Data Stream Processing and Analytics **Prof. Vasiliki Kalavri**
Spring Semester 2020 **vkalavri@bu.edu**

Assignment 1

Submission deadline: February 12 2020, 11:59 pm

The rest of this assignment assumes a UNIX-based setup. If you are a Windows user, you are advised to use Windows subsystem for Linux (WSL), Cygwin, or a Linux virtual machine to run Flink in a UNIX environment.

Before you continue, make sure you have successfully completed the Apache Flink local cluster setup tutorial and steps 1-6 of the Apache Kafka Quickstart.

1. Download and build the `clusterdata-analysis` project

The `clusterdata-analysis` project contains utility classes, template applications, tests, and examples for this and future programming assignments. Download the project from Blackboard, extract it, and build it:

```
cd clusterdata-analysis
mvn clean package
```

The first time you build the project, Maven will download all required dependencies. This might take a few minutes. If all tests pass and the build is successful, you are good to go!

2. Import the project into your IDE

Open IntelliJ and select File -> New -> Project from Existing Sources… , navigate to the location where you extracted the project and select the pom.xml file (Fig. 1). Once the import process has finished, you should be able to see the project structure on the left side of your panel (Fig. 2). You will find auxiliary code under the `datatypes`, `sources`, and `utils` packages, as well as template application code in the `exercise1` package. `clusterdata.exercise1.JobEventCount` is a complete example that continuously counts Job events.

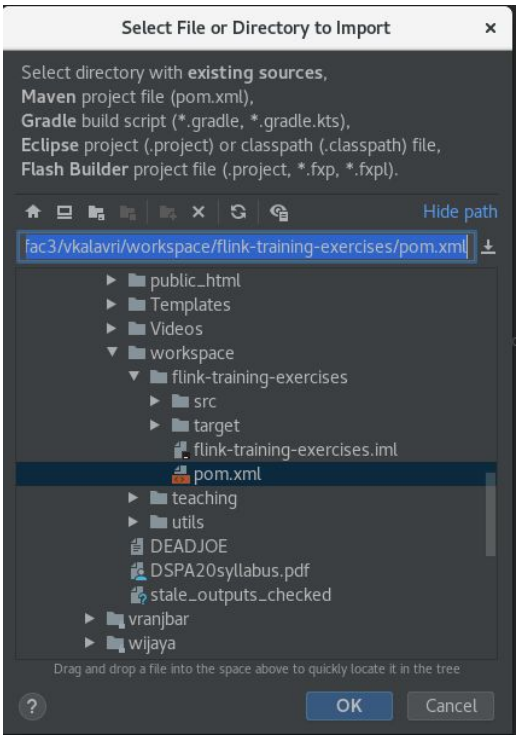

Fig.1: Import the project in IntelliJ by selecting the pom.xml file.

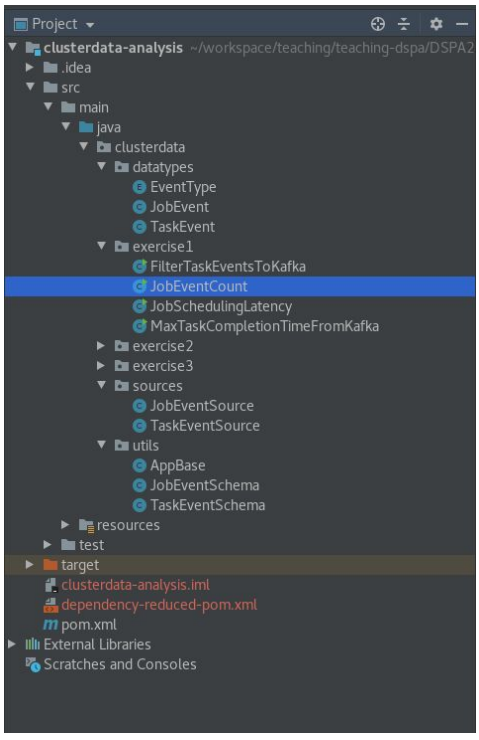

Fig 2: The Project structure.

Inside the folder `sources`, you will find the implementations of two source operators, `JobEventSource` and `TaskEventSource`. These source functions read timestamped records from gzipped files and simulate realistic streams by serving events based on their actual time of occurrence. The serving speed of the sources can be adjusted using the serving speed factor parameter. For instance, a factor of 60.0 increases the logical serving time by a factor of 60, so that events occurring within one minute (60 seconds) will be sent to the stream processor in 1 second.

3. Download the dataset and setup the input paths

For this and future assignments, we will be using a subset of traces from a large Google cluster of 12.5k machines. Make sure to carefully read the data specification and download instructions as well as the format and schema document.

Choose a location on your machine and create a folder named `clusterdata-2011-2` with a subfolder `job_events` and a subfolder `task_events`.

