# Expanse Jira Cloud Integration v2.0.0

Documentation and User Guide

---

Expanse's Jira Cloud Integration allows you to automatically create Jira issues for Issue updates in Expanse Expander. This documentation includes information about how to install this integration, configuration details, and context for building business workflows using Expanse issue details. For additional information about Expanse you can visit our Knowledge Base and if you have any questions about Jira Cloud or other Atlassian Cloud products, you can find their documentation here.

### Goals and Outcomes

Examples of goals and outcomes for customers using the Expanse Integration include:

- Integration of Expanse Issues data into existing ticket management processes
- Reducing mean time to remediation (MTTR) for your Expanse Issues

## Common Use Cases

*Creating tickets for new Issues*

Create Jira issues for new Expanse Issues in Expander, based on the type of issue, priority, and which business unit it belongs to.

*New Issue awareness on network perimeter (On-Prem and Cloud)*

Ingest and take action on newly appeared or reappeared Expanse Issues for a given time period, such as active Remote Desktop Protocol (RDP) servers or Telnet services, with relevant context like IP, port, protocol, banner response, and tags.

# Overview of Functionality

---

## Functionality

1. *Creating new Tasks for new and untracked Expanse Issues*

   This integration will create a new Task type issue in a specified Jira Project for all new Expanse Issues that match the filters provided. It will also create a new issue for updates on existing Expanse Issues if no matching issue in Jira is found.

2. *Auto-Closing related Tasks based on resolved Issues*

   The integration will automatically close an open issue for an Expanse Issue if it detects that the Issue has been closed via a new update on the Issue's activity status.

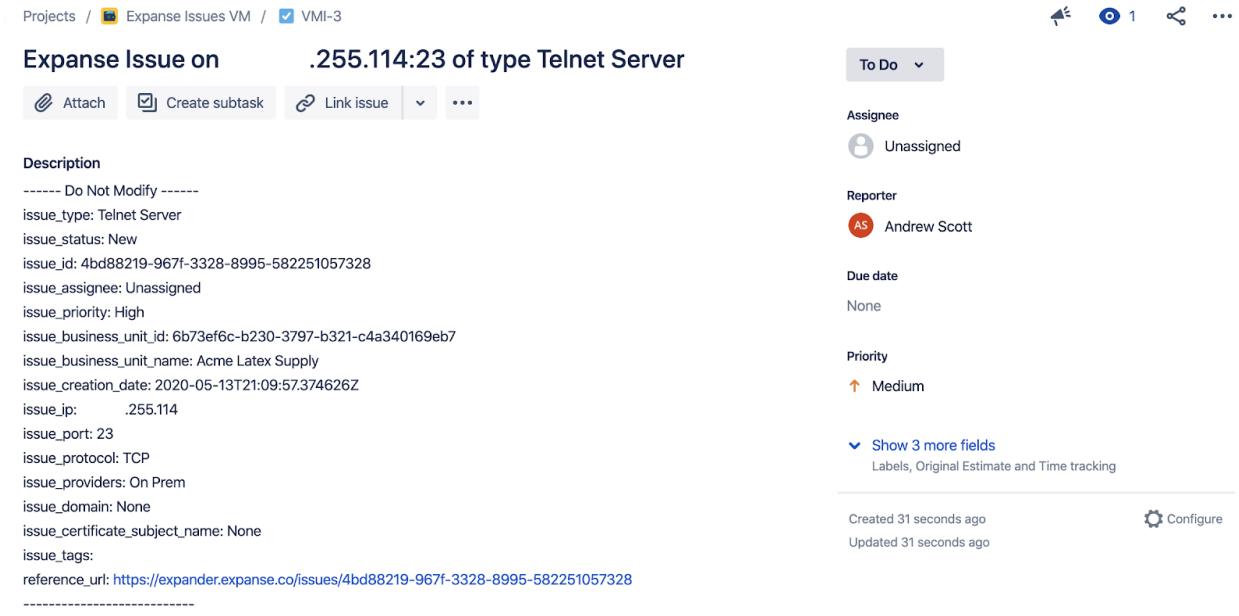
3. *Creating Custom Fields for Expanse Issues [OPTIONAL]*

