# **Cluster Capacity Issues**

- Introduction
- Diagnosis
- Lower Lane Environment testing
- Conclussion:

### Introduction

The RHACM observability dashboard is either incorrect or can be misleading with respect to CPU request as a percentage of allocatable CPU within the cluster. This is a key metric for cluster health and capacity management, so it is essential that this is accurate.

A locally customised prometheus expression appears to be more accurate and can be further developed and integrated into our cluster fleet management.

We will raise a case with Red Hat for RCA on the dashboard metrics & suggestions on an ideal metric for alerting and capacity management of our fleet of clusters.

## Diagnosis

Ilustration from a production cluster (useast16), using RHACM grafana dashboard



This shows near linear growth of CPU requests due to "completed" jobs and shows the current value to be well over 100% (112%); which is patently inaccurate as the kubernetes scheduler will not overallocate cpu\_requests.

The local openshift grafana dashboard is showing a current snapshot value of 69.3% which also contradicts the RHACM observability dashboard.



The locally modified Prometheus expression appears more accurate , but also differs from the local openshift grafana dashboard:



Using the PROMQL below shows a cpu request utilisation of 76% of available & allocatable cpu resource as compared to 69.3% from openshift grafana & 112% from RHACM grafana

### Modified prometheus expression for accurate utilisation of CPU requests as a percentage of allocatable resource

```
sum((kube_pod_container_resource_requests{resource="cpu"} * on (pod,namespace) group_left (phase)
kube_pod_status_phase{phase="Running"}) * on (node) group_left (role) kube_node_role{role="app"})/
sum(kube_node_status_allocatable{resource="cpu"} * on (node) group_left(role) kube_node_role{role="app"})
```

Notes:

. 0

The expression attempts to be more accurate by:

- Aggregating cpu requests, kube\_pod\_container\_resource\_requests{resource="cpu"}, for pods in "Running" state only & for pods running on "APP" nodes.
- The cpu allocatable resource is also modified by only inclusing resources from "app" worker nodes.

While this is a custom prometheus expression for our "app" worker nodes, we will ask Red Hat to see if this can be improved?

### Lower Lane Environment testing

A more extreme example test in a lower lane environment where we used a cronjob to continuously run with a high CPU request; the accumulated completed jobs demonstrate that the cpu requests that accumulate are not actually impacting the cluster.

Sample cronjob details:

```
sample cronjob
```

```
apiVersion: batch/v1
kind: CronJob
metadata:
 creationTimestamp: "2023-03-06T17:17:10Z"
 generation: 5
 name: example1
 namespace: zkys6ky-gpu-namespace
  resourceVersion: "791697289"
  uid: 93c5a6bb-40d6-4a47-be21-45bd99822a1c
spec:
  concurrencyPolicy: Forbid
  failedJobsHistoryLimit: 9999999
  jobTemplate:
   metadata:
      creationTimestamp: null
    spec:
      template:
       metadata:
         creationTimestamp: null
       spec:
         containers:
          - args:
           - /bin/sh
            - -C
           - date; echo Hello from the Kubernetes cluster
            image: registry-eng.sdi.corp.bankofamerica.com/bac/ubi8-minimal:latest
            imagePullPolicy: Always
            name: hello
            resources:
             requests:
               cpu: "20"
               memory: "100"
            terminationMessagePath: /dev/termination-log
            terminationMessagePolicy: File
          dnsPolicy: ClusterFirst
          restartPolicy: Never
          schedulerName: default-scheduler
          securityContext: {}
          terminationGracePeriodSeconds: 30
  schedule: '* * * * *'
  successfulJobsHistoryLimit: 9999999
  suspend: false
```

CPU request commitment after 24 hours, shown by the overview RHACM observability dashboard looks normal at 49.74% for "local-cluster":

