# AWS Cloud Computing Introduction - Report

Warsaw University of Technology
Faculty of Power and Aeronautical Engineering

***Author:***
Krzysztof Wilczyński

***Supervisor:***
Mateusz Żbikowski PhD

February 7, 2019

Please note that this document is a copy of the repository's overview page (LINK). Furthermore, the github documentation page will be updated more frequently and is likely to contain more practical information (such as links to the useful resources).

## 1 Steps taken

- Applied for the AWS Educate financing program to get 40$ cloud experiments financing
- Studied all of the resources provided by the lecturer in order to gain a basic theoretical knowledge about the vast possibilities of AWS (which gives us much more than just parallel computing)
- Installed Anaconda, AWS CMD Tools etc. on my Linux machine.
- Created several AWS EC2 (Amazon Web Services Elastic Coputer Cloud) machines in different availability zones just for fun - to test the possibilities of the nodes and their operating systems.
- Created Key Pairs allowing for remote connection to the EC2 instances (those files are not in the repository for safety reasons - there are bots scanning github for such keys to run unauthorized apps such as Bitcoin miners on someone else's machines, thus generating incredibly high costs).
- Connected to the EC2 instances using the following pattern (Linux):

  **Code example 1:**

  ```
  $ chmod 400 <private-key-path>/my-key-pair.pem
  $ ssh -i <private-key-path>/my-key-pair.pem
      <ec2-user>@<ec2-node-public-dns-adress>
  ```

  Where:

  - private-key-path is the directory on the local system containing the SSH private key generated when creating "Key Pairs" in the AWS EC2 console.
  - ec2-user depends on the operating system running on the node (for Ubuntu linux, the login and password defaults to ubuntu:ubuntu)
  - ec2-node-public-dns-adress is the public dns adress assigned to the node (look it up in the EC2 management console)
- Deployed some simple helloworld / pi approximation scripts on a single EC2 machine (under native python shipped with Ubuntu Linux).
- Read the publication on UC Berkeley's Biostat statistical computing cluster software and made an attempt to deploy one at AWS using the instructions provided in the tutorial repository (see: Useful Resources section). The project was abandoned as too ambitious - I have no sufficient R language knowledge, it was decided to stick to Python.

- Made serveral attepmts, wasting a lot of time in order to deploy a High Performance Computing (HPC) cloud (see the next section for details)
- After failing to find any help regarding the HPC cloud deployment, a change of approach was needed - it was decided to go with the Apache Spark cloud environment deployed on three AWS EC2 machines: one "master" and two "worker" nodes.
- Configured the simple Apache Spark Cluster (release emr-5.20.0) according to the instructions provided in the "Useful Resources" section of this report.
- Installed the Apache spark client libraries (from here) within the Lunux host's PATH.
- Created the needed environmental variable:

  **Code example 2:**

  ```
  HADOOP_CONF_DIR =
      ~/<my-local-configuration-files-directory>
  ```

  Where:

  – my-local-configuration-files-directory is a path to a folder containing hadoop configuration files on my local Linux machine.

- Created a SSH tunnel to the Apache Spark AWS cluster "master node" (one of the three EC2 machines that is in control of the "workers") using dynamic port forwarding as follows:

  **Code example 3:**

  ```
  $ ssh -i <private-key-path>/mykeypair.pem -D <port>
      -N hadoop@<spark-master-node-public-dns-adress>
  ```

  Where:

  – private-key-path is the directory on the local system containing the SSH private key generated when creating "Key Pairs" in the AWS EC2 console.

  – port a local machine's port used to create the SSH tunnel

  – spark-master-node-public-dns-adress is the public dns adress assigned to the spark cluster's master node (it can be checked in the AWS EC2 management console - it should usually be in a different network than the worker nodes).

  –

- Deployed a sample job (PI number digits calculations based on Monte Carlo method) that I found online to the Apache Spark Cluster configured in the previous steps. The deployment is executed as follows (providing the spark library is present in local machine's PATH environment):

**Code example 4:**

```
$ spark-submit --class <class> --master <master>
    --deploy-mode <mode> --py-files <py-path>
    <script> <args>
```

Where:

- class is the entry point of the application (main class)
- master is the adress URL of the Apache Spark cluster's master node (spark://spark-master-node-public-dns-adress)
- deploy-mode is set to either "cluster" or "client" depending on wheather the script should be started on the cluster's worker nodes or the local client.
- py-files is the search path for the python scripts and dependencies on the local machine (useful for complex python projects)
- script is the python scipt file (.py) to be executed
- args are the space-separated arguments that will be passed to the main method of the script specified in the previous step

## 2 Encountered problems

In the first attempts, many EC2 nodes were deployed in different availability zones in order to properly deploy a High Performance Computing (HPC) cluster. The clusters could only be deployed in US-West and US-East zones but despite trying out every region and many configuration files, all of the attempts were failures. The tutorial (HPC tutorial) provided by AWS contained links to HPC configuration files that were not accesible (such as this). Other configuration JSONs proved to cause unrealistically high costs. Many hours were wasted trying to find a solution and eventually, the decision was to change the approach to AWS Apache Spark cluster.

No financial help (Amazon Educate) has been provided despite many attempts to fill in the forms and contacting the AWS support. This was a serious limitation, as the 40$ grant could really help in deploying experimental clusters.

A lot of time was wasted trying to access the cluster WEB services provided by Apache Spark on the master node using FoxyProxy Chrome addon. In the end, this was not really needed...

When I finally managed to set up an Apache Spark cluster, it was already a very busy time in the semester - all the other projects and exams made it impossible to continue the struggle and write something amazing - it is the plan for the future though! (Todo) :)

## 3 Plans for the future

1. Try again to configure HPC and deploy a Python application there.
2. Learn R statistics language and deploy the UC Berkeley Biostat cluster on AWS.
3. Make a comparison between parallel computing on a single machine and a cluster of cores - the result should be different depending on the complexity of the calculations. It is expected that the computing will actually take longer on the cluster in case of simple and short task (as the nodes have to communicate via sockets - that takes time). On the other hand, the more complex task should be much faster using a cluster such as HPC as the computation can be dissipated between many more cores than in case of a single machine.