   If configured to do so, this integration will create a new "Business Project" project in Jira to create all Expanse related issues in. It can also be configured to create new custom fields in Jira which are useful

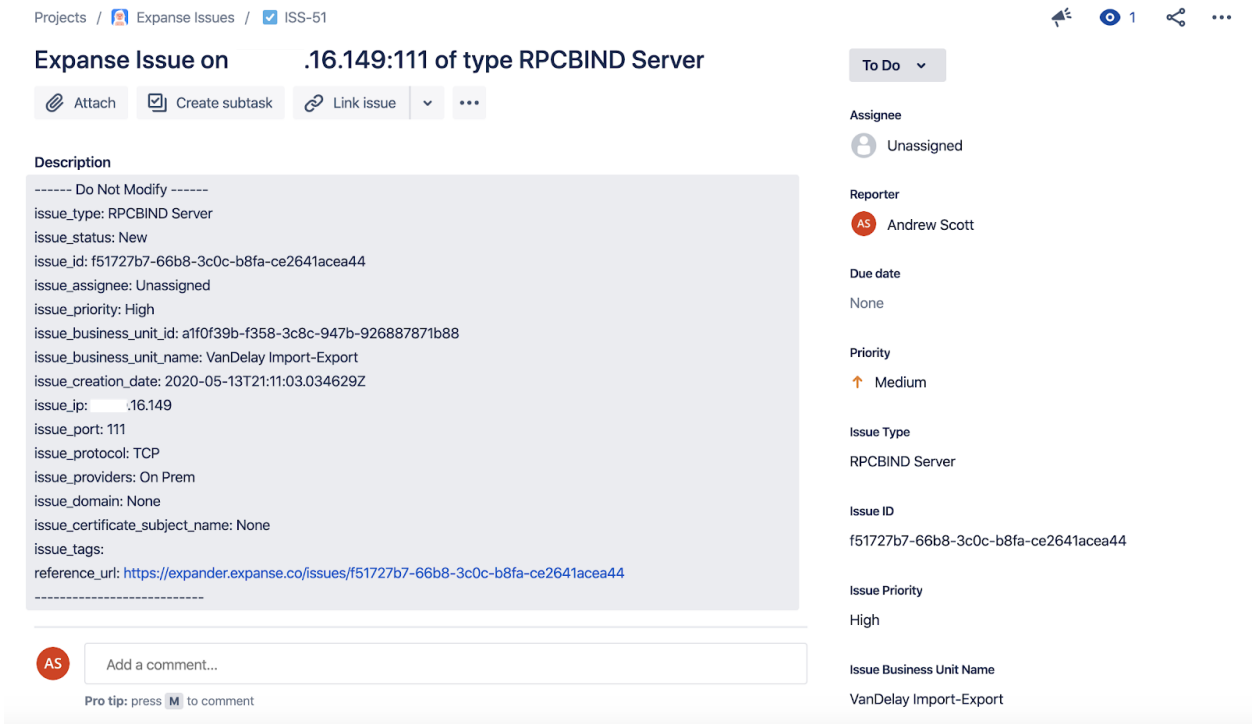
for advanced filtering, dashboards, and reports. Custom fields include **issue_type**, **issue_id**, **issue_priority**, and **issue_business_unit_name**.

# Results

## Example of an issue created in an existing project with no custom fields




## Example of an issue created using custom fields



**NOTE: This integration only supports Jira Cloud. If you're using Jira Server and would like to integrate with Expanse please contact your Sales Representative or Engagement Manager.**

# Installing the Integration

The integration is delivered as a Python package in compressed tar.gz format which can either be run natively on the host using python 3.7+ or run within a Docker container.

