IT00CE11-3005

Cloud Computing

Assignment # 2

September 19, 2024

VM performance and cost-efficiency | STUDENT NO-2402262

Saad Abdullah

saad.abdullah@abo.fi

Table of Contents

VM performance and cost-efficiency comparisons

Task 1: Link to openbenchmarking profile.

Below is the link to my openbenchmarking profile:

https://openbenchmarking.org/user/iemsaadabdullah

Task 2: Report on selected instance types

Below is my reasoning for selecting the instances I choose.

T3a.medium:

I chose the T3a.medium instance because it's a burstable instance type. This means it provides consistent, baseline performance but can handle short bursts of higher demand when needed. This is particularly useful for workloads with occasional spikes in CPU usage. Another key reason is the cost savings—T3a instances offer about 10% better cost efficiency compared to other T-class instances, which aligned perfectly with my focus on balancing performance and cost.

M5.large:

The M5.large instance stood out to me as it's part of AWS's general-purpose family. It's designed to handle a variety of workloads by offering a balanced mix of compute, memory, and network resources. I felt this would be a good instance to test since it's ideal for applications that don't need to excel in just one area but require overall moderate and steady performance. It seemed like a strong choice.

C5.large:

I picked the C5.large instance because it's optimized for compute-intensive tasks. My goal was to see how this instance performs compared to others, particularly in terms of its cost-to-compute ratio. The C5.large offers solid compute power at a relatively low price, making it a good fit for testing whether upgrading to a larger instance would be worth it for more demanding compute-heavy tasks.

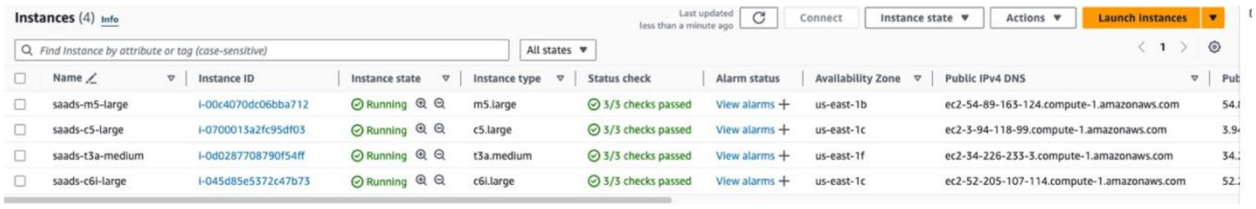
C6i.large:

I decided to test the C6i.large instance because AWS claims it delivers a 15% better price-to-performance ratio compared to the C5 family, and that too at the same price($0.085 per hour in Virginia). This seemed like a great opportunity to test whether this newer instance truly provides better performance in practice, given the same price point. I considered using the C7i instance instead but skipped it because it was more costly than c6i.

Conclusion:

My main focus throughout the benchmarking was cost-efficiency. I initially wanted to test the T4g.medium instance as well, especially because it's based on ARM architecture, which has gained popularity for its efficiency and performance, particularly after Apple's introduction of the M1 chip. Unfortunately, I couldn't test it due to its unavailability in the student access tier.

The reason I moved from T3a.medium to M5.large was to explore whether it's worth spending more to change both the instance family and size, comparing the cost-to-performance ratio of a general-purpose family to a burstable one. Similarly, I transitioned from M5.large to C5.large to see if sticking with the same size but switching to a compute-optimized instance would deliver better value. Finally, I tested the C6i.large to determine whether upgrading within the same compute-optimized family would provide significant performance benefits.


Instances (4) Info
Last updated less than a minute ago
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
1

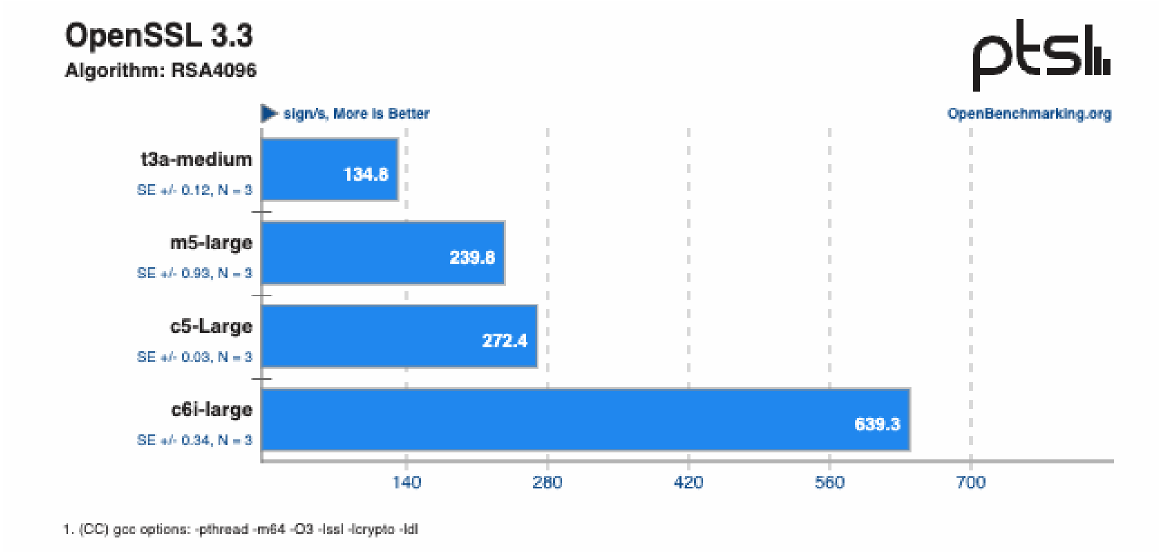
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Pub
saads-m5-large	i-00c4070dc06bba712	Running	m5.large	3/3 checks passed	View alarms +	us-east-1b	ec2-54-89-163-124.compute-1.amazonaws.com	54.
saads-c5-large	i-0700013a2fc95df03	Running	c5.large	3/3 checks passed	View alarms +	us-east-1c	ec2-3-94-118-99.compute-1.amazonaws.com	3.9
saads-t3a-medium	i-0d0287708790f54ff	Running	t3a.medium	3/3 checks passed	View alarms +	us-east-1f	ec2-34-226-233-3.compute-1.amazonaws.com	34.
saads-c6i-large	i-045d85e5372c47b73	Running	c6i.large	3/3 checks passed	View alarms +	us-east-1c	ec2-52-205-107-114.compute-1.amazonaws.com	52.

Task 3: Report on obtained results

Below is my analysis of the benchmarks.

