Building Cloud IaaS infrastructures

Introduction

Infrastructure as a Service (IaaS), or cloud infrastructure services, is a form of cloud computing in which IT infrastructure is delivered to end-users over the Internet. In the IaaS model, users manage applications, data, operating system, middleware, and runtimes, while the IaaS service provider provides virtualization, storage, networking, and servers. This organization allows to eliminate the need for an on-site datacenter and to avoid the user all the physical updating and maintenance activities necessary for these components.

The infrastructure for this project consists of a Master Node and two Worker Nodes that collaborate in an HTCondor cluster. Shared storage space is also added to the infrastructure using the NFS file system: this volume is directly attached to the Master but it is also available to all the Worker Nodes.

1. Build a small computing infrastructure on the Cloud.

1.1 Master Node initialization

Master Node (MN) was built using Amazon Web Services. The machine type was Red Hat Enterprise Linux and for the MN, I chose the t2.medium instance type. It was instantiated in the *us-east-1b* availability zone.
The security group for this machine was *launch wizard 3* and the inbound rules were configured in the following way:

Regole in entrata

Tipo	Protocollo	Intervallo porte	Origine
Tutte le regole TCP	TCP	0 - 65535	sg-02fefc3012f0034a4 (launch-wizard-3)
SSH	TCP	22	0.0.0.0/0
Tutte le regole UDP	UDP	0 - 65535	sg-02fefc3012f0034a4 (launch-wizard-3)
Tutte le regole ICMP - IPv4	ICMP	Tutti	sg-02fefc3012f0034a4 (launch-wizard-3)

All the TCP, the UDP, and the ICMP-IPv4 ports were opened to the other members of the same security groups to allow communication between master and working nodes. ICMP-IPv4 ports allowed accepting incoming ping for the test job.

2 volumes were attached to the MN from AWS. One was a new 100 Gb standard volume and the second was *BDP1_data* volume that was obtained from the available snapshots.
These 2 volumes were mounted on the MN to permanently access them. To do this, the *etc/fstab* file was modified as follows:

```
#
# /etc/fstab
# Created by anaconda on Mon Jan 28 15:24:25 2019
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
#
UUID=4a1c93d9-eb47-4f96-9f3d-920e52dc8cca /                       xfs     defaults        0 0

/dev/xvdf1 /data ext4   defaults        0       0
/dev/xvdh1      /data3  ext4    defaults        0       0
~
```

After this configuration, */data* folder contained data from BDP1_data volume and */data3* folder contained data from the new volume (another volume was mounted on */data2* but it is not relevant for the project).

```
[root@ip-172-31-19-145 ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        1.9G     0  1.9G   0% /dev
tmpfs           1.9G     0  1.9G   0% /dev/shm
tmpfs           1.9G   17M  1.9G   1% /run
tmpfs           1.9G     0  1.9G   0% /sys/fs/cgroup
/dev/xvda2       10G  3.8G  6.3G  38% /
/dev/xvdf1       99G   16G   79G  17% /data
/dev/xvdg1       15G   41M   14G   1% /data2
/dev/xvdh1       99G   61M   94G   1% /data3
tmpfs           379M     0  379M   0% /run/user/1000
```

```
[root@ip-172-31-19-145 ~]# ll /data
total 20
drwxr-xr-x. 5 root root  4096 Apr 19  2020 BDP1_2020
drwx------. 2 root root 16384 Apr 26  2020 lost+found
[root@ip-172-31-19-145 ~]#
```

/data folder contained the original data that were stored in *BDP1_data* volume: this confirmed that the volume was correctly mounted.

1.2 Working nodes initialization

Working Nodes (WN) were also built using Amazon Web Services and Red Hat Enterprise Linux as machine type. I chose the t2.large instance type and they were instantiated in the same availability zone of the Master (*us-east-1b*) so that they would be able to communicate through private IPv4 addresses.
The security group for this machine was *launch wizard 3* and the inbound rules were configured in the same way as the Master. Using these rules, the machines can communicate in a secure way.