## Requirements

- Linux/Unix Host with a minimum of 2 GB RAM
- Python 3.7+ **OR** a modern version of Docker
- An Internet connection for outgoing requests

## Installing Python Dependencies

If you intend to run the integration directly using python you can use pip to install python dependencies. Within the base directory of the integration there is a file named requirements.txt which can be used with pip to install the necessary python modules. It is also recommended that you use a virtual environment to ensure you don't encounter any global dependency clashes.

To install run:

```
pip install -r requirements.txt
```

## Installing Docker Dependencies

If you don't already have Docker installed, you can follow the Docker Installation Guide. Once you have Docker installed on your host, you should not need to do any additional setup - Docker will install all necessary dependencies when it builds a new image of the integration.

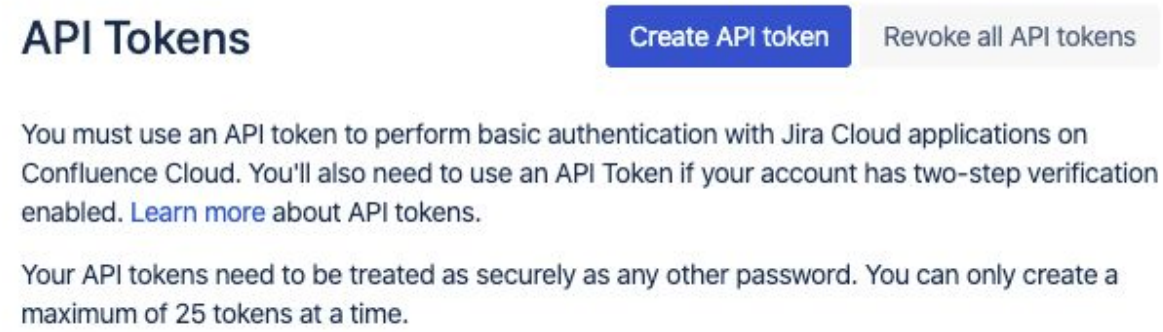
You should wait until after you've gone through the configuration steps below to actually build the image, otherwise your image may not be configured as intended.

# Configuring the Integration

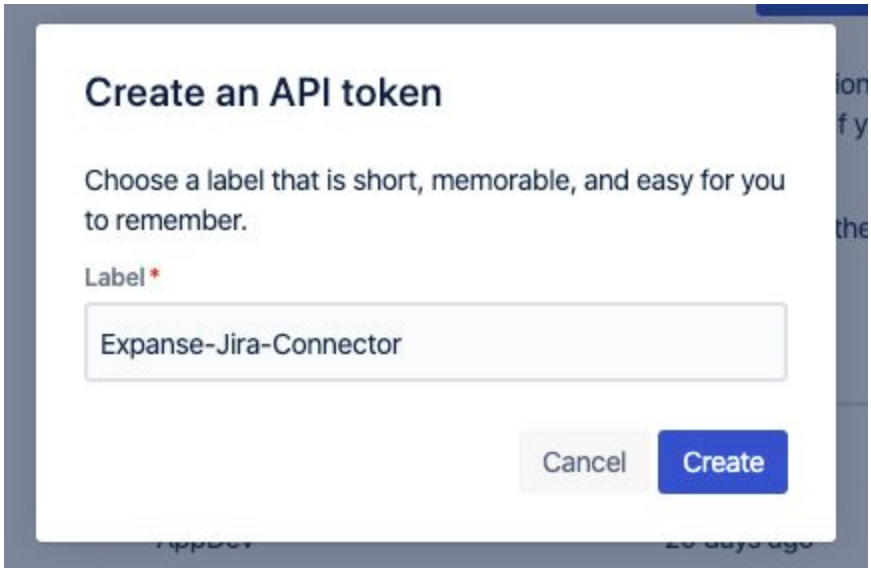
The Expanse Jira Cloud configuration is quite configurable in order to accommodate the different workflows and Jira restrictions of our users. Configuration of the integration is primarily done using a YAML file named **expanse.yml**.

