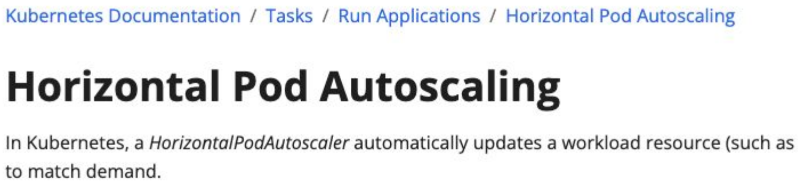
Multi-cloud application autoscaling with Thanos

Mihail Mihaylov
DevOps/SRE, MariaDB

Part 1: Autoscaling

The bible!



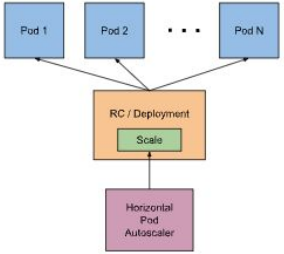
Kubernetes Documentation / Tasks / Run Applications / Horizontal Pod Autoscaling

Horizontal Pod Autoscaling

In Kubernetes, a *HorizontalPodAutoscaler* automatically updates a workload resource (such as to match demand.

…

How it works



```
--horizontal-pod-autoscaler-sync-period
```

Goals:

- ➔ No overprovisioning
- ➔ No under-provisioning
- ➔ No flapping state

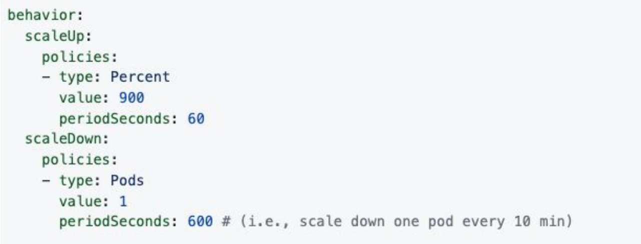
How?

- ➔ change the scaling behavior
- ➔ change the metrics/threshold
- ➔ do not touch the algorithm!

Deep dive:

SIG Autoscaling in k8s v1.18

autoscaling/v2beta2 HorizontalPodAutoscaler added a `spec.behavior`



```yaml
behavior:
  scaleUp:
    policies:
    - type: Percent
      value: 900
      periodSeconds: 60
  scaleDown:
    policies:
    - type: Pods
      value: 1
      periodSeconds: 600 # (i.e., scale down one pod every 10 min)
```

--horizontal-pod-autoscaler-downscale-stabilization => **behavior.scaleDown.stabilizationWindowSeconds**

Quick URL

HorizontalPodAutoscaler spec:

Disclaimer:

Do the calculation!

Smaller metrics resolution than scale up/down window.

The Algorithm!

```
desiredReplicas = ceil[
    currentReplicas * ( currentMetricValue / desiredMetricValue )
]
```

Current metric value: 200m

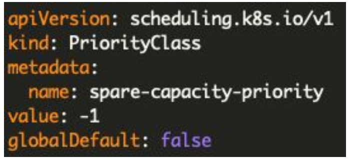
Desired metric value: 100m

=> double the replicas

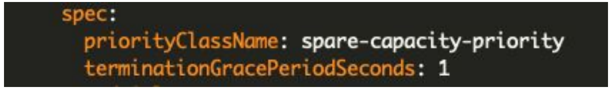
Bonus:

Cluster autoscaling

Pod Priority and Preemption



```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: spare-capacity-priority
value: -1
globalDefault: false
```



```
    spec:
      priorityClassName: spare-capacity-priority
      terminationGracePeriodSeconds: 1
```

How about the docker cache?

No silver bullet

Part 2: Metrics

Prometheus adapter

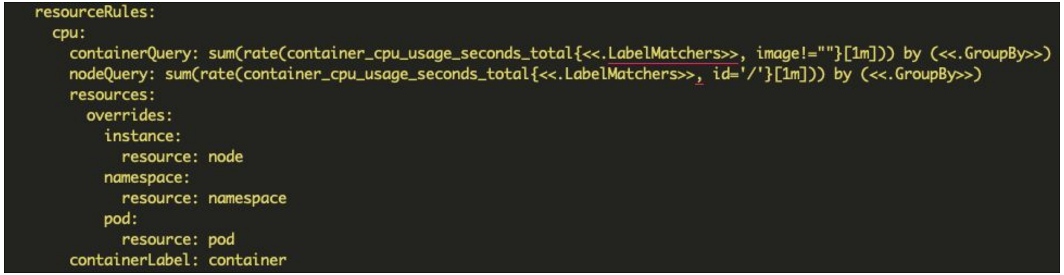
```
--prometheus-url=<url>

...

--metrics-max-age=<duration>

...

--config=<file>

...
```

