



How to Implement a Jira Migration (a 2023 Step-by-Step Guide)



Table of contents

Types of Jira Migrations

- 1. The Big Bang approach of migrating Jira
- 2. A Project-By-Project approach to migrate Jira
- 3. An issue-by-issue approach or a Live Jira Migration

The dangers of migrating Jira

- 1. Losing valuable time when preparing for the migration process
- 2. Missing crucial details when migrating Jira to a new server
- 3. Losing crucial data after Jira has been migrated

Why we prefer a live migration for Jira

How to implement a Jira to Jira live migration

- Step 1: Install the Exalate migration tool
- Step 2: Install the Exalate synchronization tool on the other instance
- Step 3: Connect the Jira instances
- Step 4: Create the project in the new instance
- Step 5: Configure the connection for migration
- Step 6: Create the migration trigger
- Step 7: Start migrating the Jira issues
- Step 8: Transfer the whole team by providing training
- Step 9: Stop migrating issues



- Step 10: Dismiss the old system

Conclusion



*The following article contains everything you need to know about the best practices and dangers of a Jira migration. At the end of this article, I have also included a step-by-step guide on how to actually implement the migration. I have come to understand migrations are virtually inevitable. Your business and organizational structure will evolve over time. This might be due to changing technical environments, the merging and splitting of teams or even the sale of the entire company via an acquisition. Whatever the reason, at some point you will have to migrate the Jira instance you're working in. In this article, we'll provide tips on the best approach for a secure Jira migration and a step-by-step guide on how to achieve this. **A side note:** We want to be as clear as possible on how to migrate Jira to a new server. That is why this article can get a little technical towards the end. So the end of the article might be mostly useful for your administrator. However, if you're just using Jira as a user, (but not as an admin) this article will still be valuable. If you're the Jira admin, great! Otherwise I'd recommend you share this with your admin as well. He (or she) will definitely thank you for it! Now, let's get into the nitty gritty of the challenge that is: implementing a clean Jira migration.*

Types of Jira Migrations

There are several types of Jira migrations, each of which come with their own pros and cons.

1. The Big Bang approach of migrating Jira

If you follow a Big Bang approach, it means you'll be migrating Jira to a new server in one go. You will achieve the migration in a single step. Going forward, all access to your old system is redirected. So there can be no confusion as to where updates should be logged. Because of these factors, a Big Bang migration can look superficially attractive. However, it will mean downtime for users, and several other major drawbacks that we'll discuss in a minute. So keep reading...

2. A Project-By-Project approach to migrate Jira

This approach means migrating projects, as required. This is typically done when a project has been maintained on one Jira instance

but then needs to be moved to another. This might be due to reasons relating to infrastructure, security or practicality. When the migration is launched, access to the project is stopped and all data (including configuration information) is sent over to the new instance in one go. Users can then be informed that the project is available on the new instance. This approach can be used to gradually migrate a complete configuration to a new environment. And while it is a more measured approach, it nevertheless comes with many of the same problems as a Big Bang migration...

3. An issue-by-issue approach or a Live Jira Migration

Following this approach, issues are migrated from one instance to another as required. So issues are simultaneously available on both platforms. Team members can then work on either platform and gradually switch over from one to the other. This is considered by many the most flexible and most foolproof approach, by some margin.

The dangers of migrating Jira

A Big Bang migration sounds simple, but it can come with some major issues. These include:

1. Losing valuable time when preparing for the migration process

time to migrate Jira A great deal of preparation is needed to ensure that the new system meets all operational requirements from the users and teams who depend on the system. You can not underestimate this! In many cases, people need to dedicate significant amounts of time to confirming that the new system is fully customized to their needs. And, all too often, they'll still miss crucial details.

2. Missing crucial details when migrating Jira to a new server

During the transition period, you will have to factor in downtime during which neither Jira instance is operational.

Downtime during Jira migration You will likely want to schedule this for a low-traffic period, such as a weekend, with an advance announcement of the planned downtime. However, if the migration fails, then everything needs to be rescheduled and

repeated. Trust me, that's not fun! You will have to follow up with further testing with repairs to the migration process. You'll then need to do this again and again until you get it right. This is the kind of big bang migration that you really don't want.

3. Losing crucial data after Jira has been migrated

When the moment of truth arrives, and staff begin really using the new environment, then no rollback is possible. At this point, you'll conclusively find out whether you've done enough preparation. If one team finds a blocking issue, while other teams are happily using the new system, then the administrator hits a wall. [Crucial data when Jira has been migrated](#) The dissatisfied staff will have to either wait for the functionality to be built or for a workaround to be devised for their issue. But failing that, they'll have to make do without it. As with the real Big Bang, you can't do a re-run...

