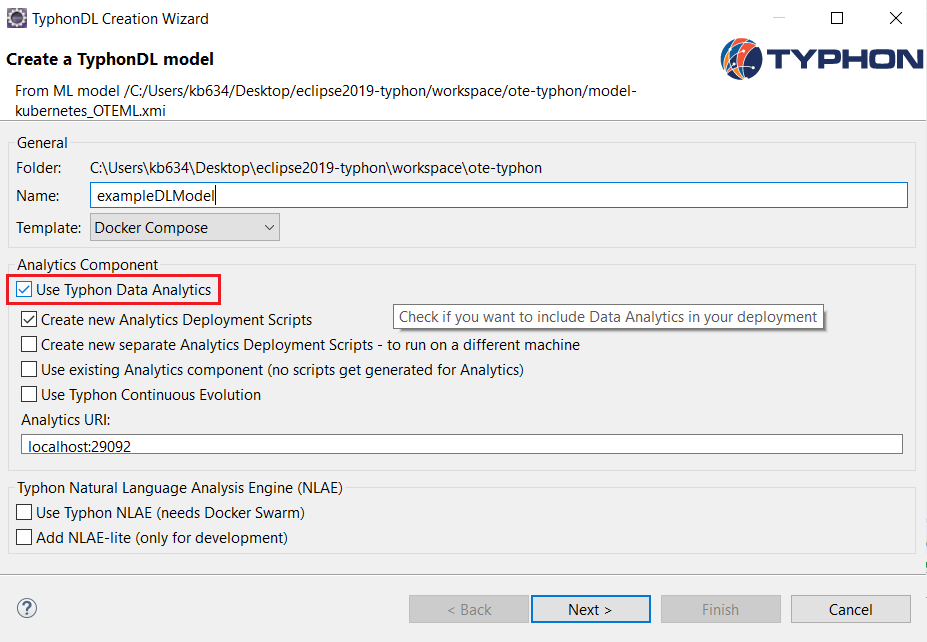
### Analytics

#### Prerequisites

1. This guide assumes that you have already installed all the necessary tools to create and run a Polystore (e.g., TyphonML, TyphonDL, etc.).
2. You need to make sure that you have those updated (from their respective Eclipse update sites and by doing a docker-compose pull) to their latest version.
3. Start by creating the Polystore as described in the previous sections. Make sure that in the appropriate step of the DL wizard you have checked the “Use Typhon Data Analytics” option, as shown in the image below.
4. Run the Polystore.

**Warning**: The guide assumes that the Polystore is deployed using Docker Compose. The Polystore and the analytics component have not yet been fully tested using a Kubernetes deployment.

#### Work with the Analytics Component

1. If working on an existing Eclipse installation (as per Section 1.1.1.1) to go step 3, otherwise download Eclipse from: <https://www.eclipse.org/downloads/packages/release/2020-06/r/eclipse-ide-java-and-dsl-developers>
2. Install the **ML** plugin by selecting Help 🡪 Install New Software 🡪Work with: <http://typhon.clmsuk.com:8082/>and selecting the **Typhon ML** option and clicking next and finish when applicable (restarting eclipse when prompted).
3. Graphical user interface, text, application

   Description automatically generatedSimilarly install Epsilon from: <http://download.eclipse.org/epsilon/updates/2.2/> (as per step 2, selecting **epsilon core** and **epsilon** **emf integration** categories as shown below)
4. Similarly install Typhon Analytics from: <http://typhon.clmsuk.com:8082/> (as per steps 2 or 3, selecting the **typhon analytics** category)
5. In Eclipse create a new java project to host either your analytics or authentication code:
   1. Analytics:
      1. Ensure the Polystore is running
      2. Right click on the project in the project explorer and use the **Typhon** context menu, selecting **Generate analytics code** and enter the URL that the Polystore can be accessed at
      3. Right click on the project again and select Configure 🡪 Convert to Maven Project
      4. You should be ready to edit the created stubs in the **src** folder to add your analytics code (the example class is called AnalyticsStub.java), executing it through the main method of the DefaultAnalyticsRunner.java class
   2. Authentication:
      1. Create a flexmi model to represent your authentication chain, for example a file called *authmodel.flexmi*:

<?nsuri authDSL?>

<chain package="test">

<task name="Task1" next="Task2"/>

<task name="Task2"/>

</chain>

And place it in the project. This example model has an authentication chain made up of two tasks, the first one being Task1 and the second being Task2

* + 1. Right click on this model in the project explorer and use the **Typhon** context menu, selecting **Generate auth analytics code**
    2. Right click on the project again and select Configure 🡪 Convert to Maven Project
    3. You should be ready to edit the created stubs in the **src** folder to add your authentication code, executing it through the main method of the PreEventAuthorizer.java class

