Building Cloud IaaS infrastructures

Roberto Chiaiese[1]
[1]Student of the Master in Bioinformatic at the University of Bologna

Abstract

This work aimed to build a cloud computing infrastructure using Amazon Web Services (AWS) to simulate a geographically distributed system for solving a biological computational challenge. The infrastructure was developed using key components such as the Network File System (NFS) for data sharing, HTCondor for job scheduling, and WebDav for efficient data transfer between sites. Existing data from the IBDPI course of the Master in Bioinformatics was attached to the master nodes and shared among worker nodes through NFS. Once the computational challenge was addressed, performance metrics, including time and cost analyses, were collected to evaluate the infrastructure. The study also explores potential improvements to further enhance the efficiency and scalability of the system.

Contact: *roberto.chiaiese@studio.unibo.it*

1. Creating the Instances for Distributed Sites

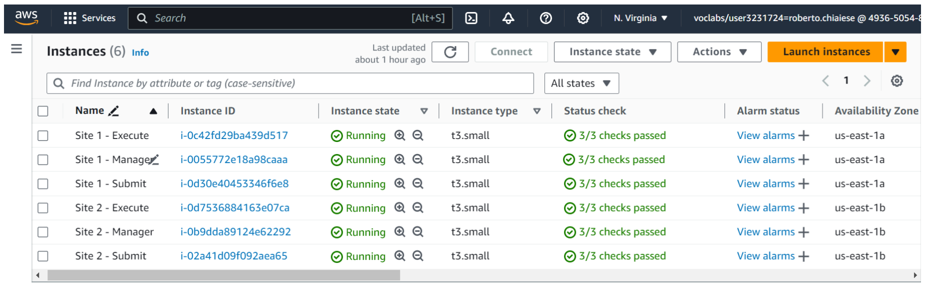
To set up a distributed cloud-based infrastructure, we configured two distinct geographic sites within the AWS environment, one located in the us-east-1a availability zone and the other in us-east-1b. Each site was designed with independent security groups, which allows us to maintain isolated configurations while simulating a geographically distributed system.

Instance Roles:

Each site comprises three virtual instances, all of which are t3.small types. The roles assigned to these instances are critical for managing the workload in a distributed manner:

- **Manager Node:** This instance controls the workflow by overseeing the distribution of computational jobs across other nodes.
- **Submit Node:** This instance is responsible for submitting jobs into the HTCondor cluster, which handles the scheduling and allocation of tasks.
- **Execute Node:** This instance serves as the compute engine, where tasks are actually processed as part of the HTCondor cluster.

All the nodes in both sites are **t3.small instances**, which offer a good balance between cost and performance, given their specifications: 2 vCPUs and 2GiB of memory per instance. Each instance is allocated **10 GiB of gp3 storage**, providing adequate space for job data while keeping costs minimal.


aws
Services
Search
[Alt+S]
N. Virginia
voclabs/user3231724=roberto.chiaiese @ 4936-5054-
Instances (6) Info
Last updated about 1 hour ago
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
Site 1 - Execute	i-0c42fd29ba439d517	Running	t3.small	3/3 checks passed	View alarms +	us-east-1a
Site 1 - Manager	i-0055772e18a98caaa	Running	t3.small	3/3 checks passed	View alarms +	us-east-1a
Site 1 - Submit	i-0d30e40453346f6e8	Running	t3.small	3/3 checks passed	View alarms +	us-east-1a
Site 2 - Execute	i-0d7536884163e07ca	Running	t3.small	3/3 checks passed	View alarms +	us-east-1b
Site 2 - Manager	i-0b9dda89124e62292	Running	t3.small	3/3 checks passed	View alarms +	us-east-1b
Site 2 - Submit	i-02a41d09f092aea65	Running	t3.small	3/3 checks passed	View alarms +	us-east-1b

Figure 1: AWS "Instances" screen displaying six active nodes, with three t3.small instances created for each site (Site 1 in us-east-1a and Site 2 in us-east-1b). Each site includes manager, submit, and execute nodes for distributed processing.

*1.0.1. **Site 1 – Manager, Execute & Submit Nodes***

For **Site 1**, the infrastructure is located in the `us-east-1a` availability zone. The operating system image used for all instances is **Red Hat Enterprise Linux 9** on a 64-bit architecture. Each instance type, as mentioned earlier, is **t3.small**, with the pricing of **0.0208 USD/hour**. Storage for each instance is provided through **10 GiB gp3**, an AWS general-purpose SSD solution, which ensures a balance between performance and cost. All instances are protected and configured under the security group named **lunch-wizard-8**.

*1.0.2. **Site 2 – Manager, Execute & Submit Nodes***

Similarly, for **Site 2**, located in the `us-east-1b` availability zone, the same configuration applies. The OS image is **Red Hat Enterprise Linux 9**, with instances of the **t3.small** type, priced at **0.0208 USD/hour**. This site is also allocated **10 GiB gp3** storage per instance. However, the security group here is different, named **lunch-wizard-1**, maintaining independent security settings from Site 1 while simulating a distinct geographical region.