OpenSSL RSA4096:

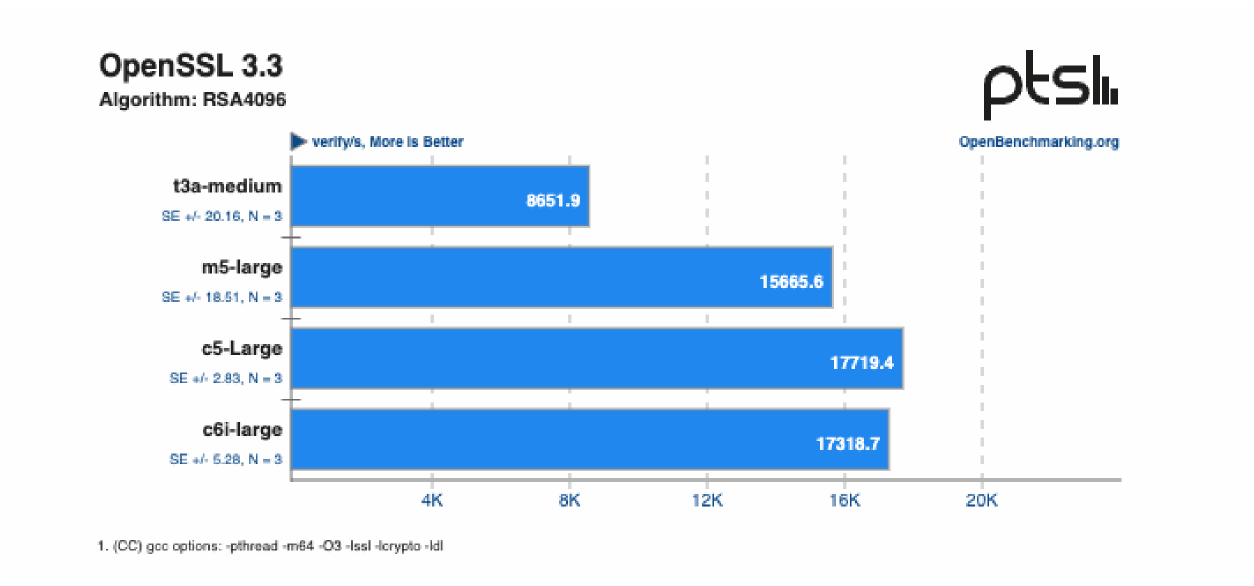
The OpenSSL RSA4096 benchmark measures the cryptographic performance of an instance when performing RSA operations with a 4096-bit key. For this benchmark, I got two matrices, and below is an explanation of these matrices.



Signing Performance (Signs/s):

This measures how many digital signatures an instance can generate using RSA4096 in one second. Digital signatures are used to verify the authenticity of data, and higher values indicate better performance.

- **T3a.medium** had the lowest performance maybe due to its burstable nature and lower sustained CPU power.
- **M5.large** showed nearly double the performance of T3a.medium, benefiting from its general-purpose design.
- **C5.large** beat M5.large as it’s optimized for compute-heavy tasks.
- **C6i.large** outperformed everyone by demonstrating the best performance, nearly tripling C5.large’s signing speed, showing clear architectural improvements for this workload.



Verification Performance (Verify/s):

This measures how many signature verifications an instance can perform per second. Signature verification is essential for confirming the integrity of signed messages, so higher values also indicate better performance.

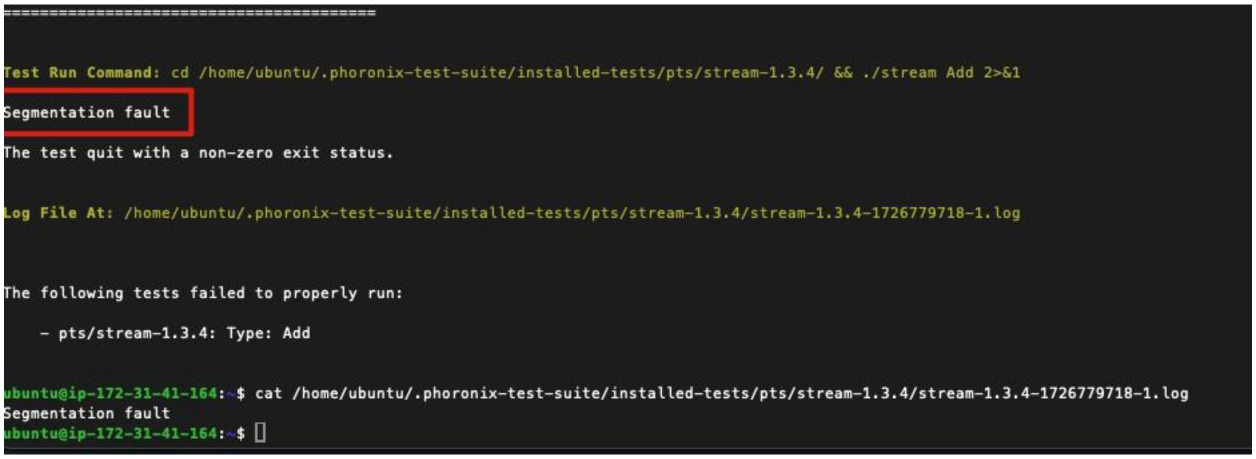
- **T3a.medium** again showed the lowest results.
- **M5.large** and C5.large both performed well, with C5.large slightly ahead.
- **C5.lage** was the winner in this test.
- **C6i.large** was close to C5.large but didn't outperform it, indicating the C5 family is still highly competitive for verification tasks.

Conclusion:

In terms of signing operations (Signs/s), performance improved significantly as I moved from T3a.medium to M5.large, with C6i.large more than doubling the performance of C5.large, making it the ideal choice for signing-heavy cryptographic tasks. However, for verification operations (Verify/s), while there was also an improvement, the difference between C5.large and C6i.large was smaller, with C5.large slightly outperforming C6i.large, suggesting that C5.large might be the more cost-effective option for verification-heavy workloads.

Stream:

The Stream Add benchmark measures the memory bandwidth of the instance by testing its ability to perform vector addition. The scale is measured in MB/s, meaning higher numbers indicate better performance in handling memory-intensive tasks. Unfortunately, the test could not be completed for the C6i.large instance despite attempts to troubleshoot, possibly due to an infrastructure issue, as can be seen in the error logs.

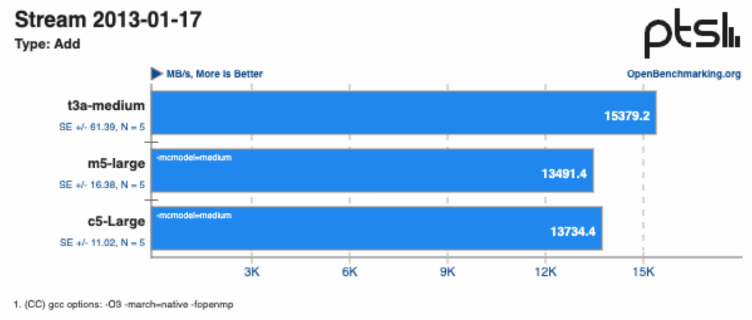


```
========================================

Test Run Command: cd /home/ubuntu/.phoronix-test-suite/installed-tests/pts/stream-1.3.4/ && ./stream Add 2>&1

Segmentation fault

The test quit with a non-zero exit status.

Log File At: /home/ubuntu/.phoronix-test-suite/installed-tests/pts/stream-1.3.4/stream-1.3.4-1726779718-1.log


The following tests failed to properly run:

   - pts/stream-1.3.4: Type: Add

ubuntu@ip-172-31-41-164:~$ cat /home/ubuntu/.phoronix-test-suite/installed-tests/pts/stream-1.3.4/stream-1.3.4-1726779718-1.log
Segmentation fault
ubuntu@ip-172-31-41-164:~$ 
```

Now below are the results for the stream test.

