Building IaaS infrastructures on the AWS Cloud

Saul Pierotti

June 17, 2020

Abstract

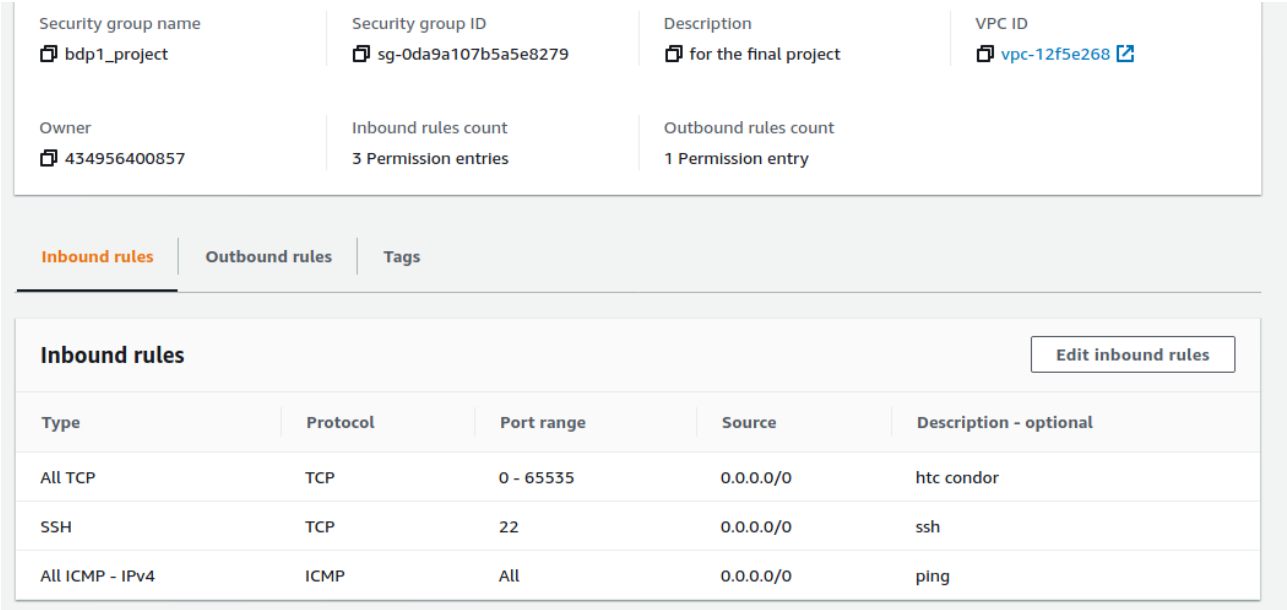
The abstract text goes here.

1 General Description of the Infrastructure

The demostrative infrastructure described in this project consists of an HTCondor cluster of three nodes. One node is configured as Master Node, while 2 nodes are configured as Worker Nodes. The infrastructure can be easily expanded by replicating the Worker Node instances. The Master Node was not used also as a Worker Node since the performance benefits and cost savings would be marginal. This was done also to avoid overloading the Master Node, on which the entire cluster depends. A shared storage space directly attached to the Master Node but available to all the Worker Nodes was also implemented using the distributed file system NFS.

2 Initialization of the instances on the AWS Cloud

Worker Nodes and the Master Node were both built on the similar machines. For the Master Node, the `t2.medium` instance type was used with a 50 Gb SSD as root storage. For the WOrker Nodes, the `t2.large` instance type was used with a 50 Gb SSD as root storage. The operating system choosen for both machine types is Ubuntu Server 18.04.4 LTS. The Master Node and the Worker Nodes were all instatiated in the same availability zone (`us-east-1a`), so that they would be able to communicate through private IPv4 adresses. The security group for the instances was configured as follows:



All the TCP ports were opened to the other members of the same security group since HTCondor deamons use a dynamically assigned port. The TCP ports were also needed for the setup of a shared NFS volume. ICMP ports were opened for accepting incoming `ping` requests for testing purposes. TCP port 22 was opened for allowing remote control of the machines via `ssh`.

