Get the most out of your JIRA data

At ServiceClarity we believe in getting the best you can from the data that you have. In practical terms this means working with the data available in operations tools such as JIRA and JIRA ServiceDesk. Operations tools often contain the most up to date and accurate picture of your organisation and as a core workflow and resource management product JIRA is plugged directly into how your team works. When combined with ServiceClarity this data can be leveraged to provide key insights and to drive optimisation.

Quality

Use ServiceClarity and JIRA to understand the quality of your work pipeline by tracking key performance indicators (KPIs) such as ***the % of new tasks without estimates*** or ***the % of new issues being rejected***. Shine a light on the quality of your output by comparing the volume of issues found internally against the number found by your customers. Monitor your customers' experience with ***time to respond*** and ***time to resolve*** KPIs and track ***reopened issues*** to help you get it right first time.

Efficiency

Use ServiceClarity to make sure that high priority tasks are being resolved as quickly as possible. Spot early warning signs of bottlenecks by monitoring cycle times, answering questions like: "How long is work waiting for customers?", "How long are tasks waiting for QA?". Use ServiceClarity ***burndown*** and ***throughput*** KPIs to track if work arriving faster than your team can cope.

Planning

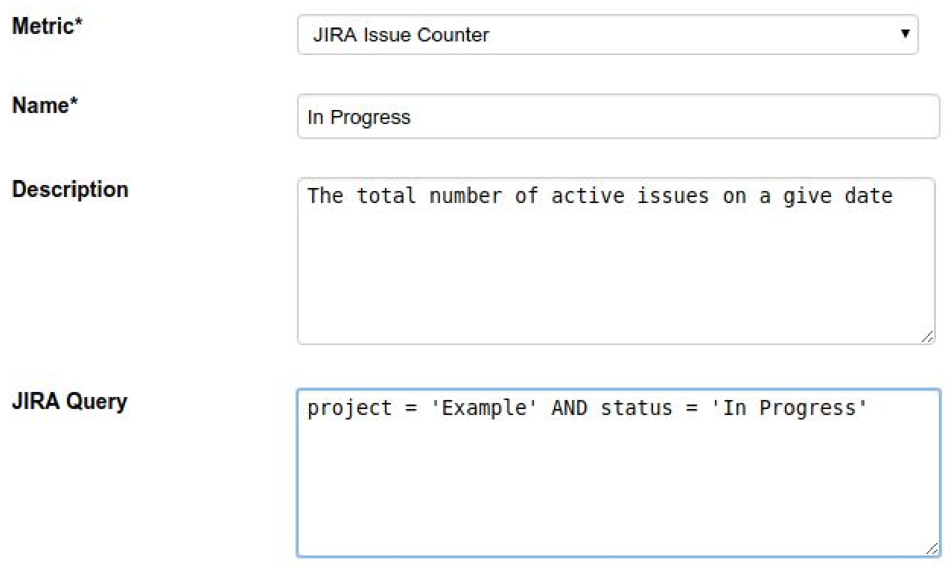
ServiceClarity with JIRA can calculate your ***work in progress*** (WIP) as well as your work throughput and ***average lead time*** helping you to plan ahead, to refine predicted delivery dates and react to changing circumstances. Use ServiceClarity to convert the ***time spent*** on JIRA tasks into costs and project these into the future to help manage budgets.

The power of JIRA's Query Language

ServiceClarity provides a set of unique data collectors specifically for working with your JIRA data. These range from simple issue counters to time trackers and also more specialised JIRA ServiceDesk SLA trackers or JIRA Agile Story Points and Epics . The value that these ServiceClarity JIRA data collectors provide builds upon the JIRA's own query capabilities: the JIRA Query Language (JQL).

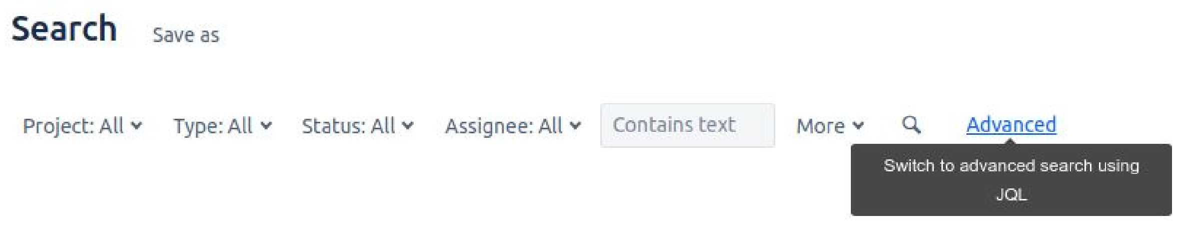
ServiceClarity leverages all of the power of JIRA's JQL search to ensure that your ServiceClarity KPIs are tracking just the right JIRA issues, enabling you to track specific projects, user groups, status changes, SLAs and Epics. In fact anything that you can find in

JIRA can be tracked in ServiceClarity by adding the JQL query to the ServiceClarity data collection configuration:

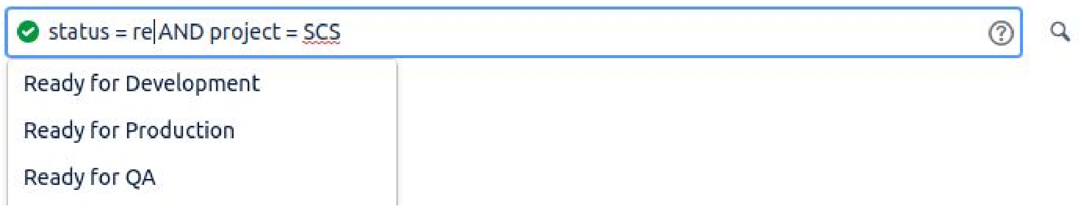


Metric*	JIRA Issue Counter
Name*	In Progress
Description	The total number of active issues on a give date
JIRA Query	project = 'Example' AND status = 'In Progress'

Although this guide is not intended as a detailed JQL reference we will look at some specific techniques and JQL queries and how they can be used within ServiceClarity. If you are unfamiliar with JIRAs search capabilities we suggest experimenting with JQL from the JIRA application. You can enable JQL search on the Issue Search page in the JIRA application to try out the powerful query language:



Once enabled the JQL search capability in JIRA provides advanced searching. JIRA JQL queries created within the JIRA application can be copied directly in ServiceClarity and used to build ServiceClarity KPIs.



```
status = re AND project = SCS
```

Comprehensive reference material on how to the get the most out of JQL is provided by Atlasstian on their support portal. We recommend the following Atlassian guide and reference documents:

- Getting Started
  https://confluence.atlassian.com/jiracorecloud/searching-for-issues-765593657.html

- Advanced Searching - functions reference
  https://confluence.atlassian.com/jiracorecloud/advanced-searching-functions-reference-765593719.html

- Advanced Searching - operators reference
  https://confluence.atlassian.com/jiracorecloud/advanced-searching-operators-reference-765593718.html

How to find the data you need

When creating ServiceClarity KPIs from JIRA data it pays to be as specific as possible with your JQL queries. This is true regardless of what you are asking ServiceClarity to track: time spent, issues closed, SLAs breached, total estimated effort, etc. The values for all of these KPIs can be affected by the selected set of JIRA issues. It is important to make sure that ServiceClarity JIRA data collection tracks the right project, the right issue type, priority and status.

For example, an important KPI available from JIRA, and a fundamental performance indicator, is a count of the number of tasks (issues, tickets, stories, etc) that are actively being worked on at any one time. This is the simplest measure of ***work in progress*** (WIP) that you can track and in the absence of detailed estimates or sizing it can be a valuable way to monitor the flow of work. The default JIRA installation and JIRA Cloud account come with a built in configuration that includes a work status called: “In Progress”. To make use of this default we could write a JQL query to find work that is currently “In Progress”:

```
status = 'In Progress'
```

Every time this JQL is run it will return the set of tasks in JIRA that currently have a status of In Progress. However, many JIRA systems contain data for multiple independent projects and the above JQL will find all tasks in all projects. We can refine this query to include only those tasks that are in a specific project:

```
status = 'In Progress' AND project = 'Example Project Name'
```

The above is a more specific query but it might still not be targeting the exact set of tasks that you want to track in ServiceClarity. For example, it is common to configure JIRA to manage multiple types of work including but not limited to: customer support issues, defects (or bugs), ongoing production tasks, design tasks, agile stories and many more. We can further refine our In Progress JQL to filter out one or more of these types:

```
status = 'In Progress' AND project = 'Example Project Name' AND issueType = 'Defects'
```

Hopefully it is easy to see how the JQL query for our ***work in progress*** KPI can be evolved to focus in on a specific area of interest.

How to write JQL to obtain historical data

In the previous section we looked at creating a JQL query that could be used within ServiceClarity to create a ***work in progress*** KPI. The example looked at filtering out work status, project and issue type to find the work that is currently active. However, unless your JIRA system or project has recently been created it will very likely contain a wealth of historical information that cannot be accessed with the previous version of our JQL example.

For ServiceClarity to look back in time at historic JIRA data we need to rethink how we select the work in progress. Fortunately there are advanced JQL features that can help us, the details of which can be found in Atlassian's support portal:

- Advanced Searching - operators reference https://confluence.atlassian.com/jiracorecloud/advanced-searching-operators-reference-765593718.html

Using the advanced JQL operator **`WAS`** … **`ON`** we can adapt our previous example to cope with finding historic data by changing the date `'2015-11-14'` in our new example JQL:

```
status WAS 'In Progress' ON '2015-11-14' AND project = 'Example Project Name' AND
issueType = 'Defects'
```

The Atlassian documentation on advanced operators includes many examples but an interesting one worth highlighting here is **`CHANGED FROM`** … **`TO`**, which can enable some useful JQL filtering within ServiceClarity.