Stream

This is a benchmark of Stream, the popular system memory (RAM) benchmark. **Learn more via the OpenBenchmarking.org test page.**



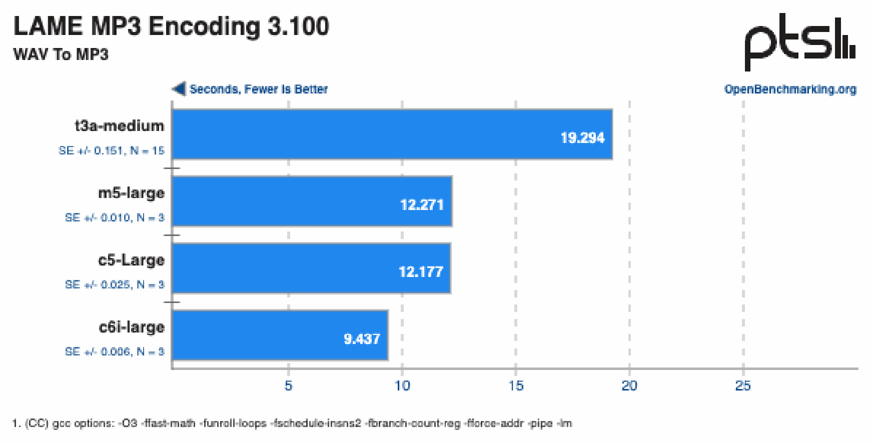
- **T3a.medium** unexpectedly outperformed the other instances in this benchmark. Since burstable instances can occasionally deliver high performance during short bursts, this could explain the unexpected result.
- **M5.large** delivered significantly lower performance compared to T3a.medium in this test, despite its general-purpose design. This suggests that the Stream Add benchmark may have favored the burstable nature of T3a.medium in this particular instance.
- **C5.large** showed a slight improvement over M5.large, which aligns with its optimization for compute-heavy tasks. However, both M5.large and C5.large delivered notably lower results compared to T3a.medium in this memory test.

Conclusion:

The results were surprising, with T3a.medium showing the best memory performance, contrary to expectations. However, C5.large remains a strong contender for workloads requiring a balance between memory bandwidth and compute power.

Encode MP3:

The Encode MP3 benchmark measures the time (in seconds) taken to convert a WAV file to MP3 format. Lower times indicate better performance, as faster encoding is desirable for media processing tasks.



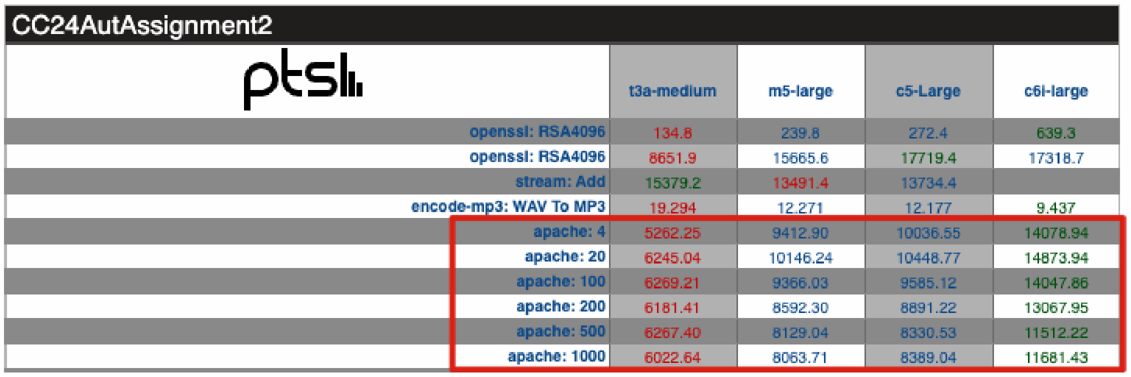
- **T3a.medium** had the slowest time, taking 19.294 seconds to complete the task, which is expected due to its lower sustained CPU power.
- **M5.large** showed a significant improvement, completing the task in 12.271 seconds, which reflects its general-purpose design that balances memory and compute power.
- **C5.large** slightly outperformed M5.large, completing the encoding in 12.177 seconds, showcasing its optimization for compute-heavy tasks.
- **C6i.large** demonstrated the best performance by far, taking only 9.437 seconds to complete the encoding, highlighting its advanced architecture and efficiency in handling media encoding tasks.

Conclusion:

The C6i.large instance clearly outperformed the others, making it ideal for media processing tasks like MP3 encoding. Both M5.large and C5.large performed similarly. The T3a.medium instance lagged behind, as expected, given its cost-efficient and burstable nature.

Apache:

The Apache Benchmark test measures how many requests per second a web server can handle while processing a given number of requests. It simulates different levels of load on the server and evaluates the server's ability to handle HTTP requests efficiently. It uses Requests per second (RPS) as matric which measures how many client requests a server can handle in one second. A higher RPS is better because it indicates that the server can efficiently handle more requests.



CC24AutAssignment2

	t3a-medium	m5-large	c5-Large	c6i-large
openssl: RSA4096	134.8	239.8	272.4	639.3
openssl: RSA4096	8651.9	15665.6	17719.4	17318.7
stream: Add	15379.2	13491.4	13734.4	
encode-mp3: WAV To MP3	19.294	12.271	12.177	9.437
apache: 4	5262.25	9412.90	10036.55	14078.94
apache: 20	6245.04	10146.24	10448.77	14873.94
apache: 100	6269.21	9366.03	9585.12	14047.86
apache: 200	6181.41	8592.30	8891.22	13067.95
apache: 500	6267.40	8129.04	8330.53	11512.22
apache: 1000	6022.64	8063.71	8389.04	11681.43

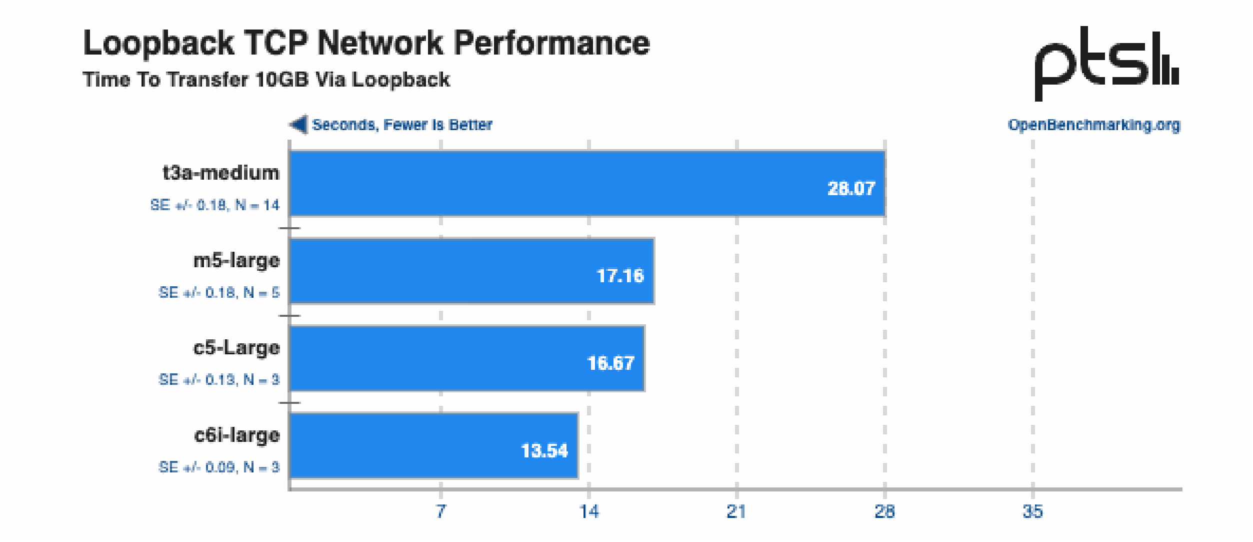
- **T3a.medium** consistently showed the slowest performance across all request loads (4, 20, 100, 200, 500, 1000). Despite being a burstable instance, it didn't show any significant advantage, at first I thought the burstable nature would help in the Apache test but sadly it didn't.
- **M5.large** offered a significant improvement over T3a.medium, benefiting from its general-purpose architecture.
- **C5.large** outperformed M5.large across all tests, showcasing its capability to handle compute-heavy and high-throughput tasks.
- **C6i.large** led in every test, demonstrating the best performance and handling the most requests per second, further confirming the benefits of its architectural enhancements for high-load web server tasks.

