Creating Your First Ontology

Introduction

The Ontology

The data that is used most regularly in Foundry is presented as **objects**. For example, you can open a specific Airline object, Aircraft object, or Customer object and see the essential data (that you have permission to see) about that particular real-world thing. This object's data is typically aggregated from many different original data sources. For example, an Aircraft object might contain information pulled from purchase records, maintenance systems, flight systems, and more.

The **Ontology** defines the set of **object types** available. Each instance of Foundry has its own unique set of object types that are specific to that organization and the problems they face. For example, an airline or a military organization would typically have an Aircraft object type, whereas a pharmaceuticals company would not. Furthermore, the Aircraft object type might have different properties and different relationships to other object types at an airline than it does for the military. In this way, the Ontology reflects the real-world objects that users at a given organization think about and operate on every day.

The Value of the Ontology

The Ontology offers several benefits:

- It makes data easier to use.
- It provides a common operational vocabulary across workflows and across the organization.
- It supports operational decisions through well-defined actions.
- It powers AI solutions in an operational day-to-day context.

This Course

In this course, you will:

- Get hands-on with an existing Ontology to better understand what it is and how it makes data easier to use.
- See how an Ontology is built and how it provides a common vocabulary to an organization.
- Learn how to start building your own Ontology by defining your own object type, relationship, and action

- Learn more about how the Ontology lies at the heart of Foundry and how that position impacts how you should think about Foundry-based solutions

Prerequisites

Specifically, this Deep Dive: Ontology course assumes you are already familiar with the basics of:

- ingesting data into Foundry
- transforming and cleaning data using Pipeline Builder
- creating your first object type
- using objects to build a Workshop application

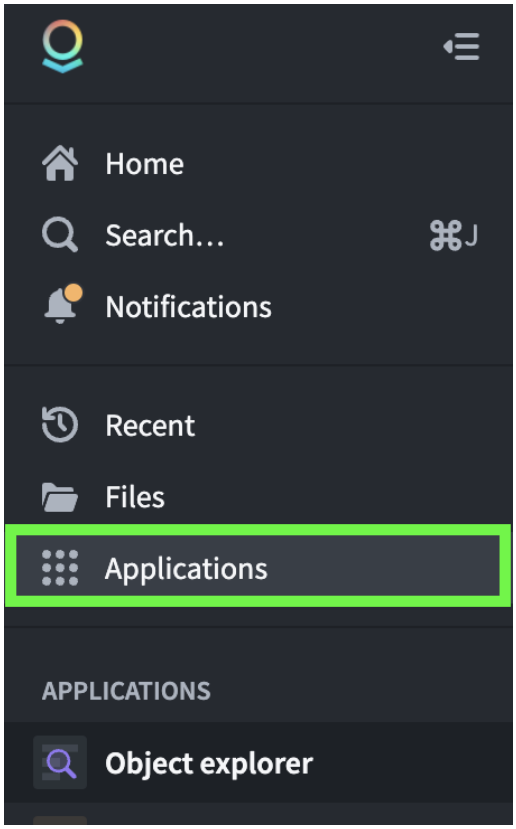
Install the Marketplace Bundle

This course requires a specific Marketplace bundle to be installed to provide the necessary data, pipelines, and object types.

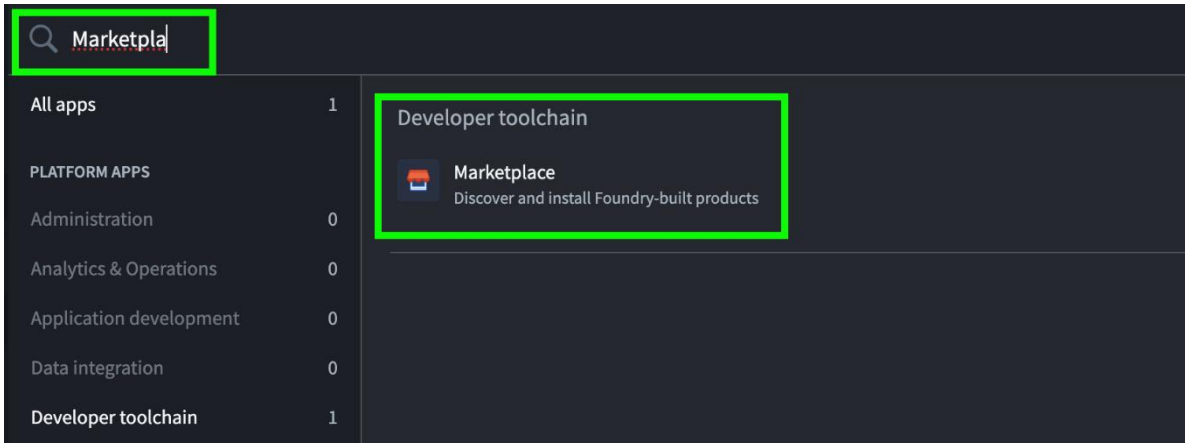
- If someone else has already installed it on your Foundry instance, you can likely reuse it.
- Otherwise, you will need to install your own copy.

Step 1: Open Marketplace

1. Click on *Applications* on the left side of the screen

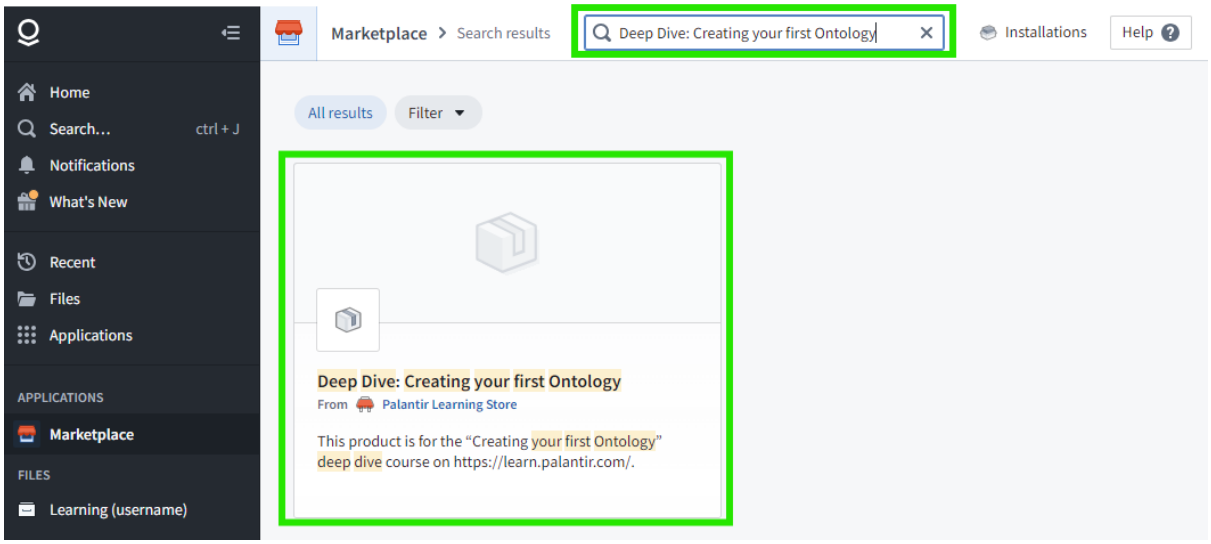


2. Search for *Marketplace* and open it.

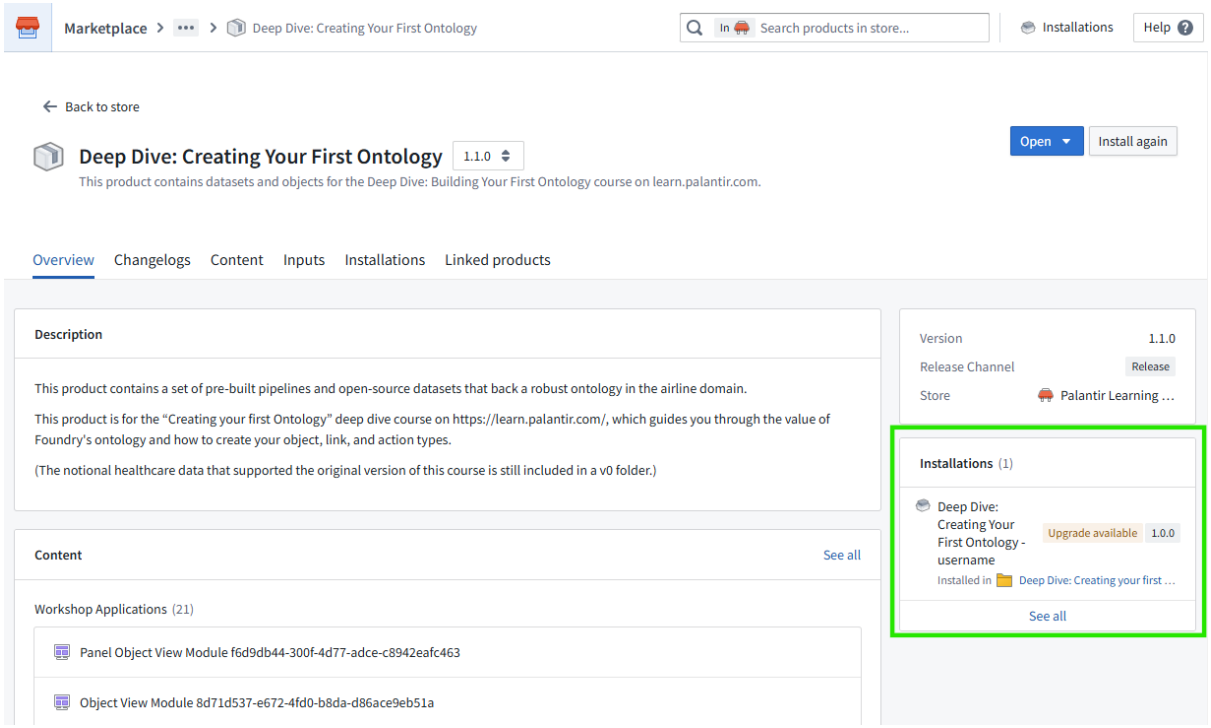


Step 2: Open *Deep Dive: Creating Your First Ontology*

1. Search for the **Deep Dive: Creating Your First Ontology** product in the top search bar labeled *Search products...* (not the lower search bar labeled *Search stores...*)
2. Click into the corresponding product



You can then check if there are any installations listed on the right side of the screen:

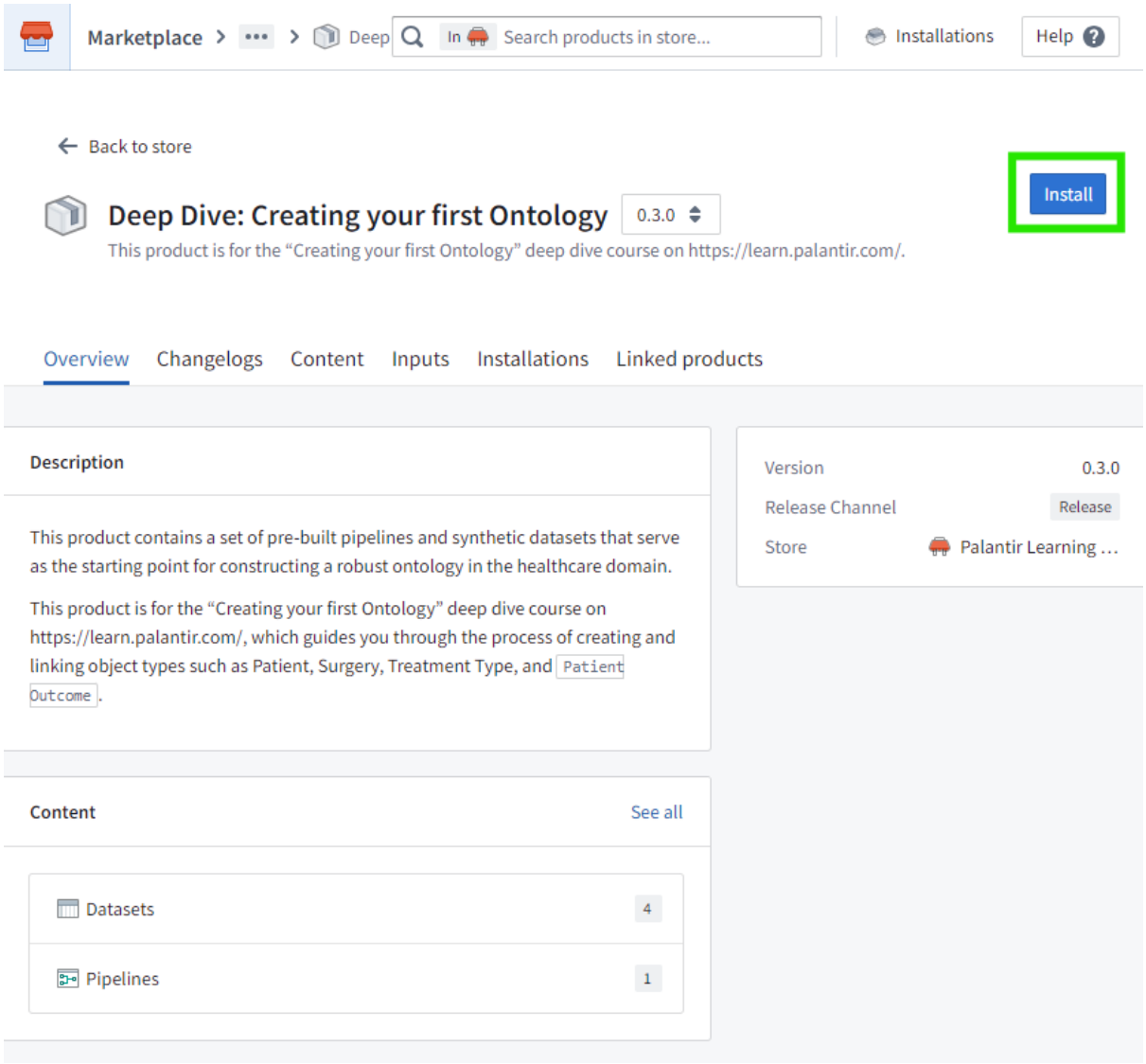


If there are any existing installations: You can skip the rest of this lesson and try using one of these installations starting with the *Object Explorer* section of this course.

