Institute for Data Engineering
Module Big Data – Lab Exercises

Project Topic C – Radiation Tracking

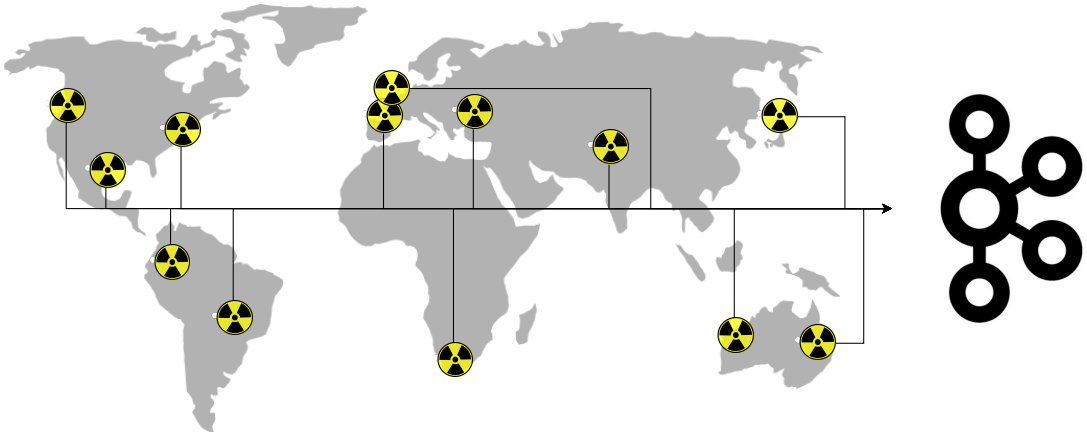

Figure 1: Example Scenario

The advent of the Internet of Things (IoT) increases data volume and velocity by providing data updates in a continuous way, allowing, e.g., the constant monitoring of human activities, locations, health states, and communication patterns [2]. A typical example for this is data generation through sensor networks, where a large number of sensors at different locations generate single data items continuously (however, not necessarily regularly), leading to *data streams* [1]. The volume of the data in such streams makes it necessary to process large amounts of data, while the velocity requires that the data is processed in (near) real-time [4]. If this is not achieved, the data is already outdated before it gets processed. In such a case, processing the data items does not lead to any benefits.

Within this lab project, it is the goal to setup and use a stream processing framework in order to be able to process large amounts of sensor data. As an example dataset, the latest version of the *Safecast Radiation Measurements*[1] dataset will be applied [3]. This dataset features millions of radiation readings from different locations all over the world, summing up to roundabout 25.8 GB of uncompressed data. The data comprises information about radiation measurements at different points of time at different locations (indicated by latitude and longitude). The radiation measurements are given in counts per minute (CPM), which says how many radioactive particles are measured in a minute at a location by a single sensor. Naturally, such an amount of data also contains noise and wrong data.

The ultimate goal of this task is to populate a (world) map with information from the Safecast dataset, and to provide a number of operations which can be performed on this data. Once the map is established, you can provide further features, e.g., summarizing single data items into larger data blobs, generating alerts whenever the radiation rate is above a certain, user-configurable threshold, or using different colors to indicate safe and dangerous radiation levels.

Please consider that most map APIs are not able to scale forever, i.e., it is surely not a good idea to add millions of data points items to a map. Hence, you will have to delete older data items from the map again. In normal scenarios, data items by fixed sensors (i.e., if the sensor location

[1] `https://safecast.org/data/download/`

has not changed) should be replaced on the map instead of simply adding another data item.

Different stream processing actions should be performed on the data. For instance, all empty (i.e., no radiation value available) data items can immediately be discarded. Last but not least, it is necessary to make the solution configurable. As has already been mentioned, a user should be able to define from which value the radiation is seen critical, e.g., 100 or 300 CPM. Also, it should be possible to define in which area data should be displayed on the map (thus decreasing the amount of data on the map) or for which timespan. For the final presentation, it is recommended to identify interesting locations and timespans. The map provided by the Safecast project[2] might be helpful for this.

