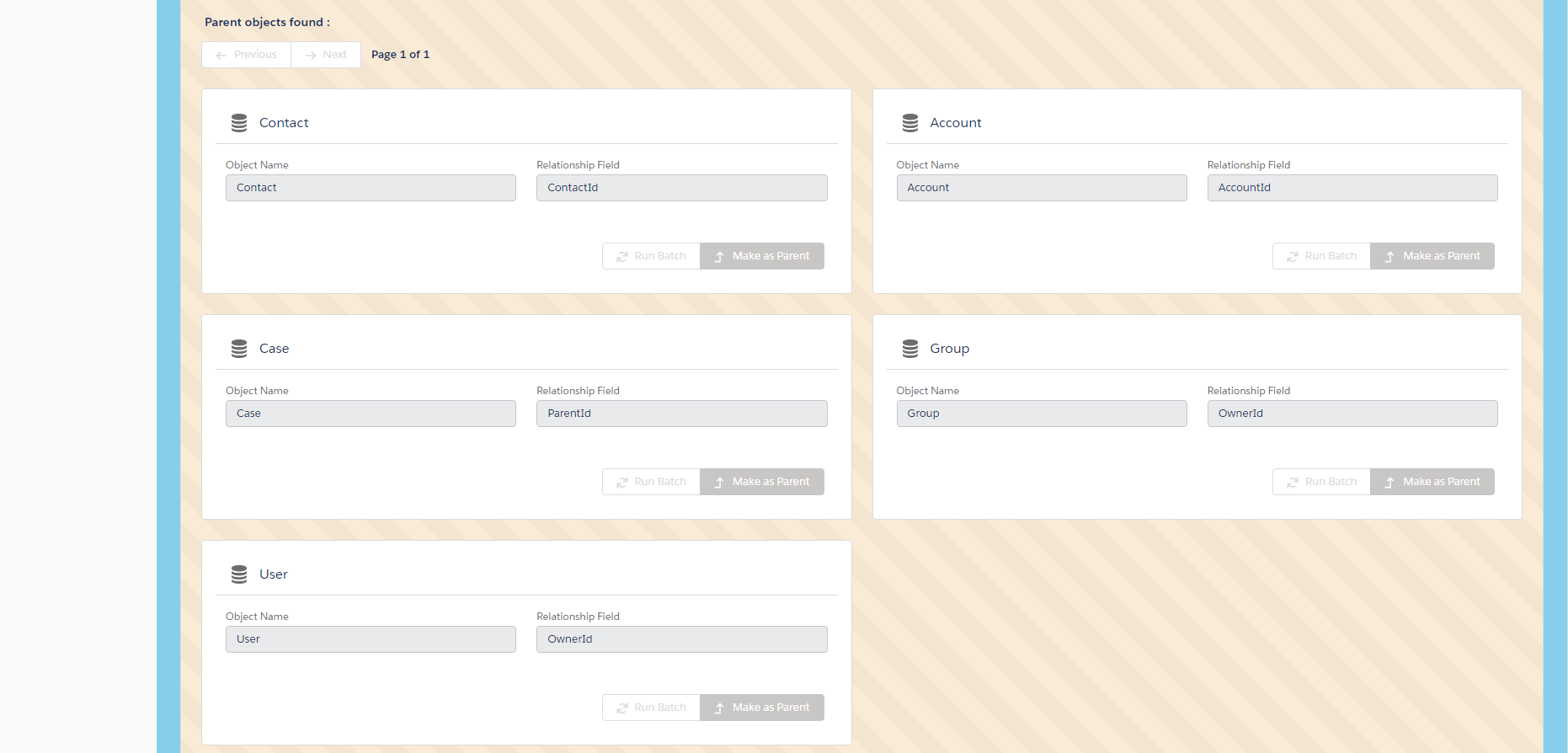
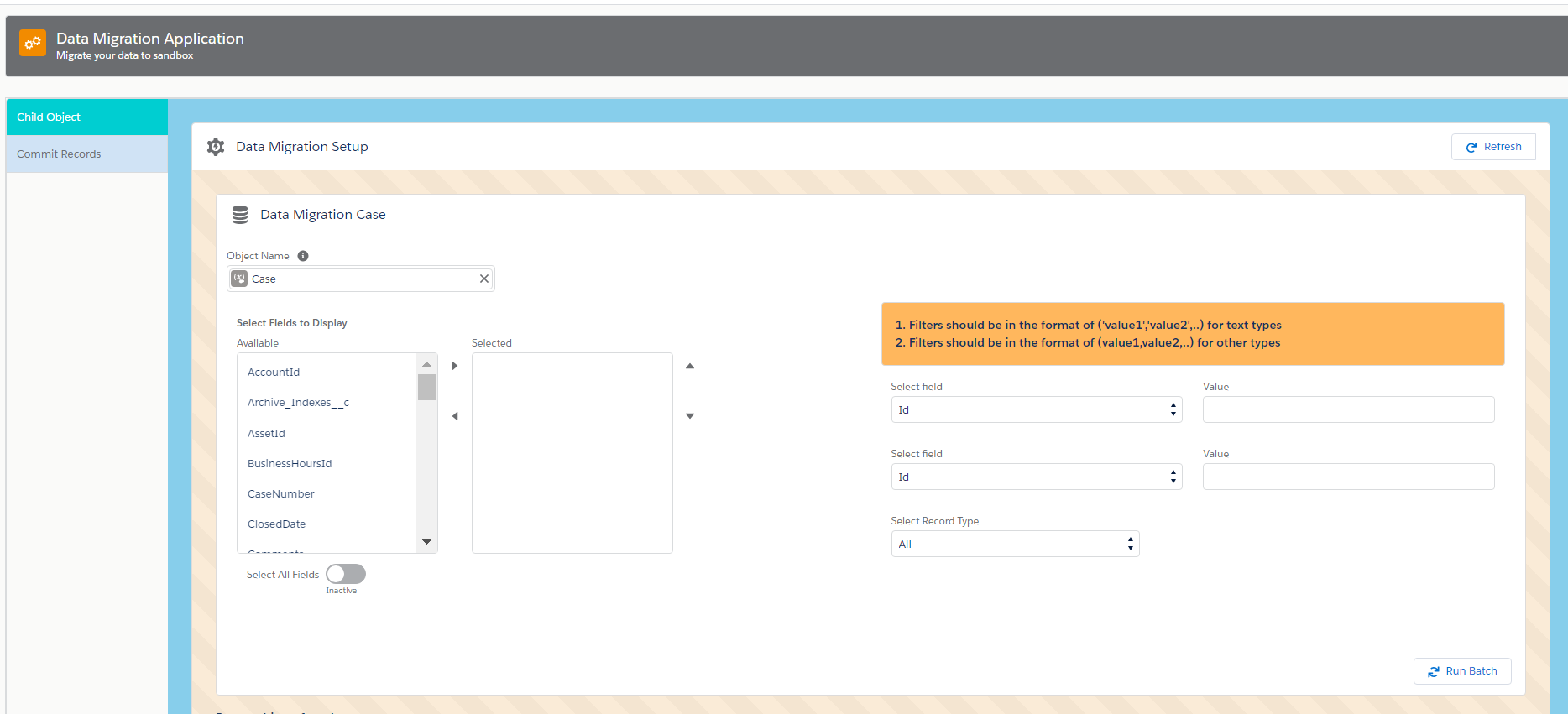
**Data Migration Application**