An example of how ServiceClarity can make use of this operator is to historically track a common support KPI: ***time to respond***. This KPI calculates the time that support issues take to move from a status of "New" to the next stage of the support process, for example: "In Progress". To calculate values for the ***time to respond*** on a date in the past ServiceClarity needs to find those issues that have made that transition from "New" to "In Progress" on that date:

```
status CHANGED FROM 'New' TO 'In Progress' ON '2015-11-14'
```

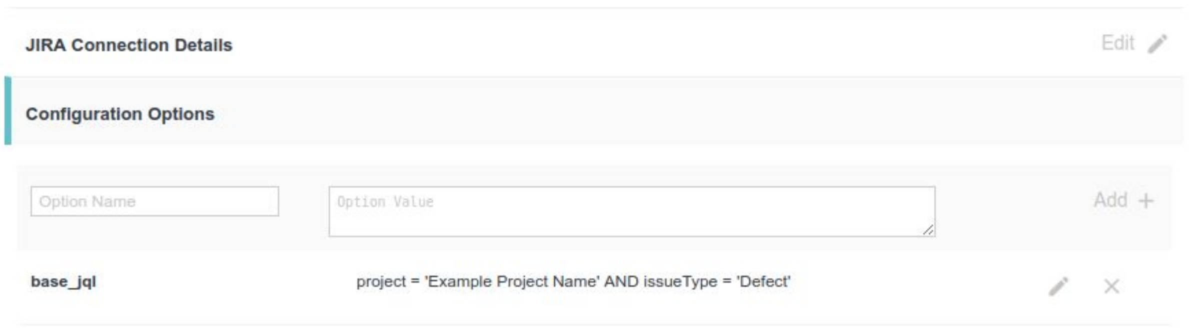
ServiceClarity JQL extensions

Variables

To help with managing complexity and promoting reuse ServiceClarity data collectors support the use of variables. You can define your own variables and share them across ServiceClarity JIRA data collectors to make working with complex JQL easier.

Continuing with our previous example of a ***work in progress*** KPI we might want to create multiple KPIs that, for our example project, track the number of new defects, closed defects, the time to resolve defects, etc. In addition our JIRA system might have more than one status that is considered “In Progress” or there may be additional condition that need added to the JQL to account for specific groups of users, product versions, etc. ServiceClarity variables can be used to help manage this complexity.

On the ServiceClarity JIRA connection page we can define a variable that incorporates a set of baseline JQL filters that we want to add to multiple JQL queries:



base_jql	project = 'Example Project Name' AND issueType = 'Defect'

Now that we have declared our **`base_jql`** variable we can rewrite our previous JQL example as:

```
{base_jql} AND status WAS 'In Progress' ON '2015-11-14'
```

When ServiceClarity requests the above data from JQL is will replace the **`{base_jql}`** part of the JQL with our declared value **`project = 'Example Project Name' AND issueType = 'Defect'`**. The immediate value of this is the reuse of this base jql for other related data collections, for example we could collect the number of new defects created, the number being closed and the total number that are unresolved.

New defects JQL:

```
{base_jql} AND status WAS 'New' ON '2015-11-14'
```

Closed defects JQL:

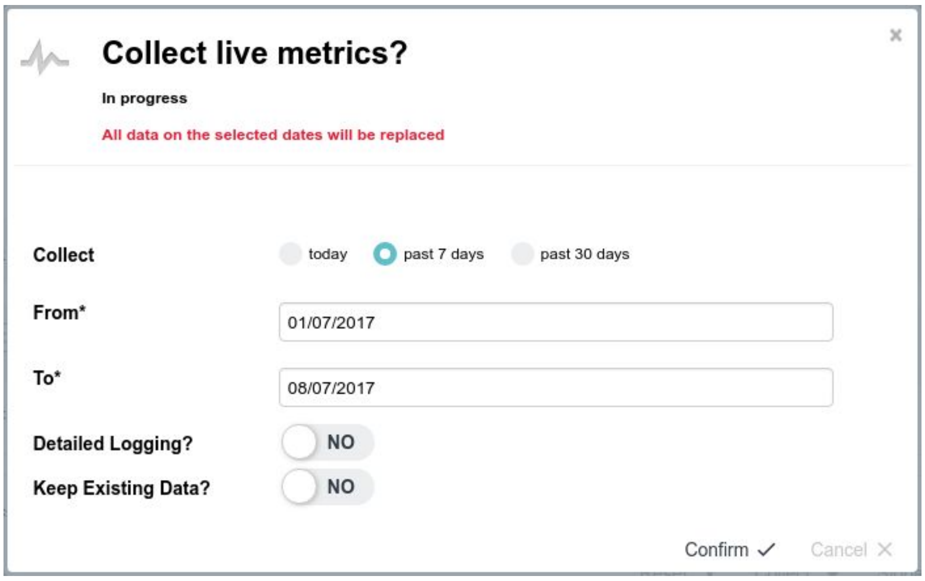
```
{base_jql} AND status WAS 'Resolved' ON '2015-11-14'
```

Unresolved defects JQL:

```
{base_jql} AND status WAS NOT 'Resolved' ON '2015-11-14'
```

Functions

As well as providing custom variables for working with complex JQL ServiceClarity provides built in date and time functions to support historic data collection through dynamically generated JQL.



Collect live metrics?

In progress

All data on the selected dates will be replaced

Collect	today [✓ past 7 days] past 30 days
From*	01/07/2017
To*	08/07/2017
Detailed Logging?	NO
Keep Existing Data?	NO

There is a complete reference of the available built in functions at the end of this document but the most relevant for our example is the **`DATE`** function. This refers to the date of the data collection, a date that by default means "today" but could for historical data refer to any date in the past, for which there is JIRA data. In the above example showing the Collect live metrics dialogue ServiceClarity will collect JIRA data for all dates between 1st July 2017 and the 8th July 2017 and for each of these dates the **`DATE`** function will rewrite the JQL query to use the correct date.

We can rewrite the JQL for our work in progress example using the built in **`DATE`** function like this:

```
{base_jql} AND status WAS 'In Progress' ON '{DATE,yyyy-MM-dd}'
```

When a ServiceClarity data collection is run using the above JQL the **`DATE`** function will calculate the correct date and substitute the formatted value into the JQL.

Other useful date related functions provided by ServiceClarity include the following, each of which can be formatted if required using optional formatting rules:

- `{END_OF_DAY}`
- `{END_OF_LAST_MONTH}`
- `{END_OF_LAST_WEEK}`
- `{END_OF_LAST_YEAR}`
- `{END_OF_MONTH}`
- `{END_OF_NEXT_MONTH}`
- `{END_OF_NEXT_WEEK}`
- `{END_OF_NEXT_YEAR}`
- `{END_OF_WEEK}`
- `{END_OF_YEAR}`
- `{LAST_MONTH}`
- `{LAST_WEEK}`
- `{LAST_YEAR}`
- `{NEXT_MONTH}`
- `{NEXT_WEEK}`
- `{NEXT_YEAR}`
- `{DATE}`
- `{START_OF_DAY}`
- `{START_OF_LAST_MONTH}`
- `{START_OF_LAST_WEEK}`
- `{START_OF_LAST_YEAR}`
- `{START_OF_MONTH}`
- `{START_OF_NEXT_MONTH}`
- `{START_OF_NEXT_WEEK}`
- `{START_OF_NEXT_YEAR}`
- `{START_OF_WEEK}`
- `{START_OF_YEAR}`

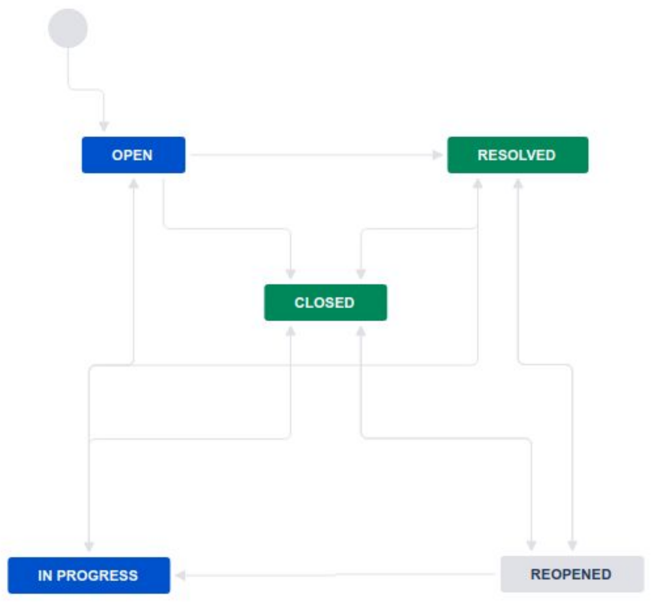
All of the above date keywords can be extended with offsets - days, weeks months or years - to provide even greater control. For example,

- `{DATE(-1d)}` - the day before the date of collection
- `{START_OF_MONTH(-1d)}` - the end of the month before the date of collection

- `{START_OF_MONTH(-6M)}` - 6 months before the date of collection start at the 1st
- `{START_OF_WEEK(-4w)}` - the Monday 4 weeks before the date of collection
- `{END_OF_WEEK(-1w)}` - the Sunday before the date of collection
- `{END_OF_MONTH(-1M)}` - the end of the month before the date of collection
- `{END_OF_MONTH(-3M)}` - the end of the month 3 months before the date of collection
- `{START_OF_YEAR(-1y)}` - the 1st January he year of the collection

JIRA Workflow

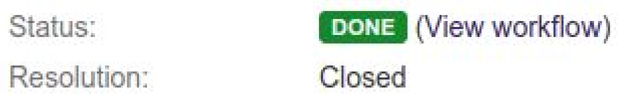
Although JIRA and JIRA ServiceDesk are used in a wide range of organisations to solve a seemingly endless variety of operational problems at its core JIRA is a work tracking tool built around the concept of tracking work items (tasks, problems, customer issues, bugs, etc) through a set of defined stages, or steps. The stages that items in JIRA move through are collectively known as JIRA Workflow and by default JIRA and JIRA cloud accounts come preconfigured with a simple JIRA Workflow, sometimes referred to as “JIRA classic”.



