

BDP1 Project: Building Cloud IaaS infrastructures

Configuration of an HTCondor cluster on Amazon Web Service for NGS reads alignment

Andrea Sambugaro

October 2020

1 Introduction

An Infrastructure-as-a-Service (IaaS) is a model in which different virtualized hardware resources are made available, so that the users can create and manage their own infrastructure on the cloud, without worrying about where resources are allocated.

In our case I want to build an infrastructure and configure a batch system that manage three different nodes to perform the alignment of sequencing reads against a reference human genome (Figure 1).

The Amazon cloud service provider Amazon Web Service (AWS, <https://aws.amazon.com/>) was used for this project. I configure a batch system using HTCondor, a specialized workload management system for compute-intensive jobs. Our infrastructure is based on an Red Hat Linux distribution (RHEL-7.6-HVM-GA-20190128-x86-64-0-Hourly2-GP2). All the three instances (a Master node (bdp-master) and two worker nodes (bdp1.WN1, bdp1.WN2)(Figure 1, Figure 2) share the following properties: 1) Instance type: t2.large (<https://aws.amazon.com/it/about-aws/whats-new/2015/06/introducing-a-new-amazon-ec2-t2-instance-t2-large/>) 2) A SSD volume of 10 GB (volume type gp2)

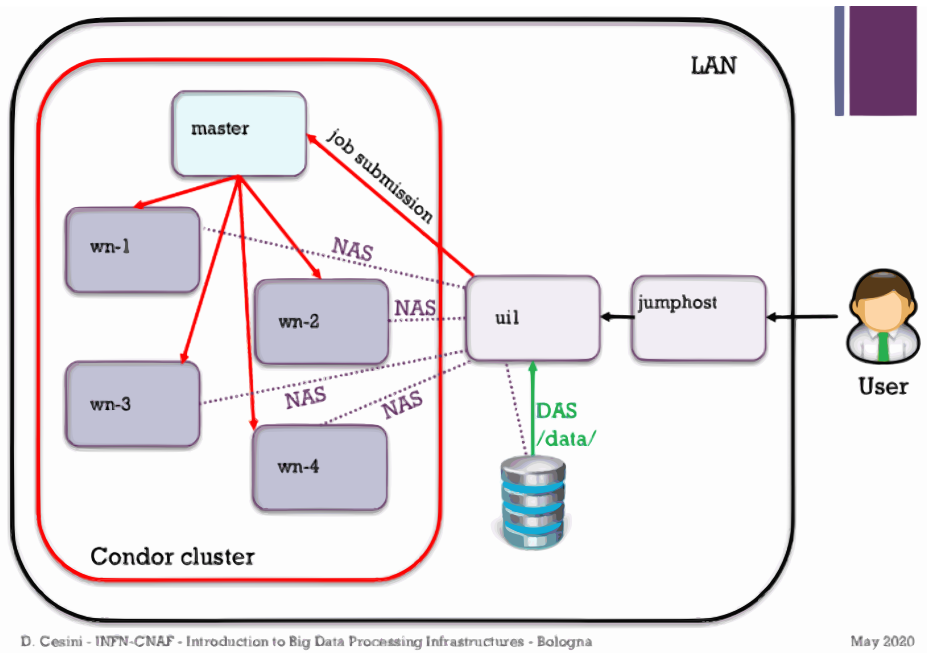


Figure 1: Graphical representation of our cloud infrastructure with condor cluster.

To allow the communication between all the nodes I configure all the security groups editing the inbound rules in order to create a cluster of nodes. Security Group must allow TCP for ports 0-65535 from the same security group, i.e.:

- All TCP (TCP 0-65535, sg-008742ba0467986fe)
- Security group must allow ping from the same security group
- All ICMP-IPv4, all N/A sg-008742ba0467986fe
- Security group must allow ssh on port 22 from everywhere

In order to provide communication between the various nodes of the cluster, they have been instantiated from the same availability zone (us-east-1b).

In addition I created a magnetic volume of 100 GB that was attached to the master node but then it was shared among all the nodes. To do this I move from a DAS (Direct Attached Storage) to a NAS (Network Attached Storage). I configured the master node as the server (using a nfs-utils) exporting the data to the destination host IP, the two remaining nodes (worker nodes, the clients): in this way all the nodes are able to access the shared volume of 100 GB with data (Figure 3).