**URL -** <https://nfx--qa.lightning.force.com/c/DataMigrationApp.app>

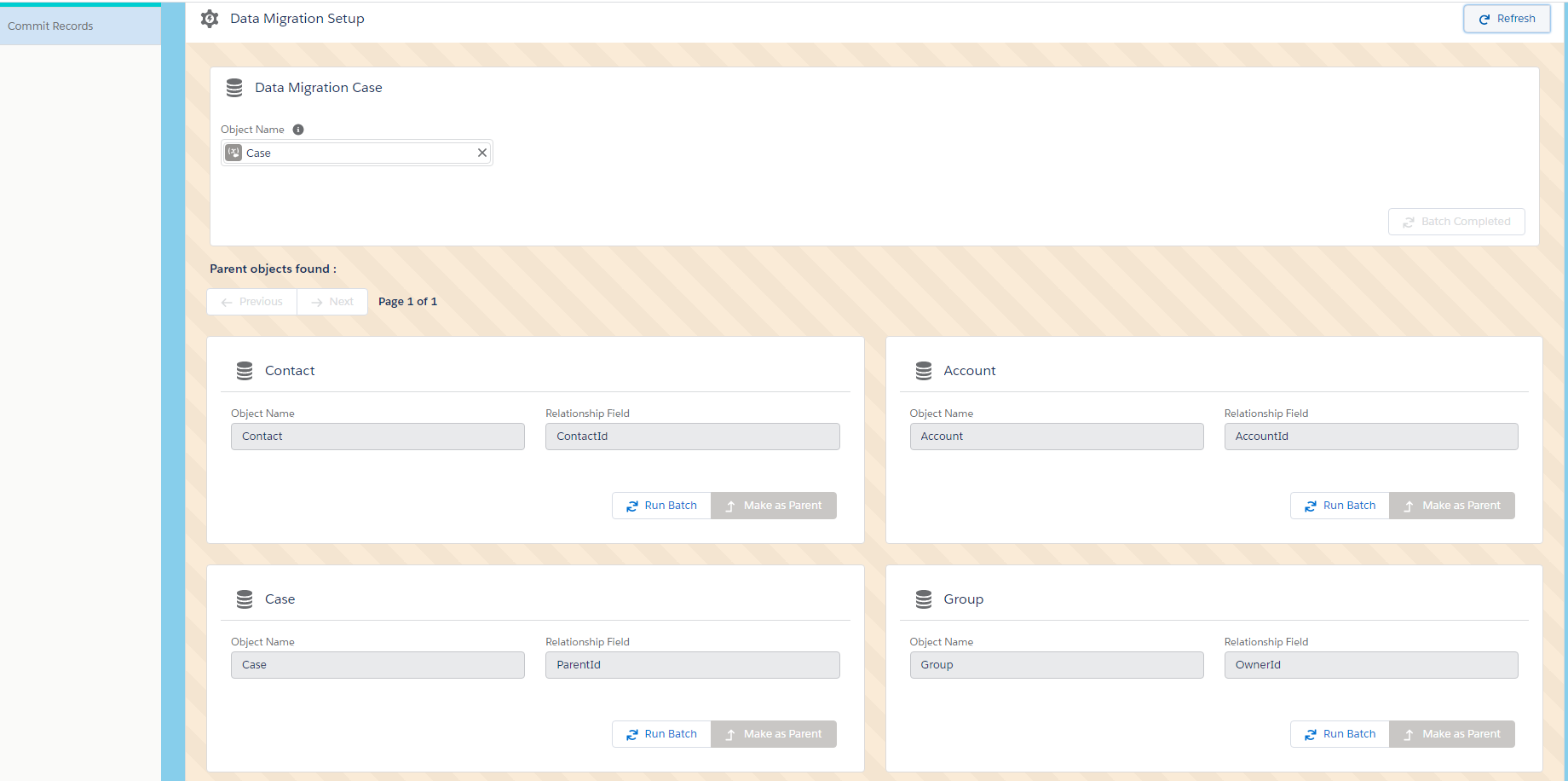
This application is used for migrating standard and custom object records from one salesforce org to another along with all parent object records in the hierarchy. It also provides a save and commit mechanism where the users can first run jobs to move the records to target org and then commit them to database when all the records in the hierarchy are moved by specifying the commit order.



