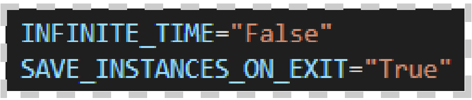
2. Real-time Streaming

This is the second out of the four guiding pdf files that aims to make it easier for the developer to get started with the project. In X sections, we will dive into the essentials of running all containers to produce and process the data in real-time into the PostgresSQL database

1) **Automation:** The information in this pdf file is based on the shell script "/instructions-executions/executionary/2. real-time-streaming.sh". Use this script as a reference to the explanations provided in this pdf. To be clear, you can just run "2. real-time-streaming.sh" and everything is taken care of automatically. With that said, this file is just providing elaborated explanations of the execution procedure.

2) **Configurations:** In "2. real-time-streaming.sh", the developer can change the values of the declared variables to select the deviating behaviors of the script. These variables are divided into two separate categories:

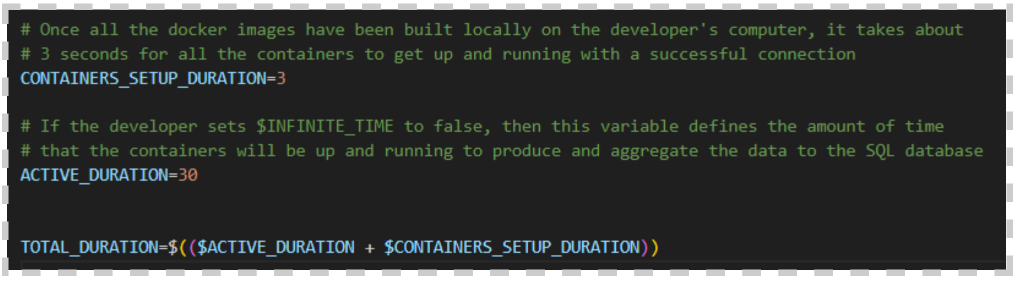
    1) **Containers:** Two variables in particular belong to this category, both of which are booleans. First, you must indicate whether you intend to run the containers until you manually stop them, or for a predefined amount of time. Secondly, you need to decide if you want to save the generated instances in the database so that you can access them the next time you start the containers, or if you simply want to reset the database as you stop the containers:



```
INFINITE_TIME="False"
SAVE_INSTANCES_ON_EXIT="True"
```

    2) **Duration:** If you set $INFINITE_TIME to "False", it is important that you pay attention to the duration you allow the containers to be

active. Therefore, it is imperative that you set an appropriate value on $ACTIVE_DURATION, as shown in the picture below.

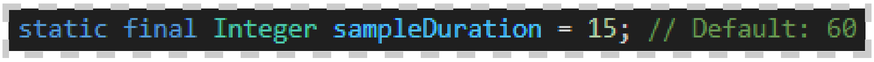


```
# Once all the docker images have been built locally on the developer's computer, it takes about
# 3 seconds for all the containers to get up and running with a successful connection
CONTAINERS_SETUP_DURATION=3

# If the developer sets $INFINITE_TIME to false, then this variable defines the amount of time
# that the containers will be up and running to produce and aggregate the data to the SQL database
ACTIVE_DURATION=30


TOTAL_DURATION=$(($ACTIVE_DURATION + $CONTAINERS_SETUP_DURATION))
```

Note that the sampling duration (the duration for the system to process all of the average temperatures and aggregate them into one final instance into the database) must be greater or equal to $TOTAL_DURATION. This variable is the sum of the containers' setup duration and your desired time frame for their activity. To be clear, since the sample duration constitutes the interval at which the city-instances are inserted to the database, we get the following relationship (if you are running the containers for a limited period of time):

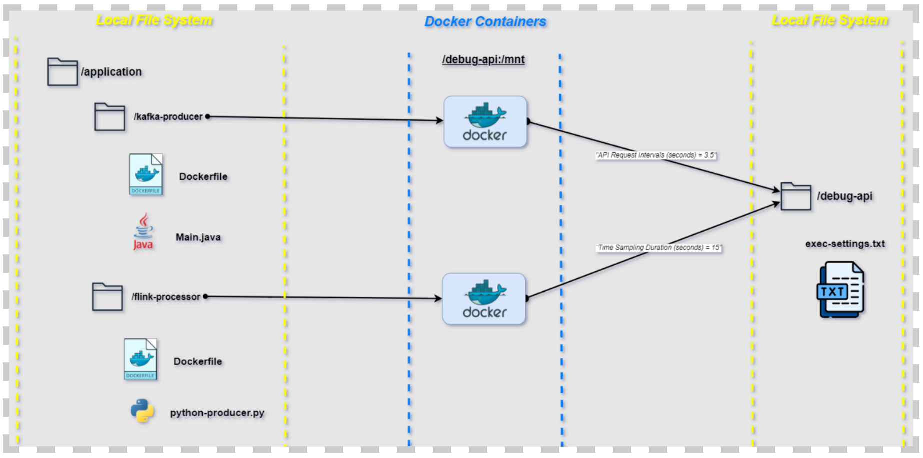
numInstancesInserted = $ACTIVE_DURATION / sampleDuration

