Stream Processing Essentials - Lab Guide

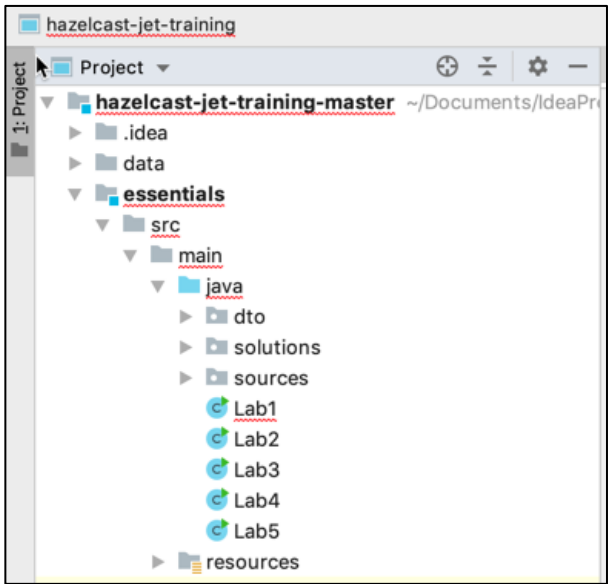
Lab Requirements

- Java 8 JDK (minimum)
- An IDE like IntelliJ IDEA, Eclipse, Netbeans
- A build system like Maven, Gradle, sbt

Steps

1. Download the lab source: https://github.com/hazelcast/hazelcast-jet-training
2. Import the labs into your IDE

What You Should See…



The actual display will depend on your IDE. In the java folder, you should see 5 files – Lab1, Lab2, and so on. You'll also see another folder labeled "solutions" that contains an example solution for each lab. You should only look at the solutions if you get stuck working on the lab exercises.

Warning – Lambdas Ahead!

Coding for stream processing relies on the use of Java lambdas. If you are not familiar with this style of programming, you will find these labs very difficult.

Lab 1: Filter Records from Stream

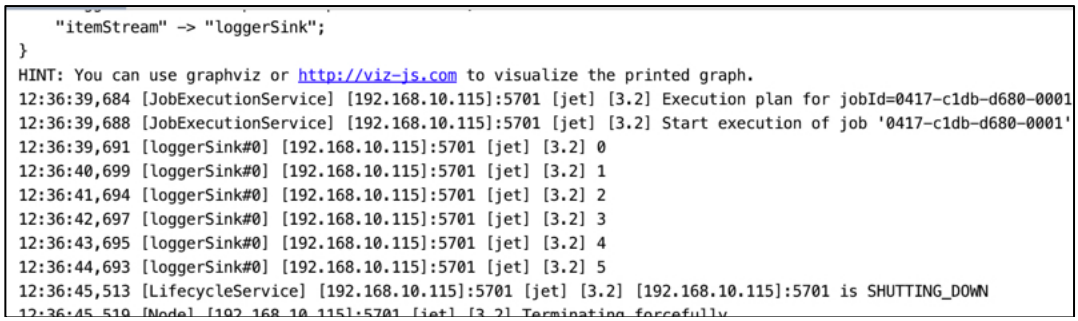
Objectives:

1. Create pipeline that prints a stream to the console.
2. Create a filter that only prints even numbers from the stream.
3. Create pipeline that processes data from a file.

```
private static Pipeline buildPipeline() {
    Pipeline p = Pipeline.create();

    // STEP 1: Print a stream to the console

    StreamSource<Long> source = TestSources.itemStream( itemsPerSecond: 1, (ts, seq) -> seq);

    p.drawFrom(source)
     .withoutTimestamps()
     .drainTo(Sinks.logger());

    // Run the code to see the results in the console
    // Stop it before continuing to step 2



    // STEP 2: Filter out odd numbers from the stream

    // Add filter() to  your pipeline
    // - Use lambda to define the predicate



    // STEP 3: Process data from a file instead of generated data

    // Create a directory somewhere in your computer and create an empty input.txt file in it

    // Replace itemStream with fileWatcher source from com.hazelcast.jet.pipeline.Sources
    // - (fileWatcher stream lines added to files in a directory.)
    // - Adjust source type - the generator was producing Longs, fileWatcher produces Strings

    // Add a mapping step before the filter to convert the stream from Strings to Longs

    // Run this pipeline to test it!
    // - Add text lines to the file.
    // - Use echo -- some text editors create a new file for every save. That results in replaying the file.
    //
    // echo "0" >> input.txt
    // echo "1" >> input.txt

    // Stop the job
```

Step 1: Create a pipeline that prints a stream to the console

Run the code as provided in the Lab 1 file. You should observe the following output in your console screen.