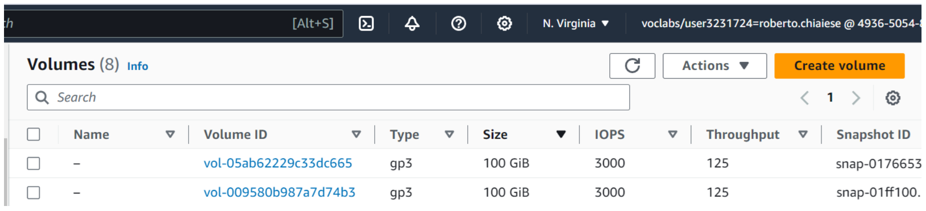
By setting up these distinct sites, we not only ensured geographic distribution but also tested how independently configured environments interact within a distributed cloud-based system. These isolated sites were essential for simulating real-world conditions, where different data centers may have varying security policies, yet still need to work in unison to process and share data efficiently.

2. Creating a Snapshot Based on a Previous Volume

The next step in setting up our distributed infrastructure involved creating and managing storage volumes using AWS snapshots. This process ensured data redundancy and enabled cross-site distribution of essential datasets.

Step 1: Initial Snapshot Creation

To begin, we created a snapshot from an existing volume, capturing all critical data required for further processing. This snapshot serves as a backup of the original volume and can be used to restore or replicate the data across different AWS regions or availability zones.
Step 2: New Volume Setup The snapshot was then utilized to provision a new volume in a different availability zone. Specifically, while the original volume was housed in the `us-east-1a` zone, the newly created volume was set up in `us-east-1b`. This process enabled geographic redundancy, ensuring that both sites have access to identical datasets without manually transferring data between regions. By leveraging AWS's snapshot and volume management, we improved the system's resilience and data accessibility.



	Name	Volume ID	Type	Size	IOPS	Throughput	Snapshot ID
☐	–	vol-05ab62229c33dc665	gp3	100 GiB	3000	125	snap-0176653
☐	–	vol-009580b987a7d74b3	gp3	100 GiB	3000	125	snap-01ff100.

Figure 2: AWS "Volumes" screen showing 100 GiB volumes created from snapshots, attached to manager nodes in different availability zones (us-east-1a and us-east-1b) for cross-site data redundancy.

2.1. *Connecting the 100 GiB Volumes to Manager Nodes*

Once the new volumes were created, we connected them to the appropriate manager nodes within each site to ensure that they had access to the required storage.

- **Access the Volumes Tab:** Navigate to the AWS management console and locate the "Volumes" section to view all available storage units.
- **Select the Volume:** Identify the 100 GiB volumes intended for use by each manager node in both `us-east-1a` and `us-east-1b` availability zones.
- **Attach the Volume:** Proceed to attach each selected volume to its respective manager node. This step ensured that each manager node could access and utilize the storage allocated for its tasks, maintaining consistency in data access across both sites.

3. Configuring Security Groups

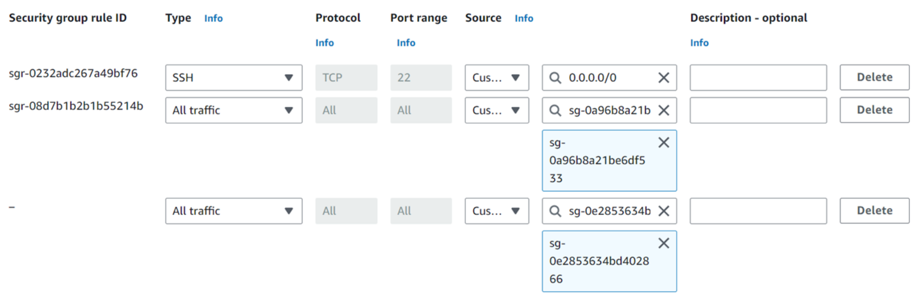
Security groups were configured to manage access and traffic between the two geographically distributed sites.

3.1. Lunch-Wizard-8 Setup

- **SSH Access:** Allowed from any IP address (Source: `0.0.0.0/0`), ensuring remote access to the nodes from any location.
- **Intra-Site Traffic:** All traffic is permitted within the security group `lunch-wizard-8`, enabling unrestricted communication between nodes within Site 1.
- **Inter-Site Communication:** To allow collaboration between the two sites, all traffic is allowed from `lunch-wizard-1`, facilitating the necessary cross-site data exchange and job scheduling.

3.2. Lunch-Wizard-1 Setup

- **SSH Access:** Similar to `lunch-wizard-8`, SSH access is open to all IPs (Source: `0.0.0.0/0`), allowing external management.
- **Intra-Site Traffic:** All traffic within `lunch-wizard-1` is unrestricted, enabling seamless communication between nodes in Site 2.
- **Inter-Site Communication:** All traffic from `lunch-wizard-8` is permitted, allowing Site 2 to interact with Site 1 for job distribution and shared data access.



