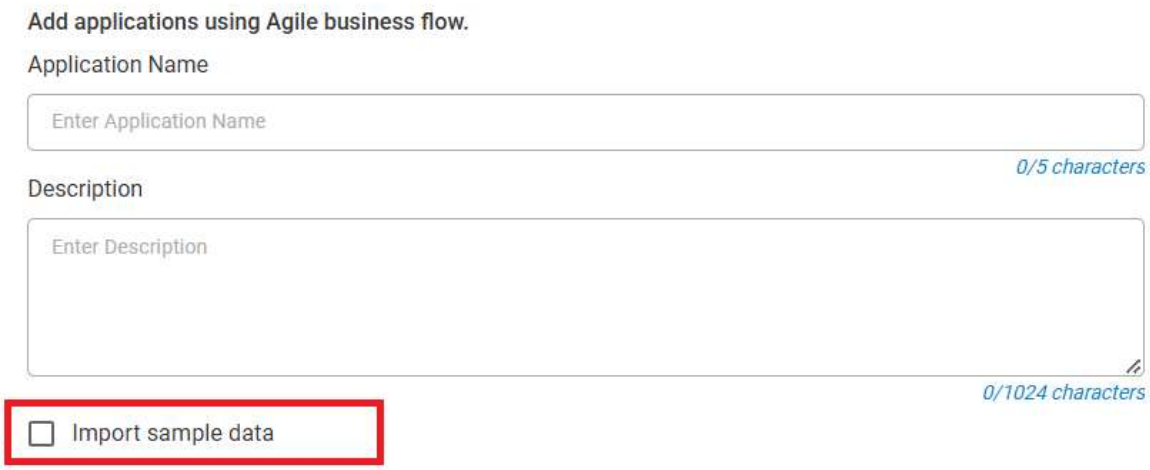
Importing Data to Plan

Introduction

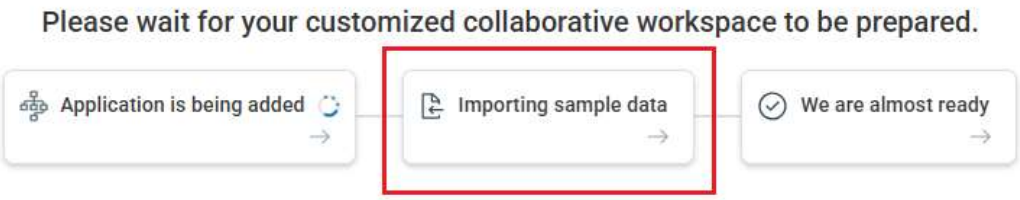
This document describes how to import data into Plan using the new Import feature. There are two ways to add bulk data to your application. Some schemas have out of the box sample data you can use as a starting point or for demonstration purposes. You can choose to import this data in either the Agile, FullSAFe or DefectTracking schemas. You can do this both while you are creating the application and after you have created it. The other way to import data is to upload a zip containing a bill of materials (bom.txt) file, one or more CSV files containing the record data, and a substitution.json file for performing any transformations on the data as you import. This document will describe how to do the imports and also describe the expected file formats for the bom.txt, substitution.json and csv files.

Importing Sample Data During Creation

As mentioned in the previous section, Agile, FullSAFe and DefectTracking schemas have sample data you can use for demonstration and exploration purposes. When you create the application, you can specify to include the sample data by checking the “Import sample data” box.