1. User selects the object to migrate and saves the settings using ‘Save Settings’ button.
2. Once saved, there will be settings created for child parent objects as well which will be disabled until the job for the child object is run. They can select the fields to be migrated and provide filters for record type and 2 additional fields which are to be in the format of

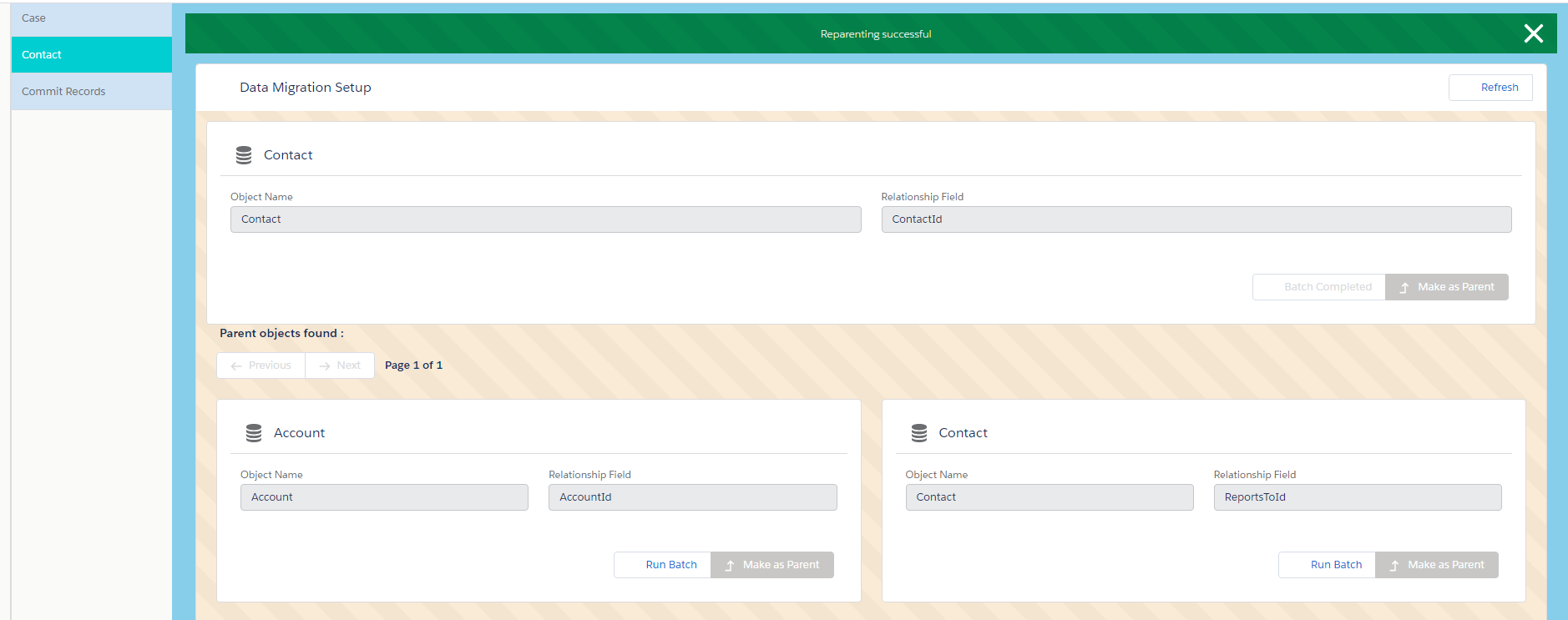
fieldname\_\_c=(‘fieldValue 1’ ,fieldValue 2’…)

1. The users then click on the ‘Run Batch’ button to start running the batch. This will keep the buttons disabled. Users can click on the refresh button to check the status of the batch whenever required.
2. Once the job is run the buttons ‘Run batch’ will be disabled on the child and will be enabled for the parent objects to run the batch for them.