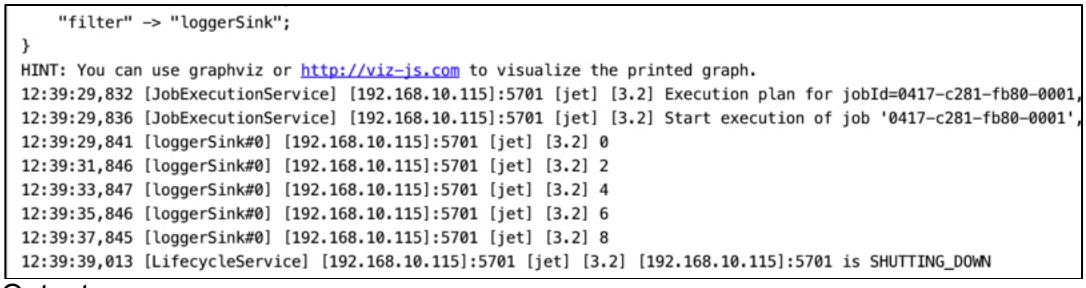


Output

Part 2: Add a filter

Add a filter so that odd numbers are not printed to the console.

Test your filter. You should observe the following output on your console screen.



Output

Part 3: Read from a file in real time

Open a terminal window. Use `touch` to create an empty file.

In your code, set the DIRECTORY variable to the location of your file.

Change the source to use FileWatcher. FileWatcher monitors a directory and captures changes to any files in the directory as input.

We want our output to be `Long`, but FileWatcher input is `String`. We need to use a map to convert the `String` to `Long` before outputting to the console.

Run your code.

To test it, go back to your terminal window. Use `echo "some number" >> filename.txt` to add lines to the data file. You should see all your input being echoed to the console.

```
Hendys-MacBook-Pro:hazelcast-jet-training hendyappleton$ cd data
Hendys-MacBook-Pro:data hendyappleton$ echo "0" >> input.txt
Hendys-MacBook-Pro:data hendyappleton$ echo "1" >> input.txt
Hendys-MacBook-Pro:data hendyappleton$ echo "2" >> input.txt
Hendys-MacBook-Pro:data hendyappleton$ echo "3" >> input.txt
Hendys-MacBook-Pro:data hendyappleton$ echo "62" >> input.txt
Hendys-MacBook-Pro:data hendyappleton$ echo "99" >> input.txt
```

Input

```
}
HINT: You can use graphviz or http://viz-js.com to visualize the printed graph.
12:42:52,797 [JobExecutionService] [192.168.10.115]:5701 [jet] [3.2] Execution plan for jobId=0417-c348-32c0-0001
12:42:52,801 [JobExecutionService] [192.168.10.115]:5701 [jet] [3.2] Start execution of job '0417-c348-32c0-0001'
12:42:52,810 [fileWatcherSource(data//*)#1] [192.168.10.115]:5701 [jet] [3.2] Started to watch directory: data
12:42:52,810 [fileWatcherSource(data//*)#0] [192.168.10.115]:5701 [jet] [3.2] Started to watch directory: data
12:43:28,820 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.2] 0
12:43:36,814 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.2] 1
12:43:44,818 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.2] 2
12:44:00,814 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.2] 3
12:44:12,812 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.2] 62
12:44:22,815 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.2] 99
```

Output

Extra credit

Add a filter to Part 3 so that only numbers divisible by 3 are copied to the console. The sample output below was generated by the same input above.

