**Analysis Report**

**on**

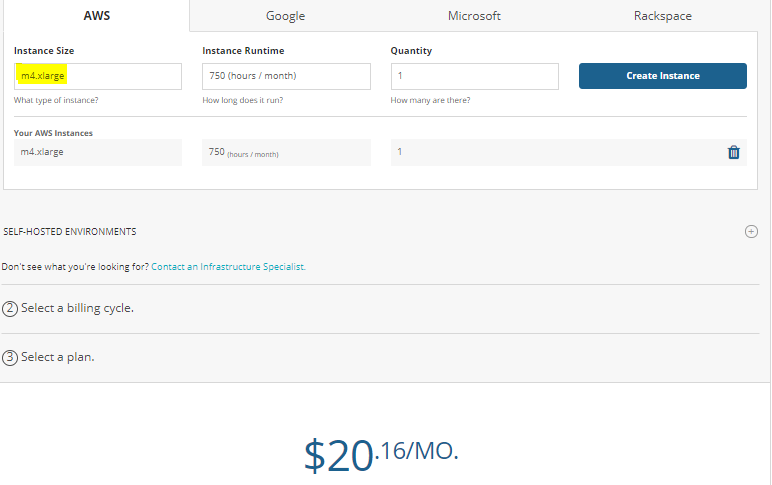
**Balluun Performance Tuning**

Basic application monitoring includes the following things.

* Monitoring the CPU, Memory, Disk I/O of your servers.
* Parsing your web server access logs to see how many requests you are getting and how long they take on average.
* Tracking and monitoring application error rates.
* Monitoring network traffic to identify slowdowns.
* Tracking key metrics from app dependencies like SQL, Redis, Elasticsearch, etc.

We have New Relic in place to monitor application performance monitoring. Where we could analyze end-user experience on the application, how long user requests are processing and how long it is taking to run the queries in DB.

Apart from application performance, we should also consider what kind of infrastructure we have and how resources are being utilized in the infrastructure. New Relic also can able to monitor your application infrastructure but it will cost more. Below is the simple calculator from new relic to estimate the cost for infrastructure monitoring for single EC2 Instance.



Below is the link for the calculator to monitor infrastructure using new relic.

<https://newrelic.com/products/infrastructure/pricing>

To monitor a single server we have to pay $20 per month. Likewise, we have 5 servers in Balluun production environment. So I have recommended choosing **ELK (opensource)** for Infrastructure monitoring.

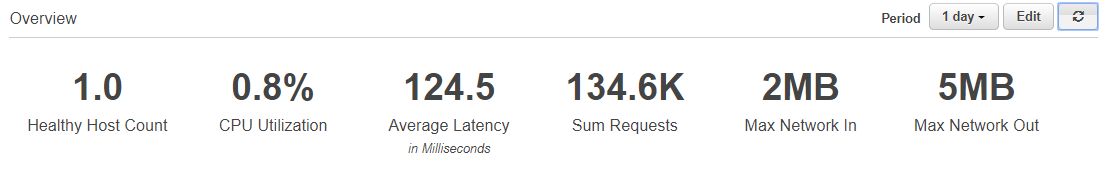
With the help of ELK, we have avoided spending **$100** per month for infrastructure monitoring.

**Servers Performance overview:**

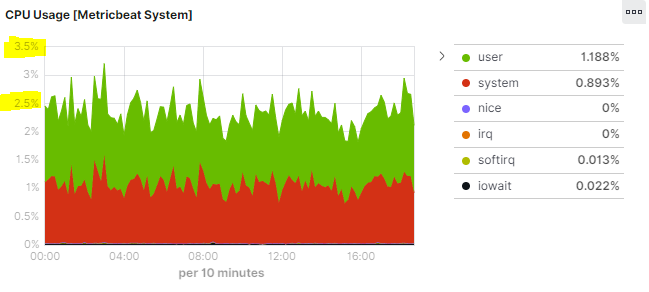
**1. Balluun UI performance:**

* Balluun web app is deployed into Elastic Beanstalk service.
* Server running with **m4.xlarge** machine type.
* Server performance is operating normally.

*Metric from Elastic beanstalk monitoring:*



*Metric from Kibana:*



Web server is not utilizing efficiently. We can downgrade this server type but David mentioned some time Balluun application has spike and resource utilization is high in some time.