2. Create a storage space (a volume) that must be shared among all the nodes

The volumes attached to the MN need to be shared with the WNs. To do this, NFS was installed:

- yum install nfs-utils rpcbind (on the master node)
- yum install nfs-utils (on the working nodes)

```
[root@ip-172-31-19-145 ~]# systemctl status nfs
● nfs-server.service - NFS server and services
   Loaded: loaded (/usr/lib/systemd/system/nfs-server.service; enabled; vendor preset: disabled)
  Drop-In: /run/systemd/generator/nfs-server.service.d
           └─order-with-mounts.conf
   Active: active (exited) since Mon 2021-02-15 16:31:13 UTC; 1h 4min ago
  Process: 1536 ExecStartPost=/bin/sh -c if systemctl -q is-active gssproxy; then systemctl reload gssproxy ; fi (code=exited, status=0/SUCCESS)
  Process: 1429 ExecStart=/usr/sbin/rpc.nfsd $RPCNFSDARGS (code=exited, status=0/SUCCESS)
  Process: 1419 ExecStartPre=/usr/sbin/exportfs -r (code=exited, status=0/SUCCESS)
 Main PID: 1429 (code=exited, status=0/SUCCESS)
    Tasks: 0
   Memory: 0B
   CGroup: /system.slice/nfs-server.service

Feb 15 16:31:13 ip-172-31-19-145.ec2.internal systemd[1]: Starting NFS server...
Feb 15 16:31:13 ip-172-31-19-145.ec2.internal systemd[1]: Started NFS server ...
Hint: Some lines were ellipsized, use -l to show in full.
[root@ip-172-31-19-145 ~]#
```

On the MN, the *etc/export* file was modified adding the following lines:

```
/data   172.31.28.239(rw,sync,no_wdelay)
/data3  172.31.28.239(rw,sync,no_wdelay)
/data   172.31.31.195(rw,sync,no_wdelay)
/data3  172.31.31.195(rw,sync,no_wdelay)
~
```

This allowed to expose */data* and */data3* folder to the Worker Nodes.
The command:

- exportfs -r

reexports all directories and removes entries that have been deleted from */etc/exports*.

On the WN the *etc/fstab* file was modified as follows:

```
#
# /etc/fstab
# Created by anaconda on Mon Jan 28 15:24:25 2019
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
#
UUID=4a1c93d9-eb47-4f96-9f3d-920e52dc8cca /                       xfs     defaults        0 0
172.31.19.145:/data /data nfs defaults 0 0
172.31.19.145:/data3 /data3 nfs defaults 0 0
~
```

After this configuration, */data* and */data3* folder were accessible for all the nodes. It was possible to verify this configuration with *df -h* on the WN:

```
[root@ip-172-31-28-239 ~]# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/xvda2             10G   10G  2.3M 100% /
devtmpfs              3.8G     0  3.8G   0% /dev
tmpfs                 3.9G     0  3.9G   0% /dev/shm
tmpfs                 3.9G   17M  3.9G   1% /run
tmpfs                 3.9G     0  3.9G   0% /sys/fs/cgroup
172.31.19.145:/data    99G   16G   79G  17% /data
172.31.19.145:/data3   99G   60M   94G   1% /data3
tmpfs                 782M     0  782M   0% /run/user/1000
```

```
[root@ip-172-31-28-239 ~]# ll /data
total 20
drwxr-xr-x. 5 root root  4096 Apr 19  2020 BDP1_2020
drwx------. 2 root root 16384 Apr 26  2020 lost+found
[root@ip-172-31-28-239 ~]#
```

3. Condor cluster configuration

I installed HTCondor on the machines. Host and clients need to communicate then it was necessary to modify */etc/condor/condor_*config files as follows:

- on the Master

```
CONDOR_HOST = 172.31.19.145

DAEMON_LIST = COLLECTOR, MASTER, NEGOTIATOR, STARTD, SCHEDD

HOSTALLOW_READ = *
HOSTALLOW_WRITE = *
HOSTALLOW_ADMINISTRATOR = *
```

- on the WN

```
CONDOR_HOST = 172.31.19.145

DAEMON_LIST = MASTER, STARTD

HOSTALLOW_READ = *
HOSTALLOW_WRITE = *
HOSTALLOW_ADMINISTRATOR = *
```

Checking the condor status I get the following outputs:

- MN

```
[root@ip-172-31-19-145 ~]# 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 Mon 2021-02-15 16:31:13 UTC; 1h 14min ago
 Main PID: 1393 (condor_master)
   Status: "All daemons are responding"
    Tasks: 7 (limit: 4194303)
   Memory: 22.8M
   CGroup: /system.slice/condor.service
           ├─1393 /usr/sbin/condor_master -f
           ├─1696 condor_procd -A /var/run/condor/procd_pipe -L /var/log/condor/ProcLog -R 1000000 -S 60 -C 995
           ├─1708 condor_shared_port -f
           ├─1858 condor_collector -f
           ├─1859 condor_negotiator -f
           ├─1860 condor_startd -f
           └─1863 condor_schedd -f

Feb 15 16:31:13 ip-172-31-19-145.ec2.internal systemd[1]: Started Condor Distributed High-Throughput-Computing.
```

