SDL Examples

Version latest, 2026-05-01

Examples

Welcome to the SDL Examples section. This documentation provides hands-on demonstrations and use cases that showcase the capabilities of SDL (SOF Data Layer).

Purpose of This Section

The SDL Examples serve multiple functions:

- **Demonstrations**: Ready-to-use scripts for showcasing SDL capabilities to stakeholders
- **Training Materials**: Hands-on examples for users learning the platform
- **Validation**: Test cases to verify successful platform deployment and configuration
- **Templates**: Starting points for developing custom workflows and applications

Available Examples

Video Demonstrations

Video Demos

Video walkthroughs of platform capabilities including mesh synchronization, federated query, streaming pipelines, cross-domain transfer, DDIL operations, and more.

Federated Query

Federated Query Demo

Demonstrates "zero ETL" querying across heterogeneous data sources using federated SQL, SPARQL ontology queries, and natural language interfaces — all without moving or duplicating data.

Data Pipeline

Simple Pipeline

Basic data pipeline example demonstrating ingestion, transformation, and storage workflows using the platform's streaming and processing capabilities.

Lattice Integration

Lattice

Integration example for connecting with Lattice systems using the platform's connector ecosystem.

How to Use These Examples

For Demonstrations

Each example is structured as a professional demo script with: * Clear prerequisites and setup requirements * Step-by-step walkthrough instructions * Expected outcomes and talking points * Visual cues and interaction guidelines

For Training

The examples provide: * Comprehensive background on capabilities * Hands-on exercises with realistic data * Progressive complexity from basic to advanced features * Troubleshooting and best practices

For Validation

Use these examples to: * Verify platform deployment success * Test integration with organizational systems * Validate performance and functionality * Confirm security and access controls

Getting Started

For End Users

If you're looking to **use** SDL capabilities, start with the [R]DP User Guide for operational guidance, then return here for hands-on examples.

For Developers

If you need to **install or configure** SDL, begin with the [R]DP Developer Guide for deployment instructions, then use these examples for validation.

Video Demos

This section contains video demonstrations of SDL capabilities in action. Use these videos to quickly understand platform features and operational workflows.

Platform Overview

End-to-end walkthrough of the platform interface and core features.

Video coming soon.

Federated Query

Querying across heterogeneous sources without data movement using SPARQL and natural language.

Video coming soon.

Streaming Pipeline

Live data flowing through transformation pipelines with bidirectional format conversion.

Video coming soon.

Policy Engine

Data-policy-as-code with row/column-level obfuscation and classification enforcement.

Video coming soon.

Cross-Domain Transfer

Data flowing across classification boundaries through XSD-validated guard systems.

Video coming soon.

DDIL Operations

Platform behavior during bandwidth degradation, disconnection, and automatic reconnection sync.

Video coming soon.

Edge Deployment

Platform running on ruggedized hardware with identical interface to cloud deployments.

Video coming soon.

Federated Query Demo

This guide walks through a step-by-step demonstration of SDL federated query capabilities. It showcases "zero ETL" — querying data where it lives without movement or duplication.

Prerequisites

- SDL deployment with federated SQL and VKG engine enabled
- Sample data loaded in at least 2 different data sources (for example, a relational database and a streaming topic)
- SPARQL endpoint access
- `curl` or SDK client installed

Demo Overview

This demo illustrates the following capabilities:

- **Federated SQL** — query across multiple heterogeneous data sources with a single SQL statement.
- **Cross-source joins** — join data from different source types without pre-staging or ETL.
- **Virtual Knowledge Graph (VKG)** — query an ontology-mapped knowledge graph using SPARQL.
- **Natural language query** — ask questions in plain English and receive structured results.
- **Asynchronous analytics** — submit long-running queries and retrieve results when ready.

Step 1: Identify Data Sources

List the data sources registered with the federated query engine.

```
curl -s https://sdl.example.com/api/v1/query/sources | jq .
```

Example response:

```
{
  "sources": [
    {
      "name": "operations_db",
      "type": "relational",
      "description": "Operational relational database"
    },
    {
      "name": "sensor_stream",
      "type": "streaming",
```

```
      "description": "Real-time sensor data topic"
    },
    {
      "name": "reports_store",
      "type": "object-storage",
      "description": "Archived reports in object storage"
    }
  ]
}
```

Step 2: Simple Federated SQL Query

Run a SQL query that targets a single data source through the federated engine.

```
curl -X POST https://sdl.example.com/api/v1/query/sql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "SELECT sensor_id, temperature, location, timestamp FROM
operations_db.sensor_readings WHERE temperature > 80.0 ORDER BY timestamp DESC LIMIT
10"
  }'
```

The federated engine routes the query to the appropriate data source and returns results in a unified format.

Step 3: Cross-Source Join

Join data from a relational database and a streaming topic in a single query.

```
curl -X POST https://sdl.example.com/api/v1/query/sql \
  -H "Content-Type: application/json" \
  -d '{
    "query": "SELECT r.sensor_id, r.location, s.current_temp, s.event_time FROM
operations_db.sensor_registry r JOIN sensor_stream.readings s ON r.sensor_id =
s.sensor_id WHERE s.current_temp > r.threshold ORDER BY s.event_time DESC LIMIT 20"
  }'
```

This query joins static registration data from the relational database with live readings from the streaming topic — without moving data between systems.

Step 4: SPARQL Ontology Query

Query the virtual knowledge graph using SPARQL.

```
curl -X POST https://sdl.example.com/api/v1/query/sparql \
```

```
  -H "Content-Type: application/sparql-query" \
  -d 'PREFIX sdl: <http://sdl.example.com/ontology#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?entity ?type ?location
WHERE {
  ?entity a rdp:SensorReading ;
          rdp:locatedIn ?location ;
          rdp:readingType ?type .
  FILTER (?type = rdp:Temperature)
}
LIMIT 25'
```

The VKG engine maps the SPARQL query to the underlying data sources through the ontology, returning results without requiring the user to know where the data physically resides.

Step 5: Natural Language Query

Submit a question in plain English and let the platform translate it into the appropriate query.

```
curl -X POST https://sdl.example.com/api/v1/query/natural-language \
  -H "Content-Type: application/json" \
  -d '{
    "question": "Which sensors in building 7 have reported temperatures above 80
degrees in the last hour?"
  }'
```

Example response:

```
{
  "answer": "3 sensors in building 7 reported temperatures above 80\u00b0F in the last
hour.",
  "generated_query": "SELECT sensor_id, temperature, timestamp FROM
operations_db.sensor_readings WHERE location = 'building-7' AND temperature > 80.0 AND
timestamp >= NOW() - INTERVAL '1 hour'",
  "results": [
    { "sensor_id": "sensor-alpha-01", "temperature": 82.1, "timestamp": "2026-02-
26T14:32:00Z" },
    { "sensor_id": "sensor-alpha-04", "temperature": 85.7, "timestamp": "2026-02-
26T14:28:00Z" },
    { "sensor_id": "sensor-alpha-07", "temperature": 80.3, "timestamp": "2026-02-
26T14:15:00Z" }
  ]
}
```

Step 6: Asynchronous Analytics

Submit a long-running analytical query and retrieve results asynchronously.

```
# Submit the query
curl -X POST https://sdl.example.com/api/v1/query/async \
  -H "Content-Type: application/json" \
  -d '{
    "query": "SELECT location, AVG(temperature) as avg_temp, MAX(temperature) as
max_temp, COUNT(*) as reading_count FROM operations_db.sensor_readings GROUP BY
location ORDER BY avg_temp DESC"
  }'
```

Example response:

```
{
  "query_id": "q-abc123-def456",
  "status": "RUNNING",
  "submitted_at": "2026-02-26T14:35:00Z"
}
```

Poll for completion and retrieve results:

```
# Check query status
curl -s https://sdl.example.com/api/v1/query/async/q-abc123-def456/status | jq .

# Retrieve results once status is COMPLETED
curl -s https://sdl.example.com/api/v1/query/async/q-abc123-def456/results | jq .
```

Expected Results

Step	Expected Behavior
Step 1: Identify Data Sources	All registered data sources are listed with their names, types, and descriptions.
Step 2: Simple Federated SQL Query	Results are returned from the target data source in a unified JSON format.
Step 3: Cross-Source Join	Data from the relational database and streaming topic is joined and returned as a single result set.
Step 4: SPARQL Ontology Query	The VKG engine resolves the SPARQL query against the ontology and returns matching entities.
Step 5: Natural Language Query	The platform translates the English question into a structured query, executes it, and returns both the answer and the generated query.

Step	Expected Behavior
Step 6: Asynchronous Analytics	The query is accepted and assigned a `query_id`; results are available once the status transitions to `COMPLETED`.

Simple Pipeline

This example demonstrates the creation of a very simple pipeline to get familiar with SDL's data pipeline engine and drag-and-drop pipeline UI.

Overview

In this example, you'll learn how to:

- Create a synthetic stream of ADS-B data
- Use the Dynamic Python Transformer to apply a simple transformation to the data
- Inspect streaming datasets in real-time

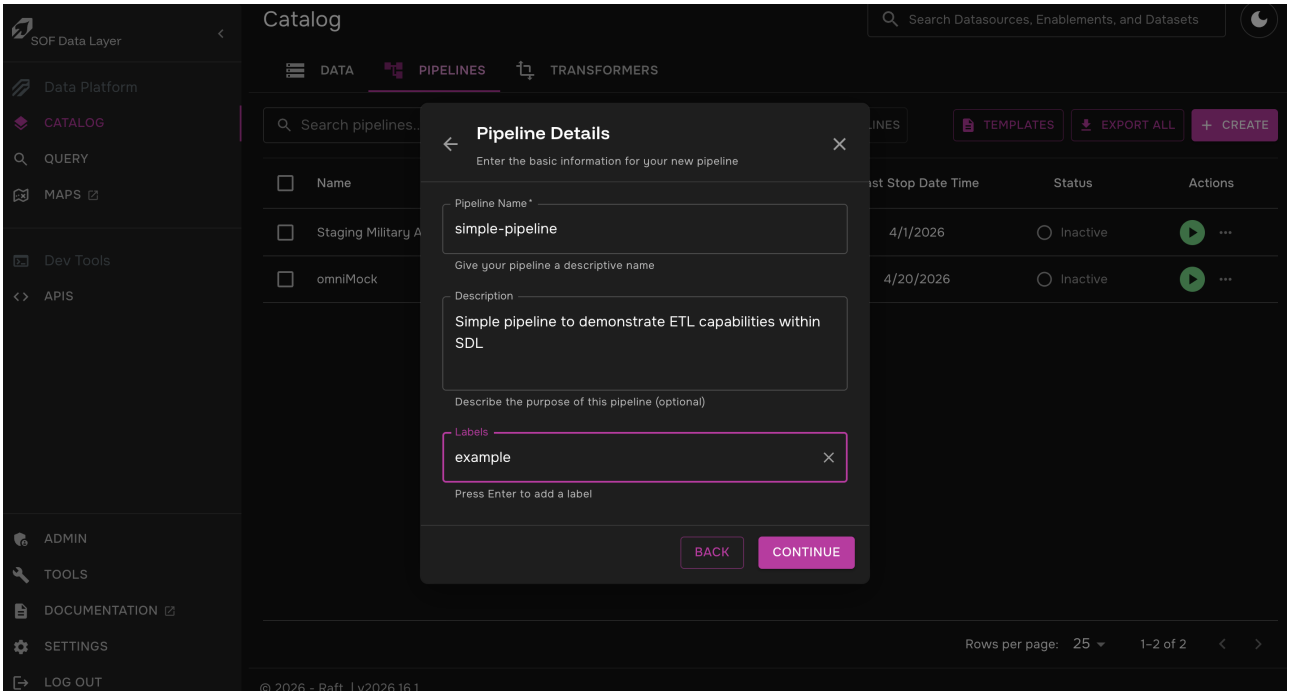
Prerequisites

Before starting this exercise, ensure you have:

- An active SDL account with appropriate permissions
- Access to the `ADS-B Generator` and `Dynamic Transformer (Python)` transformers (these should be included with your platform)

Step 1: Create a Pipeline

1. Click on **Catalog** > **Pipelines** > **Create**
2. Click "Start From Scratch"
3. Fill out the pipeline's name, description, and tags as you see fit and click "Continue"



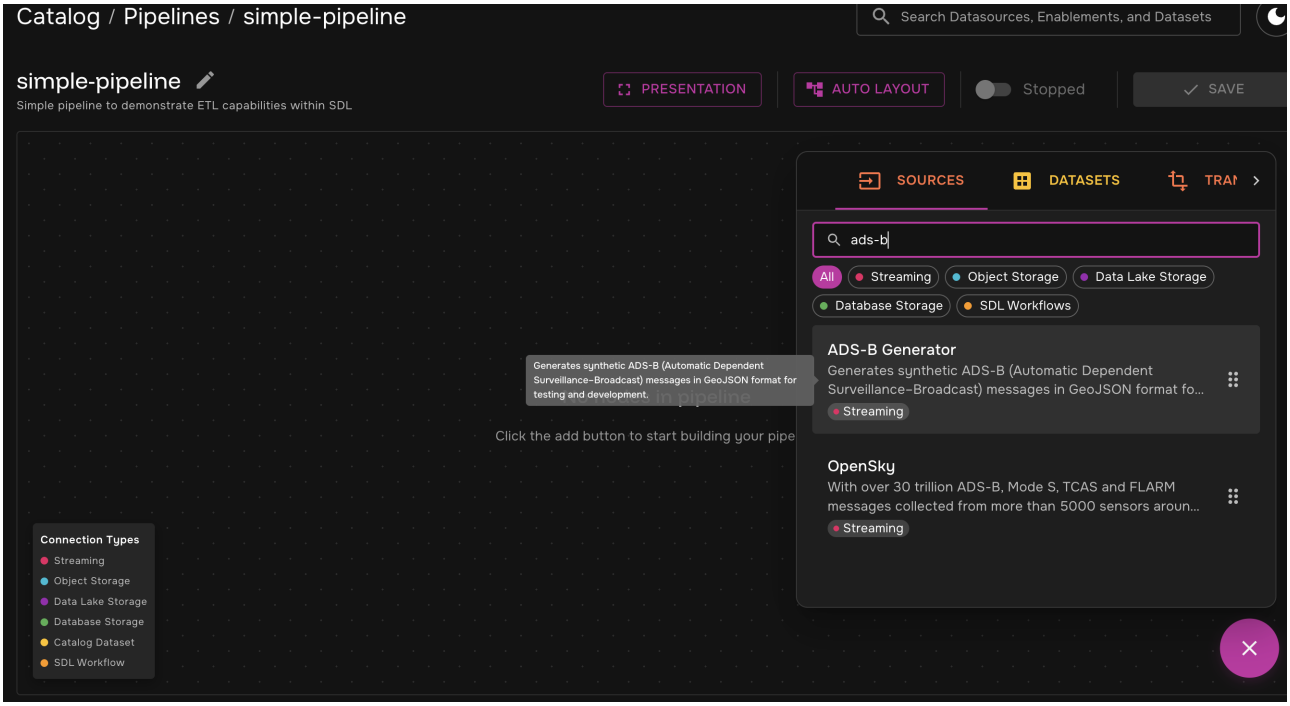
Step 2: Add Synthetic Data to the Pipeline

You will be taken to the main pipeline canvas. We will take a closer look at some of these features later, but for now let's build our pipeline.

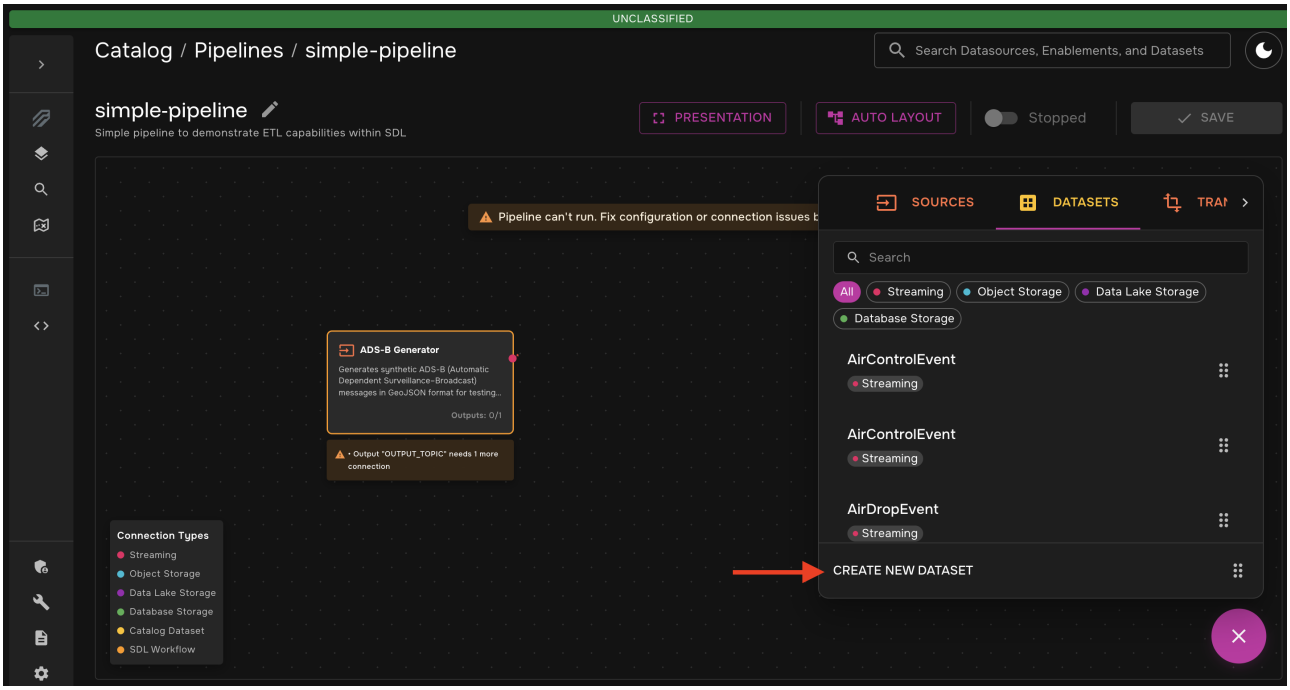
For this example, we will use the built-in `ADS-B Generator` to generate synthetic data.

SDL's data pipelines can extract data from a variety of sources, but for the sake of simplicity in setting up this example, we will have the source data be generated as part of the pipeline.

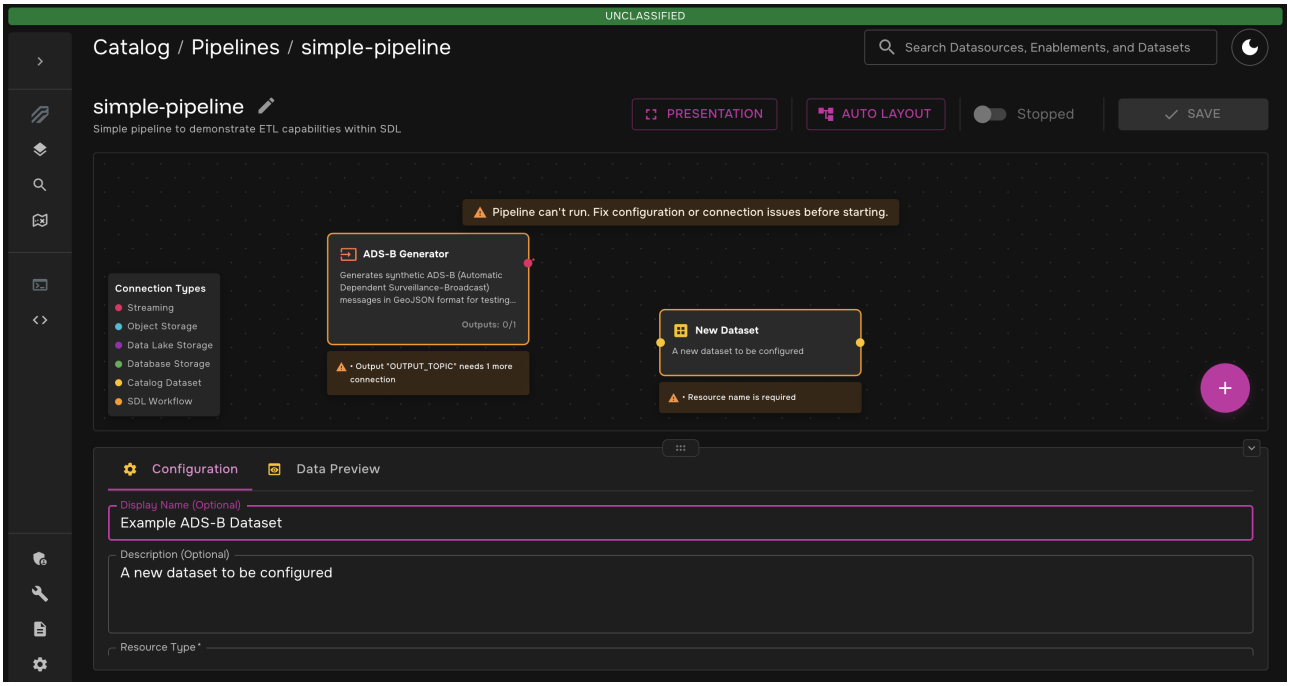
1. Click the "+" button in the bottom right corner and search for "ADS-B" in the "Sources" tab



2. Drag that transformer onto the canvas. Then, from the "Datasets" tab, drag the "Create New Dataset" box onto the canvas.