## Generating a new Jira Cloud API Token

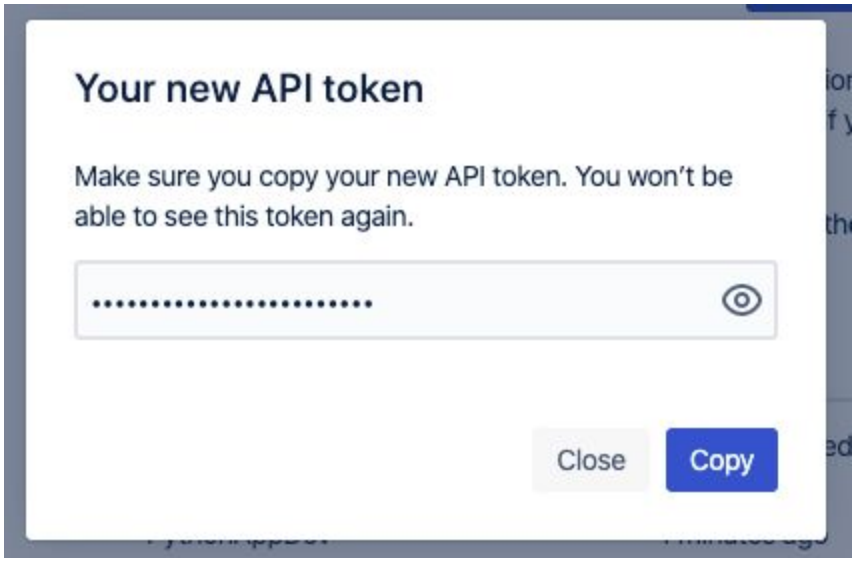
1. Go to https://id.atlassian.com/manage/api-tokens
2. Click the "Create API Token" button.




3. Give your new token a name.




4. Copy the generated API token.




## YAML Configuration

| Field Name | Description | Required | Default Value |
|---|---|---|---|
| jira.api_token | API token associated with a user in Jira Cloud used to authenticate requests. | Yes | N/A |
| jira.api_username | Jira Cloud username used to authenticate requests. | Yes | N/A |
| jira.address | Jira Cloud domain address. | Yes | N/A |
| jira.project | The Jira Cloud project key that should be used for the integration. If an existing project key is used new Expanse Issues will be added to that project. If a project key is used that does not exist a new project will be created for that key. | Yes | EXP |
| jira.project_lead | Account ID for project owner in Jira Cloud. If omitted the Account ID for jira.api_username will be used. | No | N/A |

| | | | |
|---|---|---|---|
| jira.map_to_description | You can set map_to_description to *true* if you don't want any custom fields to be created and would like all summary data to be placed in the task description. If set to *false* custom fields will be created for the following: issue_type, issue_id, issue_priority, and issue_business_unit_name. Custom fields will be added to all screens related to the jira.project value provided above. | Yes | false |
| expanse.api_token | API token for Expanse. This is used to authenticate requests to the Expanse API. | Yes | N/A |
| expanse.filter.priority | This filter can be used to specify which Expanse Issue updates generate new issues in Jira. Valid options are Critical, High, Medium, and Low. Multiple values can be used. | No | [Critical, High] |
| expanse.filter.issue_types | This filter can be used to specify which Expanse Issue updates generate new issues in Jira. Check the `Example_expanse.yml` for a full list of options.<br><br>If omitted any Issue types that match the priority specified above will be included. Priority is checked before Issue Type, so a lower priority Issue Type will not work with a higher priority general setting. | No | N/A |
| expanse.filter.business_unit_names | This filter can be used to specify which Expanse Issue updates generate new issues in Jira. This should be a single Business Unit name or a list of names. If omitted Issues from all business units will be created in Jira. This filter takes precedence above Priority and Issue Type. | No | N/A |
| general.past_days | The number of days to look back when starting. | No | 1 |
| general.check_interval | How often to check for new Issue updates (in hours). | No | 2 |
| general.logging_level | The logging level for the integration. Valid options are DEBUG, INFO, WARN, and ERROR | No | INFO |

| | | | |
|---|---|---|---|
| general.logging_location | By default the integration will log to stdout as well as to file. This sets the logging location. | Yes | N/A |