JIRA workflows can be highly customised and predefined templates are available for download as JIRA add-ons. To read more on JIRA workflows we recommend the Atlassian support document:

- Working with JIRA workflows https://confluence.atlassian.com/adminjiracloud/working-with-workflows-776636540.html

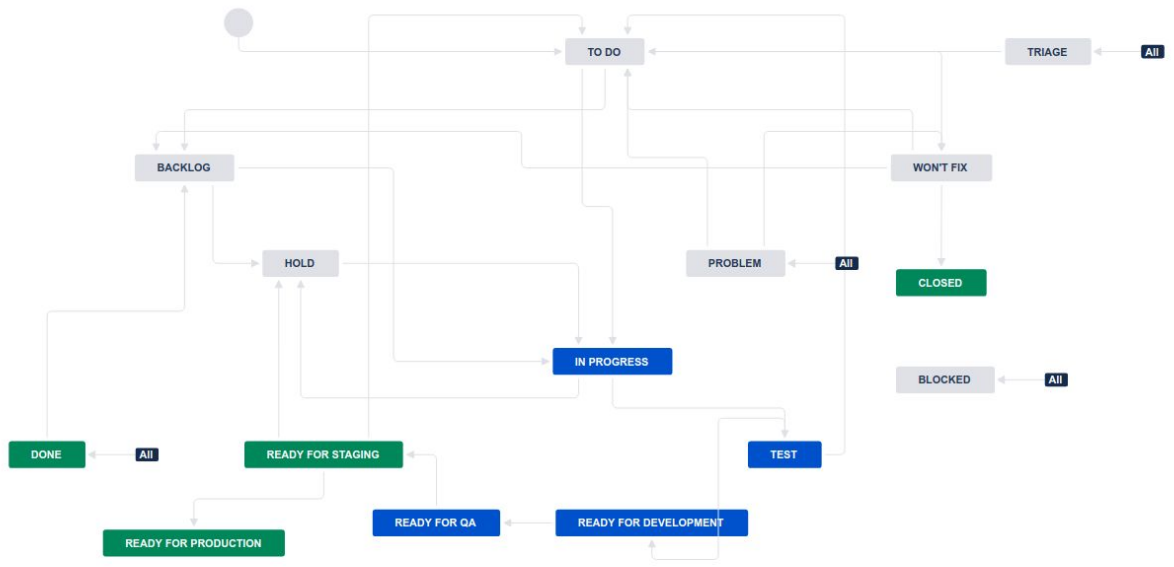
To find out what workflow your JIRA project is configured for find a JIRA issue for the project and click on the “View workflow” link on the issue status field. Note that different JIRA issue types may have different workflows.



Status: DONE (View workflow)
Resolution: Closed

Understanding your JIRA workflow is essential for getting the best data out of your JIRA system. For example, in our previous example we collected a ***work in progress*** KPI by searching for JIRA issues that have the status “In Progress”. This status is a default one, provided by the default workflow, however, many organisations operate much more sophisticated JIRA workflows with potentially multiple “In Progress” states. For example the following workflow defines potentially six states that could be considered as “In Progress”:

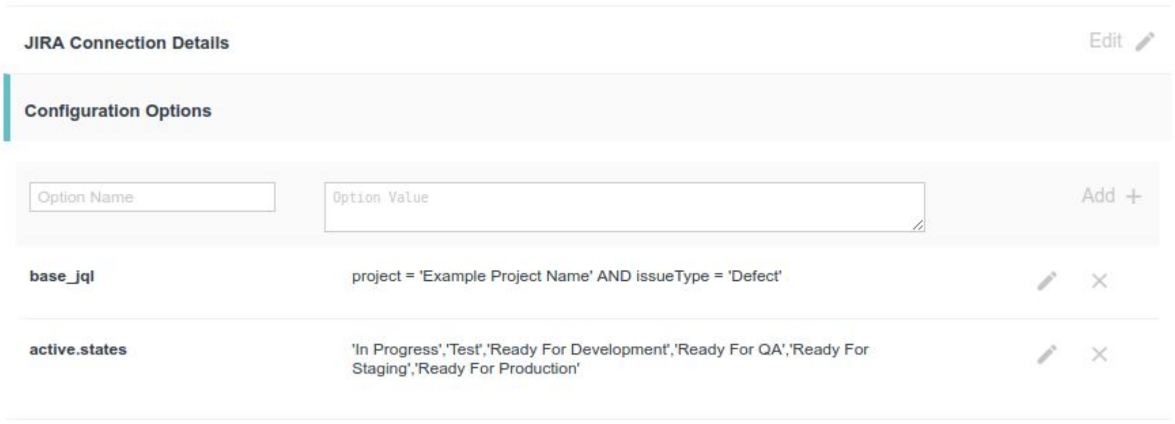
1. In Progress
2. Test
3. Ready For Development
4. Ready For QA
5. Ready For Staging
6. Ready for Production



Based on the above workflow we might rewrite our example work in progress JQL as:

```
{base_jql} AND status WAS IN ('In Progress','Test','Ready For Development','Ready For
QA','Ready For Staging','Ready For Production') ON '{DATE,yyyy-MM-dd}'
```

In the previous section we looked at simplifying complex JQL through the use of ServiceClarity variables. The ability to define a custom variable for “In Progress” states not only simplify the JQL it also enables reuse of the definition of what is in progress across multiple KPIs.


JIRA Connection Details Edit

Configuration Options

Option Name | Option Value | Add +

base_jql	project = 'Example Project Name' AND issueType = 'Defect'
active.states	'In Progress','Test','Ready For Development','Ready For QA','Ready For Staging','Ready For Production'

Making use of our new active.states variable we can rewrite our final version of the JQL for a ***work in progress*** KPI as:

```
{base_jql} AND status WAS IN ({active.states}) ON '{DATE,yyyy-MM-dd}'
```

Checking what data you have

Beyond the essential JIRA KPIs, ***work in progress***, ***work outstanding*** (backlog), etc, what KPIs can be derived from your JIRA data depends on how your JIRA is configured and how you make use of JIRA workflows and fields. For example, do you size your work with estimates? Do you have a “reopened” state in your workflow? Are JIRA issues moved promptly between states in your workflow?

However, there are some basic checks that can be performed to ensure that you get the best out of the data that you have.

Completed work

Work that has been completed, regardless of which custom workflow is used, should move to its final state with an assigned “resolution”. When an issue in JIRA is assigned a resolution certain internal JIRA processes are triggered. For tracking JIRA KPIs in ServiceClarity the most important of these triggered processes is the automatic setting of a resolution date. Although not essential certain KPIs are best expressed in terms of this resolution date.

It is possible to check if your JIRA issues are being completed with a resolution by running the following JQL in the JIRA Issue Search:

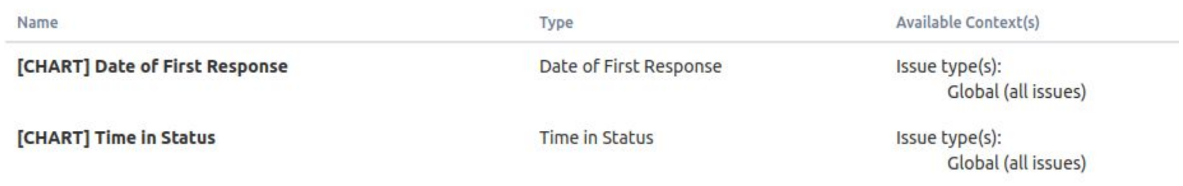
```
resolutiondate is EMPTY AND statusCategory = Done
```

The above query should not return any results. If results are returned then check that the “resolution” field is not hidden, which is a common cause of missing resolution dates.

Detailed timings

ServiceClarity is capable of tracking many different time related JIRA KPIs, including but not limited to: the total time that work takes, the average time that it is waiting, the average time to respond, etc. Most of these time based KPIs rely on detailed internal JIRA records that are not normally available in the JIRA application itself. These detailed timings are collected by one of the default Atlassian add-ons: JIRA Charting plugin. This add-ons is available and enabled by default on all JIRA Cloud accounts. On standalone JIRA Server installations it is available but can sometimes be disabled.

If you are interested in tracking ***cycle-times***, ***mean time to respond***, ***mean time to resolve***, average ***lead times***, etc then it is worth checking that your JIRA system has the JIRA Charting plugin enabled. If you have administrative privileges the easiest way to confirm that you can get timing data from your JIRA is from within the JIRA application. Access the JIRA Administration menu and navigate to the “Issues” settings and then from there go the the “Custom fields” settings. If the JIRA Charting plugin is enabled there will be two custom fields:



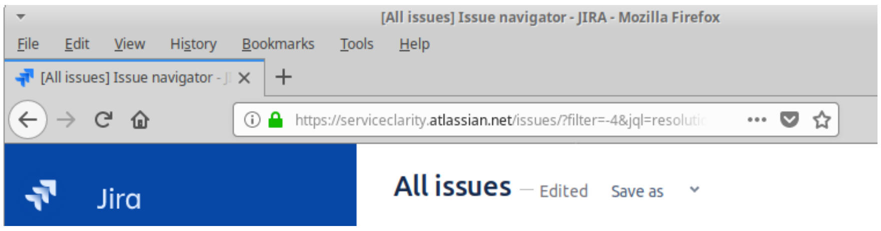
Name	Type	Available Context(s)
[CHART] Date of First Response	Date of First Response	Issue type(s): Global (all issues)
[CHART] Time in Status	Time in Status	Issue type(s): Global (all issues)

How to connect ServiceClarity to JIRA

To collect data from your JIRA system ServiceClarity needs to be configured with connection details. You will need the following JIRA details:

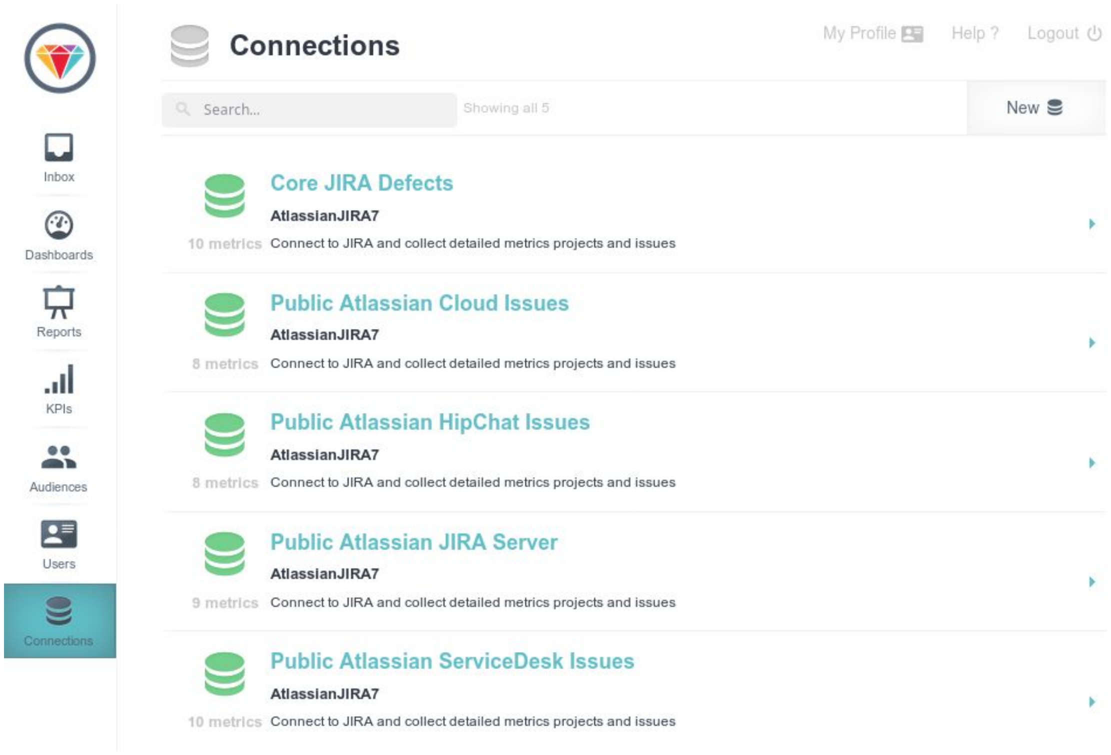
1. JIRA username
2. JIRA password
3. JIRA URL - *e.g. https://example.atlassian.net*

Although ServiceClarity can connect with your JIRA system using any user credentials you provide *we strongly recommend that you create a specific user account for ServiceClarity*. This enables you to control the JIRA projects and data that can be accessed by ServiceClarity from within JIRA - for example a JIRA user for ServiceClarity does not need write access nor does it need access to projects that will not be tracked from ServiceClarity.


[All issues] Issue navigator - JIRA - Mozilla Firefox
File
Edit
View
History
Bookmarks
Tools
Help
[All issues] Issue navigator - J
https://serviceclarity.atlassian.net/issues/?filter=-4&jql=resolutio
Jira
All issues — Edited
Save as

The JIRA URL that you will need is the *public* address of your JIRA system. To determine this login into JIRA and copy the the start of the address up to and including the .net or .com, etc. In the above example this would be: https://serviceclarity.atlassian.net

From the ServiceClarity application you can create as many connections to your JIRA system as you need. As you build your library of JIRA KPIs you may find that grouping similar sets of JIRA data into separate JIRA connections is a useful way to manage data for specific projects or types of issues, etc. In the screenshot below you can see multiple JIRA connections within ServiceClarity that separate out the data for different projects:



Core JIRA Defects
AtlassianJIRA7
10 metrics Connect to JIRA and collect detailed metrics projects and issues

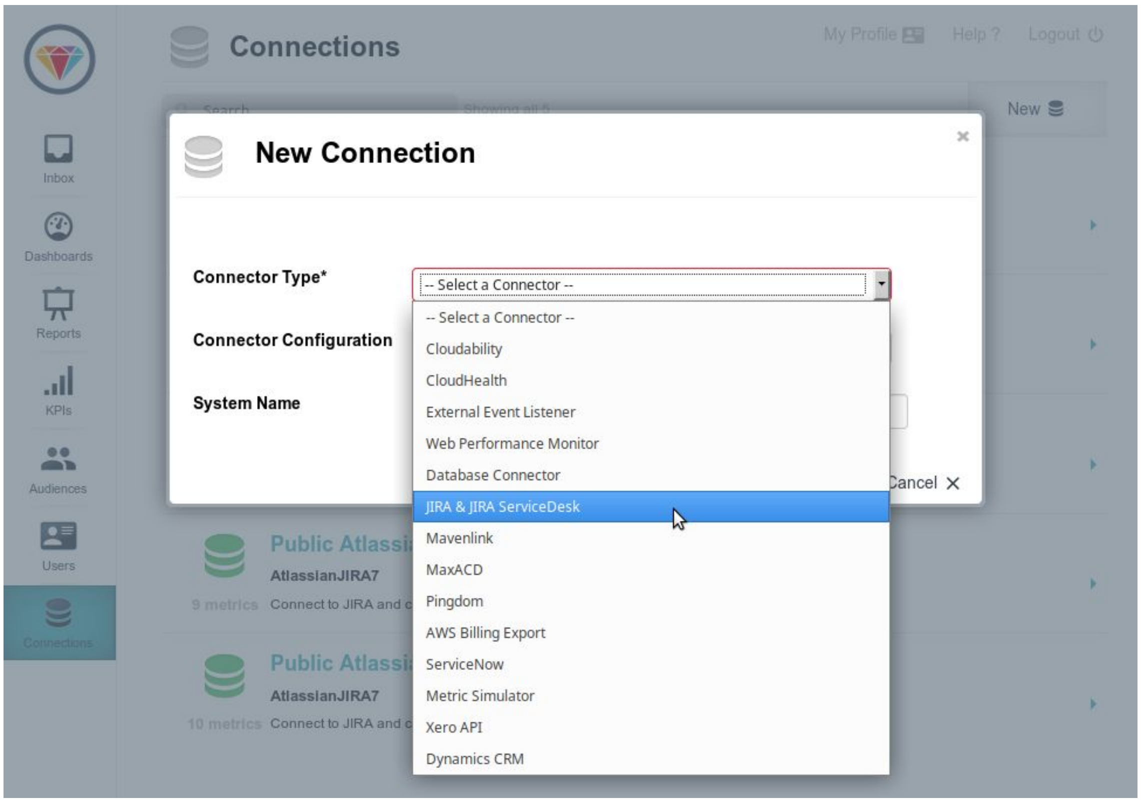
Public Atlassian Cloud Issues
AtlassianJIRA7
8 metrics Connect to JIRA and collect detailed metrics projects and issues

Public Atlassian HipChat Issues
AtlassianJIRA7
8 metrics Connect to JIRA and collect detailed metrics projects and issues

Public Atlassian JIRA Server
AtlassianJIRA7
9 metrics Connect to JIRA and collect detailed metrics projects and issues

Public Atlassian ServiceDesk Issues
AtlassianJIRA7
10 metrics Connect to JIRA and collect detailed metrics projects and issues

To create a new connection to JIRA in SerivceClarity navigate to the connections page and select the “New” button at the top right hand side of your screen. The new connections dialogue that pops-up offers a number of different connection types including one for JIRA & JIRA ServiceDesk.



New Connection

Connector Type* -- Select a Connector --

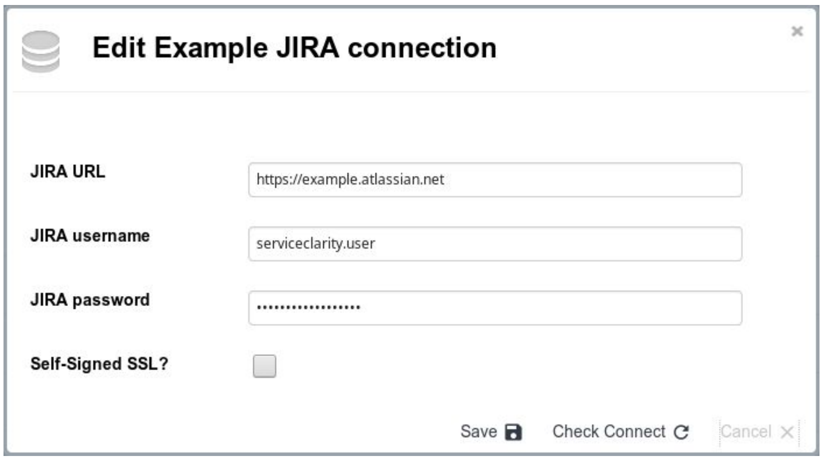
Connector Configuration

System Name

Cancel

Once you have selected JIRA & JIRA ServiceDesk from the available list you may also choose to base this new connection on an existing template, which will include a number of JIRA data collectors templates and associated KPIs. For example, if you have already created a JIRA connection you can at this point choose to copy it as a template for your new connection.

Your newly created connection will need to be configured with your JIRA details - click on the “Edit” link on the JIRA Connection Details section of the connection page.



