Application of Pyspark on AWS Cloud to a simple heat transfer problem

Arkadiusz Rybski, Juliusz Neuman
Cloud Computing 2020/21

Specification of heat transfer problem

1. As an example problem to be solved in this tutorial, a basic heat transfer issue was chosen. Solved problem was taken from Polish Physics Olympiad. It is a numeric task, which was given for the first stage of the contest: Task 4. This task was taken as a starting point and heat distribution in time was calculated.
2. Heat transfer problem is well suited for cloud computation since every time step can be easily divided into multiple groups of points for individual calculations. After each step all data should be gathered into a new time step heat map and divided again for subsequent calculation.

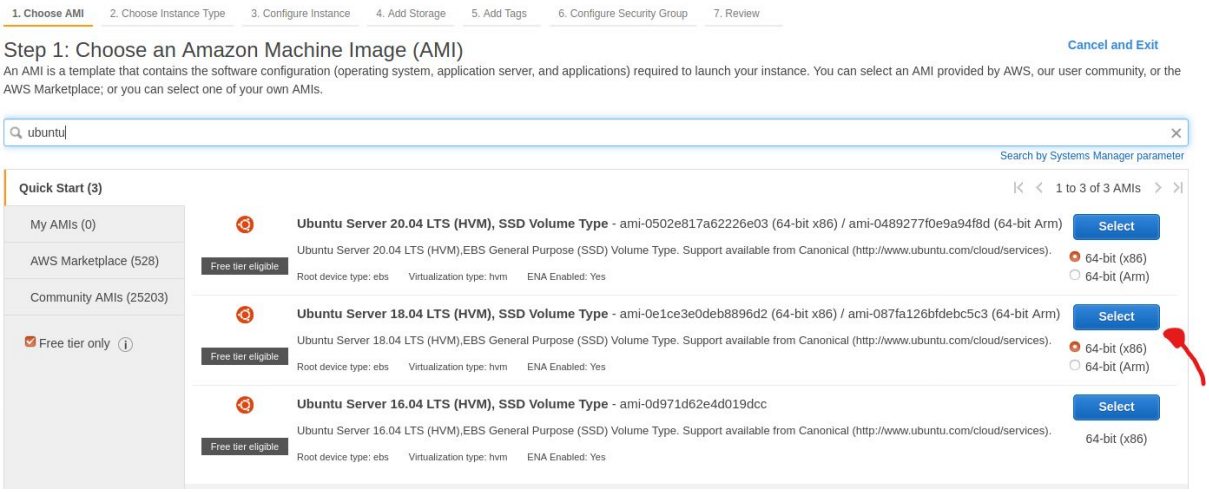
Setting up cluster on AWS with Apache Spark

This tutorial assumes that the user is familiar with the basics of Bash scripting language (Bash tutorial) and has an AWS account.

1. Creating AWS Instance

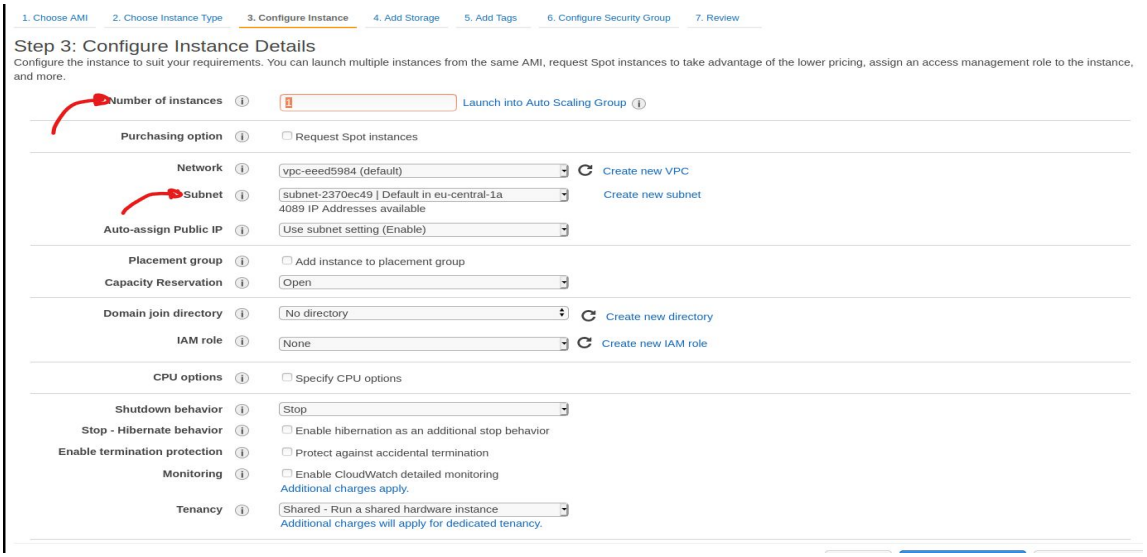
   Launching an AWS instance is described on aws page: launch instance, below we present images showing all needed settings(if not on the picture, it could be chosen as in the tutorial placed above) to set up the cluster.

