Migrate from Server to Cloud

Required:

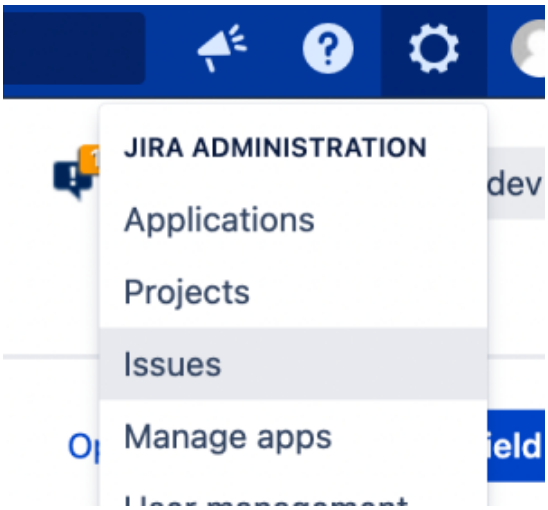
- MLCS Server 6.1.0 plugin or newer installed on the Jira Server / Datacenter instance
- Jira server / datacenter 8.20.0 or newer
- Jira Cloud instance

Migrate Tree options

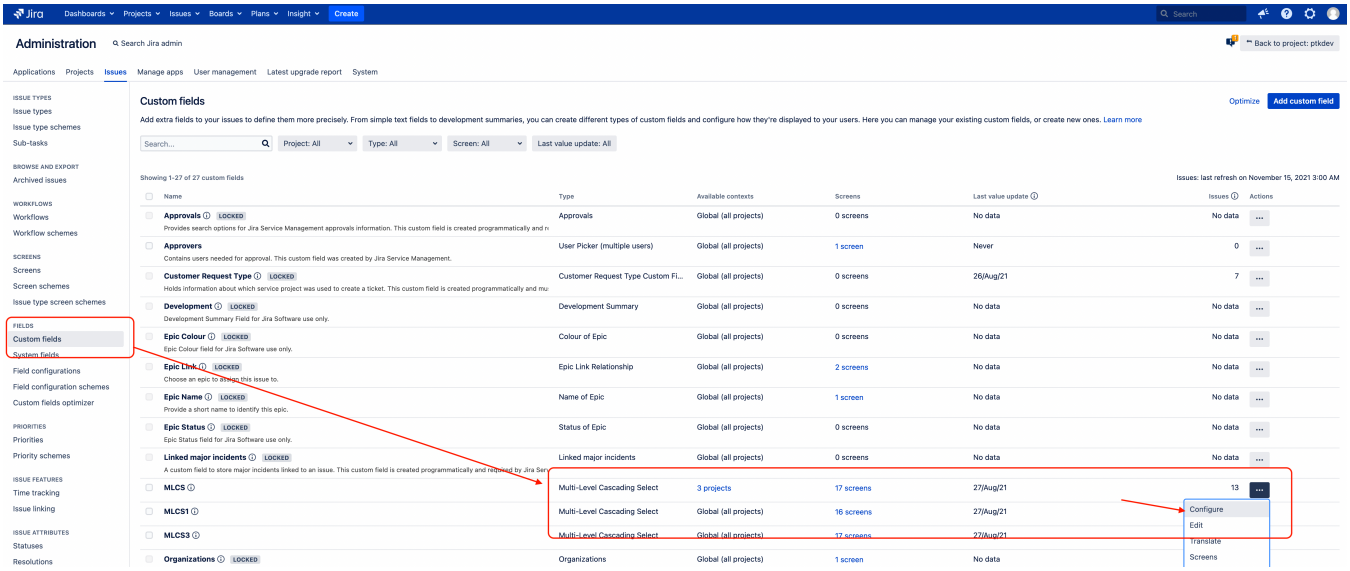
You can download json file of tree options from MLCS server / data center and import it on MLCS Cloud.