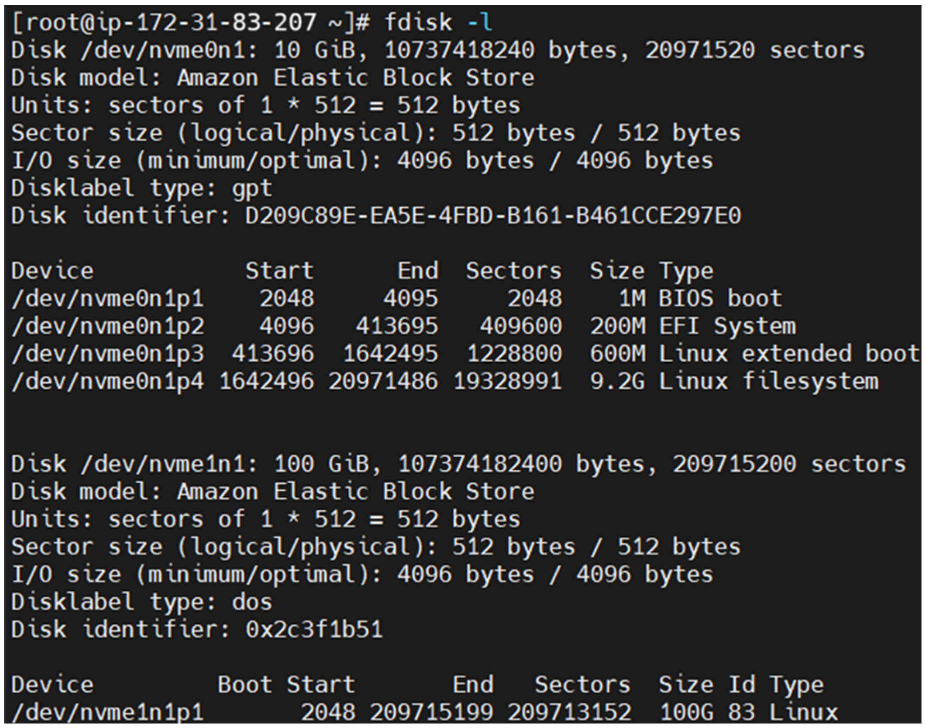
Security group rule ID	Type Info	Protocol Info	Port range Info	Source Info		Description - optional Info	
sgr-0232adc267a49bf76	SSH	TCP	22	Cus...	0.0.0.0/0		Delete
sgr-08d7b1b2b1b55214b	All traffic	All	All	Cus...	sg-0a96b8a21b sg-0a96b8a21be6df533		Delete
–	All traffic	All	All	Cus...	sg-0e2853634b sg-0e2853634bd402866		Delete

4. Mounting the Attached Volume on the Manager Node

To set up the storage volumes on the manager nodes, we first mounted the attached 100 GiB volumes on both sites.

Site 1 - Manager Node:

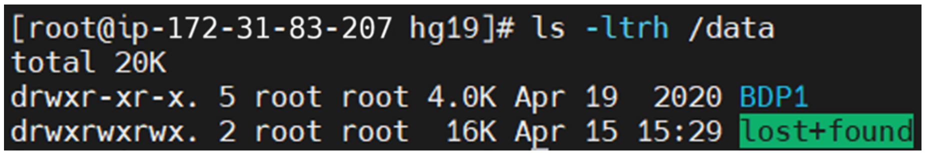
1. **Identify the Attached Volume:** Use the `fdisk {l` command to list the available volumes and identify the newly attached volume.
2. **Create a Mount Directory:** Execute the command `mkdir /data` to create the directory where the volume will be mounted.



```
[root@ip-172-31-83-207 ~]# fdisk -l
Disk /dev/nvme0n1: 10 GiB, 10737418240 bytes, 20971520 sectors
Disk model: Amazon Elastic Block Store
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disklabel type: gpt
Disk identifier: D209C89E-EA5E-4FBD-B161-B461CCE297E0

Device           Start      End  Sectors  Size Type
/dev/nvme0n1p1    2048     4095     2048    1M BIOS boot
/dev/nvme0n1p2    4096   413695   409600  200M EFI System
/dev/nvme0n1p3  413696  1642495  1228800  600M Linux extended boot
/dev/nvme0n1p4 1642496 20971486 19328991  9.2G Linux filesystem


Disk /dev/nvme1n1: 100 GiB, 107374182400 bytes, 209715200 sectors
Disk model: Amazon Elastic Block Store
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disklabel type: dos
Disk identifier: 0x2c3f1b51

Device         Boot Start       End   Sectors  Size Id Type
/dev/nvme1n1p1       2048 209715199 209713152  100G 83 Linux
```