3. Click on the dataset. The configuration window should pop up. Under "Advanced Properties", select "Streaming" for the resource type and enter `example-adsb-synthetic` for the resource name. **When finished, be sure to scroll down and click "Done" in the bottom right corner.**

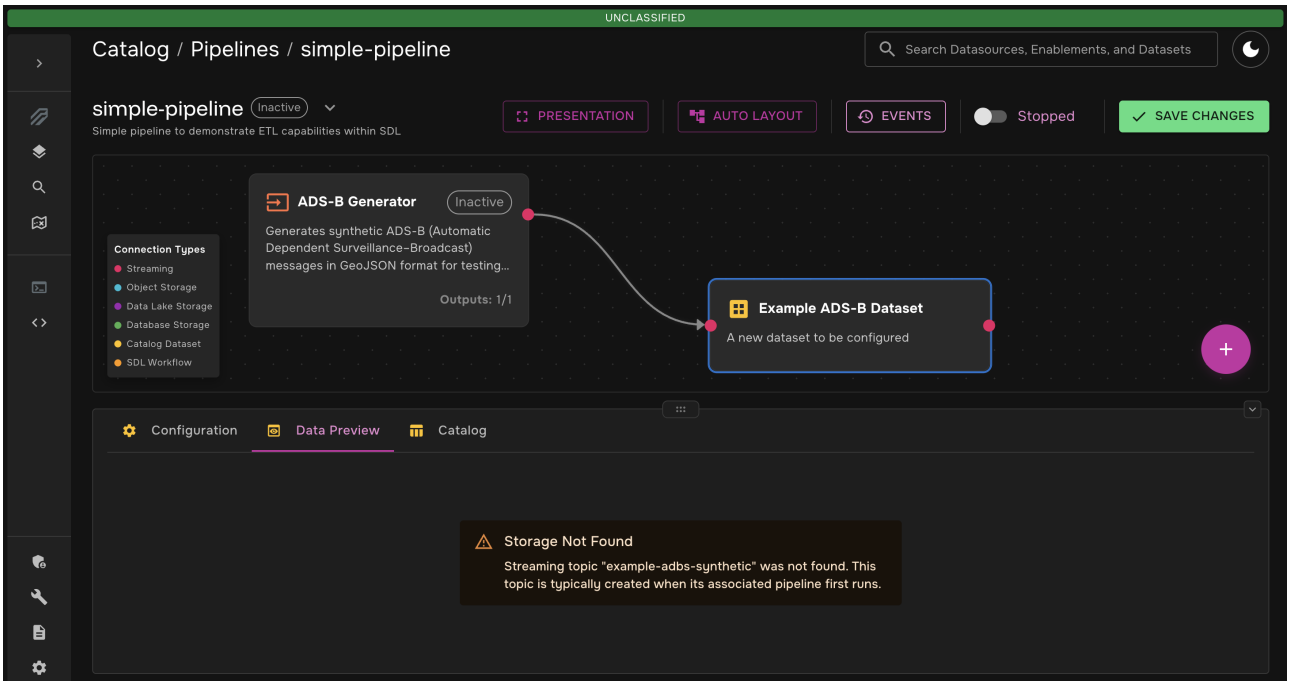


You will see that advanced Kafka Configuration options are available. You can leave those as the default values for this example, but they are a good tool to have for more complex use cases.

4. Connect the two nodes and hit "Save Changes". Click on the Dataset → Data Preview. You should have something that looks like this:

you may see a message before you hit "Save Changes" that says Storage Not Found. This is normal. The kafka topic will be created when the pipeline is run

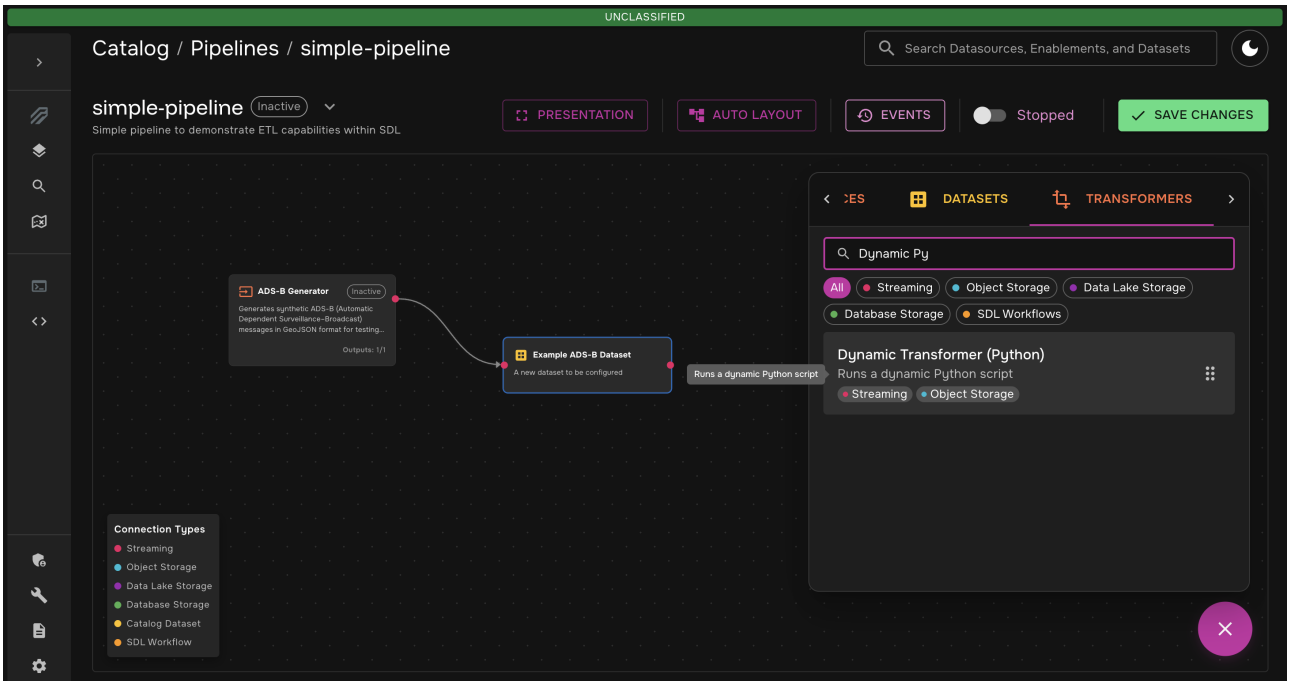
> for the first time.



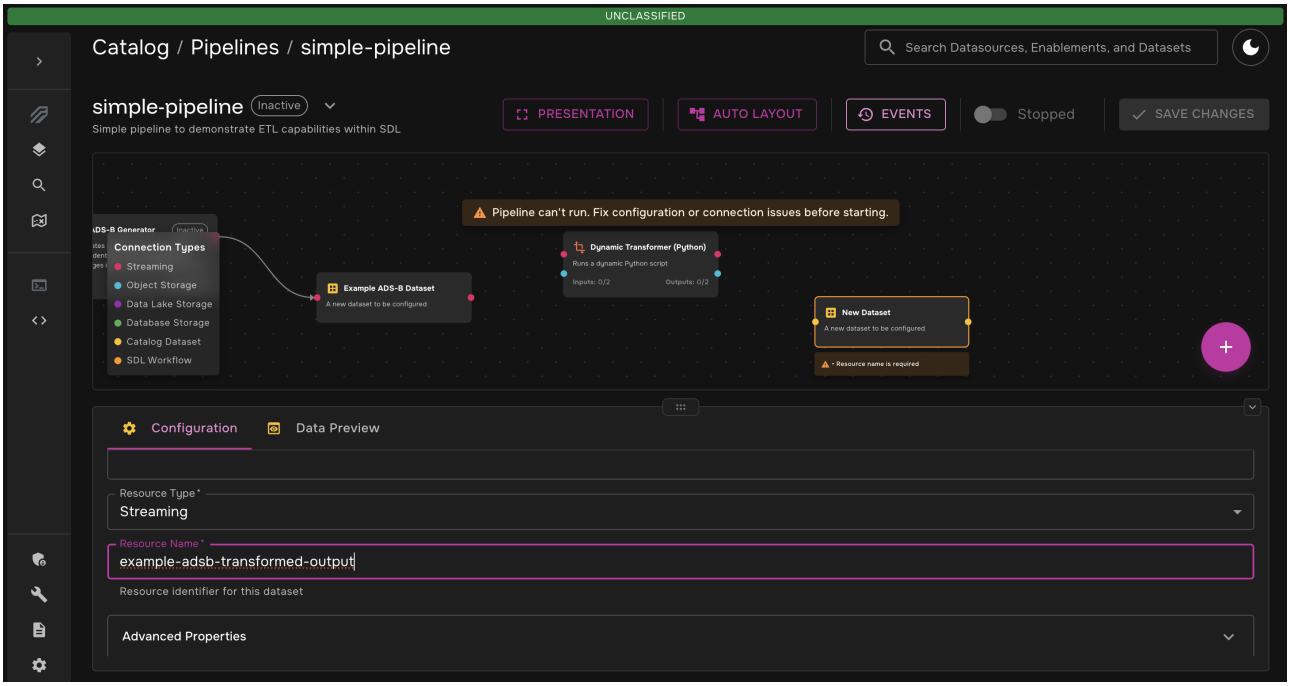
Step 3: Add a Transformer to the Pipeline

Now that we have data in our pipeline, it is time to perform an ETL (Extract, Transform, Load) operation on the data. As previously mentioned, we will use the Dynamic Python Transformer to extract the synthetic data, perform a simple transformation, and load that data into a new dataset.

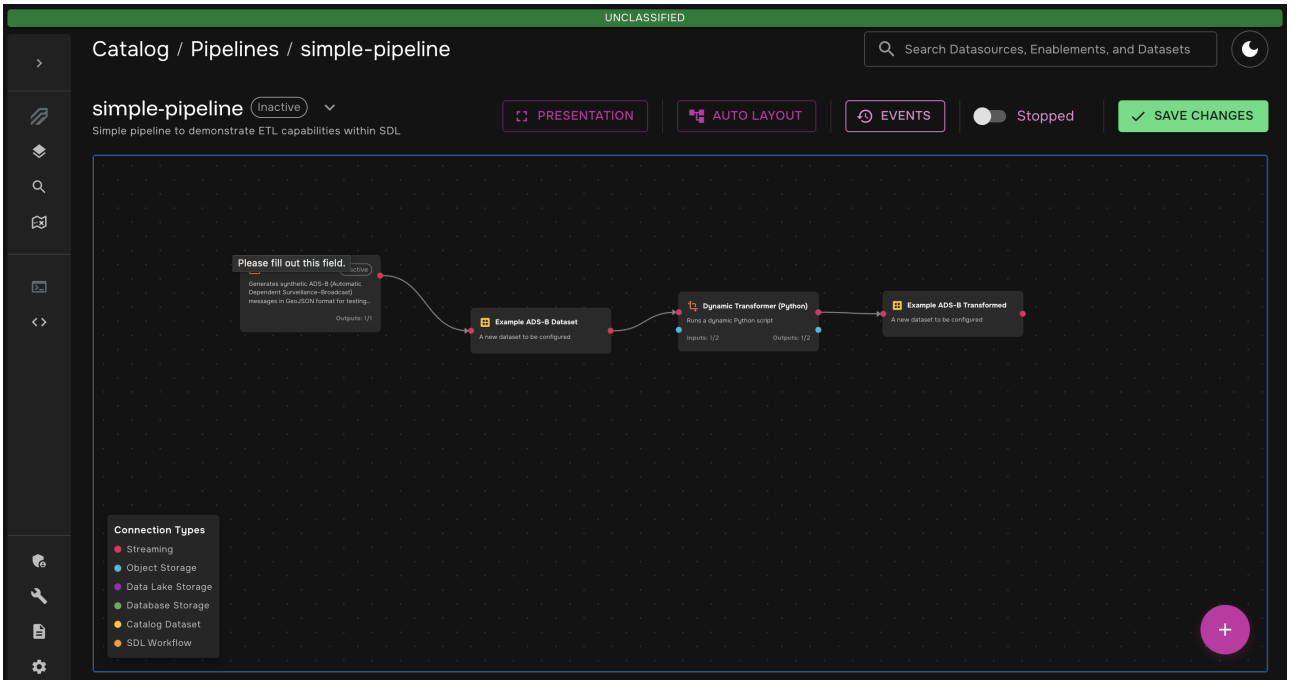
1. Add the Dynamic Python Transformer



2. Add another new dataset. This will be the dataset that holds the transformed data. Just like you did in Step 2, fill out and save the properties:



3. Connect the nodes. You will be left with something like this:



Step 4: Write the Transformation

You now have all the nodes you need for this pipeline! The last thing you need to do before running the pipeline is to apply the actual transformation in the Dynamic Python Transformer.

1. Click on the `Dynamic Transformer (Python)` node and navigate to the "Code Editor" tab. You will see that **by default, the transformer will pass each message through to the output dataset without modification.** We need to change the code to modify the data. Replace the default script with the following:

```
import os
import json
```

```
from df_daft_py.kafka.kafka import start_transform_loop


def transform(data) -> dict:
    # Get altitude from properties
    alt = data.get("properties", {}).get("altitude", 0)

    # Classify into flight level bands
    if alt < 10000:
        flight_level = "low"
    elif alt <= 35000:
        flight_level = "cruise"
    else:
        flight_level = "high"

    # Add the new field
    data["properties"]["flight_level"] = flight_level
    return data


def main():
    src_topic = os.getenv("SOURCE_KAFKA")
    dest_topic = os.getenv("DEST_KAFKA")
    start_transform_loop(src_topic, dest_topic, transform)
```

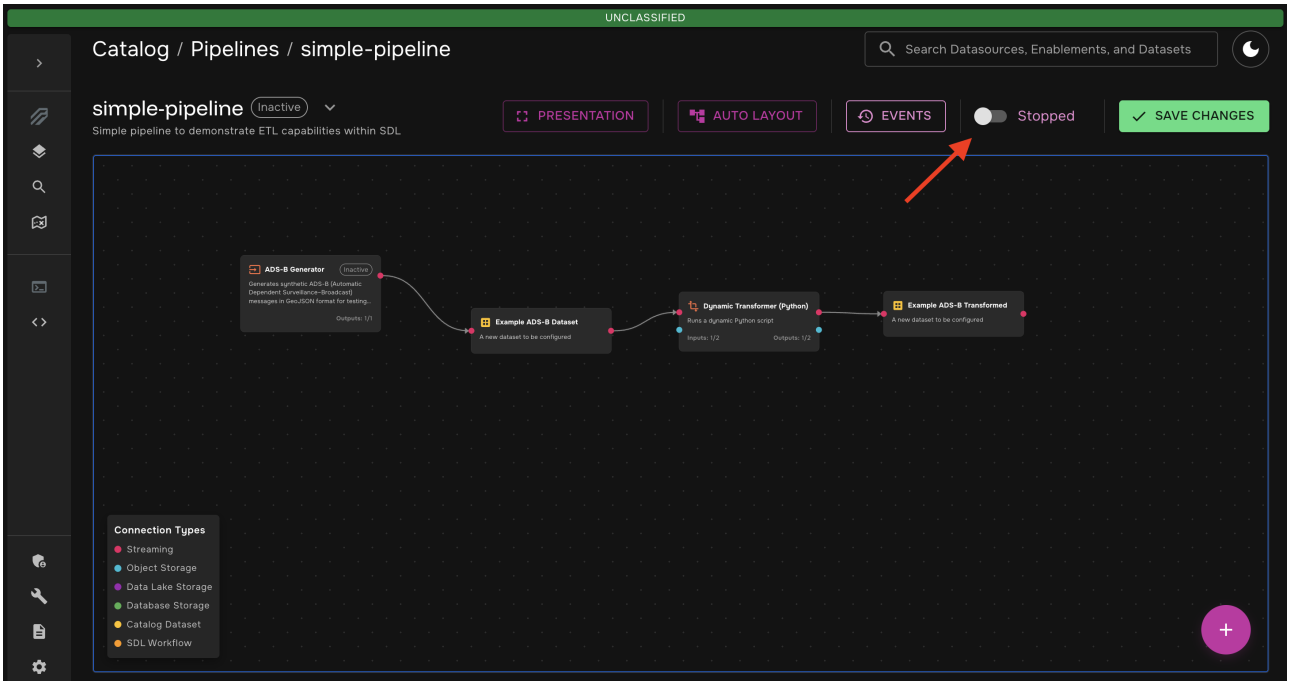
This transformation is simple - it checks the `altitude` field of the input message, determines the `flight_level` (`low`, `medium`, or `high`), and adds the `flight_level` to the data.

2. Click the "Save Changes" button in the top right corner

Step 5: Run the Pipeline and View Data

You are now ready to start the pipeline!

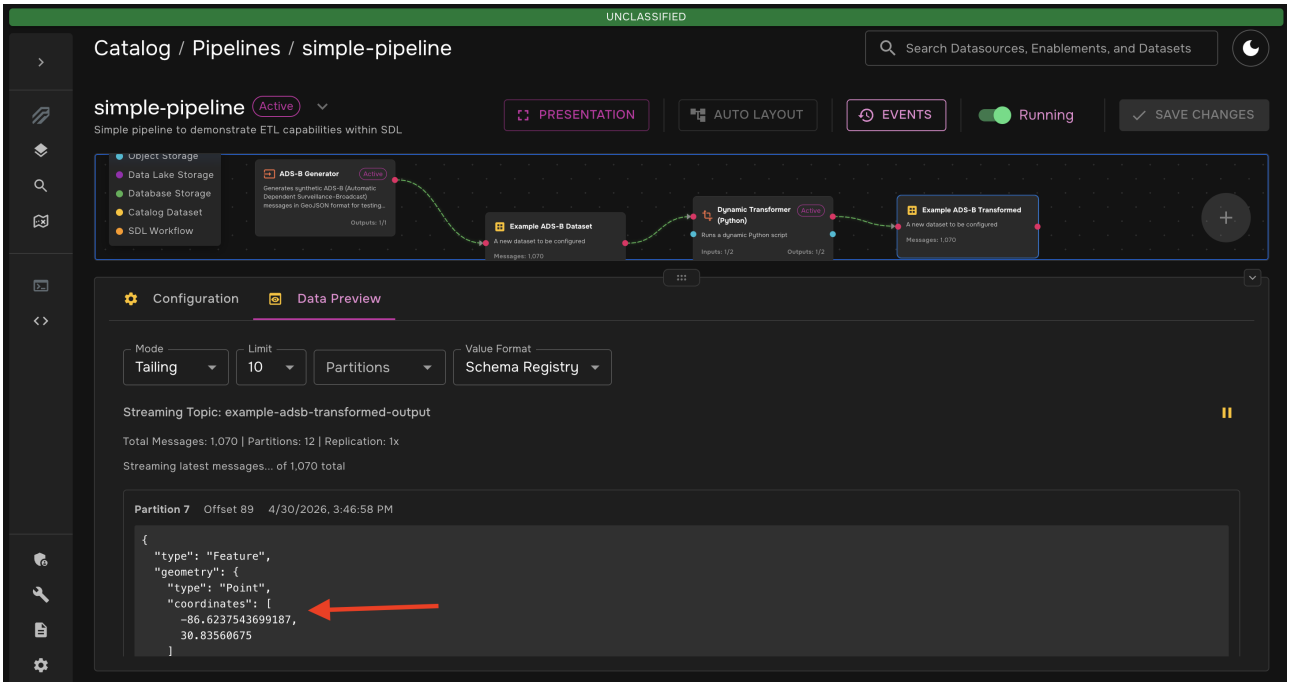
1. Click the start toggle in the top right corner



2. Wait for a few seconds while the pipeline spins up.

You can click on a transformer node and navigate to the "Logs" tab to view the logs for that stage in the pipeline. This is very useful for debugging.