Why we prefer a live migration for Jira



2 words. Less friction. When migrating Jira to a new server, running a Live Migration removes nearly all of the problems and friction that you might face. With a Big Bang migration some degree of dissatisfaction is almost inevitable. Staff want their environment to continue to work as they know it. And introducing significant changes will always have an impact on operations, however big or small. That might make your staff cringe. A little bit like this: [Avoid staff problems by doing a live Jira migration](#) Given limits to spec'ing and testing, there is a very good chance that at least some features and functionality will be missing, altered or broken. What's worse, this will often only be revealed once your users pick things up on the other side of a Big Bang migration. Our view is that bringing about change in an organisation is best done by making incremental improvements to the environment without causing disruption. A Live Migration achieves just this, allowing for projects and issues to be gradually transferred from one system to another. This makes the process simple and painless. Staff can then choose to work in either system, as they want. Which makes everybody happy again: [Jira 2 jira has been migrated safely.](#) [Synchronization of data](#) (using tools like [Exalate](#)) ensures that users will find what they need, as they need it, on both platforms. Small changes can then be applied to the new system to make it just perfect. **All the while your staff can keep using both systems in their day-to-day operations.** If something doesn't work to their needs, then they can continue to use the original system while the configuration is adapted on the new environment. Once the new

configuration is perfected, the whole team can make the switch. Only when staff is completely satisfied the old configuration can be dismissed. By building this flexibility into the transition, the migration is ultimately made far easier and far safer.

How to implement a Jira to Jira live migration

At this point you know about the types of migrations and the dangers they come along with. Now it's time to show you exactly how to migrate your Jira safely to a new server (or cloud). A heads up, this part is a bit technical. So if you're not a Jira administrator, be warned. For reasons, we explained above, we will be doing a live migration. In this case we'll be taking an example for migrating Jira to another Jira. We'll be migrating all agile information within the projects. This includes: sprints, epics, all issue-data including issue key, change history, issue links, sub-tasks, typical issue custom fields, etc. . In order to implement the live migration, we use [Exalate](#). Exalate is an [add-on for Jira](#) that allows you to easily synchronize issue data in real-time. You can trial Exalate for free for 30 days. That should be more than enough time to complete the migration. So let's get to it!

Step 1: Install the Exalate migration tool

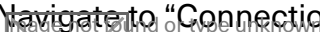
First you'll have to download Exalate. To do this, head over to the [Exalate page on the Atlassian Marketplace](#). Once there, click "Try it free".  Then choose if you would like to install it for Jira Server or Jira Cloud.  After this, go ahead and complete the installation on your [Jira](#) instance. If you'll need more details on setting up Exalate, you can have a look at the following [documentation page for Jira Server](#) or the following [documentation page for Jira Cloud](#).

Step 2: Install the Exalate synchronization tool on the other instance

In order to implement the migration, you will need to have Exalate installed on both the initiating instance as well as the destination instance. So repeat step 1 for your other instance as well. Note that if more than 2 instances will be involved in the migration, just set up

Exalate for any of the other instances as well.

Step 3: Connect the Jira instances

Now we'll configure a connection between the instances using Exalate. Exalate connects instances based on invitations. This means you'll have to initiate your connection on one side after which you will receive an invitation code on the other side. Don't worry, this is easier than it sounds. I'll walk you through it.  Navigate to "Connections" in the left pane and click "Initiate connection" (this is also

clarified on [the Exalate documentation here](#)).



After that you'll have to select your "Connection type".



Note: We'll

assume the destination instance is accessible from this side. But Exalate also provides options for migration projects in one instance and migrating to an instance which is not directly accessible from this Jira. Click "Next". Now you'll be asked

to share the url of the destination instance. Add the url and click "Next".



After that, select the "basic" project configuration.

This will let you migrate all basic issue data like: summary, description, type, assignee, reporter, comments, attachment and labels. If



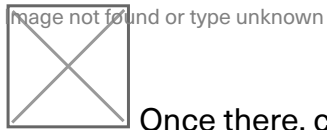
your migration would require more advanced configuration, we'll edit that later.

Now you will have to add a "Connection

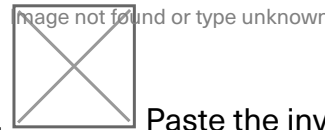
name". This will just be used for you to identify this connection-configuration in the future. The same name will apply on both sides of



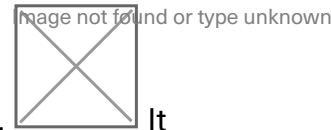
the migration. Alright, well done! Now a script will be generated to accept the invitation on the destination instance. Once that is accepted on the other side, you will be able to start the live migration of the Jira issues. So copy the script to your clipboard and move on to the other instance (or send it over to whoever has access to the new instance).



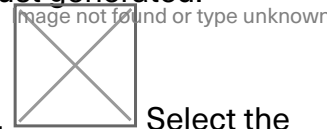
Once there, click "Accept invitation" in the Connections tab.



Paste the invitation code you just generated.



It will be automatically validated. Click "Next" to proceed. And choose the "Basic" project synchronization again.



Select the



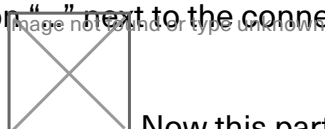
project name again where you'd like to migrate the issues to. Awesome, now the 2 instances are connected!

