STREAMER Getting Started v3.3

STREAMER team

October 2023

Welcome to STREAMER! This guide will show you how to easily install and run your first use case with STREAMER. For a detailed presentation of STREAMER read the "UserGuide.pdf".

Let's start!

First of all, download STREAMER framework + environment setup from `https://github.com/streamer-framework/streamer`

STREAMER is conceived to be used in two different ways (depending on your necessities):

1. **Development** use: (oriented to data scientists). You are interested in directly working on the code of the framework to add/develop several functionalities and test them.

2. **Product** use: (oriented to industrial use). If you want to use the framework as a product (no need to get in contact with the code but execute STREAMER), you need to have on your computer the basic services STREAMER requires and STREAMER instance already packed.

Contents

1 Getting ready for Deployment use

1. Run the basic services that STREAMER requires. Install them using the docker (recommended) following the steps of Section 3.1, or install them yourself following Section 4.

2. Install Eclipse (`https://www.eclipse.org/downloads/`) or the IDE you prefer.

3. Import the maven project: File → Import → Maven → Existing Maven Projects (and follow the steps to select the folder of STREAMER project).

4. (Optional) You are now ready to run our example use case:

   Run from eclipse the main class *ProducerMain* to launch the data ingester, or from console: java ProducerMain. You may also run directly the jar file generated as:

   ```
   java -cp
   target/streamer-1.0.0-test-jar-with-dependencies.jar
   cea.ProducerMain [setup_folder]
   ```

   Run from Eclipse the main class *LauncherMain* to launch the streaming pipeline, or from console: java LauncherMain. You may also run directly the jar file generated as:

   ```
   java -cp
   target/streamer-1.0.0-test-jar-with-dependencies.jar
   cea.LauncherMain [setup_folder]
   ```

5. Create your first use case in STREAMER by following the steps of Section 6.

6. Run your application in STREAMER as Section 5 shows.

(Important) For running our use case example or our proposed algorithms do not forget to install the packages they need in your computer for:

- Python:

  ```
  pip3 -r install services/requirementsPython.txt
  ```

- R:

```
install.packages(c("caret","RCurl","rredis",
"kernlab", "e1071", "neuralnet","xgboost"))
```

2 Getting ready for Production use

Try our example use case in STREAMER by following the steps of Section 3.

3 Using Docker

We make simple and transparent the installation of STREAMER and its services by using Docker. We provide 2 docker files that serve to:

1. **Services environment**(Section 3.1): it contains all the services used by the framework (Kafka&Zookeeper, Redis. InfluxDB, Kibana&ElasticSearch).
2. **Production environment**(Section 3.2): it contains STREAMER for production purpose.

When downloading the zip file you will find:

- streamer_environment (folder)
  - docker-compose.yml (file)
  - streamer_environment (sub-folder)
    - dockercompose.yml (file)
    - Dockerfile_producer (file)
    - Dockerfile_streamer (file)
    - Jar file (file)
    - requirements.txt (file)
    - data (sub-sub-folder)

3.1 Install & run all the services from Docker (recommended)

[Warning] for Linux-based systems, you may need to run all the commands in "sudo" mode as, for instance:

```
sudo docker-compose up --build -d
```

Follow the following steps to setup all necessary services before running STREAMER:

1. Install docker on your machine. At the following link, you will find how to install the docker for all the different operating systems (Windows, Linux, Mac): `https://docs.docker.com/get-docker/`.

2. For Linux, also install docker-compose from `https://docs.docker.com/compose/install/`.

3. Unzip the provided folder "streamer_environment".

4. Open a terminal and change the directory to this "streamer_environment" directory.

5. Run the following command to start the services:

```
docker-compose up --build -d
```

In order to check if the services are running properly, check the following command:

```
docker ps
```

To stop the services, use the following command:

```
docker-compose down
```

Note: If after following the previous steps, you face a similar error to

```
ERROR: [1] bootstrap checks failed [1]: max virtual memory
areas vm.max\_map\_count [65530] is too low, increase to at
least [262144]
```

you can solve it by increasing your virtual memory. Run the command:

```
sudo sysctl -w vm.max_map_count=262144
```

and then build the docker again with:

```
sudo docker-compose up --build -d
```

3.2 Running STREAMER in production environment

[Warning] For Linux-based systems, you may need to run all the commands in "sudo" mode as, for instance:

```
sudo docker-compose up --build -d
```

1. Before running STREAMER for production purposes, complete all the steps of case 1 above to run all the services.

2. Open a terminal and go to "*streamer_environment/streamer_environment*" directory (sub-folder of streamer_environment folder).

3. Run the following command to start the framework:

```
docker-compose up --build
```

Add (-d) to the command above to keep it in background. (outputs not displayed).