3. You should see messages start to flow through the datasets. You can inspect the messages in each dataset right in the pipeline canvas. If you click on the "Python Transformed Output" dataset to view its messages, you will see that the transformation has been applied (it may be helpful to pause the data stream so that you can easily read the messages).



Key Takeaways

This example gave you hands-on experience in using SDL's data pipelines, including:

1. **Synthetic Data Generation**: Using one of SDL's built-in synthetic data generators to build a stream of mock ADS-B data.
2. **Pipeline Architecture**: Building a full ETL pipeline with multiple transformers and datasets.
3. **Dynamic Transformations**: Writing your own data transformation natively in the SDL pipeline canvas.
4. **Live Data Inspection**: Viewing the completed data transformation directly in SDL's web interface.

Lattice

This guide walks you through creating an SDL data pipeline that transforms Cursor-on-Target (CoT) XML messages and publishes them to Anduril Lattice.

Overview

In this example, you'll learn how to:

- Create a Lattice sandbox environment for testing
- Build a data pipeline in SDL
- Transform CoT XML messages into Lattice entity format
- Publish entities to an Anduril Lattice instance
- Visualize the results in Lattice's live view

Prerequisites

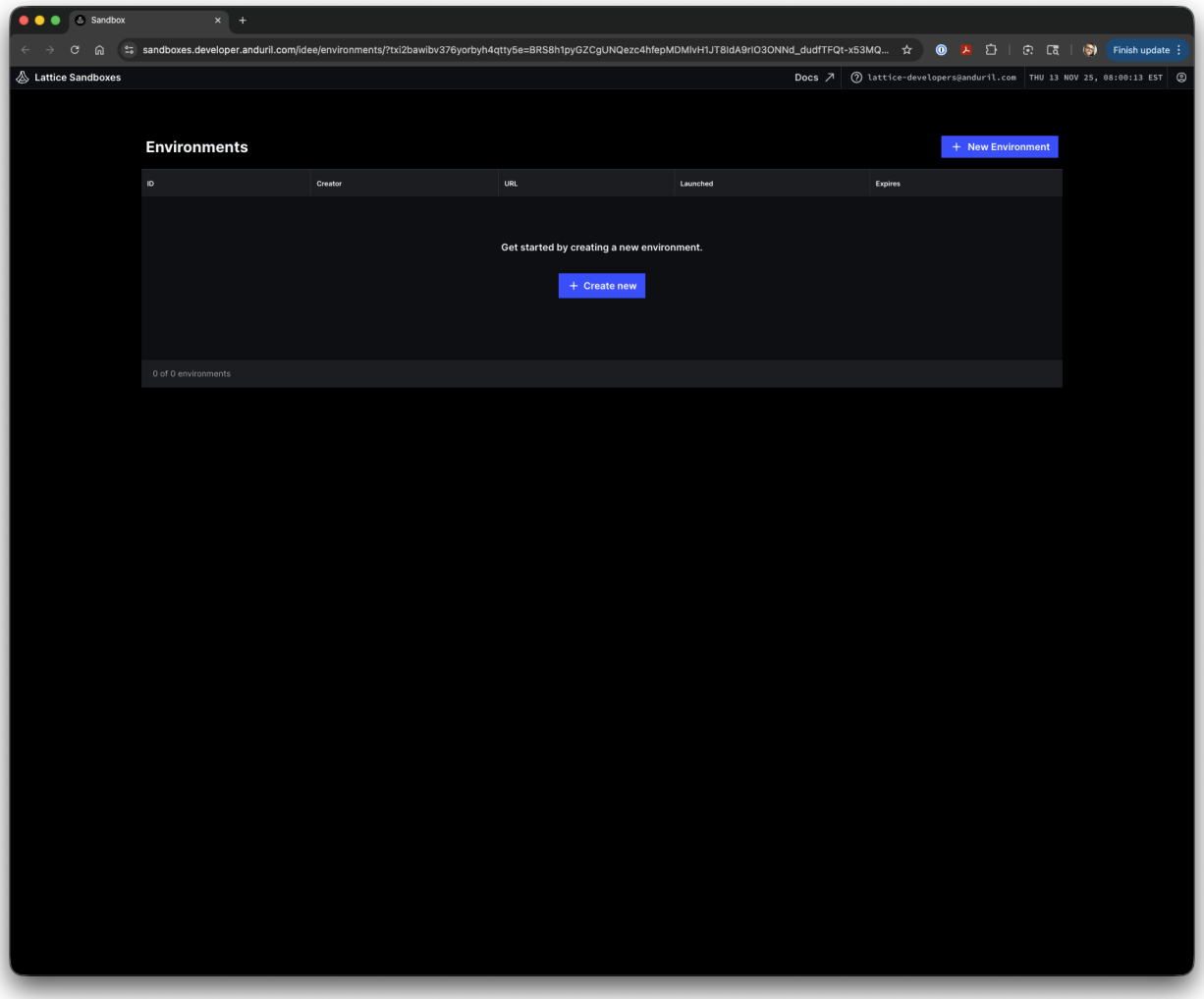
Before starting this guide, you should have:

- Access to SDL
- Access to Lattice Sandboxes (developer environment)
- Basic understanding of data pipelines

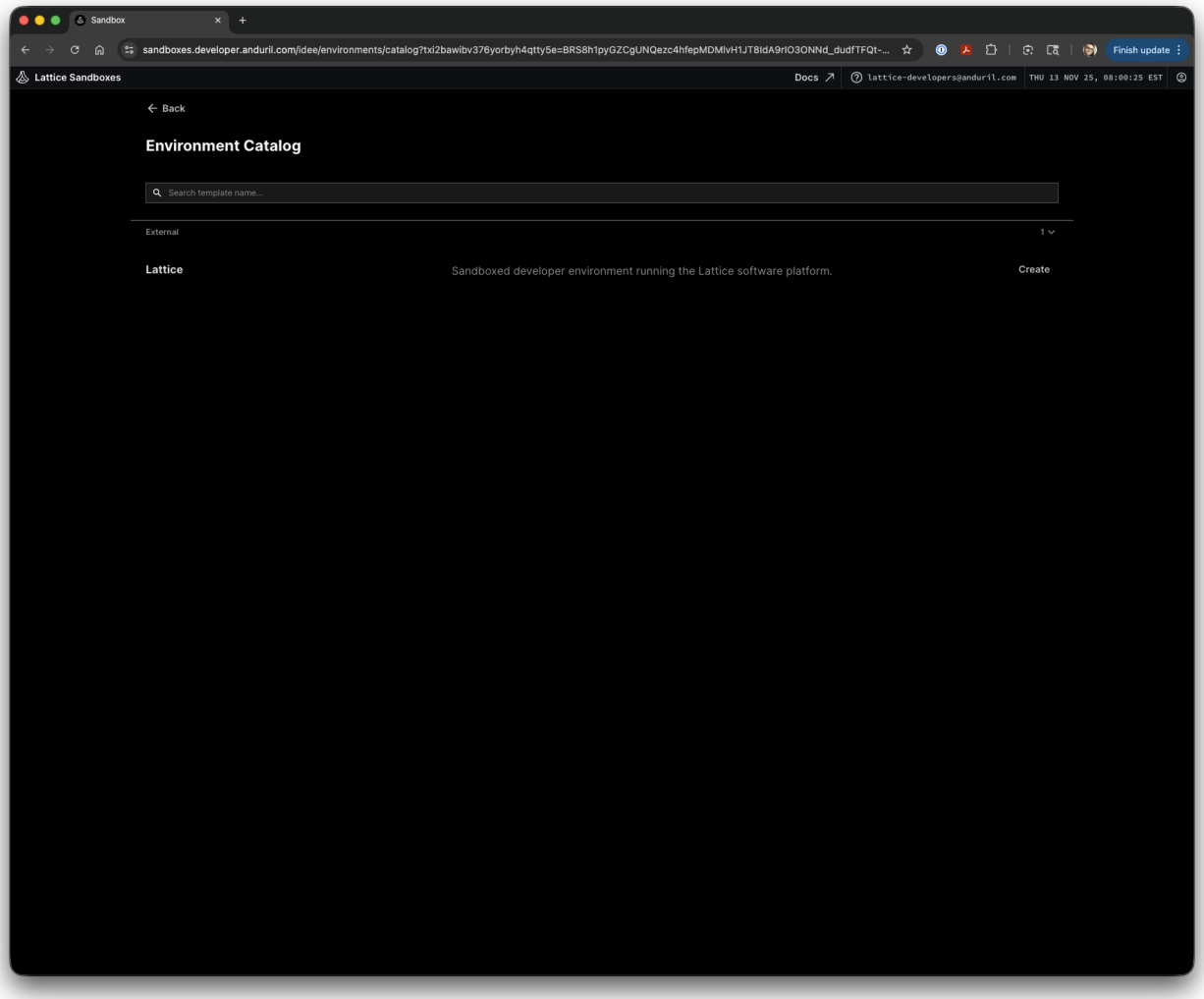
Step 1: Create a Lattice Sandbox Environment

First, you need to set up a Lattice sandbox environment for testing.

1. Navigate to the Lattice Sandboxes portal and login with your developer credentials

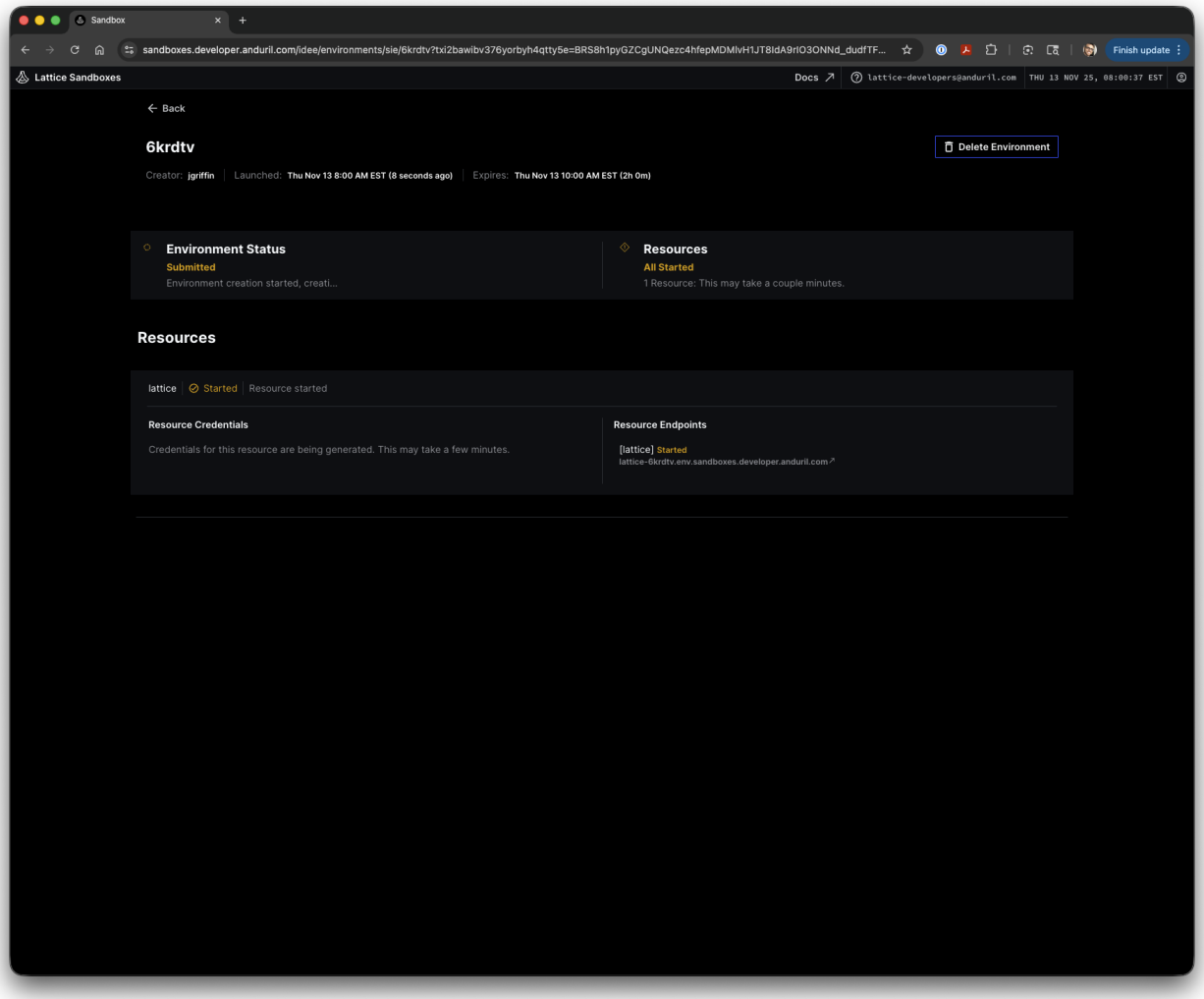


2. Click **Create new** to open the Environment Catalog



3. Select the **Lattice** template and click **Create**

   The sandbox environment will begin provisioning. This process typically takes a few minutes.

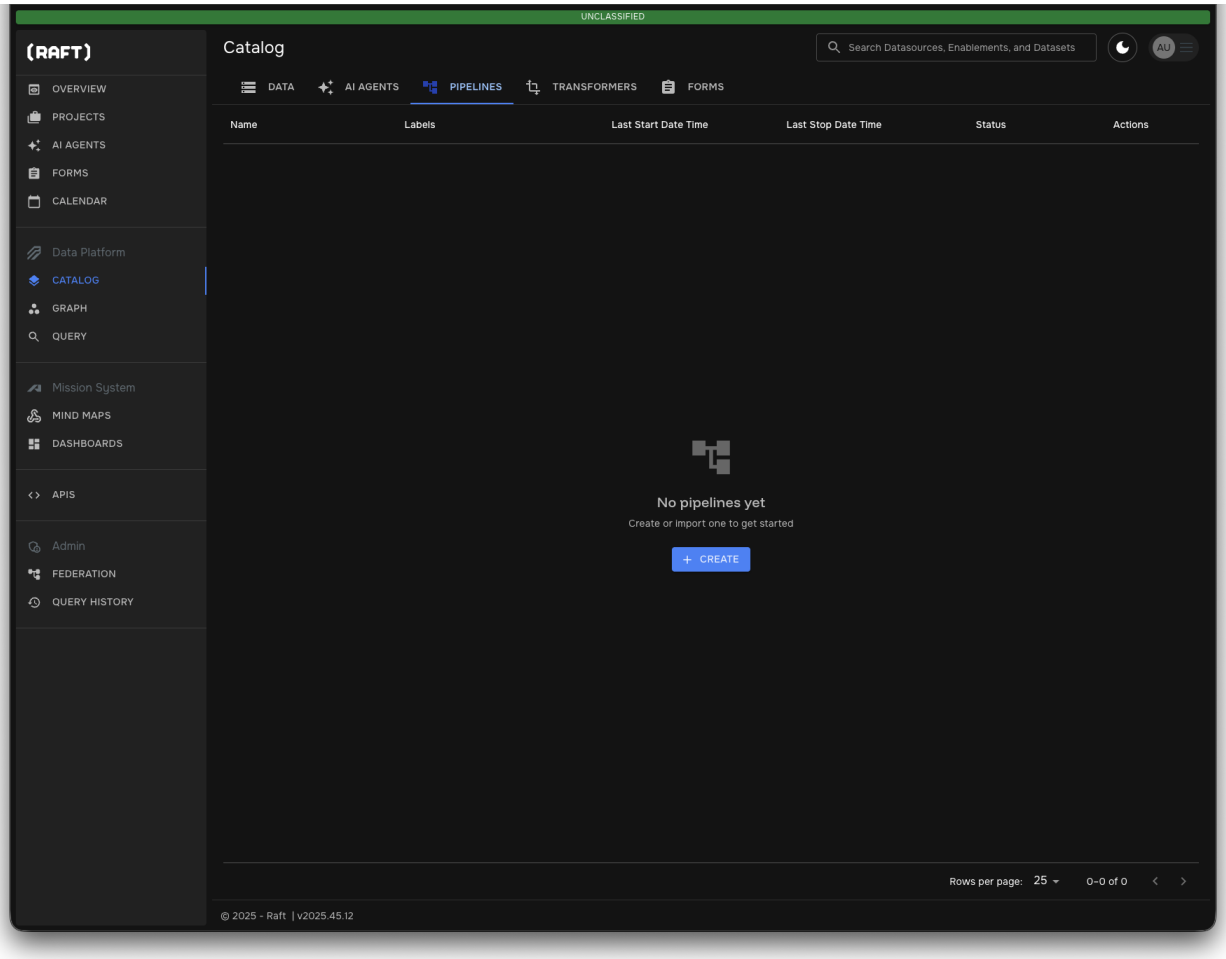


Lattice sandbox environments automatically expire after 2 hours. Plan your work accordingly or be prepared to recreate the environment if needed.

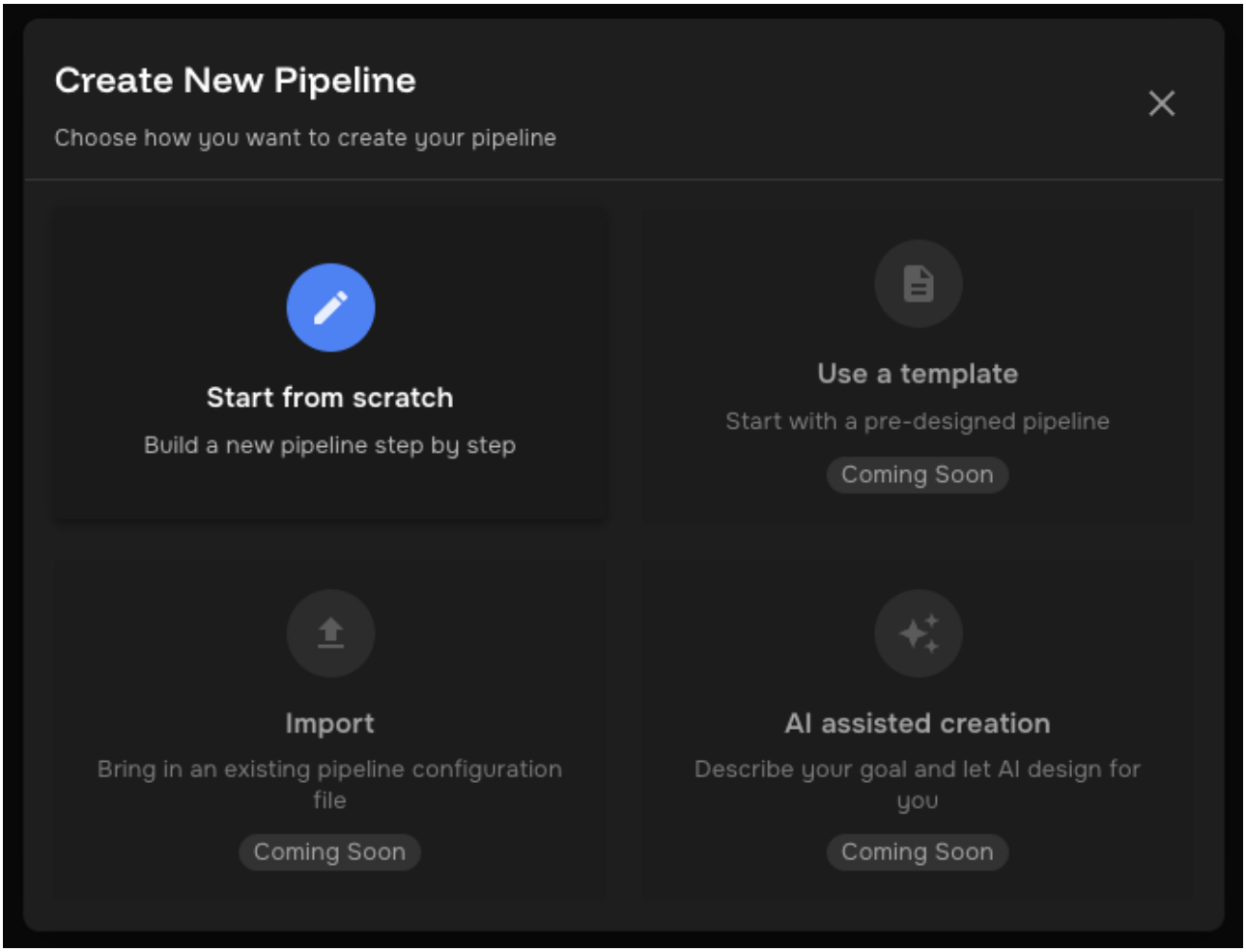
Step 2: Create a New Pipeline

With the Lattice environment starting, you can now create the data pipeline in SDL.