If there are no installations OR if you later run into ontology permission issues in this course: Continue below with these instructions to install your own version of the course resources.

Step 3: Start the installation

1. Click *Install* or *Install again* in the top right corner



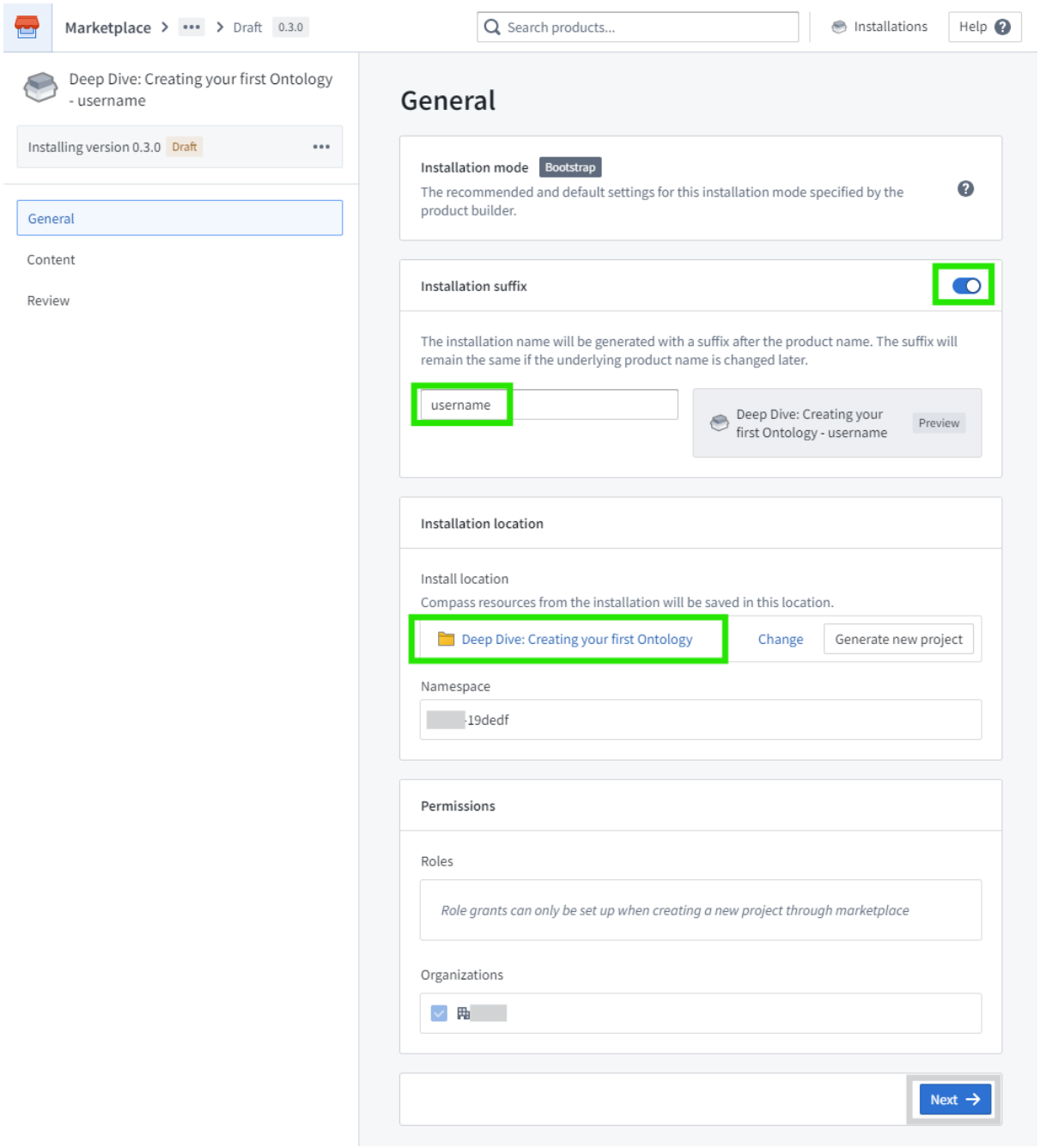
Step 4: Customize the installation

1. Set your username as the installation suffix
    1. Toggle on *Installation suffix*
    2. Enter your own <username>
2. Change the installation location to your course-specific training folder
    1. Click *Use existing location*
    2. Click *Browse*
    3. Browse to your *Learning (<username>)* project

- If you do not have one yet from completing other tutorials, click *Create new project* and create a project named Learning (*<username>*) where *<username>* is your own username.

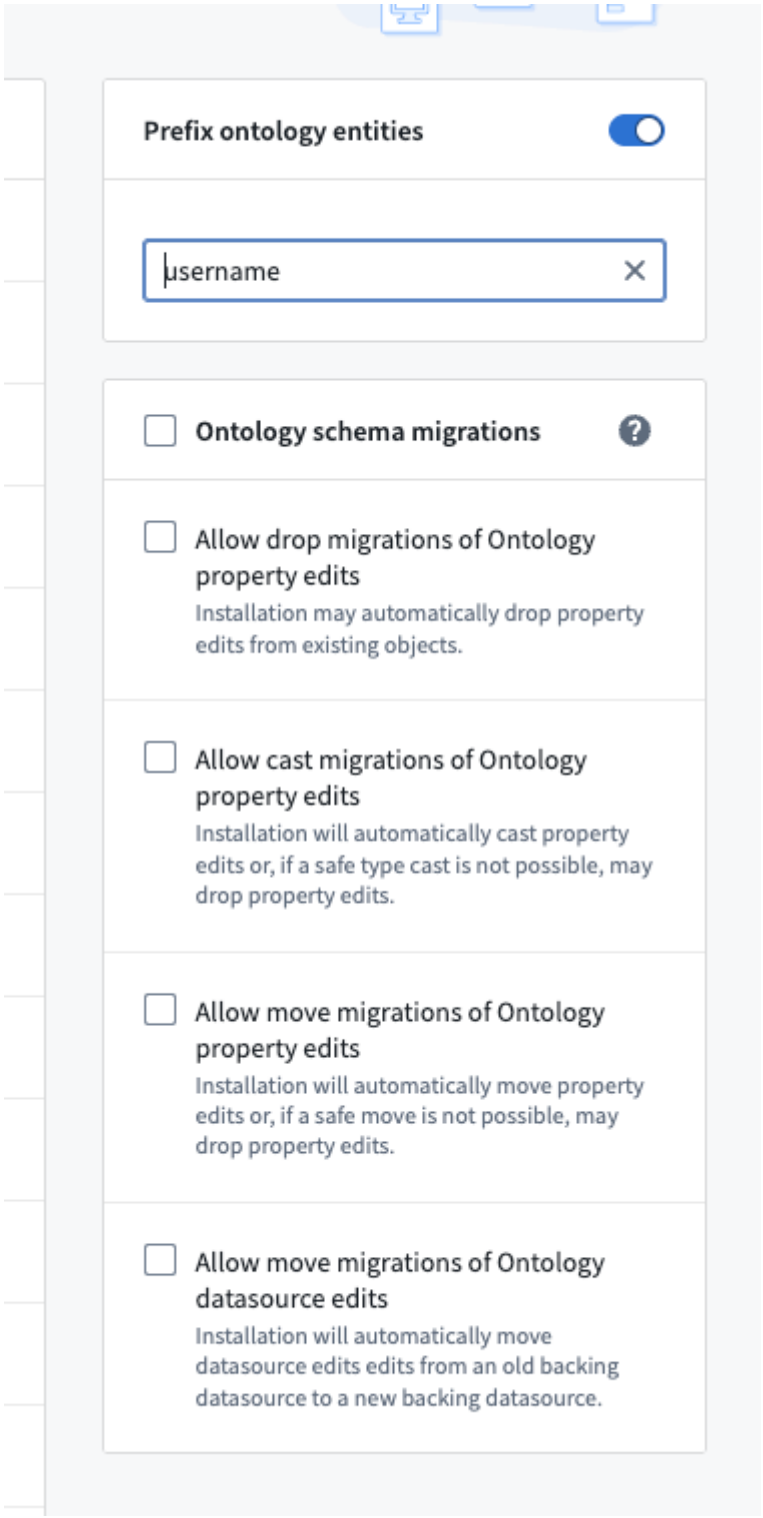
4. Create a new Deep Dive: Creating your first Ontology folder
5. Click *Save*

3. Click *Next*



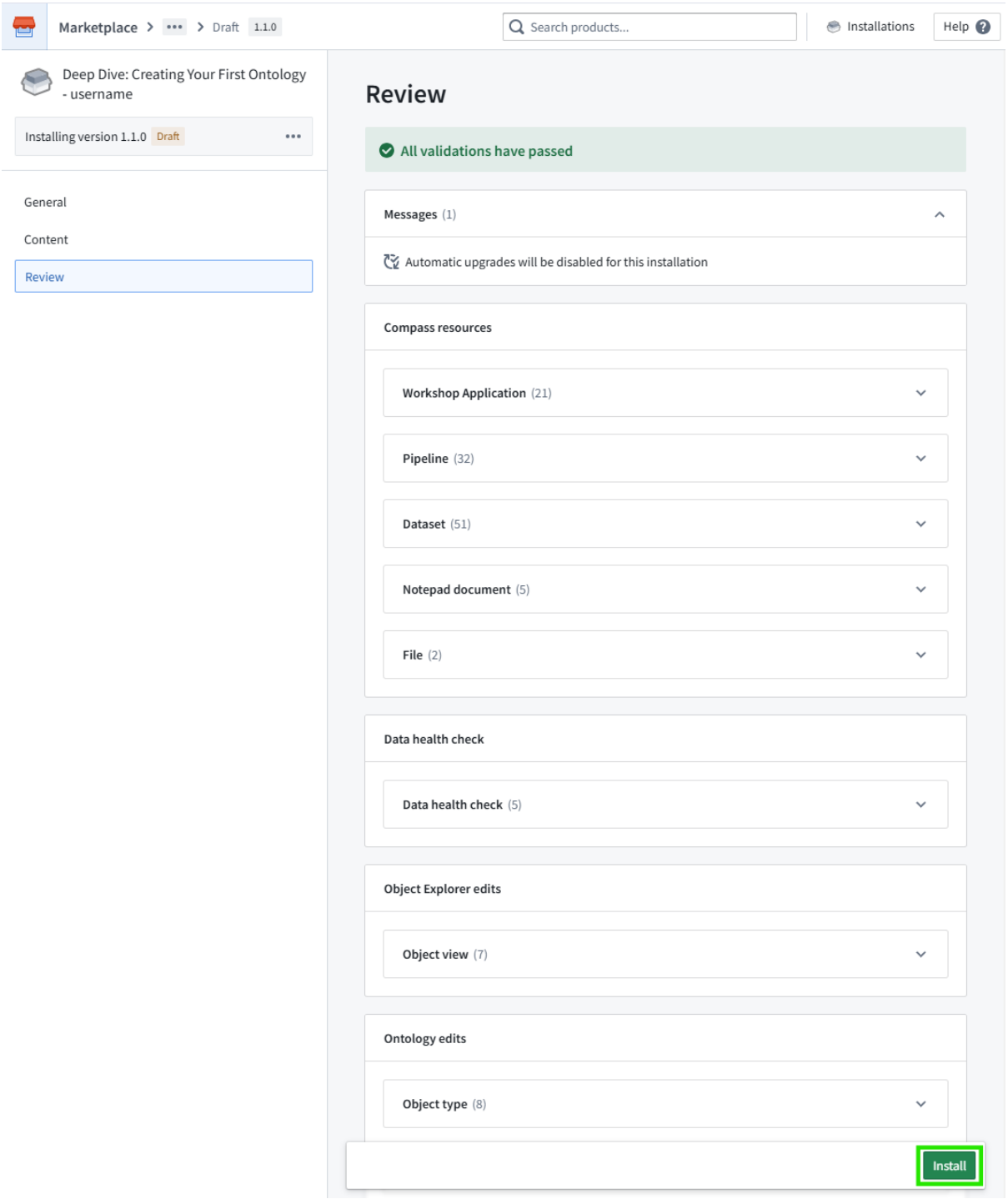
Step 5: Confirm the content

1. Decide whether to prefix ontology entity names
    1. If there are many existing installations, you can *Prefix ontology entities* with your username.
    2. If this is the first installation of this product, you can skip this.
2. Click *Next*