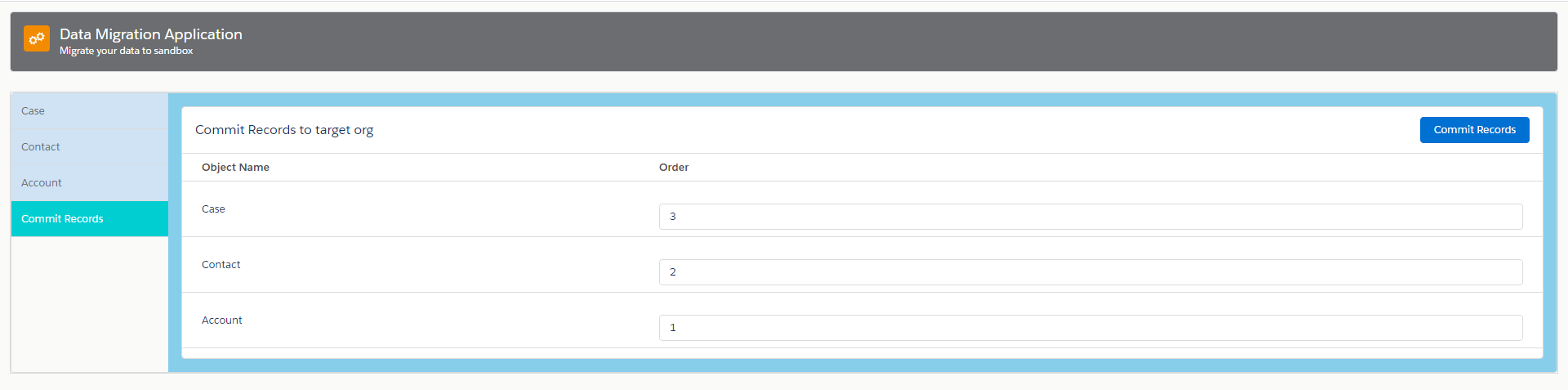


1. Users can run the batch for the parent objects and once it is completed, the ‘Make as Parent’ button will be enabled to make any parent object as the new child object in a different tab (this will remove the reparented parent object from the Parent objects list of the initial selected object which in this case is ‘Case’ object’) where the parent objects of this new child object will be visible in the same manner.

Ex: if Case Records have to be moved, once the case batch is run the parent Contact batch is run, then the account and so on. Then we make the Contact Object as the new child and the parents of Contact which are Account etc will be shown in the ‘Parent Objects found’ section. This can be done until we traverse the whole Schema of objects. Case -> Contact -> Account -> …



1. Once the batches are run for all the objects, we can go to the commit records tab and commit the records in the target org in the proper order. Parent records should be inserted first, so we start giving the order from parent and then to child. As seen in the below screenshot. The Account is given the order as 1 which means it should be committed first.



**Source Org Components:**

**Lightning Components:**

DataMigrationApp – hosts the whole application

mainAppHeader – Shows the header of the page with Icon and theme.

getAuthComponent – used to select the named credential for making callouts to target org where the data needs to be migrated.

authSuccessEvent- triggered when authentication is successful. Handled in DataMigrationSectionsComponent

DataMigrationSectionsComponent – Creates the tabs for child and parent objects and also the tab for committing records.

DataMigrationForm – Creates the primary child object card and parent object cards separately, along with Run batch button displayed conditionally.

DataMigrationRowComponent – Creates cards for each of the objects and conditionally displays the filter component and field selection dual list when the object context is the first selected object.

FilterComponent – displays 2 rows of filters along with the fieldname as dropdown and value as text field.

searchComponent – used for searching objects

searchResults- displays the matched results as user types object names

clearValueEvent – triggered when the object selection is cleared from the field

oSelectedValueEvent – triggered when an object is selected

refreshParent – triggered when one of the parent objects is made as the new child and refreshes the current tab to get the updated settings.

dataInsertComponent – Used for specifying the commit order and committing the records in the target org.

**VF Page:**

SessionIdPage – used to get the session Id in lightning context since userInfo object doesn’t provide the session Id when called in lightning. Used to use the metadata API to get named credential names.

**Objects:**

Data\_Migration\_Auth\_\_c custom setting:

Created during initial setup after selecting the named credential to be used for integration.

|  |  |
| --- | --- |
| **Field Name** | **Purpose** |
| Name | To hold the unique value. Should be ‘migration’ |
| Named\_credentials\_\_c | To hold the named credentials api name to make callouts. This should be the named credential that connects and authenticates to the target org. |

Migration\_Objects\_\_c custom setting:

Creates the custom setting records for each of the objects (parent, child) details like relationship field, hierarchy, object type, record type, filtervalue, filter field.

