High Performance Computing Project

Machine Learning on the cloud: when do we need it?

**Introduction**

The goal of this project is to perform an experimental investigation considering

a Machine Learning task implemented in different modalities and evaluating the

cost-benefit of each of them.

The reference architecture is based on PySpark, which can be run in local or on the cloud.

What it is required:

- To identify a dataset with associated ML task (see the list we provided)

- To design the implementation of the task considering a K-Fold Cross Validation

procedure to train the model. In doing that you have to consider \*at least two\*

of the following scenarios:

        + Your local machine + PySpark

        + Azure HDInsight + PySpark

        + Google Cloud Platform + PySpark

- Considering the following aspects:

        + Analysis of the processing time (difference between clock) required for the

entire training of the model

        + Analysis of the load balancing on nodes (using profiling/monitoring servers)

        + The economical costs

**Spark**

Apache Spark is an open source, general-purpose distributed computing engine used for processing and analyzing a large amount of data. Just like Hadoop MapReduce, it also works with the system to distribute data across the cluster and process the data in parallel. Spark uses master/slave architecture i.e. one central coordinator and many distributed workers. Here, the central coordinator is called the driver.

The driver runs in its own Java process. These drivers communicate with a potentially large number of distributed workers called executors. Each executor is a separate java process. A Spark Application is a combination of driver and its own executors. With the help of cluster manager, a Spark Application is launched on a set of machines. Standalone Cluster Manager is the default built in cluster manager of Spark. Apart from its built-in cluster manager, Spark also works with some open source cluster manager like Hadoop Yarn, Apache Mesos etc.

SparkContext is the heart of Spark Application. It establishes a connection to the Spark Execution environment. It is used to create Spark RDDs, accumulators, and broadcast variables, access Spark services and run jobs. SparkContext is a client of Spark execution environment and acts as the master of Spark application. The main works of Spark Context are:

* Getting the current status of spark application
* Canceling the job
* Canceling the Stage
* Running job synchronously
* Running job asynchronously
* Accessing persistent RDD
* Unpersisting RDD
* Programmable dynamic allocation

iii. Spark Application

The Spark application is a self-contained computation that runs user-supplied code to compute a result. A Spark application can have processes running on its behalf even when it’s not running a job.

iv. Task

A task is a unit of work that sends to the executor. Each stage has some task, one task per partition. The Same task is done over different partitions of RDD.

Learn: Spark Shell Commands to Interact with Spark-Scala

v. Job

The job is parallel computation consisting of multiple tasks that get spawned in response to actions in Apache Spark.

vi. Stage

Each job divides into smaller sets of tasks called stages that depend on each other. Stages are classified as computational boundaries. All computation cannot be done in a single stage. It achieves over many stages.

At run time, the main() method of the program runs in the **driver**. The driver is the process that runs the user code that creates RDDs, and performs transformation and action, and also creates SparkContext. When the Spark Shell is launched, this signifies that we have created a driver program. On the termination of the driver, the application is finished.

The driver program splits the Spark application into the task and schedules them to run on the executor. The task scheduler resides in the driver and distributes task among workers. The two main key roles of drivers are:

* Converting user program into the task.
* Scheduling task on the executor.

The structure of Spark program at a higher level is: RDDs consist of some input data, derive new RDD from existing using various transformations, and then after it performs an action to compute data. In Spark Program, the DAG (directed acyclic graph) of operations create implicitly. And when the driver runs, it converts that Spark DAG into a physical execution plan.

Spark relies on **cluster manager** to launch executors and in some cases, even the drivers launch through it. It is a pluggable component in Spark. On the cluster manager, jobs and action within a spark application scheduled by Spark Scheduler in a FIFO fashion. Alternatively, the scheduling can also be done in Round Robin fashion. The resources used by a Spark application can dynamically adjust based on the workload. Thus, the application can free unused resources and request them again when there is a demand. This is available on all coarse-grained cluster managers, i.e. standalone mode, YARN mode, and Mesos coarse-grained mode.