Step 6: Install

1. Ensure that all validations have passed
2. Click *Install*



Step 7: Complete the installation

1. Wait for the installation to finish

- It generally takes 5 to 8 minutes to install resources
- It then takes 5 to 10 more minutes to build them
- And potentially another 10 minutes or so (not shown in the UI here) to finishing indexing all object data into the object layer

Object Explorer

Context: Fresh Air

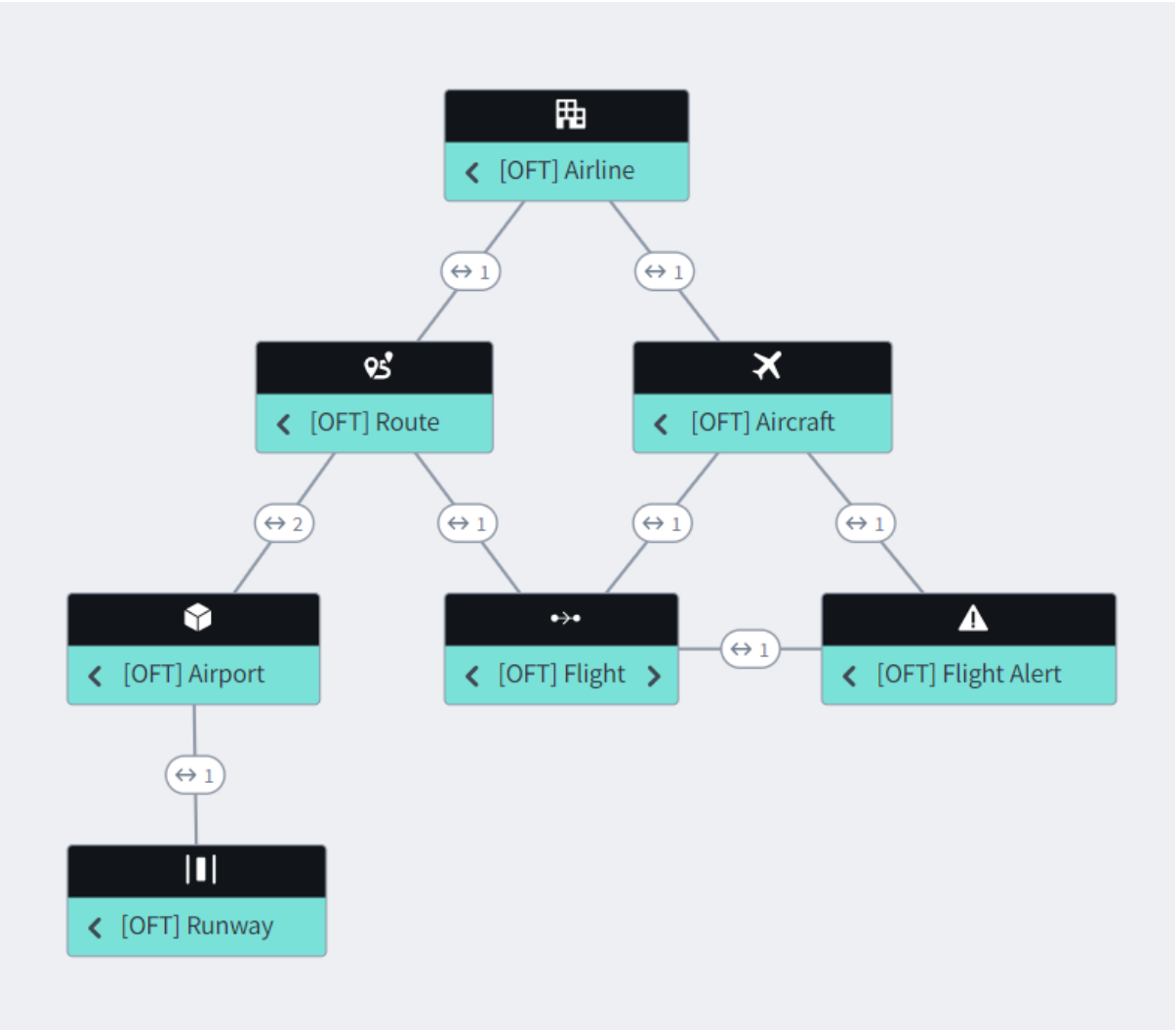
Fresh Air

For the the following exercise, assume that you are a new employee at Fresh Air. Fresh Air is a (notional) regional airline of approximately 4,500 employees that services the eastern half of the United States. They currently fly approximately 500 flights a day to 83 destinations. Although technically headquartered in Fort Worth TX, their busiest hubs are all on the east coast.

To start, assume you have just joined the Flight Operations department. Later in this course, we'll see what the same data looks like from the perspective of other roles and departments.

Ontology

As a new employee at Fresh Air, you are not familiar with all of their data systems yet. However, from your new hire orientation, you know some of the basic terms of the industry. These types are present in Fresh Air's Ontology:



In summary, an **Airline** operates a fleet of **Aircraft**. These Aircraft regularly fly from one **Airport** to another, where they land or takeoff from a given **Runway**. The connection between any two Airports along which Aircraft regularly fly is a **Route**. A **Flight** is a specific instance of an Aircraft flying along a Route at a particular date and time. Occasionally, a Flight might encounter an issue, such as delay, cancellation, or even a diversion to another airport that is not normally part of the route. These non-standard events generate a **Flight Alert**, which the Flight Operations team then responds to.

The data we will explore using this Ontology includes details from all US airlines, not only from Fresh Air. Because these are common object types beyond the bounds of this specific course, the object types include an **OFT** (Ontology Foundry Training) prefix to avoid any naming conflicts with any similar types found in your ontology.

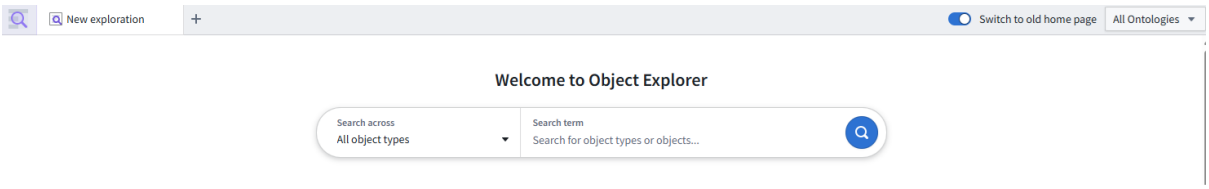
Object Explorer

There are many Foundry applications that let you work with objects and the Ontology. We will start with the simplest of them, which is Object Explorer. **Object Explorer** is used for object searches and for basic exploration and analysis across sets of objects.

Step 1: Open Object Explorer

1. Open **Object explorer**
    - You will find this in the Applications menu in the left side bar

You should then see something like this:

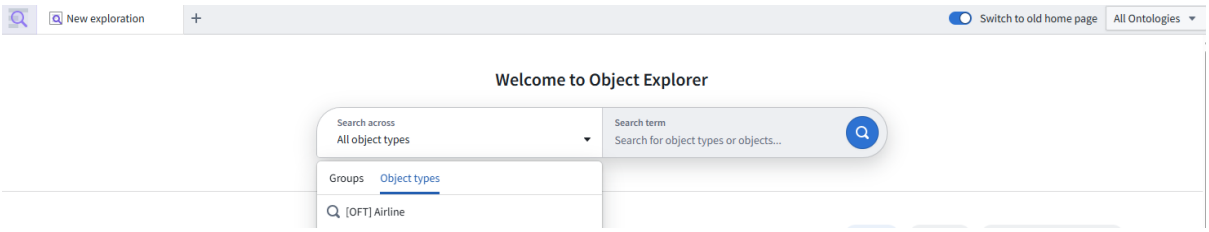


Fresh Air Operations

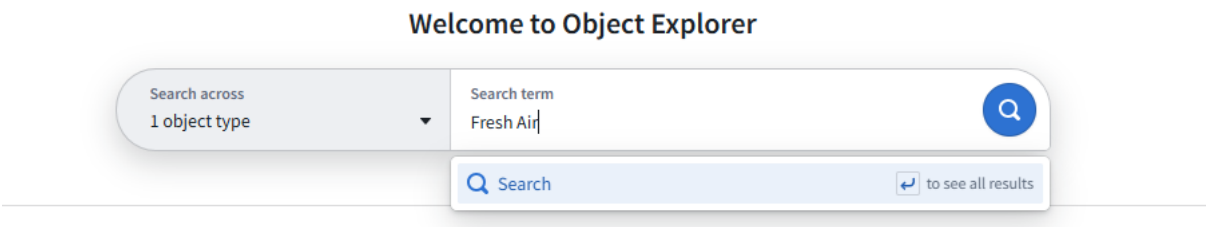
To start, find and open the **Fresh Air Inc** object. Its object type is **[OFT] Airline**. Because there may be a lot of data and many object types in a given Foundry instance, this search will be more reliable if you filter down to the object type first.

Step 1: Open Fresh Air Inc

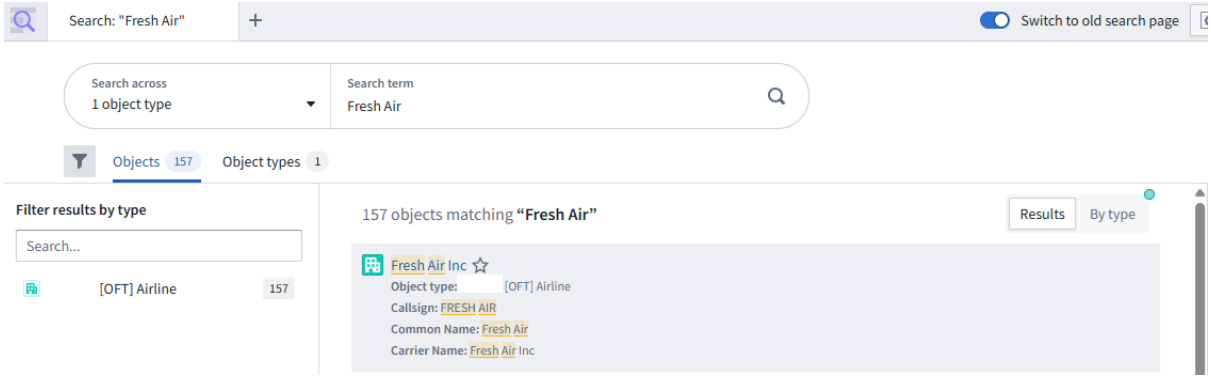
1. Change the **All** drop-down at the left of the search bar to the object type **[OFT] Airline**



2. Search for **Fresh Air**

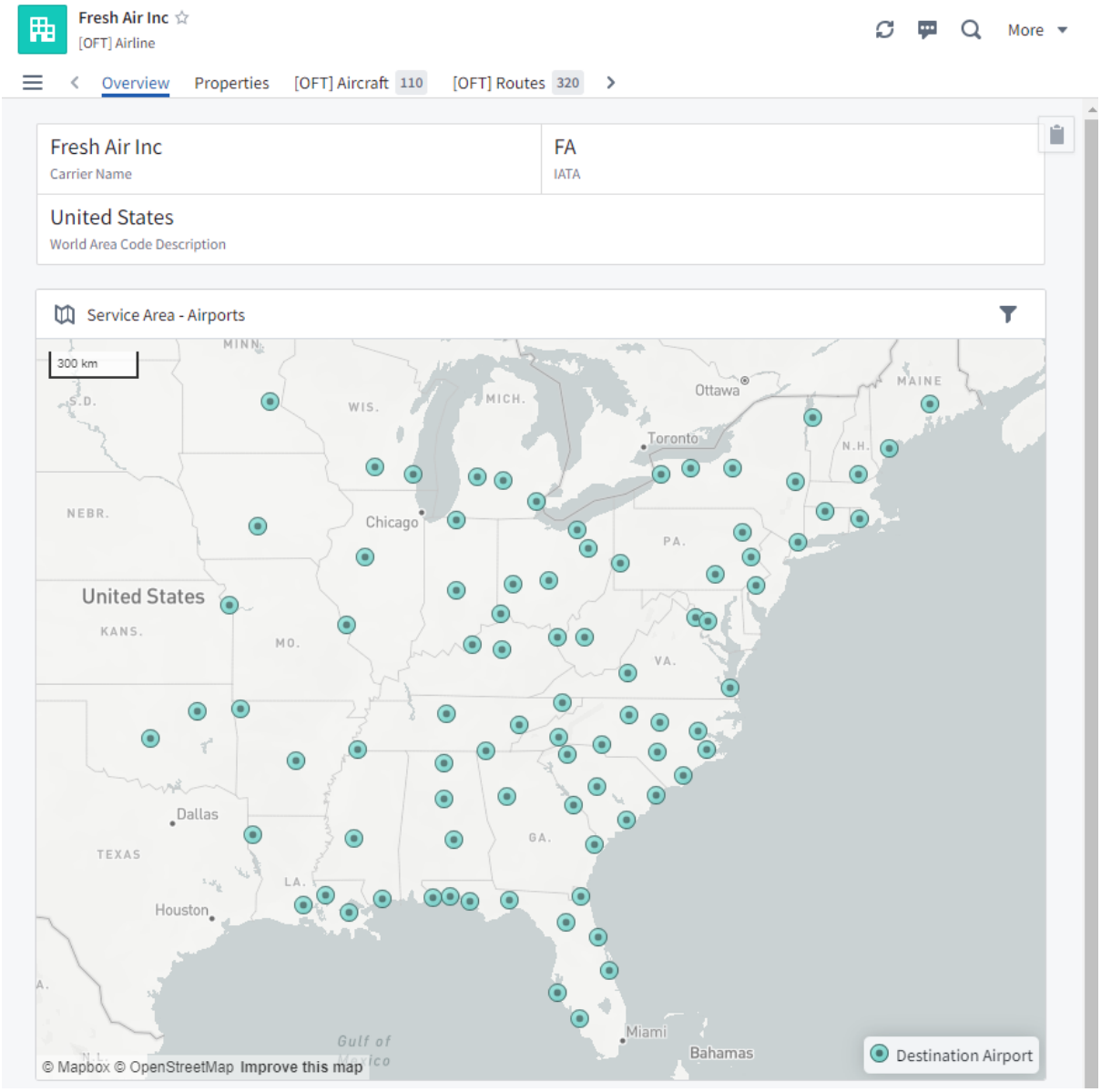


3. Click on the **Fresh Air Inc** object in your search results



Help: I can't find these object types!
The necessary marketplace bundle must be installed on your Foundry instance, the installation needs to be fully built to create all the necessary objects, and you then need permission to the see those objects and their data. See the previous *Getting Started* section for more.

