Jira and Azure DevOps Integration

Last Modified on 12/05/2024 10:04 am EST

Software developers using Azure DevOps to handle operations can integrate with other team members using Jira in order to streamline collaborations, progress updates, and data exchange.

To integrate Jira and Azure DevOps, you need to install Exalate on both systems.

Exalate supports Jira Service Management, Jira Software, and Jira Work Management.

You can install it on Jira Cloud or Jira on-premise or even set up a private connection in case any of your instances are behind a firewall.

You can also deploy Exalate for Azure DevOps and Jira on Docker.

Exalate Installation and Configuration Steps

Here is a brief overview of a Jira Azure DevOps configuration.

1. Install Exalate for Azure DevOps from the Visual Studio Marketplace. Or you can request a trial from our integrations page.
2. Install the Exalate app on Jira. You can start from the Atlassian marketplace.
3. Choose the connection type: Basic or Script.

To learn how to set up a connection between Jira and Azure DevOps using the Script mode, go to the Getting Started guide for a detailed visual breakdown.

Just choose the two platforms you want to configure, and you'll get a comprehensive explanation of how to proceed.

Advanced Jira and Azure DevOps Integration Use Cases (+ using AI Assist)

Exalate allows you to explore multiple advanced use cases thanks to its AI-powered scripting engine embedded in the sync configuration panel.

It also provides several Script Helpers to reduce the effort of scripting connections from scratch. For a step-by-step breakdown, check out a detailed Jira Azure DevOps integration guide.

Here are some Jira Azure DevOps integration use cases.

Use Case 1: Map Jira Issue Types to Azure DevOps Work Items

When a user creates a task, epic, or feature on the Azure DevOps side, it should be mapped to a story, task, or bug on the Jira side. You also need to create a 3-level hierarchy between all three work items.

Enter the code snippet or script mapping you want for your use case, or use AI Assist to generate the code by typing in a detailed prompt describing what you want to sync.

> *"I want to create a mapping so that when a user creates a task in Jira, it is mapped as a feature in Azure DevOps, when a user creates a story in Jira, it is mapped as an Epic in Azure DevOps, and when a Jira bug is created, it is mapped as an Azure DevOps task."*

Go through the generated output to confirm if it aligns with your expectations. You can continue refining the prompt until you get what you need.

Here is a snippet from the generated code:

```
def issueMap = [
       "Epic" :"Story",
       "Feature" :"Task",
       "Task" :"Bug"
       ]
   issue.typeName  = issueMap[replica.type?.name]
```

Click Discard if the generated code is incorrect. If the generated script is correct, click Insert Changes. Once you're satisfied with the scripts, click Publish to save and implement changes.

Note: Review the AI Assist prompting guidelines to improve your prompts and, thus, the output.

Use Case 2: Map Statuses Between Jira and Azure DevOps

Getting the status of work items and issues aligned will help deliver updates instantly across both systems without manually forwarding updates to both sides.

Here is a sample prompt for this use case in the Jira incoming sync textbox:

> *"I want to map the state of Azure DevOps work item to the Jira status to reflect "New" as "To Do" and "Resolved" as "Done".*

Here is the code snippet generated by AI Assist.

```
def statusMapping = ["New":"To Do", "Resolved" :"Done"]

def remoteStatusName = replica.status.name

if (statusMapping. containsKey (remoteStatusName)) {

issue.setStatus (statusMapping [remoteStatusName] )}
```

Use Case 3: Sync CreatedDate Field Between Jira and Azure DevOps

The CreatedDate field shows the date a specific issue was created for accountability and easy tracking. This field is read-only, but you can map it to the description, comments, or a custom

field.

To map the CreatedDate of a Jira issue to a custom field in Azure DevOps, here is a sample prompt to provide in the Azure DevOps Incoming sync textbox:

> *"I want to sync the created date of the Jira issue to a custom field named "Original Created Date" in Azure DevOps."*

Here is the code snippet generated by AI Assist.

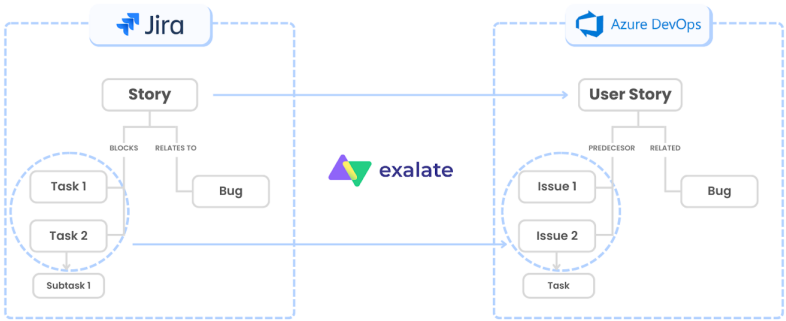
```
workItem.customFields."Original Created Date" = replica.created
```

For the Jira Outgoing sync rules textbox, here is the code snippet generated by AI Assist.

```
replica.created = workItem.createdDate
```

Use Case 4: Maintain Issue Links, Relations, and Sub-task Mappings Between Jira and Azure DevOps

In this use case, both sides try to maintain a link between work items and entities in order to make them easier to trace.