Conclusion:

Although I anticipated T3a.medium might show some interesting performance spikes due to its burstable nature, it did not deliver on that potential. All Apache tests demonstrated the same general behavior, with T3a.medium being the slowest and C6i.large consistently achieving the best results. Hence, I have summarized all Apache results under this single explanation instead of detailing each graph separately.

Network Loopback:

The Network-Loopback benchmark measures the time (in seconds) taken to transfer 10GB of data using the loopback interface (virtual network interface used by a computer to send and receive network traffic to itself, primarily for testing and internal communication). A lower value indicates better performance.



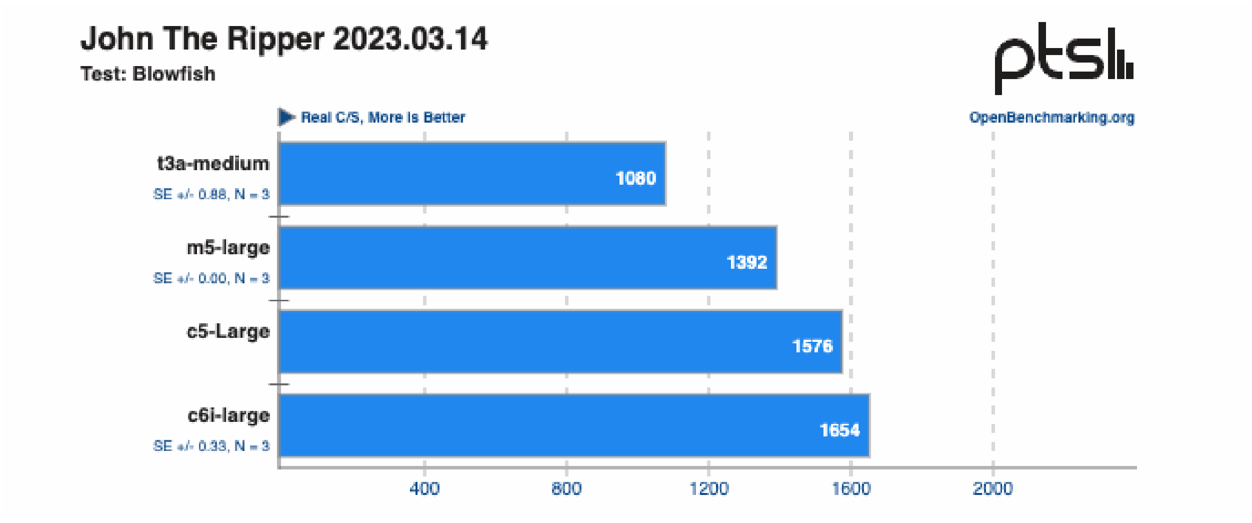
- **T3a.medium** had the slowest time, taking 28.071 seconds to transfer 10GB, which is expected given its lower overall performance in network-heavy operations.
- **M5.large** showed a significant improvement, completing the transfer in 17.16 seconds, reflecting its balanced performance across computing and networking tasks.
- **C5.large** performed slightly better than M5.large, completing the transfer in 16.67 seconds, showing its optimization for high-performance computing and networking.
- **C6i.large** delivered the best performance, transferring 10GB in 15.439 seconds, demonstrating the advantages of its enhanced architecture and networking capabilities.

Conclusion:

The C6i.large instance outperformed all others in this test, making it the best option for network-intensive tasks. The difference between C5.large and M5.large was minimal, but C5.large still had a slight edge. T3a.medium showed the slowest performance, as expected for a burstable instance.

John the Ripper:

The John the Ripper benchmark measures the time taken to crack passwords using the tool. A higher value indicates better performance, as it shows faster hash-cracking speeds. In this test, Real C/S is the combination of the candidate password and target hash per second. It reflects the computational throughput of the instance. That's why higher is better.



- **T3a.medium** had the worst performance, it tried 1080 combinations per second. The burstable nature of this instance didn't help it in this test.
- **M5.large** showed a better performance, taking 1392 combinations per second. This reflects its balanced performance across computing tasks, making it suitable for a variety of workloads.
- **C5.large** was better than M5.large with 1576 combinations per second.
- **C6i.large** was the winner, with 1654 combinations per second. This suggests that, with the same price, it performed better than c5-large, and it validated the claim of AWS.

Conclusion:

Overall, the C6i.large demonstrated the highest effectiveness for this test. There was no surprise in this test as the instances with more sustained resources performed better.

Task 4: Relative Cost Efficiency

So, in this section, I will try to evaluate the instances based on the selection criteria I described in task 2. My focus will be better cost-effectiveness, and for that, I'll choose benchmarks like OpenSSL, Apache 1000, and Stream(ADD). First of all, I'll get the costs of instances I used from **AWS Virginia region** per hour and per second to help me in calculations.

Instance Name	Price per hour in USD	Price per second in USD
T3a.medium	0.0376 $	0.00001044 $
M5.large	0.096 $	0.00002667 $
C5.large	0.085 $	0.00002361 $

C6i.large	0.085 $	0.00002361 $

Now, I'll tell a bit about how I am going to do the calculations So I have the AWS price per hour, and I know how much time each benchmark took in openbenchmarking. Using this information, I will make the comparisons per dollar computations so that we can compare results on the same scale**. For example**, the OpenSSL benchmark for t3a.medium did 134.8 signs/s, and the test took 5 minutes and 7 seconds.

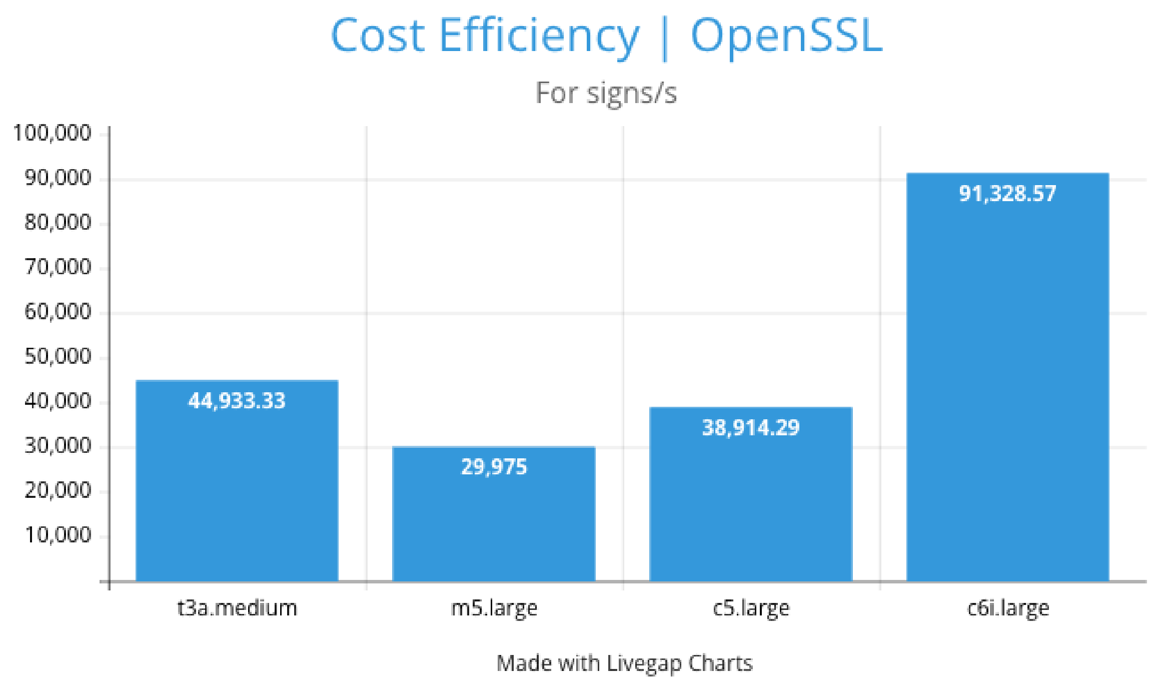
5*60 + 7 = 307 seconds, and according to the price per second of t3a.medium, the cost is approx. 0.003$. Now, for 1 dollar, the signs/s will be as follows (1/0.003)*134.8, which is approx. 44933.33 signs/s per dollar.