The technical foundation for this project is Apache Kafka. The first task is to populate an Apache Kafka[3] instance with the radiation information from the dataset. For this, you should implement a configurable data provider (see task description below). Afterwards, Kafka Streams should be used to process the data. Separate processing operators [1] should be provided by your team, one for each data analysis functionality you want to provide.

Outcomes

The expected outcomes of this project are two-fold: (1) the actual project solution, (2) an intermediary and a final presentation of your results.

Project Solution

The project has to be hosted on a Git repository in TUHH's GitLab instance[4] – you will get instructions about the repository setup at the lab kickoff meeting. Every member of your team has to use its own, separate Git account. *We will check who has contributed to the source code*, so please make sure you use your own account when submitting code to the repository. Furthermore, it is required to provide an easy-to-follow README that details how to deploy, start and test the solution. The best practice is to provide a README that describes "Plug-and-Play" instructions on how to use your solution.

For the submission (see below), you also have to create one or more Docker images, which contain(s) your complete implemented solution, i.e., including all dependencies.

Presentations

There are two presentations. The first one is during the consultation hours, and describes your status at that particular point of time. This presentation will not be graded, i.e., it is primarily a way to check your progress and to show to other groups what you are working on and how you want to solve the problem.

The second presentation is during the final meetings and contains all your results. The actual dates are announced in Stud.IP.

Every member of your team is required to present in either the first *or* the second presentation. Each presentation needs to consist of a slides part and a demo of your implementation. Think carefully about how you are going to demonstrate your implementation, as this will be part of the grading. You have 10 minutes (strict) of time for your presentation in the consultation hour, and 15 minutes (also strict) of time for your presentation at the final meeting. At the first presentation, you can go for a slide-based presentation, a live demo, or a combination of both, depending on your progress until that point of time.

The past has shown that providing a nice use case story usually helps to present the project outcomes. While providing such a use case story is not an absolute must, it will surely help the audience to understand your work better.

[2] `https://map.safecast.org/`
[3] `https://kafka.apache.org/`
[4] `https://collaborating.tuhh.de/`

Grading

A maximum of 60 points are awarded in total for the project. Of this, 70% are awarded for the implementation (taking into account both quality and creativity of the solution as well as code quality and documentation), and 30% are awarded for the final presentation (taking into account content, quality of slides, presentation skills, and discussion).

A strict policy is applied regarding plagiarism. Plagiarism in the source code will lead to 0 points for the particular student who has implemented this part of the code. If more than one group member plagiarizes, this may lead to further penalties, i.e., 0 points for the implementation of the whole group.

Deadline

The hard deadline for the project is **July 9th, 2024**, 23:59. Please submit the presentations via Stud.IP. The Docker images need to be uploaded to DockerHub and made available to the lecturer team. For this, please write a mail on how to access the containers to nisal.hemadasa@tuhh.de. The deadline for this is also July 9th, 2024, 23:59. Late submissions will not be accepted.

Test Cloud Infrastructure

This year, the Google Cloud Platform is supporting the Big Data lecture with an Education Grant, which you can use for Virtual Machines (VMs) and other cloud resources. The computational resources can be used for free, however, you need to own a Google account to use them. To access the resources, please visit the following URL (please note that this is a masking URL, i.e., you need to click it, simply copy/pasting it will *not* help): http://google.force.com/. The full URL is also provided in Stud.IP.

Please note that you need to request the resources until August 2nd, 2024; the coupons are then valid until April 2nd, 2025. Afterwards, they will become void and you will not be able to access the budget.

The e-mail address you provide in this form does *not* have to be your Google account, but it *must* be an address within the TU Hamburg domain, i.e., `@tuhh.de`. By providing such an address, you can claim a coupon code, which you can then use to redeem a $50.00 voucher for the Google Cloud Platform. Detailed instructions are provided at this URL and subsequent e-mails you will receive from Google.

