

# Usage tips to improve performance

- [Overview](#)
- [Background](#)
- [Common Tips](#)
  - [Jira specifics](#)
  - [Process](#)
    - [Specification](#)
    - [Organizing](#)
    - [Planning](#)
    - [Execution](#)
  - [Entities](#)
    - [Tests](#)
    - [Pre-Conditions](#)
    - [Test Sets](#)
    - [Test Repository](#)
    - [Test Executions](#)
    - [Test Plans](#)
      - [Test Plan Board](#)
  - [Integrations](#)
    - [Automation & Continuous Integration](#)
    - [REST API](#)
  - [Reporting](#)
    - [JQL](#)
    - [Xray calculated custom fields](#)
    - [Reports and Gadgets](#)
    - [Dashboards](#)
    - [Custom Reports](#)
  - [Administration and Customization](#)
    - [Settings](#)
    - [Your own custom fields](#)
    - [Workflows on Xray entities](#)
    - [Reindex](#)
  - [Other operations](#)
- [Data Center Tips](#)
- [References](#)

## Overview

This document provides useful tips to improve the performance of your Jira instance while using Xray; tips cover normal usage, configuration and setup. It assumes that you have some Jira and Xray background.

As Xray is built on top of Jira, the first thing you need to make sure is that your Jira instance is properly configured and tuned beforehand. Thus, we recommend looking at this [Jira specifics](#) section first.

After ensuring your Jira instance is fine-tuned, then you may proceed to [Common Tips](#) as they apply both to Server and Data Center based deployments. At the end of the document, you have some specific [Data Center Tips](#) that complement the previous section.

## Background

Xray mainly uses Jira issue types for implementing Test Management related entities, such as:

- **Test** - for specification;
- **Pre-Condition** - for complementing the specification of one or more test cases;
- **Test Set** - for organizing Tests in lists;
- **Test Execution** - for scheduling executions of tests;
- **Test Plan** - for defining the test plan against some scope (e.g. version/sprint) and track its progress.



### Please note

The number of projects or amount of issues in Jira by themselves may not affect at all Xray's performance. From Xray's perspective, performance will depend on usage patterns in terms of testing, including the amount of Test Runs and on-demand reporting.

The only relevant exception to use issue types is Test Runs. A Test Run is an internal entity managed by Xray itself, which is an instance of a Test containing also its result related data.

Every time a Test is executed within some Test Execution context, a Test Run is created containing a copy of the test specification along with the recorded results.

One of the key features of Xray is its coverage analysis capability, which is something quite powerful that most tools are unable to provide. In brief words, Xray provides the ability for you to evaluate in real-time how a given Test or a given "requirement" is for a specific context (i.e. version, version+environment); Xray calculates this based on the latest consolidated results for each context. This means that you may evaluate how a Test or a requirement is for different versions, for example, because for each case it will only consider the testing results obtained for that version.

Thus, Test Runs are a potential variable affecting performance as they will have to be analyzed and consolidated to compute the coverage status depending on the context.



Status for Tests and "requirements" is shown in several places, namely in reports such as Traceability and Overall Coverage (more info ahead on where and how to perform [Coverage Analysis](#)).

If you want a more and in-depth explanation of how coverage works, please see [Understanding coverage and the calculation of Test and requirement statuses](#).

## Common Tips

This document sums up tips to keep the performance of Xray and JIRA as optimized as possible.

The tips are grouped by area or topic; therefore, they do not follow any specific order. However, you should start by looking at the [Jira specifics](#) and [Process](#) related tips first.

Within each area/topic, tips will be presented by descending order of risk; each tip is preceded by an icon that corresponds to the risk level.

### Legend:

- - major
- - highest
- - high
- - medium
- - low
- - lowest

## Jira specifics

Xray is built on top of Atlassian's Jira, therefore it is mostly dependant on the architecture and technologies followed by Jira itself.

Atlassian provides some [performance tips related to scaling](#), that this also applies to Xray.

Large organizations or organizations with huge amounts of data and/or many users should consider [Jira Data Center](#), which increases performance and improves throughout higher concurrency usage scenarios; it also provides high-availability for critical scenarios.