Ø	원 General / ACM - Clusters Overview ☆ 👒						🗤 🛱 🚳 🕐 Last 3 h	iours × Q C 5m × ₽	G	
	✓Control Plane Health								e	
Q		Тс	op 50 Max Latency API Se	ver				etcd		e
88										e
0	local-cluster	49.8 ms	100%			local-cluster	Yes		575.14 MiB	e
G	useast19	49.6 ms	100%			useast19	Yes		286.19 MiB	e
¢										e
~	~ Optimization									
	i	I Top 50 CPU Overestimation Clusters				i Top 50 Memory Overestimation Clusters				
					Utilized					
	local-cluster	43.29%	49.74%		6.45%	local-cluster	4.34%	14.35%	10.00%	
	useast19	16.66%	19.26%		200	useast19	1.73%	6.78%	5.05%	
	<ul> <li>Capacity / Utilization</li> </ul>									
			Top 50 CPU Utilized Cluste	rs			Top 5 Utilized Clu	usters (% CPU usage)		
						100%				
	local-cluster	528	524	49.74%	6.45%					
	useast19	528	524	19.26%	2.60%	50%				
?										

However, if we drill down, into local-cluster, we see incorrect values for Overestimation & CPU Requests Commitment of 1956% :

Ø	器 General / ACM - Resource Optimiza	ild* 🛱									
	Cluster local-cluster ~										
Q	~ CPU										
88	i Overestimation		CPU U	sage							
© 4	1949%		m		manna						
	CPU Requests Commitment										
	1956%		550 1600 1610 1620 1630	16:40 16:50 17:00 17:10 1	720 1730 1740 1750 1600						
	CPU Utilisation	- ansible-automation-platform - aqua-73300 - auth-proxy - crunchy-demo - demo - dynatrace - hive - home-zk	– consul – cp-0736920 – cp-2846383 – cp-4315305 – dm16v – ibm-common-services – kiran-tests – mamin-po	cp-4606059 — cp-5808004 — cp-8110997 — cp-admin -	– cp-build – cp-mrtest – cp-redis-steve – cp-sandbox ter-management – open-cluster-management-addon-observability						
	6.76%	% espectator management agent open-cluster management agent addom — open-cluster management due open-cluster management open-cluster ma									
			CPU Quota								
	ansible-automation-platform	0.21%	0.00								
	<u>aqua-73300</u>	0.44%	0.03	6.50							
	<u>cp-0736920</u>	0.16%	0.05	30.25							
	<u>cp-2846383</u>	0.43%	0.09	21.00							
	<u>cp-4606059</u>	0.43%	0.00	1.00							
	<u>cp-5808004</u>	3.60%	0.97	27.05							
	<u>cp-8110997</u>	6.64%	0.90	13.55							
Ū.											

If we "explore" the metric "CPU requests Commitment" and display a range we see the effect of our test ranging from approximately 50% to 1950%

) Ø	Explore () Observatorium v () Split «C) Last 7 days v Q	🕆 Clear all 🔇 Run query 🗸 🌐	Conv						
	dedicsbrowser) sum(kube_pod_container_resource_requests:sum(cluster="local-cluster", resource="cpu"}) / sum(kube_node_status_allocatable(cluster="local-cluster", resource="cpu"})	0.28 @ -	Adetu						
	+ Add query 🛇 Query history 🔘 Inspector		Wood						
	Graph	•	Conv						
2									

# Conclussion:

I think this clearly demonstrates that thre is a bug in the RHACM observability dashboards.

We also want to confirm the best metrics to monitor cpu requests as a real percentage of available space.

The metric below is from the "kubernetes/Compute Resources/Cluster" dashboard shows 49.84%, which matches the RHACM overview dashboard:



While the custom query, shows a more conservative value of 59%

#### custom promql

sum((kube\_pod\_container\_resource\_requests{resource="cpu"} \* on (pod,namespace) group\_left (phase)
kube\_pod\_status\_phase{phase="Running"}) \* on (node) group\_left (role) kube\_node\_role{role="app"})/
sum(kube\_node\_status\_allocatable{resource="cpu"} \* on (node) group\_left(role) kube\_node\_role{role="app"})

#### Results:

