Kubernetes Resource Management

By Nanthini Muniapan

Speaker Bio : Nanthini Muniapan

Currently working as Lead DevOps Engineer leading a team of 3 engineers.

13 years of experience in various industries e.g supply chain management, telco industry, media & entertainment industry. Started as a Software Developer and progressed into DevOps in 2018.

Alumni of USM Bachelor of Computer Science (Hons) 2010.

MBA graduate from Unitar International University (2022).

https://www.linkedin.com/in/nanthini-muniapan

Connect with me

slido

What is the level of your experience in Kubernetes

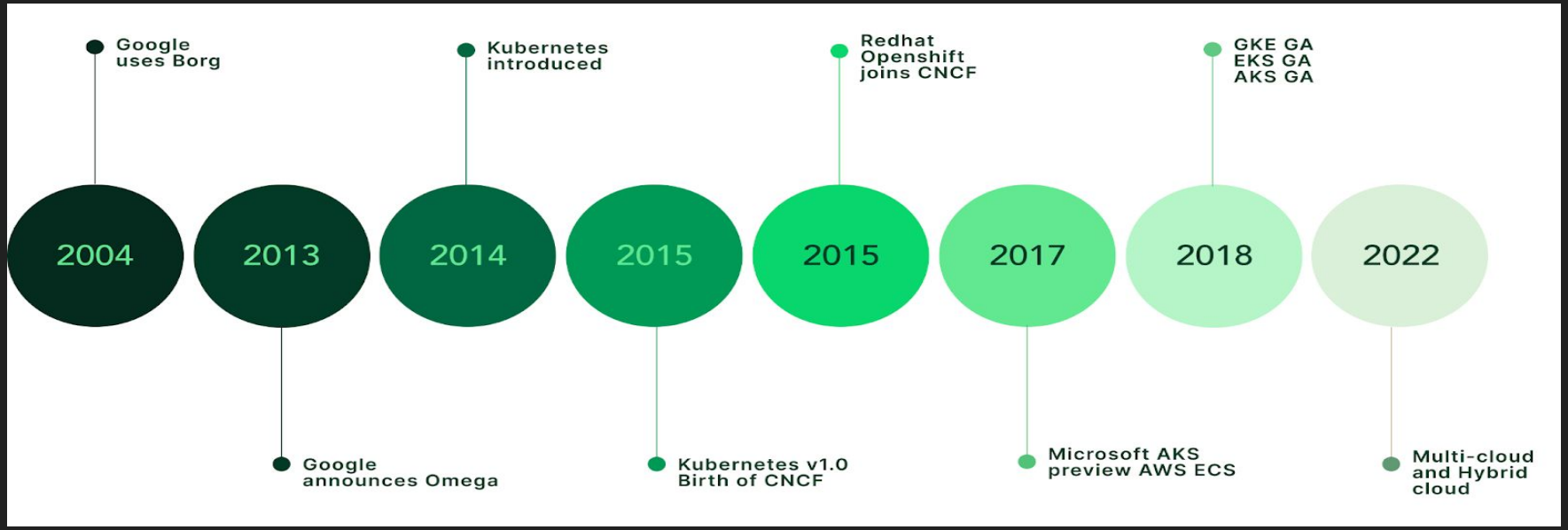
ⓘ Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

slido

Those who have worked with Kubernetes, please share your experience e.g I am hobbyist

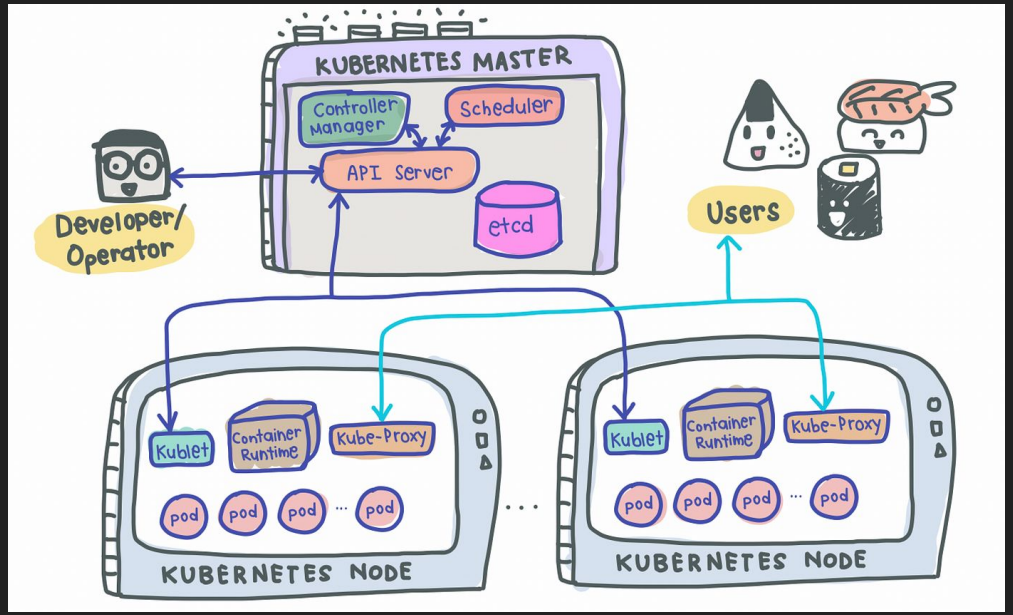
Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