We can submit a job with spark-submit command. It happens that:

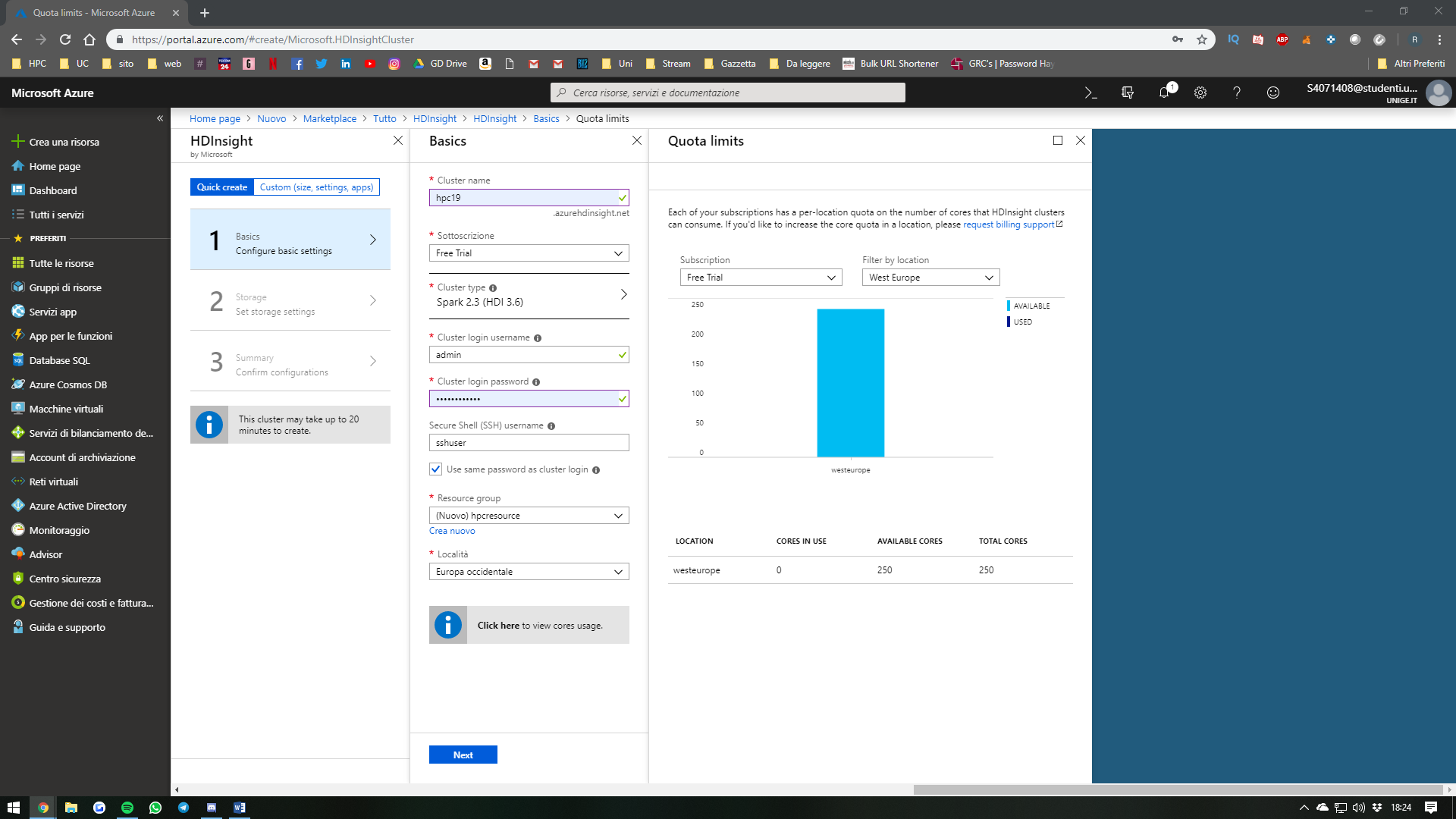
* Using spark-submit, the user submits an application.
* In spark-submit, we invoke the main() method that the user specifies. It also launches the driver program.
* The driver program asks for the resources to the cluster manager that we need to launch executors.
* The cluster manager launches executors on behalf of the driver program.
* The driver process runs with the help of user application. Based on the actions and transformation on RDDs, the driver sends work to executors in the form of tasks.
* The executors process the task and the result sends back to the driver through the cluster manager.