This should open an object that looks like this:



This is an example of a single Foundry object. Besides property and links, the view of a object can be augmented with additional derived properties and visualizations. Note that this object view includes multiple tabs along the top and links to additional objects.

Single Object

Using only the **Fresh Air Inc** object in Object Explorer, answer the following questions. Highlight the space after each *Hint* or *Answer* to reveal them.

What is Fresh Air's IATA code?

Hint: The IATA code is found near the top of the *Overview* tab or on the *Properties* tab.

Answer: FA

What is Fresh Air's ICAO code?

Hint: An airline's ICAO code is less frequently used than its IATA code, so this property is only displayed on the *Properties* tab.

Answer: FRS

What does ICAO stand for?

Hint: You can learn this by mousing over of the ICAO property on the *Properties* tab.

Answer: International Civil Aviation Organization

Who is the single manufacturer for all of Fresh Air's aircraft? Fresh Air has a fleet of 110 aircraft. While it has two different models of planes, all of those planes come from a single manufacturer.

Hint: You can learn this from the *Aircraft Fleet by Manufacturer & Model* chart on the *Overview* tab or by scrolling through the *Aircraft Fleet* tab.

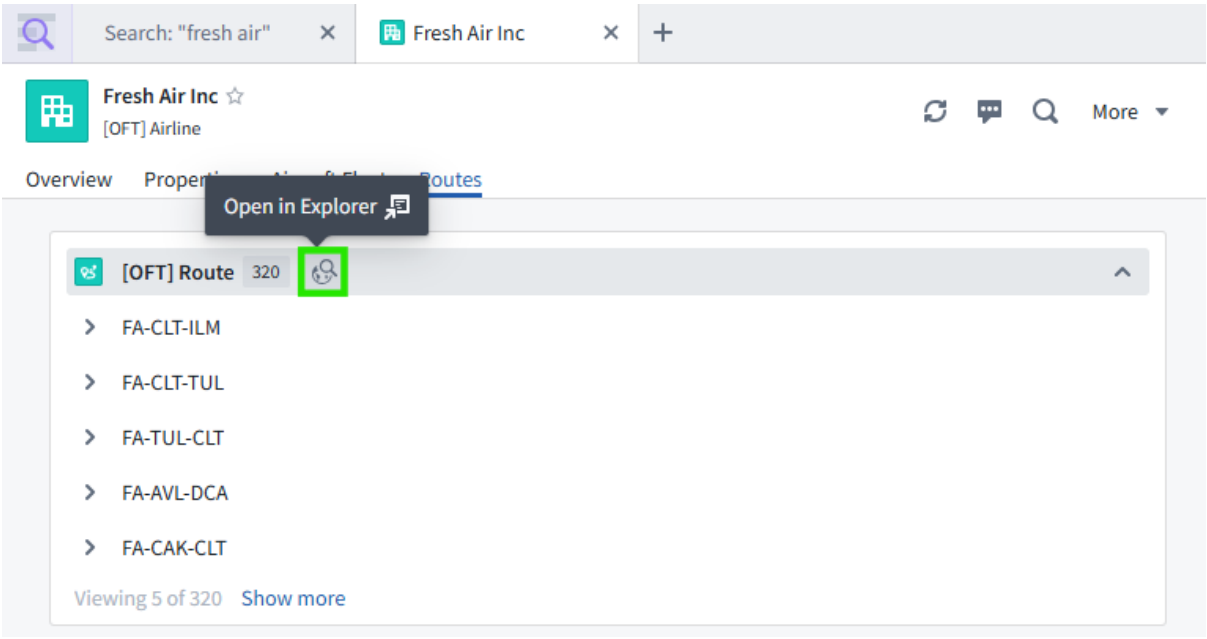
Answer: Bombardier

Fresh Air’s Routes

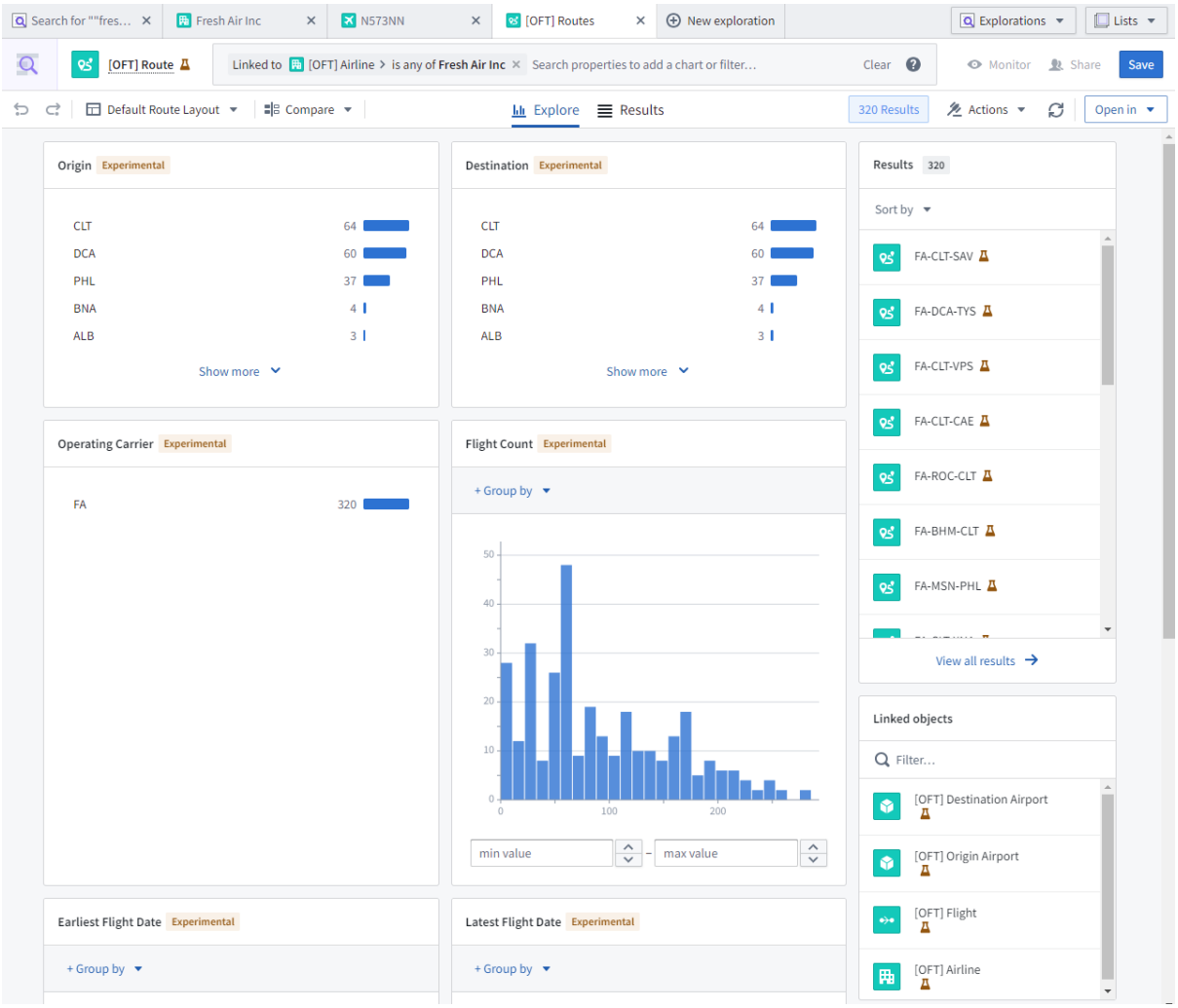
So you can learn a lot of from a single object, but you can learn much more by traversing an object’s links and by exploring object sets.

Step 2: View Fresh Air’s Routes in a new search

1. Start with the **Fresh Air Inc** object
2. Switch to the **Routes** tab
3. Hover over the top of the list of Routes and click **Open in Explorer**



You should now see an exploration of these **320 routes**:



The results count is listed in the top right of the screen.

Explorations

An *exploration* lets you explore a set of objects through a variety of charts. Each chart is derived from one of the properties of that object type. Some properties are charted by default, but it is possible to edit which properties are charted, reorder the charts, and add new charts.

As suggested by the Route titles listed in the *Results* panel on the right, a Route object is an origin-to-destination path, as serviced by a particular airline. For example, FA-CLT-ALB is the FA (Fresh Air) route from CLT (Charlotte, NC) to ALB (Albany, NY).

To get comfortable with explorations and the kinds of customizing you can do, answer the following questions. They can all be answered using only the **Origin** chart.

If you happen to accidentally delete a chart while answering these questions, click the *Undo* icon in the top left of the screen.

Which three airports together serve as the origin for the greatest number of Fresh Air routes? That is, what are top 3 origins for Fresh Air routes?

Answer: CLT (64), DCA (60), and PHL (37)

How many routes originate from the DAY airport?

Hint: One option is to click *Show more* until you find DAY in the list. However, if you mouseover the *Origin* chart, a number of control icons will appear in the upper right. Use the *Filter chart* (magnifying glass) icon found there to find this entry.

Answer: 3

Don’t forget to clear any filters you’ve applied to the Origin chart before answering the next question.

What percentage of Fresh Air's routes originate from CLT?

Hint: Use the *Show percentages* icon within the *Origin* chart.

Answer: 20.0%

For this last question, you’ll need to combine a couple charts. Don’t forget to clear any filters first.

Are there any Fresh Air routes that don't involve one of the top 3 airports as either an origin or destination?

Hint: Select the top three bars in the *Origin* chart. Change *Keeping* to *Excluding* and click *Apply filter*. Do the same for the *Destination* chart.

Answer: There are only 2 routes that do not include CLT, DCA, and PHL in some way: BNA-RDU and its return route of RDU-BNA.

Traversing Links

When you are exploring an object set, you can use the **Linked objects** entries in the lower right to switch to all of the linked objects according to some relationship link. For example, suppose we're starting with all 320 of Fresh Air's Routes. You could then switch to all of the Origin Airports for those Routes, or you could switch to all of the Flights that occur along those Routes.

For example, let's see how many Origin Airports are serviced by Fresh Air.

Step 3: Traverse links to the set Fresh Air's origin airports

1. Confirm that you are starting with 320 Results (in the top right of the screen) for Fresh Air's Routes
    1. Clear any other filters you have.
2. Click **[OFT] Origin Airport** in the lower right

Your set of charts should change because you are now looking at a new object set of Airports (not Routes).

This process of traversing links from one object set to another is often called *pivoting* from one object set to another. (If you are already familiar with pivot tables, I recommend treating this as an unrelated concept with the same name. Trying to think of link traversals in terms of pivot tables will only lead to more confusion than if you start with a fresh mental model of the process as described above.)

You can confirm that Fresh Air has the same number of **Destination Airports.**

Step 4: Traverse links to the set Fresh Air's destination airports

1. Undo the pivot that you just did by clicking the *Undo* arrow in the top left of the screen
2. Click **[OFT] Destination Airport** in the lower right

Challenge Puzzles

If you'd like more practice, here are some optional questions to answer about Fresh Air's operations:

What is the highest elevation origin airport that Fresh Air services? Does it have a runway of at least 7,000 feet?

Hint: From Fresh Air's 320 routes, pivot to Origin Airports. Then filter down Altitude in Feet and open the single result.

Answer: AVL (Asheville, NC). Yes, it has a single runway that is 7,001 feet of grooved asphalt that runs 170/350 degrees (nearly north/south).

What is the Fresh Air origin airport that is the located the farthest west?

Hint: This is easy to answer if you go back to the Fresh Air Inc object and just look at the map on the *Overview* tab. Alternatively, while exploring the set of (Fresh Air Inc > Routes > Origin Airports), go to the top search bar and search for the Longitude property. Add this chart to your view and then filter down to the most negative (westmost) value.