Istanze (3) Informazioni							
<input type="text" value="Filtra istanze"/>							
<input type="checkbox"/>	Name	ID istanza	Stato dell'ist...	Tipo di ista...	Verifica dello stato	Stato dell'...	Zona di dispon...
<input type="checkbox"/>	bdp1_master	i-0465a5a97dd01c7c5	Arrestato	t2.large	-	Nessun ...	us-east-1b
<input type="checkbox"/>	bdp1_WN1	i-06cd0f2b2bd5d4a2a	Arrestato	t2.large	-	Nessun ...	us-east-1b
<input type="checkbox"/>	bdp1_WN2	i-0556b4b6cce34f885	Arrestato	t2.large	-	Nessun ...	us-east-1b

Figure 2: Window with configured instances.

Create Volume Actions										
<input type="text" value="Filter by tags and attributes or search by keyword"/>										
<input type="checkbox"/>	Name	Volume ID	Size	Volume Type	IOPS	Snapshot	Created	Availability Zone	State	Alarm Status
<input type="checkbox"/>		vol-0535fd97...	10 GiB	gp2	100	snap-0fb6bef4...	July 13, 2020 at 8 3...	us-east-1b	in-use	None
<input type="checkbox"/>		vol-0e54319...	10 GiB	gp2	100	snap-0fb6bef4...	May 14, 2020 at 4:0...	us-east-1b	in-use	None
<input type="checkbox"/>		vol-0858998...	10 GiB	gp2	100	snap-0fb6bef4...	May 12, 2020 at 4:3...	us-east-1b	in-use	None
<input type="checkbox"/>		vol-0c90c93...	100 GiB	standard	-	snap-09ee52d...	May 11, 2020 at 5:1...	us-east-1b	in-use	None

Figure 3: Disks configured for the respective instances and magnetic disk with data.

2 HTCondor cluster configuration

2.1 Master node

First of all I have installed and configured HTCondor (Thain D., Tannenbaum T., and Livny M., 2005) in the master node. The master node is the central manager, the machine that does match-making between available machines and submitted jobs. In our case the master node was also configured as worker node.

The ***sudo systemctl status condor*** command allows you to check the correct installation of HTCondor on the machine:

```
[ec2-user@ip-172-31-23-93 ~]$ sudo systemctl status condor
* condor.service - Condor Distributed High-Throughput-Computing
   Loaded: loaded (/usr/lib/systemd/system/condor.service; enabled; vendor preset: disabled)
   Active: active (running) since Sun 2020-10-25 09:50:18 UTC; 19s ago
     Main PID: 4429 (condor_master)
    Status: "All daemons are responding"
       Tasks: 8 (limit: 32767)
      Memory: 167.9M
    CGroup: /system.slice/condor.service
            |-4429 /usr/sbin/condor_master -f
            |-4747 condor_procd -A /var/run/condor/procd_pipe -L /var/log/cond...
            |-4750 condor_shared_port -f
            |-5224 condor_collector -f
            |-6363 condor_negotiator -f
            |-6364 condor_startd -f
            |-6365 condor_schedd -f
            ~-6732 kflops
```

Figure 4: Output of the ***sudo systemctl status condor*** command.

Then I appended these lines to the **condor_config** file for completing the configuration:

```
# Master Node IP
CONDOR_HOST = <Master_Node_private_IP>
# Master Node config
DAEMON_LIST = COLLECTOR, MASTER, NEGOTIATOR, SCHEDD
HOSTALLOW_READ = *
HOSTALLOW_WRITE = *
```

```
HOSTALLOW_ADMINISTRATOR = *
```

Once the condor configuration has been completed, it has been restarted with the following command: ***sudo systemctl restart condor***.

Finally, the Master node was set up as NFS server in order to allow data access to the 100 GB magnetic disk attached to the master node also to worker nodes. Using ***fdisk*** was initialized the partition and an ext4 file system was created. Even the ***fastb*** file was modified to automatically mount the volume at boot under a new directory data using the following command:

```
</new_volume/partition> /data ext4 defaults 0 0
```

The following lines were appended to the NFS configuration ***exports*** file to expose the data folder with all the data to the Worker nodes:

```
/data <destination_host IP – Worker node 1>(rw, sync, no_wdelay)
/data <destination_host IP – Worker node 2>(rw, sync, no_wdelay)
```

As a last step, the program to perform the test job (Burrows Wheeler Aligner, BWA) has been installed with the following command:

```
yum install -y bwa
```

Now the node has been configured as both master and worker node.