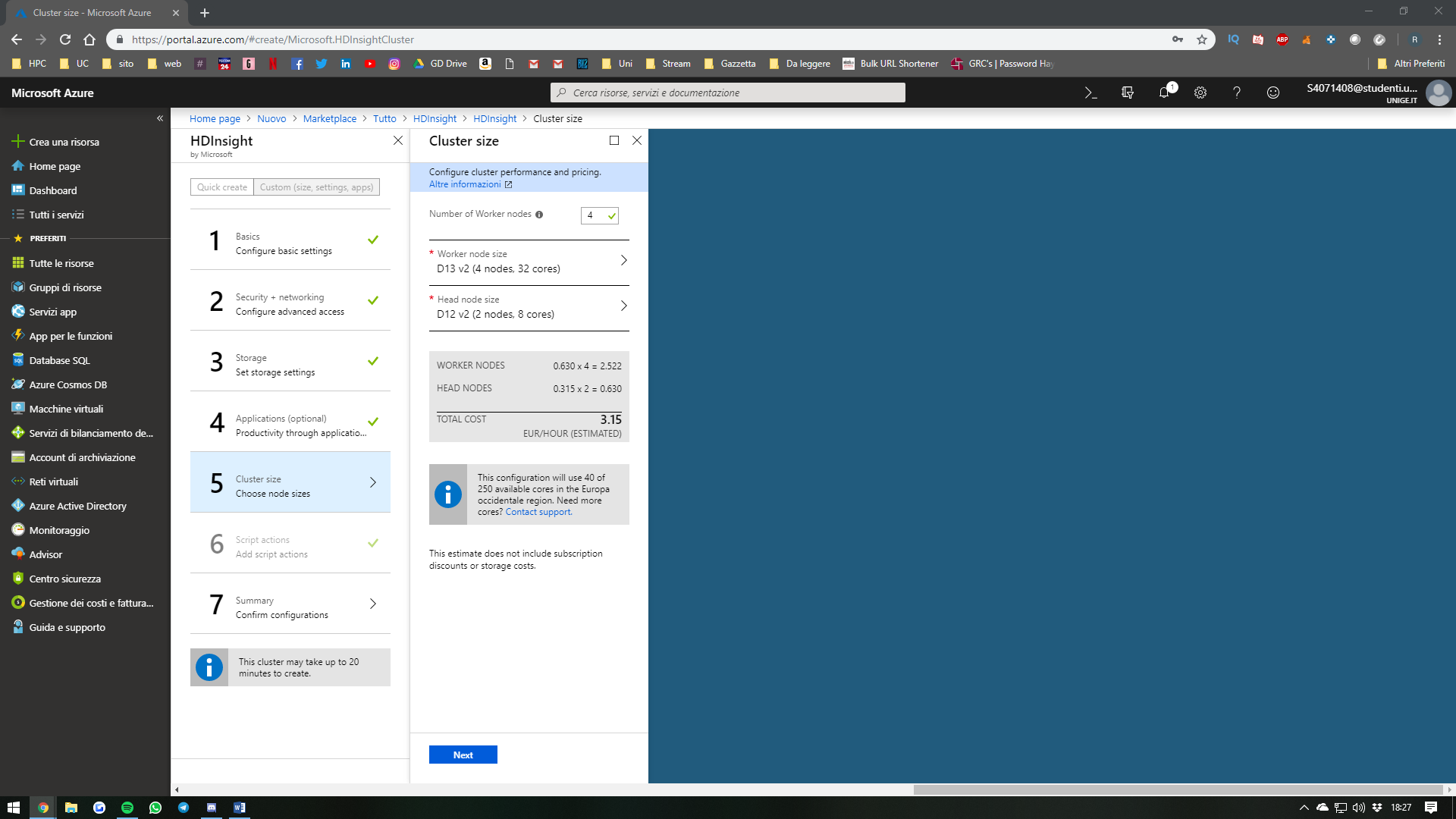
**Differences between Spark and PySpark**