Answer: OKC: Will Rogers World (Oklahoma City, OK)

Reflections

Hopefully this experience of using Object Explorer has shown how **easy** it is to work with Foundry data in real-world terms. Without searching for any backing tables, thinking about join keys, or using a query language, you were able to quickly learn a great many details about Fresh Air's operations, such as:

- Their IATA and ICAO codes
- The make-up of their aircraft fleet (2 models of Bombardier aircraft)
- Although Fresh Air services 84 different airports, nearly all of their routes start or end in one of three three hub airports (CLT, DCA, and PHL). The only exception is the BNA-RDU/RDU-BNA route.
- If you completed the challenge puzzles, you were able to find highest airport and confirm that it has a one runway that is long enough to accommodate Fresh Air's aircraft.

Object Explorer is a quick way for any user, regardless of technical background, to get started using objects. Using only point-and-click controls, you can search for single objects of interest, follow links to related objects, and explore a singe object set in detail.

Object Explorer is only one place were objects are used in Foundry.

- For more complex object-based analysis—such as doing richer numerical analysis, building visualizations, deriving timeseries, comparing multiple object sets, or building interactive reporting dashboards--you should consider Quiver instead.
- On the other hand, if you have a regular operational workflow, building a dedicated Workshop application (as you did as part of the earlier End-to-End Speedrun course) will provide users a smoother and more tailored tool than relying on Object Explorer alone.

Now that you have some idea why Fresh Air's ontology is useful to individual end users, let's see why it's valuable to data engineers and the organization as a whole.

Data Lineage

Context: Why data lineage matters

With many people working within the same platform, it can get difficult to track the current state of data. Their work might overlap without them actually speaking to each other.

A data engineer might build a pipeline to clean up the data brought in from an external system. A data analyst might augment that clean data with manually uploaded data from a different data source to build out a view specific to her use case. A year later, another colleague continues that data analyst's work in light of a new business focus.

As layers and layers of this collaborative work happen over years with a rotating cast of users, the current state of any given dataset and its relationship to others in Foundry could become very confusing.

The Data Lineage tool is intended to provide a bird's-eye view of how datasets, object types, and certain resources are connected to each other in Foundry. You can also learn a lot about the state of each dataset, as well as take action on them (such as rebuild a dataset that is out-of-date with its ancestors/inputs).

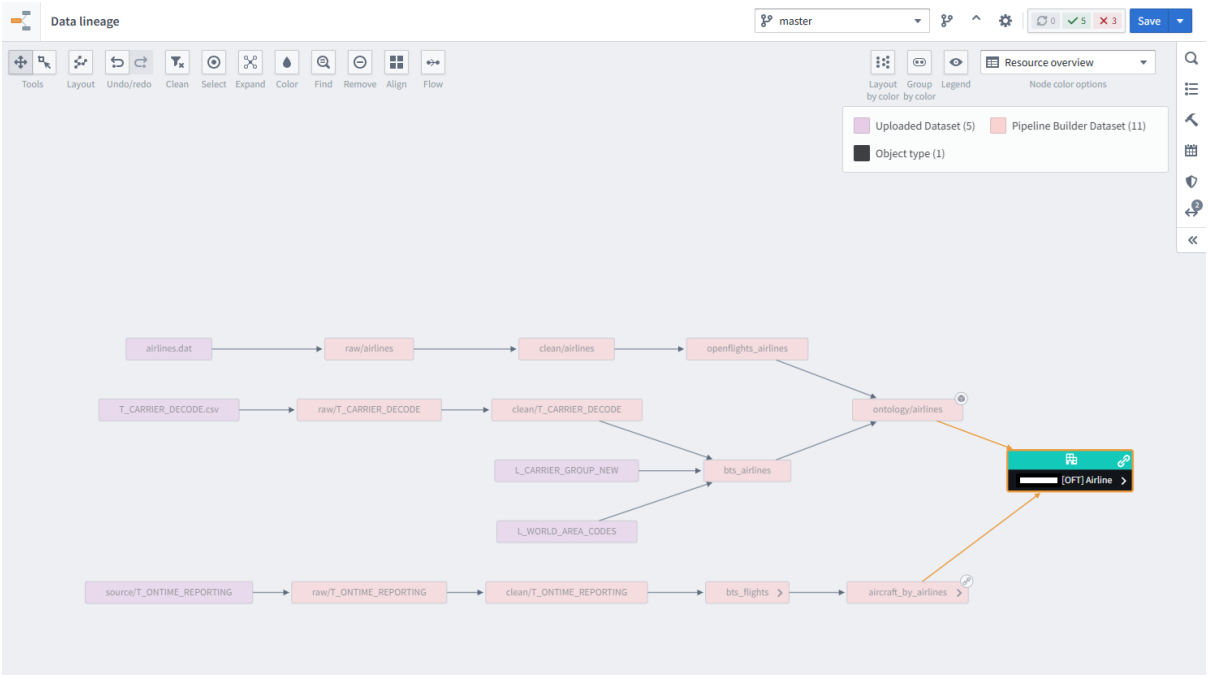
While Data Lineage is mostly often used by data engineers, it can be useful to any role in Foundry.

Data Lineage: Fresh Air pipeline

Step 1: Explore the data lineage of an Airline

1. In Object Explorer, open the **Fresh Air Inc** object
2. In the top right, select **More > Advanced > Explore data lineage**

You should then find yourself in the Data Lineage application:



Step 2: Add other [OFT] object types

1. In the right side panel, switch to the **Search Foundry** tab (which has a magnifying glass icon)
2. Select **Object types**
3. If prompted, select your ontology
4. Filter by **[OFT]**
5. Select each of the 7 object types to add them to your graph - Airline, Aircraft, Route, Flight, Airport, Runway and Flight Alert.

Step 3: Expand all nodes

1. Select all nodes on your graph.

1. You can do this manually with your mouse by holding Shift and dragging a box over all nodes or you can press Ctrl+A / Cmd+A your keyboard

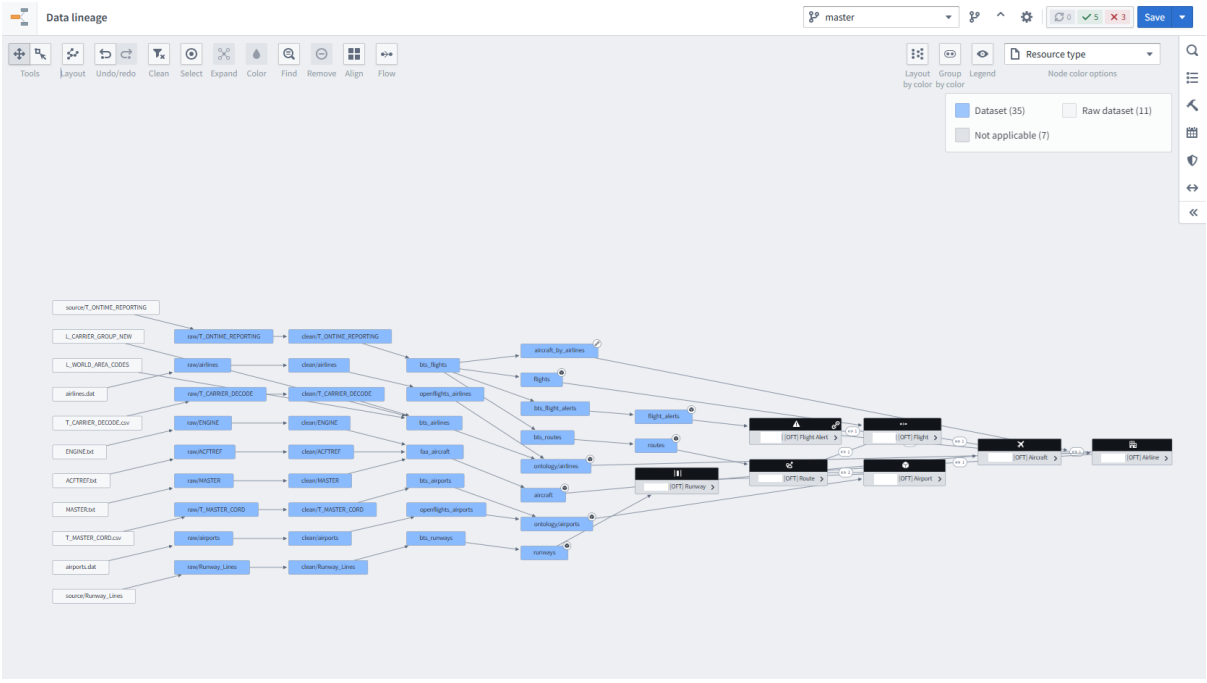
2. Click the **Expand** button in the top toolbar
3. Click the **<<** button to **Expand parents**
4. Click **Add 30 nodes**

Step 4: Arrange all nodes

1. Select all nodes again
2. Click the **Layout** button in the top toolbar
3. Click **Layout all nodes**

You can also reposition nodes manually.

At this point, you can see that you have 11 uploaded (raw input) datasets that produce a pipeline of 35 other datasets, leading to 7 object types.



Data Lineage Exploration

In the top right of the Data lineage graph is a **Node color options** drop-down. It defaults to **Resource overview** for new graphs. You will need to change that to **Resource type** within the drop-down to answer the following questions:

Are all of these datasets located in the same project?

Hint: Select *Project* in the *Node color options* drop-down.

Answer: Yes, the datasets are all in the same project.

Were all of these datasets produced by logic or code found in the same repository?

Hint: Select *Repository* in the drop-down.

Answer: No. Every dataset is produced by a different repository.

There is a tool panel found at the bottom of the screen and another on the right. Use those two panels to answer these questions:

How many rows and columns make up the ontology/airports dataset that backs the [OFT] Airport object type?

Hint: Once you select the ontology/airports dataset on your graph, you can use the *Preview* tab in the bottom panel or the *View node properties* tab in the right panel.

Answer: 4.3k rows and 18 columns

Can you view the logic that was used to derive the ontology/airports dataset? What application was used to define that logic?

Hint: You view the *Code* tab in the bottom panel or find an *Updated via* link in the right side panel.

Answer: Pipeline Builder. You should be able to open it to see the logic (transforms) defined there.

This only scratches the surface of what you can learn about a pipeline's state using Data Lineage, but it is sufficient for this exploration of Ontology.

Reflections

You've now seen the many datasets that feed into the Fresh Air Ontology, what does this context mean for your understanding of the value of the Ontology?

Ease of Use

First of all, consider again the ease of use that the Ontology provides for data. Imagine answering the same questions you answered previously in the Object Explorer module but using only this graph of datasets and any dataset-based query or analysis tools. The first challenge would be deciding which datasets to start with and then how to combine them to produce a complete picture for any particular question. That work has already been done by the data engineer(s) who built this pipeline and the Ontology.

Common vocabulary

Secondly, we can see here how the Ontology provides a common vocabulary across an organization. For example, suppose the Fresh Air Flight Operations team used to use the Bureau of Transportation Statistics data as their canonical list of airports, while the Fleet Maintenance team used the list provided by OpenFlights. It is easy to have such discrepancies when working from your preferred dataset. However, given that there should really be only one Airport object type in the Ontology, this forces some discussion and resolution between these teams to establish a common ground truth.

This established common vocabulary then becomes the obvious destination for any significant future work. For example, any new datasets about airports should not be brought into Foundry and left as simply another available dataset. Instead, it should be merged into the Airport object type. (It may be that not every user should be able to see all details about every Airport object. However, it is possible in Foundry to apply both row and column-level permissions when defining the backing data sources for an object type so that you can still have a single Airport object type that presents only the data each user is allowed to see.)

Similarly, any significant analytical insights, useful aggregations, or model-derived recommendations can be added to the relevant object type. This makes this work centrally accessible to users across the organization, rather than languishing as forked or siloed local work.

Because object types can grow richer and more detailed like this over time, it becomes much easier to bootstrap new workflows and use cases that rely on the same object type. For example, once an Airport object type is created to support a Flight Operations workflow, that object type can be reused (and augmented) to build a new Fleet Maintenance workflow.

Some of the advantages of a common vocabulary could be achieved by simply marking certain datasets as the “golden tables” for a given concept. (For example, Foundry’s Data Catalog can help do this.) However, in practice, this does not prove to be as effective in controlling the entropy produced by diverse workflows. For example, as a user looking for the authoritative dataset on airports to use as a basis for a new workflow or analysis, you might find airports, but you’ll often find a set of datasets derived from that one with names

like airports_with_geojson, airports_fresh_air_staff_enriched, and better_airports. While Data Lineage can help you understand the provenance of each of these datasets and how each of them have improved or modified the golden airports table, this work is effectively leading to local variants and dialects in what is intended to be an authoritative common vocabulary. Because the Ontology provides a object layer separate from the dataset layer, it can provide stronger controls over these types.

Ontology Manager

Context: Extending Fresh Air's Ontology

Now that you've seen the structure and some of the value of the Fresh Air Ontology, let's extend it.

As mentioned earlier, whenever a Flight is delayed, cancelled, or diverted, it would be useful to have an object to represent this situation. Each Flight Alert can then be responded to by Fresh Air's Flight Operations department. This is an example of an operational workflow in Foundry.