Export json tree from MLCS Server / DC

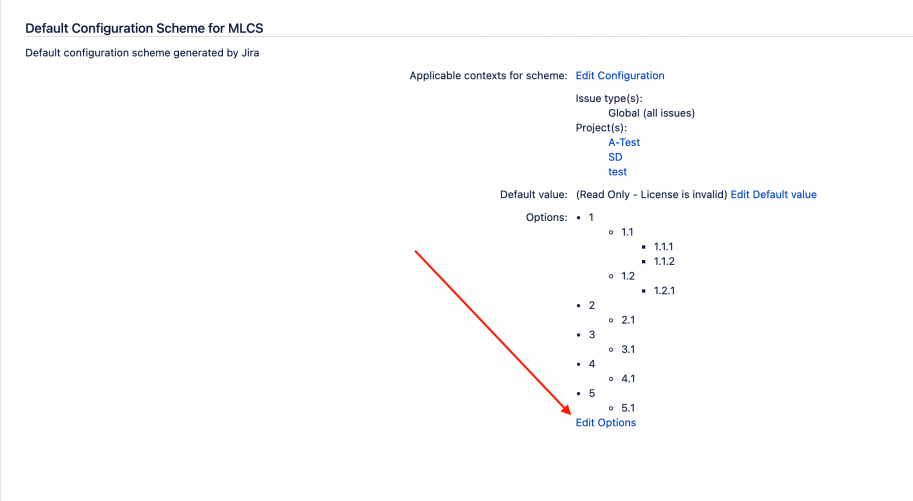
Go to **Settings Issues**



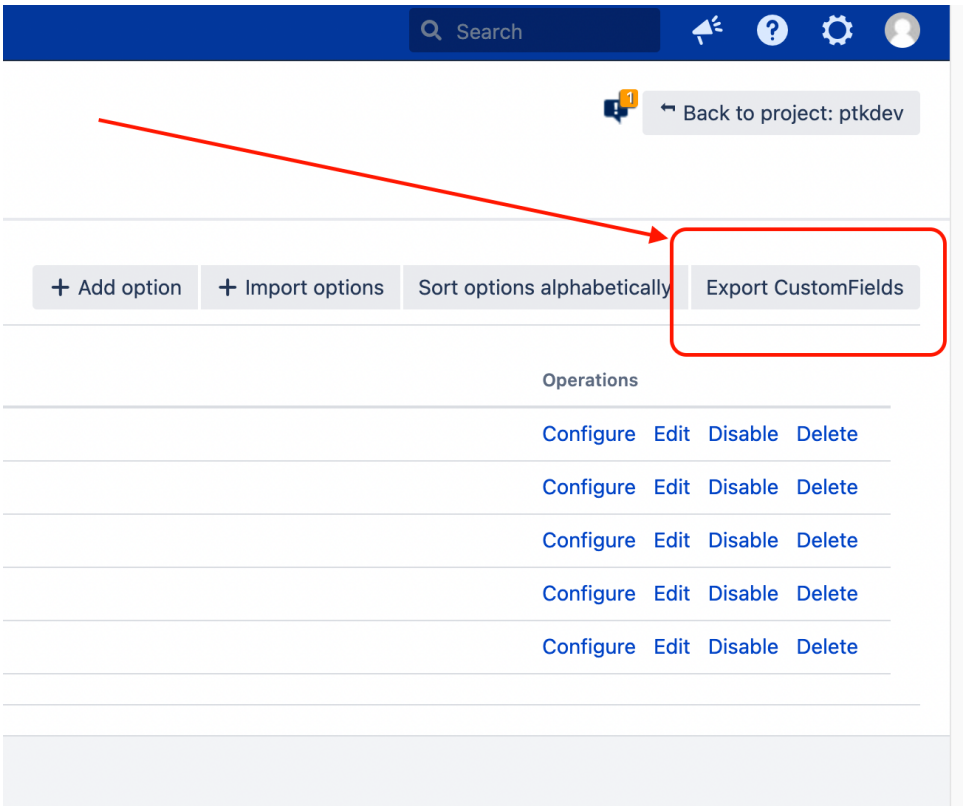
Go to **Custom fields** on left side and edit MLCS Custom field type, click on 3 dots **... configure**



Go to **edit options** of your custom field type

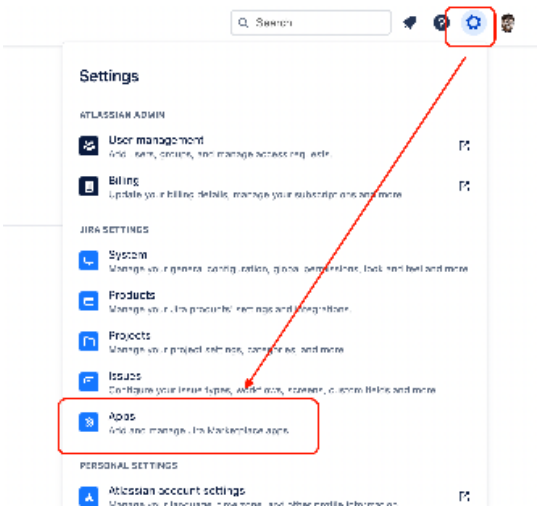


Export json tree file on **right side** (required plugin version 6.21.0 or newer)