   a. Choosing AMI

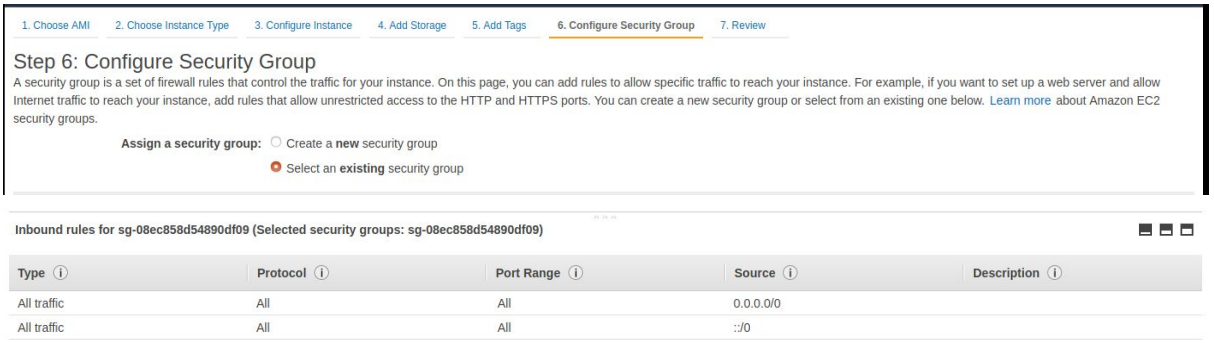


   b. Configure Instance

   Number of instance = number of required nodes + 1 (for master node)
   It is important to create all instances in the same subnet.



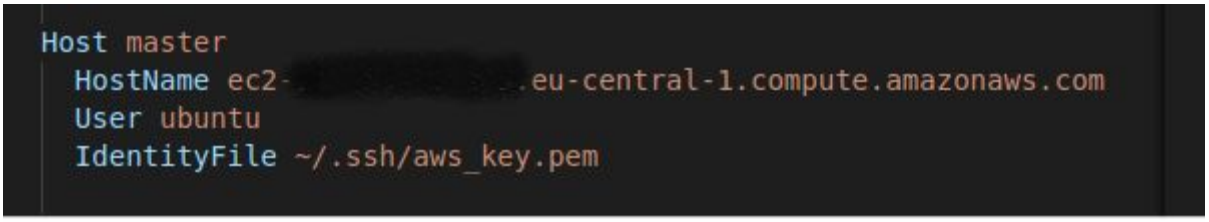
c. Configure Security Group



2. SSH config file

This point is not required but it is recommended so as to make work easier in the future. To avoid passing key it is recommended to use ssh config file all information Using the SSH Config File | Linuxize.

Sample config file:



3. Installing Spark on each node, based on Spark Install

a. Prerequisite setup

i. On **each node** it is needed to make updates. Using shell we type: *sudo apt update*

ii. On **each node** install Java and Scala

sudo apt install openjdk-8-jre-headless

sudo apt install scala

To check if scala/java was installed it is possible to use flag -version

```
ubuntu@ip-172-31-29-230:~$ java -version
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-Ubuntu-0ubuntu1.18.04)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-Ubuntu-0ubuntu1.18.04, mixed mode, sharing)
```

b. Setting up keyless ssh

i. On **master node** install openssh-server:
sudo apt install openssh-server openssh-client

ii. Create RSA key pair in .ssh folder
cd .ssh
ssh-keygen -t rsa -P “”

```
ubuntu@ip-172-31-29-230:~/.ssh$ ssh-keygen -t rsa -P ""
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ubuntu/.ssh/id_rsa): id_rsa
Your identification has been saved in id_rsa.
Your public key has been saved in id_rsa.pub.
The key fingerprint is:
```

iii. Copy id_rsa.pub into ~/.ssh/authorized_keys on **each node**. To show content of id_rsa file use:
cat ~/.ssh/id_rsa.pub
Content of the file can be copied manually (for copying in terminal use ctrl + shift + c)

iv. Check if it is possible to ssh from master to slaves. It is also possible to create SSH config file.