|  |  |
| --- | --- |
| **Field name** | **Purpose** |
| Name | Unique record name used with current timestamp |
| objectName\_\_c | API name of the object |
| filterField\_1\_\_c | Field name for the field to be filtered |
| filterField\_2\_\_c | Field name for the field to be filtered |
| filterValue\_1\_\_c | Field value to be filtered by |
| filterValue\_2\_\_c | Field value to be filtered by |
| Recordtype\_\_c | Record type to be filtered by |
| Object\_type\_\_c | Indicates whether it is a child object or parent.  Usually this will be appended with a number indicating what level of child object it is. |
| Hierarchy\_level\_\_c | Indicates what hierarchy level the object is in.  First child (case) – 1  First level Parent (Contact)– 2  Second level Parent (Account) - 3 So on |
| Operation\_type\_\_c | Value is equals to ‘migrate’ |
| Status\_\_c | Used to hold the job status. |
| Child\_setting\_name\_\_c | Acts as a lookup field. For any parent record, this will hold the custom setting name of the child record. |
| relationship\_field\_\_c | Lookup or master-detail field name. For any parent record, this will hold the lookup / master detail field api name of the immediate child object that references this child object. If there are multiple fields referencing this object from same child, the field names are comma separated.  Ex: For custom setting for Account object,  Child setting name – {custom setting name of contact object}  Object type – child3 since this is a 2nd level Child and has multiple parents  relationship\_field\_\_c –AccountId since the child from which this setting was created was Contact and for contact, lookup field to account is AccountId |

**Classes:**

dataMigrationFormController – Main controller for the application

Methods:

getMaxHierarchyLevel – gets maximum hierarchy level in the custom settings to display the tabs.

getSettings – Each tab will get the custom settings based on hierarchy level passed to this method.

deleteAllSettings – deletes all Migration Object records and clears the big object records from target org by making a callout

saveSettings – when user selects the first child object or reparents a parent object to make it the new child, the parent objects are found for that object and a parent record is created in migration object custom settings for each parent object. Hierarchy level is incremented for the new parent records to show that these objects are in higher order than the previous one.

getFieldList – gets list of editable fields for the first child that is selected.

runBatch – This will run the dataMigrationSourceBatch directly if the object hierarchy level is 1 or it is a first child selected.

If not, we run parentArchivalBatch on the first child and pass the hierarchy level of the object for which the batch dataMigrationSourceBatch is to be run.

Ex – In case we run the batch on Account, we start the parentArchivalBatch on Case object since it is the first child, get the case records using the filters that were initially provided, then call parentArchivalBatch again for these case records’ parent Contacts and then run dataMigrationSourceBatch on parent Accounts’ of these Contacts.

getObjectsPushed – gets all the objects names of the records that are saved in the target org’s big objects. It returns a wrapper of type objectOrderWrapper with objectName and order number.

commitRecords – accept the order of insertion and commits the records in the target org by making a callout.

changeParent – It is called to make any parent object of a child object to make it the next new child object. This will cause the object\_type\_\_c in Migration object settings to have a value of **‘Child’ + hierarchy\_level\_\_c -1**

ex : when we make Case’s Contact as the new Child the object type for Contact in the Migration Object record will be ‘Child2’

getLastChild – This is a recursive method called to get any child object details from migration\_objects at any level.

Ex: when we call getLastChild(case object’s setting, 2) it will return settings of Contact.

getLastChild(Contact object’s setting, 3) will return settings for Account. And so on.

dataMigrationSourceBatch –

It is called to run the batch for records of the first child object and the last batch execution in the chain while running the batch for parent objects. This batch will query the records and send those records to the target org by making an API callout.

parentArchivalBatch –

It is called when we want to run batch for parent objects at any hierarchy level.

Ex: if we want to run for Account and we had selected Case as the first child.

1. First the parentArchivalBatch will run for Case object based on the filter and record type criteria provided for Case in the beginning of the application. We also pass the settings of immediate parent object for Case which is Contact.

The query is created by using the **relationship\_field\_\_c** on Contact’s setting record which will have the field api name of the case which looks up to this contact.

In this case it is **ContactId**.

1. Then we get the contactIds from these case records and call the getLastChild method from the main controller and pass the next hierarchy level which is 3.

Contact’s hierarchy level +1

1. Then we run the parentArchivalBatch on Account object with Contact as the child. So, query will run on Contact and with getting fields from the **relationship\_field\_\_c** on Contact’s setting record to get the Contact’s Parent account Ids.
2. Now since the hierarchy level of Account match the desired hierarchy level that was sent in the beginning, we run the dataMigrationSourceBatch by passing the Ids of these account records.
3. Here we ran for the parent object at hierarchy level 3 similarly we can run this job for an object at any hierarchy level, so it gets the parents of only those first child object records for which we had provided filters.

This way we can move all records for the selected first child object and all its parents in the schema tree to the target org and then finally commit them.

searchComponentController – Used for searching sobjects in the org while selecting the first child object.

mainApplicationController – Used during the initial application setup to get the named credentials in the org so the users can select the desired one and will be saved in the Data\_Migration\_Auth\_\_c custom setting record with the record Name ‘**migration’**

MetadataService – used to get all named credential in the org to be displayed as dropdown while initial app setup.

