# Resource Optimization Dashboard

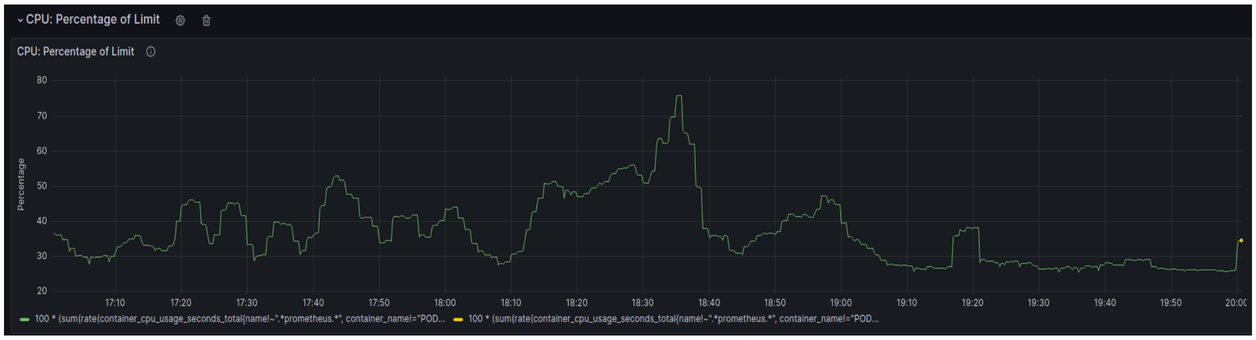
## Solution 1:

Somebody has already solved the exact problem in a beautiful manner.

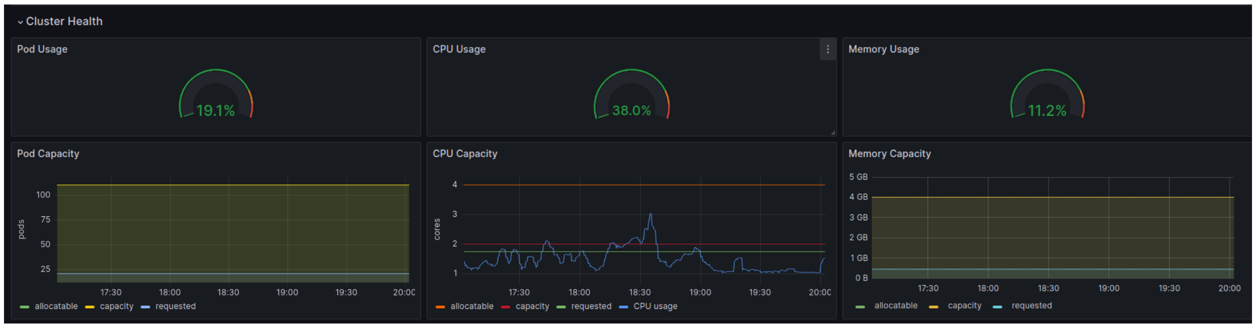
https://grafana.com/grafana/dashboards/9871-kube-eagle/

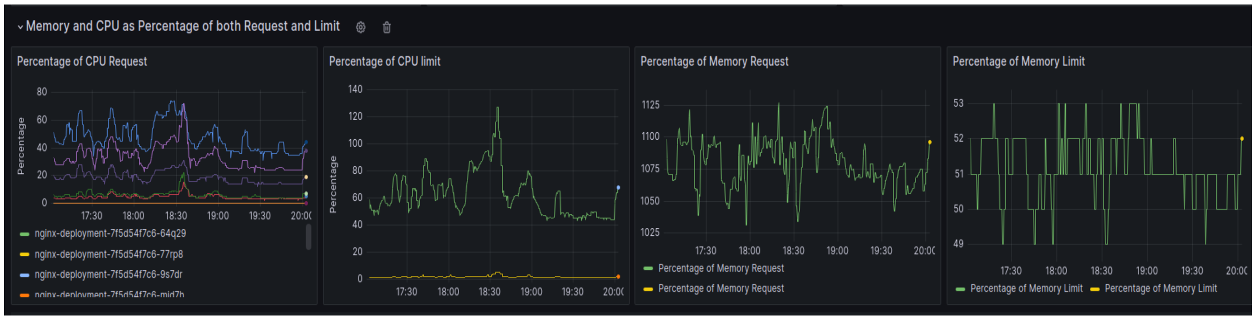
## Solution 2:

But there's not much effort in simply cloning the above dashboard, so I decided to start from scratch and see if I can build something completely on my own, and the end result was a dashboard looking like this. It may not be perfect but it's honest work.



```
100 * (sum(rate(container_cpu_usage_seconds_total{name!~".*prometheus.*", container_name!="POD...
100 * (sum(rate(container_cpu_usage_seconds_total{name!~".*prometheus.*", container_name!="POD...
```




Memory and CPU as Percentage of both Request and Limit
Percentage of CPU Request
Percentage
80
60
40
20
0
17:30
18:00
18:30
19:00
19:30
20:00
nginx-deployment-7f5d54f7c6-64q29
nginx-deployment-7f5d54f7c6-77rp8
nginx-deployment-7f5d54f7c6-9s7dr
nginx-deployment-7f5d54f7c6-mjd7h
Percentage of CPU limit
Percentage
140
120
100
80
60
40
20
0
17:30
18:00
18:30
19:00
19:30
20:00
Percentage of Memory Request
1125
1100
1075
1050
1025
17:30
18:00
18:30
19:00
19:30
20:00
Percentage of Memory Request
Percentage of Memory Request
Percentage of Memory Limit
53
52
51
50
49
17:30
18:00
18:30
19:00
19:30
20:00
Percentage of Memory Limit
Percentage of Memory Limit

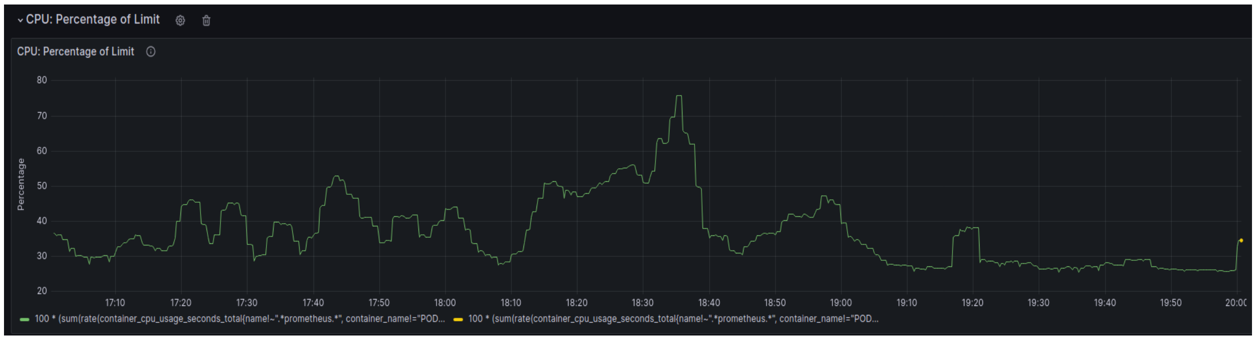
## Step I: Displaying CPU as Percentage of Limit

This is specific to k8s and containers that have CPU limits set.

To show CPU usage as a percentage of the limit given to the container, this is the Prometheus query I used to create nice graphs in Grafana.

100 * (sum(rate(container_cpu_usage_seconds_total{name!~".*prometheus.*", container_name!="POD"}[5m])) by (pod_name, container_name) / sum(container_spec_cpu_quota{name!~".*prometheus.*", container_name!="POD"}/container_spec_cpu_period{name!~".*prometheus.*", container_name!="POD"}) by (pod_name, container_name))

This results in the following graph:



## Step II: CPU: show as cores with request/limit lines

Since some applications have a small request and large limit (to save money) or have an HPA, then just showing a percentage of the limit is sometimes not useful.

So what we do now is display the CPU usage in cores and then add a horizontal line for each of the requests and limits. This shows more information and also shows the usage in the same metric that is used in k8s: CPU cores.