metrics.k8s.io

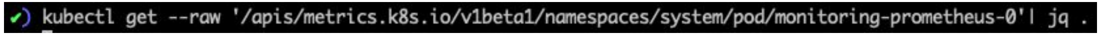
A per-pod resources metrics…

An utilization metric…

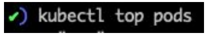
A percentage of the equivalent **resource request**…



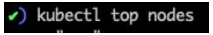
```
resourceRules:
  cpu:
    containerQuery: sum(rate(container_cpu_usage_seconds_total{<<.LabelMatchers>>, image!=""}[1m])) by (<<.GroupBy>>)
    nodeQuery: sum(rate(container_cpu_usage_seconds_total{<<.LabelMatchers>>, id='/'}[1m])) by (<<.GroupBy>>)
    resources:
      overrides:
        instance:
          resource: node
        namespace:
          resource: namespace
        pod:
          resource: pod
    containerLabel: container
```



```
✓) kubectl get --raw '/apis/metrics.k8s.io/v1beta1/namespaces/system/pod/monitoring-prometheus-0'| jq .
```



```
✓) kubectl top pods
```



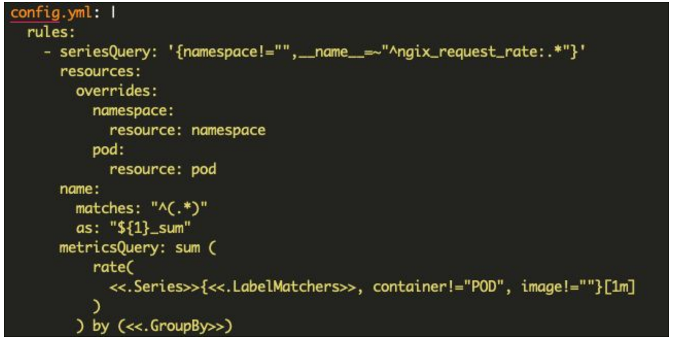
```
✓) kubectl top nodes
```

custom.metrics.k8s.io

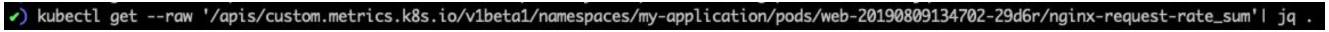
A per-pod metrics...

Not an utilization metric...

but raw metric values



```
config.yml: |
  rules:
    - seriesQuery: '{namespace!="",__name__=~"^ngix_request_rate:.*"}'
      resources:
        overrides:
          namespace:
            resource: namespace
          pod:
            resource: pod
      name:
        matches: "^(.*)"
        as: "${1}_sum"
      metricsQuery: sum (
          rate(
            <<.Series>>{<<.LabelMatchers>>, container!="POD", image!=""}[1m]
          )
        ) by (<<.GroupBy>>)
```



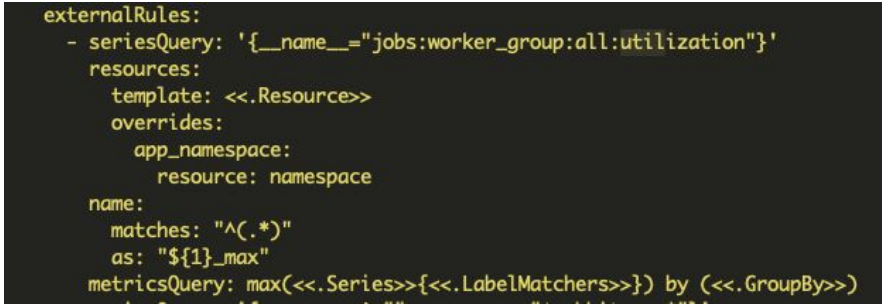
```
✔) kubectl get --raw '/apis/custom.metrics.k8s.io/v1beta1/namespaces/my-application/pods/web-20190809134702-29d6r/nginx-request-rate_sum'| jq .
```