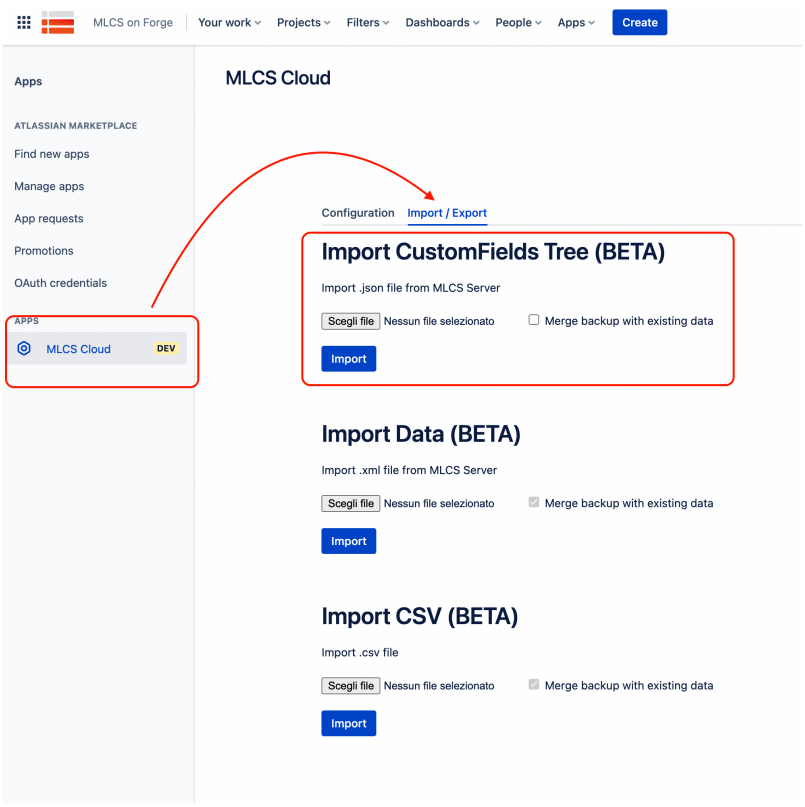


Import json tree from MLCS Cloud

Go to **Settings App**



Go to **MLCS Cloud** on left side and open tab **"Import / Export"**



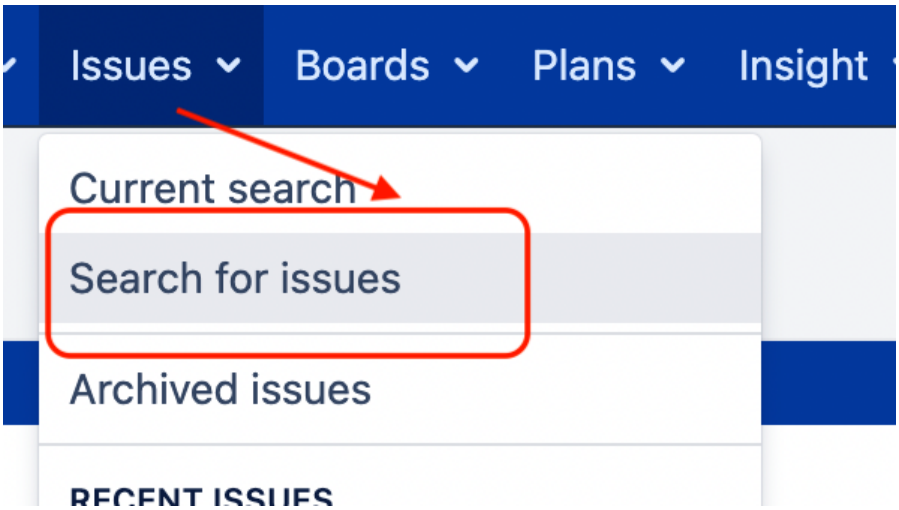
Choose json file from MLCS Server / DC and **import** it**.**

Migrate Custom Fields Type from all issue (multi selects values)