Kubernetes Introduction



https://www.kubecost.com/kubernetes-multi-cloud/kubernetes-distributions/

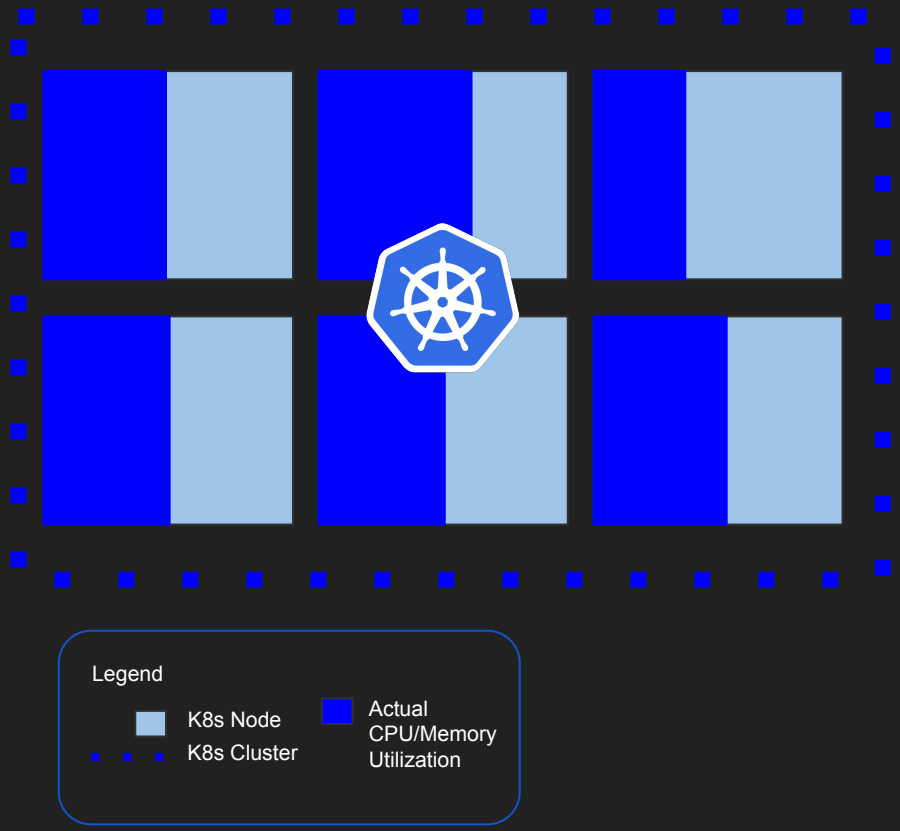
Kubernetes Introduction



https://towardsdatascience.com/a-beginner-friendly-introduction-to-kubernetes-540b5d63b3d7

1. *Load Balancing*
2. *Automatic Rollout*
3. *Automatic bin packing*
   We can control the resource on the containers like CPU or memory is allotted to the pods and containers.
4. *Self-Healing*
5. *Secret and Configuration Management*

Problem statement



The mystery of “underutilized” k8s nodes and the error “pod unschedulable due to insufficient CPU”

- Max pod per node has not exceeded
- Node utilization (EC2 or Virtual machine monitoring) is underutilized

slido

What does the number '8' refers to in k8s?

Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

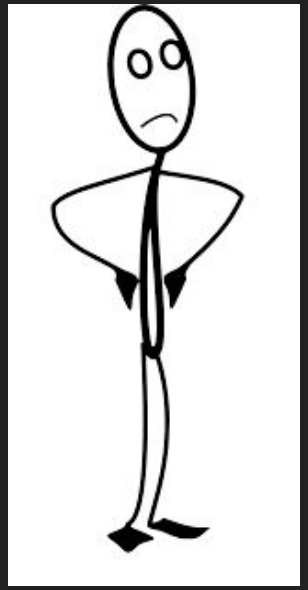
Scope of the talk

Hence inspiration behind a deep dive into POD CPU/Memory limit and request.