2.2 Worker nodes

Two other machines have been configured in AWS as worker nodes (type: t2.large). As for the master node, HTCondor has been installed in these two machines as well. The following lines were appended to the ***condor.config*** file:

```
# Master Node IP
CONDOR_HOST = <Master_Node_private_IP>
# Worker Node config
DAEMON_LIST = MASTER, STARTD
HOSTALLOW_READ = *
HOSTALLOW_WRITE = *
HOSTALLOW_ADMINISTRATOR = *
```

After installing NFS (using a *nfs-utils*) a new directory */data* was created and the ***fastb*** file was modified to enable the access to the shared volume. The volume will be mounted under the */data* directory. Lines added to the ***fastb*** file:

```
<SERVER IP–Master_Node_private_IP>:/data /data nfs defaults 0 0
```

Then the *bwa* application was installed with the following command:

```
yum install -y bwa
```

Once the communication between the different nodes has been allowed the batch system was successfully installed. The Figure 5 shows the output of the command *condor_status*.

As you can see there are the 3 machines (master node and the two worker nodes) in the cluster, two cores each other.

3 Test Job submission on the HTCondor cluster

Our purpose is to run the Burrows Wheeler Aligner package (Li H. and Durbin R. 2009) on the HTCondor cluster implemented on the AWS cloud service.

BWA allows you to align short sequencing reads against a large reference sequence such as the human genome. Specifically, our test job consists of aligning 5 FASTA files containing 1000 reads each (Figure 6).

```
[ec2-user@ip-172-31-23-93 ~]$ condor_status
```

Name	OpSys	Arch	State	Activity	LoadAv	Mem	ActvtyTime
slot1@ip-172-31-19-48.ec2.internal	LINUX	X86_64	Unclaimed	Idle	0.000	3909	0+00:00:03
slot2@ip-172-31-19-48.ec2.internal	LINUX	X86_64	Unclaimed	Idle	0.000	3909	0+00:00:31
slot1@ip-172-31-23-93.ec2.internal	LINUX	X86_64	Unclaimed	Idle	0.000	3909	0+00:00:03
slot2@ip-172-31-23-93.ec2.internal	LINUX	X86_64	Unclaimed	Idle	0.000	3909	0+00:00:28
slot1@ip-172-31-27-244.ec2.internal	LINUX	X86_64	Unclaimed	Idle	0.000	3909	0+00:00:03
slot2@ip-172-31-27-244.ec2.internal	LINUX	X86_64	Unclaimed	Idle	0.000	3909	0+00:00:31

```

Machines Owner Claimed Unclaimed Matched Preempting Drain
X86_64/LINUX      6      0      0      6      0      0      0
Total             6      0      0      6      0      0      0
[ec2-user@ip-172-31-23-93 ~]$
```