c. Spark installation

i. On **each node** install Spark:
wget https://archive.apache.org/dist/spark/spark-2.4.3/spark-2.4.3-bin-hadoop2.7.tgz

ii. Extract and move files to /usr/local/spark
tar xvf spark-2.4.3-bin-hadoop2.7.tgz
sudo mv spark-2.4.3-bin-hadoop2.7/ /usr/local/spark

iii. Add spark/bin to path variable
nano ~/.profile
add to the end of the file : *export PATH=/usr/local/spark/bin:$PATH*
source ~/.profile

d. Master and slaves configuration

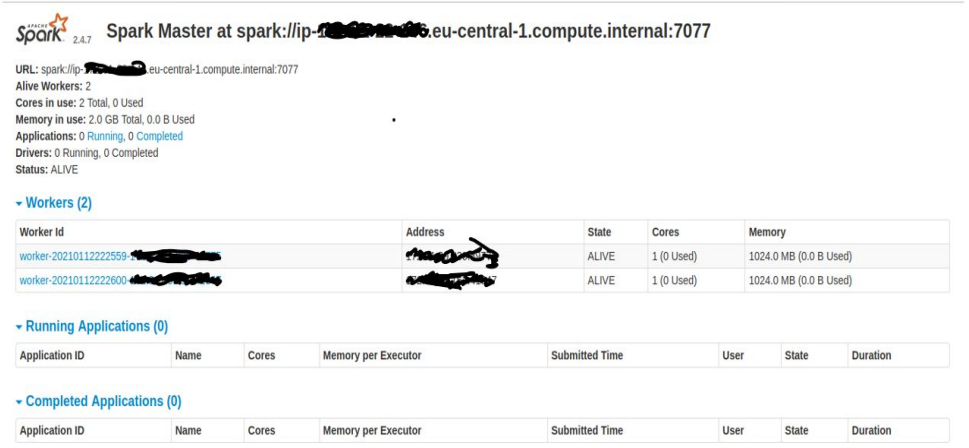
i. Copy and modify /usr/local/spark/conf/spark-env.sh file
cp /usr/local/spark/conf/spark-env.sh.template /usr/local/spark/conf/spark-env.sh
Add to the end of the spark-env.sh file:
export
SPARK_PUBLIC_DNS="ec2-XXXXXXXXXX.eu-central-1.compute.amazonaws.com"
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
For PySpark use
export PYSPARK_PYTHON=python3

ii. copy and modify /usr/local/spark/conf/slaves file
cp /usr/local/spark/conf/slaves.sh.template /usr/local/spark/conf/slaves.sh
To the end of saves.sh file add dns of slaves nodes.

iii. To the test if the configuration worked type:

bash /usr/local/spark/sbin/start-all.sh

Now all workers and master should start, it is possible to check it in browser searching in browser: localhost:8001



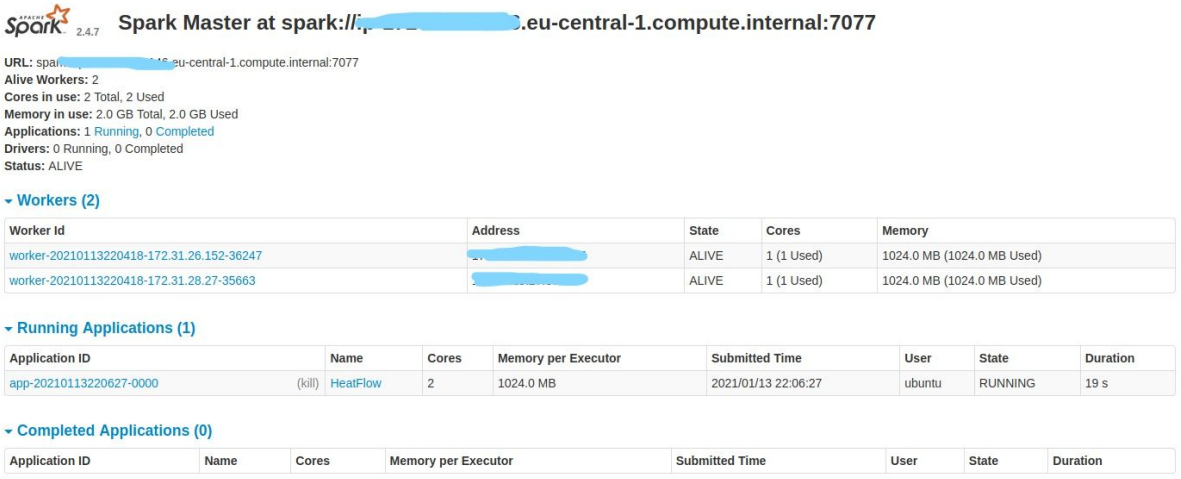
If that does not work, check the logs file which could be found in /usr/local/spark/conf/logs folder.

