created filter id. |
| Name | Filter name. |
| Jql | Filter JQL. |
| Owner | Filter owner account id. |
| Create | Create result, success or failed. |
| ChangeOwner | Change owner result, success or failed. |
| Messages | Details of the processing. You can also see these displayed in console. |

# Sample Input Files

## FilterCsv

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **id** | **name** | **jql** | **owner** | **remarks** |
| 00001 | KC1 | project = "ACM" | 62a037af954f50006fcbfef6 | Private filter belonging to owner.  Expected result is filter is created. Acting user will not be able to access this filter. |
| 00002 | KC2 | project = "ACM" | 62a037af954f50006fcbfef6 | Filter viewable by authenticated users.  Expected result is filter is created. The acting user will be able to access this filter. |
| 00003 | KC3 | filter = "KC1" and filter = "KC2" | 62a037af954f50006fcbfef6 | Private filter depending on KC1 and KC2.  The expected result is KC1 will be created as a dummy filter, while KC2 will not. KC3 will be created successfully, then the dummy KC1 will be deleted. |
| 00004 | KC1 | project = "ACM" | 62a037af954f50006fcbfef6 | Filter that already exists.  Expected result is this will fail with filter already exists message. |
| 00005 | KC4 | project = "ACM" | 62a037af954f50006fcbfef6 | Filter with viewers and editors.  Expected result is filter is created with appropriate permissions. |

Notes: Assumes acting user is not 62a037af954f50006fcbfef6.

## PermissionCsv

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| id | type | rights | param1 | param2 | remarks |
| 00002 | loggedin | 1 |  |  |  |
| 00005 | user | 1 | 62a037af954f50006fcbfef6 |  |  |
| 00005 | project | 1 | 10129 |  | Project ACM |
| 00005 | project | 3 | 10035 | 10002 | Project YY, Role administrators |

Note: Project id and project role id are from <https://consoleconnect-sandbox-824.atlassian.net/>