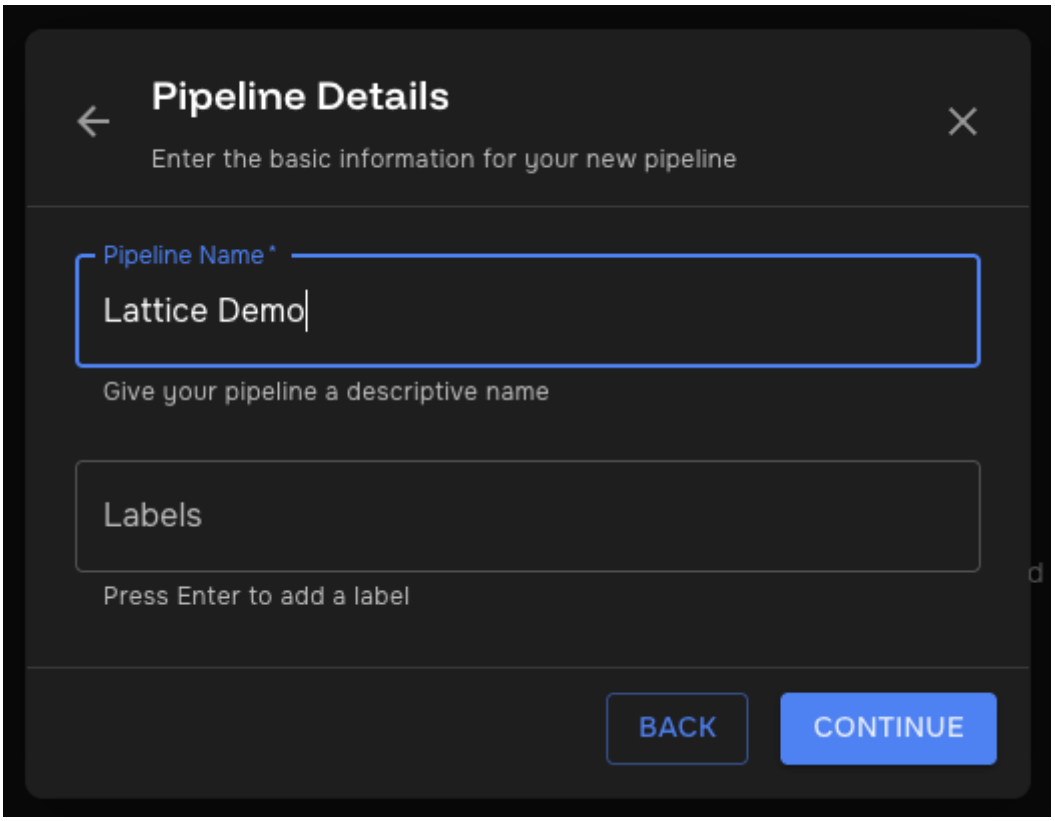
1. Navigate to **Catalog** > **Pipelines** in SDL



2. Click **CREATE** to open the pipeline creation dialog
3. Select **Start from scratch**



4. Enter a name for your pipeline (e.g., "Lattice Demo") and click **CONTINUE**

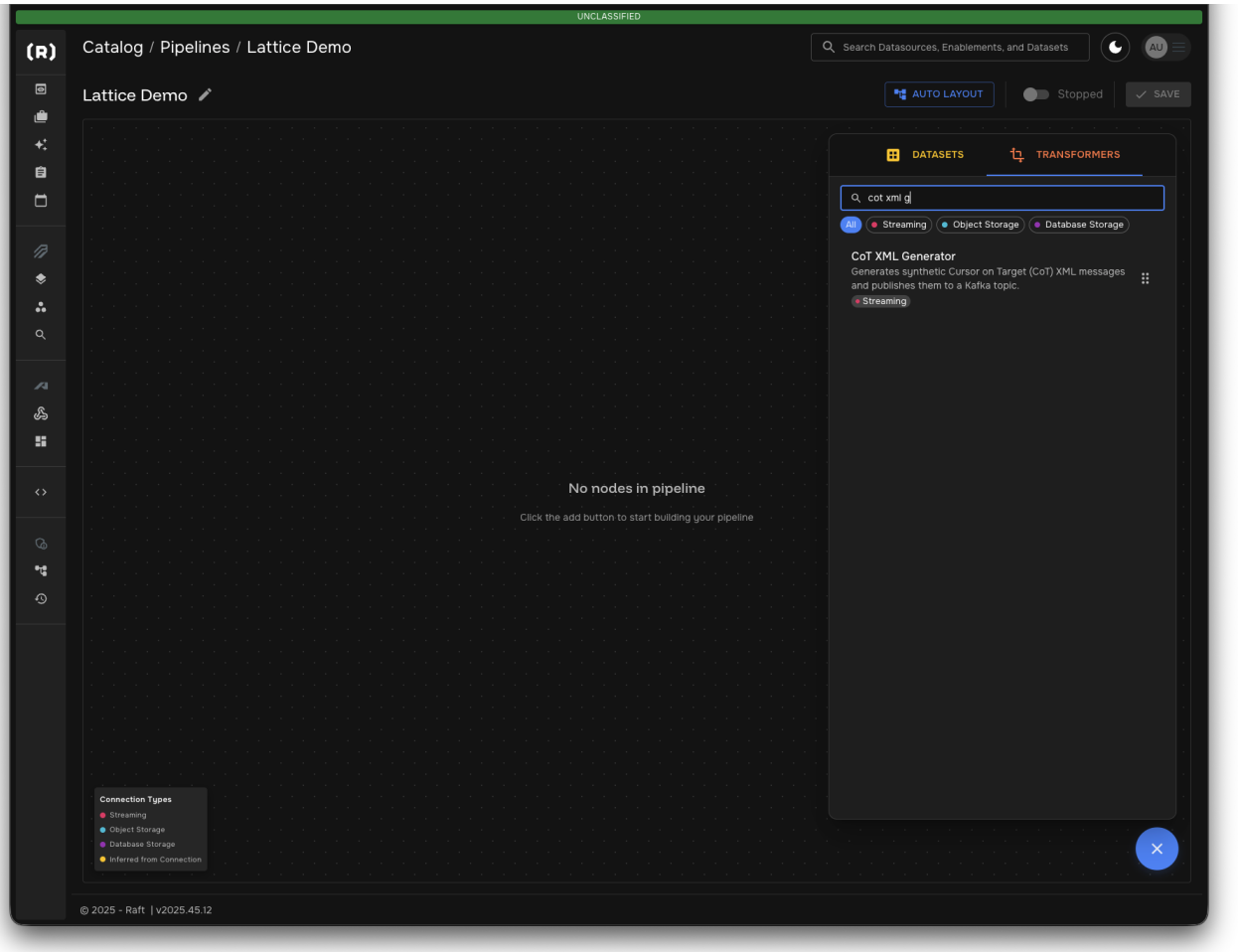


You'll be taken to the pipeline editor with an empty canvas.

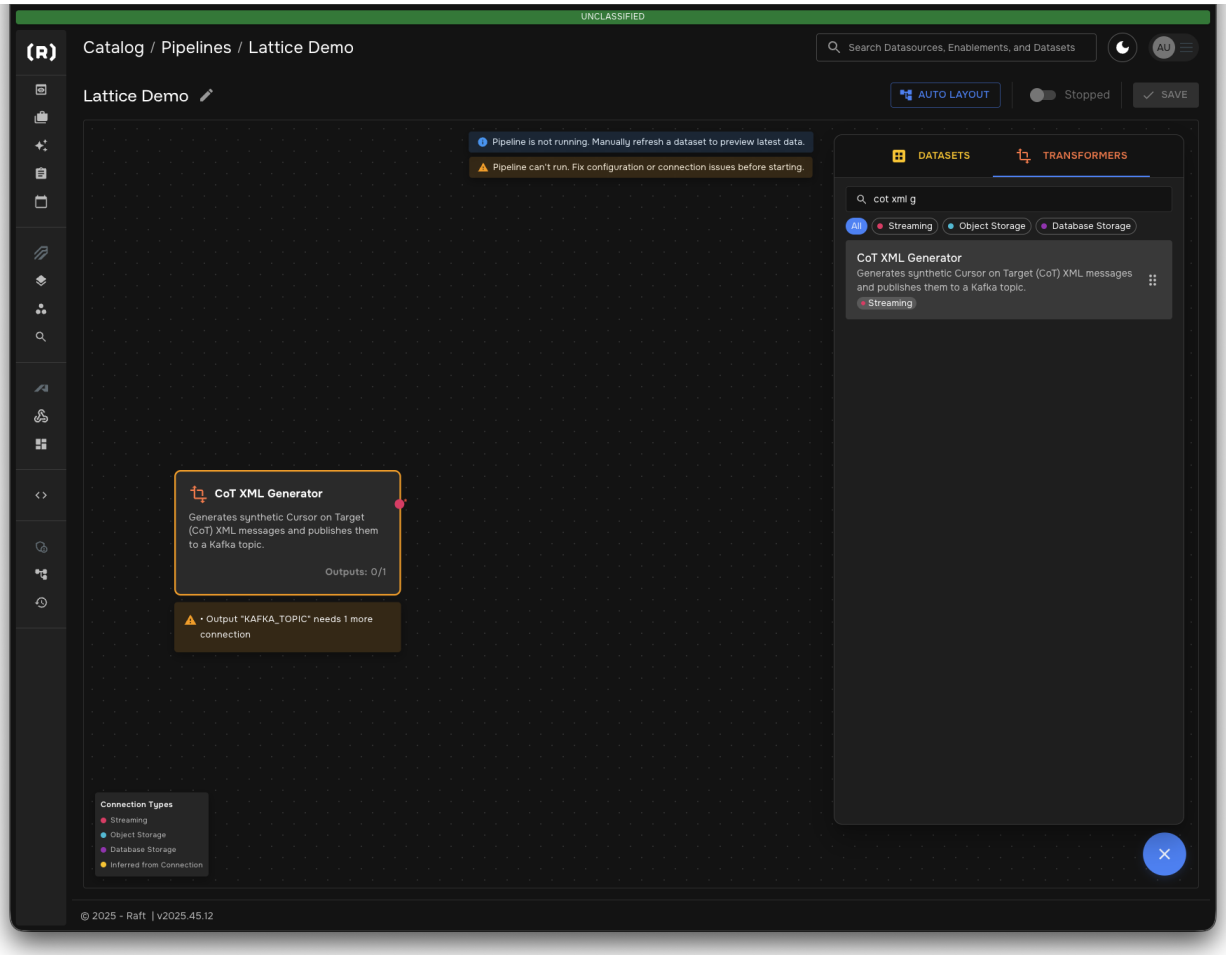
Step 3.1: Add the CoT XML Generator (alternative: connect to a TAK server)

The first component generates synthetic CoT XML messages for testing.

1. In the **TRANSFORMERS** panel, search for "cot xml generator"



2. Click on **CoT XML Generator** to add it to your pipeline

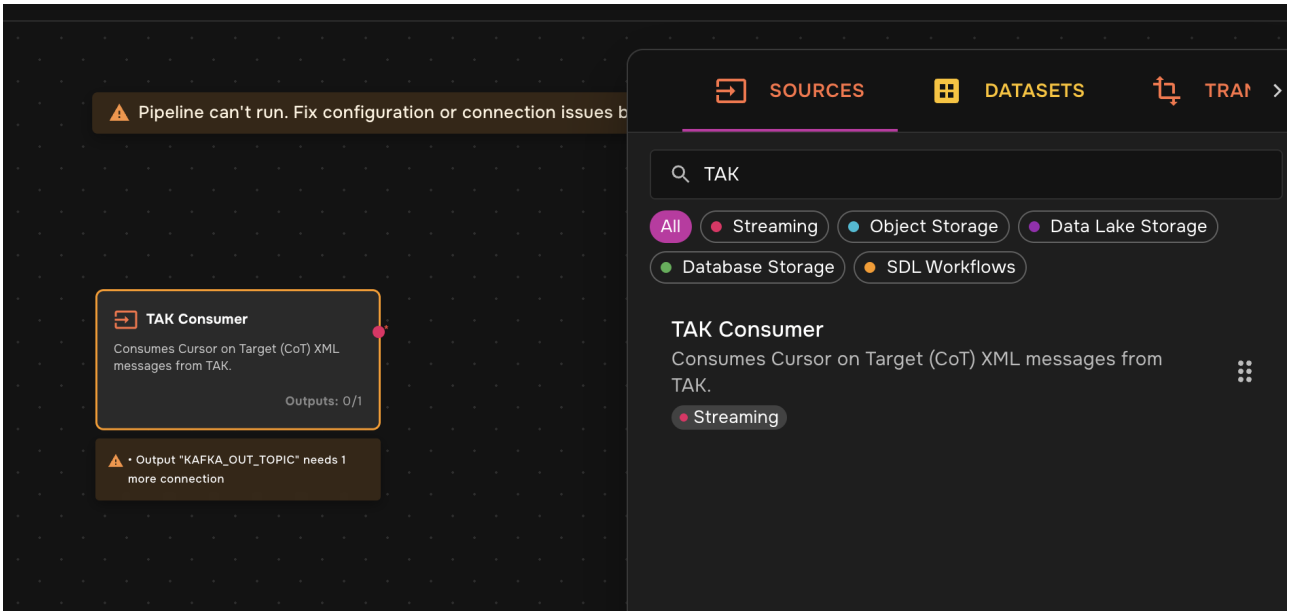


The warning icon indicates the transformer needs an output dataset.

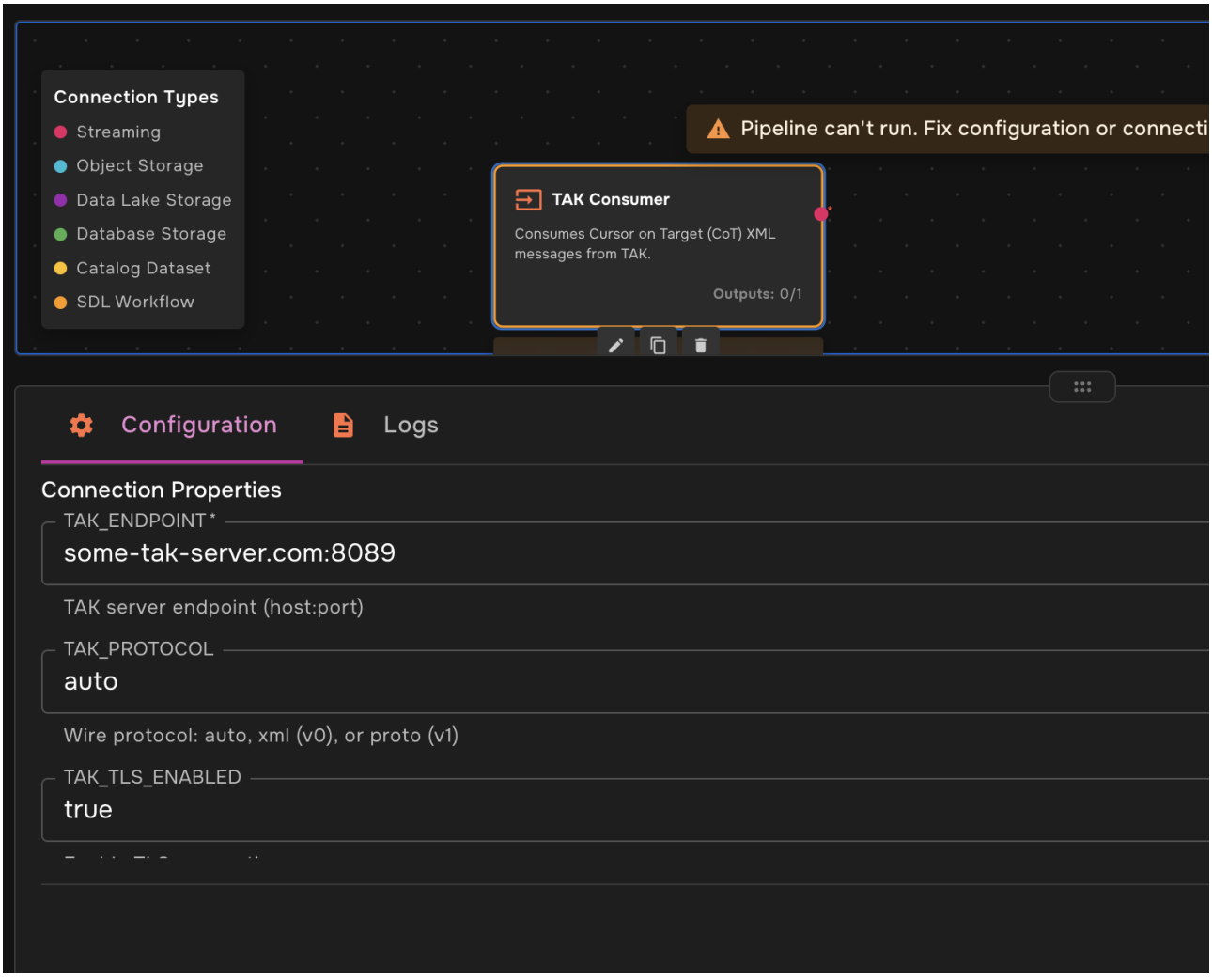
Step 3 Alternative: Add TAK Consumer

The first component consume CoT events directly from a TAK Server connection.

1. In the **SOURCES** panel, search for "tak consumer"
2. Click on **TAK Consumer** to add it to your pipeline



3. Click on the TAK Consumer card and add connection information (url, credentials) under the configuration tab

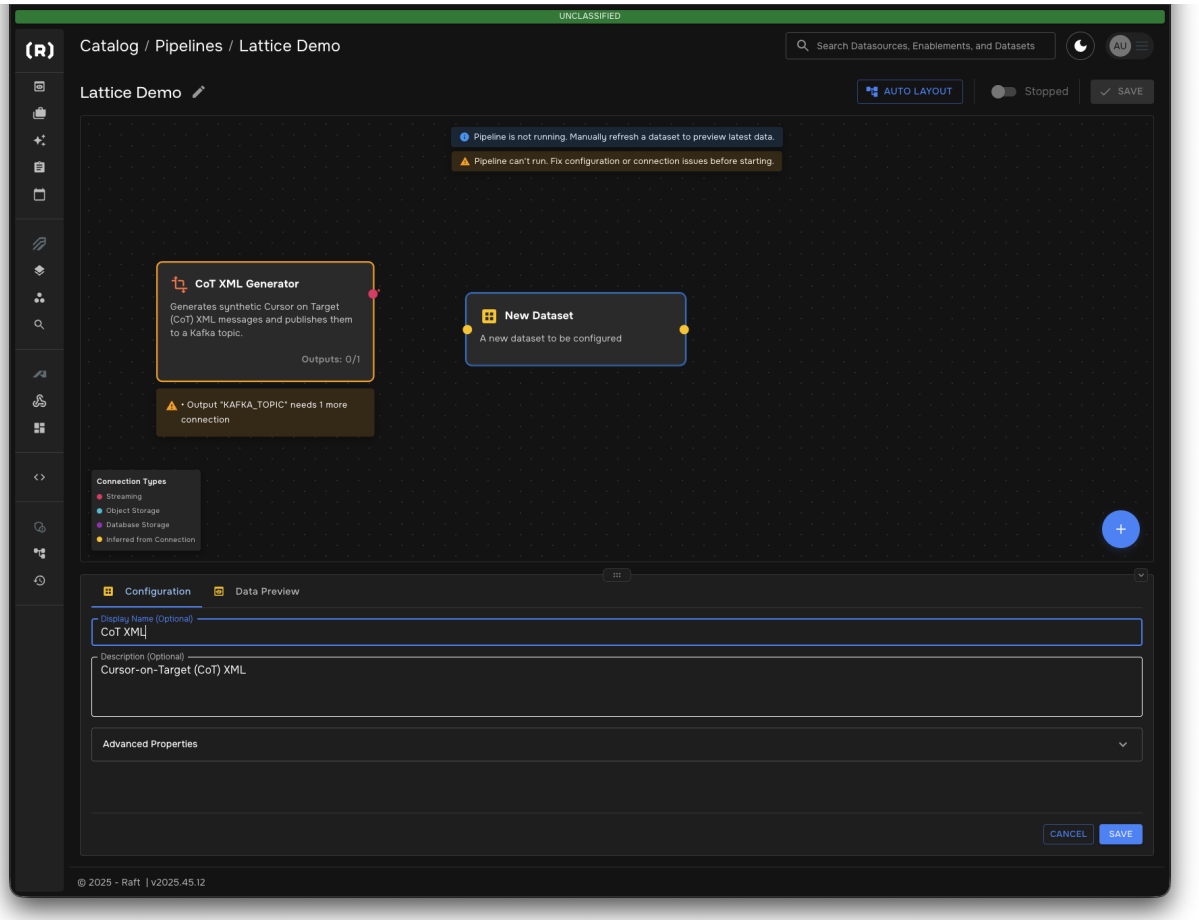


The warning icon indicates the transformer needs an output dataset and configuration details.