There are some general Jira administration related tips that you may want to consider; there are many, so please consider these just as a starting point:



1. Moderate the usage of custom fields
  - a. Managing custom fields in JIRA effectively;
  - b. Optimizing custom fields;
2. Moderate workflow usage, in terms of their complexity (i.e. the number of workflow steps);
3. Evaluate the plugins/apps you need as some may impact performance.

## Process


1. "Unified" development process: define a process that can be applied to all teams in a way to manage the STLC. Every team is different and has its own needs, therefore your process should not be too strict but it should provide some guidance on how development life cycle should be addressed, covering requirement management, bug management and test management. Having teams working completely in different ways hardens communication and leads to improper and unoptimized tool usage. If you have a well-defined process that can be used organization-wide, this is better, because this is the key to ensure optimal usage and best performance.
2. Are you adopting Agile and Scrum? Check out [Using Xray in an Agile context](#) for tips on how you can take advantage of Xray in such scenarios; the [Agile software development](#) page provides high-level overview of Agile and Agile Testing and besides background information on them, it will also provide some useful tips so your team can be more Agile and avoid doing things that are unnecessary.
3. Each Xray entity has a purpose/fit that you should try to take advantage of. You're not obliged to use all of them; you can even choose to use some over other ones. In order to have an optimum usage of Xray, we would recommend understanding, first of all, the purpose of each entity.

Entity / Issue Type	Purpose
<b>Test</b>	For making the specification of some test; a test case template.
<b>Pre-Condition</b>	For abstracting some initial condition that one or more tests must assure; reusable, i.e. linked to one or more test cases.
<b>Test Set</b>	For creating lists of test cases, so you can easily pick those test cases afterwards in case you need them.
<b>Test Execution</b>	For scheduling an execution of a bunch of test cases in some version&revision of the SUT. A Test Execution contains several Test Runs, one per each "linked" test case.
<b>Sub Test Execution</b>	Similar to a Test Execution; the difference between them is that the Sub-Test Execution is a sub-task and can be created within the context of a requirement.  Creating a Test Execution as a sub-task of the requirement issue provides you with the ability to track executions in the Agile board.
<b>Test Plan</b>	For grouping multiple Test Executions and presenting a consolidated overview of them; tracks the results of some Tests in some version/sprint of the SUT.
<b>Test Run</b>	An instance of a Test in the context of some Test Execution; contains a copy of the original Test specification along with the recorded results. <u>It's not an issue type</u> .
<b>Test Repository</b>	A per project hierarchical organization of Test cases using folders; an <u>alternate</u> approach to Test Sets for organizing test cases.
<b>Test Plan Board</b>	A per Test Plan hierarchical organization of Test cases, at the planning phase, using folders.  It is used for... <ul style="list-style-type: none"> <li>• grouping, organizing the tests in the context of the Test Plan and to easily track the results of certain Tests grouped in some folder;</li> <li>• easily changing the ranking of the Tests, to create Test Executions for them afterwards.</li> </ul>



## Specification

-  Avoid having many sub-requirements (>500) per requirement (e.g. Stories per Epic) as it can impact the calculation of their statuses
  - normally it is a signal that the requirement needs to be further decomposed. Besides hardening analysis and its management, it will also require additional resources during computation of its status upon changes in any of the related sub-requirements, that in turn are affected by the status of the related Tests.
-  Requirements being covered by many (>>100) Tests
  - normally it is a signal that the requirement needs to be further decomposed. Besides hardening analysis and its management, it will also require additional resources during computation of its status, that in turn is affected by the status of the related Tests.


## Organizing

-  If you have thousands of Tests, using the Test Repository approach may provide **additional benefits** over using "lists" (i.e. Test Set issues), as it makes management of test cases easier while avoiding the creation of issues.

## Planning

-  Instead of creating one Test Plan for your release, you may create multiple Test Plans to track different Tests (e.g. manual vs automated or regression vs NRT); this may be useful if you want to have clear visibility of how certain groups of Tests are progressing and if their execution life cycle is different from other ones. It will also make your Test Plans considerably lighter.
-  If adopting Scrum, create Test Plans per Sprint, to track the testing being done in the scope of your Sprint; manage them as artefacts of your Sprint and add them to your Scrum boards so everyone sees their progress. Per each Sprint you may have more than one Test Plan; check out some possible usage patterns [here](#).