I followed the same calculations to convert the computations per dollar for other benchmarks as well. Keeping things concise I will present the data in tabular and graphical form instead of mathematics for clear understanding.

OpenSSL RSA4096:

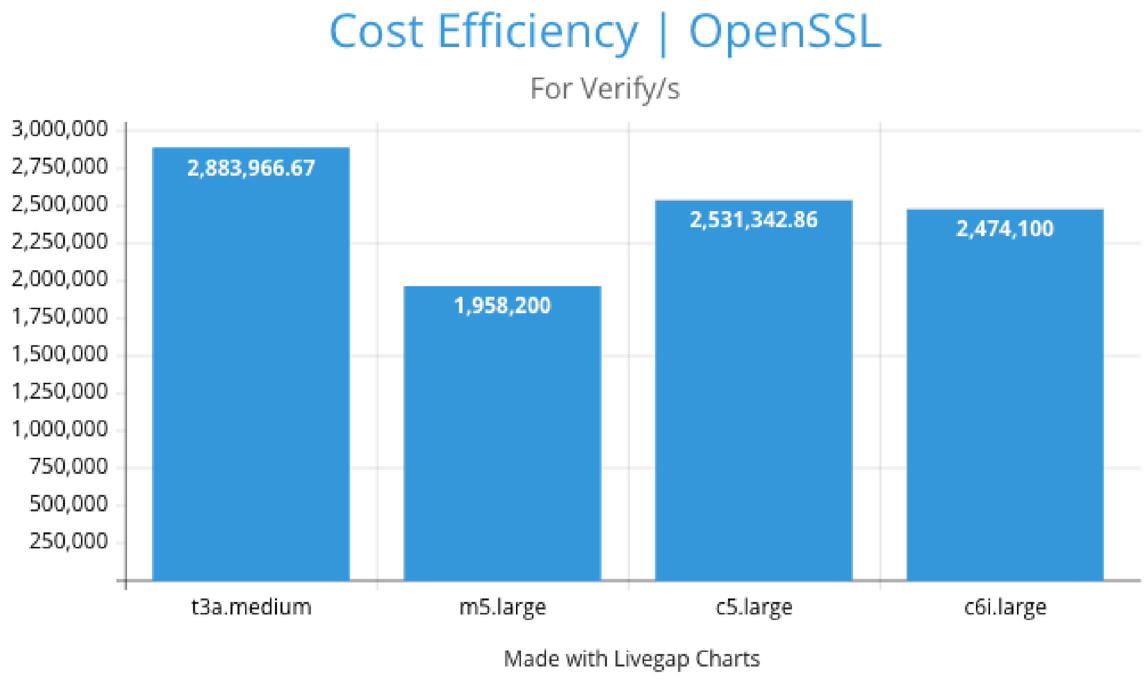
For Signs per second, the efficiency is as follows:

Instance	**Signs/s per Dollar**
T3a.medium	44933.33
M5.large	29975
C5.large	38914.29
C6i.large	91328.57



Now, Similarly, with Verify per second, the efficiency is as follows:

Instance	Verify/s per Dollar
T3a.medium	2883966.67
M5.large	1958200
C5.large	2531342.86
C6i.large	2474100



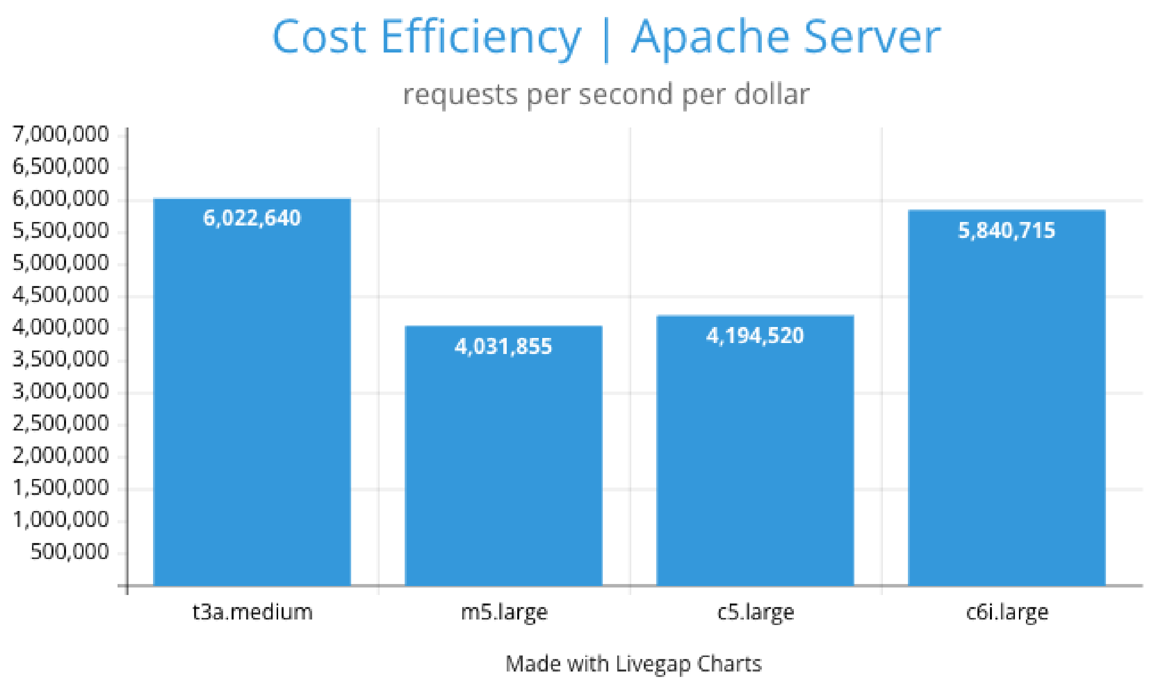
Conclusion:

I really liked the performance of t3a.medium as it's doing great, keeping in mind its price. Similarly, I will choose c6i over c5 because the price is the same, but c6i is beating c5. One key point to focus on is that it is **not necessary that upgrading the instance family will always bring better performance** this can be observed by upgrading t3a.medium with m5.large, by upgrading our effectiveness dropped.