Figure 3: Check

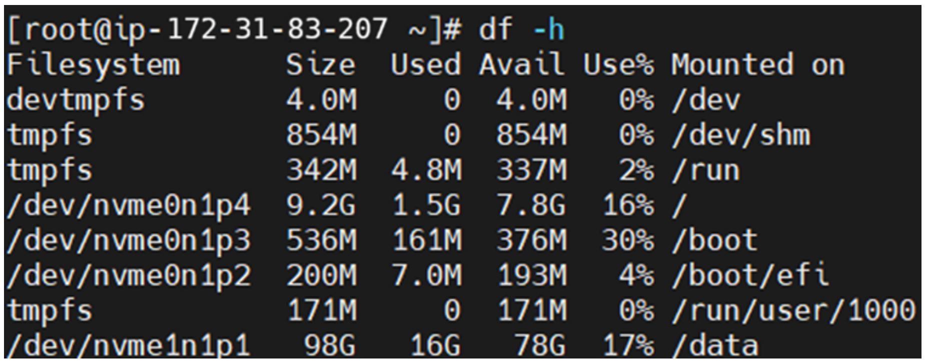
3. **Configure Mounting at Boot:** Edit the `/etc/fstab` file to ensure that the volume is mounted automatically at boot by adding the appropriate line: `/dev/nvme1n1p1 /data ext4 defaults 0 0`
4. **Activate the Mount:** Run the following commands to activate the mount and apply necessary permissions:
   - `systemctl daemon-reload`
   - `mount -a`
   - `chmod 775 /data`



```
[root@ip-172-31-83-207 hg19]# ls -ltrh /data
total 20K
drwxr-xr-x. 5 root root 4.0K Apr 19  2020 BDP1
drwxrwxrwx. 2 root root  16K Apr 15 15:29 lost+found
```

5. **Verify the Mount:** Use the command `df -h` to confirm that the volume has been successfully mounted and is available for use.

The process is then **repeated for Site 2**, following the same steps to mount the volume on the second manager node.



```
[root@ip-172-31-83-207 ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           854M     0  854M   0% /dev/shm
tmpfs           342M  4.8M  337M   2% /run
/dev/nvme0n1p4  9.2G  1.5G  7.8G  16% /
/dev/nvme0n1p3  536M  161M  376M  30% /boot
/dev/nvme0n1p2  200M  7.0M  193M   4% /boot/efi
tmpfs           171M     0  171M   0% /run/user/1000
/dev/nvme1n1p1   98G   16G   78G  17% /data
```

5. Configuring NFS to Allow Shared Volume Access

Next, we configured the Network File System (NFS) to enable shared access to the mounted volumes across the submit and execute nodes in both sites.

5.1. Server-Side Setup (Manager Node)

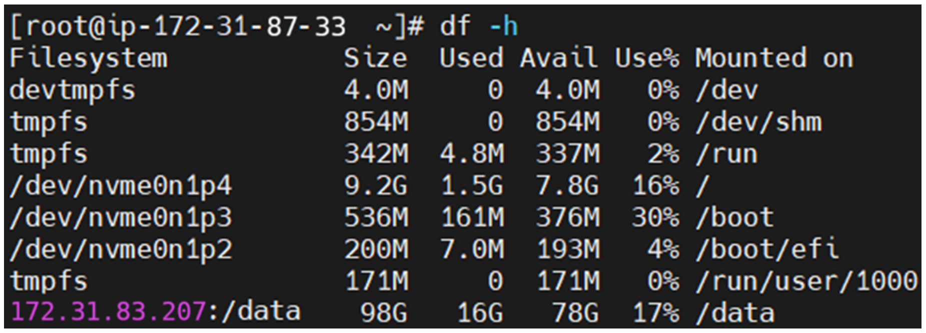
1. **Install Necessary Software:** On the manager node, install NFS utilities and services with the following command:
   `yum install nfs-utils rpcbind\\`
2. **Enable and Start Services:** Start the NFS and RPC services to enable file sharing across the network:
   - `systemctl enable nfs-server rpcbind`
   - `systemctl start rpcbind nfs-server`
   - Verify service status with `systemctl status nfs-server\\`
3. **Configure NFS Exports:** Edit the `/etc/exports` file to specify which IP addresses are allowed to access the shared `/data` directory:
   - `/data 172.31.87.33(rw,sync,no_wdelay)` for Site 1 Submit and Execute Nodes
   - `/data 172.31.30.85(rw,sync,no_wdelay)` for Site 2 Submit and Execute Nodes

4. **Apply Configuration:** Use the command `exportfs -r` to apply the new NFS configuration.

5.2. Client-Side Setup (Submit and Execute Nodes)

1. **Install NFS Utilities:** On the submit and execute nodes, install NFS support using `yum install nfs-utils`.
2. **Prepare the Mount Point:** Create a directory `/data` to mount the NFS share.

3. **Automate Mounting at Boot:** Edit the `/etc/fstab` file to automate the NFS mount by adding the following line (adjusting for the correct server IP): `<SERVER_PRIVATE_IP>:/data /data nfs defaults 0 0`
   For example, for Site 1: `172.31.83.207:/data /data nfs defaults 0 0`.
4. **Mount the NFS Share:** Run `mount -a` to mount the NFS volume.
5. **Verify the Mount:** Use `df -h` to ensure that the NFS share has been successfully mounted.



```
[root@ip-172-31-87-33  ~]# df -h
Filesystem            Size  Used Avail Use% Mounted on
devtmpfs              4.0M     0  4.0M   0% /dev
tmpfs                 854M     0  854M   0% /dev/shm
tmpfs                 342M  4.8M  337M   2% /run
/dev/nvme0n1p4        9.2G  1.5G  7.8G  16% /
/dev/nvme0n1p3        536M  161M  376M  30% /boot
/dev/nvme0n1p2        200M  7.0M  193M   4% /boot/efi
tmpfs                 171M     0  171M   0% /run/user/1000
172.31.83.207:/data    98G   16G   78G  17% /data
```

This process was repeated for Site 2, using the respective IP addresses for its submit and execute nodes to ensure seamless data sharing between the sites.