## Execution

-  Don't create dozens or hundreds of Test Environments; don't try to do data-driven testing using test environments
  - It will impact the calculations that need to be done and the size of the Lucene index.  
Test Environments should be used as a means to identify different testing stages, different browser vendors, different mobile devices; the number of environments should be well-defined and limited.

## Entities

## Tests

1. ⚠️ Don't add many custom fields to Tests, especially if they're calculated, as it will add some additional overhead to Jira.
2. ⚠️ We recommend up to 5 linked requirements per each Test; ideally, a Test should be focused on the validation of one requirement.
  - This recommendation helps improve performance both on the TestRunStatus and Requirement Status calculations and also when loading the issue screens and reports.
3. ⚠️ Promote reusability and avoid cloning Test cases, if they're the same. A Test can be reused multiple times and can be used to cover more than one requirement (if really needed), no matter in which project it is located.

## Pre-Conditions

1. ✅ Although the impact is neglectable, try to use Pre-Conditions as a means to have manageable and reusable initial conditions that you can link to multiple Tests. This will avoid creating additional steps in all those test cases.

## Test Sets

1. ⚠️ Although there isn't a hard limit, we recommend having no more than **5000** Tests in a given Test Set mostly to ease their management. This limit may be easily superseded depending on Jira instance deployment configuration.

## Test Repository

1. ⚠️ Creating a Test Plan from within some Test Repository folder that has many Tests and sub-folders can take some time if you choose to replicate the folder structure into the destination Test Plan Board. However, this overhead is just temporary.
2. ✅ Avoid many folders shown at the same time as it will impact browser performance at some time. You can do this by limiting the direct child folders you create at a given parent folder and by using the "Expand all" moderately. The amount of folders you have does not affect your Jira backend performance.

## Test Executions

1. ⚠️ Clean-up old, unneeded executions related data, to make calculations faster throughout the application and thus make reports and some panels, for example, also faster.
  - a. If your organization performs a high number of Test Executions (consequently creating also a high number of Test Runs) we recommend deleting old Test Executions issues from time to time. This recommendation applies to organizations that import many automated executions daily using the REST API.  
  
The clean-up process must be scheduled for a low Jira usage period because when Test Executions are deleted, Xray will re-calculate TestRunStatus and Requirement Status. This might temporarily slow down the Jira instance depending on the number of issues affected.
  - b. Deleting Test Runs can affect the calculated and consolidated status of your Tests and of your requirements for all scopes (e.g. versions or Test Plans + Test Environments); please proceed carefully.
2. ⚠️ Although there isn't a hard limit, we recommend having no more than **2000** Tests in a given Test Execution mostly to ease their management. This limit may be easily superseded depending on Jira instance deployment configuration.

## Test Plans

1. ⚠️ Xray provides a [setting](#) ("Max number of Tests per Test Plan") where you may define a soft limit for the number of Tests within Test Plans. Although you may adjust this value, we recommend having no more than **2000** Tests in a given Test Plan:
  - a. to ease their management;
  - b. and to make it more performant and lighter.

This limit may be easily superseded depending on Jira instance deployment configuration.  
Note that a Test Plan aggregates and consolidates the results of the related Test Executions and Tests, thus the overall number of Test Runs you'll have can add some overhead to the calculation of the consolidated results.


## Test Plan Board

1. ✅ Avoid many folders shown at the same time as it will impact browser performance at some time. You can do this by limiting the direct child folders you create at a given parent folder and by using the "Expand all" moderately. The amount of folders you have does not affect your Jira backend performance though.
2. ✅ Using the Test Plan Board as a means to do operations over certain Tests of the Test Plan (e.g. schedule Test Executions for them) and to track results can be more efficient than using the Test Plan issue screen. Although the previous two are not equivalent, the Board essentially provides the same operations while being lighter as it does not show all the information you can see in the Test Plan issue screen.