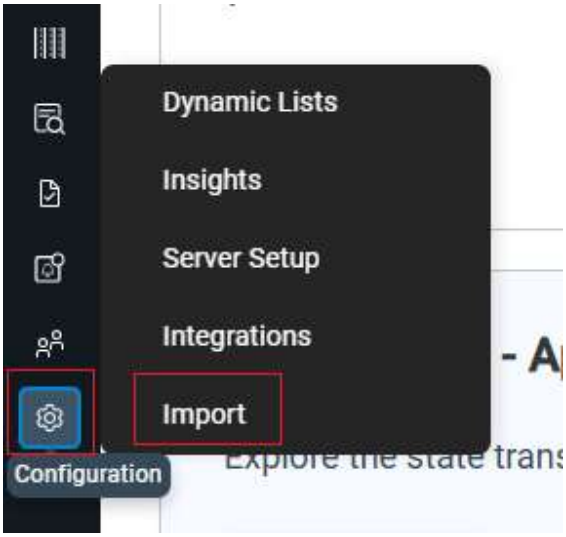


During application creation, you will see an extra box specifying when the sample data is being imported:

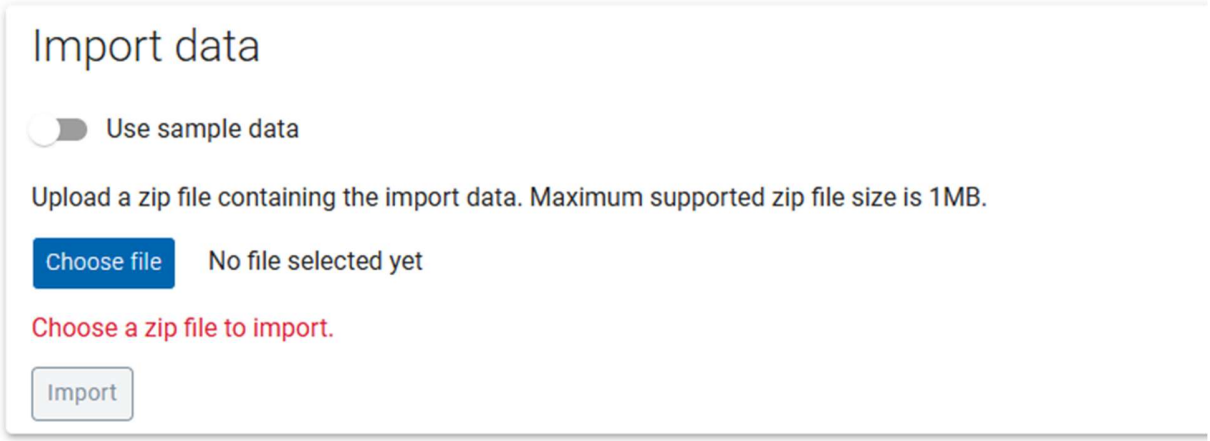


Importing Sample Data After Creation

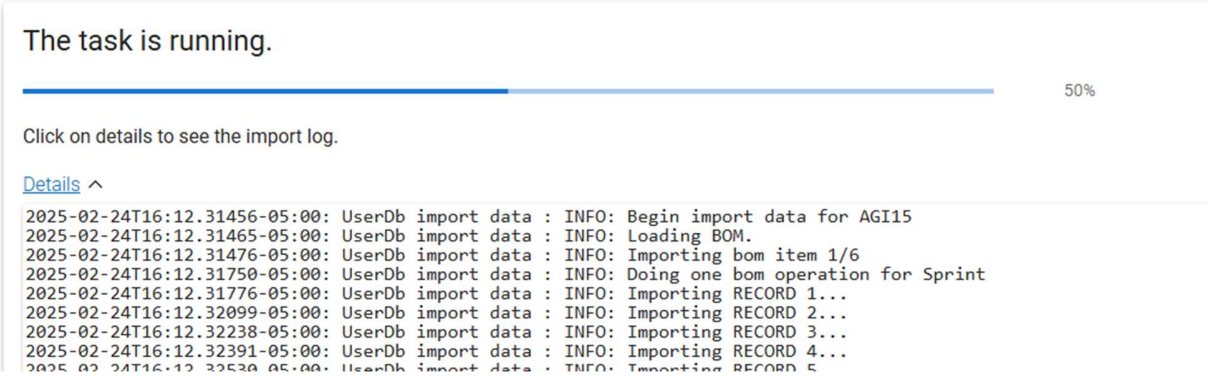
You can also choose to import the sample data after you have created the application, if you didn't already import it during creation. Choose the Import option in the configure/gear menu. Only teamspace admins/userdb super-users, will be able to import data this way.