Figure 5: Master node and the two Worker nodes in the cluster.

```

>chr1:read 981
ctctggtagcagcgccagcagcattgtcacaacaaagttagaaaaaagacGAGAGCGCAGagagactatacacaggccacacacatttggctaatttttaaatattctgtagagacaaggtcttctgaggttgccacggtatgtctaaac
>chr1:read 982
tcttggcattcaggtggcgatggttgctcattgttgtaatcgagcactttgggaagctaaaggcaggcaaatcactggaagctggagttcgagaccagcctggccaacagggtgaaacttgactctatacaaaatcacaaaatcagct
>chr1:read 983
ggcgactagtgtgcttgctctgtactcactctcgggaggtcgaggcaggagaatcaacttgaacctgggaggtggaggttgacgtggaccccatcactgcactccaccctgggtgacagagcgagactGTcaacaacacaaacaaca
>chr1:read 984
acaaaacacaaacacaaacacacaaaaaaactctggcataagacacttctctgtcttagcttcccaagccctgggattataCTGTTTCTATAATTGAACACACTTGTTCTTATACGTTTAAAGTATAAAGGACAAAAAA
>chr1:read 985
acagataatgpcacaaattgttggtaaggccgggcatggtggccgctgtaattccgaactataggagggtcgaggtgggcagatcacttgaggccaggagtatgagaccagcctgggcaacatggtaaatcccaccactacagaaaaatc
>chr1:read 986
taaaaattagccagcagctggttggtgtacactgttaatttccagctaccaggaggctgpaatgagagaatcacttgtcttgggaggtcaaggctcgagctgaactgtgtaggtcatcttgcactggcgctgagagacagagcaagcccc
>chr1:read 987
tatctagaaaaaataatgtcagtgaaggttggaggaaattggaaccacaaacatctactggtgggaacataaaattgtgtaacatttctgttgggtatttcttGTCTTCATTTTAATTGGCATTTTAAAAAATCAagcggggttt
>chr1:read 988
cacattcttgcccagcgtggtcttgaatcaagggtcgaagccatctctcctagctgagcctctctgagtgcagggtgggattacaggctgtgagccattgcaccaactcggtatagccagcttagaaaaactctggcgatttctcaaaaggcta
>chr1:read 989
aaatgcagctcatctataatgcaacaatttcaactctcaggcatatattcccaaaaaataaaatatgtccacacaaaaacttgtacaacaattctcatagcagcattattcataatgaccaatcacatggaaatcatggaaaacaccc
>chr1:read 990
tcatatctacacacagatgtagacagataaacaataagctgaggtgctctaccatggaaactgtgccatagaaggaattgaaatttgatcacacatagacataaaggaacttggaaaactcgtgtgaggggaaaaaaggccacaaagaa
>chr1:read 991
tcacatattgtcaactctatttcttcagatagpccaactatagtgtaagcaaatataacaaatggttgctcaatgcctgaaggctggggccaaggtgagtggggagatgaggagtaggtgcttgcataaggttatctctataggttaaga
>chr1:read 992
aaggttctcaaaagtgcactgttggtgtgcagtcagcctgtgaaatttctcaaaactcactgaattgcagatttcaataaaatgaagtgtaggttatgtaattttaaataagactataTTAAAAAATTAATAATACGGGCTgtggca
>chr1:read 993
cagggtgtcatgctgctgctgttaactccagcactttgggaggctgaggcaggaggaactcattggagtcaggagttttagcccgactgggagcaacatggcaagactcccgctctcatgataaaaaattagctggacatggtggcacatgtct
>chr1:read 994
gtgtcccaactcattgggagactgaagttagagaaccacttgagccaggagttgaggtctacagtgaacctgatgatgtcactgtactgtagccttaagcaacagagcgaagcgtgtctctgaaaaggaaagaacacaaATGCaagt
>chr1:read 995
tctctctatctctctgtgtagtgtagcgaagtgtggaggagaaatagacaataataaaagagcactgataatgacagtgtaggtggttaggtcaggtgtgctagctaatggtctCTAAAAAATTCATAAAGTTACAGCTCTGGGACAGCTC
>chr1:read 996
ATGTAGTCAARAAGATGAAGCCGAAATTCATTACAATTGCCCATGCTCTTATTACATGCCTCTTACTGAAAAATTCCTAAGTGCCTAAACACCAAGCTTCCAATCATAGCAGCTGTTTATTAAGACTACAAAAAGAAATCGAGcgcg
>chr1:read 997
ggcggtggtgttcacatctgtactcttgaatttgggaggctgaggcaggcagattgcttgaggtcaggagctccagaggagcctggccaacatggtgaaatcccatctctactaaaaatacaaaaaattagctgggtatggttggtggggc
>chr1:read 998
acctgttaatcccaactcactggagggtgagggcaggagaattgttgaaccacaagggtgaaggttgtagctgagccaaactgcacattgcactccagcctgggtgacaagagaaagacattctacttaaaaaaaaagaagaaaaag
>chr1:read 999
aaatTGGCATTTCTTCAGAAGATTACATCGTGTTCATGATAAAGAAGCTCTAATTTTGCATTTGTTCAAGATTGATCAGATTATCCCAATATGACACCATCTTGGATAAATGCAACACACAAATTCATTTTGTCATTAAACAAAC
>chr1:read 1000
CGATTAGTAGTACTCTataataattcgcatcttattaaaaatgacggatgttaaaaaatttggaaatttgaggccaatagattgtacaacctgttccaaaggggaattccaaaatccacacatcttgagaccatcaagtatgatgaa
[ec2-user@ip-172-31-23-93 ec2]