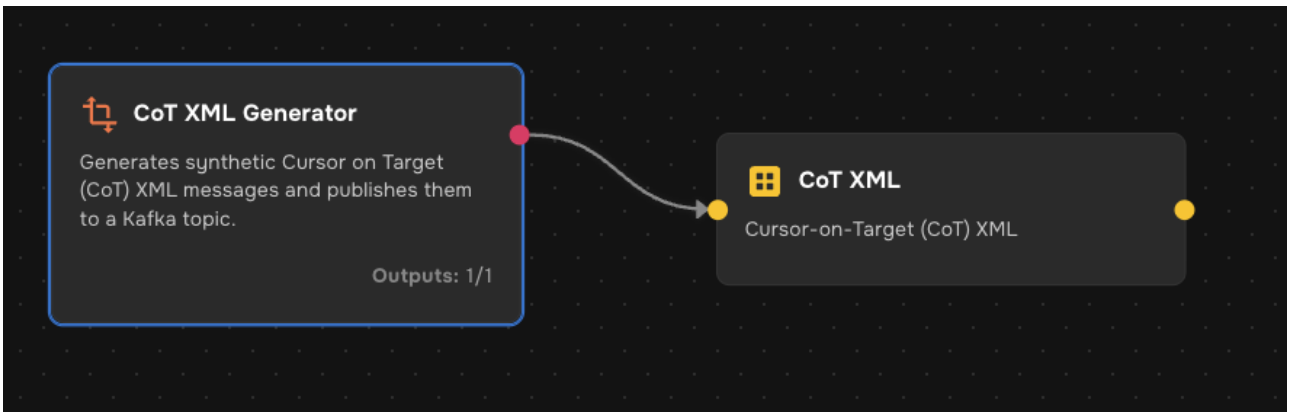
Step 4: Create CoT XML Dataset

Next, create a dataset to connect the generator output.

1. Select the **DATASETS** tab
2. Configure a new dataset:
   a. **Display Name**: `CoT XML`
   b. **Description**: `Cursor-on-Target (CoT) XML`



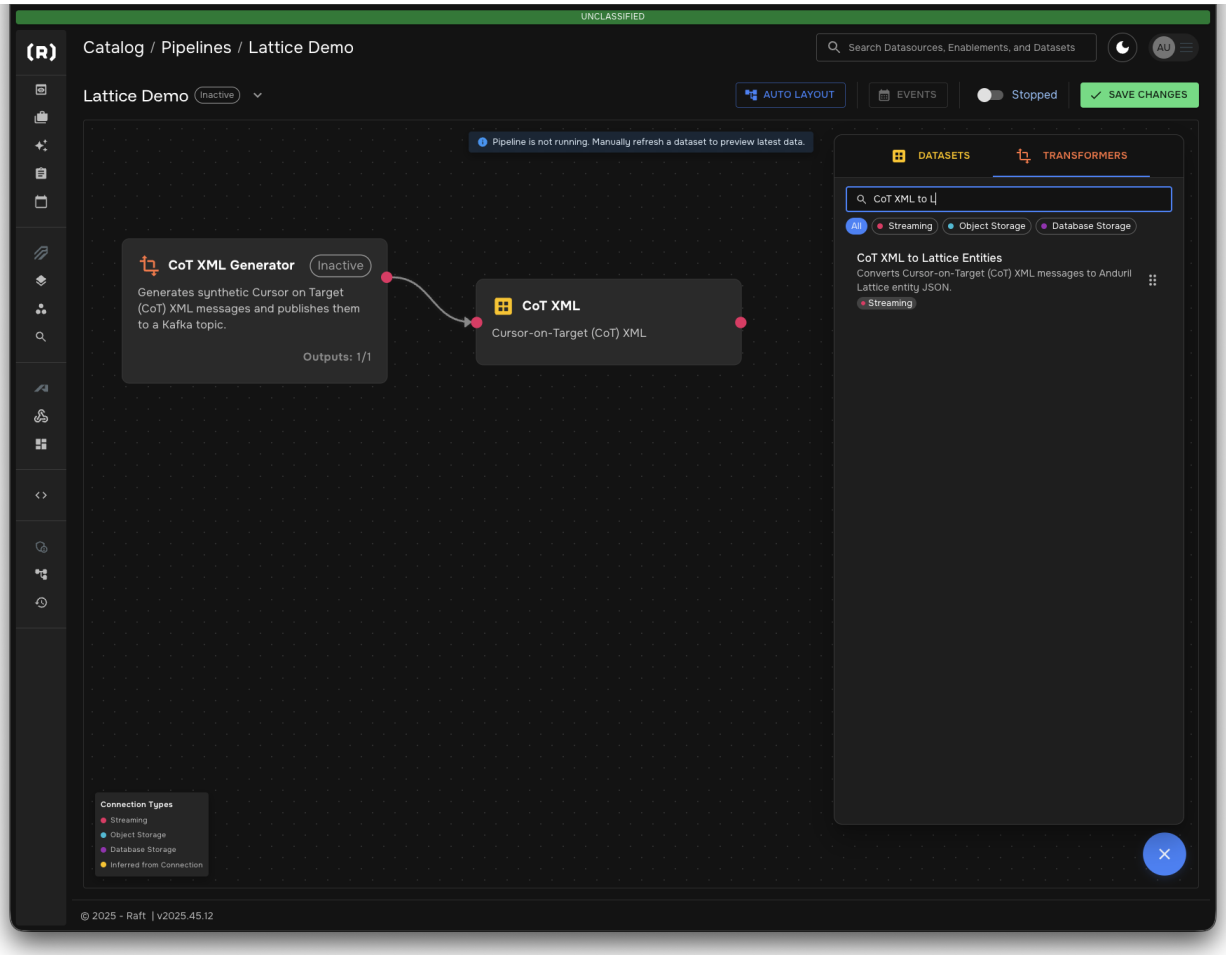
3. Connect the **CoT XML Generator** output to the **CoT XML** dataset by clicking and dragging between the connection points



Step 5: Add the CoT to Lattice Transformer

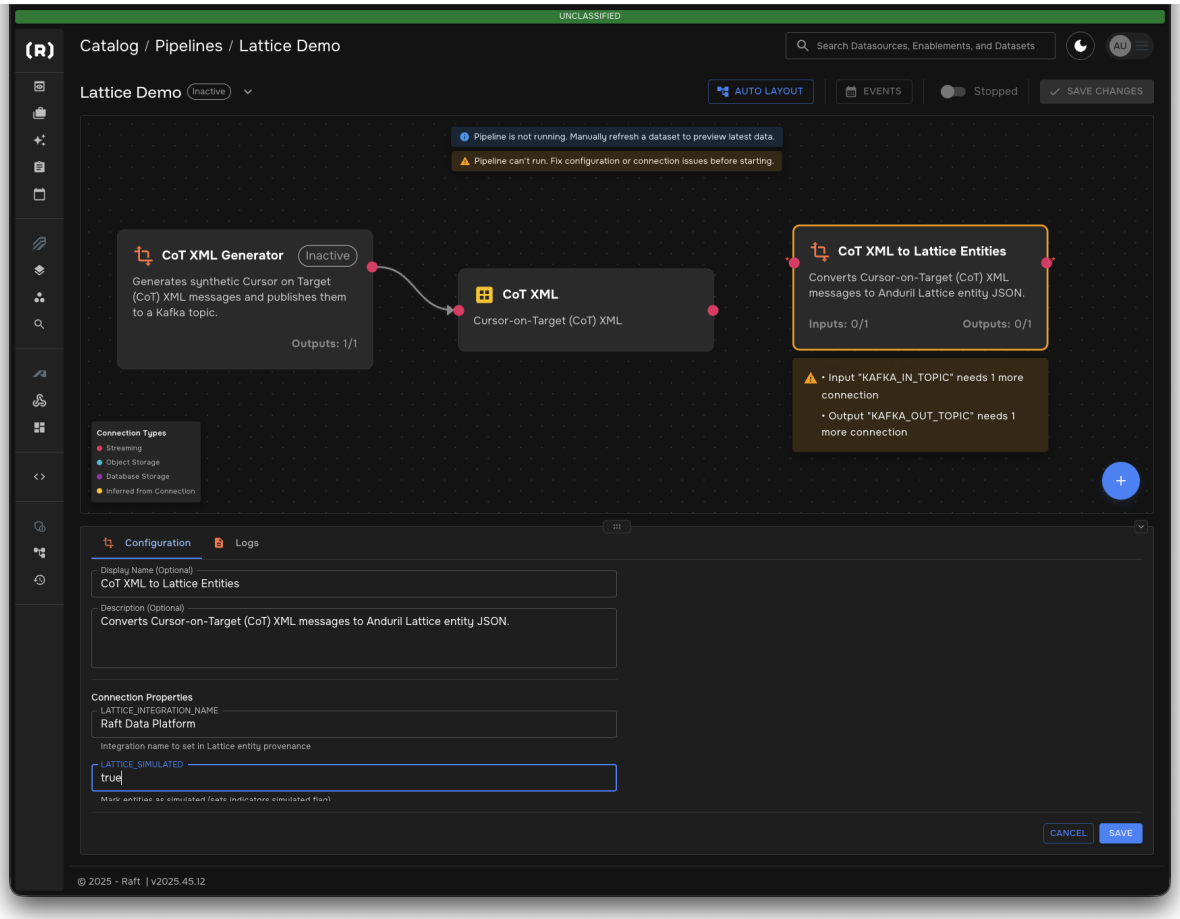
Now add the transformer that converts CoT XML to Lattice entity format.

1. Search for "CoT XML to Lattice" in the **TRANSFORMERS** panel

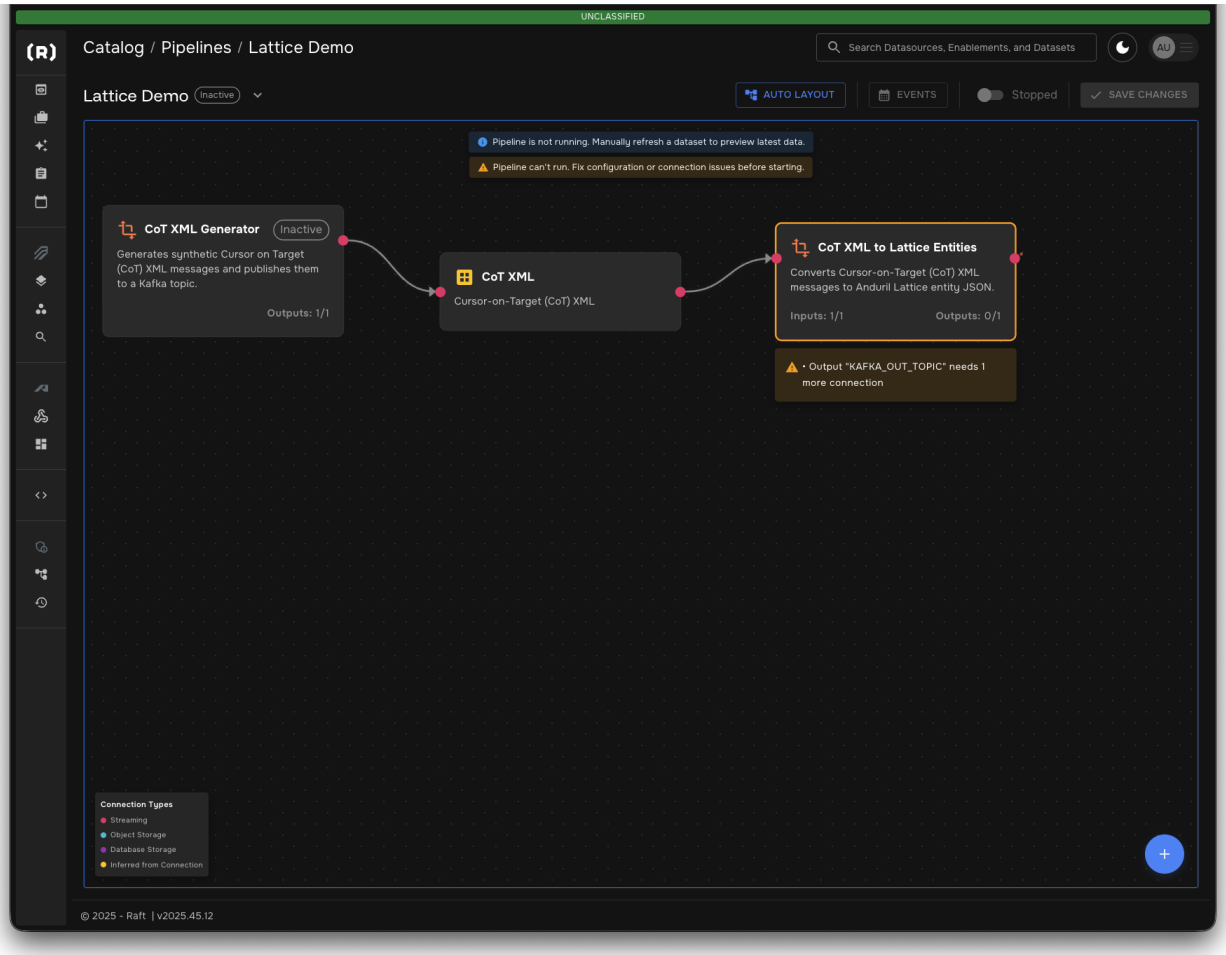


2. Select **CoT XML to Lattice Entities** and fill in the fields:
   - **LATTICE_SIMULATED**: `true`

This marks the entity so it appears as a "simulated" entity in Lattice.



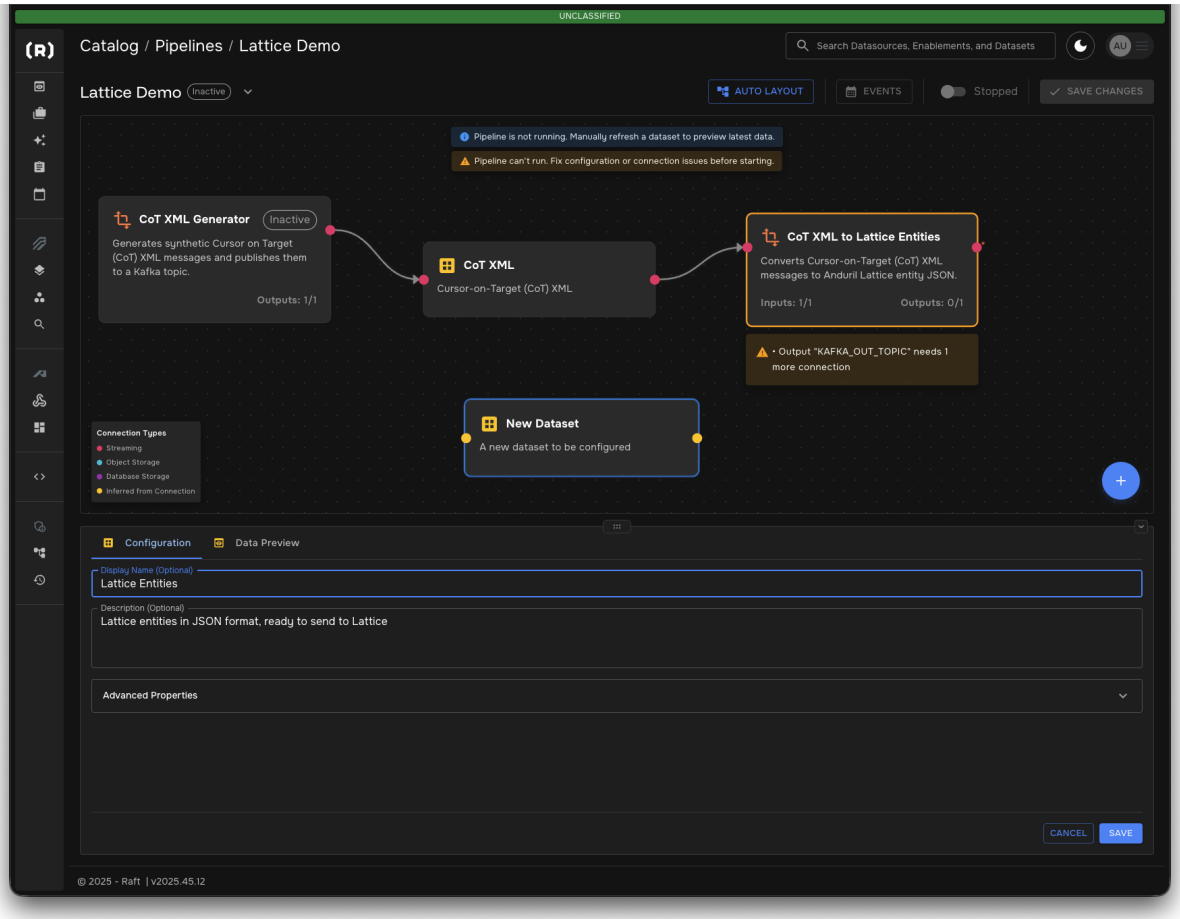
3. Connect the **CoT XML** dataset output to the **CoT XML to Lattice Entities** transformer input



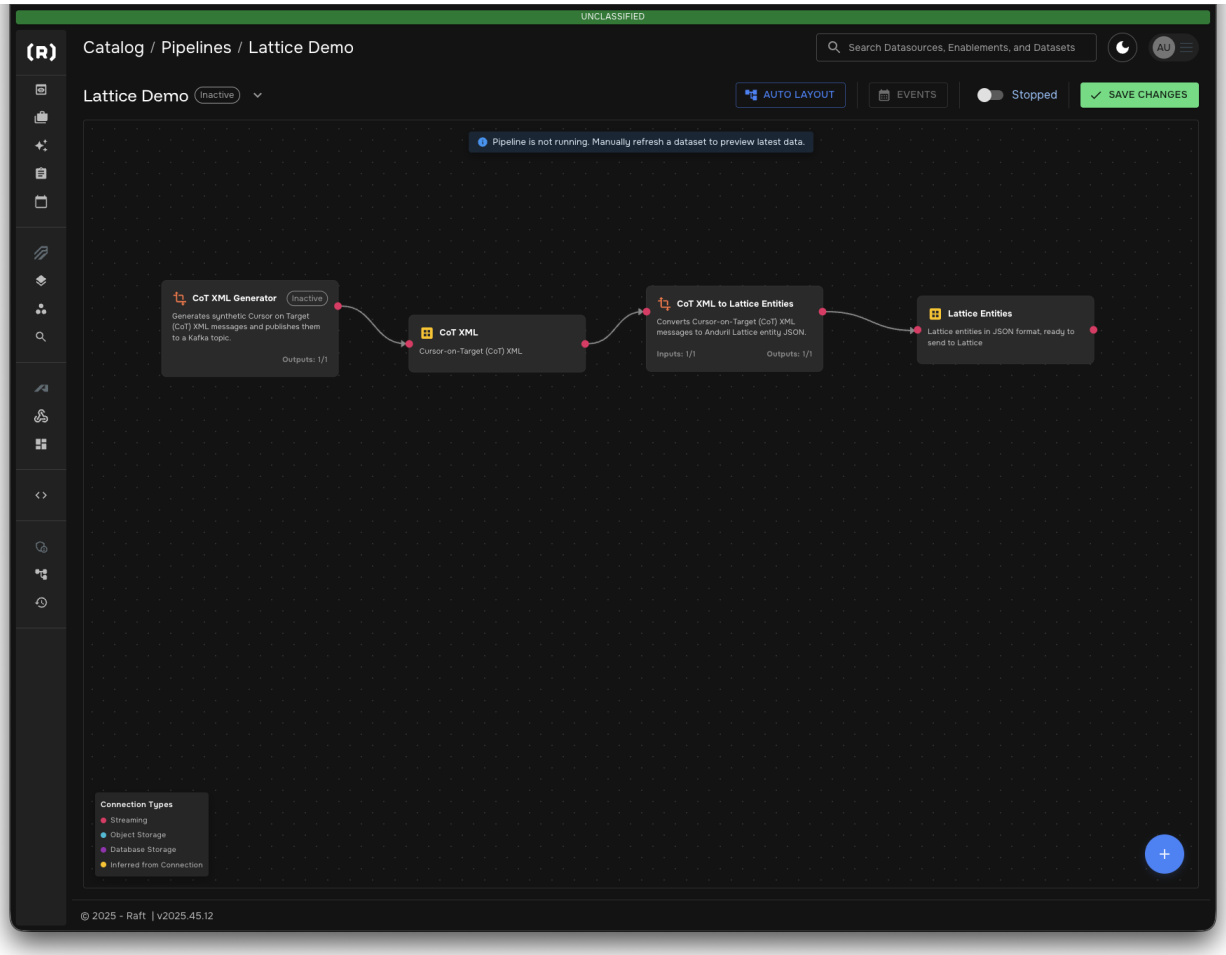
Step 6: Create Lattice Dataset

Create a dataset to hold the transformed Lattice entities.