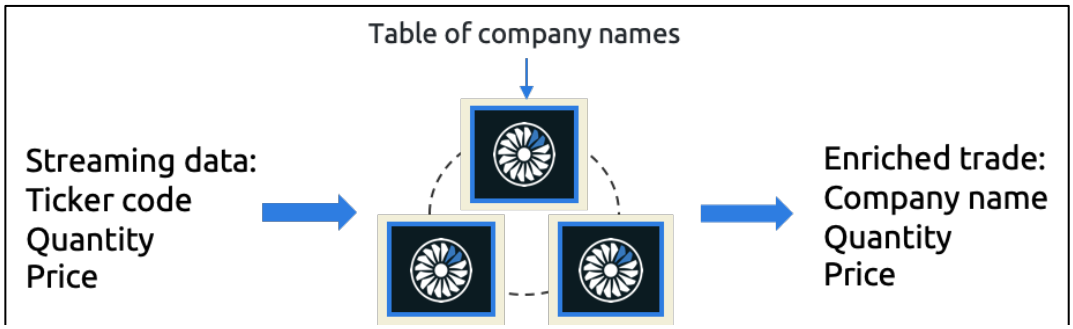
```
HINT: You can use graphviz or http://viz-js.com to visualize the printed graph.
12:45:50,591 [JobExecutionService] [192.168.10.115]:5701 [jet] [3.2] Execution plan for jobId=0417-c3f5-d500-000
12:45:50,594 [JobExecutionService] [192.168.10.115]:5701 [jet] [3.2] Start execution of job '0417-c3f5-d500-0001
12:45:50,603 [fileWatcherSource(data//*)#1] [192.168.10.115]:5701 [jet] [3.2] Started to watch directory: data
12:45:50,603 [fileWatcherSource(data//*)#0] [192.168.10.115]:5701 [jet] [3.2] Started to watch directory: data
12:46:00,613 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.2] 0
12:46:04,606 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.2] 3
12:46:08,607 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.2] 99
```

Output

Lab 2: Enriching the Stream

Objectives:

1. Copy the contents of the trade stream to the log.
2. Enrich the streaming data with the company name information.



Before you begin

Make sure you have no running instances of Jet on your local device.

Part 1: Viewing raw trade data

```
private static Pipeline buildPipeline(IMap<String, String> lookupTable) {
    Pipeline p = Pipeline.create();

    // 1 - Read from the Trade Source (sources.TradeSource) - it's custom source generating Trades (dto.Trade)

    // 2 - With native timestamps

    // 3 - Convert Trade stream to Enriched Trade stream
    // - Trade (dto.Trade) has a symbol field
    // - Use LOOKUP_TABLE to look up full company name based on the symbol
    // - Create new Enriched Trade (dto.EnrichedTrade) using Trade and company name

    // 4 - Drain to sink
```

Open the Lab 2 code skeleton, and scroll down to the pipeline section. Create a pipeline that reads from the trade generator sources.TradeSource using the native timestamp data. Drain the data to the logger sink.

Run your code to verify that you are receiving and logging trade data. Once you've verified this, stop your program.

```
15:20:04,912 [JobExecutionService] [192.168.10.115]:5701 [jet] [3.1] Execution plan for jobId=a3ed-8bd0-7085-ed16, jobName='a3ed-8bd0-7085-
15:20:04,918 [JobExecutionService] [192.168.10.115]:5701 [jet] [3.1] Start execution of job 'a3ed-8bd0-7085-ed16', execution f74e-f1dd-d164
15:20:05,939 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564006804927, symbol='AAPL', quantity=100, price=4854}
15:20:06,938 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564006805933, symbol='MSFT', quantity=100, price=1292}
15:20:07,943 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564006806934, symbol='AAPL', quantity=100, price=1980}
15:20:08,946 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564006807938, symbol='AAPL', quantity=100, price=1574}
15:20:09,947 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564006808940, symbol='MSFT', quantity=100, price=4682}
15:20:10,953 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564006809944, symbol='MSFT', quantity=100, price=242}
15:20:11,957 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564006810948, symbol='MSFT', quantity=100, price=2619}
15:20:12,962 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564006811953, symbol='MSFT', quantity=100, price=1865}
15:20:13,963 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564006812956, symbol='GOOGL', quantity=100, price=2779}
15:20:14,963 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564006813957, symbol='AAPL', quantity=100, price=323}
15:20:15,966 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564006814962, symbol='MSFT', quantity=100, price=2887}
```

Output

Part 2: Enriching the trade data

Let's look at the top of the code skeleton.

```
private static final String LOOKUP_TABLE = "lookup-table" ;

public static void main (String[] args) {
    Pipeline p = buildPipeline();

    JetInstance jet = Jet.newJetInstance();

    // symbol -> company name
    IMap<String, String> lookupTable = jet.getMap(LOOKUP_TABLE);
    lookupTable.put( k: "AAPL",  v: "Apple Inc. - Common Stock");
    lookupTable.put( k: "GOOGL",  v: "Alphabet Inc.");
    lookupTable.put( k: "MSFT",  v: "Microsoft Corporation");
```

You can see where we've created a local IMap called lookupTable that contains the company names. In a "real world" application, you would likely use a MapLoader function to populate the local cache from an external data source.