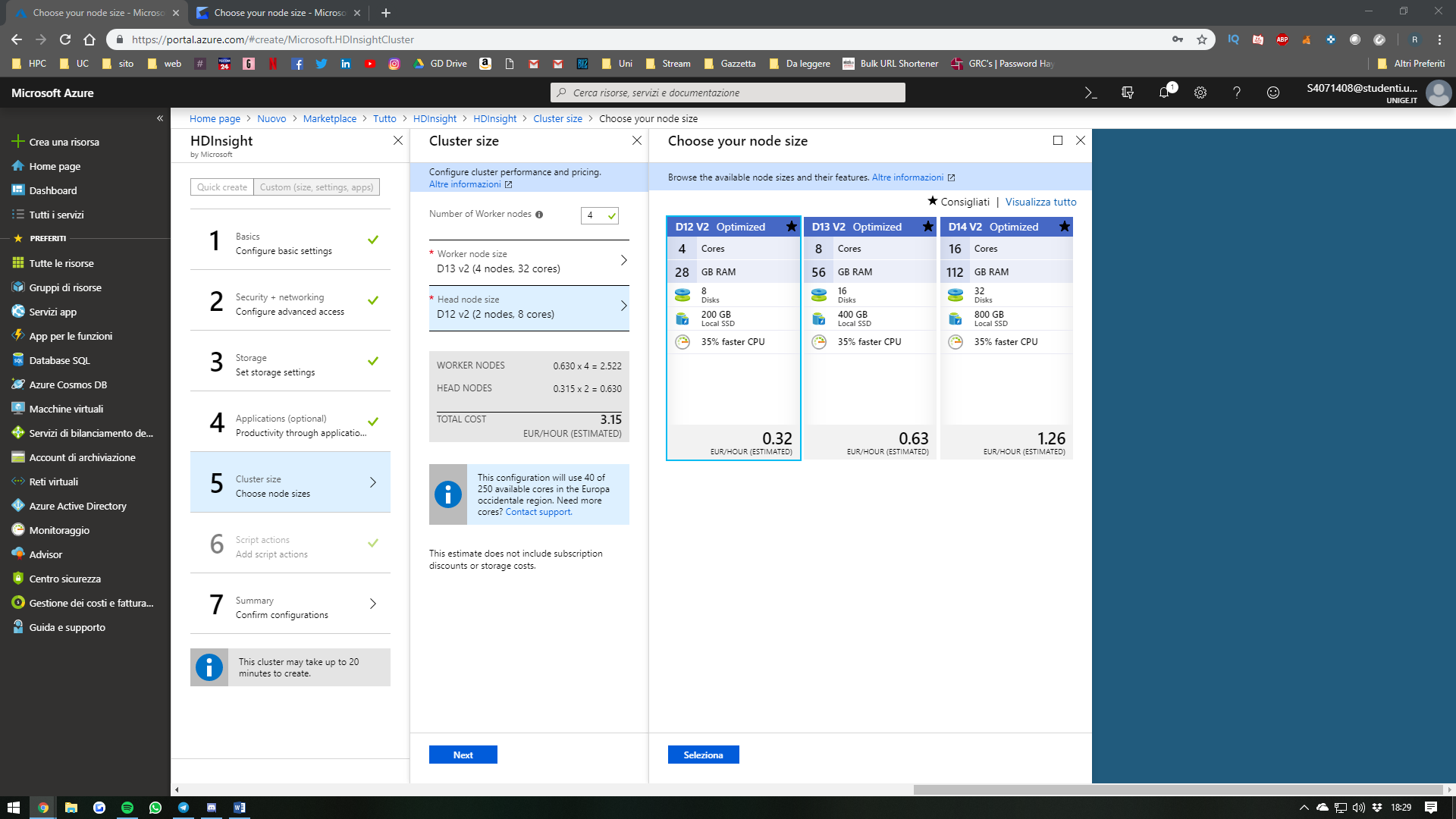
Spark refers to the Apache Spark distributed computing framework, originally accessible using the Scala programming language. PySpark is the interface that gives access to Spark using the Python programming language. Another alternative to PySpark would be SparkR, which understands the R language.

