exalate

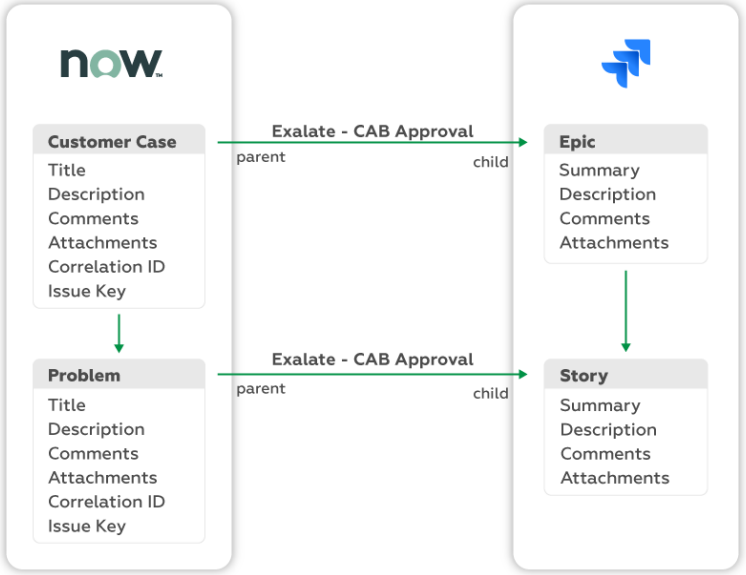
How to Synchronize a ServiceNow Customer Case to a Jira Epic

Table of contents

When it comes to cross-company integration solutions, it is highly critical to know if the solution can handle various use cases across different platforms. These use cases might vary from a simple use case like setting up a connection between two issue trackers to more technically advanced ones like implementing a synchronization from a ServiceNow customer case to a Jira epic.

Some time ago, a company asked if Exalate could handle a more advanced synchronization challenge - 'Case to Epic'. This company's request was to integrate Jira and ServiceNow in such a way that ServiceNow 'Customer Cases' could be escalated towards Jira as Epics.

Any additional problem that is uncovered in the context of that particular customer case, should create a story in the context of the epic.



As flexibility is one of Exalate's strongest suits, we decided to buckle up, accept the challenge, and get the configuration done. This piece will provide you with the details setting up such behavior yourself.

See it in action

Let's see how this can be done in action:

https://youtu.be/sBax5LE-GRc

Configuration

Configuring an Integration

With the distributed nature of Exalate, you always need to keep in mind in what context you are configuring.

Bidirectional synchronization requires that you specify how the synchronization is behaving in either direction:

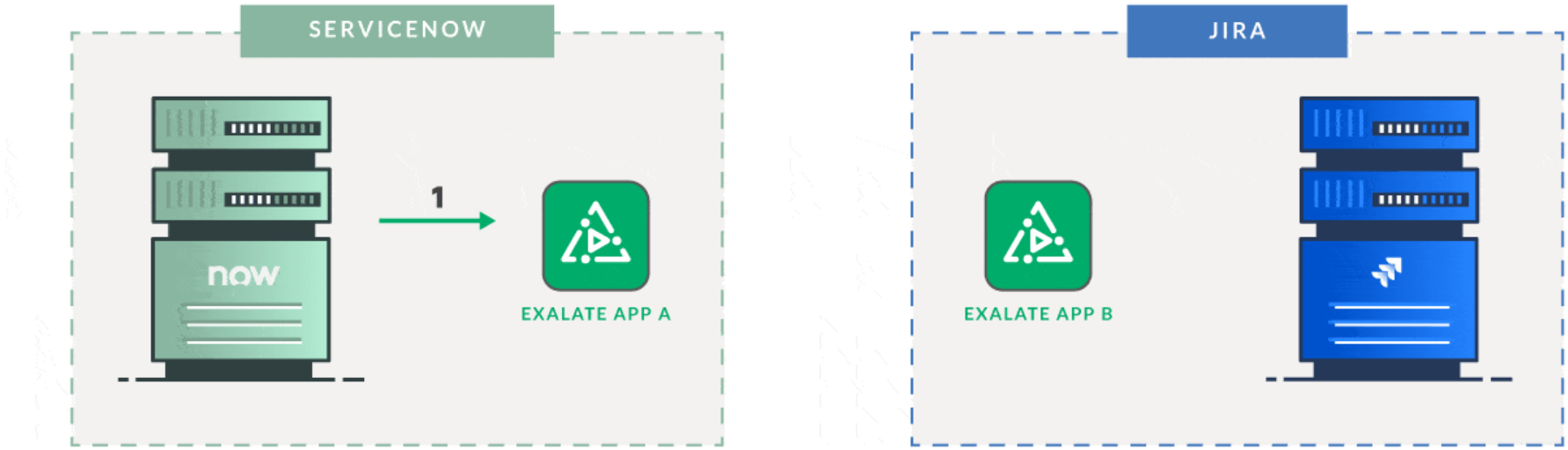
- From ServiceNow to Jira
- From Jira to ServiceNow

Furthermore, for each direction, you need to specify what information is sent from the source and how it is processed on the receiving end.

