Spark Covid Analysis

APACHE Spark

Goals

Given the data collected by the Centre for Disease Prevention and Contro the application compute:

- Seven days moving average of new reported cases, for each county and for each day.
- Percentage increase (with respect to the day before) of the seven days moving average, for each country and for each day.
- Top 10 countries with the highest percentage increase of the seven days moving average, for
- each day.

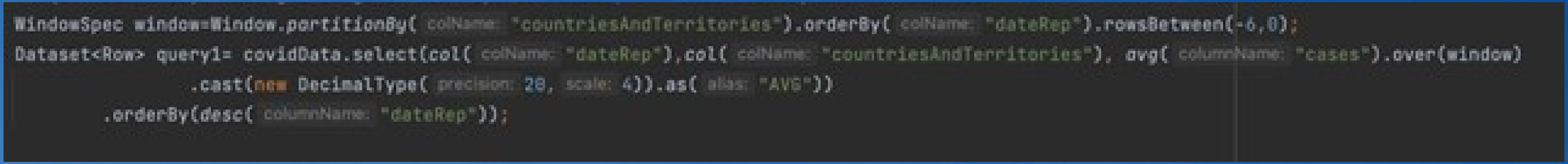
Why Spark SQL

Spark SQL is Apache Spark's module for working with structured data.

Among its main features are particularly these helpful:

- Working with different dataset type
- Easy query writing
- Automatic optimization

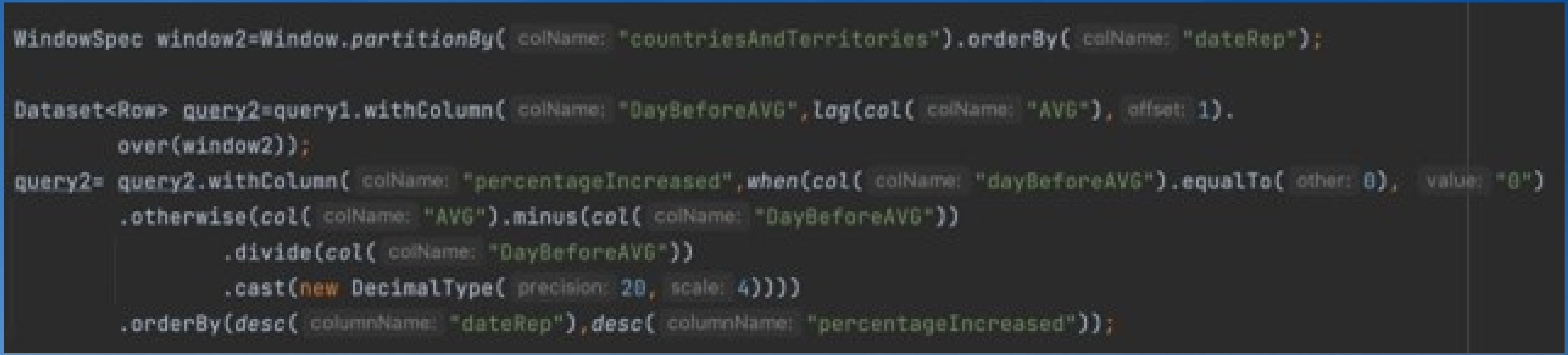
Seven days moving average of new reported cases, for each county and for each day



```
WindowSpec window=Window.partitionBy( colName: "countriesAndTerritories").orderBy( colName: "dateRep").rowsBetween(-6,0);
Dataset<Row> query1= covidData.select(col( colName: "dateRep"),col( colName: "countriesAndTerritories"), avg( columnName: "cases").over(window)
                .cast(new DecimalType( precision: 28, scale: 4)).as( alias: "AVG"))
        .orderBy(desc( columnName: "dateRep"));
```

The seven days moving average of new reported cases, fore each country and for each day is calculated reading the data with a Window that partition all the data by country and then sort those data by date. The window will cover the entries of the last 7 days of the collection at time and will move forward day by day. In this way the average of the reported case for the days in the window is calculated.