```

Figure 6: Fasta file with 1000 reads

3.1 Burrows Wheeler Aligner application

Burrows Wheeler Aligner is a software package for mapping low-divergent sequences against a large reference genome, such as the human genome. It consists of three algorithms: BWA-backtrack, BWA-SW and BWA-MEM. The first algorithm is designed for Illumina sequence reads up to 100bp, while the rest two for longer sequences ranged from 70bp to 1Mbp. BWA-MEM and BWA-SW share similar features such as long-read support and split alignment, but BWA-MEM, which is the latest, is generally recommended for high-quality queries as it is faster and more accurate. BWA-MEM also has better performance than BWA-backtrack for 70-100bp Illumina reads. For all the algorithms, BWA first needs to construct the FM-index for the reference genome (the index command). Alignment algorithms are invoked with different sub-commands: `aln/samse/sampe` for BWA-backtrack, `bwasw` for BWA-SW and `mem` for the BWA-MEM algorithm.

3.2 Test job

The volume of 100 GB was created from a snapshot volume used during the BDP1 course. It contains different data including NGS reads organized in one directory for each patient that contains around 550 files with 1000 reads each.

The test job was based on the alignment of 5 FASTA files from patient1 to the human genome build (hg19). The BWA package takes the advantage of indexing the entire genome to speed up the alignment process. In our case also the index was already built in the shared volume but during the testing phase due to strange outputs and errors it was appropriate to re-index the genome. The five files are copied from the shared volume to a local folder of the Master node to run our test workflow. First, the **bwa_batch.job** file was create as input of the HTCondor cluster. This file calls as executable the **alignment_reads.py**.

The bwa_batch.job file

```
#####
##### Alignment test #####
#####

#####The program that will be executed #####

executable = alignment_reads.py
read = $(Process)+1
arguments = read.$INT(read).fa

##### Input Sandbox #####

Input = read.$INT(read).fa
transfer_input_files = read.$INT(read).fa

##### Output Sandbox #####

# condor log
Log = read.$INT(read).log

# standard output
Output = read.$INT(read).out

#standard error
Error = read.$INT(read).error

##### condor control variables #####

should_transfer_files = YES
when_to_transfer_output = ON_EXIT

Universe = vanilla

#####

Queue 5
```

The alignment_reads.py file

```
#!/usr/bin/python
import sys,os
from timeit import default_timer as timer

start = timer()
```

```

dbpath = "/data/BDP1_2020/hg19/"
dbname = "hg19bwaidx"
query = sys.argv[1]
out_name = query[:-3]
fa_file = query
samfile = out_name+".sam"
gzipfile = out_name + ".sam.gz"
saifile = out_name + ".sai"
md5file = out_name + ".md5"

print "Input: " , query

command = "bwa aln -t 1 " + dbpath + dbname + " " + fa_file + " > "
+ saifile
print "launching command: " , command
os.system(command)

command = "bwa samse -n 10 " + dbpath + dbname + " " + saifile + " "
+ fa_file + " > " + samfile
print "launching command: " , command
os.system(command)

#Checksums
print "Creating md5sums"
os.system("md5sum " + samfile + " > " + md5file)

print "gzipping out text file"
command = "gzip " + samfile
print "launching command: " , command
os.system(command)

#Timer
execution_time = timer() - start

print "Execution time: " + str(execution_time)
print "exiting"

exit(0)