Edit Example JIRA connection

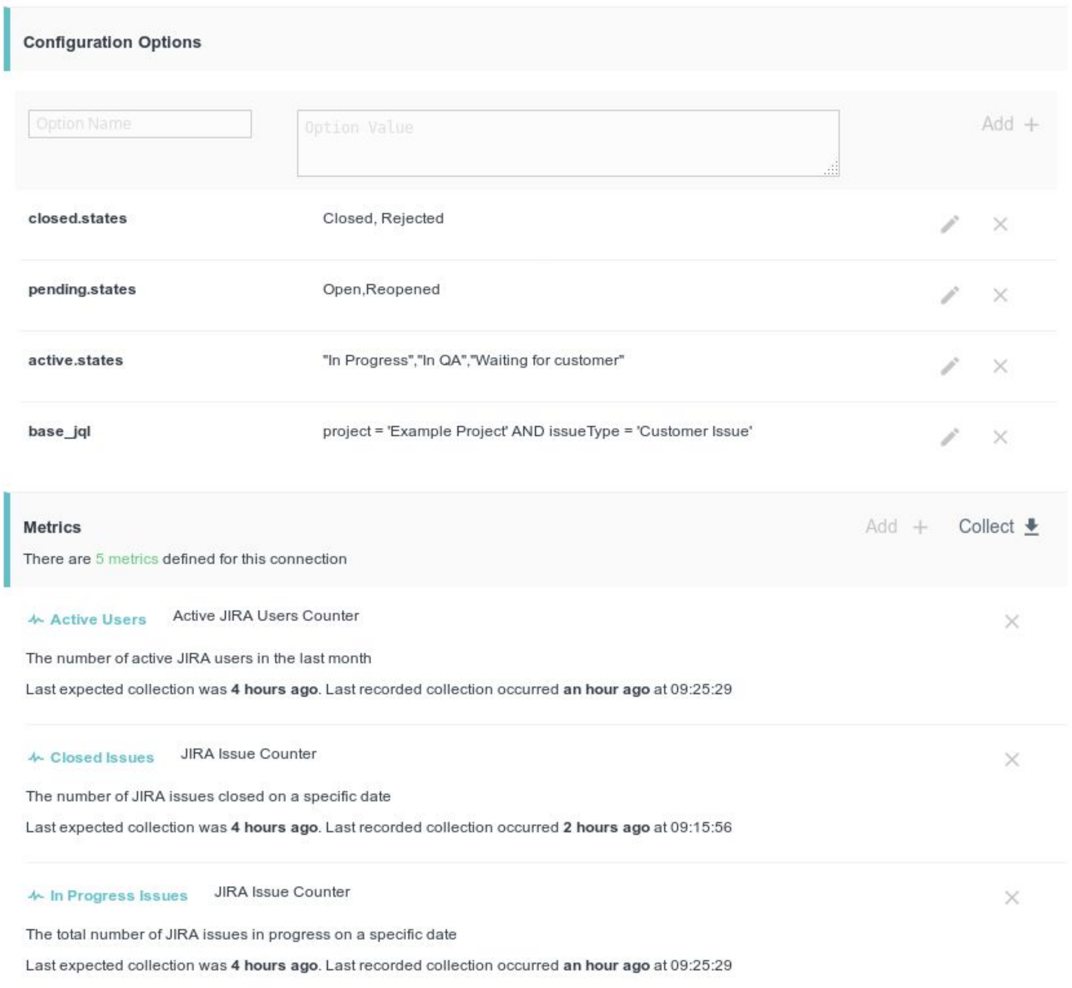
JIRA URL: https://example.atlassian.net

JIRA username: serviceclarity.user

JIRA password: ....

Self-Signed SSL? ☐

If you have opted to create a new JIRA connection from an existing template your new connection will have a set of pre-provisioned configuration options and metrics.



Configuration Options

Option Name	Option Value
closed.states	Closed, Rejected
pending.states	Open,Reopened
active.states	"In Progress","In QA","Waiting for customer"
base_jql	project = 'Example Project' AND issueType = 'Customer Issue'

Metrics

There are 5 metrics defined for this connection

Active Users — Active JIRA Users Counter

The number of active JIRA users in the last month

Last expected collection was **4 hours ago**. Last recorded collection occurred **an hour ago** at 09:25:29

Closed Issues — JIRA Issue Counter

The number of JIRA issues closed on a specific date

Last expected collection was **4 hours ago**. Last recorded collection occurred **2 hours ago** at 09:15:56

In Progress Issues — JIRA Issue Counter

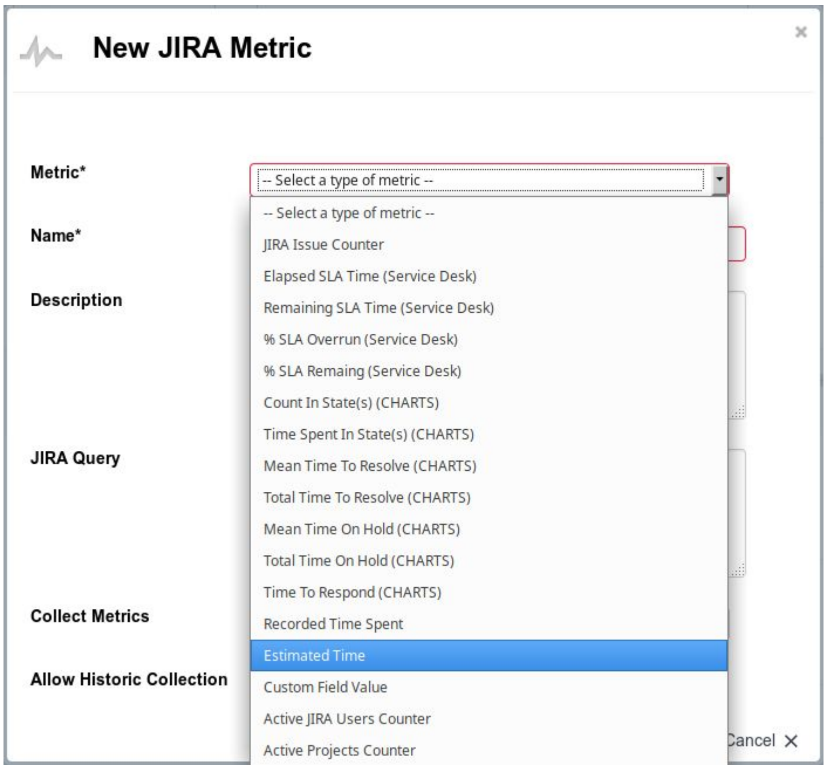
The total number of JIRA issues in progress on a specific date

Last expected collection was **4 hours ago**. Last recorded collection occurred **an hour ago** at 09:25:29

The example JIRA connection configuration shown here includes 3 JIRA data collectors for ***active users***, ***closed issues*** and ***in progress issues***. All of these have been inherited from an existing connection. There are also 4 configured ServiceClarity variables - described in a previous section on ServiceClarity JQL variables. At this point it is worth reviewing these variables as they may need updated to reflect the purpose of this JIRA connection, perhaps a change of project or issue type in the ***base_jql*** variable or perhaps this connection does not include “Waiting for customer” as one of the ***active.states***. It is also worth reviewing the inherited JIRA data collectors (metrics) to make sure that they do not need refined.

ServiceClarity JIRA data collectors

There are a range of specialised ServiceClarity data collectors available of JIRA and JIRA ServiceDesk - data collectors are also known as “metrics” because they are low level metrics extracted from JIRA. When adding a new data collector to a JIRA connection you can select a standard “Issue Counter” or a JIRA ServiceDesk specific SLA collector or one of our cycle time collectors, to name a few. Each of these different collector types will make use of JIRAs JQL to narrow the focus of the collection, as described in previous sections. Some of these collectors require additional configuration; the Custom Field collector for example must be configured with the name of the custom field to collect as well as an aggregation function to apply - `SUM`, `AVG`, `MAX`, etc.


New JIRA Metric
Metric*
Name*
Description
JIRA Query
Collect Metrics
Allow Historic Collection
-- Select a type of metric --
-- Select a type of metric --
JIRA Issue Counter
Elapsed SLA Time (Service Desk)
Remaining SLA Time (Service Desk)
% SLA Overrun (Service Desk)
% SLA Remaing (Service Desk)
Count In State(s) (CHARTS)
Time Spent In State(s) (CHARTS)
Mean Time To Resolve (CHARTS)
Total Time To Resolve (CHARTS)
Mean Time On Hold (CHARTS)
Total Time On Hold (CHARTS)
Time To Respond (CHARTS)
Recorded Time Spent
Estimated Time
Custom Field Value
Active JIRA Users Counter
Active Projects Counter
Cancel

JIRA Issue Counter

Options:

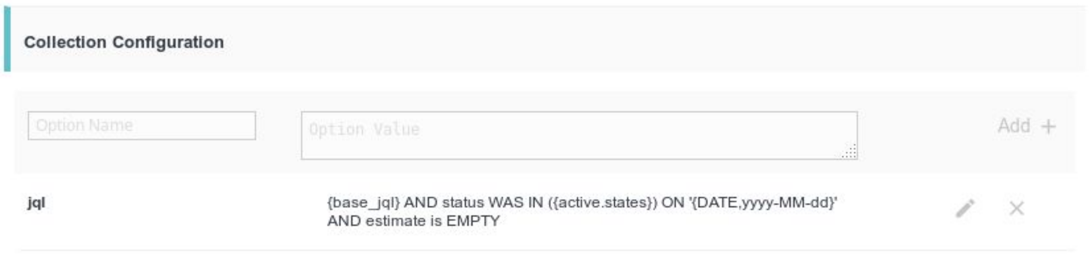
- **`jql`** - a ServiceClarity JQL filter to find the JIRA issues of interest

Returns:

- The number of JIRA issues returned by the JQL query

The JIRA Issue Counter is the default ServiceClarity data collector for JIRA. It is the cornerstone of many essential JIRA KPIs, ***work in progress***, ***backlog***, ***burndown***, ***% issues rejected***, etc.

To configure a JIRA Issue Counter you need to specify the JQL that finds the JIRA issues (or bugs, tasks, etc) that are to be counted. No additional configuration is required, although it is recommend that you consider defining a list of JIRA issue fields a ***breakdown*** for the returned data. For example, a ***backlog*** collection of outstanding work might be broken down by issue type or issue priority or by and custom field configured for your JIRA system. The JIRA Issue Counter can also be used to track quality KPIs such as the number of ***active issues without estimates***, which depending on your process could represent ***unplanned work***.


