API Capacity Planning Guidelines

MuleSoft C4E Standards & Best Practices
v0.9

Status	Draft
Date	

Table of Contents

Revision History 3
Objective 3
Audience 3
Worker Capacity 3
Capacity Planning for on-boarding existing On Prem Mule APIs to Cloudhub 5
Application vCore Sizing 6
When to Upgrade from vCore 01. /0.2 to V1 7
Grouping/Consolidating of Microservices 8
Other Considerations 8
Zero Downtime Deployment Model 8

Revision History

Date	Version	Description	Author
08/01/2023	1.0.0		Sai Manivannan

Objective

The following guide is intended to come up with Guidelines for Capacity Planning that primarily involves estimation of vCores needed

Audience

- C4E Architect / Architects

Worker Capacity

Capacity

Each worker has a specific amount of capacity to process data. Select the size of your workers when configuring an application.

Isolation

Each worker runs in a separate container from every other application.

Manageability

Each worker is deployed and monitored independently.

Locality

Each worker runs in a specific worker cloud, such as the US, EU, or Asia-Pacific.

Reference: https://docs.mulesoft.com/runtime-manager/cloudhub-architecture

Worker Size	Heap Memory	Storage
0.1 vCores	500 MB	8 GB
0.2 vCores	1 GB	8 GB
1 vCore	1.5 GB	12 GB
2 vCores	3.5 GB	40 GB

4 vCores	7.5 GB	88 GB
8 vCores	15 GB	168 GB
16 vCores	32 GB	328 GB

PRODUCTION:
Software or Cloud Offerings used in a live production environment, being actively used to process data or provide information to end-users, but not being simultaneously used for development or pre-production purposes.

PRE-PRODUCTION:
Software or Cloud Offerings used in any non-Production environment for development, sandbox, quality assurance, testing, or staging purposes.

CORE:
This may refer to either a physical CPU core, or a "virtual core" (also referred to as a vCPU), which is a logical partition of a physical CPU core. When deploying on Amazon EC2, each EC2 vCPU shall be counted as 1 core, except for t2 instances. For t2 instances, the number of cores in use by an EC2 instance shall be calculated as the total number of CPU minutes available in a 24 hour period, divided by the total number of minutes in a 24 hour period.

VCORE:

A unit of compute capacity for processing on CloudHub, which is equal to one virtual core. Up to ten Mule Applications can be deployed for every VCore purchased.

FRACTIONAL VCORE notation

The fractional vCore notation uses decimal points to represent a portion of a total vCore. Therefore: 0.1 vCore is the same as one-tenth of a total vCore and 0.2v Cores is the same as one-fifth of a vCore. If 10 CloudHub Applications were deployed with 0.1 vCores, this would consume 1 full vCore.

Design vCores will be consumed from the subscriptions Design vCores entitlements. Applications will be deployed in Environments created of type "Design". Design vCores differ from Pre-Production (often referred to as Sandbox) vCores, in that Design vCores cannot be utilized for applications created in Anypoint Studio. This is why it is not possible to manually deploy Applications or API's to environments which are of type "Design". It is not possible, if you have consumed all your Sandbox vCores, to use the design vCores instead.

CAPACITY PLANNING FOR ON-BOARDING EXISTING ON PREM MULE APIS TO CLOUDHUB

For a given existing Mule API ,identify the right sizing by performing Load Tests.

- Initiate the deployment of your applications in a lower-tier environment, provisioning them with vCore Size that you deem reasonable (say V0.2) based upon sizing considerations.
- Next, simulate the peak load expected in the production environment within your lower-tier setup. Tools such as JMeter can be used to generate HTTP requests, or you can use simpler alternatives like the 'hey' load generator: GitHub - rakyll/hey: HTTP load generator, ApacheBench (ab) replacement
- Monitor CPU and memory usage, as well as failed requests, in real-time and post-testing using Anypoint Monitoring.
- If the application demonstrates stable performance with no errors or noticeable lag, you may experiment with reducing the vCore size.
- However, if you observe increased response times or unfulfilled requests (confirming that backend APIs are not causing the bottleneck via Anypoint Monitoring), consider increasing the vCore size if the heap size is maxed out for a considerable period of time. If resource scarcity persists, try increasing the number of Workers which should help with the throughput.

Continue repeating steps 3-5 until you establish the optimal required memory, CPU limit, and CPU request that enables the application to exceed performance expectations during a load test.

The ultimate objective is to identify the optimal resource allocation that allows a Mule API to withstand peak load without any performance degradation.

Before executing performance testing:

- Confirm that your Mule app and its functions work as expected because a wrong flow can give false-positive data.
- Establish performance test criteria by asking yourself the following questions:
    - What are the expected average and peak workloads?
    - What specific need does your use case address?:
        - Throughput, when handling a large volume of transactions is a high priority.
        - Response time or latency, if spikes in activity negatively affect user experience.
        - Concurrency, if it is necessary to support a large number of users connecting at the same time.
        - Managing large messages, when the application is transferring, caching, storing, or processing a payload bigger than 1 MB.