**Recommendations:**

If we are looking for cost optimization we could downgrade machine size to **t3a.large** and enable autoscaling to handle the load whenever an application has huge traffic. This is a standard way to deploy any application.

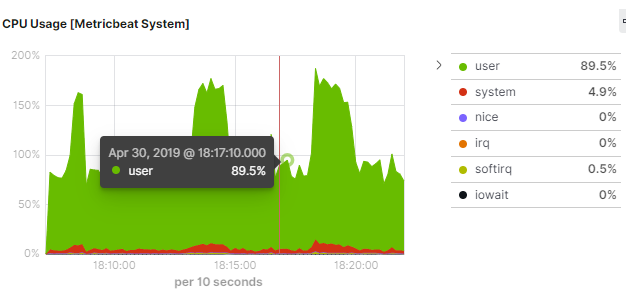
In the current configuration, we have already enabled load balancing. So we just need to increase machine count and design auto-scaling policy (CPU load /Memory utilization).

In this case, resource utilization will be fair and you will get less billing. But Still, if you have any concern in downgrading this, I would recommend updating **m4.xlarge** to **m5.xlarge.** In the latest general purpose family, **N/W performance** increased from **1 Gb/s** to **up to 10 Gb/s**. It will help us in reducing application latency.

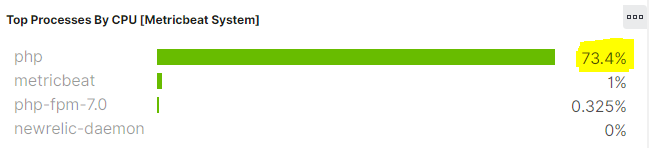
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Server** | **type** | **VCPU's** | **Mem (GiB)** | **Storage** | **Networking Performance (Gbps)** | **Cost per Hour** | **Affected Cost per month** |
| **Ballunn-UI** | m4.xlarge | 4 | 16 | EBS-only | High | $0.20 | $144 |
| **Recommended Types** | | | | | | | |
| Recommendation 1 | m5.xlarge | 4 | 16 | EBS-only | Up to 10 | $0.19 | $138.24 |

**2. Ballun API Performance:**

* Balluun API is deployed into the Ec2 machine.
* Server running with **r4.xlarge** machine type (4 vCPU’s and 30 GB RAM).
* Server performance not operating normally.
* Please check the below metric over the period of 3 days.

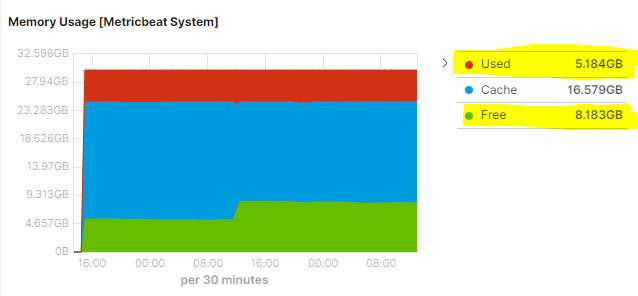


It is constant at 75%-85% and getting spikes more times every 5 minutes.



In the above snapshot **PHP** process, itself is taking more than 70%.

See RAM utilization status:



Here comparing with memory CPU is consuming more.

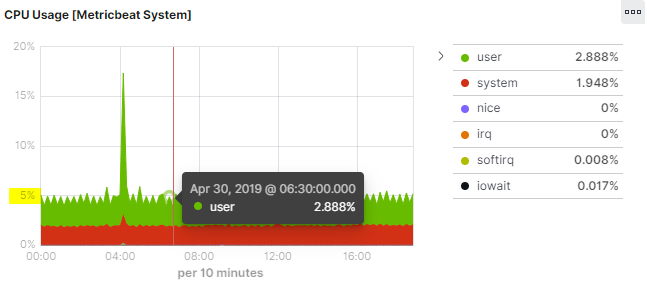
**Recommendation:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Server** | **type** | **VCPU's** | **Mem (GiB)** | **Storage** | **Networking Performance (Gbps)** | **Cost per Hour** | **Affected Cost per month** |
| **API server** | r4.xlarge | 4 | 30.5 | EBS-Only | Up to 10 | $0.27 | $191.52 |
| **Recommended Types** | | | | | | | |
| Recommendation 1 | c5n.xlarge | 4 | 10.5 | EBS-Only | Up to 25 | $0.22 | $155.52 |
| Recommendation 2 | c5.2xlarge | 8 | 16 | EBS-Only | Up to 10 | $0.34 | $244.80 |
| Recommendation 3 | c5n.2xlarge | 8 | 21 | EBS-Only | Up to 25 | $0.43 | $311.04 |