**Target Org Components:**

Data\_Migration\_Index\_\_c custom setting: Used to hold a record which will store the current index value in the big object to maintain uniqueness while inserting them.

|  |  |
| --- | --- |
| **Field Name** | **Purpose** |
| Name | Should be equal to ‘[migrationIndex](https://nfx--qa.my.salesforce.com/a0EW0000005HPOK?setupid=CustomSettings)’ |
| Record\_id\_\_c | Stores the record Id of the same record to update without querying |
| indexNumber\_\_c | Stores the next index number to be used for big object insertion. |

Migration\_record\_Id\_Map\_\_c custom setting: Used to hold the map of the record Id from source org and corresponding record Id for target org after record is inserted. This is used to get the new relationship field values to be used when saving the records in target.

This is a temporary storage and once all records are committed the values will be moved to Data\_Migration\_Record\_\_b big object and cleared from the custom setting.

|  |  |
| --- | --- |
| **Field Name** | **Purpose** |
| Name | Should be equal to the record Id from source |
| new\_Record\_Id\_\_c | Stores the corresponding record Id in the target org. |

Ex: Case has the lookup to Contact. Since parent records need to be given the lowest order while committing which means it is committed first, we will insert it and create a record in Migration\_record\_Id\_Map\_\_c with the Name field as the record Id from source and new\_Record\_Id\_\_c as the corresponding record Id of the contact in the target org.

Now when inserting case records, we use a logic to get all relationship fields of case and get the values of those fields in the case and replace it with the new values from Migration\_record\_Id\_Map\_\_c.

Data\_Migration\_Record\_\_b big object: Used to store the records from the source in the target org before they are committed. The records are stored in a JSON format in recordJSON field.

There is a 0th record which holds additional information of the records stored in the big object.

1. The indexValue of this record is 0
2. recordJson\_\_c stores the comma seperated list of object names of all the records stored in the big object and not been committed
3. objectAPIName\_\_c holds the value ‘Header’
4. self\_lookups\_\_c holds the values of object names of all records stored in the big object that have self-lookup.

|  |  |
| --- | --- |
| **Field Name** | **Purpose** |
| indexValue\_\_c | Indexed field which is in increments of 1. The current index values are tracked in data\_migration\_Index\_\_c custom setting record. |
| objectAPIName\_\_c | API Name of the object of the record being stored. |
| recordJSON\_\_c | Holds the entire record from source in JSON format |
| record\_id\_\_c | Holds the record Id from source |
| self\_lookups\_\_c | Used only in the record where the indexValue\_\_c is 0. It indicates how many objects are self-lookup, so the batch runs with the batch size as 1. |

Data\_migration\_record\_id\_\_b: Holds all the old and new record Id mapping for all the records that has been created through this application during migration. Serves the same purpose as Migration\_record\_Id\_Map\_\_c, but this is a permanent storage and not to be cleared.

|  |  |
| --- | --- |
| **Field Name** | **Purpose** |
| oldRecordId\_\_c | Should be equal to the record Id from source. Serves as the big object index field |
| objectName\_\_c | Name of the object that the record belongs to |
| recordJSON\_\_c | Holds the entire record from source in JSON format |
| record\_id\_\_c | Should be equal to the record Id from target org that has been inserted |

This object also avoids any duplicate record creation by checking if the record id from source in the Data\_migration\_record\_\_b big object record exists in Data\_migration\_record\_id\_\_b big object record that maps the record Id to any record Id from target org.

**Classes:**

destinationDataMigration –

Rest API class that has a mapping of ‘DataMigration/\*’. This will be called from the source org to get the object names stored in the Data\_Migration\_Record\_\_b big object.

Methods:

getObjectsInBigObj –

‘/DataMigration/’ - gets all the objects who the records stored in the big object Data\_migration\_record\_\_b in JSON format belong to. This is sent to the source to be displayed in the commit records tab to specify the order of insertion.

‘/DataMigration/ deleteAll’ - deletes the records.

recordInsertion –

‘/DataMigration/ saveRecords’- Inserts the records from source in the big object Data\_Migration\_Record\_\_b and updates the Header record which is the object names and self-lookup object names.

‘/DataMigration/ commitRecords’ - starts the destinationRecordInsertBatch and decides the batch size based on whether there are records in the big object with self-lookup. If there is self-lookup then the batch size=1, else it is 50.

destinationRecordInsertBatch-

Accepts a list of objectOrderWrapper which contains the object name and the order in which it should be inserted. Based on this order the batch gets the records from Data\_Migration\_Records\_\_b big object filtered by objectAPIName\_\_c and **upserts** these records into the sobjects.

Once records are upserted, the old and new record id mapping is stored in the data\_migration\_record\_Id\_map\_\_c custom setting and data\_migration\_record\_Id\_\_b big object.

After the first object is done, we call the same batch class again with the next object in order and this continues until all records are inserted.