- In order to check if the framework is running properly, check the following command:

```
docker ps
```

- To stop the services, use the following command: If you started the framework without using (-d) property, first press (ctrl + c) and then the following (with (-d) or not):

```
docker-compose down
```

- To check the logs: Get the container ID corresponding to the container name: "streamer_environment_streamer" using the following command:

```
docker ps
```

Then use that ID to access the container using the following command:

```
docker exec -it container_id bash
```

Then you can check the logs by accessing the logs folder using:

```
cd logs
```

4 Installing STREAMER services yourself (from sources)

STREAMER uses the following services that you can install yourself:

[Compulsory]:

- apache kafka / soft / Apache License 2.0 / kafka.version=2.7.0
- zookeeper / soft / Apache License 2.0 / zookeeper.version=3.5.9 `https://kafka.apache.org/quickstart` or apache kafka & zookeeper confluent 6.2.1
- redis server / soft / BSD / Redis version=6.2 `https://redis.io/`
- influxdb / soft / MIT / version=1.8.9 `https://portal.influxdata.com/`

[Optional]:

- elasticsearch / soft / Apache License 2.0 / version=7.15.0 `https://www.elastic.co/elasticsearch`
- kibana / soft / Elastic License/Apache License / version=7.15.0 `https://www.elastic.co/kibana`

5 How to run a STREAMER

For using Kibana graphical interface, open a web browser and type `http://localhost:5601`. You may:

- Create your own dashboard.
- Import a dashboard by going to Settings → Stack Management → Saved Objects → Import, and add your '.ndjson' file. There are some existing dashboards in */services/kibana-dashboards* folder that you can use or adapt. Finally, click Settings → Analytics → Dashboard and select the dashboard you want to open.

5.1 For a classical use of STREAMER

1. Set up all the properties in the files .props within src/main/resources.
2. For using the data producer simulator functionality, run *ProducerMain.java* (placed in folder *streamer/src/test/java/cea/*) with the following arguments:

   ```
   ProducerMain.java [origin1 ... originN]
   ```

   A producer is now writing in Kafka topic(s). [origin1 ... originN] Arguments are optional. Each of them runs a problem in a separate process; its name originX indicates the folder where the properties files are located for this specific execution. If no arguments indicated, the system considers the properties files directly placed in *src/main/resources/setup* folder.
3. For using the streaming pipeline functionality, run *LauncherMain.java* (placed in folder *streamer/src/test/java/cea/*) with the following arguments:

   ```
   LauncherMain.java [origin1 ... originN]
   ```

   [origin1 ... originN] Arguments are optional. Each of them runs a problem in a separate process; its name originX indicates the folder where the properties files are located for this specific execution. If no arguments indicated, the system considers the properties files directly placed in *src/main/resources/setup* folder.
4. For using the learning API in offline mode (algorithms test and/or train) run *AlgInvokerMain_offline.java* (placed in folder *streamer/src/test/java/cea/*) with the following arguments:

   ```
   AlgInvokerMain_offline.java arg1 arg2 arg3 arg4 [arg5]
   ```

   Arguments are:

(a) arg1. Data source type: From where data must be retrieved. Option: [influx or file]. It could be either "influx" to build the model using the data stored in InfluxDB or "file" to directly use the data from the file specified in algs.properties (data.training).

(b) arg2. Origin: [influx-data-base-name or origin]. origin indicates the folder where the properties files are located for this specific execution. If no arguments are indicated, the system considers the properties files directly placed in *src/main/resources/setup* folder.

(c) arg3. Perform training [true or false] (train).

(d) arg4. Perform model running [true or false] (test).

(e) arg5. [Optional] Path from where to store/retrieve the trained model.

5.2 For a distributed environment of STREAMER

1. Set up all the properties in the files .props within *src/main/resources*.

2. Launch a server by running *ServerLauncher.java* (placed in folder streamer/src/federated) with the following arguments:

   ```
   java ServerLauncher.java [origin]
   ```

   'origin' argument is optional. It indicates the folder where the properties files are located for this server execution. If no argument is indicated, the system considers the properties file of the distributed learning module demo, directly placed in *src/main/resources/setup/cmapss/server* folder.

3. Launch an online or offline client by:

   (a) running a classic STREAMER instance (for an online client) as in section 5 (steps 1 to 3). The 'distributedMode' property should be set to 'true' in the federated.props file.

   (b) running ClientLauncher.java (for offline client(s), placed in folder streamer/src/federated) with the following arguments:

   ```
   java ClientLauncher.java [origin1 ... originN]
   ```

   [origin1 ... originN] arguments are optional. Each of them runs an offline client in a separate process; its name *originX* indicates the folder where the properties files are located for this specific execution. If no argument is indicated, the system considers the properties files of the distributed learning module demo, directly placed in *src/main/resources/setup/cmapss/client_0* folder.