The entire dataset is very large and will take too long to download. Instead, for the purpose of this assignment, we will use parts 0, 1, and 2 of the Job and Task Events tables. To download the first part of job events (part 0), execute the following command, after specifying the path to your `clusterdata-2011-2/job_events/` folder:

```
gsutil cp -R gs://clusterdata-2011-2/job_events/part-00000-of-00500.csv.gz /your/path/
```

Do not extract the .gz files!

After you have downloaded parts 0 and 1 of the Job events and Task events tables, open the `clusterdata.utils.AppBase` class and point the variables `pathToJobEventData` and `pathToTaskEventData` to the appropriate locations.

4. Run the `JobEventCount` example

In IntelliJ, open the `clusterdata.exercise1.JobEventCount` class and inspect the code. Run it as a Java application and watch the output in the IntelliJ console. You should see a stream of (jobId, count) pairs where the counts are continuously updated as more and more job events are processed.

5. Tasks

The assignment consists of the following four tasks. Each task's contribution to the assignment grade is indicated inside parentheses.

~~a. Implement the TaskEvent class (10/100)~~

~~Inspect the code in the `JobEvent` class and use it as an example to implement the `TaskEvent` class. Consult the dataset format and schema document for a description of the various record fields.~~

a. Implement the TaskEventSchema class (10/100)

The `TaskEventSchema` class defines how `TaskEvent` objects can be serialized and deserialized in order to exchange them between Flink and Kafka. Implement the `serialize()` method that receives a `TaskEvent` and transforms it into a byte array and the `deserialize()` method which receives a byte array and turns it into a `TaskEvent` object.

b. Calculate Job scheduling latency (30/100)

Write a Flink program that measures the time between submitting and scheduling each job in the cluster. For every jobID, record the timestamp of the SUBMIT event and monitor the stream for a subsequent SCHEDULE event. Once received, output the jobId and the job's duration to the standard output. If a job is submitted multiple times, then measure the latency since the first submission. Use the template code in `clusterdata.exercise1.JobSchedulingLatency`.

c. Filter TaskEvents and write them to a Kafka topic (20/100)

Implement a Flink program that reads task events from a `TaskEventSource`, filters out any events that do not correspond to SUBMIT or FINISH transitions, and writes the resulting stream to a Kafka topic. Use the template code in `clusterdata.exercise1.FilterTaskEventsToKafka`. Create the Kafka topic first, then launch your Flink job, and inspect the topic contents with the console consumer.

Hint #1: You can use the `printOrTest()` method to ensure you are ingesting TaskEvents correctly before writing them to Kafka.

Hint #2: In order to write events to a Kafka topic, you will need to implement a serialization method that instructs Flink how to convert `TaskEvent` objects to Kafka messages, i.e. byte arrays. A `TaskEventSchema` class is already provided. You will need to implement the `serialize()` method.

d. Calculate the maximum task duration per priority (40/100)

Implement a Flink program that reads the filtered TaskEvents back from Kafka and computes the maximum task duration per priority. Write a transformation that monitors the event stream for SUBMIT and FINISH events per task, computes the task duration once both events have been received, and outputs a tuple-2 of (priority, duration). After grouping the resulting stream by priority, write another transformation that emits to the standard output the maximum task duration seen so far for each priority.
Use the code in `clusterdata.exercise1.MaxTaskCompletionTimeFromKafka`.

Hint #1: In order to read events from Kafka, you will need to implement a deserialization method that instructs Flink how to convert Kafka messages (byte arrays) to TaskEvent objects. You will need to implement the `deserialize()` method in the `TaskEventSchema` class.

Hint #2: If you need to use a composite key, you can group events by more than one fields, by adding as many arguments as necessary to the `keyBy()` method of the DataStream API. Consult the "Specifying Keys" section in the Flink documentation.

6. Tests

A testing infrastructure is already in place in the template project provided to you. If you want to write more tests, you can use the `JobEventCountTest` class as an example. For the test base classes to work properly, you will need to use the `printOrTest()` method as the dataflow sink.

7. Deliverables

To submit your solutions to this assignment, **upload the `src` folder of your clusterdata-analysis project to Blackboard**. If you have edited the `pom.xml` file in any way, make sure to provide the updated file as well. Note that **well-documented code is always easier to understand and grade**, so please make sure your code is clean, readable, and has comments.

8. Resources

The Apache Flink v1.9 documentation and Apache Kafka v2.4 documentation pages contain all the information you can possibly need to complete this assignment. Spend some time to become familiar with the different sections and read the provided example code snippets. You will find the following sections particularly useful:

- Basic API concepts
- Flink DataStream API Programming Guide
- Operators
- Apache Kafka Connector
- Configuration