Now add the enrichment to your pipeline. You'll use .mapUsingIMap as explained in the course material to

- Reference the cached IMap data
- Get the Trade ticker info – it is the foreign key.
- Create the Enriched Trade that contains the company name from the cache and the quantity and price from the stream.

```
15:53:46,461 [JobExecutionService] [192.168.10.115]:5701 [jet] [3.1] Execution plan for jobId=98d4-e92a-fda9-d49b, jobName='98d4-e92a-fda9-d49b', executionId=9bd5-32b2-3
15:53:46,466 [JobExecutionService] [192.168.10.115]:5701 [jet] [3.1] Start execution of job '98d4-e92a-fda9-d49b', execution 9bd5-32b2-3eb9-f061 from coordinator [192.16
15:53:47,481 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564008826471, company name='Microsoft Corporation', quantity=100, price=1260}
15:53:48,480 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564008827476, company name='Apple Inc. - Common Stock', quantity=100, price=3530}
15:53:49,487 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564008828476, company name='Apple Inc. - Common Stock', quantity=100, price=2897}
15:53:50,489 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564008829480, company name='Microsoft Corporation', quantity=100, price=1692}
15:53:51,491 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564008830483, company name='Apple Inc. - Common Stock', quantity=100, price=2278}
15:53:52,497 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564008831487, company name='Apple Inc. - Common Stock', quantity=100, price=1166}
15:53:53,498 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564008832491, company name='Apple Inc. - Common Stock', quantity=100, price=670}
15:53:54,503 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564008833494, company name='Microsoft Corporation', quantity=100, price=3623}
15:53:55,507 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564008834498, company name='Alphabet Inc.', quantity=100, price=889}
15:53:56,509 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564008835503, company name='Alphabet Inc.', quantity=100, price=3312}
15:53:57,513 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564008836506, company name='Microsoft Corporation', quantity=100, price=869}
```

Output

Once you've verified that you are enriching the trade data, stop your program.

Extra Credit

To demonstrate that you can change the cached table data without interrupting stream processing operations, modify the main try loop as shown.

```
    try {
        Job job = jet.newJob(p);

        Thread.sleep( millis: 5000);

        lookupTable.put( k: "AAPL",  v: "Apple");
        lookupTable.put( k: "GOOGL",  v: "Alphabet");
        lookupTable.put( k: "MSFT",  v: "Microsoft");

        job.join();

    } finally {
        jet.shutdown();
    }
```

When you run the program now, you should see the enriched data change from the first format to the second.

```
16:05:55,788 [JobExecutionService] [192.168.10.115]:5701 [jet] [3.1] Execution plan for jobId=440a-63dd-f5cb-57e3, jobName='440a-63dd-f5cb-57e3', executionId=6c81-4e
16:05:55,798 [JobExecutionService] [192.168.10.115]:5701 [jet] [3.1] Start execution of job '440a-63dd-f5cb-57e3', execution 6c81-4efc-5cee-665d from coordinator [192
16:05:56,821 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564009555806, company name='Microsoft Corporation', quantity=100, price=2047}
16:05:57,821 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564009556813, company name='Apple Inc. - Common Stock', quantity=100, price=3265}
16:05:58,825 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564009557817, company name='Alphabet Inc.', quantity=100, price=3991}
16:05:59,824 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564009558819, company name='Microsoft Corporation', quantity=100, price=3465}
16:06:00,827 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564009559821, company name='Microsoft', quantity=100, price=170}
16:06:01,831 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564009560822, company name='Alphabet', quantity=100, price=2981}
16:06:02,836 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564009561825, company name='Microsoft', quantity=100, price=2859}
16:06:03,836 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564009562829, company name='Apple', quantity=100, price=1516}
16:06:04,840 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564009563831, company name='Alphabet', quantity=100, price=1959}
16:06:05,841 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564009564834, company name='Alphabet', quantity=100, price=3903}
16:06:06,841 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] Trade{time=1564009565835, company name='Apple', quantity=100, price=3562}
```

Company name format changed

Output

Lab 3: Stateful Processing

Objective: monitor the trade stream and report if a price drops by more than 200.