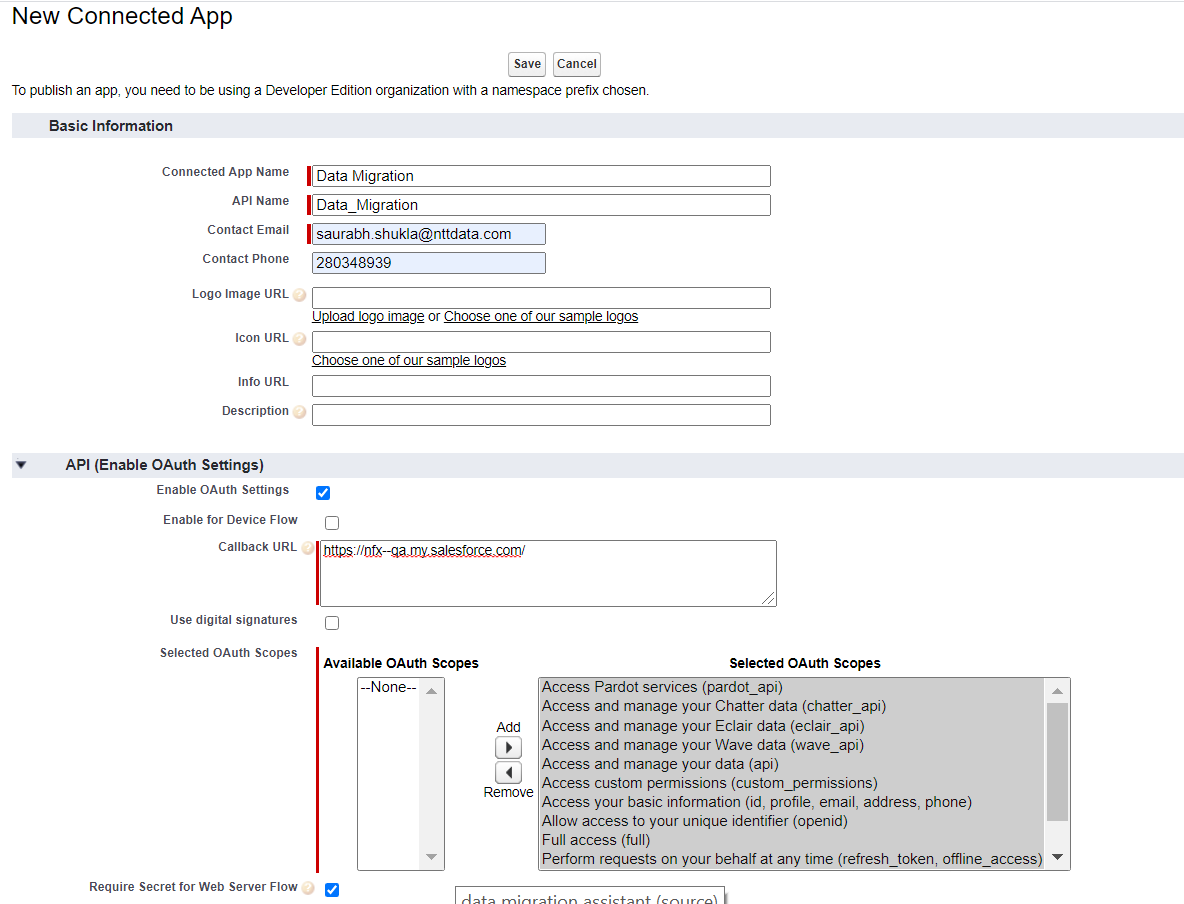
deleteBigObjectsBatch-

deletes the big object records from Data\_Migration\_Records\_\_b and custom setting records from data\_migration\_record\_Id\_map\_\_c. It then resets the indexValue from data\_migration\_index\_\_c custom setting record with record name ‘migration’.