Collection Configuration

Option Name	Option Value
jql	{base_jql} AND status WAS IN ({active.states}) ON '{DATE,yyyy-MM-dd}' AND estimate is EMPTY

Elapsed SLA Time (Service Desk)

Options:

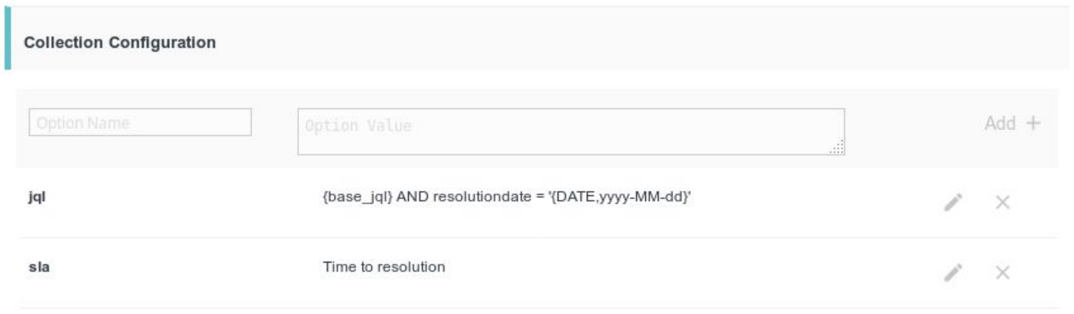
- **`jql`** - a ServiceClarity JQL filter to find the JIRA issues of interest
- **`sla`** - the name of the JIRA ServiceDesk SLA

Returns:

- The average elapsed SLA time in seconds of JIRA ServiceDesk issues returned by the JQL query

The Elapsed SLA Time collector is specific to JIRA ServiceDesk and requires that your JIRA Cloud account or JIRA Server instance is configured with at least one SLA. This collector will calculate the average elapsed SLA time of all JIRA issues returned from the JQL query. This could be used to calculate the average time of issues that have been resolved, to give an ***average response time*** KPI or an ***average resolution time*** KPI.

To configure an Elapsed SLA Time collector you need to specify the JQL and the name of the SLA, for example an ***average resolution time*** collector could be configured as:


Collection Configuration

Option Name	Option Value
jql	{base_jql} AND resolutiondate = '{DATE,yyyy-MM-dd}'
sla	Time to resolution

Remaining SLA Time (Service Desk)

Options:
- `jql` - a ServiceClarity JQL filter to find the JIRA issues of interest
- `sla` - the name of the JIRA ServiceDesk SLA

Returns:
- The average remaining SLA time in seconds of JIRA ServiceDesk issues returned by the JQL query

The Remaining SLA Time collector is specific to JIRA ServiceDesk and requires that your JIRA Cloud account or JIRA Server instance is configured with at least one SLA. This collector will calculate the average SLA time remaining of all JIRA issues returned from the JQL query. With the right JQL this could be used to track an ***SLA risk factor*** or ***SLA margin*** KPI.

To configure a Remaining SLA Time collector you need to specify the JQL and the name of the SLA.

% SLA Overrun (Service Desk)

Options:
- `jql` - a ServiceClarity JQL filter to find the JIRA issues of interest
- `sla` - the name of the JIRA ServiceDesk SLA

Returns:
- The % SLA overrun of JIRA ServiceDesk issues returned by the JQL query

The % SLA Overrun collector is specific to JIRA ServiceDesk and requires that your JIRA Cloud account or JIRA Server instance is configured with at least one SLA. This collector will calculate the % of overrun on SLAs that have been breached, depending on the JIRA issues returned from the JQL query. With the right JQL this could be used to monitor a quality/performance KPI.

To configure a % SLA Overrun collector you need to specify the JQL and the name of the SLA.

% SLA Remaining (Service Desk)

Options:

- `jql` - a ServiceClarity JQL filter to find the JIRA issues of interest
- `sla` - the name of the JIRA ServiceDesk SLA

Returns:

- The % SLA remaining of JIRA ServiceDesk issues returned by the JQL query

The % SLA Remaining collector is specific to JIRA ServiceDesk and requires that your JIRA Cloud account or JIRA Server instance is configured with at least one SLA. This collector will calculate the % of remaining of all JIRA issues returned from the JQL query. With the right JQL this could be used to monitor a quality/performance KPI.

To configure a % SLA Remaining collector you need to specify the JQL and the name of the SLA.

Count In State(s) (CHARTS)

Options:

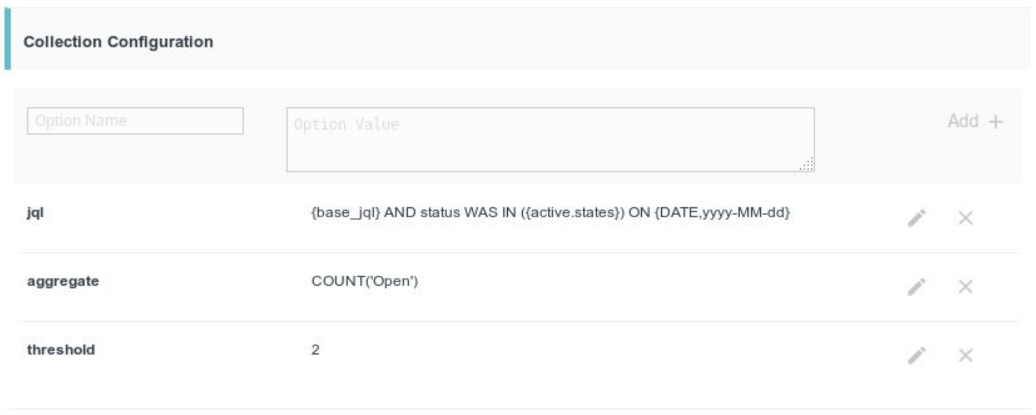
- `jql` - a ServiceClarity JQL filter to find the JIRA issues of interest
- `aggregate` - the aggregate function to apply, e.g. `COUNT('Open')`
- `threshold` - the number of times in a state before an issue is counted

Returns:

- The count JIRA issues returned by the JQL query that have been in the specified aggregate state for the number of times specified as a threshold

The Count In State collector requires that your JIRA cloud account or JIRA Service installation has the JIRA Charts plugin enabled (see the section on Checking what data you have). This collector can count the number of JIRA issues that have been in a stage of the JIRA workflow more than once. Depending on your workflow, and with the right JQL, it can be used to track workflow KPIs such as: ***blocked tasks*** or ***reopened issues***.

To configure a Count In State collector you need to specify the JQL, the name of the workflow status and a threshold value for inclusion in the count. For example, you could track work that has been in the "Open" state more than once to create a ***churn*** KPI.



Collection Configuration

jql	{base_jql} AND status WAS IN ({active.states}) ON {DATE,yyyy-MM-dd}
aggregate	COUNT('Open')
threshold	2

Time Spent In State(s) (CHARTS)

Options:

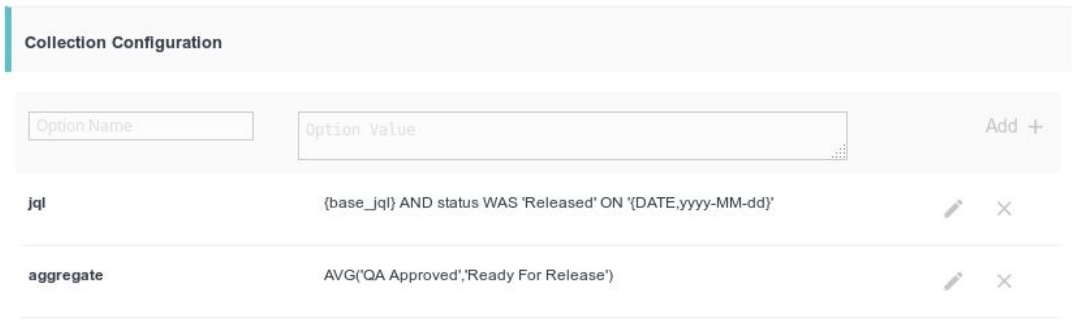
- **`jql`** - a ServiceClarity JQL filter to find the JIRA issues of interest
- **`aggregate`** - the aggregate function to apply, e.g. `AVG('In Progress')` or `SUM('In Progress')` or in multiple states `AVG('Waiting on third party', 'Waiting on customer')`

Returns:

- The aggregate `SUM` or `AVG` of the time spent, in minutes, in one or more workflow states of JIRA issues returned by the JQL query

The Time Spent In State collector requires that your JIRA Cloud account or JIRA Service installation has the JIRA Charts plugin enabled (see the section on Checking what data you have). This collector can count the average or total time that JIRA issues spend in one or more stages of the JIRA workflow. For example, with this collector it is possible to track KPIs such as ***total time in progress***, ***mean time in QA*** and ***total time waiting for release***.

To configure a Time Spent In State collector you need to specify the JQL, the name of the workflow status and an aggregation function to apply. For example, you could track the average time spent waiting to create a ***waiting for release*** KPI.



Collection Configuration

jql	{base_jql} AND status WAS 'Released' ON '{DATE,yyyy-MM-dd}'
aggregate	AVG('QA Approved','Ready For Release')

Mean Time To Resolve (CHARTS)

Options:

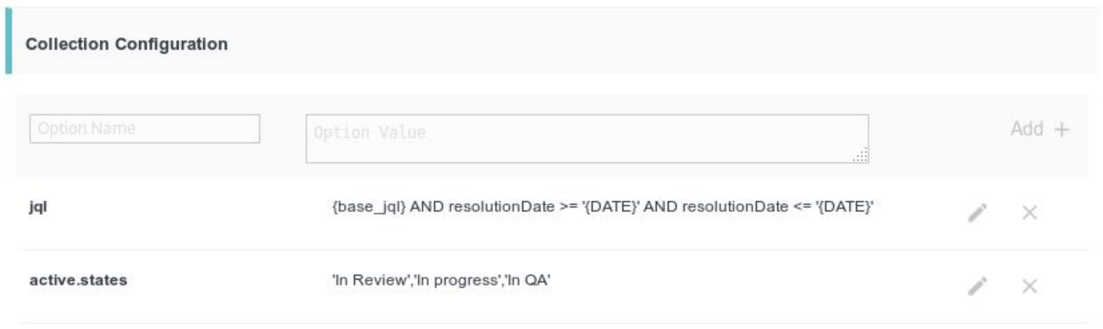
- **`jql`** - a ServiceClarity JQL filter to find the JIRA issues of interest
- **`active.states`** - a list of JIRA workflow states that are considered to be active, e.g. `‘In Progress,‘In QA’`

Returns:

- The average of the time spent, in minutes, in one or more workflow states - defined by the **`active.states`** option - of JIRA issues returned by the JQL query

Mean Time To Resolve collector requires that your JIRA Cloud account or JIRA Service installation has the JIRA Charts plugin enabled (see the section on Checking what data you have). This collector will count the average time that JIRA issues spend in one or more designated “active” stages of the JIRA workflow. Simply define the list of workflow states that indicate work is in progress and a JQL query that finds recently completed work to track a ***mean time to resolve*** KPI.