Before you begin

Make sure you have no running instances of Jet on your local device.

```
private static Pipeline buildPipeline() {
    Pipeline p = Pipeline.create();

    p.drawFrom(TradeSource.tradeSource( tradesPerSec: 1))
      .withNativeTimestamps( l: 0 )

     // Detect if price between two consecutive trades drops by more than 200

     // Use the mapStateful to keep price of previous Trade
     // - Consider using com.hazelcast.jet.accumulator.LongAccumulator as a mutable container for long values
     // - Return the price difference if drop is detected, nothing otherwise

     .drainTo(Sinks.logger( m -> "Price drop: " + m));

    return p;
}
```

Use the mapStateful structure to store the price of the previous trade. Then write code that compares the current trade with the previous trade, and prints output to the console only if the trade price drops by 200 or more.

```
12:49:05,144 [JobExecutionService] [192.168.10.115]:5702 [jet] [3.2] Execution plan for jobId=0417-c4b3
12:49:05,149 [JobExecutionService] [192.168.10.115]:5702 [jet] [3.2] Start execution of job '0417-c4b3-
12:49:07,159 [loggerSink#0] [192.168.10.115]:5702 [jet] [3.2] Price drop: 4935
12:49:09,163 [loggerSink#0] [192.168.10.115]:5702 [jet] [3.2] Price drop: 2756
12:49:10,160 [loggerSink#0] [192.168.10.115]:5702 [jet] [3.2] Price drop: 1597
12:49:12,167 [loggerSink#0] [192.168.10.115]:5702 [jet] [3.2] Price drop: 408
12:49:14,167 [loggerSink#0] [192.168.10.115]:5702 [jet] [3.2] Price drop: 360
12:49:16,172 [loggerSink#0] [192.168.10.115]:5702 [jet] [3.2] Price drop: 865
12:49:17,177 [loggerSink#0] [192.168.10.115]:5702 [jet] [3.2] Price drop: 1956
```

Output

Your output will differ depending the randomly-generated trade data, but you will eventually see price drop output.

Lab 4: Windowing

Objectives:

1. Compute sum of trades in three second windows.
2. Compute sum of trades in three second windows, with results to date in one second intervals.
3. Compute sum of trades over 3 seconds, update every second.

Part 1: Three second window, three second update

```
private static Pipeline buildPipeline() {
    Pipeline p = Pipeline.create();

    // 1 - Read from the Trade Source (sources.TradeSource)

    // 2 - Use Native timestamps, no lag allowed

    // 3 - Compute sum of trades for 3-second intervals
    //
    // STEP 3.1
    // - Use 3 sec tumbling windows (defined in WindowDef.tumbling with size 3000
    // - Sum trade prices

    // STEP 3.2
    // - Get speculative results every second
    // - Use early results when defining the window
    // - Watch the early result flag in the console output

    // 4 - Drain to logger sink
```

Set up your pipeline to read from sources.TradeSource, using native timestamps as in previous labs. Use a tumbling window with a size of 3000 milliseconds. Your aggregate will be a sum of prices of all the trades completed in the configured window. Drain the output to the logger.

Run your program and observe the output. You should see an entry in the log every three seconds.

```
20:59:51,211 [JobExecutionService] [172.26.8.253]:5701 [jet] [3.1] Execution plan for jobId=0e05-853f-6d53-0824, jobName='0e05-853f-6d53-0824',
20:59:51,214 [JobExecutionService] [172.26.8.253]:5701 [jet] [3.1] Start execution of job '0e05-853f-6d53-0824', execution cff4-537e-743e-d8b6 f
20:59:55,262 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=20:59:51.000, end=20:59:54.000, value='7526922', isEarly=false}
20:59:58,266 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=20:59:54.000, end=20:59:57.000, value='7548932', isEarly=false}
21:00:01,284 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=20:59:57.000, end=21:00:00.000, value='7428674', isEarly=false}
21:00:04,293 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=21:00:00.000, end=21:00:03.000, value='7514148', isEarly=false}
21:00:07,305 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=21:00:03.000, end=21:00:06.000, value='7659601', isEarly=false}
```

Output – note the window start and end.