* It is recommended to chose to compute optimized type rather than having a memory-optimized type of machine.
* Change this server type to **c5n.xlarge** and test the performance. Still, if we see the same problem we have to increase vCPU’s capacity.
* In the above failure case, **C5.2xlarge** will be the best fit into this. But before that, I would recommend you to try **c5n.xlarge** in production or testing environment**.**

**3. Balluun Worker Performance:**

* Balluun Worker is deployed into the Ec2 machine.
* Server running with **m4.large** machine type (2 vCPU’s and 8 GB RAM).
* Server performance not operating normally.
* Please check the below metric over the period of 3 days.

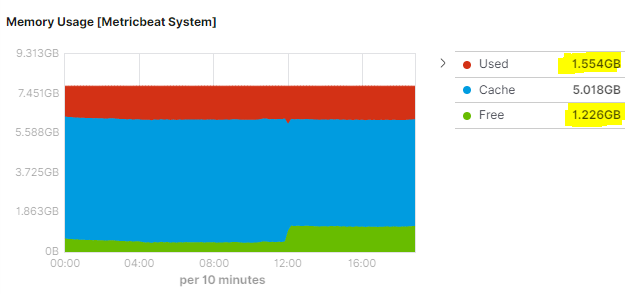


The CPU utilization is constant at 5%.

**Recommendation:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Server** | **type** | **VCPU's** | **Mem (GiB)** | **Storage** | **Networking Performance (Gbps)** | **Cost per Hour** | **Affected Cost per month** |
| **Worker** | m4.large | 2 | 8 | EBS-only | Moderate | $0.10 | $72 |
| **Recommended Types** | | | | | | | |
| Recommendation 1 | t3a.medium | 2 (20%) | 4 | EBS-only | Up to 5 | $0.04 | $27 |
| Recommendation 2 | t3a.large | 2 (30%) | 8 | EBS-only | Up to 5 | $0.07 | $50 |

Observe the below snapshot on memory utilization by worker server

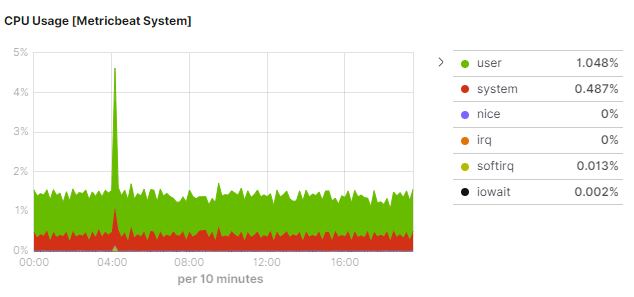


If we choose **t3a.medium** machine type we will get the following benefits

1. Network performance will be increased more.
2. CPU credits will be added and used when it needs to run at 100% utilization.
3. It will reduce the cost from $72 to $27.

**4. Balluun Rabbit-MQ & Redis Performance:**

* Balluun Rabbit-MQ & Redis is deployed into the Ec2 machine.
* Server running with **m4.xlarge** machine type (4 vCPU’s and 16 GB RAM).
* Server performance operating normally.
* Please check the below metric over the period of 3 days.



We can keep the same instance type as two services are running inside the same machine.