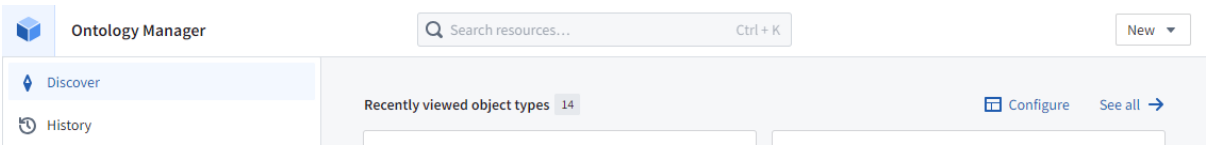
In this section, you'll build your own version of the Flight Alert object type. (You can use the existing version as a reference if you get stuck.) This will include a link to the relevant Flight and an action to update the root cause of the alert.

Ontology Manager

Step 1: Open Ontology Manager

1. Open **Ontology Manager**
    1. This can be found by searching the Applications menu found in the left side bar

You should then see an app header that looks like this:



Ontology Manager (sometimes called the Ontology Management Application or OMA) enables you to build and maintain your organization's Ontology.

While you can also define object and link types from Pipeline Builder (as you did in the End-to-End Speedrun course), Ontology Manager offers many more capabilities and options.

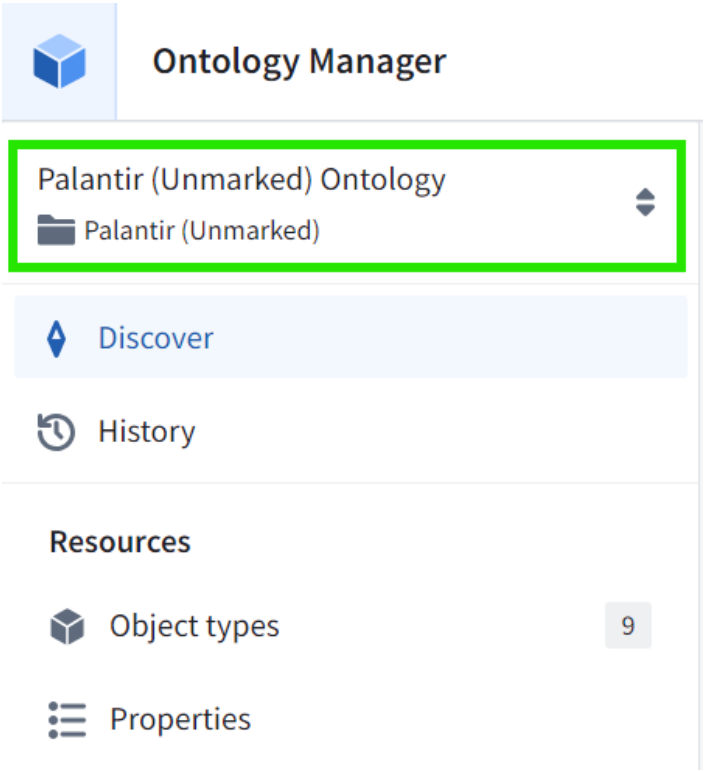
Because you can currently (at the time of this writing) only edit ontology type definitions from the application you used to originally create it, it is generally recommended that you use Ontology Manager as your default ontology tool, unless you know that Pipeline Builder's lighter feature set will be sufficient for your needs.

Create an object type

Aside: Permissions

You may not have permission to create new object types in the default Ontology at your organization. If you run into an issue creating a new object type in the next step, see the *Permissions & Help* lesson early in this tutorial, the *Install the Marketplace Bundle* lesson, and/or try these two potential fixes:

1) Change the current Ontology. In the top left corner of the Ontology Manager application, there is a drop-down to select between different Ontologies (if more than one is available). If you explore this drop-down, you may find an Ontology that you have permission to edit and that contains the rest of the Fresh Air (OFT) Ontology.

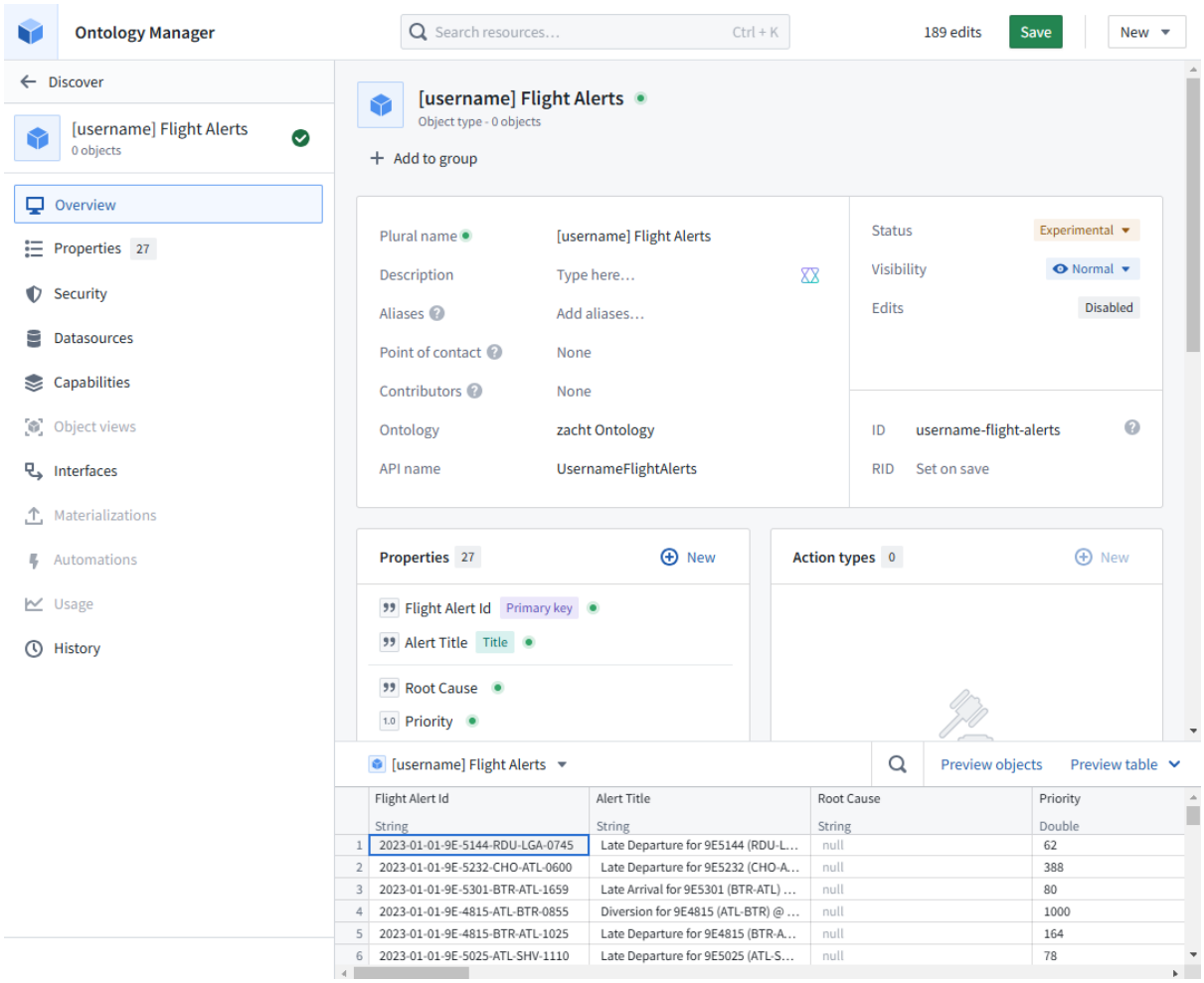


2) Use Ontology Proposals. Some organizations limit edits to the production Ontology via Ontology Proposals. If you are unable to create new Objects in an available Ontology, you may need to create a custom branch for the steps below and seek approval from your organization's Administrator for merging in the changes.

Step 1: Create Flight Alert

1. Click **New > Object type** in the top right corner of the screen
2. Select the **flight_alerts** dataset from the **Deep Dive: Creating Your First Ontology** folder as an existing datasource
    1. Click **Use existing datasource**
    2. Click **Select datasource**
    3. Search for the **flight_alerts** dataset found in you **Deep Dive: Creating Your First Ontology/data/flight alerts/ontology/** folder
    4. Click **Next**
3. Name your object type [<username>] Flight Alert (where *<username>* is your own username)
    1. For example, if your username is jsmith, you would create [jsmith] Flight Alert
    2. The **Plural name** should update accordingly
    3. Click **Next**
4. Set the **Primary key** and **Title** properties
    1. Set **Primary key** to Flight Alert Id
    2. Set **Title** to Alert Title
    3. Click **Next**
5. Skip creating any generic action types
6. Click the green **Create** button

This should result in an overview of your new object type:



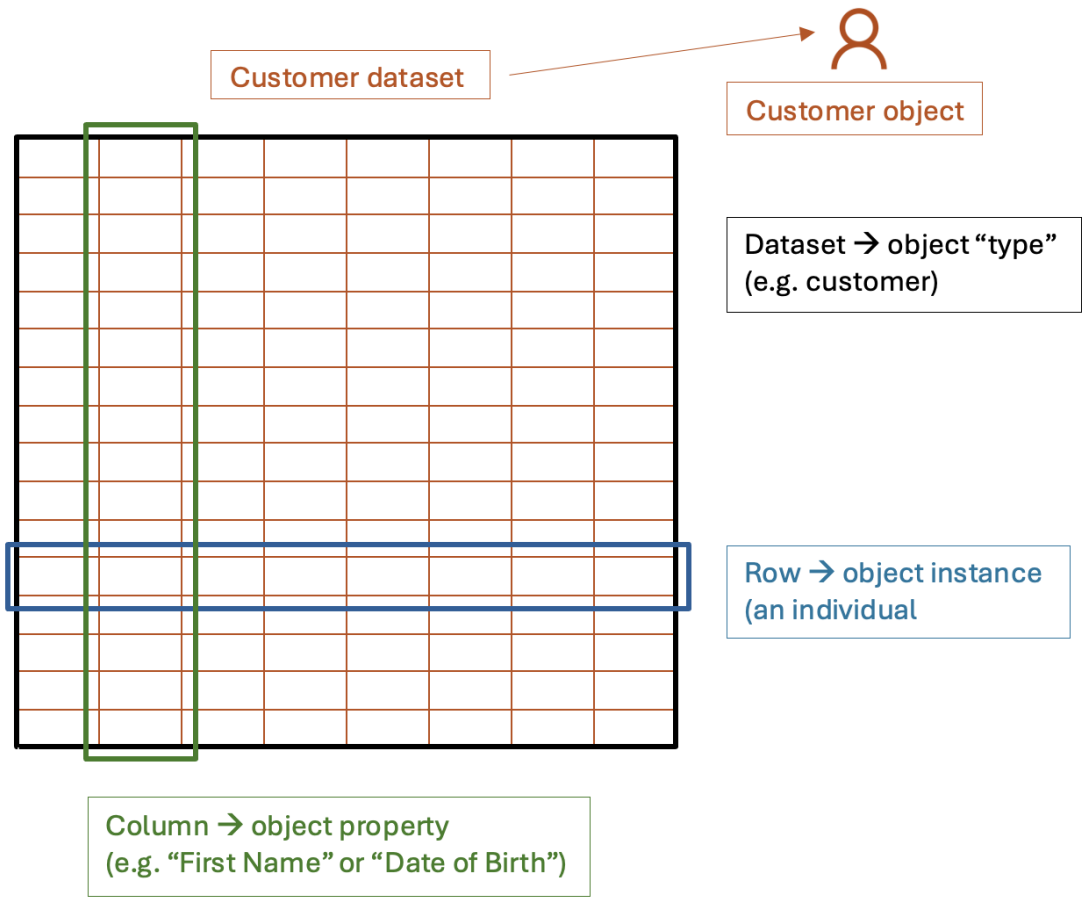
Concepts

Mapping a datasource to an object type

When you define a new **object type**, you are defining a kind of blueprint or template. This object type is backed by one or more datasources. This **datasource** is typically a dataset, but it could also be restricted view, virtual table, etc.

Each row of data in the backing datasource will produce one object that is a specific instance of that object type. Each column typically maps to a property. The permissions a user has to the backing datasource(s) also determines whether they can see the corresponding object data.

Here is example using Customers:



You can explore this row-to-object mapping in Ontology Manager. If you look at the bottom of your screen, you'll find a preview of the backing datasource. At the top of that panel, click **Preview objects** to preview the objects that correspond to each row.

By using actions, it is possible for users to create additional objects that do not exist in the backing datasource. Users can also delete objects. Therefore, over time, the mapping from rows to objects may not always exist for all rows.

Columns verses Properties

By default, when you create a new object type, each column of the backing datasource is mapped to a property in the new object type. However, you can change this mapping. For example, you can rename a property or choose not to map some columns to properties.

It is also possible to add properties that are not backed by a column. If you create a new property, you can set its *Source* to *User edits*. These values can only be created or changed later through actions on a per-object basis.

If you look through the preview of the **flights_alerts** dataset, you'll find that the root_cause column is empty (contains only null values). When initially building this pipeline, there was no satisfactory way to determine the root_cause for each alert from the

other data available. Instead, we plan to collect this information from Flight Operations users as they investigate and close each alert. In this case, the Root Cause would have been a good candidate for an edit-only property without a backing column. However, creating an empty column as part of the data pipeline also works. This will make it easier to insert a data-derived root cause value later if we should ever integrate another data source that would let us update the data pipeline to determine at least some instances of this automatically.