Disclaimer:
The talk is inspired by my experience with AWS EKS hence some of the tools used are tested against EKS and not other platforms.

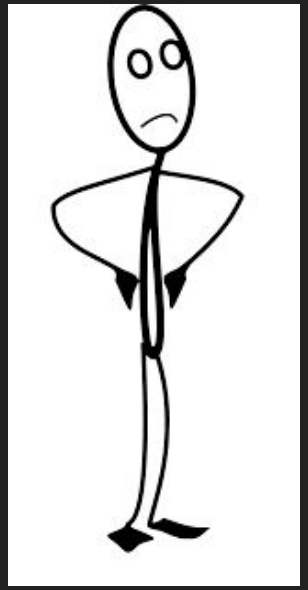
Request vs Limit

Request - Analogy



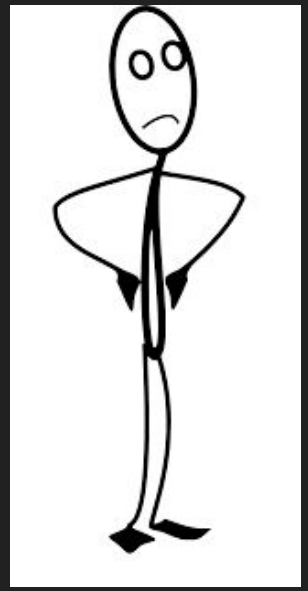
This is you…

Request - Analogy



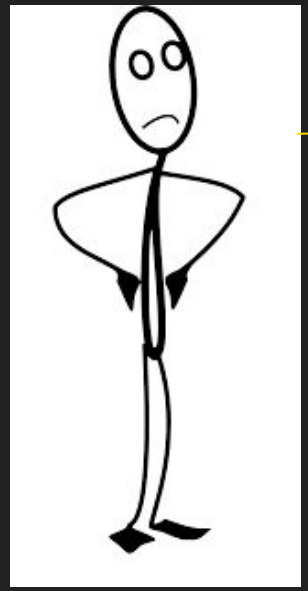
… feeling very hungry

Request - Analogy



You (Very Hungry)

Request - Analogy

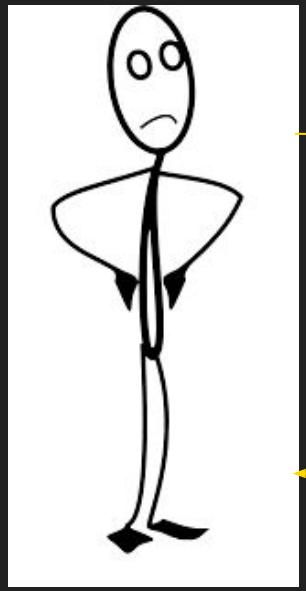


You (Very Hungry)

Give me 1 nasi lemak bungkus please!

Canteen Operator

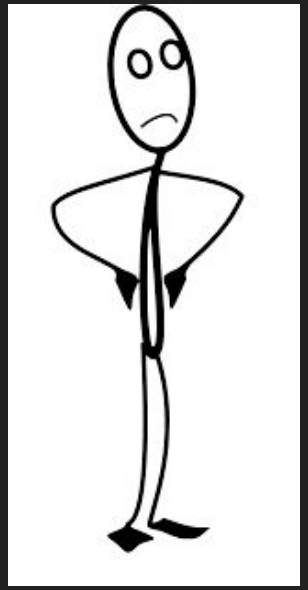
Request - Analogy



You (Very Hungry)

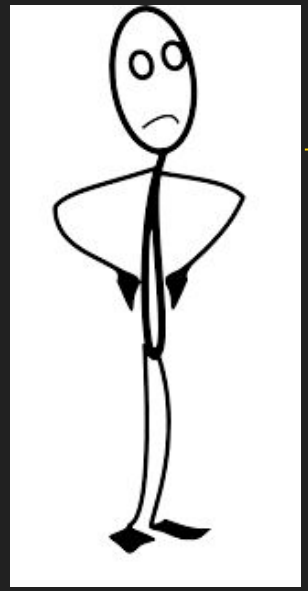
Canteen Operator

Request - Analogy



You.. are still hungry!