Apache:

Since all Apache benchmark versions showed the same results, I'll go with the 1000 concurrent requests version. Below are the stats:

Instance	req/s per Dollar
T3a.medium	6022640
M5.large	4031855
C5.large	4194520
C6i.large	5840715



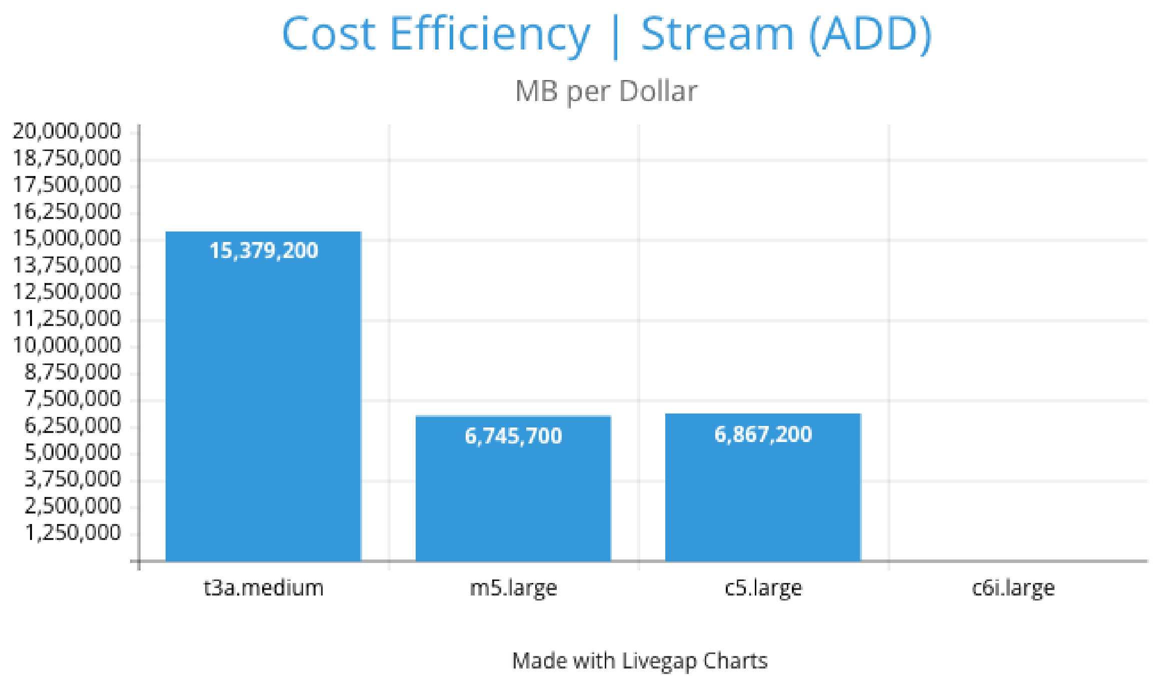
Conclusion:

The conclusion here supports the previous benchmark's conclusion. T3a.medium is performing really well as compared to its price. One key point to note here is to **focus on your needs while choosing the instance**. Because even though c5 is compute-intensive, it still performs better than m5 on a task that doesn't require much computation.

Stream:

In this case, the unit for comparison is MB per Dollar, since I was unable to perform this benchmark on c6i that's why its field is empty. Below are the stats:

Instance	MB per Dollar
T3a.medium	15379200
M5.large	6745700
C5.large	6867200
C6i.large	N/A



Conclusion:

As can be seen here, the t3a.medium is leading again. I am not sure about the c6i because enough data is not available. But one thing to note here is to **focus on burstable instances before upgrading the instance** as the burstable instance has shown some great results in compliance to the price point.

Cost-Efficiency Analysis:

Below are my findings so far:

- **T3a.medium** consistently offers the best cost-efficiency across multiple benchmarks (OpenSSL, Apache requests, and Stream). This suggests that for cost-constrained use cases, sticking with a burstable instance can provide substantial value, even if the raw performance is lower.
- **M5.large**, while showing strong performance improvements over T3a.medium, doesn't fare as well in cost-efficiency due to its higher per-second price. It might still be a good option if performance is the primary concern and cost is secondary.
- **C5.large** strikes a good balance between performance and cost, offering better cost-efficiency than M5.large in most cases, especially in compute-heavy tasks.
- **C6i.large** provides the best performance across the board but falls behind T3a.medium in cost-efficiency for certain tasks, particularly Apache and OpenSSL(verify/s). However, its clear advantage in OpenSSL signing (where it more than doubles the performance of C5.large) makes it the best option for specific, compute-heavy workloads.

Conclusion:

The **T3a.medium instance offers the highest cost-efficiency for general use cases**, especially if you're looking to balance costs with performance. However, **C6i.large stands out for specific high-**

performance tasks, particularly cryptographic workloads. My exploration from T3a.medium to M5.large did show some performance improvement but at a higher cost, which was justified in tasks where M5's general-purpose power was required. Transitioning from M5.large to C5.large proved more efficient, as the compute-optimized instance offered better value. Finally, the move to C6i.large demonstrated significant performance benefits, especially in compute-intensive benchmarks like OpenSSL, making it the top choice for tasks that can fully utilize its architecture.

Reflection:

Have you learned anything completely new?

Yes, doing cost comparisons for cloud services was new for me.

Did anything surprise you?

Yes, before the cost analysis, I was not bothered much about burstable instances, but after comparison, I was shocked at how it performed.

Did you find anything challenging? Why?

The mathematical part required a lot of attention but it was interesting.

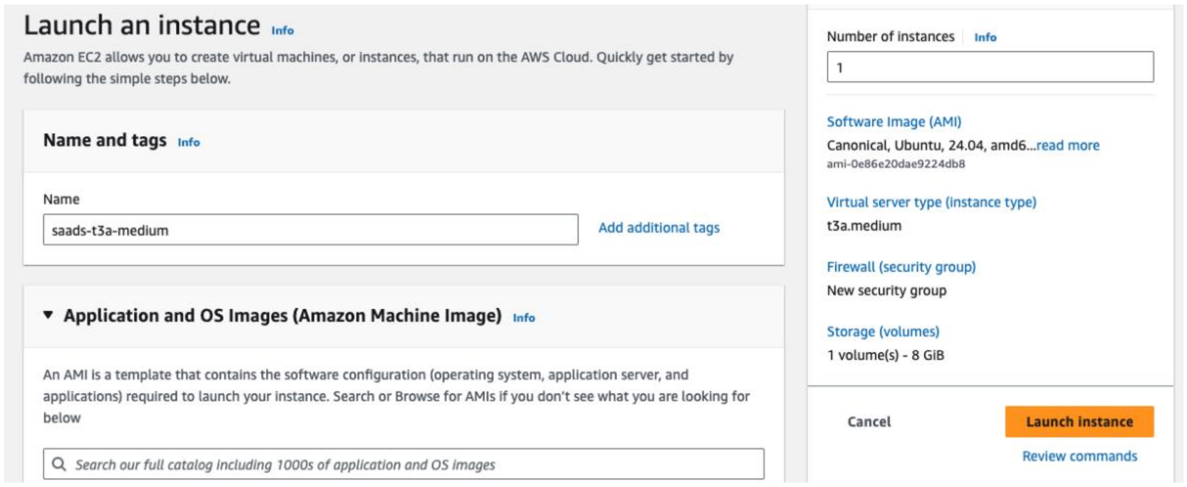
Did you find anything satisfying? Why?

Completing the assignment was satisfying ☺

Steps for benchmarking:

Below are the steps I followed to create instances and do benchmarking.