external.metrics.k8s.io

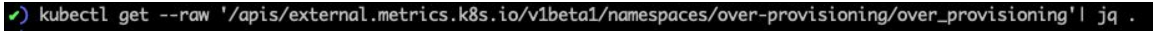
A non-pod metrics…

Single metric that describes the object…

It can be anything…



```
externalRules:
  - seriesQuery: '{__name__="jobs:worker_group:all:utilization"}'
    resources:
      template: <<.Resource>>
      overrides:
        app_namespace:
          resource: namespace
    name:
      matches: "^(.*)"
      as: "${1}_max"
    metricsQuery: max(<<.Series>>{<<.LabelMatchers>>}) by (<<.GroupBy>>)
```



```
✔) kubectl get --raw '/apis/external.metrics.k8s.io/v1beta1/namespaces/over-provisioning/over_provisioning'| jq .
```

Disclaimer:

Choose your metrics wisely!

Ready state = counted by the HPA

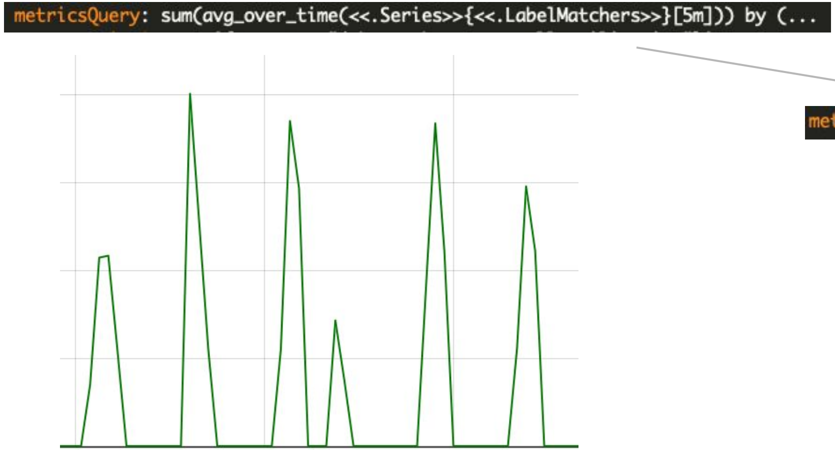
Disclaimer 2:

HPA scaling rules can be combined!

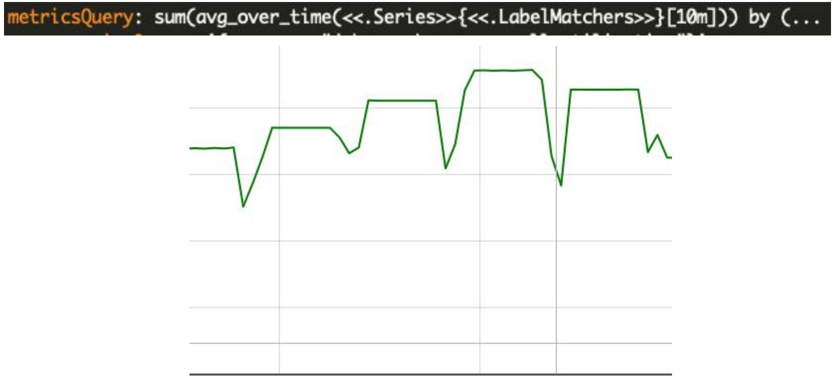
Define a safety net

Disclaimer 3:

Tweak the metrics not the HPA behaviour!



```
metricsQuery: sum(avg_over_time(<<.Series>>{<<.LabelMatchers>>}[5m])) by (...
```