```

The figure below shows the output of the **condor_q** and **condor_status**.

```
[ec2-user@ip-172-31-23-93 bwa]$ condor_q

-- Schedd: ip-172-31-23-93.ec2.internal : <172.31.23.93:9618?... @ 10/22/20 18:08:07
OWNER   BATCH_NAME   SUBMITTED   DONE   RUN    IDLE   TOTAL   JOB_IDS
ec2-user ID: 89    10/22 18:08   -      5      -      5 89.0-4

Total for query: 5 jobs; 0 completed, 0 removed, 0 idle, 5 running, 0 held, 0 suspended
Total for ec2-user: 5 jobs; 0 completed, 0 removed, 0 idle, 5 running, 0 held, 0 suspended
Total for all users: 5 jobs; 0 completed, 0 removed, 0 idle, 5 running, 0 held, 0 suspended

[ec2-user@ip-172-31-23-93 bwa]$ condor_status

Name                                     OpSys      Arch      State      Activity LoadAv Mem      ActvtyTime
slot1@ip-172-31-19-48.ec2.internal      LINUX      X86_64    Claimed    Busy      0.000 3909  0+00:00:03
slot2@ip-172-31-19-48.ec2.internal      LINUX      X86_64    Claimed    Busy      0.000 3909  0+00:00:03
slot1@ip-172-31-23-93.ec2.internal      LINUX      X86_64    Claimed    Busy      0.000 3909  0+00:00:03
slot2@ip-172-31-23-93.ec2.internal      LINUX      X86_64    Claimed    Busy      0.000 3909  0+00:00:03
slot1@ip-172-31-27-244.ec2.internal     LINUX      X86_64    Claimed    Busy      0.000 3909  0+00:00:03
slot2@ip-172-31-27-244.ec2.internal     LINUX      X86_64    Unclaimed  Idle      0.000 3909  0+00:00:03

Machines Owner Claimed Unclaimed Matched Preempting Drain
X86_64/LINUX      6      0      5      1      0      0      0
Total             6      0      5      1      0      0      0
```

Figure 7: Condor_q and condor_status commands after test job submission

The following image shows the time required for completing the job for the five considered FASTA files.

```
Execution time: 108.432608843
Execution time: 138.391930103
Execution time: 105.939919949
Execution time: 105.939915895
Execution time: 64.9570920467
[ec2-user@ip-172-31-23-93 bwa]$
```

Figure 8: Execution time different reads.

The average time required for one file with a single file with 1000 reads of 150 residues is: **104.7321** seconds.

3.3 Errors and issues

The human genome was already indexed and the different files were present in the original snapshot but during the testing phase some errors were obtained and a strange output filled the 10 GB SSD memory of a node making it unusable.

```
condor_user@ec2-user:~$ condor_status
[ec2-user@ip-172-31-23-93 ~]$ condor_status
Error: communication error
CEDAR:6001:Failed to connect to <172.31.23.93:9618>
[ec2-user@ip-172-31-23-93 ~]$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/xvda2      10G   10G   20K  100% /
devtmpfs        1.9G     0  1.9G   0% /dev
tmpfs           1.9G   4.0K  1.9G   1% /dev/shm
tmpfs           1.9G   17M  1.9G   1% /run
tmpfs           1.9G     0  1.9G   0% /sys/fs/cgroup
/dev/xvdf1      99G   16G   78G  17% /data
tmpfs           379M     0  379M   0% /run/user/1000
[ec2-user@ip-172-31-23-93 ~]$
```

Figure 9: Memory error.

In addition, the following errors were reported in the standard errors file:


```
[bwa_aln] fail to locate the index
[bwa_sai2sam_se] fail to locate the index
```

In order to solve all these problems, the entire reference genome has been re-indexed.

4 Data Management

The data, different sequence reads, are stored in the shared volume. The main problem is that all the machines in order to perform the jobs in the condor cluster access the data in the magnetic disk and in many cases in the same moment because of parallel jobs: if I create a infrastructure with more nodes is not efficient to use only one disk with all the data. In our case I can split the different reads into different disks with higher performance than a magnetic disk in order to increase the velocity in retrieving the data to be process.

For this task, the input and output data are transferred via Sandboxes to run the condor job. For condor there is a very strict limit on sandbox size. In our case I want to align different reads to the reference genome: I have many data to process so I need different methods to transfer those files, both in input and in output. I use this workflow: I send small data as sandboxes using the sandbox statement. I use a CPU-driven Computing model but, for the above mentioned reasons, in a real case Data-driven Computing model is better because the concurrent access to the same disk for the different batch jobs can result in a bottleneck process. Another important point is the Policy and the Privacy of data because I'm considering medical data: in a real application the General Data Protection Regulation (GDPR) must be taken into account.

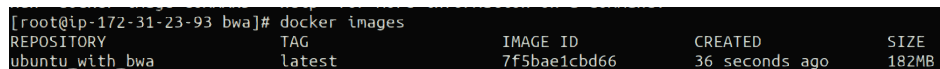
5 Containerized version of the application

To build the container version of BWA I use Docker. Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and deploy it as one package. This gives a significant performance boost and reduces the size of the application.