To configure a Mean Time To Resolve collector you need to specify the JQL and the list of workflow statuses that are considered to be “active”.


Collection Configuration

Option Name	Option Value
jql	{base_jql} AND resolutionDate >= '{DATE}' AND resolutionDate <= '{DATE}'
active.states	'In Review','In progress','In QA'

Total Time To Resolve (CHARTS)

Options:

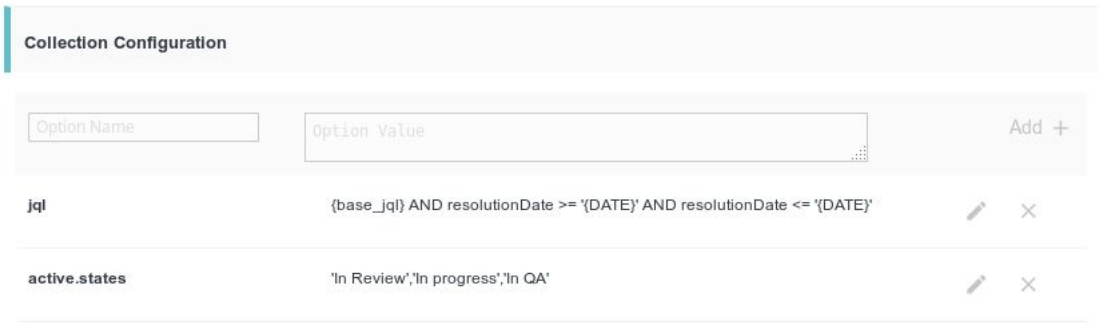
- **`jql`** - a ServiceClarity JQL filter to find the JIRA issues of interest
- **`active.states`** - a list of JIRA workflow states that are considered to be active, e.g. `‘In Progress,‘In QA’`

Returns:

- The aggregate `SUM` of all time spent, in minutes, in one or more workflow states - defined by the **`active.states`** option - of JIRA issues returned by the JQL query

Total Time To Resolve collector requires that your JIRA Cloud account or JIRA Service installation has the JIRA Charts plugin enabled (see the section on [Checking what data you have](#)). This collector will count the total time that all JIRA issues returned by the JQL query spend in one or more designated “active” stages of the JIRA workflow. Simply define the list of workflow states that indicate work is in progress and a JQL query that finds recently completed work to track a ***time spent in progress*** KPI.

To configure a Total Time To Resolve collector you need to specify the JQL and the list of workflow statuses that are considered to be “active”.


Collection Configuration

jql	{base_jql} AND resolutionDate >= '{DATE}' AND resolutionDate <= '{DATE}'	
active.states	'In Review','In progress','In QA'	

Mean Time On Hold (CHARTS)

Options:

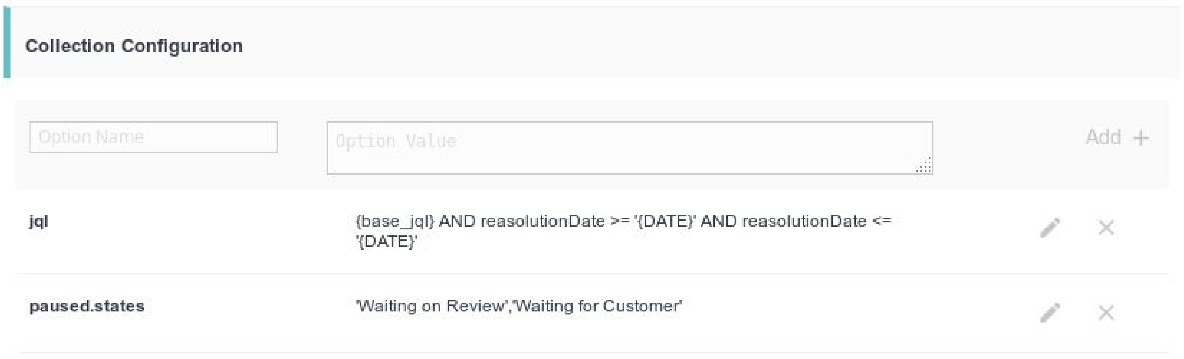
- **`jql`** - a ServiceClarity JQL filter to find the JIRA issues of interest
- **`paused.states`** - a list of JIRA workflow states that are considered to be “on hold”, e.g. `‘Waiting for customer’,‘In Review’`

Returns:

- The average time spent, in minutes, in one or more workflow states - defined by the **`paused.states`** option - of JIRA issues returned by the JQL query

Mean Time On Hold collector requires that your JIRA Cloud account or JIRA Service installation has the JIRA Charts plugin enabled (see the section on [Checking what data you have](#)). This collector will count the average time that all JIRA issues returned by the JQL query spend in one or more designated “paused” stages of the JIRA workflow. Simply define the list of workflow states that indicate work has paused and a JQL query that finds recently completed work to track a ***mean time spent blocked*** KPI.

To configure a Mean Time On Hold collector you need to specify the JQL and the list of workflow statuses that are considered to be “paused”.


Collection Configuration

Option Name	Option Value
jql	{base_jql} AND reasolutionDate >= '{DATE}' AND reasolutionDate <= '{DATE}'
paused.states	'Waiting on Review','Waiting for Customer'

Total Time On Hold (CHARTS)

Options:

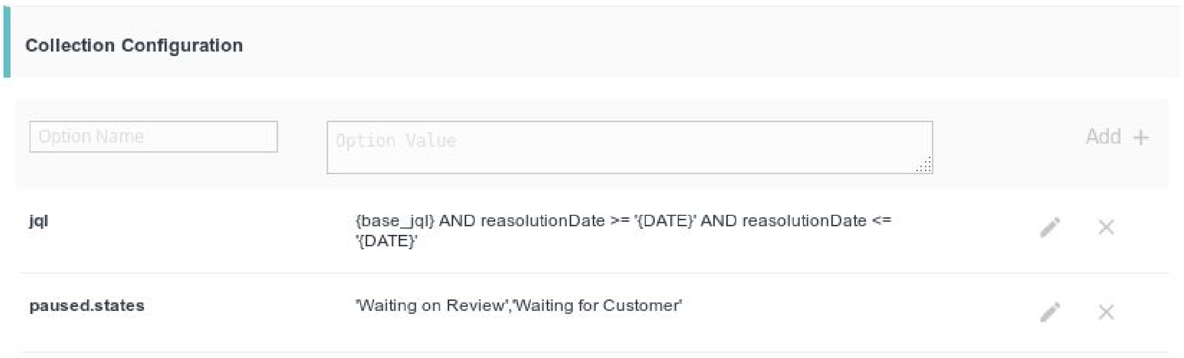
- **`jql`** - a ServiceClarity JQL filter to find the JIRA issues of interest
- **`paused.states`** - a list of JIRA workflow states that are considered to be "on hold", e.g. `'Waiting for customer','In Review'`

Returns:

- The aggregate `SUM` of all time spent, in minutes, in one or more workflow states - defined by the **`paused.states`** option - of JIRA issues returned by the JQL query

Total Time On Hold collector requires that your JIRA Cloud account or JIRA Service installation has the JIRA Charts plugin enabled (see the section on Checking what data you have). This collector will count the total time that all JIRA issues returned by the JQL query spend in one or more designated "paused" stages of the JIRA workflow. Simply define the list of workflow states that indicate work has paused and a JQL query that finds recently completed work to track a ***time spent blocked*** KPI.

To configure a Total Time On Hold collector you need to specify the JQL and the list of workflow statuses that are considered to be "paused".


Collection Configuration

Option Name	Option Value
jql	{base_jql} AND reasolutionDate >= '{DATE}' AND reasolutionDate <= '{DATE}'
paused.states	'Waiting on Review','Waiting for Customer'

Time To Respond (CHARTS)

Options:

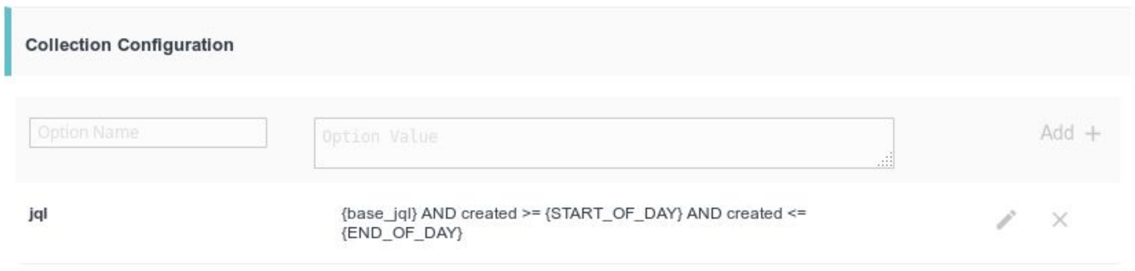
- `jql` - a ServiceClarity JQL filter to find the JIRA issues of interest

Returns:

- The average time spent, in minutes, to respond to newly created issues calculated from the JIRA issues returned by the JQL query

Time To Respond collector requires that your JIRA Cloud account or JIRA Service installation has the JIRA Charts plugin enabled (see the section on Checking what data you have). This collector will calculate the average time that it took for an assignee to respond to issues returned by the JQL query creating a ***mean time to response*** KPI.

