AWS CloudMetrics

Below are suggested health related AWS metrics (health-check related measurements)

|  |  |  |
| --- | --- | --- |
| Service | Metrics | Notes |
| S3 | * Number of objects In bucket * Total Request Latency ($) * Get Requests ($) * Get Requests ($) * 500 errors ($) | ($) AWS charge for duration that metrics are enabled |
| API Gateway | * 5XXError – 500 errors * Latency * Count (# of Requests) * Cached Hit Count \* * Cached Miss Count \* | \*applicable if request caching is enabled (caching not enabled by default) |
| Lambda | * Number of Invocations * Duration * Errors * Throttles \* | \*applicable in high volume scenarios only which may have more than 1000 concurrent requests. |
| Beanstalk | * Application Latency P95 ($) * Application Requests5XX($) * Application Requests Total ($) * InstancesOk ($) * InstancesDegraded ($) * CPU User ($) * CPU System ($) * CPU IOWait ($) | ($) AWS charge for duration that metrics are enabled |
| EC2 | * CPU Utilization * CPU Credit Usage \* * CPU Credit Balance \* | \*applicable to T family only |
| EBS | * VolumeQueueLength * VolumeReadOps * VolumeWriteOps * BurstBalance \* | \*applicable to non-provisioned IOPs volumes only. |
| Aurora (MySQL) | * CPU Utilization * Buffer Cache Hit Ratio * DB Connections * Write IOPs * Read IOPs * Swap Usage * Select Latency * DML Latency * Read IOPs * Write IOPs * DiskQueueDepth | Enabling Enhanced Monitoring provides a deep dive health view of CPU, Memory, and Disk. |
| DynamoDB | * Request Latency * Consumed Read Capacity Units * Consumed Write Capacity Units * Throttled Read Requests\* * Throttled Write Requests\* | \* Important to track if performance degrades as a result of exceeding provisioned capacity |

Performance/Costing Implications

Depending on the use case, these are some things you may want to factor in as how it may relate to the performance profile and impact to cost

* S3: There are different types of storage. For most cases, standard storage is adequate (particularly if using for event based processing).

Questions to answer:

* + How often files are going to be accessed?
  + What is the lifetime of a file in the bucket?
  + How many files can in the bucket at a given time
  + Is file versioning a requirement?
* Lambda:  There is a start-up cost for lambda Java functions when they are idle for a period of time. First-hand experience I have observed a cost of 10-20 seconds to initialize before consuming the first event after waking up. Afterwards, lambda response time is dependent on the custom handler logic.
  + The memory tunable setting impacts CPU power as well for function, having an effect on response time.
* Beanstalk: Four factors to consider relating to performance and minimize footprint cost:
  + What type workload (processing activities) will be done (CPU heavy, Memory Heavy, etc.)? This helps answer what EC2 instance type to use. For the majority of web applications, which require balanced amounts of CPU and memory, either a T or M family would be applicable for general purpose needs/
  + After selecting an instance type, the next question is what instance type size is needed ? For example, does the application need a T2.medium, T2.large, or T2.xlarge? Suggestion is start with the smallest size that would be appropriate based on early estimates and then tune up based on performance test results.
  + Lastly, How many EC2 instances of that type and size are required? Start small, preferably with 1 and determine capacity supported to project additional instances needed. Then validate with results
  + How many instances do I need for BAU processing vs high demand processing? This is where auto scaling comes in. For predictable traffic surges (end of week, month), time based scaling is a good option to enable. For unpredictable times when traffic may increase, scaling trigger based on a resource threshold is more adequate. Metric and threshold to use are suggested to be based on performance test results to assess which resource gets strained as load is increased.
* API Gateway: May want to consider enabling request caching if you have use cases for frequent/high usage requests which response may not change in a given time window (e.g. different clients making same type of request). This can help improve request latency for those requests. How long you want to cache the response is configurable.
* DynamoDB: Two important factors are data partitioning and determining how many read and write capacity units you need to achieve desired query throughput. This will help set optimal settings for auto scaling dynamo capacity. The determining optimal capacity is best achieved with running some experimental benchmark tests. One caveat with auto scaling is that partitions created, as a result adding provisioned read and write capacity, won’t be deleted if capacity is reduced (so don’t want to set provisioned capacity too high).
  + Amazon offers Dynamo Accelarator (DAX) as an in-memory cache solution designed for DynamoDB workloads. Something to consider if it meets your use case.