First of all I installed docker on the machines, then I create a docker container starting from the following Dockerfile. I built an ubuntu image because with Red Hat I have some errors.

```
FROM ubuntu
COPY ./alignment_reads.py alignment_reads.py
RUN chmod +x alignment_reads.py
RUN apt update
RUN apt install -y bwa
RUN apt install -y python
```

The following figure shows the output of the **docker images** command:



REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu_with_bwa	latest	7f5bae1cbd66	36 seconds ago	182MB

Figure 10: Docker images

Then I pushed the new created image on my personal Docker Hub account as shown in the Figure 11.

Because I'm able to install docker only on the master node while for unknown reasons I'm not able to configure the worker nodes (Figure 12), I run the job on a single FASTA file. The time required for aligning a single task is: 46.65743 seconds. The **bwa_batch.job** file was modified from Universe=vanilla to Universe=docker.

An already build containerized version of BWA was already available on Docker Hub, link <https://hub.docker.com/r/biocontainers/bwa>.

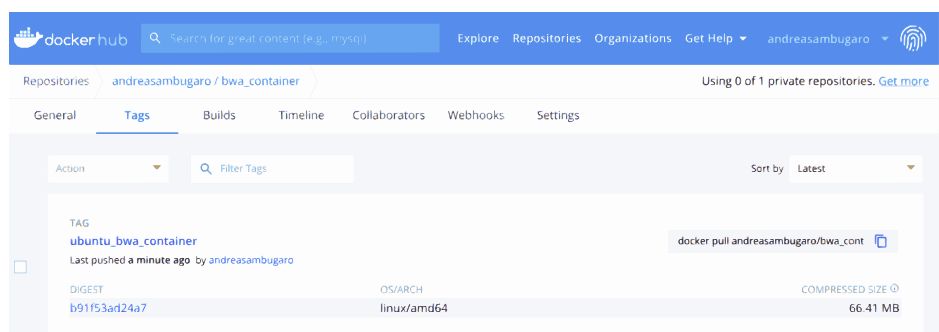


Figure 11: Docker Hub window

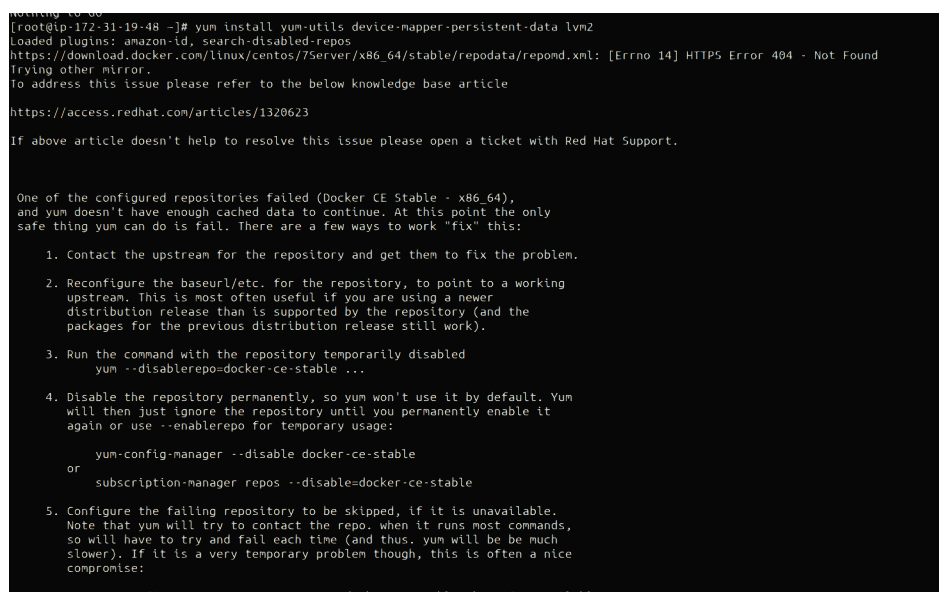


Figure 12: Docker error on worker nodes

6 Use-case: Supposing I want to map a genome to the reference human genome (hg19)

A possible real case is represented by the alignment of the reads of an Next Generation Sequencing (NGS) experiment to the human reference genome assembly hg19. A non-trivial challenge can be a Whole-Genome Sequencing (WGS) for analyzing an entire genome.