Request - Analogy



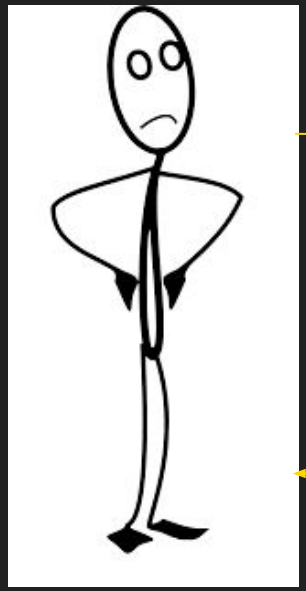
You (Very Hungry)

Give me 1 **MORE** nasi lemak bungkus please!

Canteen Operator

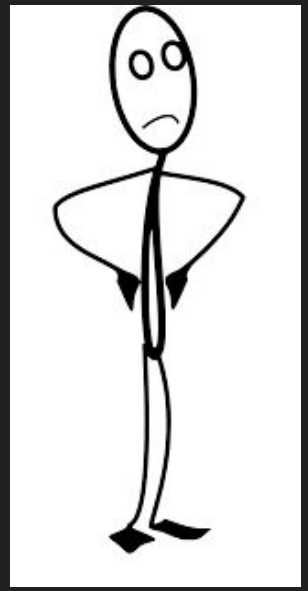
Request - Analogy

Canteen Operator



You (Very Hungry)

Request - Translated in K8s context



You == Pod

Nasi lemak ==
CPU/MEM Request

Canteen == K8s Control

Request

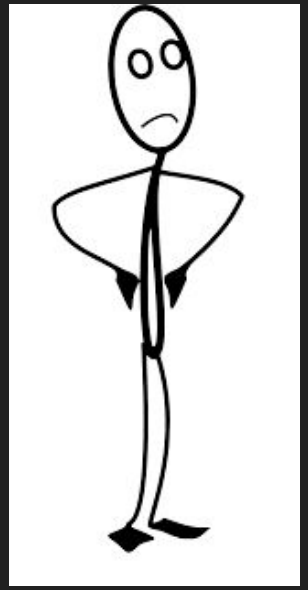
Requests is the minimum guaranteed amount of a resource that is reserved for a POD.

```
0 <= request <= Node Allocatable
```

Limit - Analogy (Contd)

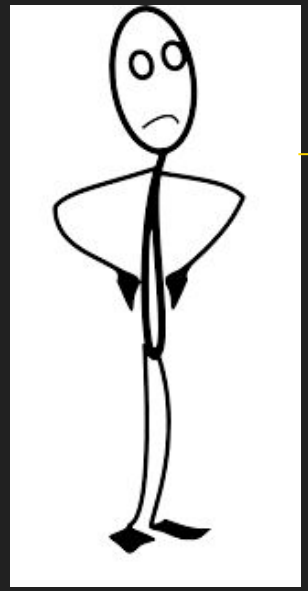
The Canteen Operator only has 100 nasi lemaks to sell each day (limited resource). Hence, cannot afford to sell more than 2 (limit) nasi lemak per student.

Limit - Analogy



You (Still Very Hungry)

Limit - Analogy

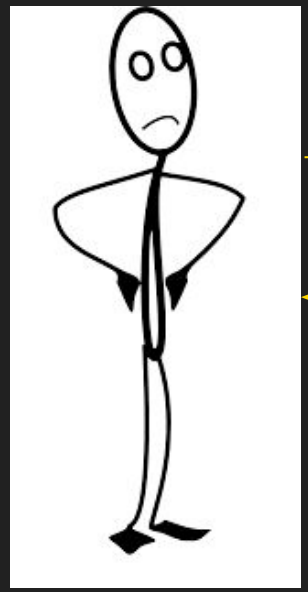


You (Very Hungry)

Give me 1 **MORE** nasi lemak bungkus please!

Canteen Operator

Limit - Analogy



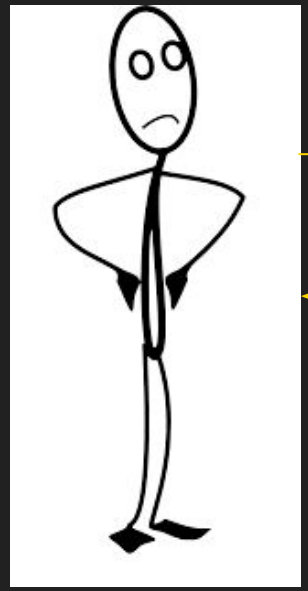
You (Still very hungry)

Give me 1 MORE nasi lemak bungkus please!

Sorry no more nasi lemak for you!

Canteen Operator

Limit - Translated in K8s context



You == POD

Give me 1 MORE nasi lemak bungkus please!