**Recommendation:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Server** | **type** | **VCPU's** | **Mem (GiB)** | **Storage** | **Networking Performance (Gbps)** | **Cost per Hour** | **Affected Cost per month** |
| **Rabbit-MQ** | m4.xlarge | 4 | 16 | EBS-only | High | $0.20 | $144 |
| **Recommended Types** | | | | | | | |
| Recommendation 1 | t3a.large | 2 (30%) | 8 | EBS-only | Up to 5 | $0.07 | $50 |
| Recommendation 2 | m5.xlarge | 4 | 16 | EBS-only | Up to 10 | $0.19 | $138.24 |

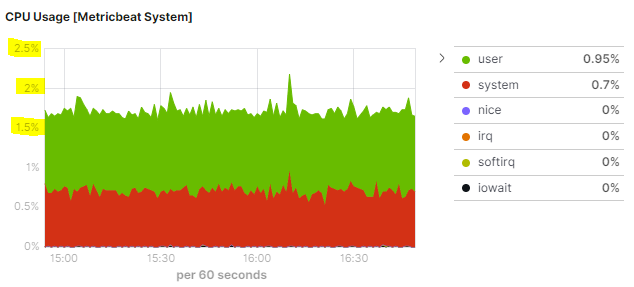
**Recommendations:**

In case if we look for cost optimization we can choose **t3a.large** machine but RAM size will be reduced more. We may face memory issues when we have spike in the applications as two services are running in the same server.

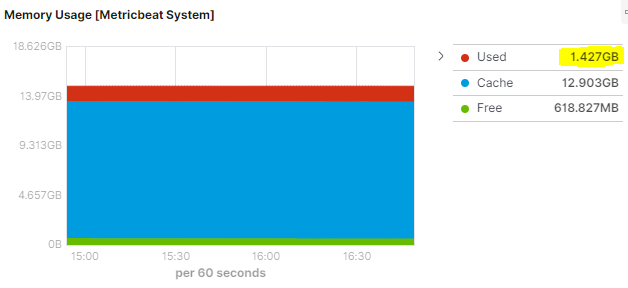
It is good to maintain the same capacity but update the **instance family**. In the latest family, **N/W performance** was more (Ref: Recommendation 2). It would help us in increasing the performance of Rabbit-MQ and Redis.

**5. Balluun API-Edge server performance:**

* Balluun API-Edge server is deployed into the Ec2 machine.
* Server running with **r4.large** machine type (2 vCPU’s and 15 GB RAM).
* Server performance operating normally.
* Please check the below metric over the period of 3 days.

****

CPU is not utilizing efficiently



Memory usage is 1.42 GB and most of the RAM is utilizing by the cache.

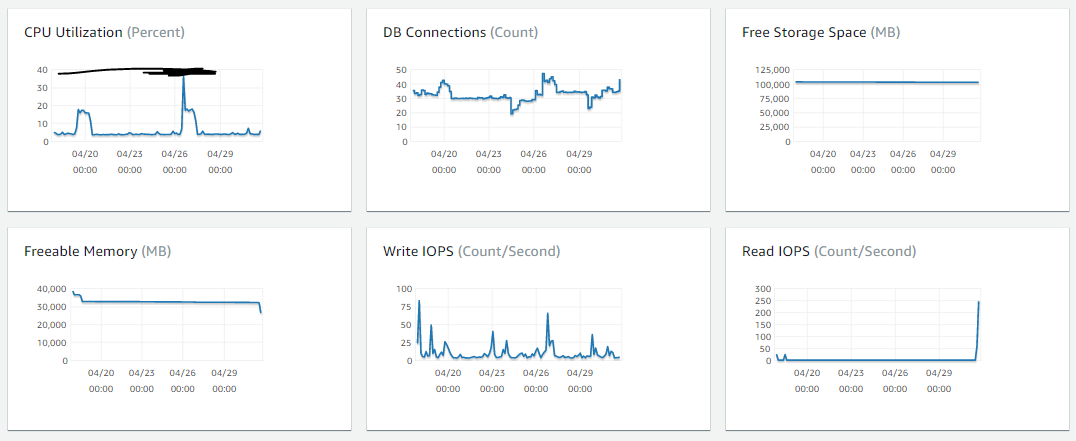
**Recommendation:**

We can downgrade this to any of the **t3 family** to reduce the cost and use it in a cost-effective way.

**Balluun Database Performance:**

* Balluun production database is configured with the **db.m4.10xlarge** class which has vCPU: 40, RAM: 160 GB.
* MySQL DB version 5.7
* Resource utilization was 30%-40%. But it is generating more billing.

Below are the metrics for the last 2 weeks in production.



**Recommendation:**

I have recommended using amazon aurora.

* Amazon Aurora is serverless compute capacity. We will pay for what we use, but it will support for MySQL 5.6 version only.
* Whenever there is no usage of your DB it will pause your compute capacity.
* You will be paying for storage only during the above case.