- What is the minimum acceptable throughput?
- What is the maximum acceptable response time?

Application vCore Sizing

When determining the appropriate sizing for a deployment on CloudHub, one must consider principles similar to those in any other system. However, there are unique aspects to CloudHub that require attention. Specifically, understanding CloudHub's fixed hardware settings and ensuring they closely match the testing environment is crucial.

Even with an attempt to mirror settings like heap, memory, and core count between the testing environment and CloudHub, it's essential to note that a direct one-to-one comparison may not always be possible. For instance, while an organization might allocate 1GB of memory for their application with 500MB dedicated to heap size, replicating a 0.1 vCore environment on CloudHub might present challenges. This is primarily because CloudHub's non-heap 500MB is also tasked with supporting its OS, monitoring routines, and other essential software functions. Thus, if the 500MB is solely reserved for runtime and the application in the testing phase, the results on CloudHub might differ from those anticipated.

Moreover, it's essential to remember that the CPU on CloudHub workers is allocated a finite number of CPU credits. Consequently, an application that runs seamlessly on a local machine with 4 CPU cores might exhibit performance lags when transitioned to CloudHub over extended periods.

When unsure of the workload their application can sustain or if the observed performance isn't on par with expectations, it might be beneficial to contemplate an increase in worker size. If upscaling resolves the issue, it may indicate initial sizing challenges. However, if the problem persists, although with reduced frequency, it could signal the presence of a performance-impacting bug or perhaps an unresolved sizing concern relative to the workload the application is expected to handle.

A performance impacting bug refers to a situation where the performance of an application deteriorates, even when it's undergone appropriate sizing procedures and is operating under standard conditions - such as normal load and healthy hardware. However, it's crucial to differentiate between such bugs and common system resource exhaustion scenarios.

The depletion of system resources, such as CPU time, memory, or disk space, is usually not indicative of a performance impacting bug. The same applies to instances of an out-of-memory error, an unresponsive application, a runtime restart, or a 'too many open files' error. These issues often signal sizing problems rather than bugs that directly impact performance.

A performance impacting bug typically stems from a fault in the MuleSoft code, leading to an exhaustion of system resources. What distinguishes it is that these issues persist regardless of the amount of available hardware or additional resources provided. In essence, a performance impacting bug is a scenario where, despite having ample resources at disposal, an error within

the MuleSoft code prevents the efficient utilization of these resources, causing continual exhaustion and subsequent performance deterioration.

When to Upgrade from vCore 01. /0.2 to V1

0.1 and 0.2 vCore workers. These instances leverage the CPU burst capabilities, a feature provided by the underlying infrastructure that enables a worker to momentarily surge to higher CPU capacities. However, this burst ability is finite, operating on a system of earning and consuming CPU credits. A worker accrues these credits during periods of idleness or when it uses less than the base CPU level, which may be 10% or 20%.

However, when CPU usage escalates beyond the base level, these credits are used up. Once all credits have been exhausted, the worker is confined to the base CPU level, leading to situations where the worker may not surpass 10% or 20% CPU usage.

Therefore, for applications that need occasional bursts of high CPU usage followed by periods of low or idle time, a 0.1 or 0.2 vCore worker may be sufficient. However, for applications requiring sustained high CPU usage for more than an hour continuously, it is recommended to opt for a non-fractional vCore worker. In such scenarios, a 1vCore worker might be suitable, but only if the application does not need to operate at 100% CPU for extended periods, since this configuration also possesses burst capabilities at that consumption level. It should be noted that 2 vCores and above do not offer burst capability; instead, the full CPU capacity remains accessible indefinitely.

Grouping/Consolidating of Microservices

While Grouping functionally and categorically related Services into APIs to minimize vCore cost and ensure scalability is important , it's crucial to strike the right balance. It is essential to consider the functional requirements and lifecycle of the resources involved when doing so.

Also, if the Microservice constitutes a mission critical functionality, one would be better served to have this Microservice as a “non Grouped” API. For these mission critical APIs, in order to ensure Zero down time, another worker may need to be available as HA. Planning for such situations would be prudent.

Other Considerations

Zero Downtime Deployment Model

The goal is to be able to quickly make changes to the environment without impacting the SLAs; including upgrading infrastructure and the applications running on the infrastructure. Typically zero downtime deployments leverage a side-by-side deployment, where the old and new coexist for a short period of time. This is in contrast to an in-place deployment where the service may experience reduced capacity to complete downtime.

Zero Downtime Deployment model helps achieve Continuous operations - those characteristics of a data-processing system that reduce or eliminate the need for planned downtime, such as scheduled maintenance. One element of 24-hour-a-day, seven-day-a-week operation