REQUEST > LIMIT

Sorry no more nasi lemak for you! ==

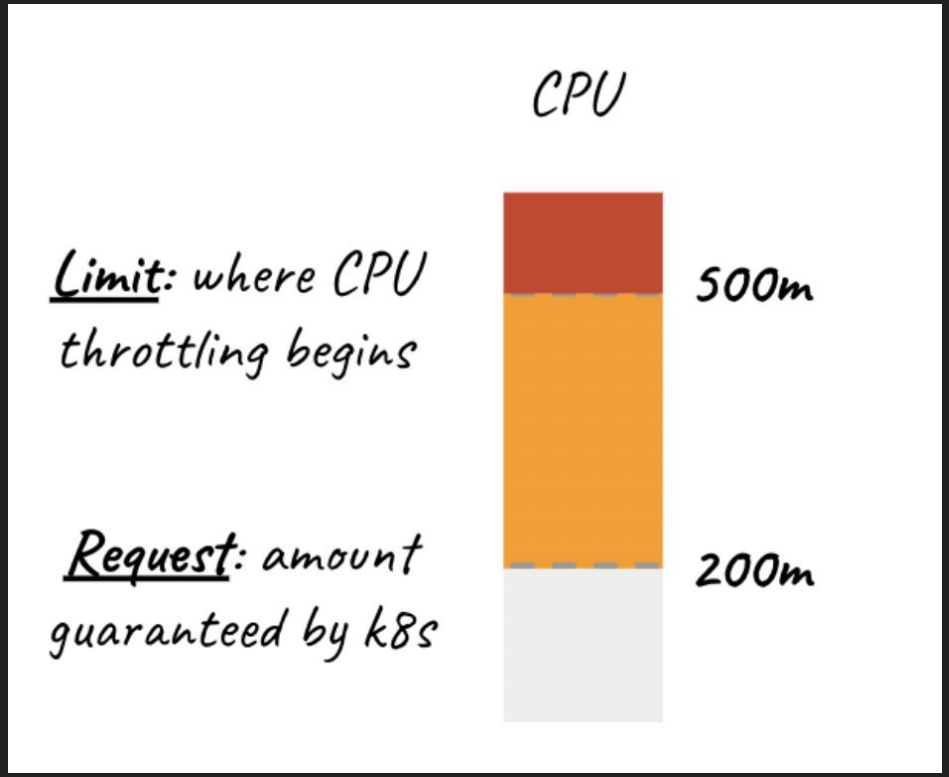
OOM KILLED

Canteen Operator == K8s Control

Limit

The maximum amount of a resource to be used by a container. This means that the container can never consume more than the memory amount or CPU amount indicated.

CPU




```yaml
Resources:
    requests:
      memory: "64Mi"
      cpu: "200m"
    limits:
      memory: "128Mi"
      cpu: "500m"
```