You can redeem one coupon code per e-mail address at a time. The voucher of $50.00 is in most cases sufficient to conduct the implementation tasks of the lab exercises. However, if this is not the case, please send a mail to nisal.hemadasa@tuhh.de. Please also take into account that there are the free tier resources of the Google Cloud Platform, which are usually sufficient to test and run your solutions.

Note that exceeding this budget will lead to immediate shutdown of the project by Google, and we have no means of influencing this, or providing extensions. We therefore recommend you to monitor your resource usage, in order to avoid unnecessary spending, for instance, by VMs left running.

You are allowed to use the Google Cloud Resources only for the purpose of this module. Both Google and TU Hamburg are monitoring the use of the resources, and any misuse (hosting of illegal data, cryptocurrency mining, etc.) will be punished. In case of any questions regarding the organization of the Google Cloud Platform, please send an e-mail to nisal.hemadasa@tuhh.de.

Please take care that you do not publish any credentials or (ssh) keys on the Internet. For your implementation, please use dedicated credential files and add these credential files and the keys to the .gitignore file or put them on a system path outside of your Git repository. If you accidentally leak any credentials or ssh keys, inform us immediately. Google is pretty good at identifying leaked credentials, and will then shut down your account at least temporarily. If this happens just before the deadline, there is a good chance that your account will not be released again in due time.

In case of noncompliance, the person(s) responsible will fail the lab!

Stage 1

In the first stage, the focus should be on setting up and configuring the infrastructure, i.e., Apache Kafka.

Tasks:

1. Setup and configure Apache Kafka in a Docker container. You may run the container locally at this stage.
2. Implement the data provider, which represents a producer for Apache Kafka. This data provider retrieves the radiation and location information from the Safecast dataset, and submits it to Apache Kafka. The data needs to be replayed in the same order as it was recorded (the "Time of Capture"), i.e., all location information with the same timestamp need to be submitted at the same time. It is advisable to make the submission speed configurable to apply different submission speeds during testing and presentation of the system.

   Note: Do *not* use the REST API provided by Safecast. Instead, download the measurements and develop the data provider as described.
3. Integrate first data items into Apache Kafka and implement one operator, i.e., one data processing functionality as discussed above.

Please note: Stage 1 is what you should have achieved roundabout half-way through the semester. This is a recommendation, not a must. But it will help to avoid too much crunch time at the end of the semester.

Stage 2

In Stage 2, the solution from Stage 1 needs to be extended by more sophisticated functionalities. Most importantly, the map needs to be shown in a user interface and the functionalities for configuration and data filtering need to be added.

Tasks:

1. Implement a Web-based GUI, featuring the map and the configuration means. The GUI should be user-friendly and easy to present, since it will be an important means for the presentation of your results in the final presentation.
2. Retrieve the processed information from the stream processing topology and display the data on the map. Make the speed of showing data configurable, so that you can speed up adding new data items during the presentation.
3. Implement additional stream processing operators.
4. At this stage, your solution (user interface and any middleware you may need) needs to be deployable in the Google Cloud Platform (while it is sufficient to run it locally during Stage 1).

References

[1] Henrique Andrade, Bugra Gedik, and Deepak Turaga. *Fundamentals of Stream Processing.* Cambridge University Press, 2014.

[2] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54:2787–2805, 2010.

[3] Azby Brown, Pieter Franken, Sean Bonner, Nick Dolezal, and Joe Moross. Safecast: successful citizen-science for radiation measurement and communication after Fukushima. *Journal of Radiological Protection*, 36(2):S82–S101, 2016.

[4] Andrew McAfee and Erik Brynjolfsson. Big Data: The Management Revolution. *Harvard Business Review*, 90(10):61–67, 2012.