- WN

```
[root@ip-172-31-28-239 ~]# 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 Mon 2021-02-15 16:40:11 UTC; 1h 5min ago
 Main PID: 4339 (condor_master)
   Status: "All daemons are responding"
    Tasks: 4 (limit: 4194303)
   Memory: 13.3M
   CGroup: /system.slice/condor.service
           ├─4339 /usr/sbin/condor_master -f
           ├─4647 condor_procd -A /var/run/condor/procd_pipe -L /var/log/condor/ProcLog -R 1000000 -S 60 -C 995
           ├─4654 condor_shared_port -f
           └─4938 condor_startd -f

Feb 15 16:40:11 ip-172-31-28-239.ec2.internal systemd[1]: Started Condor Distributed High-Throughput-Computing.
```

4. Submit some test jobs to the installed batch system

BDP1_data volume contained NGS reads from 3 different patients. Each patient had a folder with around 500 fasta files. In this test job, I aligned 5 fasta files from patient 1 (from read_1.fa to read_5.fa) to the human genome build hg19 (also stored in the shared volume).
For this purpose, I used the BWA alignment tool: its advantage is the genome indexing for speeding up the alignment (the index for hg19 build was already present in *BDP1_data* volume).

4.1 Data management model

NFS allows us to share data between VMs and this is very useful when we manage large files as the hg19 genome file and genome index. Using this method, we can store a single copy of this data and it is accessible from each machine.
Input Sandbox is used to transfer input files and executables. This movement was not a problem for the cluster since these files are small.
At the end of the process, Output Sandbox contains the condor files (log, error, and output) that need to be easily accessible by the user to analyze the job's performance. They were small files, so it was not a problem to move them in the cluster. Aligned reads are big and they need to be compressed and stored in the shared volume.

4.2 Job submission

The file submitted to the Condor cluster was the following .job file:

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

Executable = alignment_test.py
num = $(Process)+1
arguments = read_$INT(num).fa

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

Input      = read_$INT(num).fa
transfer_input_files = read_$INT(num).fa

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

Log        = read_$INT(num).log
# will contain condor log

Output     = read_$INT(num).out
# will contain the standard output

Error      = read_$INT(num).error
# will contain the standard error

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

should_transfer_files = YES
when_to_transfer_output = ON_EXIT

Universe = vanilla

#####################################################

Queue 5
```

I used *$(Process)* to simplify reading the input name for the files in the queue: processes were in the range 0-4 so it was necessary to add 1.

The executable file called from .job was the following:

```
#!/usr/bin/python
###This script performs the following operations:
### 1. bwa aln for the sequence alignment
### 2. bwa samse generates alignments in the SAM format
### 3. check the checksum to verify that the file transfer was successful
### 4. transfer files to shared volume and clean the Output Sandbox
### 5. calculate the total time required for these operations

import sys,os
from timeit import default_timer as timer


time_0 = timer()
dbpath = "/data/BDP1_2020/hg19/"
dbname = "hg19bwaidx"
queryname = sys.argv[1]
out_name = queryname[:-3]

gzipfile = out_name + ".sam.gz"

md5file = out_name + ".md5"

print "Input: ", queryname

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

command = "bwa samse -n 10 " + dbpath + dbname + " " + out_name + ".sai "  + queryname + " > " + out_name + ".sam"
print "launching command: " , command
os.system(command)

print "Creating md5sums"
os.system("md5sum " + out_name + ".sai"  + " > " + md5file)
os.system("md5sum " + out_name + ".sam"  + " >> " + md5file)

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

print "Transferring files + Clearing the Output Sandbox"
os.system("mv "+ gzipfile + " /data/outputs/"+ gzipfile)
os.system("mv "+ md5file + " /data/outputs/"+ md5file)
os.system("rm "+ out_name + ".sai")

total_time = timer() - time_0

print "Total time: " + str(total_time)
print "exiting"

exit(0)
~
```

Condor_q allowed observing the progress of the condor processes in the cluster.