## Integrations

### Automation & Continuous Integration

1. ⚠️ Upload only relevant test results (e.g. don't upload unit test results); choose properly what testing results you want to track within Xray.
2. ⚠️ Choose properly the upload frequency

- a. Aggregate relevant results in some job run periodically (e.g. hourly, daily) and submit those.
3.  Don't mix the CI tool with TM tool
  - Leave the highly detailed execution info on the CI tool.


## REST API

1.  Review the "Max results per request" setting in the [Miscellaneous administration settings](#) as it controls the pagination on the REST API calls. The default value should be ok.
2.  Limit API calls (to Jira and Xray related endpoints) using a reverse proxy
  - a. Evaluate what REST API calls are being used, discuss their real need with users
    - i. Make sure that pagination is being used on the REST API calls
  - b. Restrict access to REST API calls
    - i. Limit access to well-known hosts/applications
3.  [Export results endpoint](#) (i.e. [/rest/raven/1.0/testruns](#)) allows you to include custom fields from the Test issues in the response, using **includeTestFields** parameters; please choose carefully what fields you choose to include, as some of these may be calculated and thus add some additional overhead to the request.
4.  Whenever searching for issues using [Jira's REST API](#) (i.e. [api/2/search](#)), please choose explicitly what fields to return using the **fields** parameter; that will avoid including unnecessary fields (e.g. "Requirement Status, Test Count, Test Set Status, Test Execution Defects, Test Plan Status) that are included by default and that add overhead to the request. This can be aggravated if this endpoint is used automatically by some integration with an external application. This is relevant for "requirement" like issues, Tests, Test Sets, Test Executions and Test Plans.


## Reporting

### JQL

Xray provides [dozens of JQL functions](#) but you have to use them carefully to make sure your instance is not affected. Please do train your users on JQL before "allowing" to use them throughout Jira.

1.  Using unoptimized JQL queries can degrade performance substantially
  - a. most times this happens because users don't understand how JQL works first of all; JQL is not like SQL (see [Understanding JQL Performance](#)). Thus, filtering issues by project by adding the *"project = <xxx>"* clause is not the same as specifying the project as argument to the subsequent JQL function.
    - i. Example:
      1. Use
 

```
issue in requirements('OK','CALC')
```
      - ...instead of ...
 

```
project = 'CALC' and issue in requirements('OK')
```
2.  Some JQL functions, such as the ones dealing with requirement coverage, may be more intensive than other ones, since Xray may have to, for example, load all the related Test Runs in order to obtain relevant data. Some care should be taken with the following JQL functions:
  - a. **testPlanTests()** - whenever by tests in a given status; the current workaround is to search using the "TestRunStatus" custom field.



#### Please note

When searching for Tests with a certain status inside a Test Plan, we recommend you to use the custom field search instead.

Xray has created a new way of searching with big improvements when filtering by test status, using the Custom Fields:

(3)

issuetype = Test and TestRunStatus = "DEMO-10 - TODO"

(4)


issuetype = Test and TestRunStatus = "DEMO-10 - TODO environment:IOS"

- b. **requirements()** - whenever filtering by dates, as shown in the following example:



```
requirements('OK',
             'Calculator',
             'v1.0',
             'chrome'
             'false'
             '2014-08-30')
```

- c. **testExecutionTests()** - it will depend on the amount of Tests you have on the Test Execution





## Xray calculated custom fields

1.  Xray provides some [specific custom fields](#) that calculate their values on the fly. This means that you should have that in mind, especially if you're including them in tables/issue listings/gadgets.
  - a. The most intensive custom field is the "Test Set Status." The "Test Count," as it does an aggregation, is also intensive if you use it for multiple issues.



## Reports and Gadgets

1.  One way of doing reporting is by using gadgets. Gadgets are great to share information between team members and even between different teams; however, if not used carefully, they can degrade Jira performance if all users have the same report on their dashboard as they will probably generate multiple requests once users access the dashboard. Thus, carefully use the most intensive gadgets such as the "Historical Daily Requirement Coverage" and the "Tests Evolution" gadget and others that do aggregations (e.g. "Test Runs Summary" gadget). Gadgets that just "list" entities should not affect performance significantly.
2.  Limit the target issues for the reports/gadgets, i.e. generate the Overall Requirement Coverage (report/gadget) just for issues that you really need and not all Jira requirements or projects; the setting "**Max number of requirements per report or gadget results**" (available under [Miscellaneous](#)) acts as a maximum limit for some reports/gadgets.

