ID2221 : Data-Intensive Computing

Lab 3 - Building Data Stream Applications with Flink

What you will learn

This session is about *data engineering* and the goal is to offer a glimpse of data-centric application building in real life. The lab will have three parts:

1. **Software installation** of all the necessary development tools.
2. A **Tutorial** for stream processing. During the live lab session we will try to get some practical experience by using Apache Flink while thinking together as a team with **mob programming**, an exciting new way to develop.
3. **Homework** : The final problem that you have to solve and report will combine everything you have learned and let you give some good tips to airports.

Part 1 - Software Installation and Setup

"The most fun part of data engineering is setting-up your programming environment"

- said no one.ever

Notebooks are perfect, easy and simple to use but they are only useful for data exploration. Building complex data-driven applications is typically done more conveniently using a powerful IDE, git, maven, sbt etc. That is why we are going to guide you through some basic steps to install, develop and run our applications using Apache Flink. You can find the full original instructions at the following page, located at the official Flink training web site, however, in this document we are highlighting the most important parts for your convenience:

I- Install Software Dependencies

Flink supports Linux, OS X, and Windows for developing and executing programs locally. Before you proceed make sure you install each of these dependencies:

- Oracle Java JDK 7 or 8 - Java Development and Execution Engine (JVM)
- Apache Maven 3.x - Tool for Building and Packaging projects
- Git - Tool for Software Versioning and Sharing
- IntelliJ IDEA (community edition) - The development environment (instructions)

II- Generate a Flink project with **maven**

From this point everything is simple. Nothing is needed to install. Flink provides Maven archetypes which set-up automatically a Maven project for developing Java or Scala Flink programs right away. First go to your default development directory via your command line/terminal and run one of following maven commands to generate your project (**Windows Users :** remove the backslashes from the maven commands above.):

For **Java**:

```
mvn archetype:generate                               \
    -DarchetypeGroupId=org.apache.flink              \
    -DarchetypeArtifactId=flink-quickstart-java      \
    -DarchetypeVersion=1.3.2                         \
    -DgroupId=org.apache.flink.quickstart            \
    -DartifactId=flink-java-project                  \
    -Dversion=0.1                                    \
    -Dpackage=org.apache.flink.quickstart            \
    -DinteractiveMode=false
```

and for **Scala**:

```
mvn archetype:generate                               \
    -DarchetypeGroupId=org.apache.flink              \
    -DarchetypeArtifactId=flink-quickstart-scala     \
    -DarchetypeVersion=1.3.2                         \
    -DgroupId=org.apache.flink.quickstart            \
    -DartifactId=flink-scala-project                 \
    -Dversion=0.1                                    \
    -Dpackage=org.apache.flink.quickstart            \
    -DinteractiveMode=false
```

If everything worked fine you will get a new directory (`flink-java-project` for Java or `flink-scala-project` for Scala).

III- Clone and build the flink-training-exercises project using **git**

The `flink-training-exercises` project contains utility classes and reference solutions for the practice exercises (but not for your final task). Go back to your base development directory, clone the `flink-training-exercises` project from Github and build it as follows:

```
git clone https://github.com/dataArtisans/flink-training-exercises.git
cd flink-training-exercises
mvn clean install
```

IV- Now go back to **your project** and update your dependencies

We somehow need to make our project use the libraries of flink-training-exercises. To add a dependency, go inside your project (e.g., `flink-java-project` or `flink-scala-project`), open the `pom.xml` file with a text editor and add the following dependency after the other flink dependencies:

```
<dependency>
  <groupId>com.data-artisans</groupId>
  <artifactId>flink-training-exercises</artifactId>
  <version>0.11.0</version>
</dependency>
```

V- Build your Flink quickstart project

In order to test the generated project and to download all required dependencies run the following command in the `flink-java-project` (`flink-scala-project` for Scala projects) folder.

```
mvn clean package
```

Maven will download all required dependencies and build the Flink quickstart project for you.

VI- Import the Flink Maven project in Intellij

It is time to import your project so that you can work in Intellij as follows:

1. Select *“File”* -> *“Import Project”*
2. Select the root folder of your project (*flink-java-project* or *flink-scala-project*)
3. Select *“Import project from external model”*, select *“Maven”*
4. Continue, making sure the SDK dialog has a valid path to a JDK and leaving all other options to their default values, and finish the import

VII- Get the data events

We are going to use real-time events gathered from taxi rides around NYC from 2009 to 2015. Each taxi ride is represented by two events, a **trip start** and an **trip end** event. Each event consist of nine fields.

```
rideId          : Long    // a unique id for each ride
isStart         : Boolean // TRUE for ride start events, FALSE for ride end events
startTime       : String  // the start time of a ride
endTime         : String  // the end time of a ride,
startLon        : Float   // the longitude of the ride start location
startLat        : Float   // the latitude of the ride start location
endLon          : Float   // the longitude of the ride end location
endLat          : Float   // the latitude of the ride end location
passengerCnt    : Short   // number of passengers on the ride
```

You can download the taxi data preferably in your project directory by running the following command:

```
wget http://training.data-artisans.com/trainingData/nycTaxiRides.gz
```

Please **do not decompress or rename** the .gz file.

You can read more about the input file and loading taxi ride streams here

VIII- Try doing a ‘test drive’

Flink programs can be executed and debugged from within an IDE. This significantly eases the development process and gives a programming experience similar to working on a regular Java application. Starting a Flink program in your IDE is as easy as starting its `main()` method. Under the hood, the `ExecutionEnvironment` will start a local Flink instance within the execution process. Hence it is also possible to put breakpoints everywhere in your code and debug it. Assuming you have an IDE with a Flink quickstart project imported, you can execute and debug the example `WordCount` program which is included in the quickstart project as follows:

- Open the `org.apache.flink.quickstart.WordCount` class in your IDE
- Place a breakpoint somewhere in the `flatMap()` method of the `LineSplitter` class which is defined in the `WordCount` class.
- Execute or debug the `main()` method of the `WordCount` class using your IDE.

Part 2 - Flink Training and Examples

Practice Task 1- Taxi Cleansing

The task of the “Taxi Ride Cleansing” exercise is to cleanse a stream of TaxiRide events by removing events that do not start or end in New York City. The `GeoUtils` utility class provides a static method `isInNYC(float lon, float lat)` to check if a location is within the NYC area.The result of the exercise should be a `DataStream<TaxiRide>` that only contains events of taxi rides which start and end in the New York City area as defined by `GeoUtils.isInNYC()`.

Relevant Tutorial *: DataStream API Basics*

Learning Outcome: Program skeleton and basic transformations of DataStream programs.

For Full Description, Code Hints, Documentation and Solutions check here.

Practice Task 2- Find Popular Places

Now we are ready to identify popular places from the taxi ride data stream. This is can be done by counting every five minutes the number of taxi rides that started and ended in the same area within the last 15 minutes. Arrival and departure locations should be separately counted. Only locations with more arrivals or departures than a provided popularity threshold should be forwarded to the result stream. The `GeoUtils` class provides a static method `GeoUtils.mapToGridCell(float lon, float lat)` which maps a location (longitude, latitude) to a cell id that refers to an area of approximately 100x100 meters size. The `GeoUtils` class also provides reverse methods to compute the longitude and latitude of the center of a grid cell. Note that the program should operate in event time. The result of this exercise is a data stream of `Tuple5<Float, Float, Long, Boolean, Integer>` records. Each record contains the longitude and latitude of the location cell (two `Float` values), the timestamp of the count (`Long`), a flag indicating arrival or departure counts (`Boolean`), and the actual count (`Integer`).

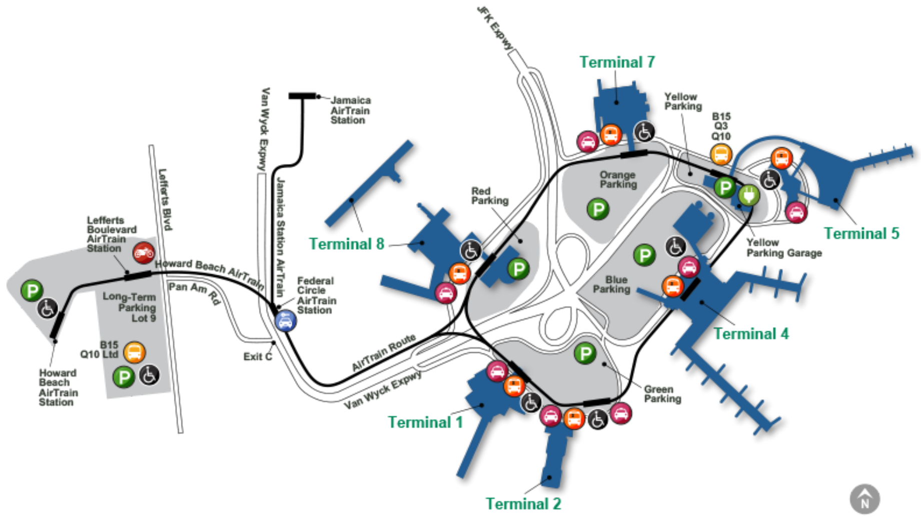
Tutorial : Time and Watermarks

Learning Outcome: Event-time DataStream windows and aggregations.