```
[ec2-user@ip-172-31-19-145 ~]$ condor_q


-- Schedd: ip-172-31-19-145.ec2.internal : <172.31.19.145:9618?... @ 02/16/21 13:29:41
OWNER    BATCH_NAME    SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
ec2-user ID: 27       2/16 13:29      _      2      3      5 27.0-4

Total for query: 5 jobs; 0 completed, 0 removed, 3 idle, 2 running, 0 held, 0 suspended
Total for ec2-user: 5 jobs; 0 completed, 0 removed, 3 idle, 2 running, 0 held, 0 suspended
Total for all users: 5 jobs; 0 completed, 0 removed, 3 idle, 2 running, 0 held, 0 suspended
```

In the end, the *data/outputs* folder contained these files:

```
[root@ip-172-31-19-145 ec2-user]# ll /data/outputs/
total 40
-rw-r--r--. 1 ec2-user ec2-user  45 Feb 16 13:30 read_1.md5
-rw-r--r--. 1 ec2-user ec2-user  31 Feb 16 13:30 read_1.sam.gz
-rw-r--r--. 1 ec2-user ec2-user  45 Feb 16 13:31 read_2.md5
-rw-r--r--. 1 ec2-user ec2-user  31 Feb 16 13:30 read_2.sam.gz
-rw-r--r--. 1 ec2-user ec2-user  45 Feb 16 13:30 read_3.md5
-rw-r--r--. 1 ec2-user ec2-user  31 Feb 16 13:30 read_3.sam.gz
-rw-r--r--. 1 ec2-user ec2-user  45 Feb 16 13:30 read_4.md5
-rw-r--r--. 1 ec2-user ec2-user  31 Feb 16 13:30 read_4.sam.gz
-rw-r--r--. 1 ec2-user ec2-user  45 Feb 16 13:32 read_5.md5
-rw-r--r--. 1 ec2-user ec2-user 893 Feb 16 13:32 read_5.sam.gz
```

Final condor outputs were similar to the following:

```
Input:  read_1.fa
launching command:  bwa aln -t 1 /data/BDP1_2020/hg19/hg19bwaidx read_1.fa > read_1.sai
launching command:  bwa samse -n 10 /data/BDP1_2020/hg19/hg19bwaidx read_1.sai read_1.fa > read_1.sam
Creating md5sums
gzipping out text file
launching command:  gzip read_1.sam
Transferring files + Clearing the Output Sandbox
Total time: 69.796998024
exiting
```

As the last step, I compared different execution times:

```
[ec2-user@ip-172-31-19-145 output]$ cat *.out | grep "Total"
Total time: 69.796998024
Total time: 212.005355835
Total time: 89.164672851
Total time: 168.093282938
Total time: 328.0871710777
```

5. Create a container image and run the test jobs using the containerized version

Containers are “lightweight Vms”: Docker Engine comprises just the application and its dependencies. They require fewer resources and they allow to easily encapsulate applications in a controlled and extensible way.
In this part of the project, the same task performed before were repeated encapsulating the BWA application in a Docker container.

Docker was installed on my VM using commands in “Docker_Excersises.sh”

```
[root@ip-172-31-25-63 ~]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor prese
t: disabled)
   Active: active (running) since Wed 2021-02-17 09:57:21 UTC; 18s ago
     Docs: https://docs.docker.com
 Main PID: 18370 (dockerd)
   CGroup: /system.slice/docker.service
           └─18370 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/con...
```

Shared */data* volume needs to be accessible from Docker, then *etc/condor/condor_config* was modified as follows:

```
# Docker configs
DOCKER_VOLUMES = BDP1_DATA
DOCKER_VOLUME_DIR_BDP1_DATA = /data
DOCKER_MOUNT_VOLUMES = BDP1_DATA
```

I built *becchi/bwa* image from the following Dockerfile:

```
FROM ubuntu
COPY ./alignment_test.py alignment_test.py
RUN apt update
RUN apt install -y python
RUN apt install -y bwa
~
```

alignment_test.py was the same file used for the original version of the cluster. I modified the alignment_test.job changing the Universe from Vanilla to docker.

```
###############Alignment Test ######################
#####################################################


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

docker_image = becchi/bwa
Executable = /alignment_test.py
num = $(Process)+1
arguments = read_$INT(num).fa

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

Input      = read_$INT(num).fa
transfer_input_files = read_$INT(num).fa

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

Log        = read_$INT(num).log
# will contain condor log

Output     = read_$INT(num).out
# will contain the standard output

Error      = read_$INT(num).error
# will contain the standard error

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

should_transfer_files = YES
when_to_transfer_output = ON_EXIT

Universe = docker

#####################################################

Queue 5
```

After the submission, we get the output file as in the original cluster and it is possible to analyze the time.