1. Click the + button on the dataset node that appears after the transformer
2. Configure a new dataset:
    a. **Display Name**: `Lattice Entities`
    b. **Description**: `Lattice entities in JSON format, ready to send to Lattice`



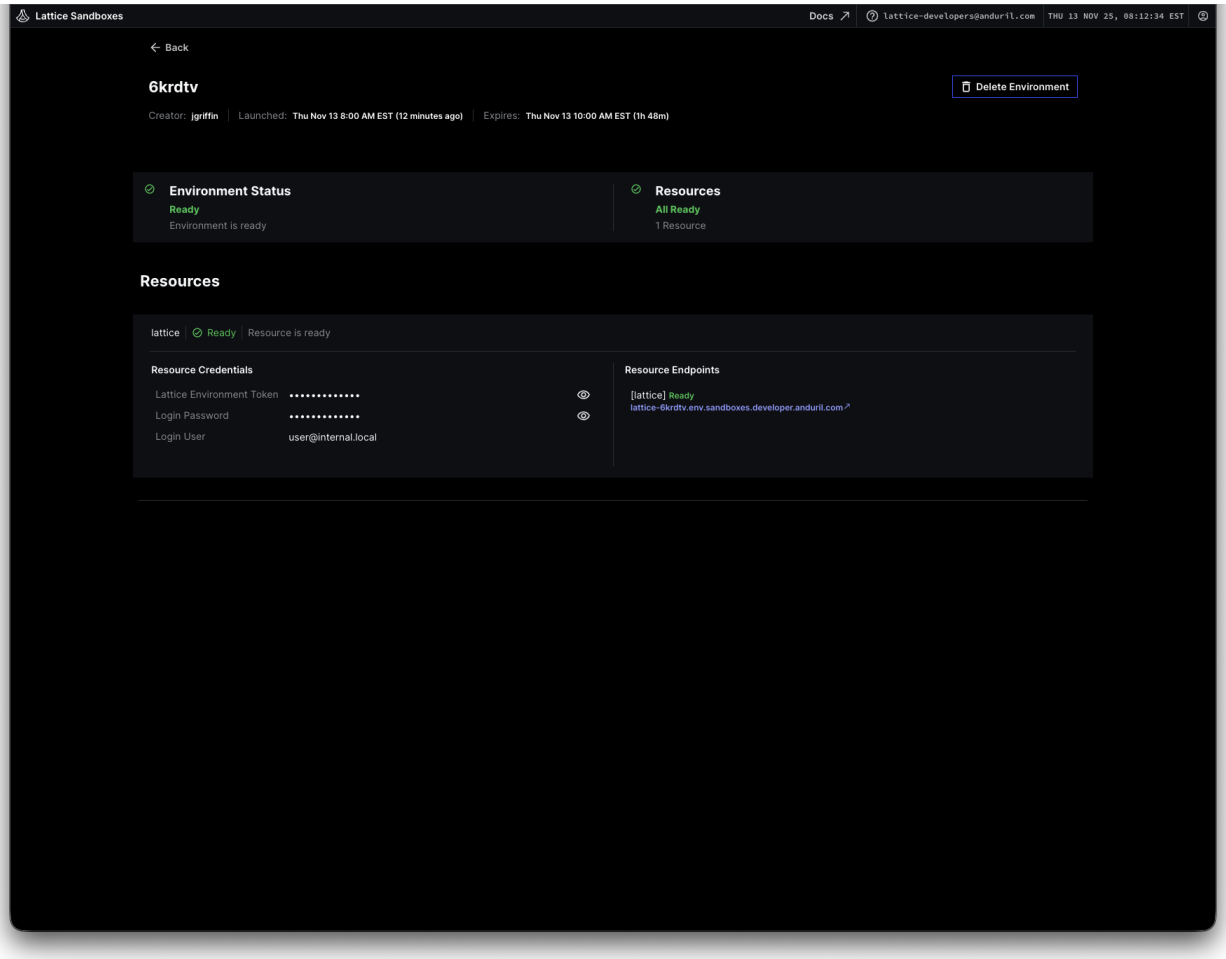
3. Connect the output from the **CoT XML to Lattice Entities** transformer to the **Lattice Entities** dataset.



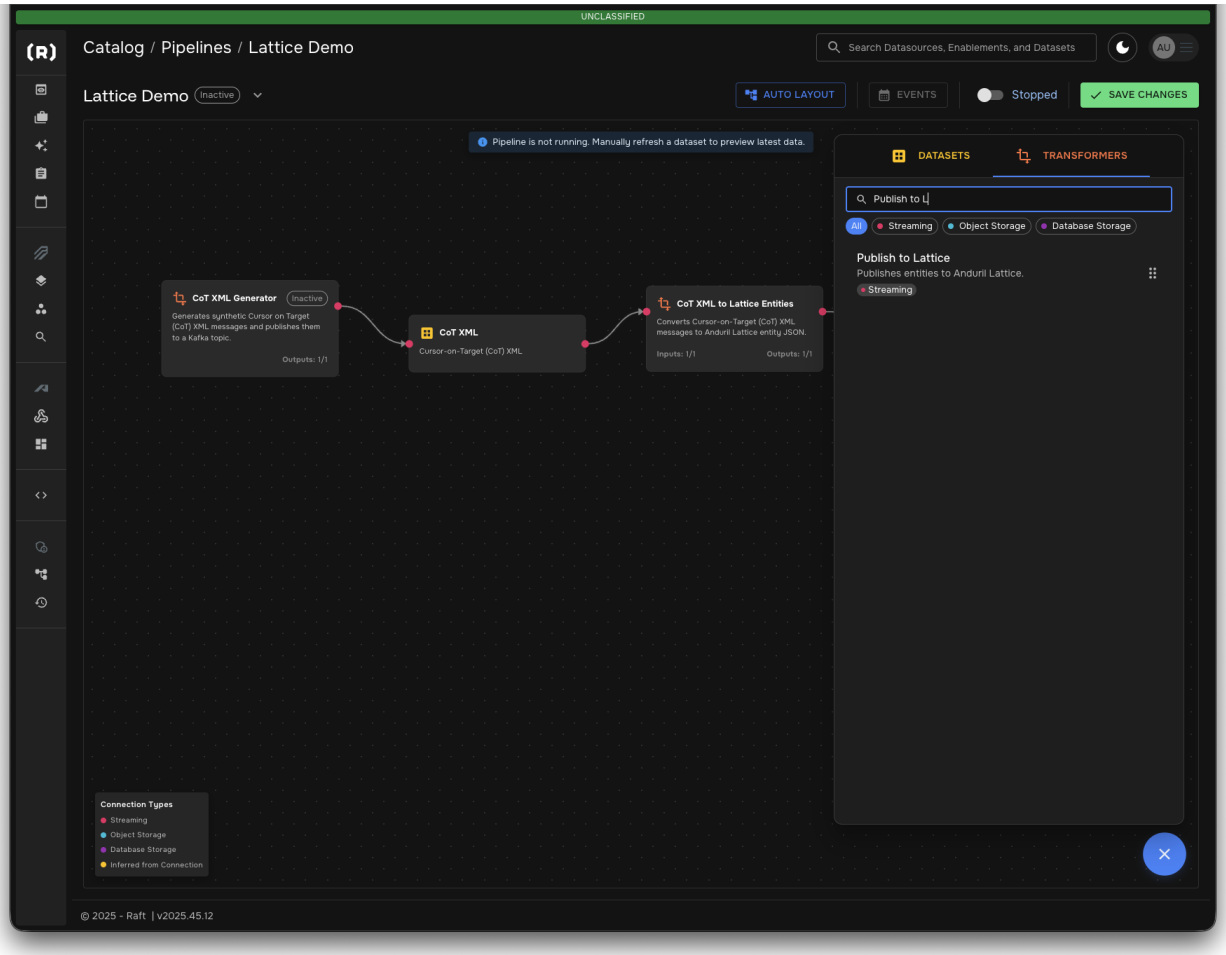
Step 7: Add the Publish to Lattice Transformer

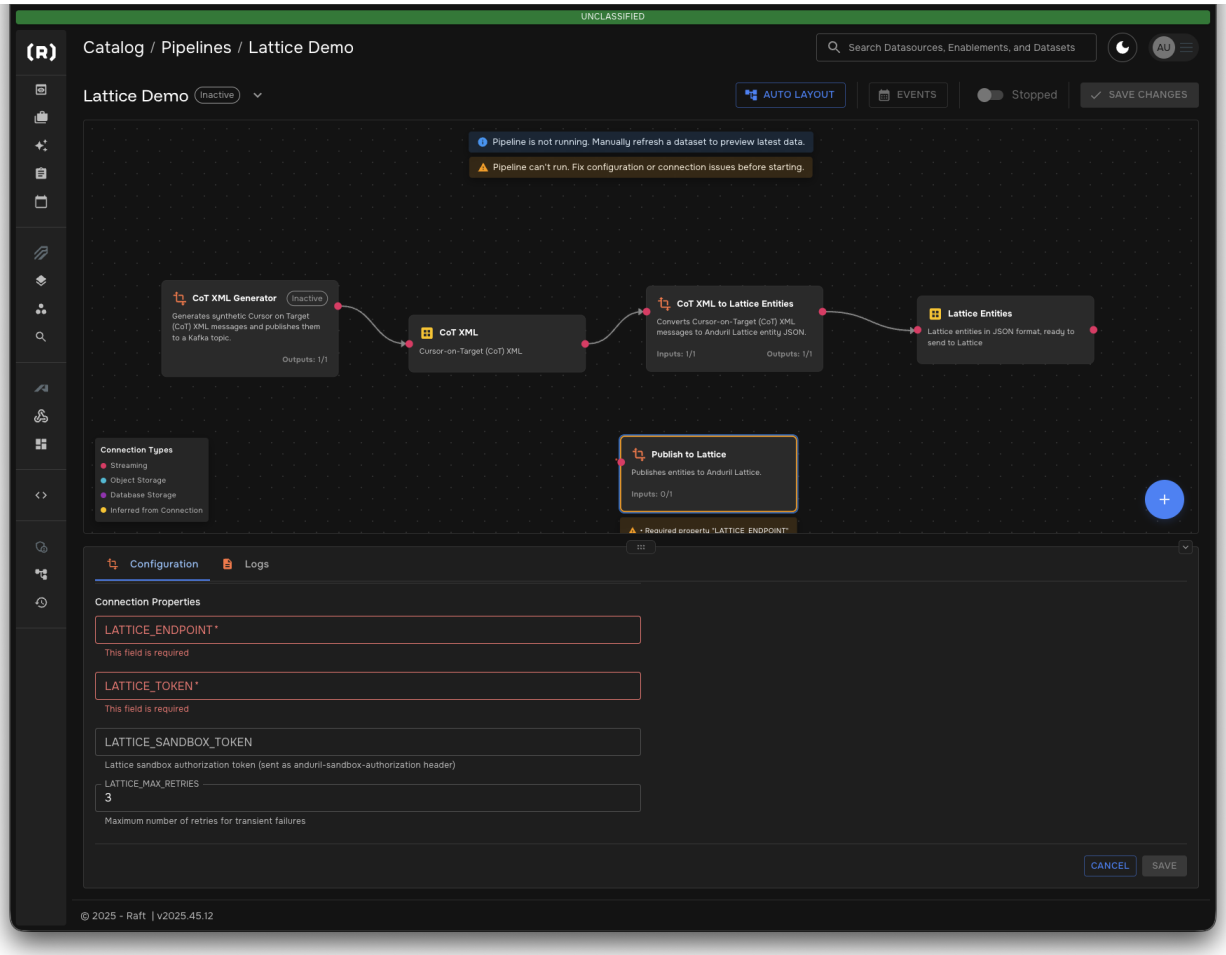
Finally, add the transformer that publishes entities to Lattice.

1. Return to your Lattice sandbox and wait for it to finish provisioning

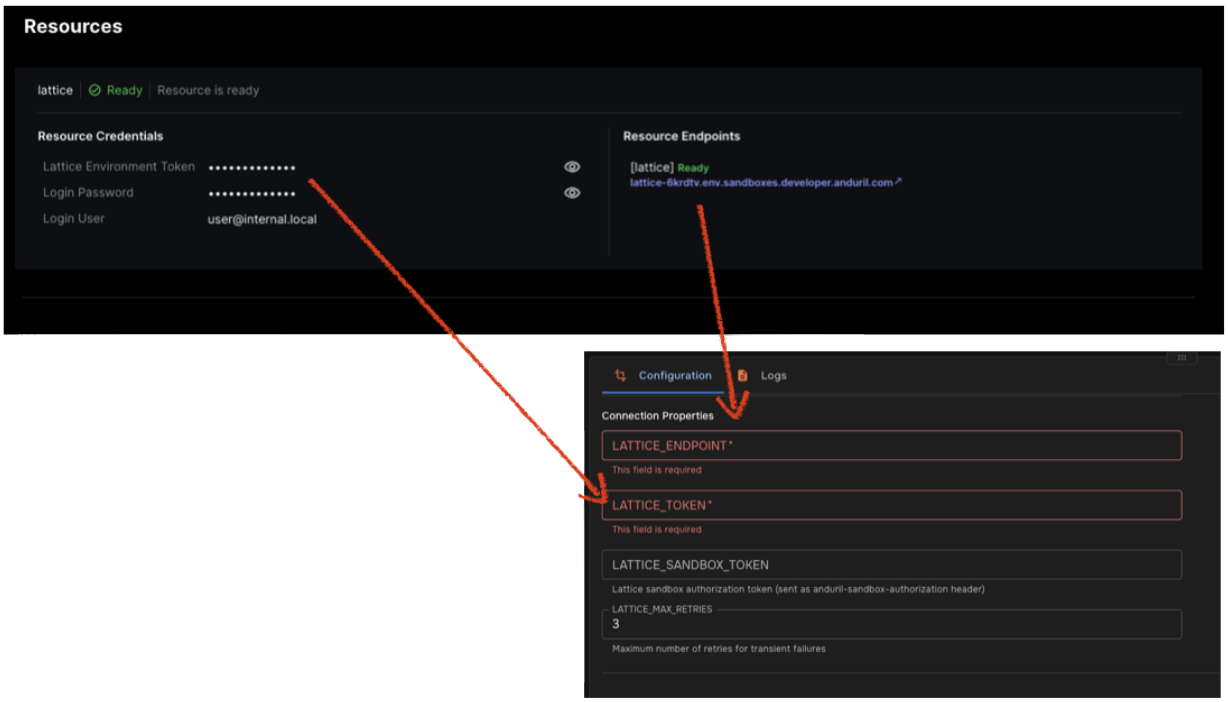


2. Search for "Publish to Lattice" in the **TRANSFORMERS** panel

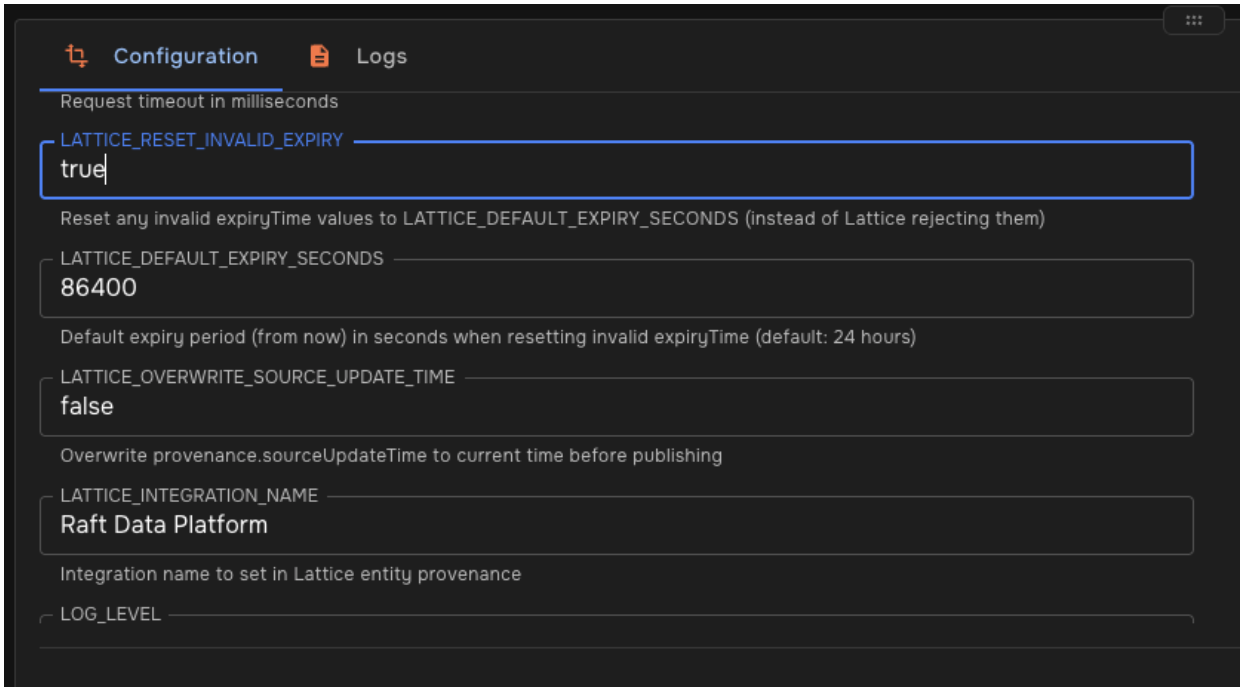

UNCLASSIFIED
Catalog / Pipelines / Lattice Demo
Search Datasources, Enablements, and Datasets
Lattice Demo
Inactive
AUTO LAYOUT
EVENTS
Stopped
SAVE CHANGES
Pipeline is not running. Manually refresh a dataset to preview latest data.
DATASETS
TRANSFORMERS
Publish to L
All
Streaming
Object Storage
Database Storage
Publish to Lattice
Publishes entities to Anduril Lattice.
Streaming
CoT XML Generator
Inactive
Generates synthetic Cursor on Target (CoT) XML messages and publishes them to a Kafka topic.
Outputs: 1/1
CoT XML
Cursor-on-Target (CoT) XML
CoT XML to Lattice Entities
Converts Cursor-on-Target (CoT) XML messages to Anduril Lattice entity JSON.
Inputs: 1/1
Outputs: 1/1
Connection Types
Streaming
Object Storage
Database Storage
Inferred from Connection
© 2025 - Raft | v2025.45.12