6. Installing HTCondor on Different Nodes

To establish a distributed computing environment, HTCondor was installed on the manager, submit, and execute nodes. Each installation was configured based on the specific role of the node within the HTCondor cluster.

6.1. Installation on the Manager Node

The manager node acts as the central controller for job submission and execution across the system. The following command was used for installation:
$curl{-}fsSLhttps://get.htcondor.org|sudoGET_HTCONDOR_PASSWORD = "BDP1_2024"/bin/bash - s - -no - dry - run - -central - manager$

6.1.1. Installation on the Submit Node

The submit node is responsible for submitting jobs to the cluster. The installation was carried out using the following command:
$curl{-}fsSLhttps://get.htcondor.org|sudoGET_HTCONDOR_PASSWORD = "BDP1_2024"/bin/bash - s - -no - dry - run - -submit$

6.2. Installation on the Execute Node

The execute node performs the tasks assigned by the manager and submit nodes. Its installation was done with the following command:
$curl{-}fsSLhttps://get.htcondor.org|sudoGET_HTCONDOR_PASSWORD = "BDP1_2024"/bin/bash - s - -no - dry - run - -execute$

6.3. ***Verifying Installation***

To ensure the installation was successful, the command `condor_status` was run on the manager node to verify the system status and confirm that the nodes were communicating correctly.

7. Installing WebDAV on Manager Node and Configuring Client Access

To facilitate efficient file transfer between sites, WebDAV was installed on the manager node. This allowed for HTTP-based file sharing within the distributed system.

7.1. Install and Configure Apache HTTP Server

Apache HTTP server was installed on the manager node, and its configuration was modified to enable WebDAV. The following steps were taken:

1. Install Apache:

```
yum install httpd
```

2. Disable the welcome page:

```
sed -i ’s/^/#&/g’ /etc/httpd/conf.d/welcome.conf
```

3. Modify the configuration for WebDAV:

```
sed -i "s/Options Indexes FollowSymLinks/Options FollowSymLinks/" /etc/httpd/conf/httpd.conf
systemctl start httpd.service
```

4. Verify DAV modules:

```
httpd -M | grep dav
```

Expected output: - $dav_module(shared) - dav_fs_module(shared) - dav_lock_module(shared)$

Setup WebDAV Directory and Permissions: 1. Create the WebDAV directory and set appropriate permissions:

```
mkdir /var/www/html/webdav
chown -R apache:apache /var/www/html
chmod -R 755 /var/www/html
```

Create User Account for WebDAV Access
A user account was created for access to the WebDAV directory:

```
htpasswd -c /etc/httpd/.htpasswd user001
chown root:apache /etc/httpd/.htpasswd
chmod 640 /etc/httpd/.htpasswd
```

Configure WebDAV Access in Apache
The ‘/etc/httpd/conf.d/webdav.conf‘ file was edited to enable WebDAV access with authentication:

```
DavLockDB /var/www/html/DavLock
<VirtualHost *:80>
  ServerAdmin webmaster@localhost
  DocumentRoot /var/www/html/webdav/
  ErrorLog /var/log/httpd/error.log
  CustomLog /var/log/httpd/access.log combined
  Alias /webdav /var/www/html/webdav
  <Directory /var/www/html/webdav>
    DAV On
    AuthType Basic
    AuthName "webdav"
    AuthUserFile /etc/httpd/.htpasswd
    Require valid-user
  </Directory>
</VirtualHost>
```

Disable SELinux and Restart Apache: To ensure smooth operation, SELinux was disabled and the Apache server was restarted:

```
setenforce 0
systemctl restart httpd.service
```

—

7.2. Client-Side Setup (Site 2 - Manager Node)

To access the WebDAV server from the manager node at Site 2, the following steps were performed:
1. Install Cadaver (WebDAV Client)

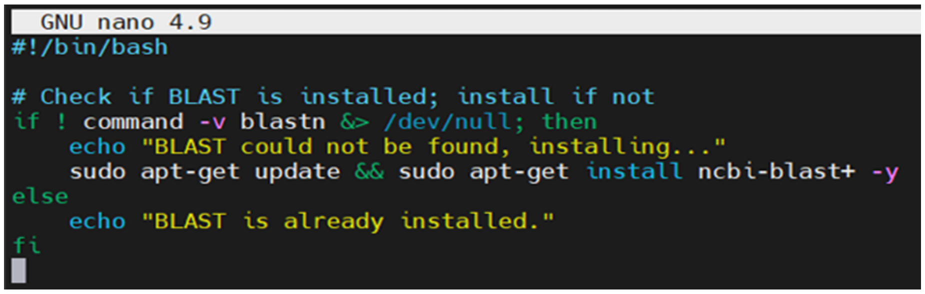
```
yum install cadaver
```

2. Connect to WebDAV Server: The following command was used to connect to the WebDAV server from Site 2:

```
cadaver http://<your-server-ip>/webdav/
```

8. Environment Setup Script (setup_environment.sh)

This script will check if BLAST is installed on the nodes and install it if it isn't. It should be run on each execute node. 1.The script checks if blastn (the command-line tool from the BLAST+ suite) is available. 2. If it is not found, the script updates the package lists for upgrades of packages that need upgrading, as well as new packages that have just come to the repositories. Then it installs the BLAST+ package. 3. If BLAST is already installed, it prints a message indicating so.