Percentage increase (with respect to the day before) of the seven days moving average, for each country and for each day

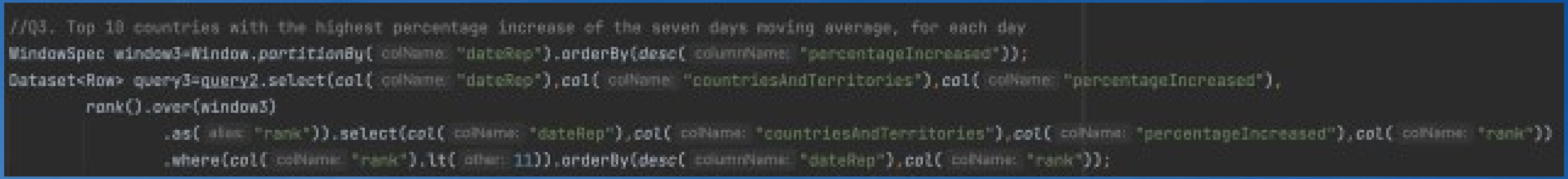


```
WindowSpec window2=Window.partitionBy( colName: "countriesAndTerritories").orderBy( colName: "dateRep");

Dataset<Row> query2=query1.withColumn( colName: "DayBeforeAVG",lag(col( colName: "AVG"), offset: 1).
        over(window2));
query2= query2.withColumn( colName: "percentageIncreased",when(col( colName: "dayBeforeAVG").equalTo( other: 0),  value: "0")
        .otherwise(col( colName: "AVG").minus(col( colName: "DayBeforeAVG"))
                .divide(col( colName: "DayBeforeAVG"))
                .cast(new DecimalType( precision: 20, scale: 4))))
        .orderBy(desc( columnName: "dateRep"),desc( columnName: "percentageIncreased"));
```

The percentage increase of the seven days moving average is calculated starting from the previous query. First of all, the column “dayBeforeAvg” is added to the result of the first query, in this column there is the seven days moving average of the day before for each day and for each country(as before a window is used to read data of a range of day, but this time the window cover only the selected days and the one right before). Then another column is added which contain the percentage
increase of the seven days moving average.

top 10 countries with the highest percentage increase of the seven days moving average, for each day

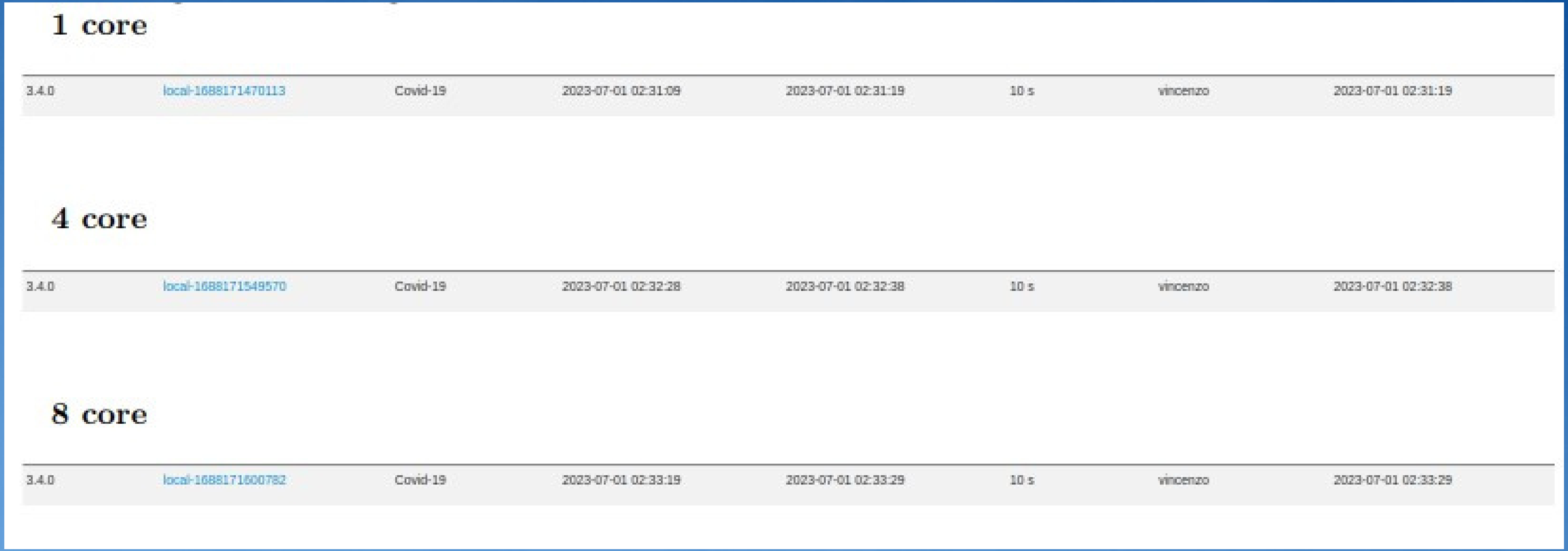


```
//Q3. Top 10 countries with the highest percentage increase of the seven days moving average, for each day
WindowSpec window3=Window.partitionBy( colName: "dateRep").orderBy(desc( columnName: "percentageIncreased"));
Dataset<Row> query3=query2.select(col( colName: "dateRep"),col( colName: "countriesAndTerritories"),col( colName: "percentageIncreased"),
        rank().over(window3)
                .as( alias: "rank")).select(col( colName: "dateRep"),col( colName: "countriesAndTerritories"),col( colName: "percentageIncreased"),col( colName: "rank"))
                .where(col( colName: "rank").lt( other: 11)).orderBy(desc( columnName: "dateRep"),col( colName: "rank"));
```

This query is also calculated starting from the previous one. The data of the second query are partitioned by day and sorted using the “percentageIncreased” attribute. With the function rank() to each day is assigned a value based on its rank. Then only the country with rank less than 11 are chosen to be part of the result

Testing: Official Dataset

These are the time that the application need to execute all the three query with the dataset released by ECDC:



1 core

3.4.0	local-1688171470113	Covid-19	2023-07-01 02:31:09	2023-07-01 02:31:19	10 s	vincenzo	2023-07-01 02:31:19

4 core

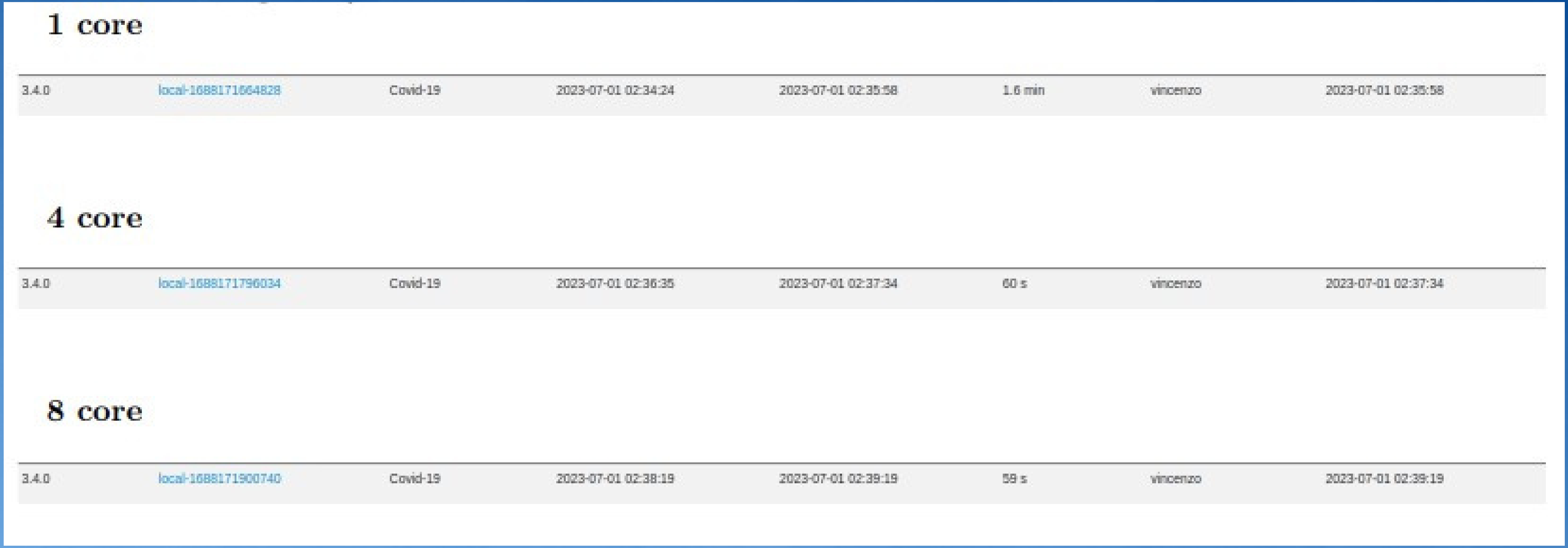
3.4.0	local-1688171549570	Covid-19	2023-07-01 02:32:28	2023-07-01 02:32:38	10 s	vincenzo	2023-07-01 02:32:38

8 core

3.4.0	local-1688171600782	Covid-19	2023-07-01 02:33:19	2023-07-01 02:33:29	10 s	vincenzo	2023-07-01 02:33:29

Testing: Dataset 100x bigger

These are the result with a dataset 100x bigger than the official one



1 core

3.4.0	local-1688171664828	Covid-19	2023-07-01 02:34:24	2023-07-01 02:35:58	1.6 min	vincenzo	2023-07-01 02:35:58

4 core

3.4.0	local-1688171796034	Covid-19	2023-07-01 02:36:35	2023-07-01 02:37:34	60 s	vincenzo	2023-07-01 02:37:34

8 core

3.4.0	local-1688171900740	Covid-19	2023-07-01 02:38:19	2023-07-01 02:39:19	59 s	vincenzo	2023-07-01 02:39:19