4. Repeat step 3 to launch as many offline or online clients as you want.

Note: You can launch several clients at the beginning of an FL experience and then launch new clients or disconnect existing ones at any time during the experience.

6 How to integrate your use case + algorithm in STREAMER

6.1 For a classical use of STREAMER

In order to create your running application in STREAMER, two main phases are required:

6.1.1 Integrate or create your algorithm

1. Add the main algorithm class to the cea.streamer.algs package :
   - Implement both methods learn and run for training and testing phases (if necessary).
   - If your model is not coded in Java, you will have to use Redis as an intermediary data storage to save the arriving records to be accessible from the algorithm (in the example below, the algorithm is coded in Python).
2. Add the training and testing python files to the path *src/main/resources/algs*
   - While coding the algorithms in Python, pay attention to the training file to read the data from Redis and write the model in Redis.
   - Likewise, for the testing file, pay attention to read the data and the model from Redis, write the predictions as one line String in Redis, and precise the separator used between the predictions' values in another Redis key. These predictions are read by the framework in order to do the realtime evaluation of the model's performance.

6.1.2 Integrate or create use case application

1. Add a folder with the project name.
2. Add two configuration files to this folder:
   - Algs.props
   - Streaming.props

   Note: The user can choose metrics for evaluating the model by simply adding the name of the metric class. It is also possible for the user to extend the existing metrics with new ones.
3. Create the record class:
   (a) It is essential to name the record class in this exact way: problem.type + "Record" + ".java"
   (b) You can precise the headers of the dataset

(c) Implement the fill method to indicate how the records should be filled with the correct values.

(d) In case the dataset does not contain a timestamp, you may use function *sleep()* and generate a new timestamp.

As you arrived this far, it is time to run the application:

```
\$ java -cp dsplatform-1.0.0-test-jar-with-dependencies.jar
cea.streamer.LauncherMain kdd-cup-99

\$ java -cp dsplatform-1.0.0-test-jar-with-dependencies.jar
cea.streamer.ProducerMain kdd-cup-99
```

6.2 For a distributed use of STREAMER

In order to create your running distributed application in STREAMER, two main phases are required:

6.2.1 Integrate or create your algorithm

With the first version of the distributed learning module, you can only use Python algorithms during the distributed learning process.

1. Add the folder containing your Python algorithm to the path *src/main/resources/algs*.
   - Create a Python file that manages the process of the possible actions (initialization, training, evaluation, inference). In fact, STREAMER executes the Python code with an argument '*state*' that represents the action. So, the Python file should implement and run the corresponding methods according to the following *state* values it receives:
     - *init*: Initializes the model and pushes it to Redis.
     - *train*: Fits the model and pushes the updated one to Redis.
     - *evaluate*: Evaluates the model and pushes the metrics (the loss) to Redis.
     - *inference*: Makes the inference and pushes the prediction to Redis.
2. Add an algorithm java class in *cea.streamer.algs* package only if you want to perform the inference with an online client.
   - Only the *run()* method should be implemented as the training step will be controlled by the server and called through the 'Client' federated overlay.
   - You must use *Redis* as an intermediary data storage to save the arriving records to be accessible from the algorithm as the model is coded in Python.

6.2.2 Integrate or create use case application

For a distributed use of STREAMER, you have to setup a server and then, as many clients (offline or online) as you want.

1. Setup for the server:
   (a) Add a folder with the project name + '-server'.
   (b) Add a *federated.props* file to this folder.
2. Setup for one client (repeat it for several clients):
   (a) Add a folder with the project name + '_client_id' (*id* is a unique identifier for the client). For instance *folder_client_0*.
   (b) Add three configuration files to this folder:
      - *algs.props*
      - *streaming.props*
      - *federated.props*

      Warning: These 3 properties files must be completed in a specific way, depending on the nature of the client (online or offline). See the user guide for more information.
   (c) Create the record class:
      i. It is essential to name the record class in this exact way: problem.type + "Record" + ".java".
      ii. You can precise the headers of the dataset.
      iii. Implement the *fill()* method to indicate how the records should be filled with the correct values.
      iv. In case the dataset does not contain timestamp, you may use function *sleep()* and generate a new timestamp.

      Note: In the same use case, you can use this class for several clients if each dataset has a similar format, or create new ones if needed.

Once you have setup the server and the desired number of clients, run the application by following the steps of section 5.2 with your proper 'origin' arguments (corresponding to the folders of the server and the clients, where the properties files are located). For instance:

```
java ServerLauncher.java cmapss/server

java ClientLauncher.java cmapss/client_0 cmapss/client_1 cmapss/client_2
```