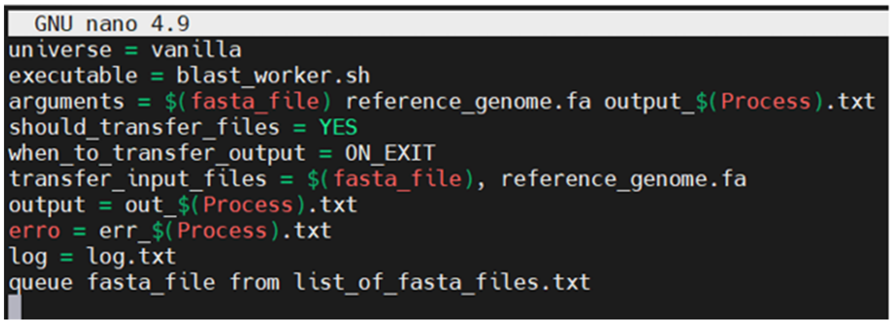
```
 GNU nano 4.9
#!/bin/bash

# Check if BLAST is installed; install if not
if ! command -v blastn &> /dev/null; then
    echo "BLAST could not be found, installing..."
    sudo apt-get update && sudo apt-get install ncbi-blast+ -y
else
    echo "BLAST is already installed."
fi
```

9. Worker Script (‘blast_worker.sh‘)

This script will be executed on each worker node. It takes a fasta file, runs BLAST alignment against a reference genome, and saves the output 1.Variable Assignment: Takes command line arguments and assigns them to variables. 2.BLAST Command: Runs blastn with the input fasta file as the query, the reference genome as the database, and outputs the results to a specified file.

10. Job Submission File (‘blast_job.sub‘)

This is the HTCondor job submission file. It defines how the jobs should be distributed and managed. 1.**Universe**: Specifies the ”vanilla” environment,

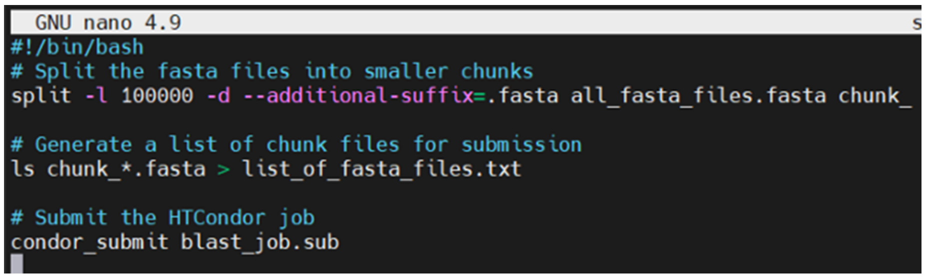


```
 GNU nano 4.9
universe = vanilla
executable = blast_worker.sh
arguments = $(fasta_file) reference_genome.fa output_$(Process).txt
should_transfer_files = YES
when_to_transfer_output = ON_EXIT
transfer_input_files = $(fasta_file), reference_genome.fa
output = out_$(Process).txt
erro = err_$(Process).txt
log = log.txt
queue fasta_file from list_of_fasta_files.txt
```

indicating a standard environment.
2.**Executable**: The script to be executed for each job, here blast_worker.sh.
3.**Arguments**: The arguments to pass to the executable; includes the input fasta file, the reference genome, and the name of the output file.
4.**File Transfers**: Specifies that files should be transferred and sets when outputs should be moved back to the submission directory.
5.**Queue:** Specifies that the executable should be run once for each fasta file listed in list_of_fasta_files.txt.

11. Main Submission Script (‘submit_jobs.sh‘)

This script will be used on the manager node to split the fasta files into manageable chunks and submit the jobs. 1.**File Splitting**: Splits the main fasta



```
  GNU nano 4.9                                                          s
#!/bin/bash
# Split the fasta files into smaller chunks
split -l 100000 -d --additional-suffix=.fasta all_fasta_files.fasta chunk_

# Generate a list of chunk files for submission
ls chunk_*.fasta > list_of_fasta_files.txt

# Submit the HTCondor job
condor_submit blast_job.sub
```

file into smaller chunks containing 100,000 lines each. This helps in distributing the workload evenly across nodes.
2.**List Generation**: Generates a list of the chunk files created in the previous step. This list is used by the HTCondor submission file to queue jobs.
3.**Job Submission**: Submits the jobs to the HTCondor scheduler using the blast_job.sub file.