A human genome has a size of about 3.2 billion base pairs, but for the base calling procedure, researchers determine the necessary coverage. In fact, one of the most important aspect of this type of experiments is defining the NGS coverage level: the recommended coverage based on an Illumina platform for a human WGS experiment is in the range 30x to 50x (depending on application and statistical model). Given that, an overall amount of raw output data from the sequencer consist of at least 96 billion base pairs with 30x coverage that can be approximated to 100 billion base pair per single genome. For the cost estimation, here I consider a read length of 150 base that is the one recommended by the Illumina manufacturer for WGS and the length (150 bp) used for the time estimation in the test job. Satisfying these premises, almost 640 million reads would need to be aligned for a single patient that correspond to 640000 FASTA files with 1000 reads each.

The average time required for aligning a single FASTA file with 1000 reads of 150 residues length is approximately 105 seconds on a single core of our RHEL HTCondor cluster with 3 machines each with two cores. This corresponds to 67.200.000 seconds to complete the task using a single core while 11.200.000 seconds running the BWA application

on the instantiated infrastructure. This corresponds to about half a year, too long time for a real application. A possible solution to drastically reduce the total execution time while maintaining the same type of instance (t2.large) on AWS is to increase the number of nodes, especially the worker nodes, to have a parallelization of the entire work. The alignment of the individual reads is independent and do not affect the result for the other reads so it can be performed simultaneously.

This is an example of High-throughput computing (HTC) that involves the running of many independent tasks that require a large amount of computing power. With HTC, users can run many copies of their software simultaneously across many different nodes. To complete the alignment of all reads produced in a WGS analysis in about 5 days the Iaas infrastructure should be dimensioned with about 150 worker nodes.

The **t2.large** instance on AWS costs \$ 0.1528 per hour for RHEL: I require 150 of them for a user case infrastructure (the master node can be configured with a t2.medium instance if sets to work only as workload and not as worker node to reduce a bit the overall cost). The price for data storage on general-purpose SSD EBS devices costs \$ 0.1 per GB per month. A disk of 1TB is enough for storing the human genome, the correspondent FM-index, the application and the WGS reads of a patient.

6.1 Real case after a research in literature

In order to align a single human genome (37x coverage) our infrastructure should be dimensioned with low worker nodes and with instances with more computing power. The cloud infrastructure must be redimensioned as follows:

- instance type: cc2.8xlarge (60GB of memory, 32 virtual CPUs)
- cluster with a total of 21 nodes: a master node with 20 worker nodes

Using this type of instances with a total of 21 nodes and implementing a complex workflow making optimal use of resources in the cluster, the time required for mapping the the whole genome reads to the reference genome is around 14 hours. Employing an Amazon S3 storage the estimated average cost is 200\$.

The overall cost is highly influenced by the software and and the different applications able to optimize the time and consequently the total prize in order to complete also this complex job.

7 References

Thain, D., Tannenbaum, T. and Livny, M. (2005), Distributed computing in practice: the Condor experience. *Concurrency Computat.: Pract. Exper.*, 17: 323-356. doi:10.1002/cpe.938
Red Hat: <http://www.linuxfromscratch.org/blfs/view/svn/basicnet/nfs-utils.html>
Illumina: <https://emea.illumina.com/science/technology/next-generation-sequencing.html>
Li H, Durbin R. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*. 2009 Jul 15;25(14):1754-60. doi: 10.1093/bioinformatics/btp324. Epub 2009 May 18. PMID: 19451168; PMCID: PMC2705234.
Bwa manual: <http://bio-bwa.sourceforge.net/bwa.shtml>
SAM file format: <https://samtools.github.io/hts-specs/SAMv1.pdf>
FASTA file format: https://en.wikipedia.org/wiki/FASTA_format
AmazonAWS: <https://aws.amazon.com/it/pricing/>
Souilmi Y, Lancaster AK, Jung JY, et al. Scalable and cost-effective NGS genotyping in the cloud. *BMC Med Genomics*. 2015;8:64. Published 2015 Oct 15. doi:10.1186/s12920-015-0134-9
Docker: <https://www.docker.com/>
GDPR: <https://www.garanteprivacy.it/il-testo-del-regolamento>