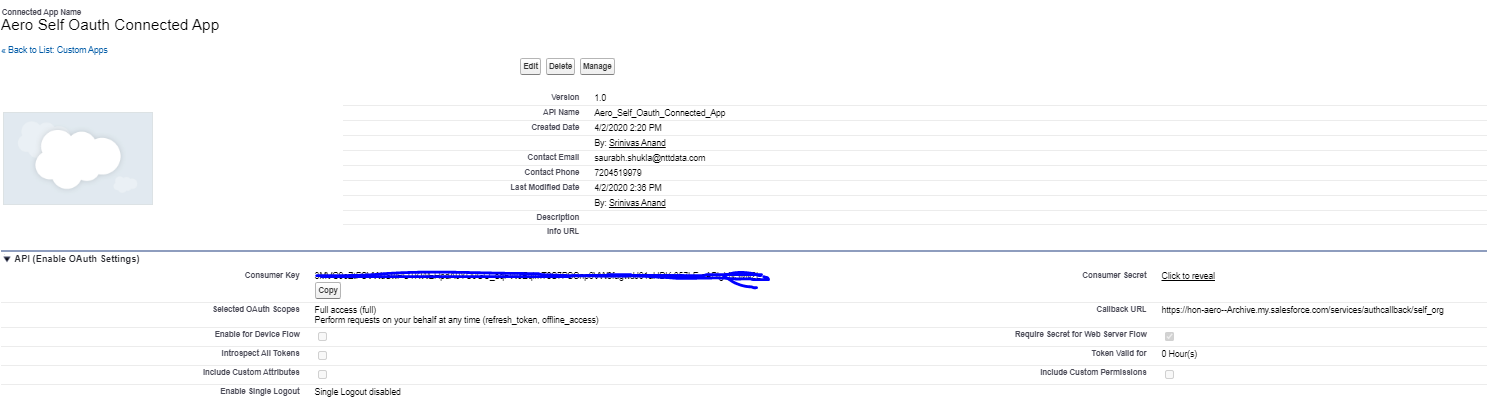
**Post Installation steps:**

**Connected App setup:**

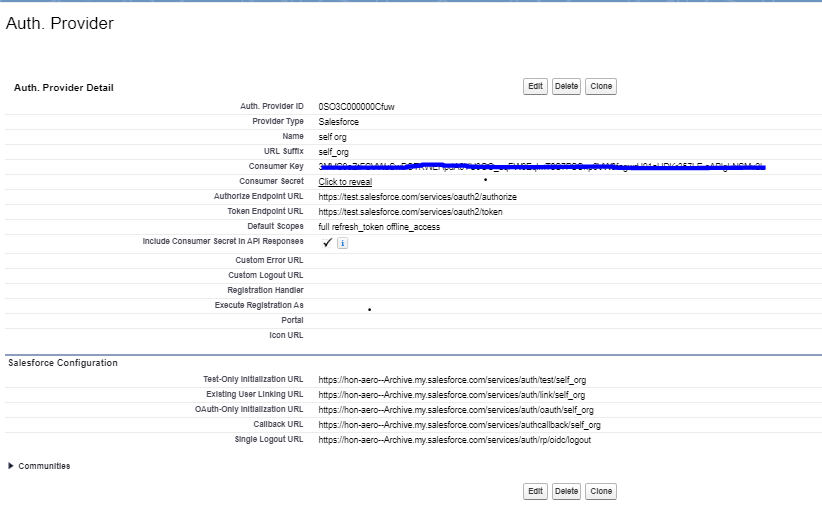
1. Create a connected app in the **target org** with oauth enabled and below settings. Callback url can be the source org for now and select all oauth scopes.



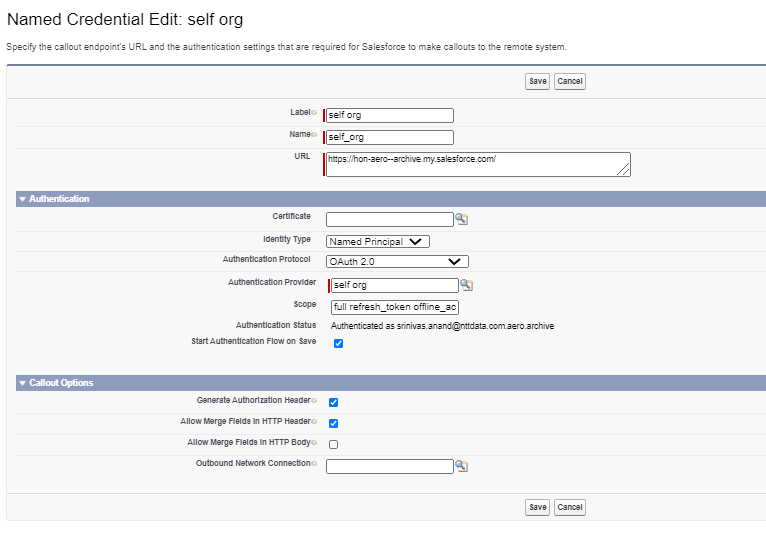
1. Save the app.



1. Create a auth. Provider in the **source org** with salesforce settings. Copy the client Id and client secret from the above created connected app and paste it in the auth provider.



1. Save the auth. Provider and you will find a callback url in the related list. Copy and paste the callback url in the connected app callback url.
2. Create a named credential in the **Source Org** use the Named principal authentication and oauth 2.0 option.
3. Lookup the auth provider that was created and use the same in the named credential.
4. Allow merge fields in the Http header, generate authorization header, start authentication flow on save should be checked.



1. Authorize the named credential on Save.

**Target Org:**

Create a custom setting record in Data\_migration\_index\_\_c with below details:

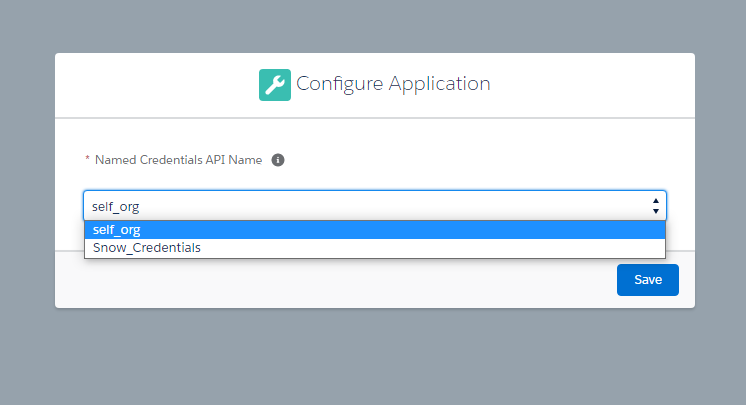
Name- migration

indexNumber- 1

Record Id - use the record Id of the same record after saving.

**Source Org**

On the first time opening the app, you will see the below screen.



Select the named credential that was created before and Save. This named credential will be used in all the callouts made to the target org.