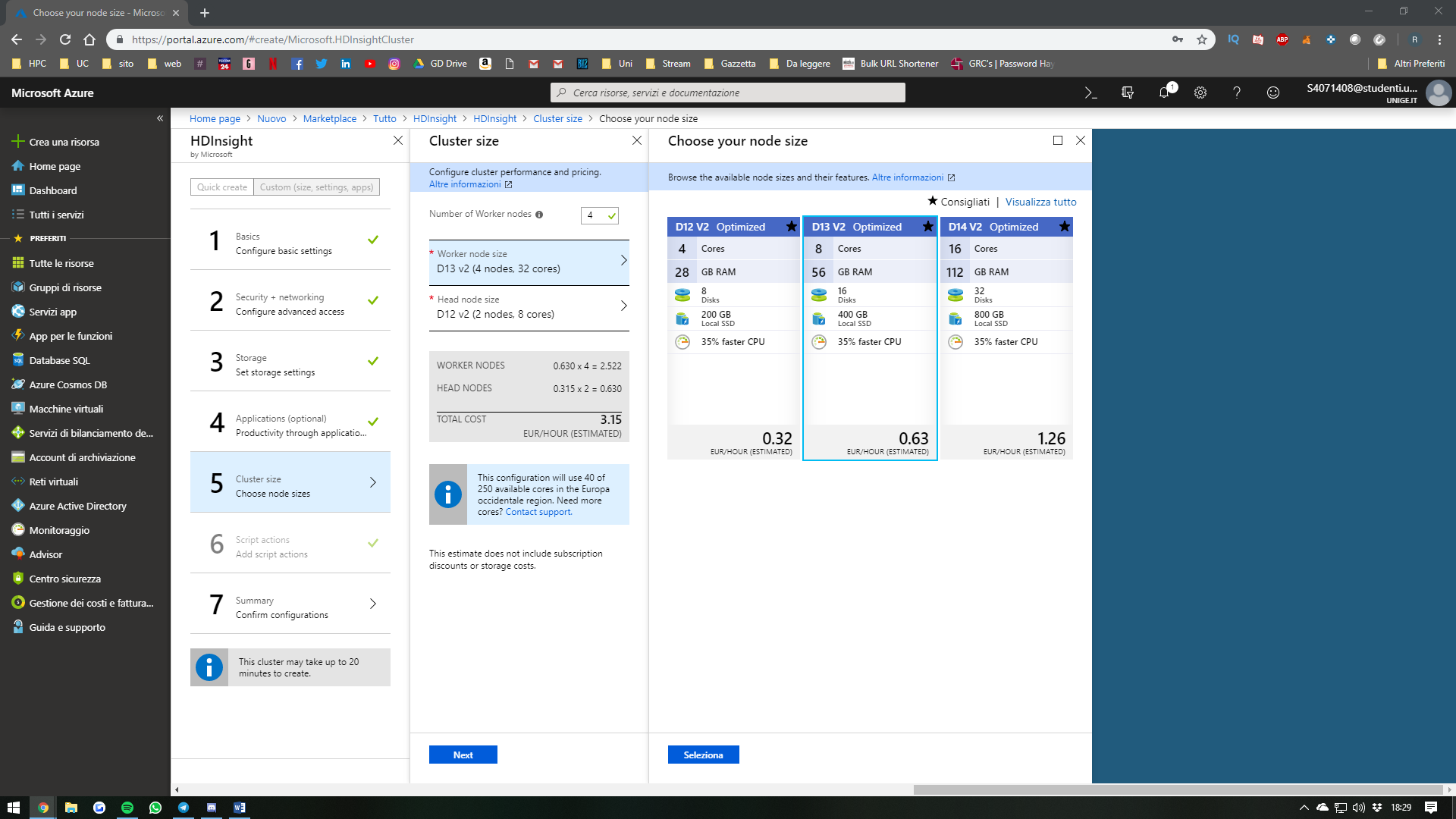
**Azure**

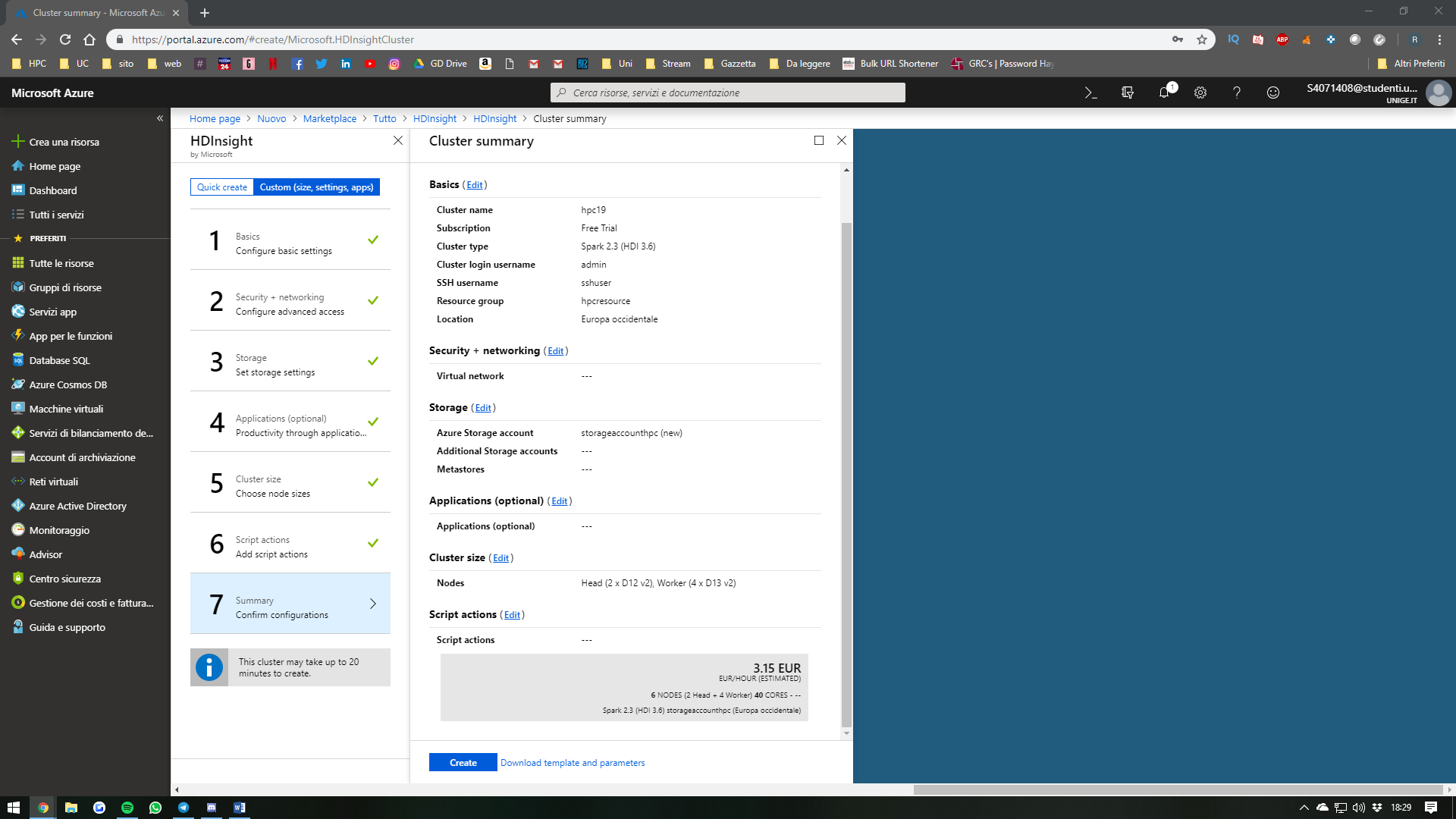
Cluster creation