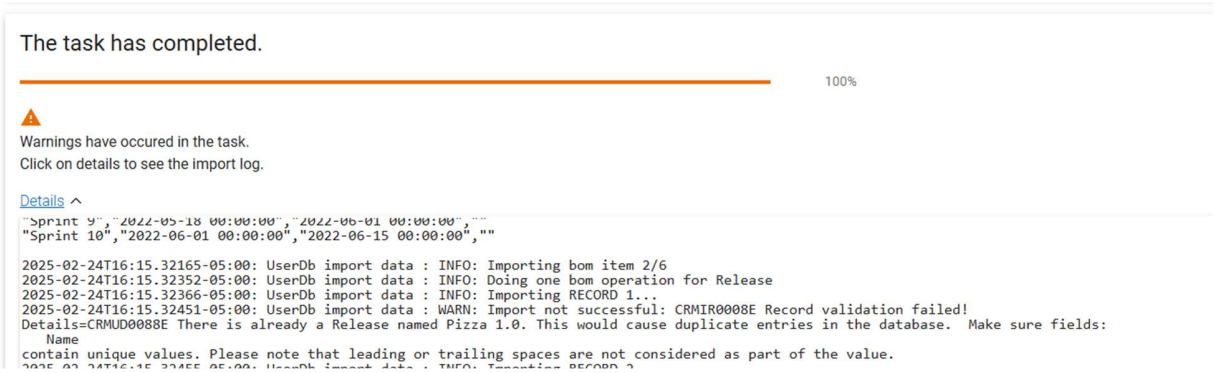
When you choose this method, you have the option of uploading your own zip with record data, or using the supplied sample data. The "Use sample data" option uses the same sample data that application creation uses.



When you click on Import, you will see a progress screen which gives you an idea of how much is left and whether warnings or errors have occurred.



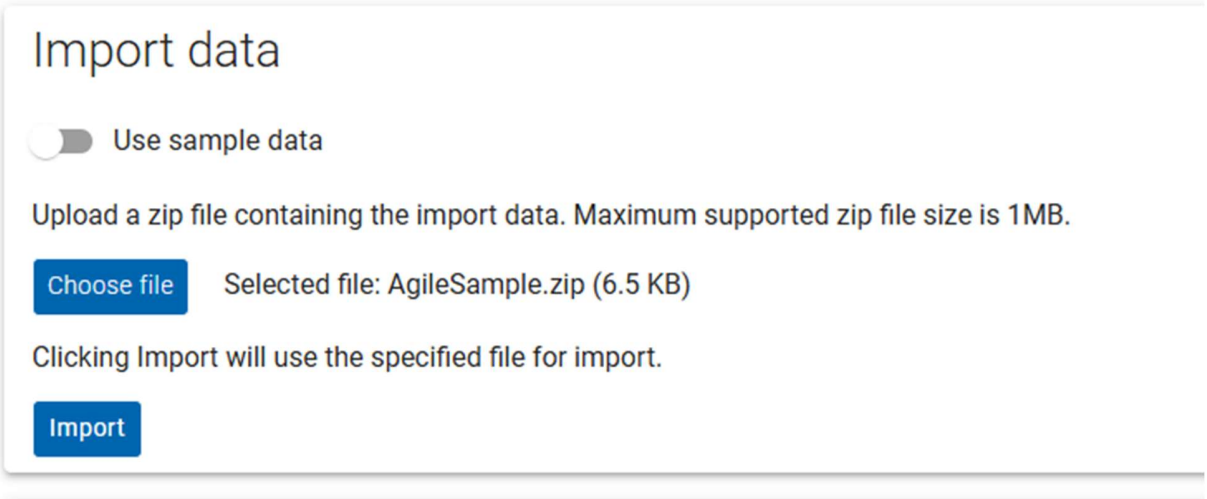
If a warning occurs, the progress bar will be a different color and a warning icon will appear:



You can scroll the details to look for “WARN” messages. In the above screenshot it is saying one of the imports failed because there was already a “Release” record named “Pizza 1.0”. This might be okay if you already have imported it or created it.