4. Running pyspark scripts.

To test if pyspark works properly it is possible to submit a sample application.

spark-submit --master spark://ip-XXX-XX-XX-XX.eu-central-1.compute.internal:7077 /usr/local/src/spark/examples/src/python/pi.py 100

After running pyspark script the similar image should appear on your screen:



5. Preparing code.
   a. Clone repo: https://github.com/superjulek/cloud-computing
   b. Follow instructions given in README.md file

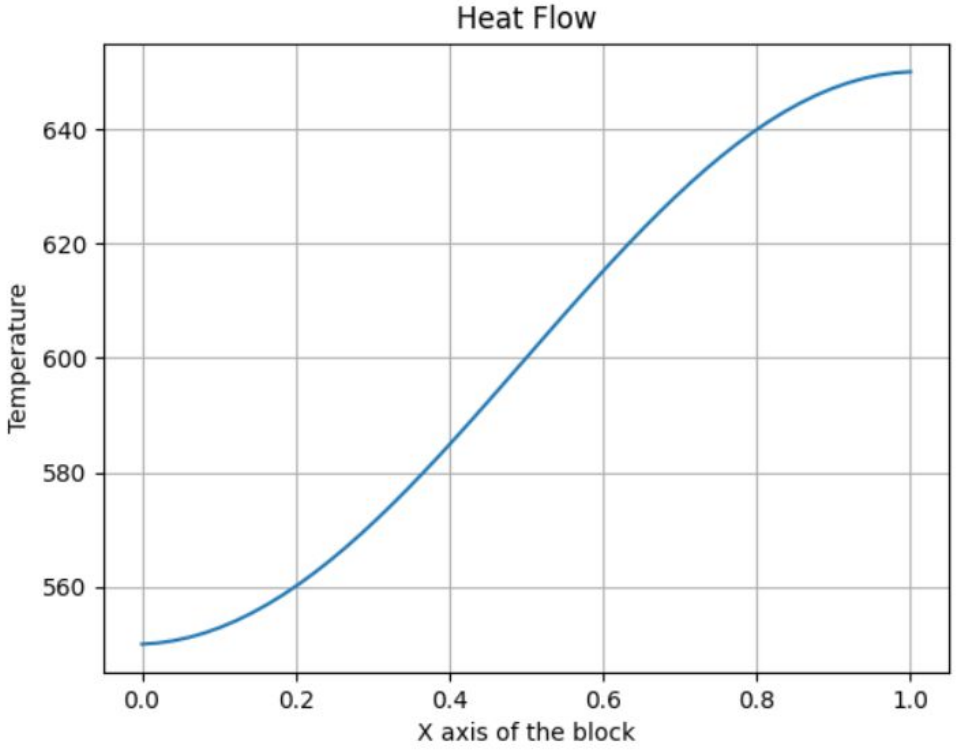
Code description

1. The code was written in Python due to its simplicity

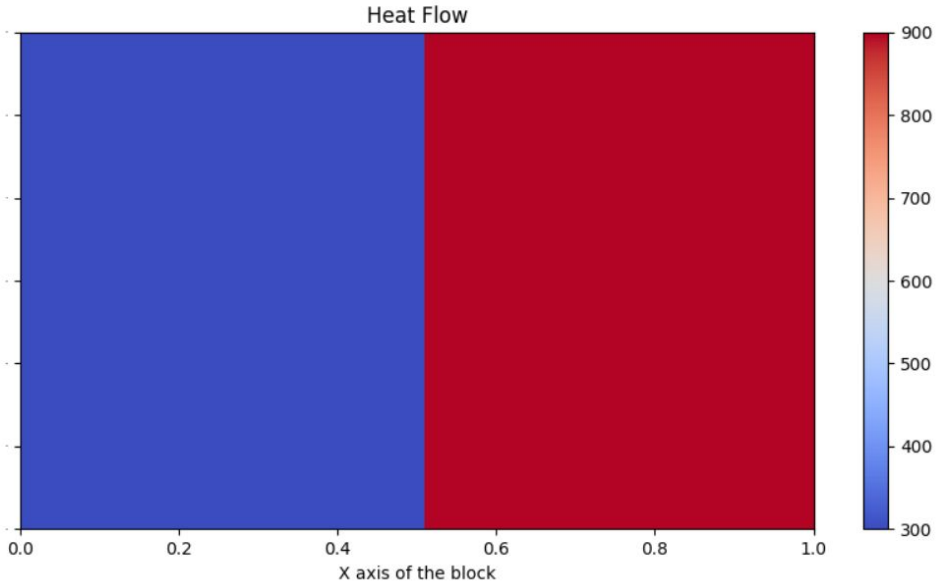
2. By no means is the code meant to be the fastest possible solution of the discussed problem, it was written in order to present pyspark parallel computing possibilities.
3. The code is split into separate files for clarity.
4. There is a bash script in the main directory called ‘run.sh’, starting computation and passing additional, optional parameters. It is meant for local runs.
5. Program starts in the ‘main.py’ file, where optionals arguments are parsed.
6. Then the ‘run’ function of the ‘run.py’ file is called.