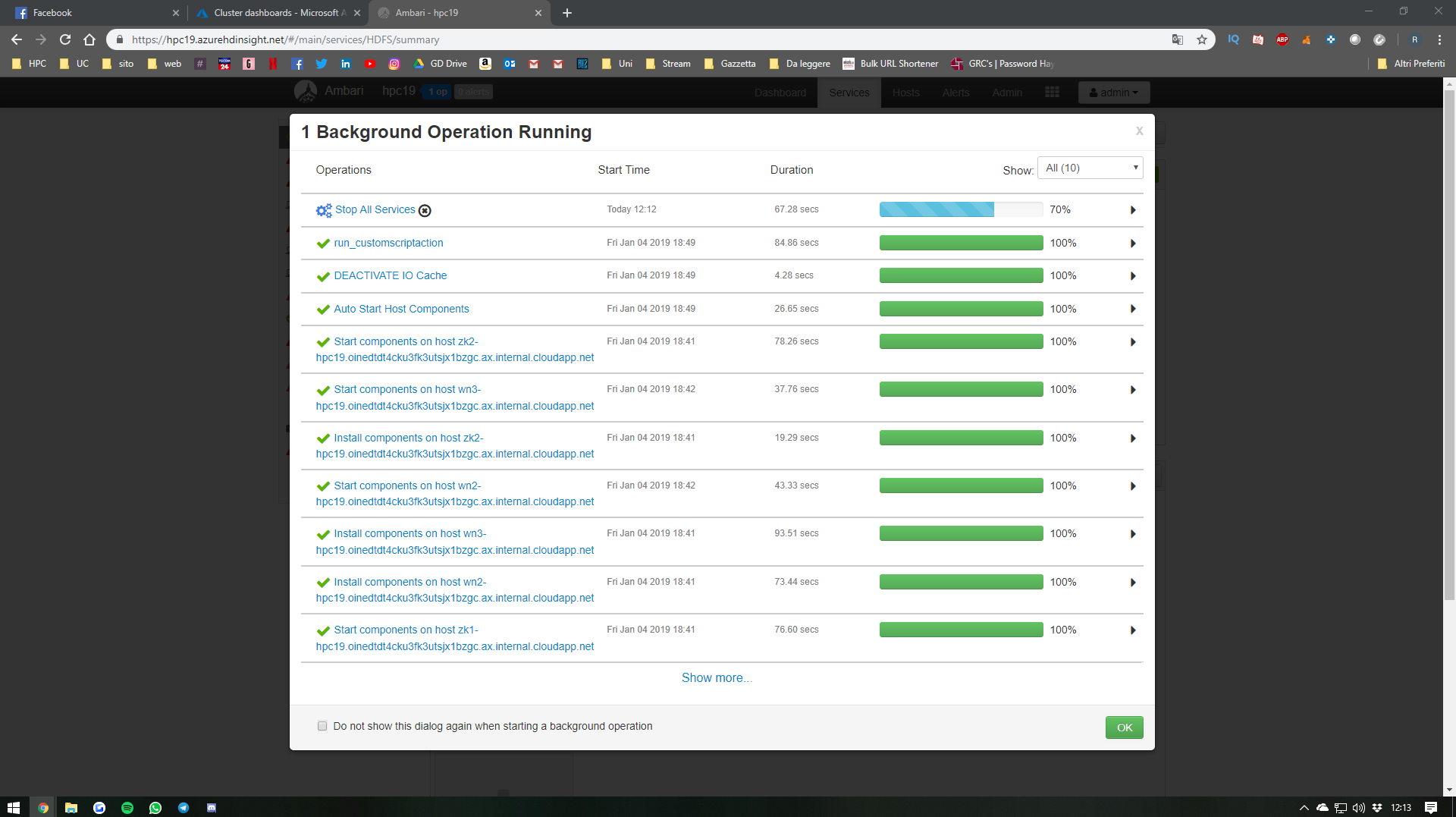




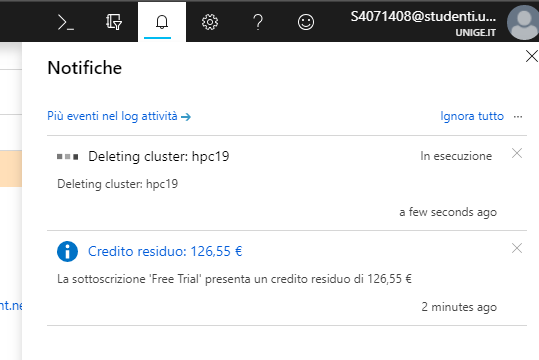




Stop services



Delete cluster



**References**

* https://data-flair.training/blogs/how-apache-spark-works/