Setup the first t3a instance as following:



Launch an instance Info

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tags Info

Name: saads-t3a-medium

Add additional tags

▼ Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images

Number of instances Info: 1

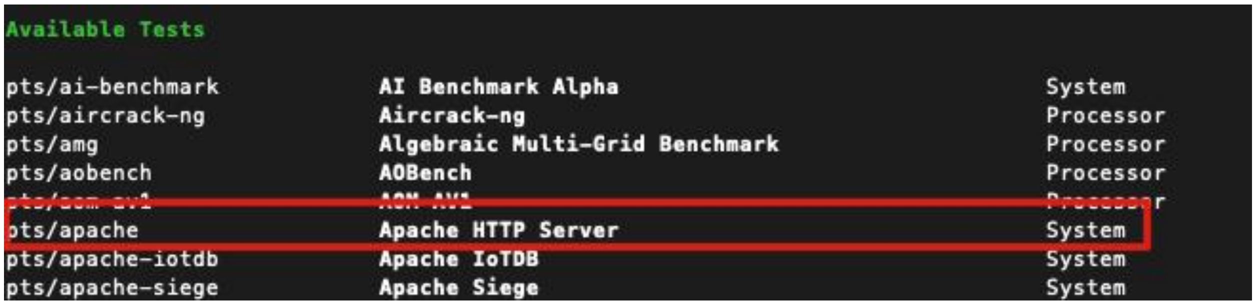
Software Image (AMI)
Canonical, Ubuntu, 24.04, amd6...read more
ami-0e86e20dae9224db8

Virtual server type (instance type)
t3a.medium

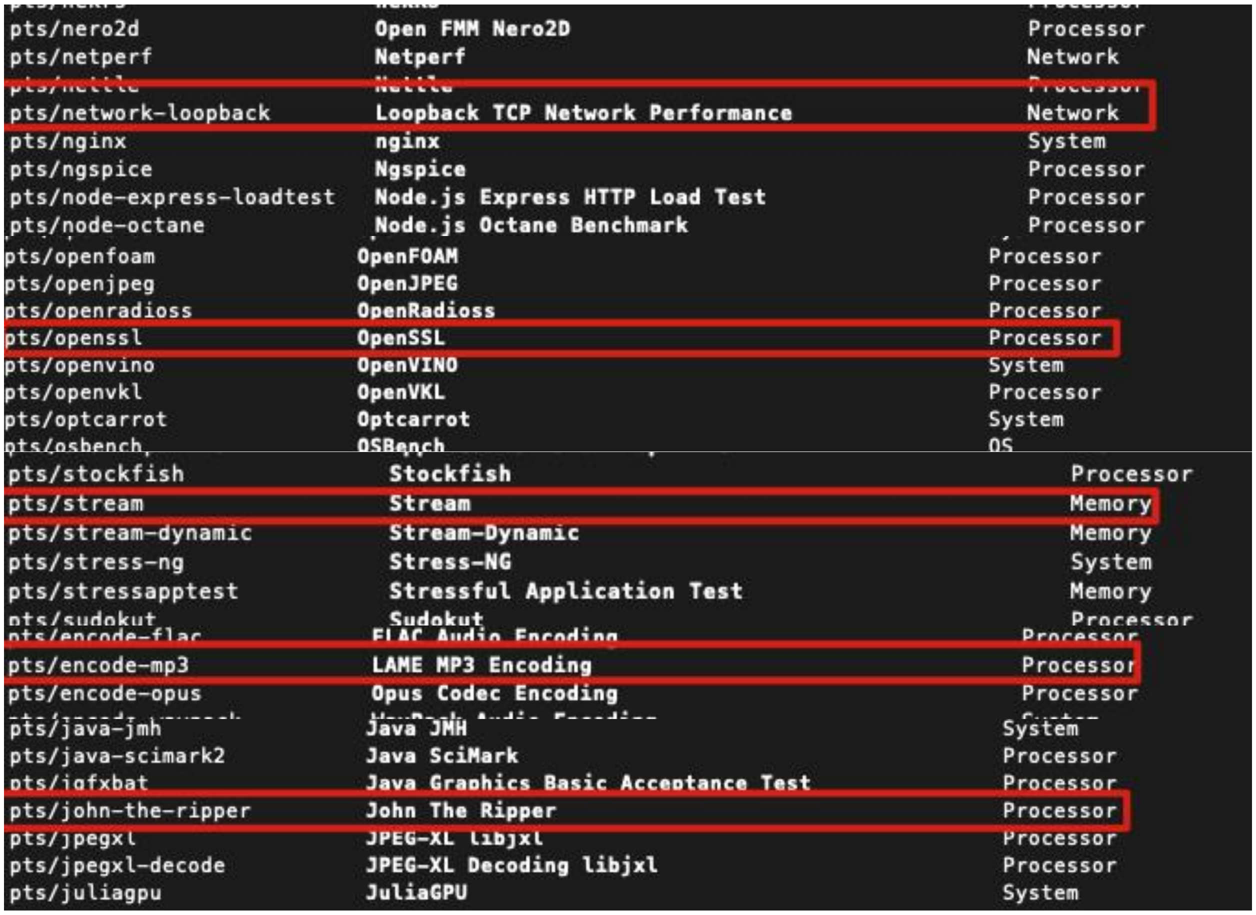
Firewall (security group)
New security group

Storage (volumes)
1 volume(s) - 8 GiB

Installed all required benchmarking tests in phoronix-test-suit

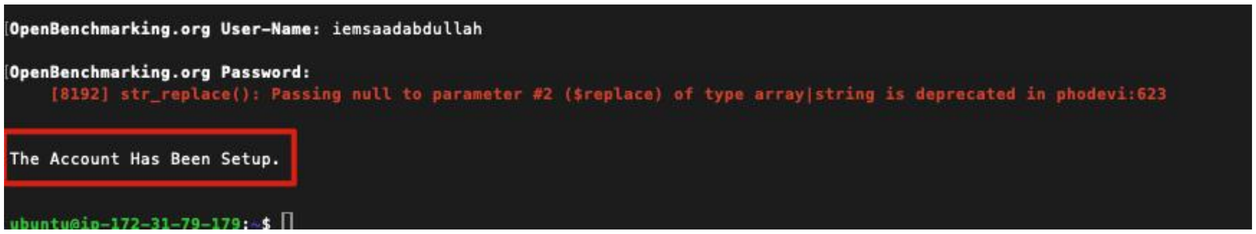


```
Available Tests

pts/ai-benchmark           AI Benchmark Alpha                         System
pts/aircrack-ng            Aircrack-ng                                Processor
pts/amg                    Algebraic Multi-Grid Benchmark             Processor
pts/aobench                AOBench                                    Processor
pts/aom-av1                AOM AV1                                    Processor
pts/apache                 Apache HTTP Server                         System
pts/apache-iotdb           Apache IoTDB                               System
pts/apache-siege           Apache Siege                               System
```



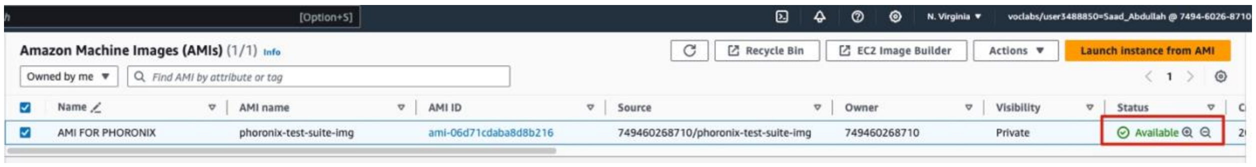
```
pts/nero2d                 Open FMM Nero2D                            Processor
pts/netperf                Netperf                                    Network
pts/nettle                 Nettle                                     Processor
pts/network-loopback       Loopback TCP Network Performance           Network
pts/nginx                  nginx                                      System
pts/ngspice                Ngspice                                    Processor
pts/node-express-loadtest  Node.js Express HTTP Load Test             Processor
pts/node-octane            Node.js Octane Benchmark                   Processor
pts/openfoam               OpenFOAM                                   Processor
pts/openjpeg               OpenJPEG                                   Processor
pts/openradioss            OpenRadioss                                Processor
pts/openssl                OpenSSL                                    Processor
pts/openvino               OpenVINO                                   System
pts/openvkl                OpenVKL                                    Processor
pts/optcarrot              Optcarrot                                  System
pts/osbench                OSBench                                    OS
pts/stockfish              Stockfish                                  Processor
pts/stream                 Stream                                     Memory
pts/stream-dynamic         Stream-Dynamic                             Memory
pts/stress-ng              Stress-NG                                  System
pts/stressapptest          Stressful Application Test                 Memory
pts/sudokut                Sudokut                                    Processor
pts/encode-flac            FLAC Audio Encoding                        Processor
pts/encode-mp3             LAME MP3 Encoding                          Processor
pts/encode-opus            Opus Codec Encoding                        Processor
pts/java-jmh               Java JMH                                   System
pts/java-scimark2          Java SciMark                               Processor
pts/jgfxbat                Java Graphics Basic Acceptance Test        Processor
pts/john-the-ripper        John The Ripper                            Processor
pts/jpegxl                 JPEG-XL libjxl                             Processor
pts/jpegxl-decode          JPEG-XL Decoding libjxl                    Processor
pts/juliagpu               JuliaGPU                                   System
```