## Examples

This example would be for a customer who wanted to filter for RDP, TELNET, and SNMP Issues and wants to leverage an existing Jira project named RISK without creating any custom fields.

```
jira:
  api_token: MxyOaYYdQasApoCf6OUpFD75
  api_username: admin@exampleorg.com
  address: exampleorg.atlassian.net
  project: RISK
  map_to_description: true

expanse:
  api_token: lX1ppps4d1ZYRjW321Y2v1EotvvDkiOQ-CjOrZ0vKKWi2
  filter:
    issue_types:
      - TelnetServer
      - SnmpServer
      - RdpServer

general:
  past_days: 2
  check_interval: 2
  logging_level: INFO
  logging_location: expanse_jira.log
```

This second example would be for a customer who wanted to include all Critical and High issues belonging to a single Business Unit and add these to a new Jira Project named EXP that will include custom fields.

```
jira:
  api_token: MxyOaYYdQasApoCf6OUpFD75
  api_username: admin@exampleorg.com
  address: exampleorg.atlassian.net
  project: EXP
  map_to_description: false

expanse:
  api_token: lX1ppps4d1ZYRjW321Y2v1EotvvDkiOQ-CjOrZ0vKKWi2
  filter:
    priority:
      - Critical
      - High
    business_unit_names:
      - Acme Financial

general:
```

```
past_days: 1
check_interval: 2
logging_level: INFO
logging_location: expanse_jira.log
```

## Environment Variables

For some of the more sensitive values, environment variables can also be used in addition to the YAML config file. If environment variables are provided they will override any values provided in the YAML config file.

| Name | Corresponding YAML Field | Description |
|---|---|---|
| JIRA_CLOUD_API_TOKEN | jira.api_token | API token associated with a user in Jira Cloud used to authenticate requests. |
| JIRA_CLOUD_API_USERNAME | jira.api_username | Jira Cloud username used to authenticate requests. |
| JIRA_CLOUD_API_URL | jira.address | Jira Cloud domain address. |
| EXPANSE_API_TOKEN | expanse.api_token | API token for Expanse. This is used to authenticate requests to the Expanse API. |

## Command Line Arguments

In addition to the YAML config file and environment variable, there are also some runtime configurations that can be made via command line arguments.

| Arg Name | Description | Required | Default |
|---|---|---|---|
| --conf | The location of the YAML config file. | No | ./expanse.yml |
| --single-run | Run the issue ingest a single time using the **general.past_days** value from the YAML config file. | No | false |

# Running the Integration

## Running with Python

1. [Optional] Create a new virtual environment
2. Install python dependencies using pip.
3. Update the PYTHONPATH environment variable to include the path of the integration.

```
export PYTHONPATH=$PYTHONPATH:$(pwd)
```

4. Create a new file in the base of the directory of the integration named **expanse.yml**. Alternatively you can actually place this file in any directory you want with any name you want if you intend to use --conf command line argument. Configure this file based on the options and examples above.
5. [OPTIONAL] configure any environment variable based on the options above.
6. Start the integration by running

```
python expanse_jira
```

7. [OPTIONAL] You can also ensure that your python process doesn't get killed by using a tool like tmux or nohup to run the python command. Please consult third party tool documentation for running with these tools.

## Running with Docker

1. Create a new file in the base of the directory of the integration named **expanse.yml**.
2. Run the following Docker command to build and run a new container

```
docker build -t expanse_jira . && docker run -it expanse_jira
```

# Troubleshooting the Integration

If you encounter issues with the integration you are encouraged to enable debug logging to facilitate faster diagnosis of issues. You can do this by setting **general.logging_level** to **DEBUG** in the YAML config.