To configure a Time To Respond collector you need to specify the JQL that finds the target JIRA issues, for example tracking the response time of issues newly created on any given date would enable you to aggregate daily values to weekly and monthly trends in response time.



jql {base_jql} AND created >= {START_OF_DAY} AND created <= {END_OF_DAY}

Recorded Time Spent

Options:

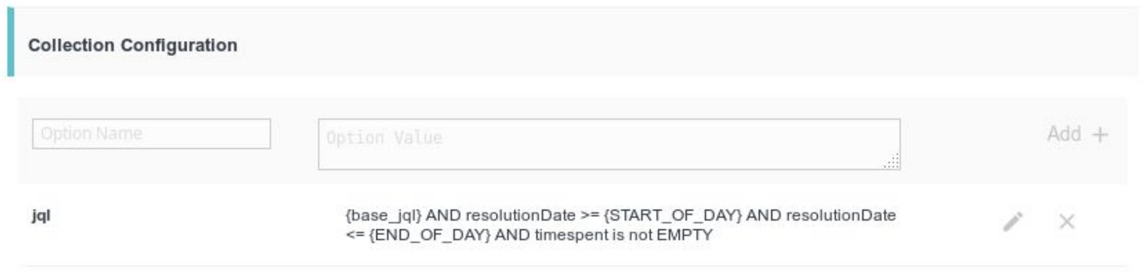
- `jql` - a ServiceClarity JQL filter to find the JIRA issues of interest

Returns:

- The aggregate `SUM` of all time spent, in seconds, recorded by assignees against the JIRA issues returned by the JQL query

The Recorded Time Spent collector requires that the assignee of a JIRA issue records their time using the standard JIRA field: Time spent. This collector will calculate the total time (in seconds) recorded by the assignee against all the JIRA issues returned by the JQL query. The recorded time spent can be used directly as an ***effort*** or ***billable hours*** KPI or combined with other data to calculate an ***actual vs estimates*** KPI.

To configure a Recorded Time Spent collector you need to specify the JQL that finds the target JIRA issues, in general the total time spent is best calculated after JIRA issues have been resolved - when all work is completed. Tracking the time spent of resolved issues on any given date would enable you to aggregate daily values to weekly and monthly trends.



jql	{base_jql} AND resolutionDate >= {START_OF_DAY} AND resolutionDate <= {END_OF_DAY} AND timespent is not EMPTY

Estimated Time

Options:

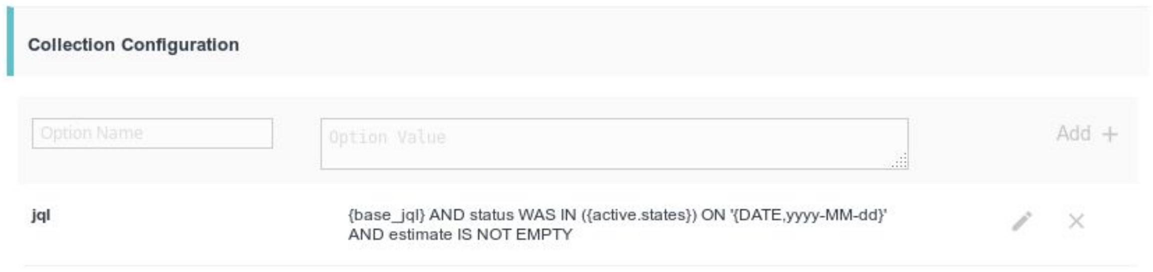
- **`jql`** - a ServiceClarity JQL filter to find the JIRA issues of interest

Returns:

- The aggregate `SUM` of all estimated time, in seconds, recorded by assignees against the JIRA issues returned by the JQL query

The Estimated Time collector requires that the standard JIRA field: Estimate is completed on all issues of interest. This collector will calculate the total time (in seconds) recorded as an estimate against all the JIRA issues returned by the JQL query. The estimated time can be used directly as a measure of ***work in progress*** KPI or combined with other data to calculate an ***actual vs estimates*** KPI.

To configure a Estimated Time collector you need to specify the JQL that finds the target JIRA issues, in general the estimated is best calculated after JIRA issues have been resolved - when all work is completed. Tracking the time spent of resolved issues on any given date would enable you to aggregate daily values to weekly and monthly trends.



jql	{base_jql} AND status WAS IN ({active.states}) ON '{DATE,yyyy-MM-dd}' AND estimate IS NOT EMPTY

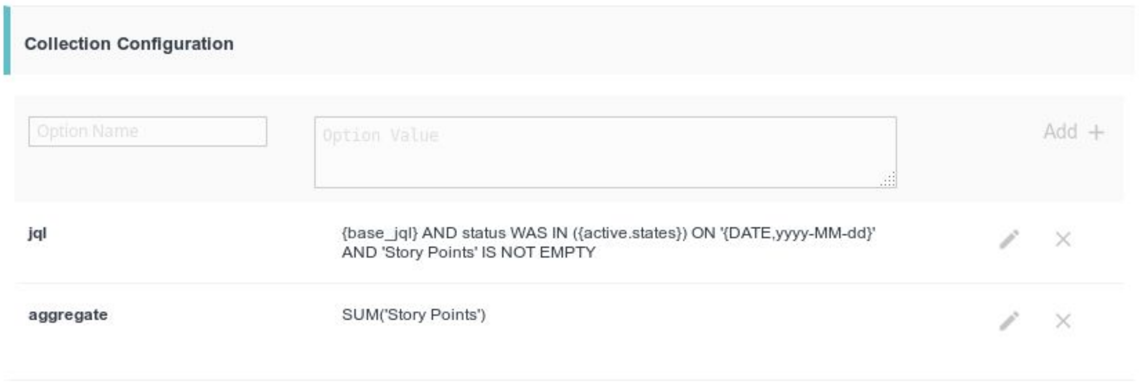
Custom Field Value

Options:
- **`jql`** - a ServiceClarity JQL filter to find the JIRA issues of interest
- **`aggregate`** - the aggregate function to apply, e.g. `AVG('T-Shirt Size')` or `SUM('Story Points')`

Returns:
- The aggregate `SUM` or `AVG` of any numeric JIRA field calculated across all JIRA issues returned by the JQL query

The Custom Field collector will calculate the aggregate value of any numeric field including fields added by JIRA plugins such as JIRA Agile and custom fields unique to your JIRA instance or JIRA Cloud account. This collector can `count`, `sum` and `average` the value of custom fields for all JIRA issues returned by the JQL query. It can also find the `max` and `min` values. For example with this collector it is possible to track a ***work in progress*** KPI from JIRA Agile Story Points.

To configure a Custom Field collector you need to specify a suitable JQL query to find JIRA issues of interest and an aggregate function to apply to your custom field.


Collection Configuration

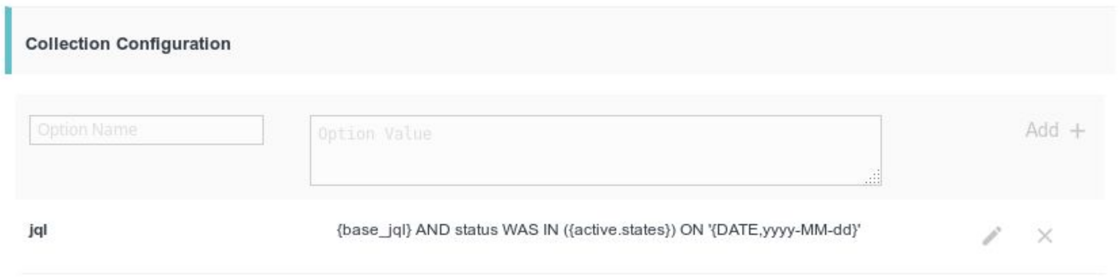
jql	{base_jql} AND status WAS IN ({active.states}) ON '{DATE,yyyy-MM-dd}' AND 'Story Points' IS NOT EMPTY
aggregate	SUM('Story Points')

Active JIRA Users Counter

Options:
- **`jql`** - a ServiceClarity JQL filter to find the JIRA issues of interest

Returns:
- The total number of unique assignees calculated across all JIRA issues returned by the JQL query

The Active JIRA Users collector will calculate the total number of unique JIRA assignees of all issues returned by the JQL query. With an appropriate JQL query it can be used to track an ***active users*** KPI day by day, aggregating activity over weeks and months are required. It can also be used in combination with other KPIs such as ***work in progress*** and estimates to project delivery dates and plan release dates.


Collection Configuration

jql	{base_jql} AND status WAS IN ({active.states}) ON '{DATE,yyyy-MM-dd}'

Active Projects Counter

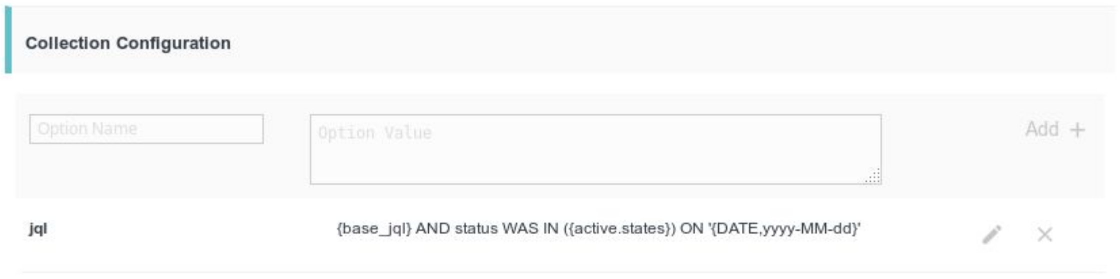
Options:

- `jql` - a ServiceClarity JQL filter to find the JIRA issues of interest

Returns:

- The total number of unique projects calculated across all JIRA issues returned by the JQL query

The Active Projects Counter will calculate the total number of unique JIRA projects across the issues returned by the JQL query. With an appropriate JQL query it can be used to track an ***active projects*** KPI day by day, aggregating activity over weeks and months are required.


Collection Configuration

jql	{base_jql} AND status WAS IN ({active.states}) ON '{DATE,yyyy-MM-dd}'