**Aurora DB Testing Status:**  We have created a cluster in Dev Environment and testing the functionality.We are trying to downgrade DB version to 5.6 and testing it with staging environment.

**Database Performance Tuning key points:**

* **IOPS** – The number of I/O operations completed each second. This metric is reported as the average IOPS for a given time interval. Amazon RDS reports read and write IOPS separately on 1-minute intervals. Total IOPS is the sum of the read and write IOPS. Typical values for IOPS range from zero to tens of thousands per second.
* **Latency** – The elapsed time between the submission of an I/O request and its completion. This metric is reported as the average latency for a given time interval. Amazon RDS reports read and write latency separately on 1-minute intervals in units of seconds. Typical values for latency are in the millisecond (ms). For example, Amazon RDS reports 2 ms as 0.002 seconds.
* **Throughput** – The number of bytes each second that are transferred to or from disk. This metric is reported as the average throughput for a given time interval. Amazon RDS reports read and write throughput separately on 1-minute intervals using units of megabytes per second (MB/s). Typical values for throughput range from zero to the I/O channel’s maximum bandwidth.
* **Queue Depth** – The number of I/O requests in the queue waiting to be serviced. These are I/O requests that have been submitted by the application but have not been sent to the device because the device is busy servicing other I/O requests. Time spent waiting in the queue is a component of latency and service time (not available as a metric). This metric is reported as the average queue depth for a given time interval. Amazon RDS reports queue depth in 1-minute intervals. Typical values for queue depth range from zero to several hundred.

**9 AWS Best practices to reduce spend in AWS Cloud**

1. **Delete unattached EBS Volumes:**

By continuously checking for unattached EBS volumes in your infrastructure, you can cut thousands of dollars from your monthly AWS bill.

1. **Delete Aged Snapshots:**

Organizations can help get EBS snapshots back under control by monitoring snapshot cost and usage per instance to make sure they do not spike out of control. Set a standard in your organization for how many snapshots should be retained per instance

1. **Delete disassociated Elastic IP address:**

From a best practice standpoint, monthly Elastic IP charges should be as close to zero as possible. If disassociated Elastic IPs are within the AWS accounts, they should either be reassociated to an instance or outright deleted in order to avoid the wasted cost.

1. **Terminate zombie Assets:**
   1. Zombie assets are infrastructure components that are running in your cloud environment but not be used for any purpose.
   2. Zombie assets come in many forms. For example, they could be EC2 once used for a particular purpose but are no longer in use and have not been turned off.
   3. Zombie assets can also come in the form of idle Elastic Load Balancers (ELB) that aren’t being used effectively or an idle Relational Database Service (RDS) instance. No matter the cause, AWS will charge for them as long as these assets are in a running state
2. **Upgrade Instances to the latest generation:**

Every few years, AWS releases the next generation of instances with improved price-per-compute performance and additional functionality like clustering, enhanced networking, and the ability to attach new types of EBS volumes. For example, upgrading a c1.xlarge to a c3.xlarge will cut costs by up to 60% while offering significantly faster processing.

1. **Right size EC2 instances and EBS Volumes:**
   1. Because it’s common for instances to be underutilized, you can reduce costs by assuring that all instances are the right size.
   2. If a volume is attached to an instance and barely has any read/writes on that volume, the instance is either inactive or the volume is unnecessary. These are good candidates to flag for rightsizing evaluation.”
2. **Stop and Start Instances on a schedule:**

Assume Instances that are running 24/7, Amazon will bill for 672 to 744 hours per instance, depending on the month. If an instance is turned off between 5 pm and 9 am on weekdays and stopped weekends and holidays, then total billable hours per month would range from 152 to 184 hours per instance, saving you 488 to 592 instance hours per month

1. **Buy Reserved instances on EC2, RDS and Automate optimization**

RIs can save you up to 75% compared to on-demand pricing, so they’re a no-brainer for any company with sustained EC2 or RDS usage.

1. **Move Object data to lower cost tiers:**

ABest practice is that any objects residing in S3 that are older than 30 days should be converted to S3 Infrequent Access. While standard storage class pricing is tiered based on the amount of content within the bucket, with a minimum price of $0.0275 per GB per month, Infrequent Access storage remains consistent at $0.0125 per GB per month.