7. Problem config is written in ‘config.py’ file.
8. Mesh is described in ‘mesh.py’ file and it is made of points described in ‘point.py’ file. This is where heat flow calculations are described.
9. Per every time step mesh of points is distributed among workers, which compute new temperatures in every point. Then all data is gathered back, rewritten and distributed again for next time step computation.
10. Computation runs until the expected condition is fulfilled. All temperature data is stored and can be used for further analysis.

Results

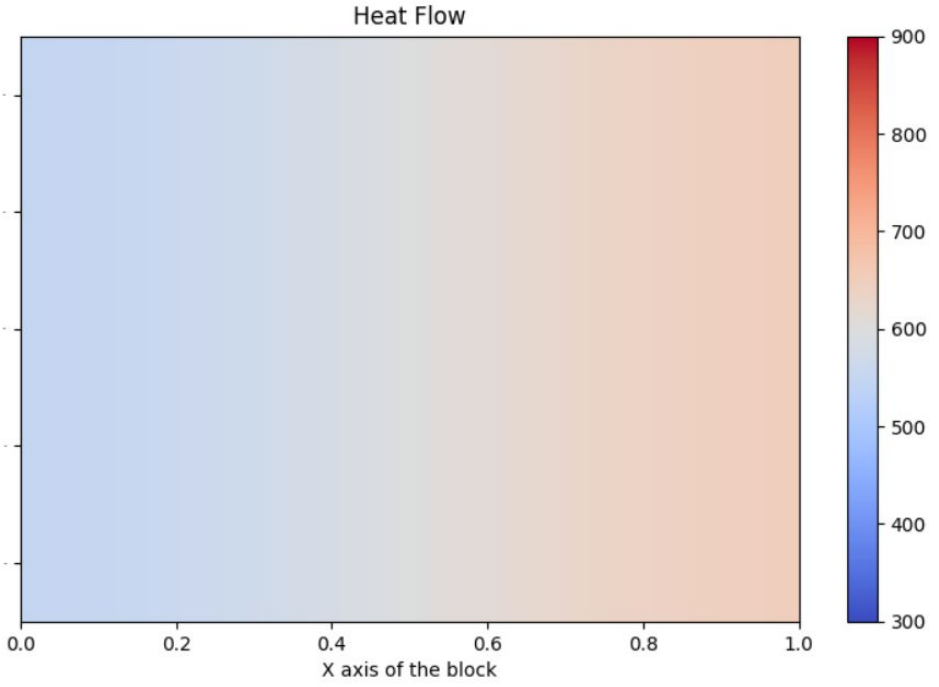
1. Calculation was run twice, with different time step end space delta.
2. It was observed that time division should not be much different then space division, otherwise huge errors in gradient calculation occur.
3. Chart showing final temperature distribution:



4. Heat map at the beginning of the simulation



5. Heat map at the end of the simulation



Summary

1. Pyspark is an efficient tool for running parallel computation on clusters.
2. AWS cluster gives great opportunity to familiarize cloud computing.