Part 2: Three second window, results to date every second

```
private static Pipeline buildPipeline() {
    Pipeline p = Pipeline.create();

    // 1 - Read from the Trade Source (sources.TradeSource)

    // 2 - Use Native timestamps, no lag allowed

    // 3 - Compute sum of trades for 3-second intervals
    //
    // STEP 3.1
    // - Use 3 sec tumbling windows (defined in WindowDef.tumbling with size 3000
    // - Sum trade prices

    // STEP 3.2
    // - Get speculative results every second
    // - Use early results when defining the window
    // - Watch the early result flag in the console output

    // 4 - Drain to logger sink
```

Use the same source and sink, and the same aggregate. Change the tumbling window to display early results every second.

```
21:06:05,813 [JobExecutionService] [172.26.8.253]:5701 [jet] [3.1] Execution plan for jobId=b287-0fa1-db8e-84a8, jobName='b287-0fa1-db8e-84a8', e
21:06:05,817 [JobExecutionService] [172.26.8.253]:5701 [jet] [3.1] Start execution of job 'b287-0fa1-db8e-84a8', execution bd11-b9c1-5272-9da9 fr
21:06:06,842 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=21:06:03.000, end=21:06:06.000, value='1109389', isEarly=true}
21:06:07,836 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=21:06:03.000, end=21:06:06.000, value='2529927', isEarly=true}
21:06:07,840 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=21:06:03.000, end=21:06:06.000, value='2529927', isEarly=false}
21:06:08,839 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=21:06:06.000, end=21:06:09.000, value='2469629', isEarly=true}
21:06:09,835 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=21:06:06.000, end=21:06:09.000, value='5012901', isEarly=true}
21:06:10,839 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=21:06:06.000, end=21:06:09.000, value='7494170', isEarly=true}
21:06:10,852 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=21:06:06.000, end=21:06:09.000, value='7494170', isEarly=false}
21:06:11,836 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=21:06:09.000, end=21:06:12.000, value='2433919', isEarly=true}
21:06:12,838 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=21:06:09.000, end=21:06:12.000, value='4913787', isEarly=true}
21:06:13,836 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=21:06:09.000, end=21:06:12.000, value='7360555', isEarly=true}
21:06:13,872 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=21:06:09.000, end=21:06:12.000, value='7360555', isEarly=false}
21:06:14,838 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=21:06:12.000, end=21:06:15.000, value='2469325', isEarly=true}
```

Output – note the values increasing within the three second window. Also note the “is early” flag is set for the results “inside” the three second window.

Part 3: Three second interval, updating every second

Use a sliding window with a three second interval, sliding every second. The aggregate is the same.

```
20:55:52,966 [JobExecutionService] [172.26.8.253]:5701 [jet] [3.1] Execution plan for jobId=8e81-5915-b276-892a, jobName='8e81-5915-b276-892a', e
20:55:52,970 [JobExecutionService] [172.26.8.253]:5701 [jet] [3.1] Start execution of job '8e81-5915-b276-892a', execution c737-5333-06c2-99b0 fr
20:55:55,012 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=20:55:50.000, end=20:55:53.000, value='2595348', isEarly=false}
20:55:56,014 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=20:55:51.000, end=20:55:54.000, value='5096510', isEarly=false}
20:55:56,014 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=20:55:52.000, end=20:55:55.000, value='5096510', isEarly=false}
20:55:57,020 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=20:55:53.000, end=20:55:56.000, value='5061589', isEarly=false}
20:55:58,028 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=20:55:54.000, end=20:55:57.000, value='5026729', isEarly=false}
20:55:59,031 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=20:55:55.000, end=20:55:58.000, value='7523644', isEarly=false}
20:56:00,037 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=20:55:56.000, end=20:55:59.000, value='7399780', isEarly=false}
20:56:01,040 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=20:55:57.000, end=20:56:00.000, value='7450061', isEarly=false}
20:56:02,044 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=20:55:58.000, end=20:56:01.000, value='7527235', isEarly=false}
20:56:03,052 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=20:55:59.000, end=20:56:02.000, value='7591225', isEarly=false}
20:56:04,054 [loggerSink#0] [172.26.8.253]:5701 [jet] [3.1] WindowResult{start=20:56:00.000, end=20:56:03.000, value='7583781', isEarly=false}
```

Output: Note the overlapping start and end times

Lab 5: Windowing with Grouping

Objective: For each stock symbol, display the sum of stock trades over a three second window.

```
private static Pipeline buildPipeline() {
    Pipeline p = Pipeline.create();

    // 1 - Read from the Trade Source (sources.TradeSource)

    // 2 - Use Native timestamps, no lag allowed

    // 3 - Compute sum of trades in 3-second intervals for each symbol

    // 4 - Drain to logger sink

    return p;
```