3 Configuration of the Master Node

The PS1 prompt of the Master Node was changed so to make the node easily identifiable from the command line.

```
ubuntu@bdp1-master-node:~$ echo $PS1
\[\e]0;\u@\h: \w\a\]${debian_chroot:+($debian_chroot)}\[\033[01;32m\
]\u@bdp1-master-node\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$
```

HTCondor was then installed with the following commands:

```
sudo su
wget -qO - https://research.cs.wisc.edu/htcondor/ubuntu/HTCondor-Release.gpg.key | apt-key add - #
    import the gpg key of HTCondor
echo "deb http://research.cs.wisc.edu/htcondor/ubuntu/8.8/bionic bionic contrib" >> /etc/apt/sources.
    list # add the repository
echo "deb-src http://research.cs.wisc.edu/htcondor/ubuntu/8.8/bionic bionic contrib" >> /etc/apt/
    sources.list
apt update
apt install htcondor
systemctl start condor # start and enable the condor service
systemctl enable condor
```

The correct proceeding of the installation and the start of the `condor` service where checked with the following commands:

```
ubuntu@bdp1-master-node:~$ sudo systemctl status condor
● condor.service - Condor Distributed High-Throughput-Computing
   Loaded: loaded (/lib/systemd/system/condor.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2020-06-16 10:31:25 UTC; 1min 16s ago
 Main PID: 15225 (condor_master)
   Status: "All daemons are responding"
    Tasks: 3 (limit: 32767)
   CGroup: /system.slice/condor.service
           ├─15225 /usr/sbin/condor_master -f
           ├─15266 condor_procd -A /var/run/condor/procd_pipe -L /var/log/condor/ProcLog -R 1000000 -S 60 -C 111
           └─15268 condor_shared_port -f

Jun 16 10:31:25 ip-172-31-8-109 systemd[1]: Started Condor Distributed High-Throughput-Computing.
ubuntu@bdp1-master-node:~$ ps ax | grep condor
15225 ?        Ss     0:00 /usr/sbin/condor_master -f
15266 ?        S      0:00 condor_procd -A /var/run/condor/procd_pipe -L /var/log/condor/ProcLog -R 1000000 -S 60 -C 111
15268 ?        Ss     0:00 condor_shared_port -f
15286 pts/0    S+     0:00 grep --color=auto condor
ubuntu@bdp1-master-node:~$
```

The following lines where appended at the end of the main HTCondor configuration file, located at `/etc/condor/condor_config`:

```
# Master Node IP
CONDOR_HOST = <Master_Node_private_IP>

# Master Node config
DAEMON_LIST = COLLECTOR, MASTER, NEGOTIATOR, STARTD
```

Finally, the `condor` service was restarted with the following command:

```
sudo systemctl restart condor
```

The NFS server was then implemented in the Master Node. A new 100 Gb standard magnetic volume was created from the AWS interface and attached to the Master Node. From the server, a primary partiton was initialized on the volume using `fdisk` and an `Ext4` file system was created onto it using `mkfs.ext4`. The file `/etc/fstab` of the Master Node was modified so that the machine would mount the volume automatically at boot under the newly created directory `/data`. The following line was appended to `/etc/fstab`:

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

The following commands were then issued, so to install the appropriate packages:

```
sudo apt install nfs-kernel-server
```

The following line was appended to the NFS configuration file `/etc/exports`:

```
/data 172.31.0.0/16(rw,sync,no_wdelay)
```

Finally, the owner and group of the shared folder was changed to `nobody:nogroup` and the permission of the folder were edited so to grant unlimited access to it:

```
sudo chown nobody:nogroup /data
sudo chmod 777 /data
```

The `/data` folder was so made available to all the Worker Nodes on the address range `172.31.0.0/16`. This configuration does not pose a security risk since the the Master Node and Worker Nodes machines belong to a Virtual Private Cloud (VPC), and so only machines instatiated on it will be able to access the volume. Moreover, this configuration grants immediately access to the volume to newly instantiated Worker Nodes. A mock file was created on the `/data` folder so to be able to recognize it when mounted.

```
touch /data/this_is_a_shared_NFS_volume
```

4 Configuration of the Worker Nodes

A virtual machine identical to the one used for the Master Node was instantiated. The PS1 prompt of this first Worker Node was changed so to make the node easily identifiable from the command line.

```
ubuntu@bpd1-worker-node:~$ echo $PS1
\[\e]0;\u@\h: \w\a\]${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@bpd1-worker-node\[\033[00m\]:\[\
033[01;34m\]\w\[\033[00m\]\$
```

HTCondor was installed in this system with the same procedure used for the Master Node. Only the `/etc/condor/condor_config` file was configured differently, by appending the following lines to it:

```
# Master Node IP
CONDOR_HOST = <Master_Node_private_IP>

# Worker Node config
DAEMON_LIST = MASTER, STARTD

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

On the Worker Node access to the shared NFS volume was then set up. The following command was issued to install the required packages and enable the respective services:

```
sudo apt install nfs-common
sudo systemctl start nfs-server
sudo systemctl enable nfs-server
```

A new directory was then created at `/data` using the `mkdir` command. The `/etc/fstab` file was edited by appending the following line, so that the shared volume would be automatically mounted at boot under the directory `/data`:

```
<Master_Node_private_IP>:/data   /data   nfs   defaults            0 0
```

It was then verified that the shared volume was accessible from the Worker Node.