For Full Description, Code Hints, Documentation and Solutions check here.

Part 3 - The Homework Assignment

Now let's solve a real problem. The New York F.Kennedy Airport has way too many terminals. This makes the life of cab drivers and companies a living nightmare, trying to decide on the terminal to serve based on the demand. In fact, taxi drivers observed that there is no single most popular terminal and demand actually changes every hour of the day.



Your job as a data engineer is to use Flink and all data that you got at your disposal (taxi rides from before) in order to generate a live application that tracks the traffic trends within terminals per hour so the airport can pre-order the appropriate amount of cabs to be ready to serve on each terminal at every single hour of the day.

Homework Task 1

The first step in attacking the problem is to generate a data stream that provides some statistics about each terminal per hour. More specifically, you will have to consider only taxi rides that start or end at an airport terminal and count the number of taxis in that terminal per (event-time) hour. The output of your pipeline should be a **(Terminal, count, hour)** and look somehow like the following:

```
5> (TERMINAL_6,9,4)
5> (TERMINAL_6,1,5)
6> (TERMINAL_4,45,5)
1> (TERMINAL_1,2,5)
2> (TERMINAL_3,17,5)
3> (TERMINAL_2,2,6)
1> (TERMINAL_1,4,6)
2> (TERMINAL_3,7,6)
6> (TERMINAL_4,31,6)
5> (TERMINAL_6,3,6)
2> (TERMINAL_3,26,7)
6> (TERMINAL_4,24,7)
3> (TERMINAL_2,2,7)
5> (TERMINAL_6,9,7)
1> (TERMINAL_1,5,7)
```

To aid you with this task, here is a skeleton with some helper functions you can use to implement your pipeline:

Scala Skeleton
Java Skeleton

Hints: You should filter out rides that are starting or ending in an airport terminal quite early. Also, remember that you can still use `GeoUtils.mapToGridCell(float lon, float lat)` to map coordinates to grid cells. Furthermore, the helper function `gridToTerminal` provided in the skeleton code can convert grid cells to terminals. Finally, you can convert a long timestamp to an hour of the day (Integer) in NYC as follows:

```
Calendar calendar = Calendar.getInstance();
calendar.setTimeZone(TimeZone.getTimeZone("America/New_York"));
calendar.setTimeInMillis(longTimestamp);
calendar.get(Calendar.HOUR_OF_DAY)
```

Homework Task 2

The airport has a single jumbo bus that can pick up people from the most popular terminal only **once per hour** and drive them to New York City center. Your task is to **extend** the previous application in order to generate a stream of the **single most popular** terminal per hour along with the number of rides and the time of the day as follows:

```
6> (TERMINAL_3,43,9)
7> (TERMINAL_3,59,10)
8> (TERMINAL_4,51,11)
```

```
1> (TERMINAL_3,59,12)
2> (TERMINAL_3,57,13)
3> (TERMINAL_3,71,14)
4> (TERMINAL_3,38,15)
5> (TERMINAL_3,54,16)
6> (TERMINAL_3,51,17)
7> (TERMINAL_3,58,18)
8> (TERMINAL_4,45,19)
1> (TERMINAL_3,45,20)
2> (TERMINAL_3,16,21)
3> (TERMINAL_3,5,22)
4> (TERMINAL_3,3,23)
5> (TERMINAL_3,4,0)
6> (TERMINAL_4,52,1)
7> (TERMINAL_3,48,2)
```

Hint: You need to run a global/non-parallel window (timeWindowAll) to include counts across terminals. Then it will be easy to pick the terminal with the maximum count.

Final Report

In your final report copy only your solution code (should NOT be much) and explain with a few words what each data stream transformation in your code is doing. Also try to briefly explain how windows are actually formed (watermarks?) and motivate all your choices.

Happy Coding!