Use the grouping API to group events by ticker symbol. Use the windowing API to define the tumbling window and the aggregate operation.

```
23:27:08,472 [JobExecutionService] [192.168.10.115]:5701 [jet] [3.1] Execution plan for jobId=725a-eab4-2b87-188f, jobName='725a-eab4-2b87-188f', executionId=9001-902d
23:27:08,476 [JobExecutionService] [192.168.10.115]:5701 [jet] [3.1] Start execution of job '725a-eab4-2b87-188f', execution 9001-902d-3253-892c from coordinator [192.
23:27:10,517 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] KeyedWindowResult{start=23:27:06.000, end=23:27:09.000, key='MSFT', value='835809', isEarly=false}
23:27:10,517 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] KeyedWindowResult{start=23:27:06.000, end=23:27:09.000, key='AAPL', value='873752', isEarly=false}
23:27:10,518 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] KeyedWindowResult{start=23:27:06.000, end=23:27:09.000, key='GOOGL', value='822253', isEarly=false}
23:27:13,533 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] KeyedWindowResult{start=23:27:09.000, end=23:27:12.000, key='MSFT', value='2414207', isEarly=false}
23:27:13,533 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] KeyedWindowResult{start=23:27:09.000, end=23:27:12.000, key='AAPL', value='2479877', isEarly=false}
23:27:13,534 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] KeyedWindowResult{start=23:27:09.000, end=23:27:12.000, key='GOOGL', value='2398158', isEarly=false}
23:27:16,544 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] KeyedWindowResult{start=23:27:12.000, end=23:27:15.000, key='MSFT', value='2668216', isEarly=false}
23:27:16,545 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] KeyedWindowResult{start=23:27:12.000, end=23:27:15.000, key='AAPL', value='2513964', isEarly=false}
23:27:16,545 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] KeyedWindowResult{start=23:27:12.000, end=23:27:15.000, key='GOOGL', value='2430999', isEarly=false}
23:27:19,551 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] KeyedWindowResult{start=23:27:15.000, end=23:27:18.000, key='MSFT', value='2432876', isEarly=false}
23:27:19,551 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] KeyedWindowResult{start=23:27:15.000, end=23:27:18.000, key='AAPL', value='2528018', isEarly=false}
23:27:19,551 [loggerSink#0] [192.168.10.115]:5701 [jet] [3.1] KeyedWindowResult{start=23:27:15.000, end=23:27:18.000, key='GOOGL', value='2490624', isEarly=false}
```

Output

Extra credit: Put it all together!

Try the following variations:

1) Add enrichment to your output as in Lab 2.
2) Calculate the average price per symbol.
3) Use a sliding window instead of a tumbling window.

Lab 6: Jet from the CLI

Objectives:

1. Start Jet from the system command line.
2. Use Jet CLI tools to monitor operations
3. Submit a job using the command line.

Steps:

1. Download the Jet archive (.ZIP or .JAR) from jet.hazelcast.org and unpack it.

2. For this lab, we need to configure Jet to only look for cluster members on the local host. Open `$(JET)/config/hazelcast.xml` and make the following changes:

    a. Disable multicast discovery
    ```
    <multicast enabled="false">
    ```

    b. Enable local discovery by adding the following under `<join>`
    ```
    <tcp-ip enabled="true">
        <member>localhost</member>
    </tcp-ip>
    ```

3. Start an instance of Jet.
    ```
    $(JET)/bin/jet-start
    ```

4. Use the `jet.sh` tool to check the cluster state.

5. Get the JAR by building the job-deployment module of the training labs (mvn package).

6. Use the `jet.sh` tool to submit the produced JAR to the Jet instance.

7. Use the `jet.sh` tool to monitor the submitted job.

8. Use the `jet.sh` tool to stop the submitted job.

9. Use `jet-stop.sh` to stop the Jet instance.

Lab 7: Jet Management Center

Objective: Monitor Jet cluster and job operations using Jet Management Center

Steps:

1. Download the management center archive (.ZIP or .JAR) from jet.hazelcast.org and unpack it.

2. Start the cluster and submit the job as in Lab 6.

3. Run the appropriate script for your operating system without any arguments to start Hazelcast Jet Management Center.
   ```
   ./jet-management-center.sh
   ./jet-management-center.bat
   ```

4. Browse to http://localhost:8081. Login: admin Password: admin

5. Explore the Management Center GUI.
   a. View job status
   b. Double-click into job to view DAG and job details
   c. View cluster status