```
[ec2-user@ip-172-31-19-145 output_dock]$ cat *.out |grep "Total"
Total time: 56.574838575
Total time: 142.27844733
Total time: 101.69922948
Total time: 135.39245812
Total time: 284.58102932
```

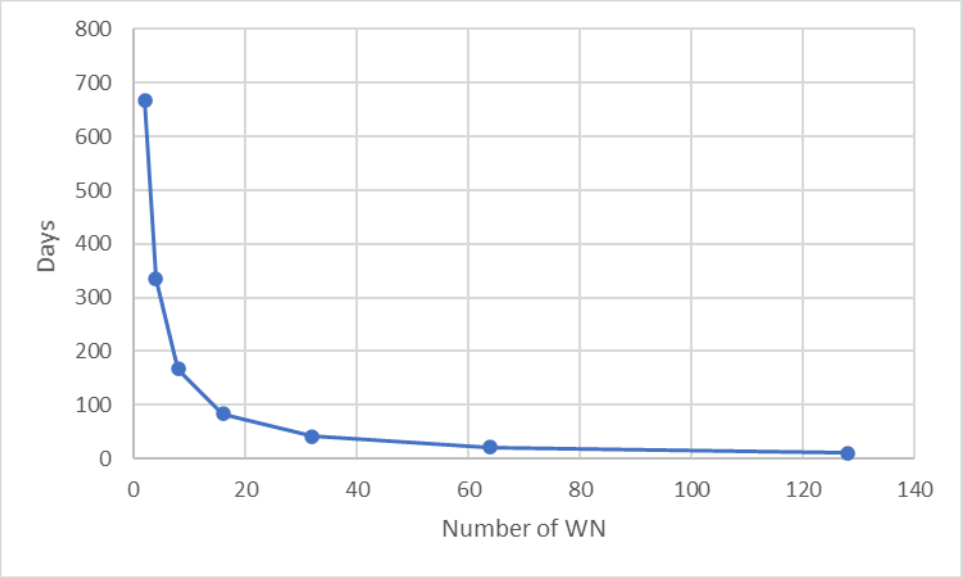
The two performances are quite similar. Docker takes less time on average than the native version (144,2 seconds vs 173,4 seconds).

The main advantage of Docker containers is the possibility to modify the application or its configuration flexibly. To change the application, just change the name of the image.
One possible Docker disadvantage is that it requires more resources, but it is possible to avoid this problem including only the components that are strictly required by the application.

6. Evaluate the time needed to address a use-case based on the chosen application

A non-trivial challenge that can be implemented on this HTC cluster is the alignment of a big number of fasta files to the human reference genome assembly hg19.
The minimum recommended coverage by Illumina is 30X. This means that a human genome with about 3.3 billion base pairs generates an output of about 100 billion base pairs.
If we consider fasta files with 1000 reads each and reads with 150 bp each, we get about 666000 fasta files that need to be aligned.

Our original cluster takes on average 173.4 seconds to align a fasta file with 1000 reads to hg19. This means a total time of 115484400 seconds for the alignment of a 30X sequencing of the whole genome of a patient.
Our cluster is composed by one MN and two WN, this means that the cluster can use two cores and the task requires 57.742.200 seconds to complete the task.
This is a very high number and, in the reality, it is possible to increase the cluster's performance replicating the Worker Nodes.
Assuming that total time and number of WN are inversely proportional, it is possible to obtain the following graph that shows how many days are required for the task when the number of WN increases.



7. Estimate the cost of the system to complete the challenge supposing that the Cloud provider is Amazon

Nome	vCPU	RAM (GiB)	Crediti CPU/ora	Prezzo on demand/ora*
t2.nano	1	0,5	3	0,0058 USD
t2.micro	1	1,0	6	0,0116 USD
t2.small	1	2	12	0,023 USD
t2.medium	2	4	24	0,0464 USD
t2.large	2	8	36	0,0928 USD

This table shows the cost/hour of each instance type. In the preceding paragraph, we observe that our task requires about 100 WN which are t2. large. This means a total cost of 9.28 dollars/hour. We must add the cost of the single MS which is a t2. medium. In the end, the total cost is 9.33 dollars/hour.

8. References

1. https://www.awseducate.com/student/s/classrooms
2. https://emea.illumina.com/science/technology/next-generation-sequencing/plan-experiments/coverage.html
3. https://aws.amazon.com/it/ec2/instance-types/t2/