The takeaway here is that, while datasource columns inform which object properties you have, they do not always map 1-to-1 because object data is separate from datasource data. (We'll explore this in a little more detail in Object Storage below.)

Primary Keys and Titles

Every object must have a unique identifier that is consistent for the life of that object. This is its **Primary key** property. When selecting a primary key, you must choose a column that will contain a unique value for each row and that is never null. Strings are typically the best data type to use for this.

If your backing datasource doesn't contain a column that will serve as a valid identifier, you might derive a new column that concatenates multiple columns (and optionally hashes the result) to generate a valid key. This must be a deterministic repeatable process. You should never generate a random ID or GUID for this purpose as part of a data pipeline, because, if the pipeline is ever rerun, all of your object IDs will change.

Every object should also have a **Title**. This property is used as the user-friendly and human-readable name of the object in search results and in most Foundry applications. While the primary key can also be used for this purpose, unique keys are often fairly unreadable or opaque to everyday users. Again, you might derive a useful title from other columns as part of your backing data pipeline.

Actions

Actions provide a controlled way to create, edit, and/or delete objects. Ontology Manager offers to create a generic version of these when you create a new object type.

In this example Fresh Air use case, we plan to only create new Flight Alerts through the data pipeline, so we don't need a *Create* action for users to do so manually. While we will resolve alerts, we won't delete them, so we also don't need a *Delete* action. We will allow users to edit an alert, but only by touching certain fields. So we will create a more targeted action for that purpose in a later step.

Step 2: Save your changes

1. Click **Save** in the top right of the screen

2. Click **Save to ontology**
3. Click **Save changes**

If other users have made changes to the ontology while you were working, you may be prompted to update your view of the Ontology to pull in their changes before you can save your own.

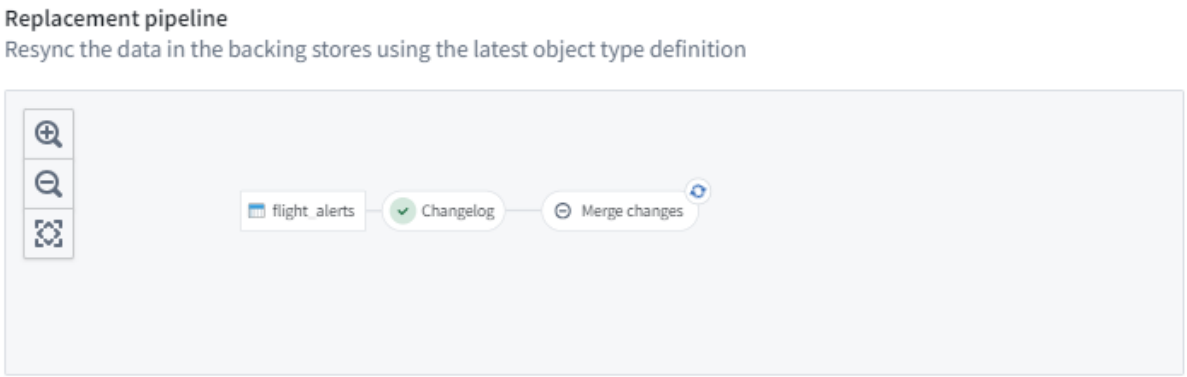
If you get any errors at this point, you can review the steps above to see if you made any mistakes in configuring the object type. If you get stuck, you can discard all of your edits and start over.

Concepts

Object Storage

Once you save, you'll see a little progress spinner appear in the upper left of the screen:

This indicates that Foundry is currently building an index of all of your object data for this object type in Object Storage v2. If you click this spinner, you can view the progress. This will take you the *Datasources* tab:



If Foundry encounters any issues—such as non-unique primary keys—you'll be able to get to detailed job details and error messages from here.

Only once this indexing pipeline completes successfully will you be able to see your new objects in Object Explorer and other Foundry applications. Once it's done, you can refresh the page to see an object count in top left of your screen: *183, 999 objects*.

You do not need to wait for this pipeline to finish running to move on to the next step of this tutorial.

Create a link type

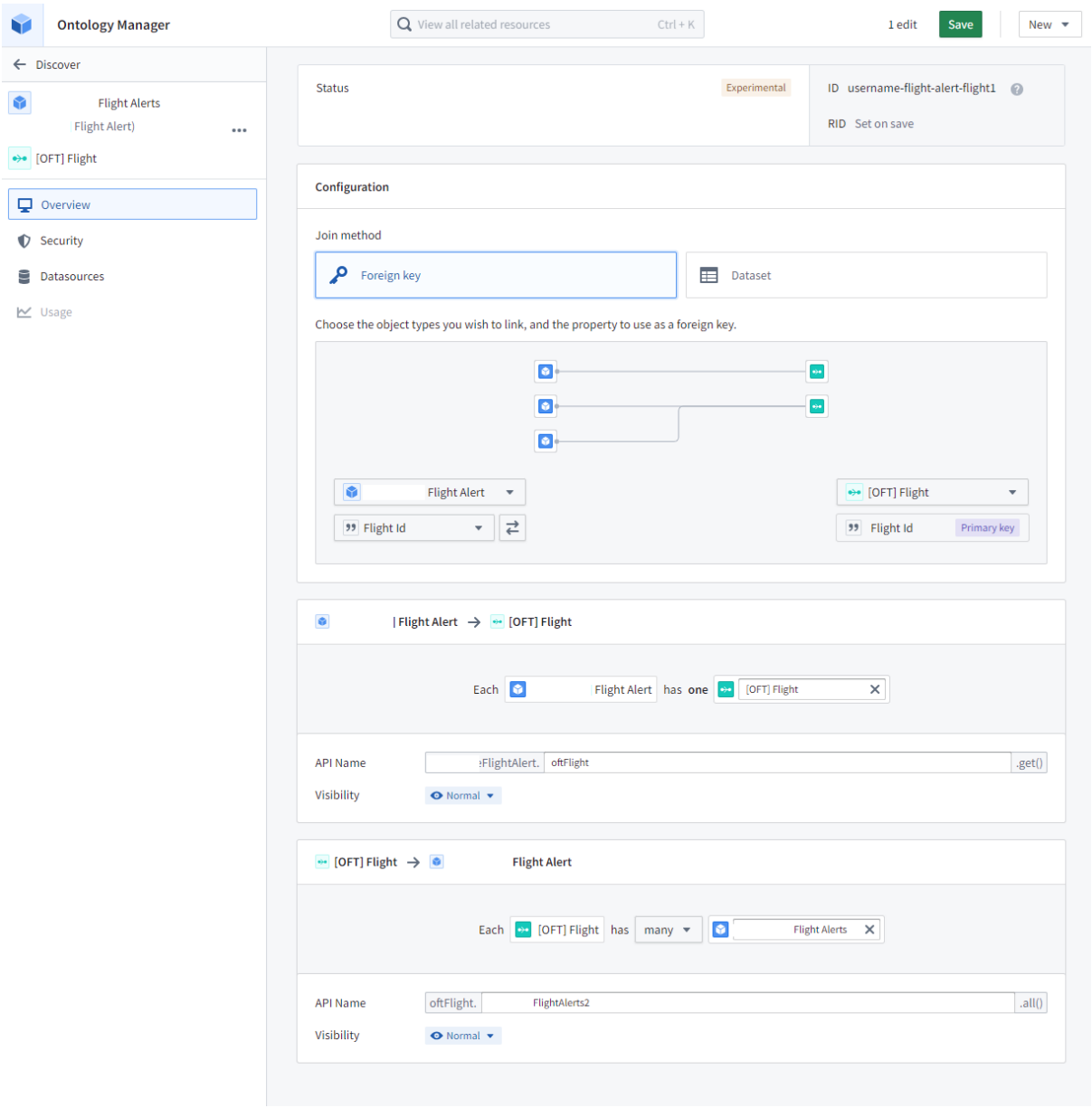
It would be useful to be able to quickly navigate from each Flight Alert to its corresponding Flight object. There is a lot of ancillary information available on the Flight object that we don't need to repeat on the Flight Alert object, but that might be useful to users who are responding to an alert.

Each Flight Alert includes a Flight Id property that contains the primary key of the corresponding Flight.

Step 1: Create Flight Alert <> Flight link

1. Click **New > Link type** in the top right corner of the screen
2. Create a foreign key relationship type
    1. Select **Foreign key**
    2. Click **Next**
3. Choose the two object types
    1. On the left: **[<username>] Flight Alert** (where <username> is your username)
    2. On the right: **[OFT] Flight**
4. Set the **Foreign key** for Flight Alert: Flight Id
    1. Click **Next**
5. Set **Cardinality**: **One** Flight to **Many** Flight Alerts
6. Click **Submit**

You should then have a link type definition that looks something like this:



Concepts

Foreign key

Similar to the concept used in relational databases, a foreign key property in one object type contains values that appear as a primary key value in another object type.

Cardinality

This is another data modeling concept that is not specific to Foundry. The cardinality of a relationship indicates how many entities might be involved on the two sides of that relationship.

This will depend on your underlying data and how you are modeling the world. For example, if you are modelling the relationship between an Airline and its Aircraft as “present-day ownership”, it would be **one-to-many**: Each Airline might own many Aircraft, but each Aircraft is owned by only one Airline at a time.

On the other hand, if you are modeling ownership over time (where planes are sold from one airline to another) or operations (where partner airlines might use codeshares or an airline might lease aircraft to its subsidiary airlines), then the relationship would be **many-to-many**: An Airline has many Aircraft, while an Aircraft is operated by many Airlines. (The Airline-Aircraft link in the Fresh Air Ontology is many-to-many if you would like to see an example.)

With regards to the Flight Alert you just created, for this notional data pipeline, we generated only one Flight Alert per significantly delayed, cancelled, or diverted Flight. This would make this a practically **one-to-one** relationship. However, that is not a constraint we would want to impose here. Conceptually (and perhaps in a future version of this pipeline), we would expect this to be a **one-to-many** relationship: While each Flight Alert should only be related to one Flight, a Flight might generate multiple Alerts. For example a flight could depart late and then, several hours later, be redirected to a different airport. This would ideally produce two separate alerts.

Step 2: Save your changes

1. Click **Save** in the top right of the screen
2. Click **Save to ontology**
3. Click **Save changes**

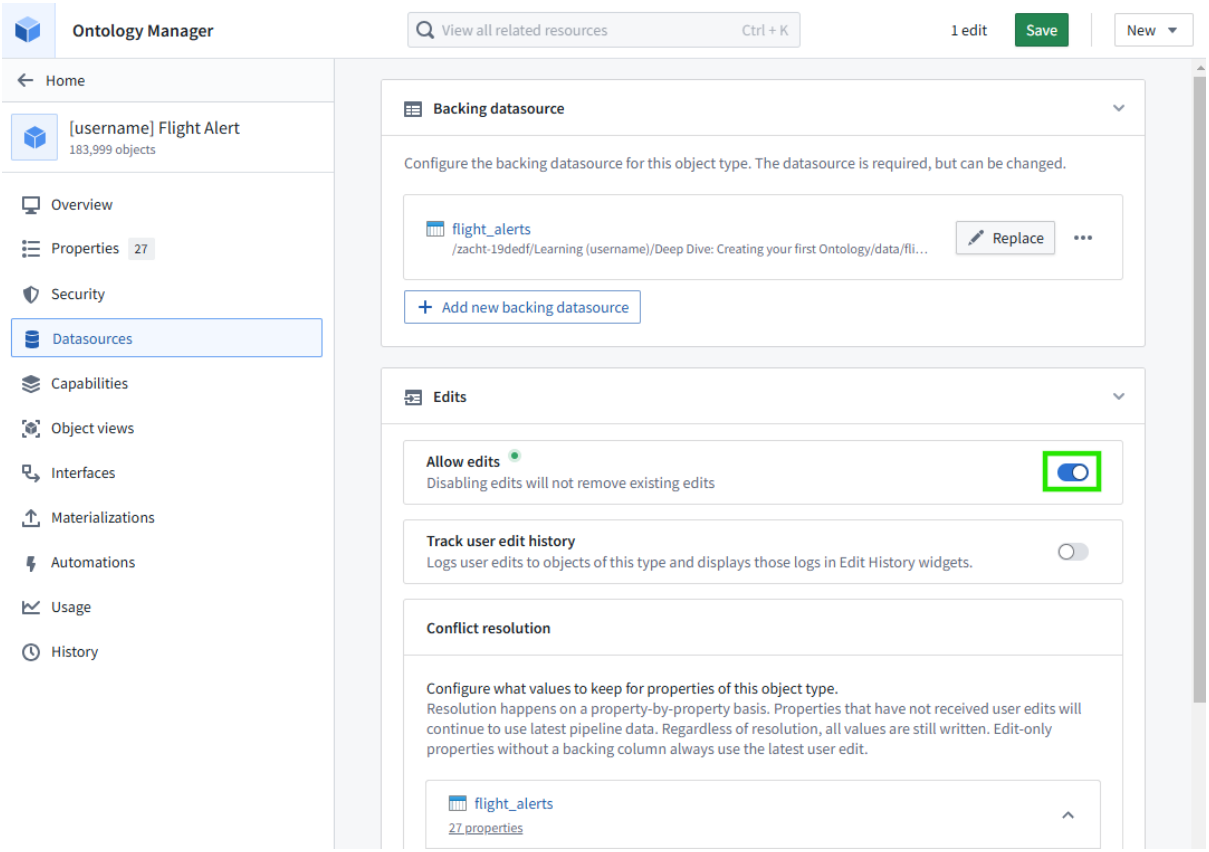
Create an action type

There’s a lot of different decisions the Flight Operations team might take in response to a Flight Alert. For example, they might need to reassign gate crews, contact the maintenance team, arrange for a plane swap, etc.