Step 4: Create the project in the new instance

You'll need to make sure there is a project the issues can migrate to. If you're migrating to a completely new instance, I'd recommend creating a Jira project with the exact same name as the project you'd like to migrate.

Step 5: Configure the connection for migration

After the connection is set, you'll still need to do some minor configuration to the synchronization rules. This way we'll make sure the connection is correct for your specific migration use case. To access the configuration, head over to "Connections" (from the left menu). Once there, click on "-" next to the connection you would like to configure and click "Edit". After that, go to "Sync rules". This is



where you should end up: Now this part might be a little tricky, so stay with me. Here you will have access to the scripts of:

1. Datafilter (Outgoing migration)
2. Create processor (Incoming sync for new issues you would like to migrate)
3. Change processor (Incoming sync for existing issues you would like to change)

You will have to copy/paste new scripts into each one of these. The scripts will differ based on if you're migrating from cloud to server, or from server to server or to cloud and so on. In the following documentation pages you will find the scripts for the "Data Filter", "Create Processor" and "Change Processor". You will find them for both your initiating instance as well as your destination instance.

- [Migrate Jira Server to Jira Server](#)
- [Migrate Jira Cloud to Jira Cloud](#)
- [Migrate Jira Server to Jira Cloud](#)

So copy the pages

Step 6: Create the migration trigger

Now we'll need to create a "Trigger". This will be super easy. The "Trigger" will determine when issues will be migrated/synchronized to the other side. As well as choose when not to migrate issues. Of course, the second choice is completely optional. To create the trigger, navigate to "Triggers" in the left side-pane. Here you'll want to choose to sync certain projects. Just add that project to the

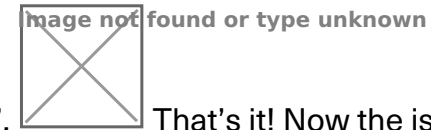


trigger like so: Again, in this example, I decided to call the project "scrumdemo", but yours will have a different name. **Tip:**

You can select issues manually you do want want to sync, by adding a label in the trigger. For example, choose to not migrate issues with the label "nosync" or empty labels. This is the line should add example above: "and (labels != nosync or labels is empty)". That makes the final Trigger: "project = scrumdemo and (labels != nosync or labels is empty)". After that select the connection-name you've made. In the example above I created a connection called "red-blue". If you're satisfied with the trigger-rules, click "Create".

Step 7: Start migrating the Jira issues

After you've managed to successfully create the trigger, you'll be able to start migrating issues. To achieve this, click on the "..." button



in the Action column next to the correct trigger. There you'll be able to select "Bulk Exalate". That's it! Now the issues in the project you've selected will be automatically synchronized to the to the other side. You can check the "Sync queue" to see if any errors are occurring. Everything should be in order. But if you are receiving error notifications, just talk to support through their live chat on the right.

Step 8: Transfer the whole team by providing training

After migration has been completed, it's possible your team will require some time to adjust to the new system. In order to streamline the efficiency in using the system, I'd recommend dedicating some resource to training staff. This will save you valuable time, human errors and all-round confusion in the end.

Step 9: Stop migrating issues

When all of your data has been transferred and teams are familiar with the new system, you can stop the synchronization. To stop migrating new issues, navigate to "Support Tools" in the left side-pane. Than select the connection used for migrating Jira end click

"Remove".



Step 10: Dismiss the old system

Is everything running smoothly now? Is staff getting adjusted with the new Jira? If so, it's time to pull the plug on the old system! Congratulations, you've completed a smooth transition from your staff to your new environment. Hope it was a smooth ride. Kudos!

Conclusion

Performing a migration from one system to another is never easy. Fundamentally, though, the challenges in the transition are not just about technology. It's about the people using it and the way they work. That's why it's crucial to ensure the new configuration completely meets their needs. The problem here is that getting user feedback is always difficult. Users will rarely have enough time to explain their requirements in the necessary detail or validate new software to the degree that is required. Because of this, a Big Bang migration may well be followed by a 'WTF' moment. This usually only happens when staff has to put the new instance to use. And at this point you might be left without having the option to roll things back. The gradual transition made possible by a Live Migration, on the other hand, allows for an iterative and controlled evolution of the environment. That's why we recommended this approach by some way for most use cases. *How about you? Did you find this article useful? Did you have any "cringy" migration stories people could learn from? Share your thoughts in the comments below!* **Recommended Reads:**

- [A guide to Jira workflow best practices \(with examples\)](#)
- [Dissecting Jira pricing: How much does a Jira license cost?](#)
- [Getting started with Trello: A Comprehensive Guide](#)
- [How to set up the perfect Jira notification Scheme](#)
- [Jira Bitbucket Integration: the complete guide for 2020](#)
- [Jira Confluence Integration: The complete 2020 guide](#)
- [Key Updates of Atlassian Products 2020 \(what you should know\)](#)
- [Why Jira is better than Trello, even for non-developers](#)
- [10 Expert Tips to 10x your Productivity in Jira](#)