Exalate provides a groovy-based scripting engine which brings the necessary flexibility to implement this behavior. As an Exalate administrator, you will configure an incoming and outgoing processor on each side. The outgoing processor is used to specify what information is being sent from the local tracker, and the incoming processor is used to define how incoming messages need to be

processed.

So here are the steps to setting up a bidirectional configuration:



1. Exalate A interfaces with ServiceNow for retrieving issue information

- Specify what information is being sent from ServiceNow to Jira by configuring the outgoing processor on the ServiceNow side.
- Specify how this information is being applied on the Jira side by configuring the incoming processor on the Jira side.
- Specify what information is being sent from Jira to ServiceNow by configuring the outgoing processor on the Jira side.
- Specify how this information is being applied on the ServiceNow side by configuring the incoming processor on the ServiceNow side.

Whenever a connection is being set up, the incoming and outgoing processors are configured with a template allowing them to synchronize summary, description, comments, and attachments. To configure this particular case, a number of additional configurations are required.

Detailed Specifications

Before starting to implement the use case, it is always good to have an overview:

- Whenever a case is assigned to an ITSM Engineering assignment group - Exalate it towards the Jira, the issue created on the Jira side must then become an epic. *Exalate* is the action where an incident (problem. issue ...) is being escalated towards another system. Subsequent updates on the object are called synchronizations.
- The epic name (which is a mandatory field in Jira) must be set to the case identifier.
- On the ServiceNow side, Correlation.id should be set to the key of the Jira issue and a custom field 'issue link' should provide a direct link to the Jira issue.
- Whenever a problem is created in the context of a customer case, that problem needs to be automatically Exalated towards Jira, in the context of the corresponding twin.
- For problems, correlation.id and the issue link also need to be set.
- It is sufficient to synchronize comments and attachments, but Jira comments on the ServiceNow side must be work notes (not visible for the customers).

Installing Exalate on ServiceNow and on Jira

To implement the autonomy requirement, Exalate needs to be installed on both sides of the connection.

The instructions to setup Exalate on both sides are detailed in these pages:

- Installing Exalate on ServiceNow
- Installing Exalate on Jira Cloud / on Jira Server

Setting up the Connection

Configuring the connection is pretty straightforward. Check out this step-by-step guide or just follow these instructions:

- Configure Connection on Exalate ServiceNow
- Configure Connection on Exalate Jira Cloud
- Configure Connection on Exalate Jira Server

You can also check out this video if you'd prefer to learn it visually:

https://youtu.be/2rGAWEdEkjY

ServiceNow to Jira Configuration

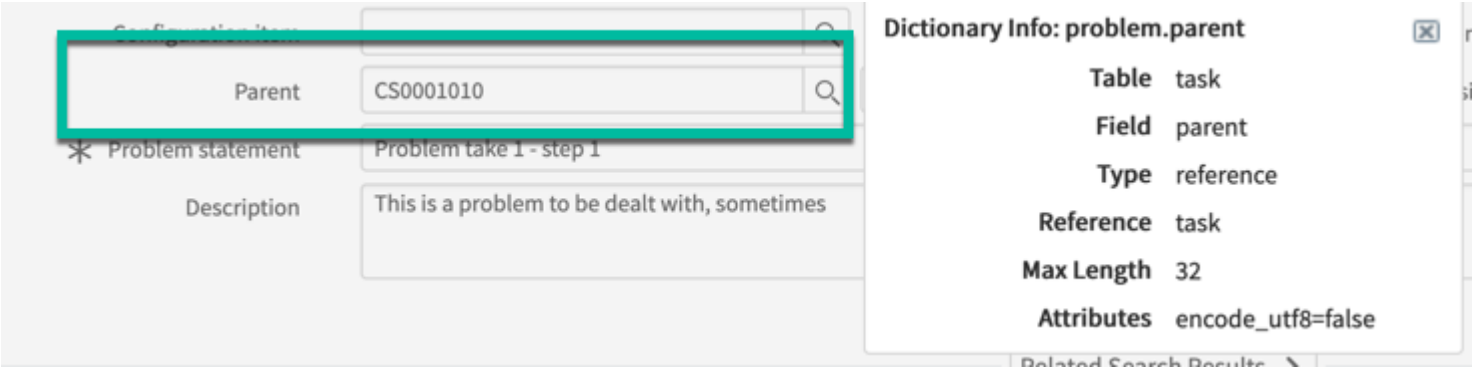
The Outgoing Message

The messages sent from ServiceNow to Jira should contain:

- entity type (customer case or problem) such that a decision can be made on the Jira side
- identifying the number
- title
- description

- comments
- attachments

In case of a problem, the customer case number is also required to find the epic back on the Jira side. The customer case number is tracked in the problem. parent field.



Inspecting the default setting of the connection - it appears that almost everything is already pre-configured, except for the entity type.

We could decide that the status is not necessary, but we can leave it in as well. You never know what the Jira colleagues would do with it.

```
// Configure the replica with the fields from the customer case and problem


// EntityType is required to distinguish between a customer case and a problem (or anything else)
replica.customKeys.entityType = entityType

if(entityType == "customerCase") {
```

```
    replica.key             = customerCase.key
    replica.summary         = customerCase.short_description
    replica.description     = customerCase.description
    replica.attachments     = customerCase.attachments
    replica.comments        = customerCase.comments
    replica.status          = customerCase.status

}


if(entityType == "problem") {
    replica.key             = problem.key
    replica.summary         = problem.short_description
    replica.description     = problem.description
    replica.attachments     = problem.attachments
    replica.comments        = problem.comments
    replica.status          = problem.status

    // the URN of the customerCase is required to look up the corresponding Epic

    replica.customKeys.parentURN = problem.parent


}
```

Configure the incoming Processor on the Jira Side

Whenever a message is accepted on the Jira side, in the case of Exalate, an Epic needs to be created:

```
/*
** if firstSync is true, then an issue needs to be created in the right project and the right type
```

```
** The project is 'CCSNOW' (customer cases from ServiceNow)
**
** Depending on the case, create an epic, a story or a task (in all other cases)
** The epic name is the key of the customer case
**
**
** In case of a problem, the twin epic (which corresponds to the customer case) needs to be found
** The nodeHelper.getLocalIssueFromRemoteURN method is being used.
*/


if (firstSync){
   issue.projectKey = "CCSNOW"

   switch (replica.customKeys?.entityType) {
          case "customerCase":
               issue.typeName = "Epic"

               issue.customFields."Epic Name".value = replica.key
               break

        case "problem":
            issue.typeName = "Story"

            // the getLocalIssueKeyFromRemoteURN is returning a BasicHubKey with has the URN as one of its at
            def parentIssueURN = nodeHelper.getLocalIssueKeyFromRemoteUrn(replica.customKeys.parentURN,"custo
            if (parentIssueURN) {

                // the Epic Link custom field requires an issue object
                def im = com.atlassian.jira.component.ComponentAccessor.getIssueManager()
                def localIssue = im.getIssueByCurrentKey(parentIssueURN)
                issue.customFields."Epic Link".value = localIssue
            }
```

```
            break
        default:
            issue.typeName = "Task"
            break

    }
}
// set all other fields

issue.summary      = replica.summary
issue.description  = replica.description
issue.comments     = commentHelper.mergeComments(issue, replica)
issue.attachments  = attachmentHelper.mergeAttachments(issue, replica)
```

Jira to ServiceNow Configuration

The specifications require that the correlation.id and issue link are set to the remote issue key and a link to the Jira link. This requirement can be implemented as long as the Jira issue key is known on the ServiceNow side. The issue key is part of the message sent from Jira to ServiceNow, so no modification is required for the outgoing sync on the Jira side

In the incoming processor on the ServiceNow side - the following statements need to be added:

```
...
if (entityType == "customerCase") {
    …
    customerCase.correlation_id = replica.key
    customerCase.u_issue_link = “https://targetjira/browse/“ + replica.key
}

if (entityType == “Problem”) {
    …
```

```
  problem.correlation_id = replica.key
  problem.u_issue_link = “https://targetjira/browse/“ + replica.key
}
```

Note: To automatically transfer the issue key after the object has been created on the Jira side, it is important that this information is sent back from Jira once the issue has bee created. The syncHelper.syncBackAfterProcessing() method allows us to trigger this functionality.

Wrap it up

The proof of the pudding is in the eating! So set up the synchronization yourself and try it out. and let us know if you have any questions or comments below.