With this algebraic establishment, we can predict the quantity of new instances that should be stored in the PostgresSQL database after the containers has stopped. If you wish to play around with the sampleDuration, navigate to "/application/flink-processor/src/main/java/Main.java" and change this variable:



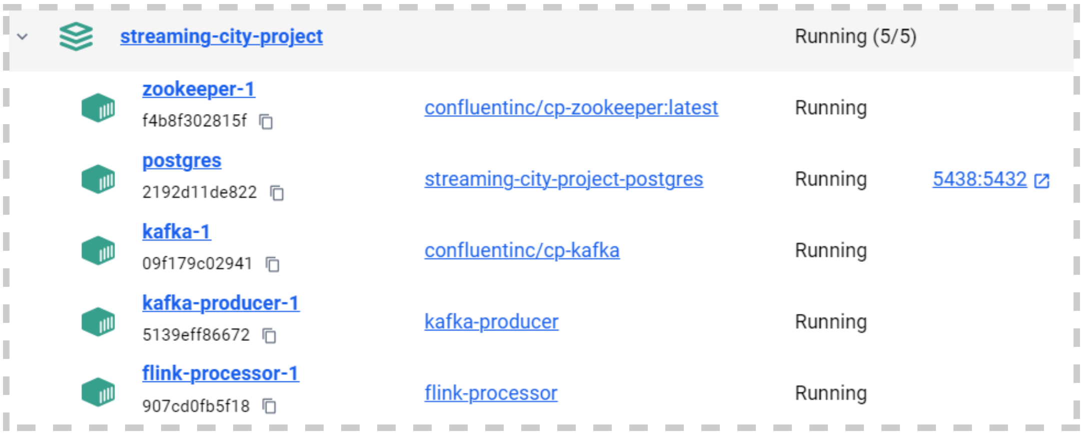
```
static final Integer sampleDuration = 15; // Default: 60
```

For further information, turn to "/debug-api/exec-settings.txt" that is connected through docker volumes and contains 2 strings. One of which represents the Weather API Request intervals (the interval in seconds to which the system fetches weather-data, that after $sampleDuration seconds gets aggregated to one instance of their average temperature value. The other string is the sampleDuration in seconds:



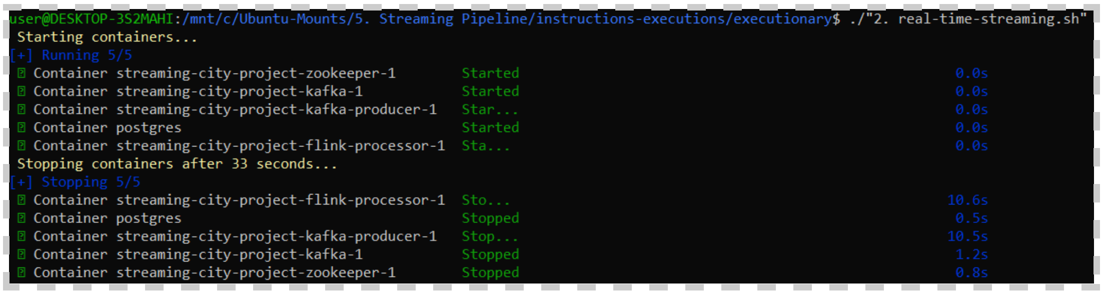
3) **Stream data:** The final step is to execute the shell script. Based on your inputted configurations in the previous section, the containers will either run for a limited amount of time or until you manually stop them. The expected outputs are:

Docker Desktop **(UI)**:



streaming-city-project		Running (5/5)	
zookeeper-1 f4b8f302815f	confluentinc/cp-zookeeper:latest	Running	
postgres 2192d11de822	streaming-city-project-postgres	Running	5438:5432
kafka-1 09f179c02941	confluentinc/cp-kafka	Running	
kafka-producer-1 5139eff86672	kafka-producer	Running	
flink-processor-1 907cd0fb5f18	flink-processor	Running	

Terminal **(CLI)**:



```
user@DESKTOP-3S2MAHI:/mnt/c/Ubuntu-Mounts/5. Streaming Pipeline/instructions-executions/executionary$ ./"2. real-time-streaming.sh"
 Starting containers...
[+] Running 5/5
 ▯ Container streaming-city-project-zookeeper-1         Started                                               0.0s
 ▯ Container streaming-city-project-kafka-1             Started                                               0.0s
 ▯ Container streaming-city-project-kafka-producer-1    Star...                                               0.0s
 ▯ Container postgres                                   Started                                               0.0s
 ▯ Container streaming-city-project-flink-processor-1   Sta...                                                0.0s
 Stopping containers after 33 seconds...
[+] Stopping 5/5
 ▯ Container streaming-city-project-flink-processor-1   Sto...                                               10.6s
 ▯ Container postgres                                   Stopped                                               0.5s
 ▯ Container streaming-city-project-kafka-producer-1    Stop...                                              10.5s
 ▯ Container streaming-city-project-kafka-1             Stopped                                               1.2s
 ▯ Container streaming-city-project-zookeeper-1         Stopped                                               0.8s
```

4) **The next step:** In the next tutorial, we will be going through "3. postgres-debugging.sh" so that we can debug and view the real-time produced as well as the stored SQL data that the containers in this tutorial generates.