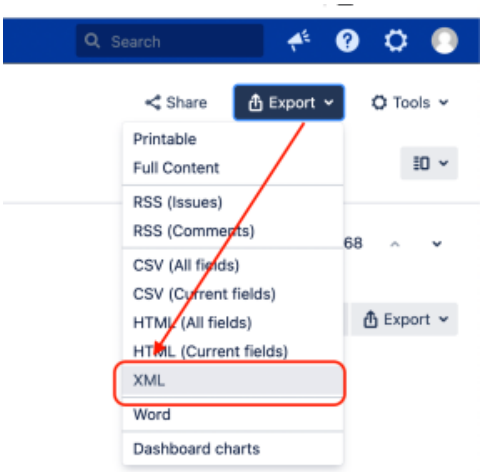
There are two ways to migrate MLCs field values from server to cloud:

1. Using xml data importer

Go to server instance, **Issues Search for issues**



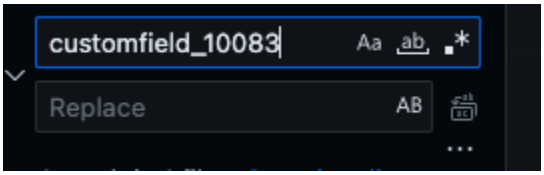
On right side, **Export XML**



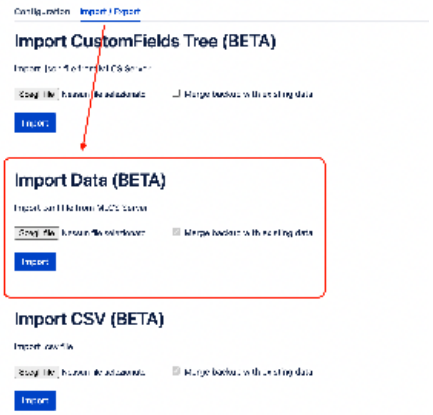
hing this issue

Create the field if it does not exist.

Open the XML file using your preferred editor, and replace all instances of the server's MLCs custom field ID (ex: customfield_13243) with the cloud MLCs custom field id



Now you can go to **Cloud** version of plugin and **Import Data (XML)**:



2. Using ScriptRunner Console

Configuration Steps

1. **Set Up Jira Cloud API Access**:
    - Determine your Jira Cloud REST API Base URL. It typically follows the format `https://[YourInstance].atlassian.net`.
    - Create an API token from your Atlassian account. Follow Atlassian's guide for creating an API token.
    - Combine your email and API token in the format `email:token` . This will be used for basic authentication.
2. **Identify the Custom Field**:
    - Find the name of the MLCS custom field in your Jira Server instance that you wish to migrate.
    - Obtain the ID of the corresponding custom field in your Jira Cloud instance.
3. **Update the Script**:
    - Replace `jiraCloudApiBaseUrl` with your Jira Cloud instance's base URL.
    - Update `encodedAuthString` with your email and API token.
    - Update `customFieldName` with the name of your MLCS custom field in the Jira Server instance.
    - Replace `cloudCustomFieldId` with the ID of the corresponding custom field in your Jira Cloud instance.