Uploading and importing data

The previous sections described how to import sample data that ships with the product. You may have some data that you want to import from another repository, or maybe you want to migrate from a different product to Plan, or maybe you have your own demonstration data you want to use. You can specify an upload during the Import process. To do this, you would switch off “Use sample data” and click on “Choose file”. You can then specify the zip file you want to upload for the import. The zip file is limited to 1MB. If you want to import larger amounts of data, you will need to do separate import operations.



Once you click “Import” you’ll see the same progress display as you did for importing sample data after application creation.

Import Data Format

This section will describe the expected format in case you want to build your own zips of data to import into Plan. The zip is made up of 3 types of files.

1) A “bill of materials” file, “bom.txt”, that describes all the csv files and what data the csv files contain.
2) A substitution file, “substitution.json”, file that you can use to transform any incoming data.
3) CSV files that contain the actual data being imported.

The Bill of Materials (bom.txt)

This file tells the importer where the data is in your zip file, and what record types the data represents. Here is an example bom.txt for the Agile workflow sample data:

```
records,Sprint,agile_sample_sprint.csv
records,Release,agile_sample_release.csv
records,Component,agile_sample_component.csv
records,Project,agile_sample_project.csv
records,WorkItem,agile_sample_workitem_epic.csv,agile_sample_workitem_
epic_history.csv
records,WorkItem,agile_sample_workitem_story.csv,agile_sample_workitem
_story_history.csv
```

Each line is a comma separate list of elements. The first specifies what kind of import the line describes. Currently only the “records” import type is supported. The second element is the record type that the data represents. You can specify both stateless and stateful records. For example, Sprint is a stateless record type in the Agile workflow. The third element on the line is the name of the csv file that contains the record data. The fourth element is optional and specifies the name of the file that contains history data for the imported records.
The importer will read each line and import the records in the order they are specified. This is important in the case where record data references other records. You want to make sure the referenced records are imported first.

The Substitution file (substitution.json)

You may find that the data you are importing needs some modification as you import it. For example, suppose you are importing demonstration data and you want the dates to be brought forward to a date relative to the time you are doing the demonstration, rather than whatever dates may be stored in the csv. In this case you can do a “relative date” transformation to ensure sprints and releases are planned in the future, and work item actions happened recently relative to the current date and time.
Here is an example substitution.json file for the Agile workflow sample data:

```
[
      {
            "pattern": "*",
            "strategy":"replaceExact",
```

```
            "map": {
                  "admin":"%CURRENT_USER%",
                  "lead": "%CURRENT_USER%",
                  "QE":"%CURRENT_USER%",
                  "user":"%CURRENT_USER%",
                  "engineer":"%CURRENT_USER%"
            }
      },
    {
        "pattern":
"action_timestamp|SubmitDate|ModifyDate|CloseDate|StartDate|EndDate",
        "strategy": "relativeDate",
        "inFormat": ["yyyy-M-d H:mm:ss", "M/d/yy H:mm:ss", "M/d/yyyy
H:mm:ss", "M/d/yy H:mm", "M/d/yyyy H:mm"],
        "outFormat": "yyyy-M-d H:mm:ss",
        "origin": "2022-01-26 00:00:00"
    }
]
```

Strategy: replaceExact

In this file, there are two transformation strategies that are executed on the incoming data. The first strategy is the “replaceExact” strategy. This strategy will replace an exact matching value with the one specified in the **map** structure. For example, “admin” will match but “Admin” won’t, and neither would “administrator”. The match must be exact. Furthermore, the strategy will operate on every field, since the **pattern** is “*” (wildcard). The replacement value is %CURRENT_USER% which is a dynamically calculated value and is equal to the currently logged in user performing the import. For sample data, we have to change all the users to the current user because the users specified in the sample data likely do not exist in the database yet, and import will fail.