12. Estimation

12.0.1. Time Estimation for Analyzing 300 Million Sequences Against the hg38 Database

To estimate the time required to analyze 300 million sequences using AWS infrastructure with **m5.4xlarge** instances, we base our calculations on the performance of BLAST (Basic Local Alignment Search Tool) running on these instances.

Performance of BLAST on m5.4xlarge Instances. An **m5.4xlarge** instance provides 16 vCPUs. Based on performance assumptions, each vCPU is capable of processing approximately 8 sequences per minute. Taking into account parallelization across all 16 vCPUs, the total processing capacity for one instance is:

$$\text{Sequences per minute per instance} = 16 \times 8 = 128\,\text{sequences/minute}$$

Batch Processing and Job Division. To efficiently manage the workload, the 300 million sequences will be divided into smaller batches. Each batch will be processed in parallel across multiple **m5.4xlarge** instances.

Time Estimation Calculation. Using the above throughput, we can estimate the total time required for a single instance to process all 300 million sequences.

$$\text{Total sequences} = 300,000,000$$

$$\text{Throughput per instance} = 128\,\text{sequences/minute}$$

$$\text{Total minutes per instance} = \frac{300,000,000}{128} = 2,343,750\,\text{minutes}$$

$$\text{Total hours per instance} = \frac{2,343,750}{60} = 39,063\,\text{hours}$$

$$\text{Total days per instance} = \frac{39,063}{24} = 1,627\,\text{days}$$

Scaling with Multiple Instances. By deploying **30 m5.4xlarge instances**, the total processing time can be significantly reduced, as each instance handles a portion of the workload in parallel. The estimated time with 30 instances is calculated as:

$$\text{Days with 30 instances} = \frac{1,627}{30} \approx 54\,\text{days}$$

Real-World Efficiency Adjustments. While the above estimate provides a theoretical processing time, real-world factors can influence the overall duration:

- **Network Latency and Job Scheduling Overhead:** These factors may introduce additional delays, particularly when distributing jobs and collecting results across the cloud infrastructure.
- **Data Transfer Speeds:** Large datasets may require time to be moved between nodes and storage, adding to the total processing time.
- **Spot Instance Interruptions:** If Spot Instances are used to lower costs, there is a risk of job interruptions. This could result in jobs needing to restart or be reallocated, extending the overall runtime.

Impact of Spot Instance Interruptions. Using Spot Instances introduces a variable risk of instance termination, potentially delaying jobs and increasing the overall time required to complete the analysis. These interruptions may necessitate job rescheduling, thereby affecting the final time estimate.

Conclusion. Under optimal conditions, utilizing **30 m5.4xlarge instances** would result in an estimated processing time of **54 days** to analyze the 300 million sequences against the hg38 database. However, real-world conditions, such as network latency, job overhead, and potential Spot Instance interruptions, could extend this time.

For more accurate projections, it is recommended to conduct a pilot run using a smaller subset of the data. This would provide real-world performance metrics, allowing the infrastructure to be fine-tuned for improved efficiency and cost-effectiveness.

13. Cost calculation

Key Factors in Cost Calculation.

13.1. Instance Costs

The selected instance type, **c5.4xlarge**, costs \$0.68 per hour for On-Demand usage. While there is the potential to reduce this cost by using Spot Instances, for the sake of clarity, this section will focus on On-Demand pricing.

13.2. Data Storage Costs

For storing input and output files, Amazon S3 and Amazon EFS are two common options:

- **Amazon S3**: The cost for Standard storage is \$0.023 per GB.
- **Amazon EFS**: Standard storage pricing is \$0.30 per GB (regional).

13.3. Data Transfer Costs

- **Inbound Data Transfer**: Data transfer into AWS is free.
- **Outbound Data Transfer**: Data transfer out of AWS to the internet typically costs \$0.09 per GB for the first 10 TB.

13.4. Other Potential Costs

Additional costs might include network usage, monitoring services (e.g., AWS CloudWatch), and other AWS services such as Lambda functions for job orchestration.

13.5. Instance Cost Calculation

From previous analysis, we estimated that processing 300 million sequences using a single **c5.4xlarge** instance would take approximately 1,627 days. By deploying 30 instances, we can reduce the total time to 54 days.

Cost per instance:

- One instance running for 54 days would total:

$$54\,\text{days} \times 24\,\text{hours/day} = 1,296\,\text{hours}$$

- The cost per instance is:

$$1,296\,\text{hours} \times 0.68\,\$/\text{hour} = 881.28\,\$\text{ per instance}$$

Total cost for 30 instances:

The total cost for running 30 instances over the course of 54 days is:

$$30\,\text{instances} \times 881.28\,\$ = 26,438.40\,\$ \text{ for the entire job}$$

13.6. Data Storage Costs

Assuming input data amounts to 1 TB and output data to 600 TB, the storage costs are as follows:

- **S3 Storage for Input Data**:

$$1,024\,\text{GB} \times 0.023\,\$/\text{GB} = 23.55\,\$ \text{ per month}$$

- **S3 Storage for Output Data**:

$$\text{Total storage cost} = 23.55\,\$\,(\text{input}) + 13,800\,\$\,(\text{output}) = 13,823.55\,\$\,\text{per month}$$

Total storage cost for 2 months:

With the job running for 54 days (approximately 2 months), the total storage cost is

$$13,823.55\,\$ \times 2 = 27,647.1\,\$$$

13.7. Data Transfer Costs

Transferring 600 GB of output data out of AWS will incur the following cost:

$$\text{Cost of data transfer out} = 600\,\text{TB} \times 0.09\,\$/\text{GB} = 54,000\,\$$$

13.8. Other Costs

We estimate an additional \$100 for network usage, monitoring (CloudWatch), and miscellaneous AWS services.