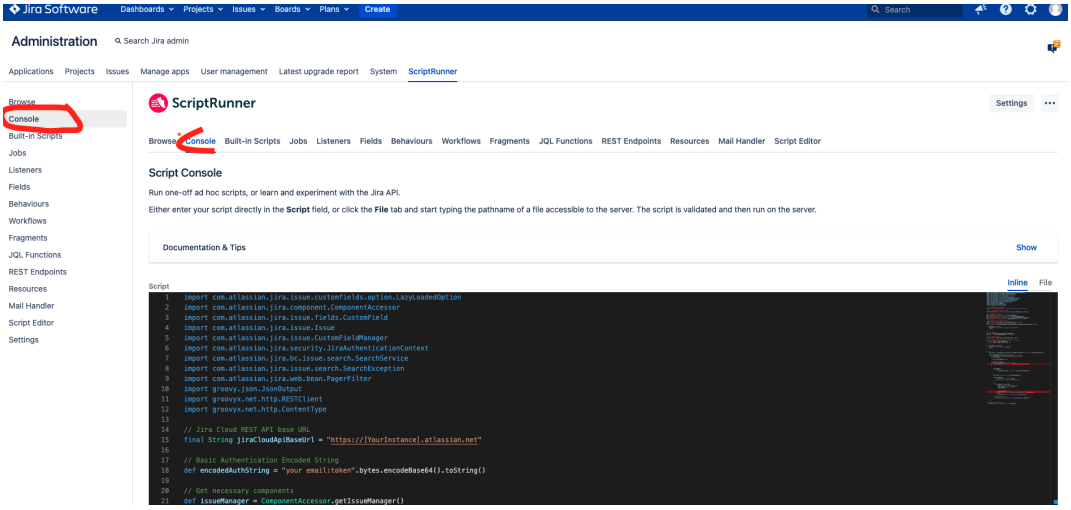
```
import com.atlassian.jira.issue.customfields.option.LazyLoadedOption
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.issue.fields.CustomField
import com.atlassian.jira.issue.Issue
import com.atlassian.jira.issue.CustomFieldManager
import com.atlassian.jira.security.JiraAuthenticationContext
import com.atlassian.jira.bc.issue.search.SearchService
import com.atlassian.jira.issue.search.SearchException
import com.atlassian.jira.web.bean.PagerFilter
import groovy.json.JsonOutput
import groovyx.net.http.RESTClient
import groovyx.net.http.ContentType

// Jira Cloud REST API base URL
final String jiraCloudApiBaseUrl = "https://[YourInstance].atlassian.net"

// Basic Authentication Encoded String
def encodedAuthString = "your email:token".bytes.encodeBase64().toString()

// Get necessary components
def issueManager = ComponentAccessor.getIssueManager()
def customFieldManager = ComponentAccessor.getCustomFieldManager()
def authenticationContext = ComponentAccessor.getJiraAuthenticationContext()
def searchService = ComponentAccessor.getComponent(SearchService.class)

// Define and find the custom field
final String customFieldName = "mlcs employee"
final String cloudCustomFieldId = "customfield_10101"
def customField = customFieldManager.getCustomFieldObjectsByName(customFieldName)?.first()
```

```
if (customField == null) {
    log.error "Custom field not found: $customFieldName"
    return
}

// Fetch issues
def user = authenticationContext.getLoggedInUser()
def query = "\"${customFieldName}\" is not EMPTY"

// Parse the JQL query
def parseResult = searchService.parseQuery(user, query)

def transformElement(LazyLoadedOption element, int index) {
    String label = element.getValue()
    return [label: label, value: index + 1]
}

if (!parseResult.isValid()) {
    log.error "Invalid JQL Query: ${query}"
    return
}

try {
    def results = searchService.search(user, parseResult.getQuery(), PagerFilter.getUnlimitedFilter())

    results.getResults().each { Issue issue ->
       List serverValue = issue.getCustomFieldValue(customField)
        if (serverValue) {

           def transformedArray = [:]

        // Populate the map
        serverValue.eachWithIndex { element, index ->
            transformedArray["lv$index"] = transformElement(element as LazyLoadedOption, index)
        }


            def payload = [
                fields: [
                    (cloudCustomFieldId):  JsonOutput.toJson(transformedArray)
                ]
            ]

            def client = new RESTClient(jiraCloudApiBaseUrl)

            client.setHeaders([
                'Content-Type' : ContentType.JSON,
                'Authorization': "Basic $encodedAuthString"
            ])

            try {
                def response = client.put(
                    path: "/rest/api/3/issue/$issue.key",
                    contentType: ContentType.JSON,
                    body: payload
                )

                if (response.status != 204) {
                    log.error("Failed to update issue ${issue.key}: ${response.data}")
                } else {
                    log.info("Successfully updated issue ${issue.key}")
                }
            } catch(Exception e) {
                log.error("Error updating issue ${issue.key}: ${e.message}")
            }
        }
    }
} catch (SearchException e) {
    log.error("Error executing search: ${e.message}")
```

```
}
```

Execution

1. **Run the Script**:
    - Open the ScriptRunner Console in your Jira Server instance.
    - Paste the updated script into the console.
    - Execute the script.



Troubleshooting

If issues fail to update, check the log for error messages. Ensure that all instance URLs, field names, and IDs are correct. Verify that your API token has appropriate permissions. Also, be aware of the following HTTP status codes and their implications:

1. **400 Bad Request**
    - This error occurs if the user lacks the necessary permissions to edit the issue or to view it, or if the mlcs field is not found in the cloud or not associated with the issue's edit screen.
2. **401 Unauthorized**
    - This status is returned if the token or the email is not valid.
3. **403 Forbidden**
    - This can happen if the user does not have the necessary permissions to edit the issue or to view it.
4. **404 Not Found**
    - Returned if the issue is not found or the user does not have permission to view it.