```
ubuntu@bpd1-worker-node:~$ ll /data
total 24
drwxrwxrwx  3 nobody nogroup  4096 Jun 17 07:16 ./
drwxr-xr-x 24 root   root     4096 Jun 17 06:46 ../
drwx------  2 root   root    16384 Jun 16 17:38 lost+found/
-rw-rw-r--  1 ubuntu ubuntu      0 Jun 17 07:16 this_is_a_shared_NFS_volume
```

On the Worker Nodes the application BWA was installed with the command:

```
sudo apt install -y bwa
```

An image of the virtual machine was taken through the AWS interface. In this way, the Worker Nodes would be easily replicable when more computational power is needed by instaciating new virtual machines from the image, without the need for manual configuration.

5 Submission of a test job to the HTCondor cluster

For demostrating the use of the newly created cluster, a new Worker Node was instanciated from the Worker Node image, so that the cluster would contain the Master Node and 2 Worker Nodes.

A new volume was created from the AWS interface from a snapshot containing test data for the BDP1 course (`snap-09ee52d8038fb8094`, BDP1_2020). This snapshot contains biological data to be used for testing the infrastructure. It contains NGS reads from 3 different patients. Each patient has a folder with around 500 fasta files, with 1000 reads each. This data will be used for testing the infrastructure. This new volume was mounted on the Master Node under `/data`, substituting the empty volume used before. The `/etc/fstab` file was update accordingly and the `nfs-server` service restarted. It was then tested that the new volume was accessible from the Worker Nodes.

```
ubuntu@bpd1-worker-node:~$ ll /data
total 28
drwxrwxrwx  4 root root  4096 Apr 26 13:38 ./
drwxr-xr-x 24 root root  4096 Jun 17 06:46 ../
drwxr-xr-x  5 root root  4096 Apr 19 14:39 BDP1_2020/
drwx------  2 root root 16384 Apr 26 09:37 lost+found/
```

The test job consisted in aligning 1 fasta file from 1 patient to the human genome build hg19, also stored on the shared volume. The BWA alignment tool was used for the scope. This tool takes advantage of indexing the genome for speeding up the alignment of low-divergent reads to it. The index for the hg19 build was already present in the volume snapshot, so there was no need to create it from scratch. The test reads were copied from the shared volume to the home folder of the Master Node. The test job file `alignment_test.job` was created, with the following content:

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


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

Executable = alignment_test.py -

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

Input     = -

transfer_input_files = -

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

Log       = condor.log
# will contain condor log

Output    = condor.out
# will contain the standard output

Error     = condor.error
# will contain the standard error

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

should_transfer_files = YES
when_to_transfer_output = ON_EXIT

Universe = vanilla

#####################################################
Queue
```

The script `alignment_test.py` was the following:

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

start = timer()
dbpath = "/data/BDP1_2020/hg19/"
dbname = "hg19bwaidx"
basename = sys.argv[1]
queryname = basename+".fa"
out_name = basename
md5file = out_name+".md5"

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)

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

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

# Transfer files to shared volume and clean the Output Sandbox
print "Moving files and clearing the Output Sandbox"
gzipfile = out_name + ".sam.gz"
saifile = out_name + ".sai"
os.system("cp "+ gzipfile + " /data/outputs/"+ gzipfile)
os.system("cp "+ md5file + " /data/outputs/"+ md5file)
os.system("rm "+ saifile)

execution_time = timer() - start

print "Total execution time: " + str(execution_time)
print "exiting"

exit(0)
```

Ten instances of this test job were run on the cluster. The cluster was able to successfully complete them in

6 Data Management Model

The general model followed would be that of transferring the executables and the read file using the HTCondor Input Sandbox. This is because those files are generally small and their transfer will not overload the Master Node. In addition, this setup allows flexibility in choosing the application to be used and the input data. On the contrary, the large hg19 genome and index files will be made availabe to the Worker Nodes through the shared NFS volume. The Output Sandbox will contain the condor log, the standard output and the standard error. The aligned reads will instead be compressed and put in the shared volume, so to avoid moving large data on the Output Sandbox.

6.1 Creation of a Container Image for the Application

A containerized version of the BWA application was built using `docker`. A container image for the BWA application is already available on DockerHub (`biocontainers/bwa`), but for educational purposes a new image was built from scratch. The Ubuntu 18.04.4 LTS Linux docker image was used as a base. The following Dockerfile was created:

The dockerfile