Image source:https://blog.kubecost.com/blog/requests-and-limits/

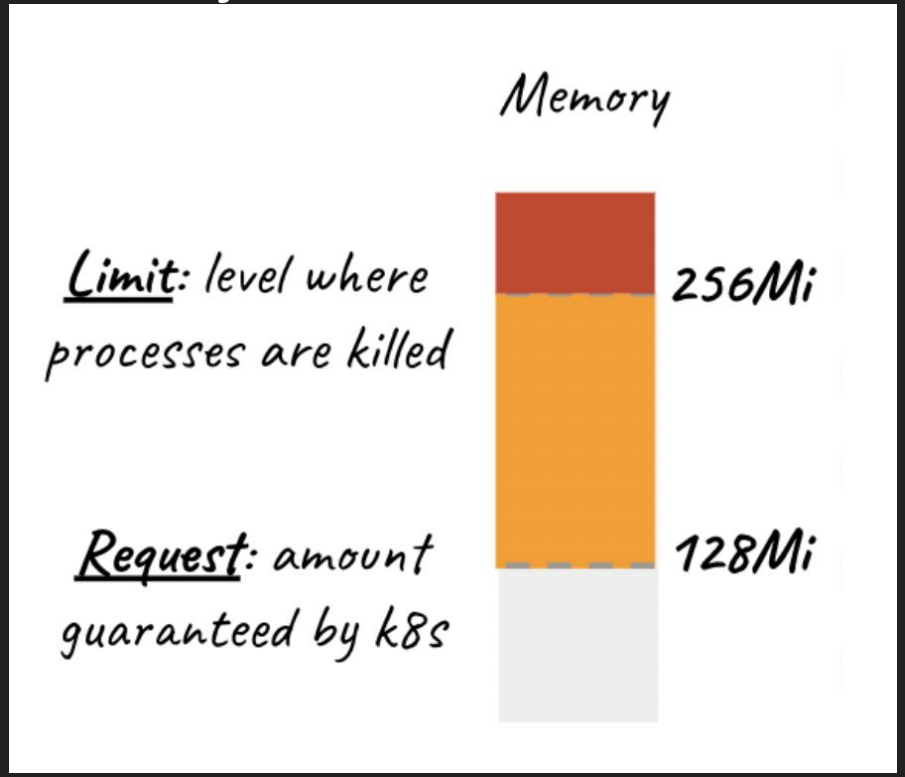
CPU

CPU is a compressible resource

“For future reference, note that some resources, such as CPU and network bandwidth, are *compressible*, which means that their usage can potentially be throttled in a relatively benign manner.”

Source: Kubernetes Design Proposal Archive (https://github.com/kubernetes/design-proposals-archive/blob/8da1442ea29adccea40693357d04727127e045ed/node/resource-qos.md#compressible-resource-guaranteess)

Memory




```yaml
Resources:
    requests:
      memory: "128Mi"
      cpu: "250m"
    limits:
      memory: "256Mi"
      cpu: "500m"
```

Image source:https://blog.kubecost.com/blog/requests-and-limits/

Memory

Memory is a incompressible resource

“For future reference, note that some resources, such as CPU and network bandwidth, are *compressible*, which means that their usage can potentially be throttled in a relatively benign manner. All other resources are *incompressible*, which means that any attempt to throttle them is likely to cause grief. This distinction will be important if a Kubernetes implementation supports over-committing of resources.”

Source: Kubernetes Design Proposal Archive (https://github.com/kubernetes/design-proposals-archive/blob/8da1442ea29adccea40693357d04727127e045ed/node/resource-qos.md#compressible-resource-guaranteess)

Best Practice of CPU/Mem Limit & Request Definition

Best Practice of CPU/Mem Limit & Request Definition

(as far as my research goes)

Best practice CPU

1. Use CPU requests for everything
2. Make sure they are accurate
3. Do **not** use CPU limits.

Source:https://home.robusta.dev/blog/stop-using-cpu-limits

Tim Hockin (one of the original Kubernetes maintainers at Google) has recommended the same for years.

ref:https://twitter.com/thockin/status/1134193838841401345?lang=en

Best practice CPU

All pods have...	CPU limits	No CPU limits
CPU requests	You are guaranteed CPU between the request and limit Excess cpu is unavailable beyond the limit	You are guaranteed your request. Excess CPU is available and not wasted! **
No CPU requests	You are guaranteed the limit, no more, no less* Excess cpu is unavailable	No one is guaranteed any CPU! Wild west.

* This is because Kubernetes automatically sets the request to the limit
** Excess CPU is given to whoever needs it, prioritized by the size of their request

Image source:https://home.robusta.dev/blog/stop-using-cpu-limits

"Excess CPU resources will be distributed based on the amount of CPU requested. For example, suppose container A requests for 600 milli CPUs, and container B requests for 300 milli CPUs. Suppose that both containers are trying to use as much CPU as they can. Then the extra 100 milli CPUs will be distributed to A and B in a 2:1 ratio (implementation discussed in later sections)."

ref:https://github.com/kubernetes/design-proposals-archive/blob/8da1442ea29adccea40693357d04727127e045ed/node/resource-qos.md#compressible-resource-guaranteess

Best practice Memory

1. Always use memory limits
2. Always use memory requests
3. Always set your memory requests equal to your limits

Source:https://home.robusta.dev/blog/stop-using-cpu-limits

Muhammed Danish @muhammeddanis7 · Nov 1, 2022

Could you please explain why you believe setting a memory limit == request is a good practise?

1

Tim Hockin (thockin.yaml)
@thockin

Memory can't always be reclaimed.

Consider a 4GiB node w/ 4 pods, each request=1GiB and limit=3GiB

Scheduler is happy: sum(requests[memory]) <= 4GiB

Pod0 uses 3GiB in anon pages
Pod1 uses 512MiB (ok)
Pod2 uses 512 MiB (ok)
Pod3 crashes

Pod0 caused a DoS for other pods.