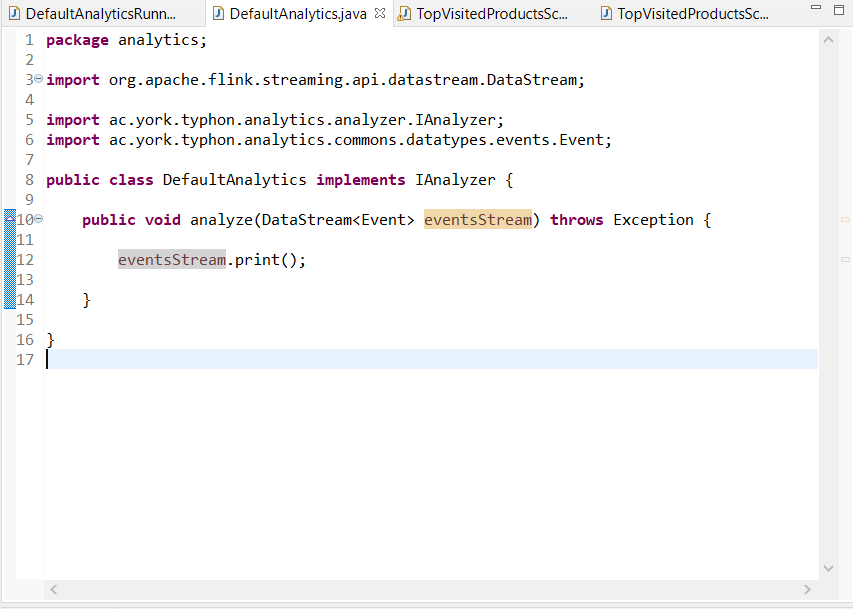
**Please note:** The generated docker compose defines port 29092 for external (outside Docker) access to the Kafka queue and port 9092 for internal (inside Docker) access. As this guide describes how to write analytics in your local IDE, the configuration is set to access port 29092. If you want to export the jar and run it inside Docker, then you need to open the “resources/typhonAnalyticsConfig.properties” file and set the port in line 12 to 9092.

#### Writing Analytics Code with Flink

**IMPORTANT!!!** Post events in Typhon are created every time a TyphonQL query is executed. Thus, your code will produce results, if and only if you start using the Polystore and execute some TyphonQL queries.

Flink is a distributed execution framework. By using its **operators** Flink can easily distribute you code without requiring user’s input/configuration. The goal of this guide is not to train people on writing Flink code. There are plenty of resources on this online. The basic idea is that Flink works with streams (in the context of the analytics component). Steams as the name suggests, provide continuous real-time input to your programs. In the analytics component, the stream of events is the “eventsStream” parameter is the analyse method. This is configured to automatically consume all the events coming to the POST queue.

To consume streams using Flink, one should use Flink Operators. A comprehensive list of all the available Flink operators is available here: <https://ci.apache.org/projects/flink/flink-docs-stable/dev/stream/operators/>. This should be the starting point of anyone trying to use Flink as they contain a brief description and an example of how to make them work. You will find yourself mostly having to use the filter and map function (the first filters events based on a condition, the second is used to transform objects to other forms). Experiment with these 2 first and then you can proceed to more complicated operators.

Below is a simple example using an ecommerce model, that consumes Typhon PostEvents, printing them to console. The deployment folder of this example can be found at <https://github.com/typhon-project/analytics/tree/master/ac.york.typhon.analytics.examples.ecommerce_tutorial>and can be used to run a polystore with this example. The analytics folder of this example can be found at <https://github.com/typhon-project/analytics/tree/master/ac.york.typhon.analytics.examples.ecommerce_tutorial_analytics> and has two runners: a simple one which will just pretty print any query sent to the polystore and one that calculates the top products over specific time windows (this is part of the models2020 tutorial found at <https://github.com/typhon-project/MoDELS-Tutorial> containing a detailed description of this scenario and a generator of queries against a running polystore using this model (<https://github.com/typhon-project/MoDELS-Tutorial/blob/master/Analytics/ac.york.typhon.analytics.models2020.generator/src/ac/york/typhon/analytics/models2020/generator/GeneratorRunner.java>)).

To execute this simple flink program on your local machine, you can run the main method in the DefaultAnalyticsRunner.java class, and similarly you can instead run the TopVisitedProductsScenarioRunner.java class for the top products one.

#### Write Authorisation Tasks

The new class will include two methods you need to implement called “checkCondition (Event event)” and “shouldIReject(Event event).” In the first you need to provide the logic that decides if this authorisation task is responsible for checking the event passed as parameter. For example, if this task is responsible for checking the validity of a review, then the logic included here should filter and accept only such “insert Review” events. The second, should include the rejection logic. For example, include the logic that rejects reviews that are too short (less than 10 characters long).

The auth folder of this example can be found at <https://github.com/typhon-project/analytics/tree/master/ac.york.typhon.analytics.examples.ecommerce_tutorial_authorisation>

To run the authorisation chain on your local machine, you execute the main method in the generated PreEventAuthorizer.java class found in your auth project.

**IMPORTANT!!!**

1. Events in Typhon are created every time a TyphonQL query is executed. Thus, your code will produce results, if and only if you start using the Polystore and execute some TyphonQL queries.
2. The current Polystore implementation provides a default authoriser that authorises all the queries for execution. This means that if you need to include authorisation tasks as part of one of your use case scenarios then you should **not** run the container that authorises all the queries. This can be done by removing (or commenting out) the following service from your docker-compose file

A picture containing knife

Description automatically generated