**CPU Usage:**

```
sum(rate(container_cpu_usage_seconds_total{container_name!="POD"}[5m])) by
(container_name, pod_name)
```

**CPU Request:**

```
sum(kube_pod_container_resource_requests{resource="cpu"})
```

**CPU Limit:**

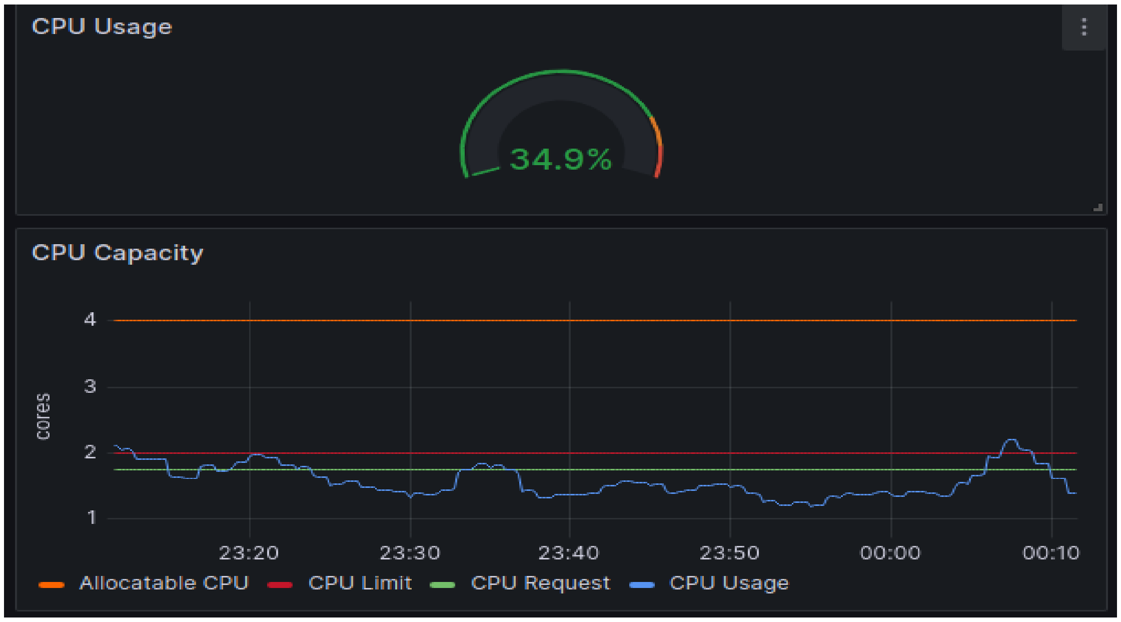
```
sum(kube_pod_container_resource_limits{resource="cpu"})
```

The pod request/limit metrics come from kube-state-metrics.

**Allocatable CPU resources:**

```
sum(kube_node_status_allocatable{resource="cpu"})
```

This results in a dashboard like this:



At this point, let us revisit the definition of ‘oversized’ and ‘undersized’ deployments.

**Oversized Deployment:** If a deployment requests a large amount of CPU or memory but only uses a fraction of it, it is considered "oversized" as it inefficiently consumes resources.

**Undersized Deployment:** If a deployment consistently maxes out its requested CPU or memory, leading to potential performance issues, it is considered "undersized".

Looking at the graph above, we can easily see that there are certain deployments that fall in the above categories. But the graph does not filter the CPU usage by deployment, so it's basically incomplete. I tried tweaking the query but ultimately was unable to do it 😅.

## Additional Queries and Graphs:

### Queries to show memory and CPU as percentage of both request and limit

(i) Percentage of CPU request:

```
round(
100 *
sum(
rate(container_cpu_usage_seconds_total{container_name!="POD"}[5m])
) by (pod, container_name, namespace, slave)
/
sum(
kube_pod_container_resource_requests{container_name!="POD", resource="cpu"}
) by (pod, container_name, namespace, slave)
)
```

(ii) Percentage of CPU Limit:

```
round(
  100 *
    sum(
      rate(container_cpu_usage_seconds_total{container_name!="POD"}[5m])
    ) by (pod_name, container_name, namespace, slave)
      /
    sum(
      container_spec_cpu_quota{container_name!="POD"} /
container_spec_cpu_period{container_name!="POD"}
    ) by (pod_name, container_name, namespace, slave)
)
```

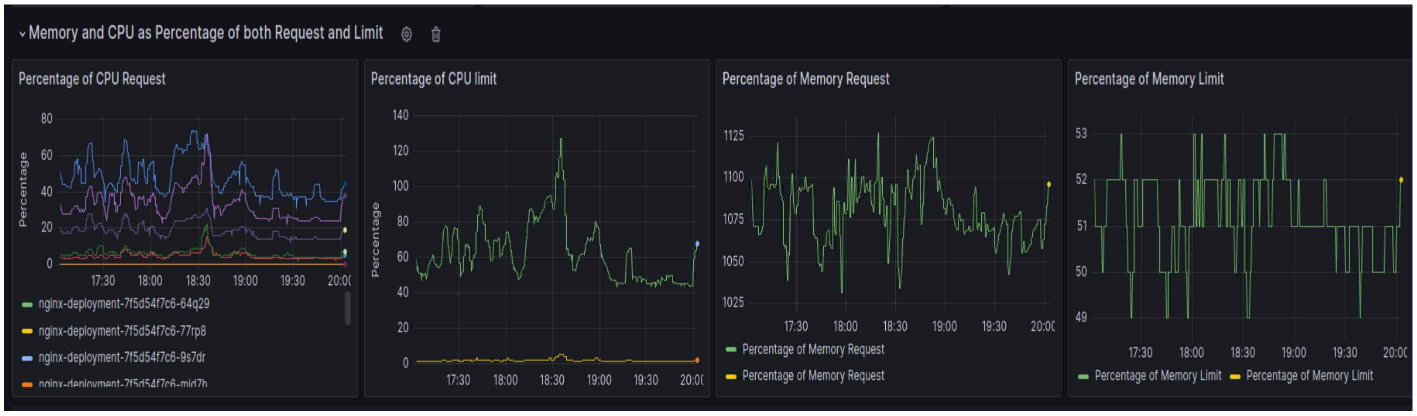
(iii) Percentage of memory request:

```
round(
  100 *
    sum(container_memory_working_set_bytes{container_name!="POD"}) by (container, pod,
namespace, slave)
      /
    sum(kube_pod_container_resource_requests_memory_bytes{container_name!="POD"} > 0)
by (container, pod, namespace, slave)
)
```

(iv) Percentage of Memory Limit:

```
round(
  100 *
    sum(container_memory_working_set_bytes{container_name!="POD"}) by (container,
pod_name, namespace, slave)
      /
    sum(container_spec_memory_limit_bytes{container_name!="POD"} > 0) by (container,
pod_name, namespace, slave)
)
```

**Each of the queries listed above result in graphs shown below:**


Memory and CPU as Percentage of both Request and Limit
Percentage of CPU Request
Percentage
nginx-deployment-7f5d54f7c6-64q29
nginx-deployment-7f5d54f7c6-77rp8
nginx-deployment-7f5d54f7c6-9s7dr
Percentage of CPU limit
Percentage
Percentage of Memory Request
Percentage of Memory Request
Percentage of Memory Request
Percentage of Memory Limit
Percentage of Memory Limit
Percentage of Memory Limit

## Potential challenges / limitations in creating or using this dashboard

- **Lack of Standardization:** One of the main challenges in using this dashboard is the lack of standardization. Different organizations have their own unique approaches to DevOps implementation, making it difficult to develop a standard set of metrics that can be used to measure success. This makes it challenging to benchmark progress, and can lead to confusion and frustration amongst stakeholders. The dashboard is not a one-size fits-all solution and might not fit the needs of all companies.

- **Limitation - Inability to view resource utilization by deployment:** Another limitation specific to this dashboard is that in its current state, the dashboard cannot be used to view resource utilization by deployment. Hence one can see that there are certain deployments that are oversized / undersized, but not pin-point which deployments are those. However, further optimizations can be done to display it properly.