UNCLASSIFIED
Catalog / Pipelines / Lattice Demo
Search Datasources, Enablements, and Datasets
Lattice Demo
Inactive
AUTO LAYOUT
EVENTS
Stopped
SAVE CHANGES
Pipeline is not running. Manually refresh a dataset to preview latest data.
Pipeline can't run. Fix configuration or connection issues before starting.
CoT XML Generator
Inactive
Generates synthetic Cursor on Target (CoT) XML messages and publishes them to a Kafka topic.
Outputs: 1/1
CoT XML
Cursor-on-Target (CoT) XML
CoT XML to Lattice Entities
Converts Cursor-on-Target (CoT) XML messages to Anduril Lattice entity JSON.
Inputs: 1/1
Outputs: 1/1
Lattice Entities
Lattice entities in JSON format, ready to send to Lattice
Connection Types
Streaming
Object Storage
Database Storage
Inferred from Connection
Publish to Lattice
Publishes entities to Anduril Lattice.
Inputs: 0/1
Configuration
Logs
Connection Properties
LATTICE_ENDPOINT *
This field is required
LATTICE_TOKEN *
This field is required
LATTICE_SANDBOX_TOKEN
Lattice sandbox authorization token (sent as anduril-sandbox-authorization header)
LATTICE_MAX_RETRIES
3
Maximum number of retries for transient failures
CANCEL
SAVE
© 2025 - Raft | v2025.45.12

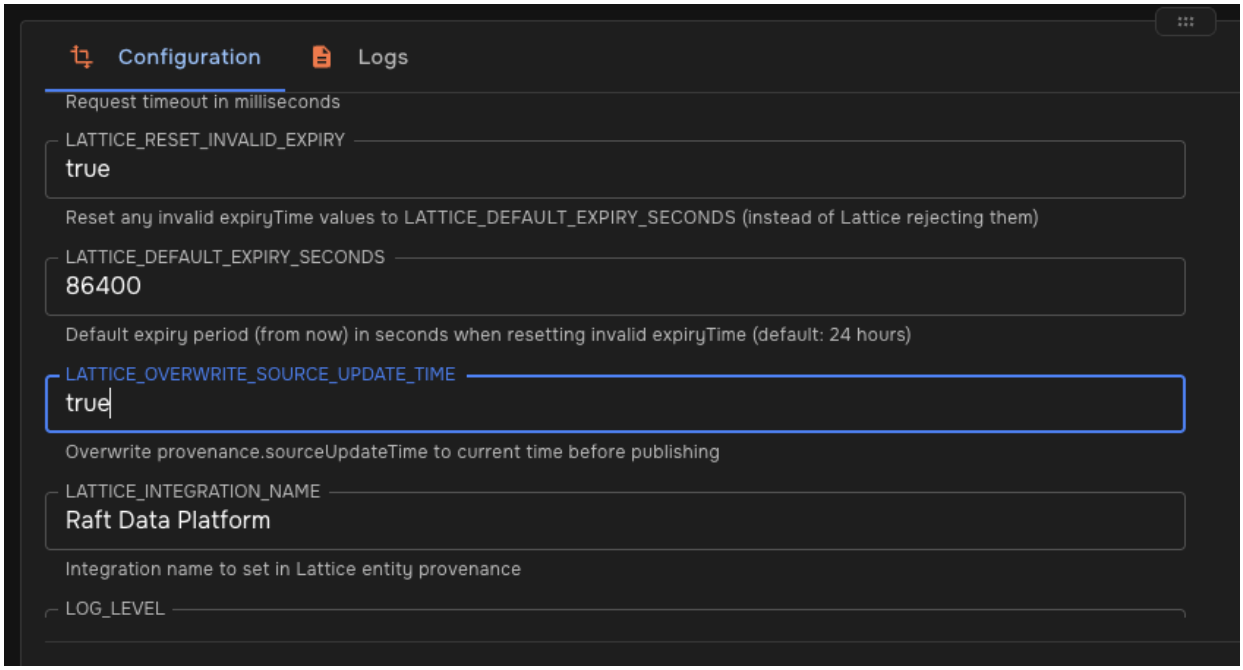
3. Configure the **Publish to Lattice** transformer
    a. **LATTICE_ENDPOINT**: endpoint to the lattice environment
    b. **LATTICE_TOKEN**: token for connecting to lattice environment
    c. **LATTICE_SANDBOX_TOKEN**: this is your private "anduril developer" sandbox token (only applies when connecting to a developer sandbox environment)



    d. **LATTICE_RESET_INVALID_EXPIRY**: `true`



    e. **LATTICE_OVERWRITE_SOURCE_UPDATE_TIME**: `true`

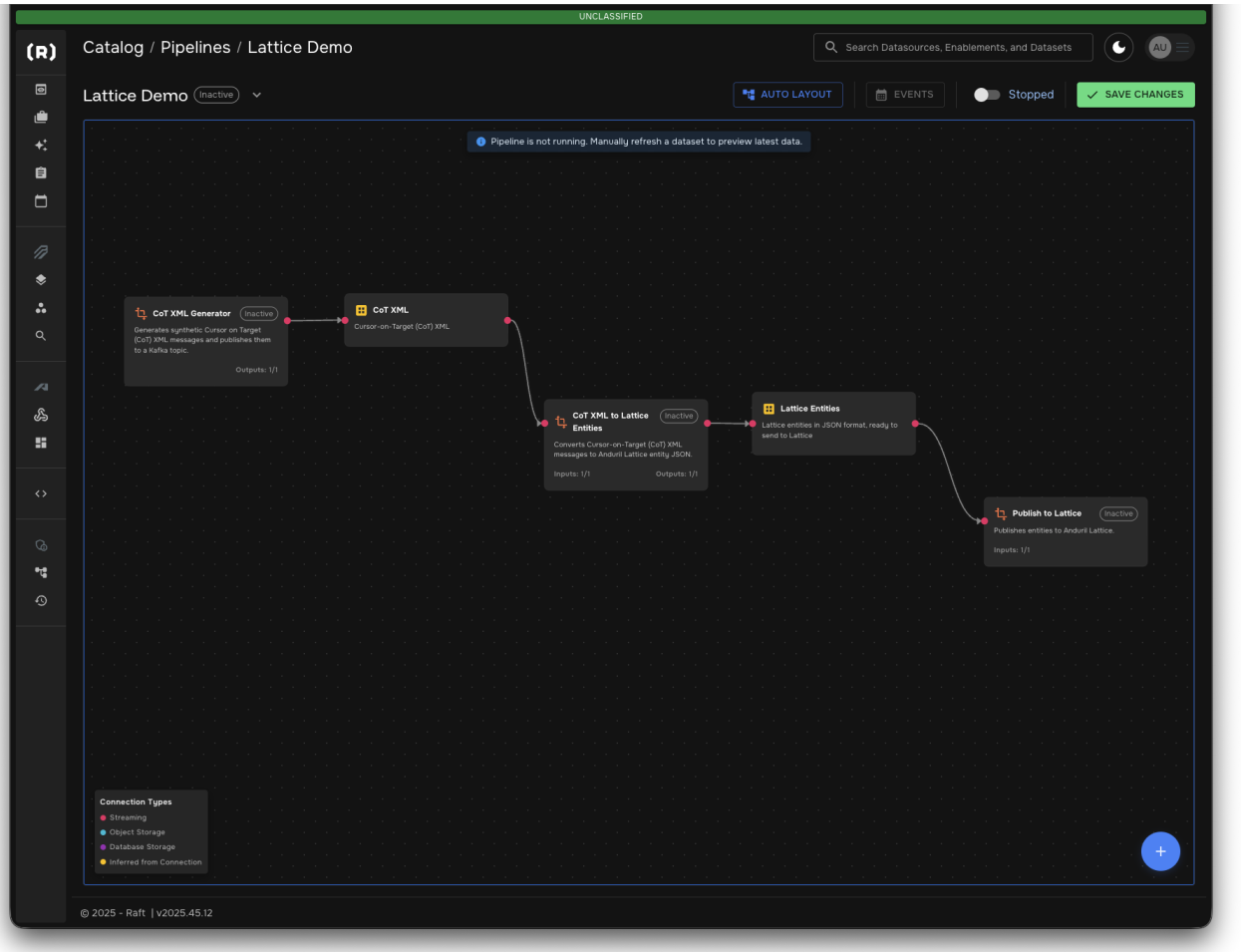


4. Connect the **Lattice Entities** dataset to the **Publish to Lattice** sink.

Step 8: Save and Run the Pipeline

Your pipeline is now complete and ready to run.

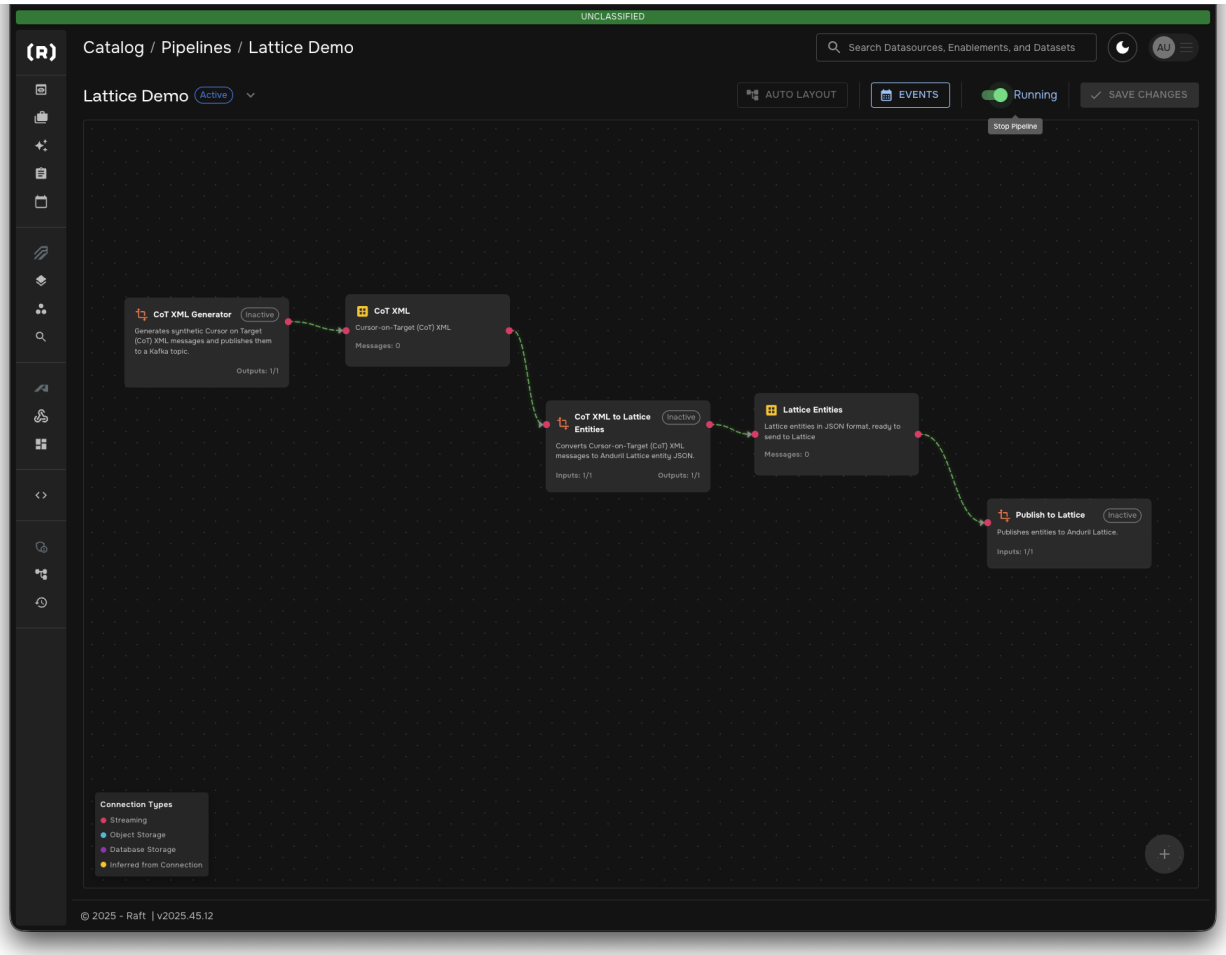
1. Review the complete pipeline:



The pipeline flows from left to right:

- CoT XML Generator (or TAK Connection) → CoT XML dataset
- CoT XML dataset → CoT XML to Lattice Entities transformer
- Lattice Entities transformer → Lattice Entities dataset
- Lattice Entities dataset → Publish to Lattice transformer

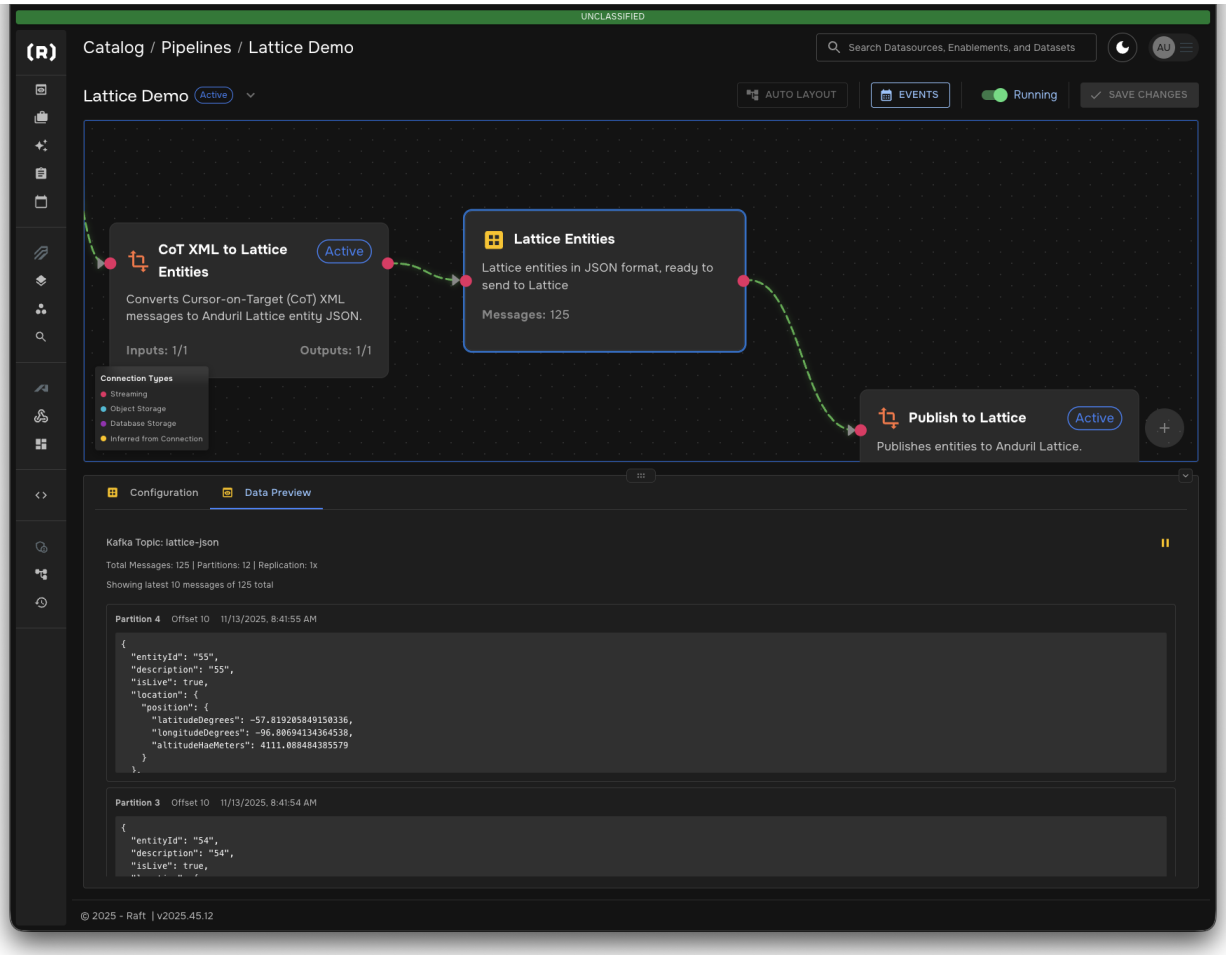
2. Click **SAVE CHANGES** in the top right
3. Click the toggle to change the pipeline state from **Stopped** to **Running**



The pipeline will start processing:

- Generating synthetic CoT messages
- Transforming them to Lattice entities
- Publishing them to your Lattice sandbox

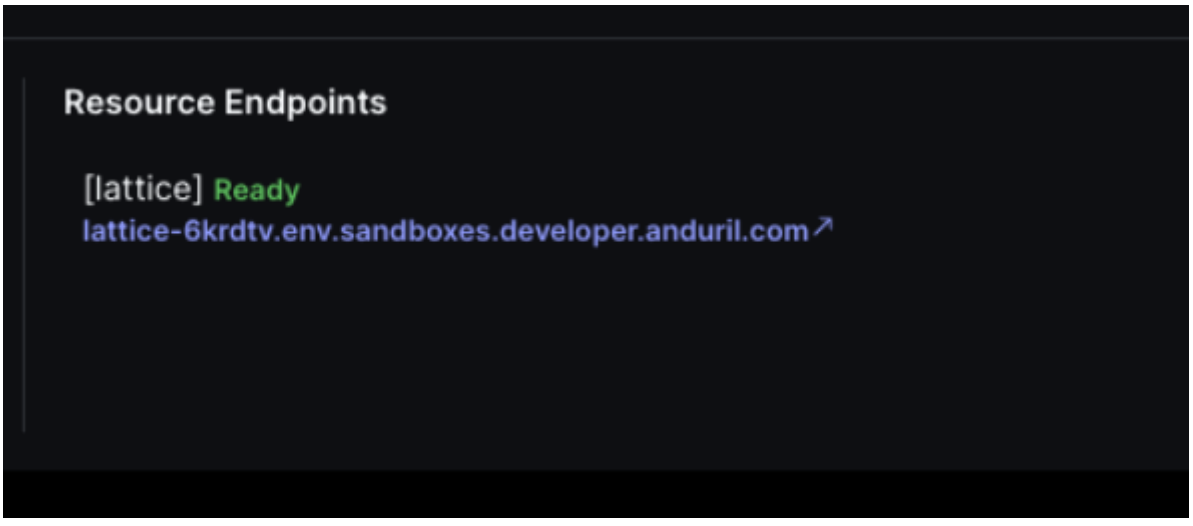
4. Select the **Lattice Entities** dataset and click the **Data Preview** tab to see messages to be published to Lattice