5:21 AM · Nov 1, 2022

Tim Hockin (one of the original Kubernetes maintainers at Google) has recommended the same for years.

ref:https://twitter.com/thockin/status/1134193838841401345?lang=en

Tools to aid in Pod Right-Sizing

kube-capacity

Project repo: https://github.com/robscott/kube-capacity

- Requires metric server
- Shows the current (point in time) utilization

```
kube-capacity --util

NODE              CPU REQUESTS    CPU LIMITS    CPU UTIL    MEMORY REQUESTS    MEMORY LIMITS    MEMORY UTIL

*                 560m (28%)      130m (7%)     40m (2%)    572Mi (9%)         770Mi (13%)      470Mi (8%)

example-node-1    220m (22%)      10m (1%)      10m (1%)    192Mi (6%)         360Mi (12%)      210Mi (7%)

  example-node-2    340m (34%)      120m (12%)    30m (3%)    380Mi (13%)        410Mi (14%)      260Mi (9%)
```

EKS-NODE-VIEWER

Project repo: https://github.com/awslabs/eks-node-viewer

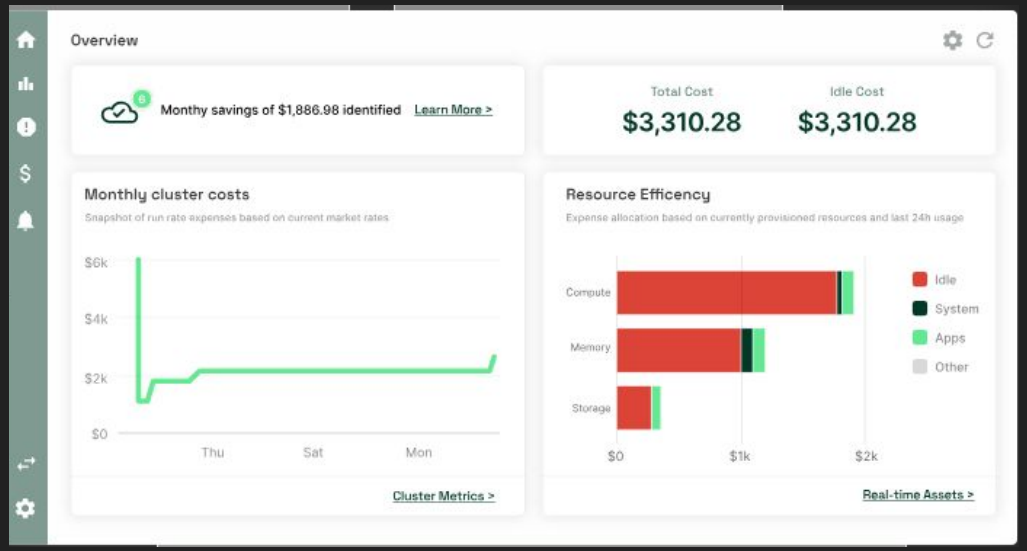
- Shows the current (point in time) utilization & cost
- Total CPU Limit /Node
- Pods/Node
- For EKS only

```
5 nodes 125500m/127330m 98.6% cpu $5.324/hour $3886.520/month
515 pods (0 pending 515 running 515 bound)

ip-192-168-132-238.us-west-2.compute.internal cpu  100% (130 pods) c6a.8xlarge/$1.224 On-Demand  ready
ip-192-168-93-114.us-west-2.compute.internal  cpu   99% ( 34 pods) c6a.2xlarge/$0.306 On-Demand  ready
ip-192-168-36-26.us-west-2.compute.internal   cpu   96% (140 pods) c4.8xlarge/$1.591 On-Demand  ready
ip-192-168-47-66.us-west-2.compute.internal   cpu   99% (145 pods) c4.8xlarge/$1.591 On-Demand  ready
ip-192-168-102-232.us-west-2.compute.internal cpu  100% ( 66 pods) c6a.4xlarge/$0.612 On-Demand  ready

Press any key to quit
```

kubecost



Project repo:
https://github.com/kubecost

- Requires prometheus
- Shows many different type of reports e.g cost/namespace or cost/pod

Robusta-KRR

Project repo: https://github.com/robusta-dev/krr

- Requires prometheus
- Default algorithm:
  - CPU request: 99.0% percentile, limit: unset
  - Memory request: max + 5.0%, limit: max + 5.0%

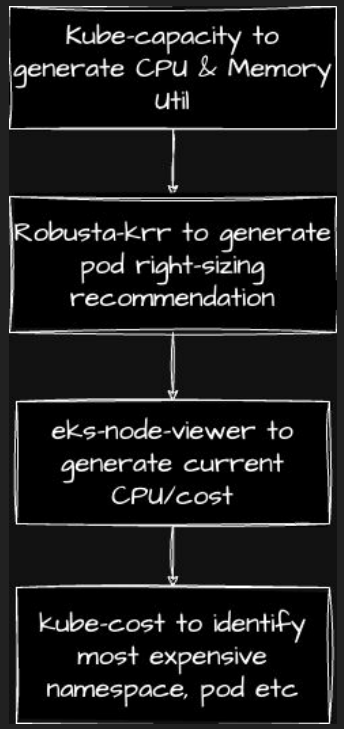
Robusta-KRR

Scan result (99.96 points)