So, stories in Jira should arrive as user stories (work items) in Azure DevOps that show links to tasks and bugs. This relationship should be replicated for stories, bugs, tasks, and sub-tasks in a coherent hierarchy that makes sense to first-time users.

You can prompt the AI on the Azure DevOps incoming side using this example:

> *“Jira has the following hierarchy: Story is the parent issue. Tasks and Bugs are child issues of the “Story”. Subtasks are the child issue of “Tasks”.*
>
> *Azure DevOps has the following hierarchy: User story is the parent work item. Issues and Bugs are the child work items of User stories. Tasks are the child work items of issues.*
>
> *As per this mapping “Story” in Jira should be reflected as a “User story” in Azure DevOps, Tasks and subtasks in Jira are issues and tasks in Azure DevOps. Make sure the hierarchy is maintained.*
>
> *Also, a Bug in Jira is mapped to a Bug in Azure DevOps. Make sure the hierarchy is maintained. "*

The output could contain this fragment:

```
..
workItem.parentId = null

if (replica.parentId) {

  def localParent = syncHelper.getLocalIssueKeyFromRemoteId(replica.parentId.toLong())

  if (localParent)

    workItem.parentId = localParent.id

...
```

Use Case 5: Syncing Epics from Azure DevOps to Jira Cloud

Let’s say you want to transfer an Epic from Azure DevOps and make some of the content from its custom fields appear on the related Jira Cloud Epic. The AI prompt for scripting this connection could look like this on the Jira Incoming side:

> *“I want to transfer an Epic from Azure DevOps into the Jira project named “TS2”. If the work item type from Azure DevOps is Epic then create a Jira issue type “Epic”, otherwise keep the*

same issue type as the work item type. If the work item type is an Epic then also map the summary of the work item to a custom field value in Jira called “User Info”.

The generated results will contain this snippet:

```
if (firstSync)
  issue.projectKey = nodeHelper.getProject(replica.project?.key)?.key?: "T
S
 if (replica.type?.name == "Epic") {

   issue.typeName = "Epic
   issue.customFields."User Info".value = replica.summary
 } el
   issue.typeName = nodeHelper.getIssueType(replica.typeName)?.na
}
}
```

You can discard, accept, or refine the prompt to nail down the specifics of your use case.

AI Assist, like any other AI, can make mistakes. So, try to be as precise and detailed as possible with your prompts.

***Note**: The code snippet might not work precisely as intended due to changes to the environment or other reasons. If you encounter any problems, reach out to us for clarification.*

Automate Jira Azure DevOps Integration Using Triggers

Triggers help to automate the sync based on the conditions you provide. Exalate uses platform-specific triggers at all integrating ends.

Exalate uses Jira Query Language (JQL) to set trigger conditions on the Jira side.

```
project = DEMO AND labels = vital
```

Any project with the name “DEMO” and the label “vital” will be synced automatically.

You can use Work Item Query Language (WIQL) in Azure DevOps to specify the filter query.

```
[System.CreatedDate] = @today-2
```

All Azure DevOps records opened two days before the specific date will be synced automatically to the Jira issue.

Supported Jira and Azure DevOps Entities

You can sync several work item entities or issues between Jira and Azure DevOps. Check out the comprehensive list of supported Azure DevOps entities.

Check out the comprehensive list of supported Jira Cloud and On-premise entities.

This is a sample mapping between Azure DevOps work items and Jira issues:

Azure DevOps work item <> Jira issue

- title ↔ summary
- priority ↔ priority
- state ↔ status
- assignedTo ↔ assignee
- comments ↔ comments
- attachments ↔ attachments
- tags ↔ labels
- custom fields ↔ custom fields
- any field available via REST APIs

Video Tutorials

- Watch the configuration tutorial for Jira and Azure DevOps.
- Watch the installation and integration tutorial videos for all connectors.

Other resources

- Download the Jira Azure DevOps integration eBook.
- For any help or support for your Jira Azure DevOps integration use case, reach out to our integration engineer.
- Talk to Aida, your AI-powered integration sidekick, and get answers to your questions faster.
- Check out the detailed security and architecture whitepaper.
- Visit the Exalate Academy to get access to learning materials.
- Subscribe to Exalate Hack to get email updates and expert tips about the product.

ON THIS PAGE

Exalate Installation and Configuration Steps

Advanced Jira and Azure DevOps Integration Use Cases

(+ using AI Assist)

Product

Automate Jira Azure DevOps Integration Using Triggers

About Us

Supported Jira and Azure DevOps Entities

Release history

Glossary

Video Tutorials

API Reference

Other resources

Security

Pricing and Licensing

Resources

Subscribe for a weekly Exalate hack

Academy

Blog

YouTube Channel

Ebooks

Still need help?

Join our Community

Visit our Service Desk

Find a Partner