Strategy: relativeDate

The second strategy used in this file is the “relativeDate” strategy. In this case the **pattern** is a regex of field names that the strategy will execute on. For example, the action_timestamp field on history will be transformed using the **relativeDate** strategy. The **relativeDate** strategy has 3 additional attributes to define its behavior. The first is inFormat, which is an array of date or date/time formats that will be used to interpret the incoming data. The format uses the DateTimeFormatter specification. For details on the specification, refer to “Patterns for Formatting and Parsing“ on this page:
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/time/format/DateTimeFormatter.html
The importer will attempt to parse the dates using this list of formats and in this order until it finds the first format that doesn’t cause a parsing error.
The second attribute is the outFormat, which specifies the format in which to import the data as. In this example we are using `"yyyy-M-d H:mm:ss"` as the format we import the fields as.

The last attribute is the one that specifies the origin for all the date times. All the incoming date times that match the field pattern will be interpreted relative to this date and then transformed to a date time relative to date time when you are doing the import. For example, if the date time being imported is “2022-01-25 00:00:00” (exactly 1 day before the origin), and you were importing the data when the current date time is “2025-02-25 12:00:00”, then it will be transformed to “2025-02-24 12:00:00”, or exactly 1 day before the current date time. Datetimes can be transformed both into the future and into the past.

Strategy: replaceAll

```
{
      "pattern":"UserMembers|RTEs|ProdMgrs|SysArchs|BusOwners|STEs|SolM
grs|SolArchs|PortfolioManagers|EnterpriseArchitects|EpicOwners|LPMs",
      "strategy": "replaceAll",
      "map": {
            "admin":"%CURRENT_USER%",
            "lead": "%CURRENT_USER%",
            "QE":"%CURRENT_USER%",
            "user":"%CURRENT_USER%",
            "engineer":"%CURRENT_USER%"
      }
}
```

The **replaceAll** strategy is similar to the **replaceExact** strategy, but rather than replacing an exact match, it will replace every instance of the mapped string with the replacement value in the map. As this example suggests, this strategy is useful for replacing strings in reference list fields. As in the case of replaceExact, you can use a dynamic variable like %CURRENT_USER% to replace the value with the currently logged in user.

Strategy: override

```
{
      "pattern":"ReleaseTrain.RTEs",
      "strategy": "override",
      "value": "%CURRENT_USER%,admin"
}
```

The **override** strategy operates like the replaceExact strategy, but rather than trying to match anything in the field, it will always override the value with the specified value in the strategy. In the above example, it will replace whatever ReleaseTrain has for the RTEs field with a list of two users, the currently logged in user, and the “admin” user (the built-in user admin user that can only be used by thick clients).

CSV Files

Now we come to the place where the data you want to import lives. The format of the files is CSV, or comma separated values. More specifically it follows the RFC4180 format, as described here:

https://datatracker.ietf.org/doc/html/rfc4180
The main things to consider when generating the CSV files is:

- The field delimiter is a comma, ",".
- The quote delimiter is double quote.
- The record separator is "\r\n" (carriage return, linefeed).
- Empty lines are not ignored.
- The first line of the file must be a list of fields corresponding to fields on the Plan record. The remaining lines are comma separated values, using double quotation marks as needed to capture multi line values and values that contain commas. If a double quote is used inside a value, it is escaped by preceding it with another double quote.

When importing stateful records, it is important to specify a unique identifier as the "id" value. This value is used during imports to ensure that references are re-established correctly. Since Plan generates its own id values as you import records, it needs to keep track of what the original "id" was of the imported record so it can re-establish those references. To do this, the import will store the "id" of the value in a field "old_id" which should exist on all the OOTB schemas supplied with the product. If you are using a workflow that has stateful records that do not have the "old_id" field, then the import of references may not succeed.

The above requirement is only applicable for references to stateful records. Stateless record "id" is automatically created from the records field values, so there is no need to have the "id" column for CSV files that contain stateless record data.