Logged in to the openbenchmarking account in the first instance.



```
OpenBenchmarking.org User-Name: iemsaadabdullah

OpenBenchmarking.org Password:
    [8192] str_replace(): Passing null to parameter #2 ($replace) of type array|string is deprecated in phodevi:623

The Account Has Been Setup.

ubuntu@ip-172-31-79-179:~$
```

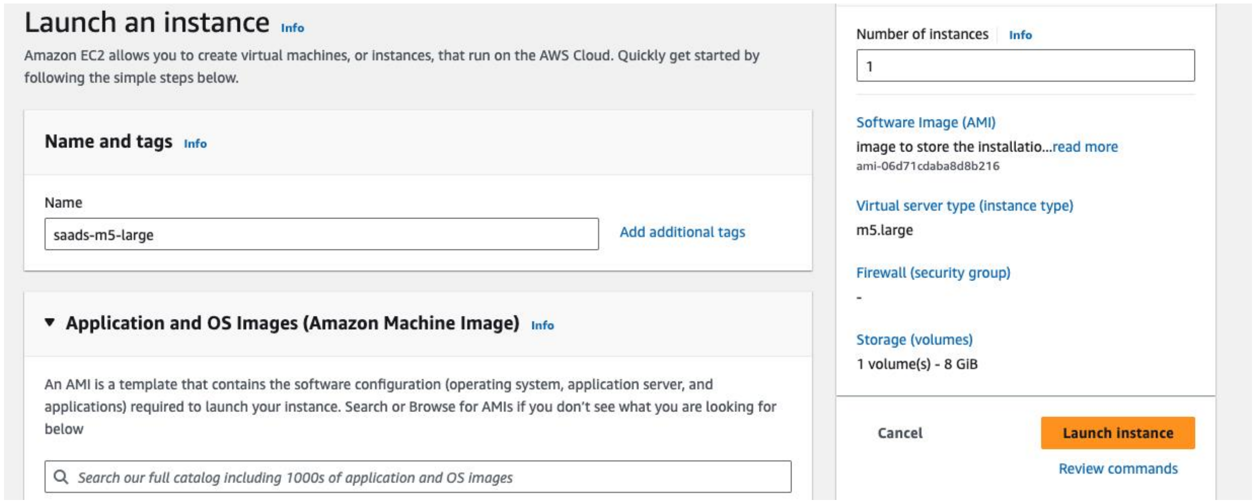
After performing all benchmarks I created my first AMI as following:



Amazon Machine Images (AMIs) (1/1) Info

Name	AMI name	AMI ID	Source	Owner	Visibility	Status
AMI FOR PHORONIX	phoronix-test-suite-img	ami-06d71cdaba8d8b216	749460268710/phoronix-test-suite-img	749460268710	Private	Available

Then I created an instance M5 from this AMI as follows:


Info
Add additional tags
Cancel
Launch instance
Review commands
Search our full catalog including 1000s of application and OS images

Launch an instance

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tags

Name: saads-m5-large

Application and OS Images (Amazon Machine Image)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Number of instances: 1

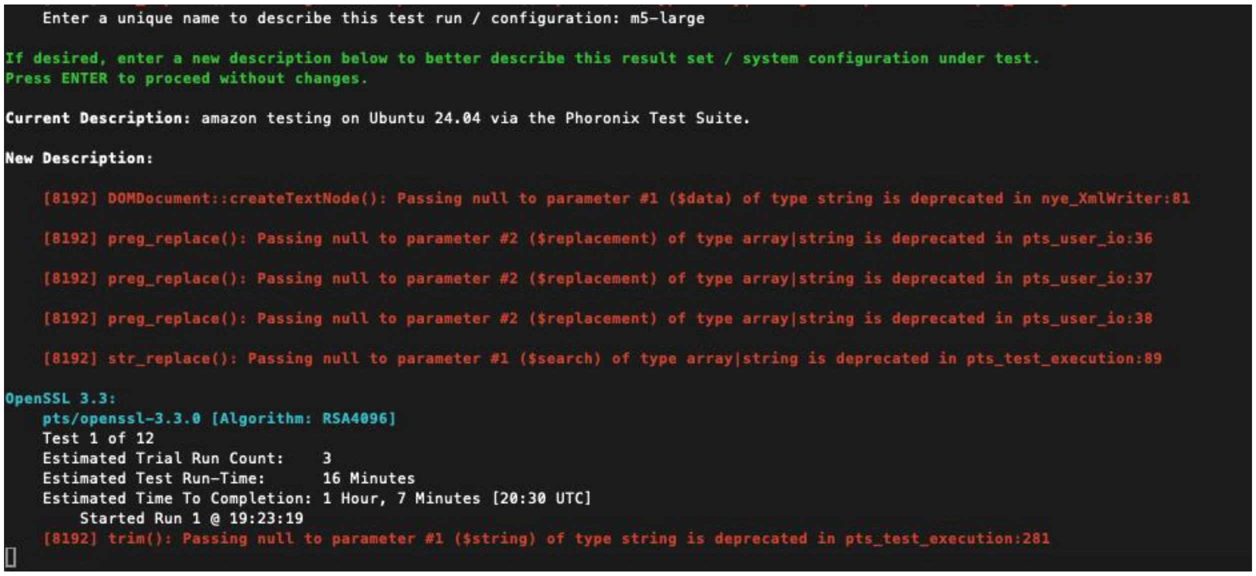
Software Image (AMI): image to store the installatio...read more
ami-06d71cdaba8d8b216

Virtual server type (instance type): m5.large

Firewall (security group): -

Storage (volumes): 1 volume(s) - 8 GiB

Then I ran benchmarks on this instance using the short link generated in openbenchmarking site:



```
    Enter a unique name to describe this test run / configuration: m5-large

If desired, enter a new description below to better describe this result set / system configuration under test.
Press ENTER to proceed without changes.

Current Description: amazon testing on Ubuntu 24.04 via the Phoronix Test Suite.

New Description:

    [8192] DOMDocument::createTextNode(): Passing null to parameter #1 ($data) of type string is deprecated in nye_XmlWriter:81

    [8192] preg_replace(): Passing null to parameter #2 ($replacement) of type array|string is deprecated in pts_user_io:36

    [8192] preg_replace(): Passing null to parameter #2 ($replacement) of type array|string is deprecated in pts_user_io:37

    [8192] preg_replace(): Passing null to parameter #2 ($replacement) of type array|string is deprecated in pts_user_io:38

    [8192] str_replace(): Passing null to parameter #1 ($search) of type array|string is deprecated in pts_test_execution:89

OpenSSL 3.3:
    pts/openssl-3.3.0 [Algorithm: RSA4096]
    Test 1 of 12
    Estimated Trial Run Count:    3
    Estimated Test Run-Time:      16 Minutes
    Estimated Time To Completion: 1 Hour, 7 Minutes [20:30 UTC]
        Started Run 1 @ 19:23:19
    [8192] trim(): Passing null to parameter #1 ($string) of type string is deprecated in pts_test_execution:281
```

Then I did the same for other instances and performed the benchmarking.

-- THE END --