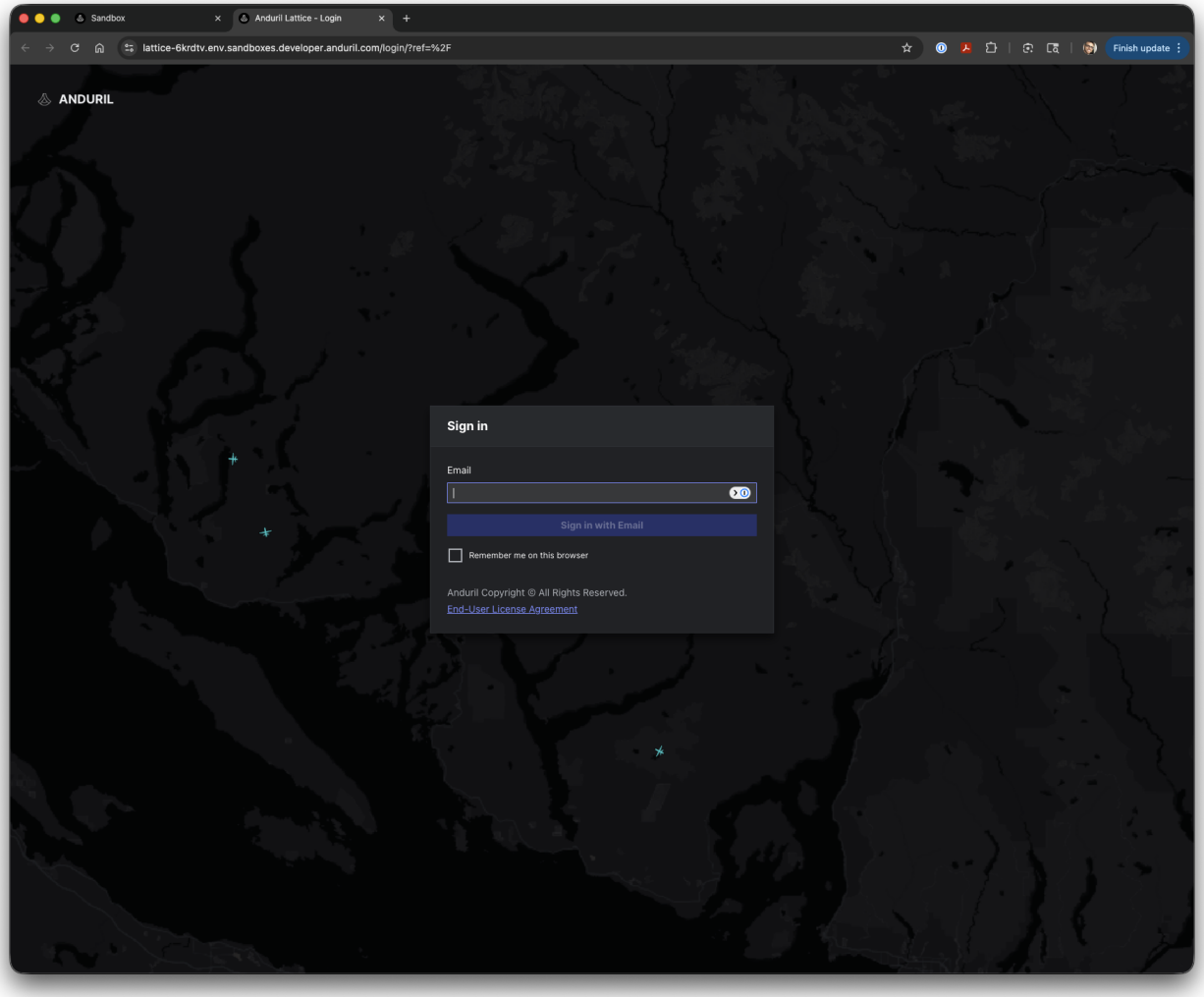


Step 9: View Results in Lattice

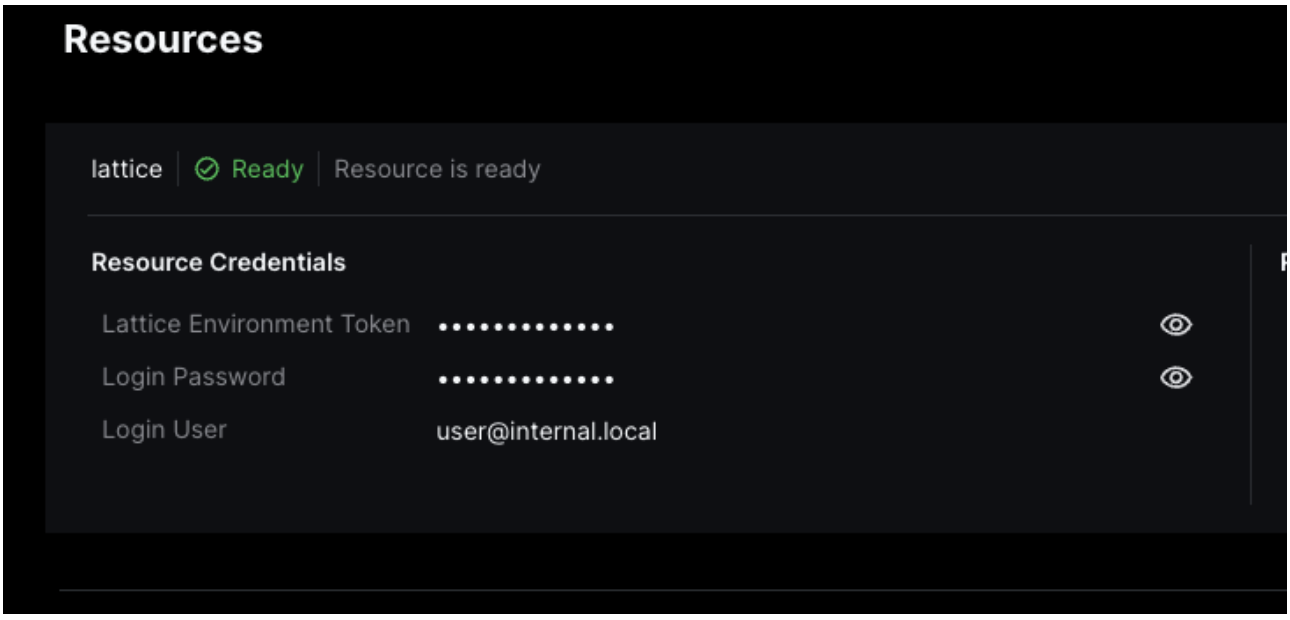
Switch to your Lattice sandbox to see the entities appear in real-time.

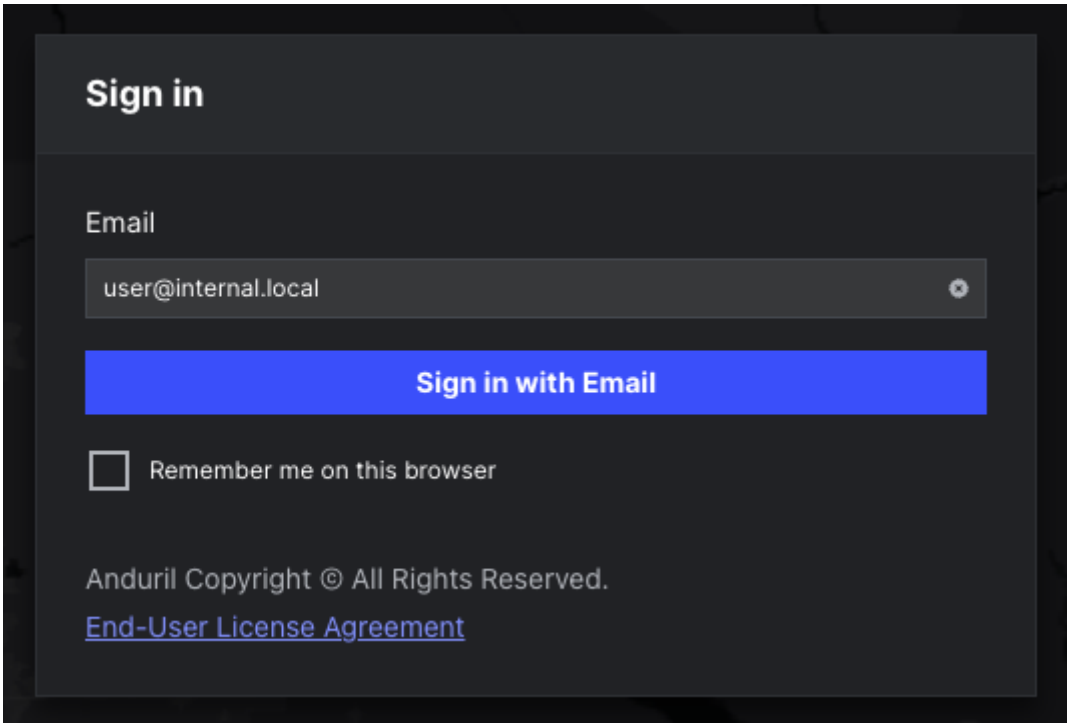
1. Navigate to your Lattice instance in the browser

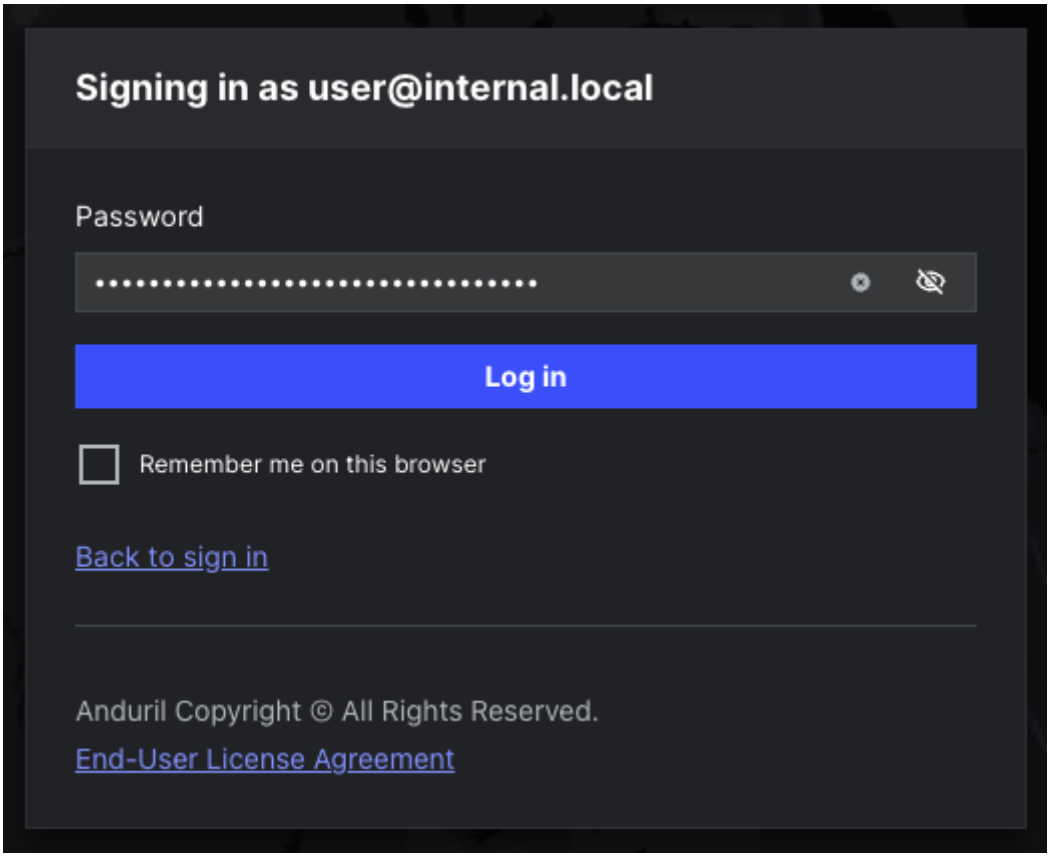




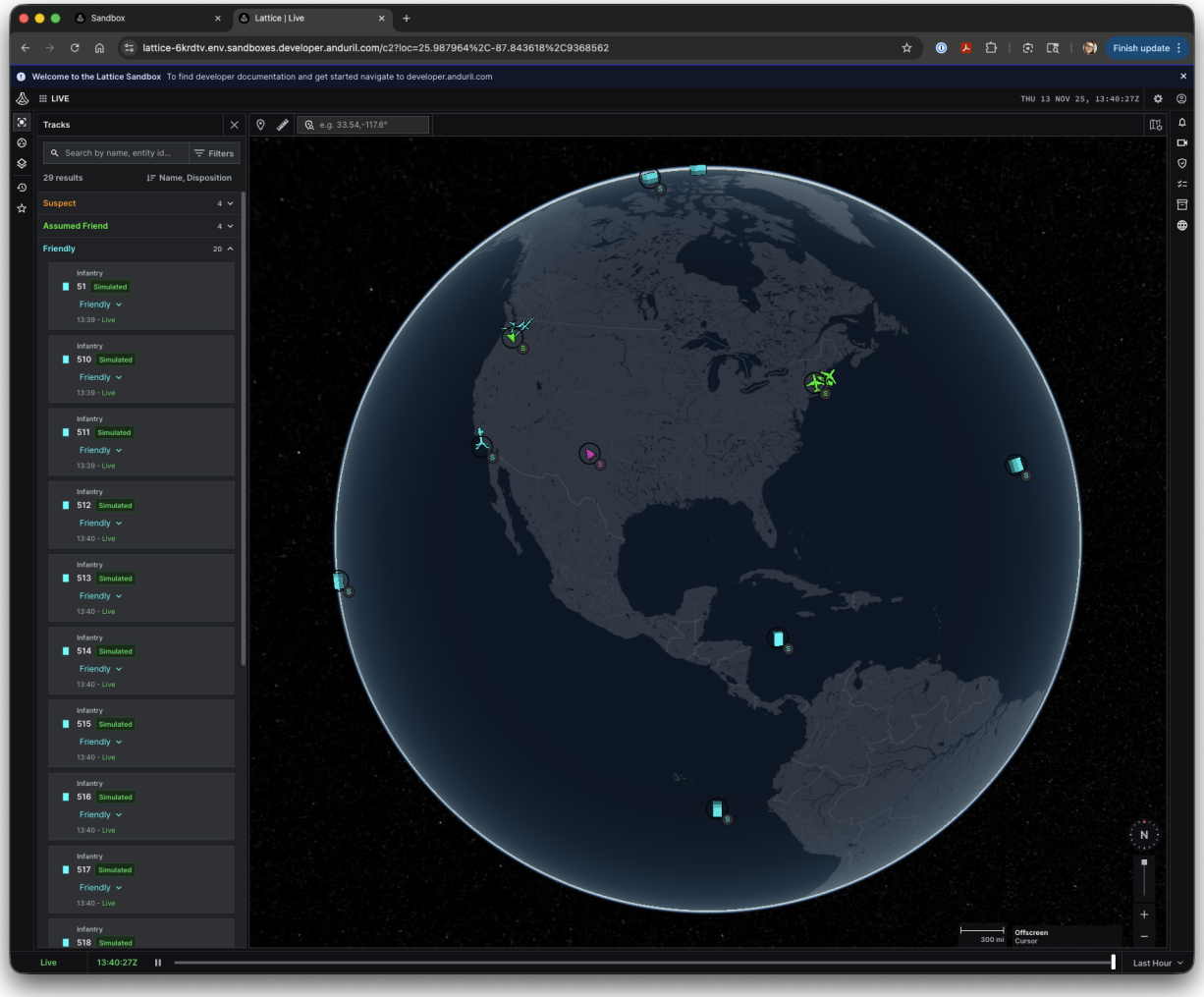
2. Sign in with your sandbox credentials (**Login User** and **Login Password**)



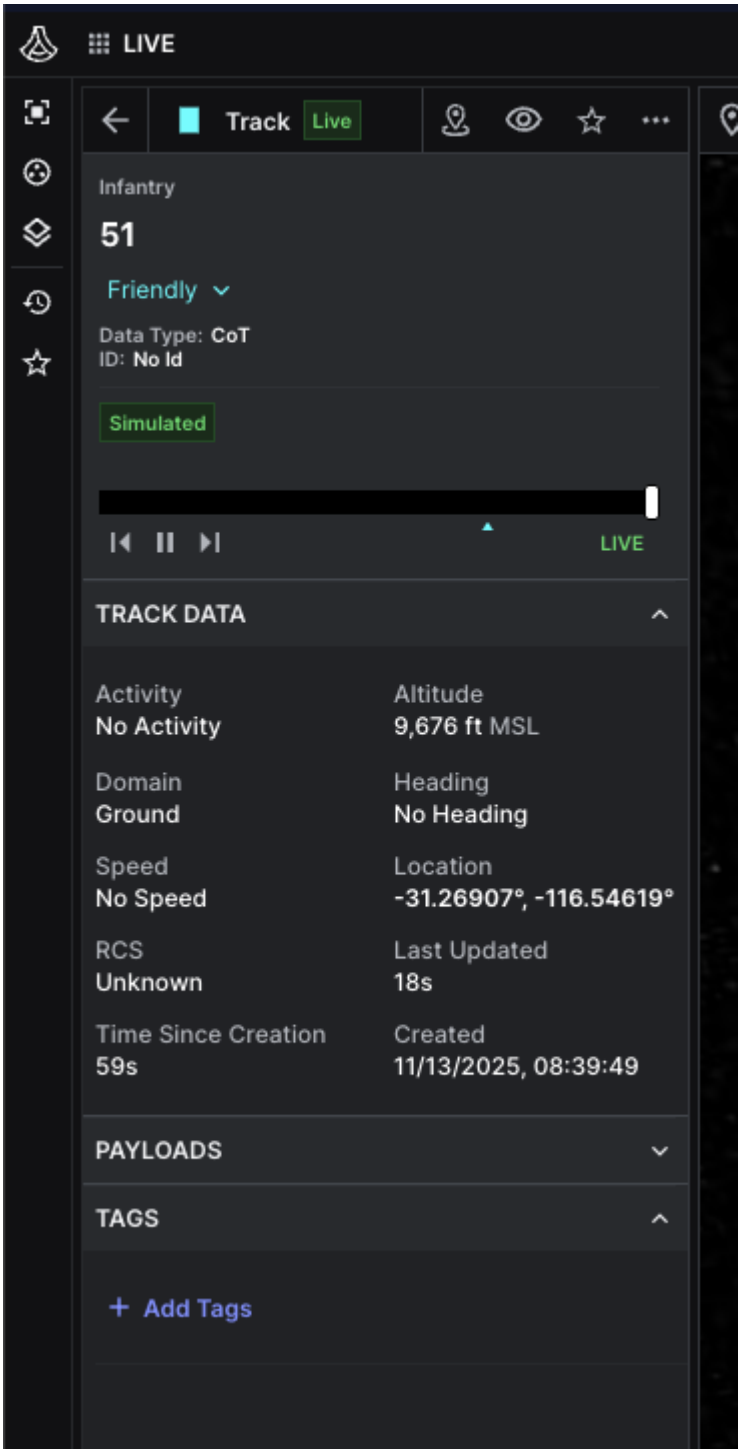




3. The tracks will appear on the map as they are published



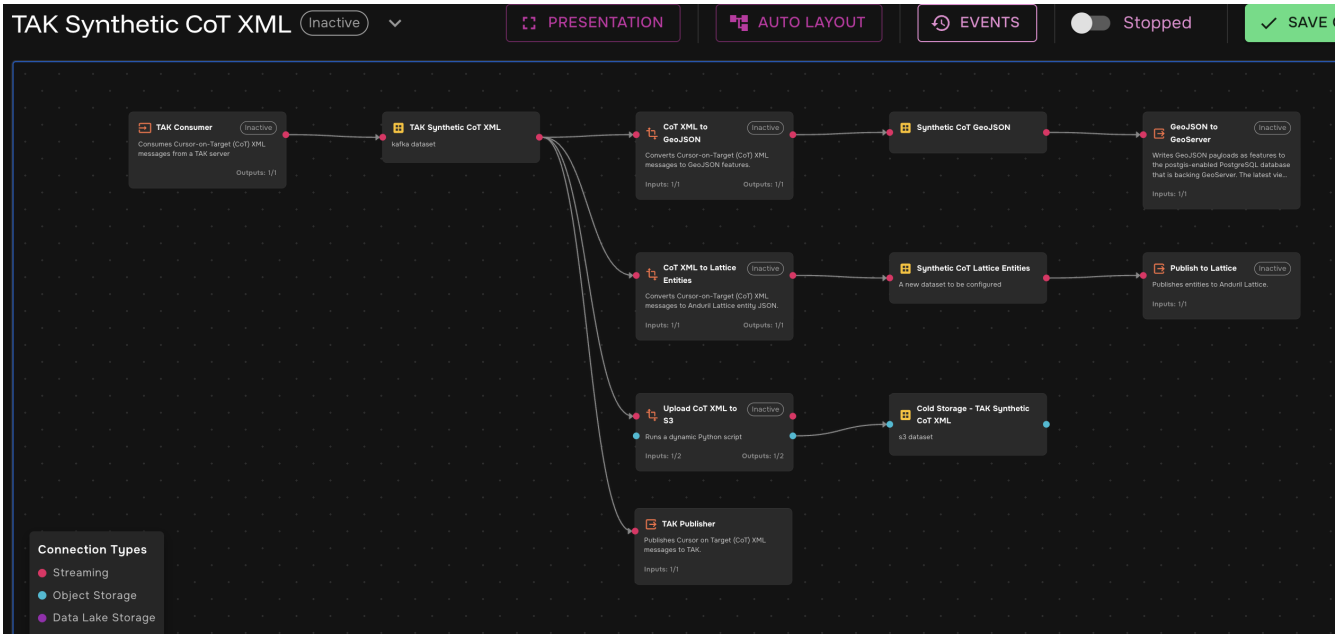
4. The sidebar shows a list of all tracked entities with their details



The CoT XML Generator creates various entity types including fishing vessels, aircraft, and surface vessels. Each entity includes location, velocity, and classification data.

Step 10: Add additional branches and transformers as desired

Stop the pipeline and add any additional branches and transformers as desired. The system supports multiple transformer consumers of a dataset. You can send CoT data to Geoserver with GeoJson transformation, save directly to an S3 bucket for cold storage, or even publish to another TAK Server.



Summary

In this guide, you learned how to:

- ✔ Set up a Lattice sandbox environment for development
- ✔ Build an end-to-end data pipeline in SDL
- ✔ Transform CoT XML messages into Lattice entity format
- ✔ Publish entities to Lattice and visualize them in real-time

Next Steps

Now that you have a working Lattice integration pipeline, you can:

- Connect a real CoT data source instead of the synthetic generator
- Add additional transformation steps to enrich or filter entities
- Create multiple pipelines to publish to different Lattice instances
- Explore other SDL transformers available in the catalog