14. Cost Reduction Strategies for Processing 300 Million Sequences

14.1. Spot instances vs On Demand Instances

Using Spot Instances could offer significant savings over On-Demand pricing, with discounts typically ranging from 50% to 90%. For example:

- **On-Demand cost for 30 instances**: \$26,438.40
- **Potential Spot Instance savings**: Assuming a 65% discount, the cost could be reduced to approximately:

However, Spot Instances come with the risk of interruption, which may require rescheduling or restarting jobs. This could extend the overall job time but would still likely result in significant cost savings.

$$26{,}438.40 \times 0.35 = 9{,}253.44\,\$$$

14.2. ***Cost Reduction Using Burrows-Wheeler Transform (BWT)***

The Burrows-Wheeler Transform (BWT) is a compression technique that rearranges character sequences to group similar characters together, making the data more suitable for efficient compression. In genomic data, such as DNA sequences, repetitive patterns are common. By using BWT, these patterns can be better organized, facilitating compression and significantly reducing storage and data transfer costs. Genomic tools like SAMtools and BWA (Burrows-Wheeler Aligner) apply BWT to compress and align genomic data. This leads to the reduction of large file sizes, such as BAM files, ultimately saving storage space and reducing the time required for data transfer.

Compression Factor. For genomic data, compression algorithms that leverage BWT can achieve compression ratios between 3:1 and 5:1, depending on the sequence complexity and repetitiveness. In some cases, particularly with highly repetitive sequences, even higher compression ratios can be achieved with optimized compression methods. Assuming a conservative average compression ratio of **4:1** with the use of BWT and additional compression algorithms.

- **Without Compression (Uncompressed Data)**: 600 TB
- **With BWT Compression**: 600 TB ÷ 4 = 150 TB

Cost Calculation Without BWT (Uncompressed Data)

- **Total Storage Requirement**: 600 TB
- **Monthly Storage Cost**:

$$600{,}000\,\text{GB} \times 0.023\,\$/\text{GB} = 138{,}000\,\$/\text{month}$$

Cost Calculation With BWT (Compressed Data)

- **Total Storage Requirement**: 150 TB
- **Monthly Storage Cost**:

$$150{,}000\,\text{GB} \times 0.023\,\$/\text{GB} = 3{,}450\,\$/\text{month}$$

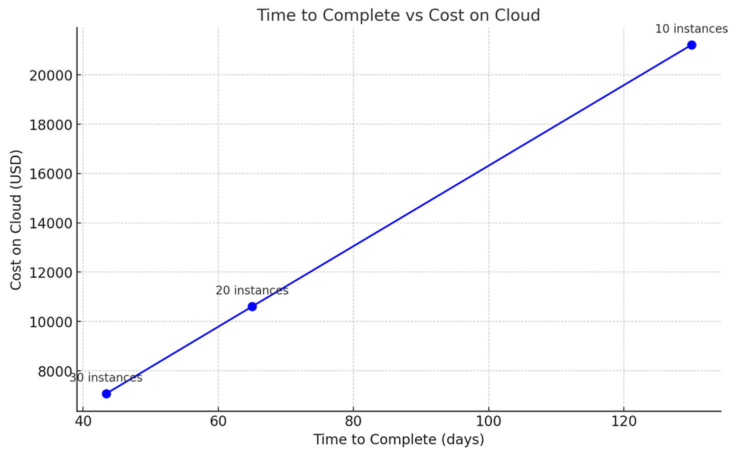
Cost Reduction & Percentage Reduction

$$13800\$ - 3450\$ = 10,350\$/\text{month}$$

$$\left(\frac{10,350\$}{13,800\$}\right) \times 100 = 75\%$$

15. Cost on Cloud vs Time to complete

The plot illustrates the relationship between the total processing time and the associated costs for completing a task on the cloud, assuming no charges for network bandwidth. As the number of instances increases, the time to complete the task decreases due to parallel processing, but the overall cost rises proportionally.



- **Time Reduction:** By deploying more instances, the time to complete the task significantly decreases, as demonstrated by the steep drop in time-to-complete with each increase in the number of instances.
- **Cost Trade-Off:** The cost increases linearly with more instances, as more computational resources are utilized, even though they shorten the processing time. Each instance contributes additional cost but accelerates the task's completion.
- **Optimal Balance**: The key takeaway is to find a balance between minimizing the time for completing the job and controlling cloud expenses, depending on the urgency of the task and the budget available.