A full cross-team workflow like this often needs to be implemented incrementally in Foundry. We’ll assume we’re just getting started with only the Flight Operations team for now. Their immediate need is to understand more concretely what is causing their delays. While root-causing a delay is still largely a manual process for them, capturing the conclusion in Foundry is a good first step. Let’s add an action to do that.

Step 1: Enable edits

1. Open your **[username] Flight Alert** object type in Ontology Manager
2. Switch to the **Datasources** tab
3. In the **Edits** section, toggle on **Allow edits**

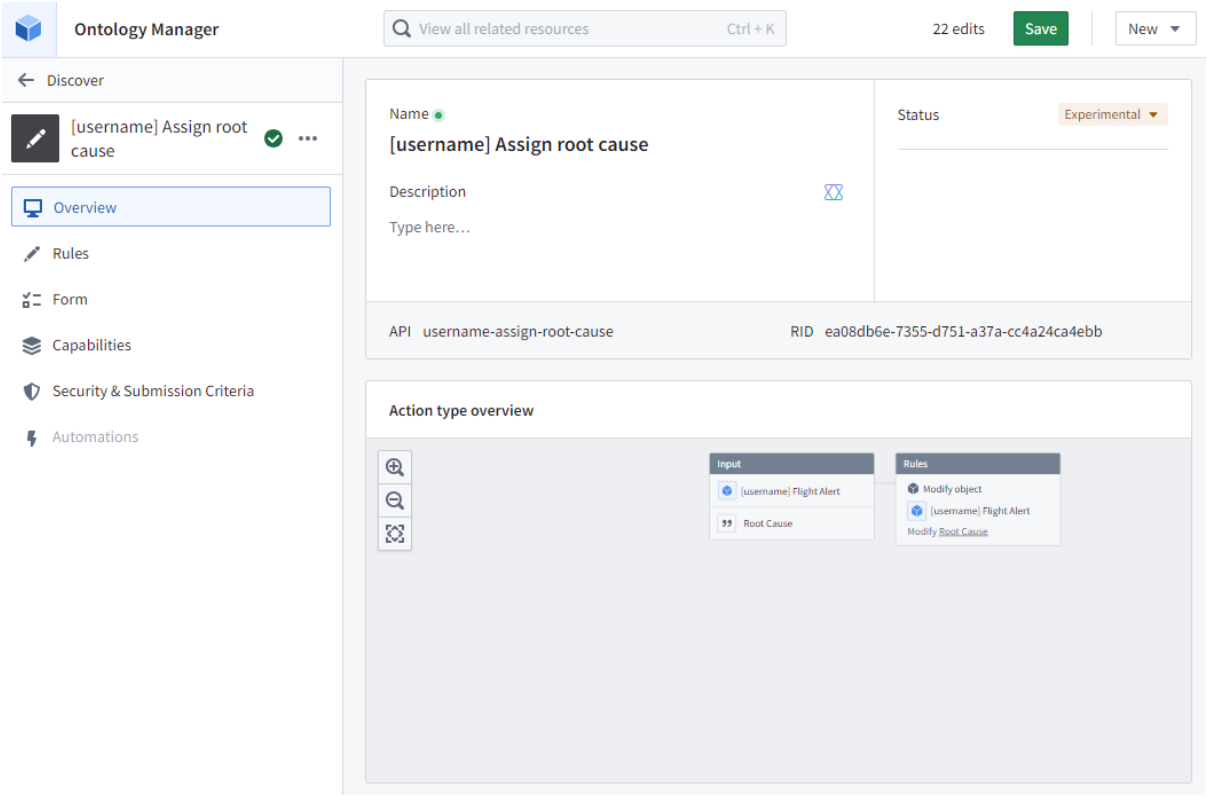


Step 2: Create a new action type

1. Click **New > Action type** in the top right of the screen
2. Create a Modify object action for your Flight Alert
    1. Select **Object** from among the available tabs (default value)
    2. Select **Object type**: Your **[username] Flight Alert**
    3. Select **Modify object(s)**
    4. Click **Next**
3. Add a Root Cause property
    1. Select **Add property > Root Cause**
    2. Click **Next**

4. Name your action type
    1. Set **Action type name**: [<username>] Assign root cause (where *<username>* is your username)
    2. Click **Next**
5. Give yourself permission to execute this action
    1. Switch to the **User** tab
    2. User is: (yourself)
6. Click **Create**

You should then see the definition of your new action:



Concepts

Actions provide a controlled way to update objects. Because we've only defined one action that touches only the Root Cause property, this is the only property that users can edit.

Action Form

Every time you invoke this action from different contexts in Foundry, a common input form will collect the necessary input from a user. If you switch to the **Parameters** tab of your action type, you can see a preview of this form.

There is a lot of further customization you could do here. For example, you can rename input fields, assign default values, hide fields, and more. For example, for now we are going to accept any free text as the new Root Cause value. If you wanted to, you could constrain this to a list of multiple choices that you either enter manually or derive from a dataset. (You can see the original *[OFT] Assign root cause* action type's Root Cause parameter for an example of this.)

This customization is another example of how actions can control changes to data: You might limit choices to only certain allowed input values.

Submission Criteria

Another control is to specify who can invoke the action at all. You set this during action type creation. You can update it on the **Security & Submission Criteria** tab.

Rules, Side-effects, & Functions

When you execute an action, it runs one or more rules, which you can see on the **Rules** tab. For now, we are applying a very simple *Modify object* rule that sets one property value. However, you could add other rules that send a Foundry notification to another Foundry user, invoke a webhook to call out to an external system, or create and delete other objects.

Instead of these rules, you can instead back your action with a function. Such code-based functions are also used elsewhere in Foundry beyond just backing actions.

Step 3: Save your changes

1. Click **Save** in the top right of the screen
2. Click **Save to ontology**
3. Click **Save changes**

Execute your action

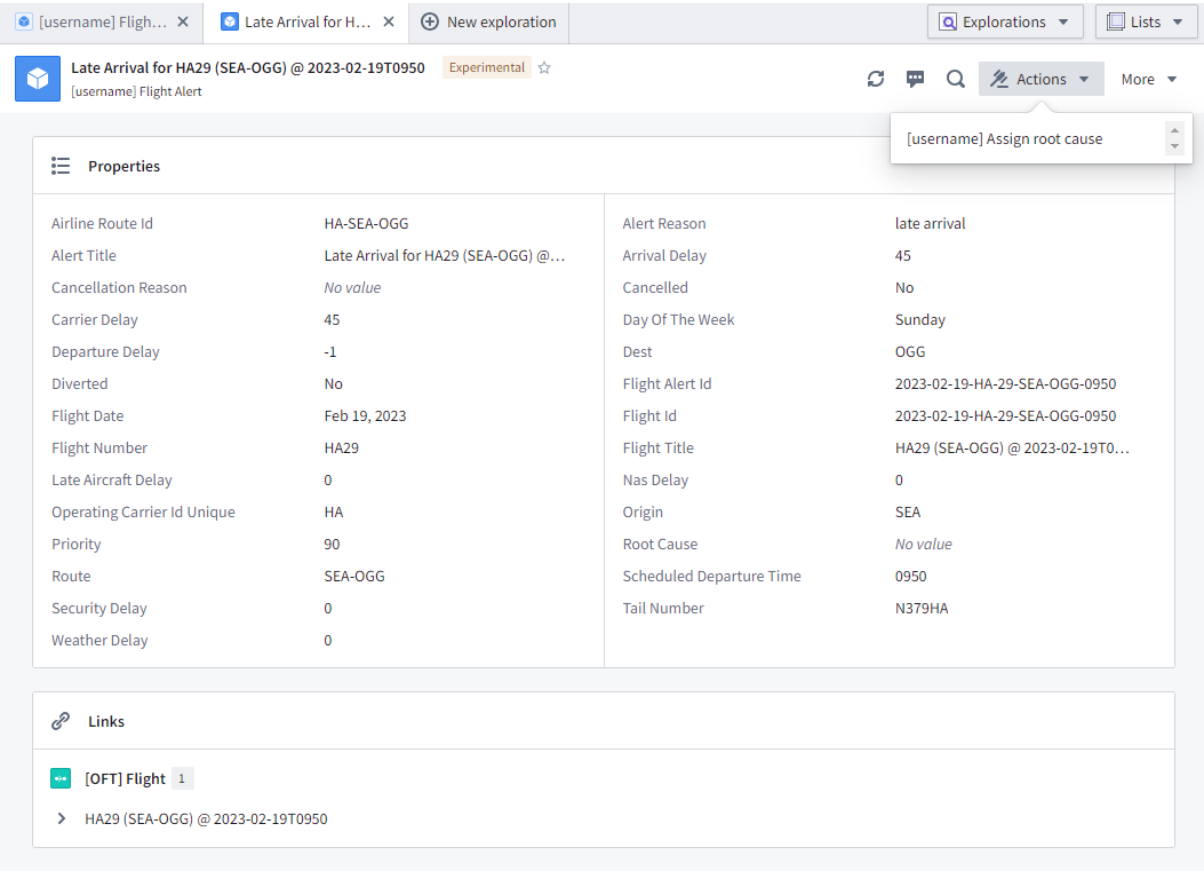
Let’s confirm that your new action works.

Step 1: Assign a root cause to a flight alert

1. Open **Object Explorer**
2. Search for your **[<username>] Flight Alert** object type
    - This should open an exploration of all your Flight Alerts
3. Click on one Flight Alert object in the **Results** list on the right

- This will open a single Flight Alert object. You should see its many properties and a link to the corresponding Flight object.

4. In the top right, select **Actions > [username] Assign root cause**
    - If you don't see the Actions menu, make sure you've saved your ontology changes and it has finished indexing those changes. If you already had this Object Explorer session open before making those changes, refresh the page to ensure it is in sync with the latest Ontology.



5. Enter an example root cause
    1. Such as: Waiting for gate assignment upon arrival
    2. Click **Submit**

You should see your change reflected on in the Root Cause property of the object view.

Reflections

The action you just defined here is very basic, recording only a single Root Cause value. However, as you saw, this action has a semantic meaning, and it provides strong controls on who can make what kinds of changes to Ontology data. It also captures a single human conclusion back into Foundry.

If we were to continue build out this workflow for the Flight Operations teams, you can imagine other decisions and the resulting actions they might want to make:

Assign an incoming Flight Alert to a given Flight Operations operator

Forward the alert to the local Flight Maintenance team for processing

Reassign the flight crew

Queue for passenger rebooking

Update the gate assignment

Reschedule this aircraft's next flight

Swap aircraft for this aircraft's next scheduled flight

Make as resolved

As you focus on these decisions that the Flight Operations team needs to make, it shines a light on what additional data they will need in the Ontology to inform those decisions:

Flight crew status and availability

Plane availability for swaps

Passenger counts and destinations to inform tradeoffs between plane swaps and rebooking

As this workflow grows in complexity, the Flight Operations team would quickly leave Object Explorer behind. Custom Workshop applications—as you built in the earlier End-to-End Speedrun course—would provide more appropriate and targeted tools. But those Workshop applications would reuse everything you built here: data pipeline, object type, links, and actions.

This is illustrates another value of the Ontology. By focusing on real-world concepts, tailored to a specific organization and reflecting its processes, it lets us build solutions around daily decisions that need to be made. This produces a dynamic "digital twin" of the organization.

Conclusion

This course has covered the basics you need to know to get started with a Foundry Ontology.

Value

We covered why the Ontology is valuable.

1. It makes data easier to use.
2. It structures data into a vocabulary that is reusable across workflows and across the organization.
3. The Ontology provides well-defined actions that can enforce constraints and make the output of regular decisions immediately to others within the platform.

In this way, the Ontology is more than another “semantic” data layer. In also includes “logic” through the rules that it encodes and “kinetics” through its actions. It is both the nouns of your organization, but also the verbs.

Ontology and AI

One last reason that the Ontology is valuable is the way that it allows you to integrate AI into daily operations.

First of all, the Ontology can provide clean and usable inputs to AI. For example, if we wanted AI-generated recommendations for how to respond to a given Flight Alert, it would need to review much of the same data that human-operators also need to power their decisions. Furthermore, the Ontology structures these concepts in real-world terms that an LLM are already familiar with.

Secondly, the Ontology provides a destination for AI outputs. The best place for a Flight Alert recommendation or prediction would be on the Flight Alert object itself. This inserts the recommendation into one or more existing operational workflows at the point where operators are already making their decisions. This makes it easier to then collect feedback on the final decisions those human operators then make—when they use the recommendations and when and why they disregard them.

The many tools of AIP aim to do exactly this by building on top of the Ontology.

Tools

In terms of specific tools, you learned about:

- Object Explorer - For object searches and for basic exploration and analysis across sets of objects
- Data Lineage - For exploring data pipelines and understanding how they lead to the Ontology
- Ontology Manager - For building the Ontology itself