Number	Cluster	Namespace	Name	Pods	Type	Container	CPU Requests	CPU Limits	Memory Requests	Memory Limits
1.	gke_robusta-development_us-east5-a_arik…	default	crashpod	3	Deployment	crashpod	none -> ?	none -> none	none -> 10M	none -> 10M
2.	gke_robusta-development_us-east5-a_arik…	default	hamster	1	Deployment	hamster	100m -> 171m	300m -> none	50Mi -> 10M	none -> 10M
3.	gke_robusta-development_us-east5-a_arik…	default	ng.inx-deployment	3	Deployment	nginx	none -> 5m	none -> none	none -> 10M	none -> 10M
4.	gke_robusta-development_us-east5-a_arik…	default	nginx-deployment	3	Deployment	nginx	none -> 5m	none -> none	none -> 10M	none -> 10M
5.	gke_robusta-development_us-east5-a_arik…	kubewatch	ng.inx-deployment	1	Deployment	nginx	none -> 5m	none -> none	none -> 10M	none -> 10M
6.	gke_robusta-development_us-east5-a_arik…	robusta	inline-crashpod	1	Deployment	crashpod	none -> ?	none -> none	none -> 10M	none -> 10M
7.	gke_robusta-development_us-east5-a_arik…	robusta	robusta-forwarder	1	Deployment	kubewatch	10m -> 8m	none -> none	512Mi -> 37M	512Mi -> 37M
8. 9. 10.	gke_robusta-development_us-east5-a_arik…	robusta	robusta-grafana	1	Deployment	grafana-sc-dashboard grafana-sc-datasources grafana	none -> 9m none -> 5m none -> 5m	none -> none none -> none none -> none	none -> 97M none -> 93M none -> 79M	none -> 97M none -> 93M none -> 79M
11.	gke_robusta-development_us-east5-a_arik…	robusta	robusta-kube-prometheus-st-operator	1	Deployment	kube-prometheus-stack	100m -> 5m	none -> none	none -> 30M	none -> 30M
12.	gke_robusta-development_us-east5-a_arik…	robusta	robusta-kube-state-metrics	1	Deployment	kube-state-metrics	10m -> 5m	none -> none	none -> 19M	none -> 19M
13.	gke_robusta-development_us-east5-a_arik…	robusta	robusta-runner	1	Deployment	runner	250m -> 105m	none -> none	1Gi -> 918M	1Gi -> 918M
14. 15.	gke_robusta-development_us-east5-a_arik…	robusta	alertmanager-robusta-kube-prometheus-st-…	1	StatefulSet	alertmanager config-reloader	50m -> 5m 200m -> 5m	none -> none none -> none	200Mi -> 36M 50Mi -> 10M	none -> 36M 50Mi -> 10M
16. 17.	gke_robusta-development_us-east5-a_arik…	robusta	prometheus-robusta-kube-prometheus-st-pr…	1	StatefulSet	prometheus config-reloader	50m -> 201m 200m -> 5m	none -> none none -> none	none -> 1060M 50Mi -> 14M	none -> 1060M 50Mi -> 14M
18.	gke_robusta-development_us-east5-a_arik…	robusta	robusta-prometheus-node-exporter	3	DaemonSet	node-exporter	50m -> 5m	none -> none	none -> 16M	none -> 16M

Image source:https://github.com/robusta-dev/krr

Possible approach in using the tools…



The tools will be very helpful to identify potential pod right-sizing opportunities within the cluster.

Do this exercise before and after right-sizing to compare resource utilization & cost saving.

Benefits

Free-up resources so that it could be allocated to all pods (less pods in pending state)

Cost Savings e.g need less node to run all your workloads

Workloads can scale better

Thank you. Any questions?

Reference

https://home.robusta.dev/blog/stop-using-cpu-limits

https://github.com/awslabs/eks-node-viewer

https://github.com/robscott/kube-capacity

https://learnk8s.io/kubernetes-instance-calculator

https://github.com/robusta-dev/krr

https://blog.kubecost.com/blog/requests-and-limits/

https://sysdig.com/blog/kubernetes-limits-requests/#:~:text=Kubernetes%20defines%20Limits%20as%20the,is%20reserved%20for%20a%20container.

https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

https://cloud.google.com/blog/products/containers-kubernetes/kubernetes-best-practices-resource-requests-and-limits