## Dashboards

1.  Choose properly the filters you use for each gadget, in order to restrict the number of issues that will be processed
2.  Don't use small (i.e. intensive) refresh times as they will add some overhead to the Jira instance
3.  Use shareable dashboards to have a high-level overview and the things that matter but avoid creating highly complex dashboards
4.  Try to normalize the dashboards and make them standard organization-wide; it will facilitate communication and avoid "wrong"/unoptimized usage



## Custom Reports

1.  If possible, use eazyBI for making custom table/chart-based reports with drill-down capabilities; it is highly flexible and works in its own database, thus not impacting Jira unless the database is hosted in the same instance of Jira server/database
2.  If using the Xporter app to build custom PDF, Word or Excel based reports, then please see [Xporter specific recommendations](#).



## Administration and Customization

### Settings


Xray comes with default settings that are ok for most organizations. Nevertheless, have a look at the different settings; it's an opportunity to know the product better and for you to think about ways to fine tune it later.

1.  Review the settings "**Max Test Runs for bulk operations**", "**Max number of requirements per report or gadget results**", "**Max number of Tests per Test Plan**", "**Max results per request**" in the [Miscellaneous administration settings](#). The default values should be ok.
2.  Using the requirement coverage strategy "Use versioned Test Sets for Requirement Coverage" will require additional computational resources on daily usage; therefore, the default "Use versioned Test Executions for Requirement Coverage" strategy is the recommended one. Learn more in [Requirements Coverage](#). If you change this setting, then it can take a while until all statuses are re-calculated depending on your instance dimension.


## Your own custom fields

1.  Beware with calculated custom fields implemented using some customization app (e.g. Jira Misc Custom Fields, ScriptRunner, etc) as they will be calculated each time they're accessed and they can impact things such as:
  - indexing time;
  - REST API calls (e.g. whenever searching issues);
  - on listings (e.g. on Issues search page, Filters Results gadget, Xray tabular sections) if they're included as columns.
2.  Adding a lot of custom fields on Jira will add some overhead on performance; therefore, Atlassian itself [recommends](#) a responsible usage of custom fields. Please see more on [Optimizing custom fields](#).


## Workflows on Xray entities

1.  Don't use complex workflows for Xray entities as it will harden their usage; if the workflows have many steps then it will impact overall Jira performance.




## Reindex

1.  Xray provides some maintenance operations that can be used under certain circumstances if needed. If you clear some Xray calculated custom fields, either using [Integrity Checker > Calculated Custom Field Values](#) option in administration settings or using the available bulk operations for **Requirement Status** and **TestRunStatus** custom fields, it will impact the reindex time considerably as the values have to be all recalculated.
  - a. Example:
    - i. GET <http://jiraserver/rest/api/2/search?jql=key=CALC-3214&fields=key,summary,description,fixVersions>

## Other operations

1.  Avoid bulk cloning Test plans or Test Executions using some specific apps for that purpose; cloning multiple issues at the same time can add some overhead to Jira as these can affect the calculations of the statuses of Tests and requirements.

## Data Center Tips

1.  Review [Atlassian's best practices for Data Center](#) and the [Node sizing overview for Atlassian Data Center](#) article.
2.  Configure your load balancer properly to use dedicated nodes for REST API calls, which can reduce the impact on other application nodes.
3. Integration with other apps
  - a.  Please see easyBI's Data Center related recommendations [here](#).

## References

- Jira in general
  - [Managing custom fields in JIRA effectively](#)
  - [Optimizing custom fields](#)
  - [Understanding JQL Performance](#)
- Data Center
  - [Best practices for Atlassian Data Center](#)
  - [Node sizing overview for Atlassian Data Center](#)
  - [Traffic distribution with Atlassian Data Center](#)
  - [Jira Data Center size profiles](#)
- Other
  - [Knowledge is Power: Visualizing JIRA's Performance Data](#)