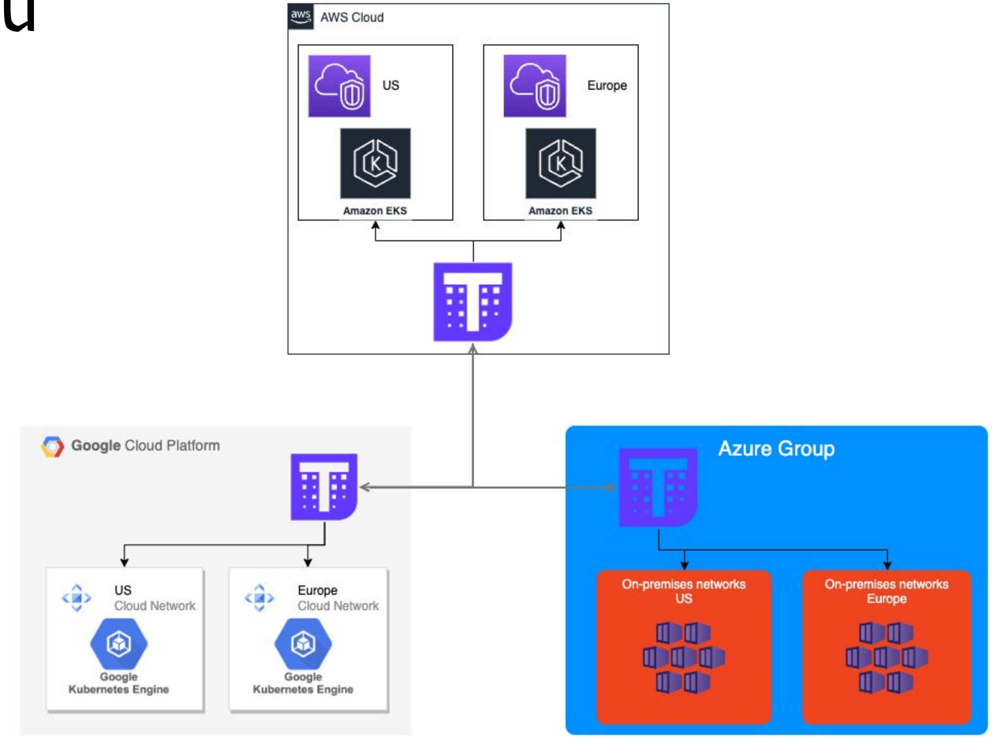
```
metricsQuery: sum(avg_over_time(<<.Series>>{<<.LabelMatchers>>}[10m])) by (...
```

Part 3: Multi-cluster Thanos

(The cool stuff)

HPA is as reliable as it’s metrics are

Multi-cloud


AWS Cloud
US
Amazon EKS
Europe
Amazon EKS
Google Cloud Platform
US
Cloud Network
Google
Kubernetes Engine
Europe
Cloud Network
Google
Kubernetes Engine
Azure Group
On-premises networks
US
On-premises networks
Europe

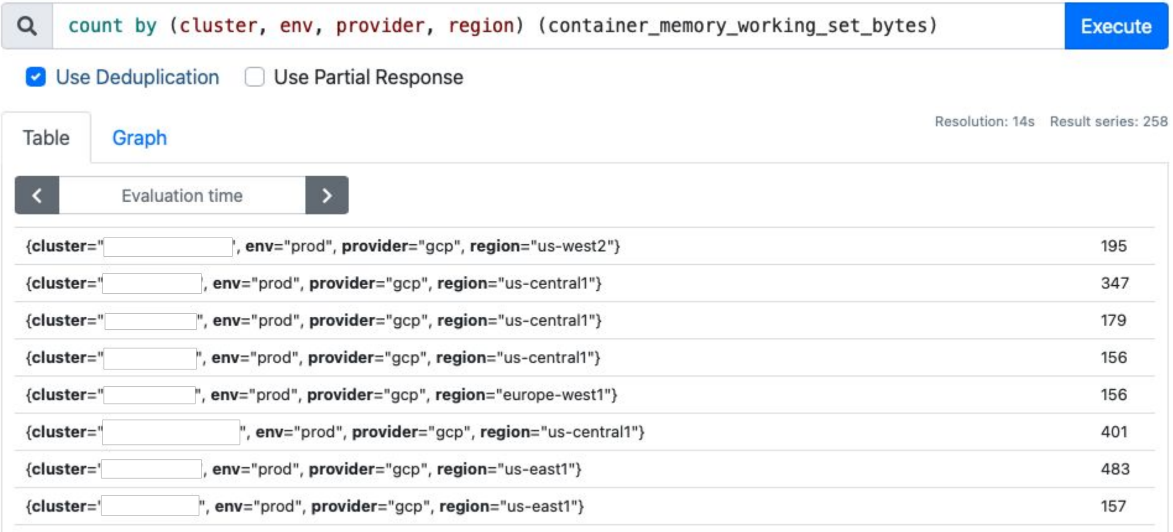
Example:

Cloud aware…

Zone aware…

Region aware…

Cluster aware…

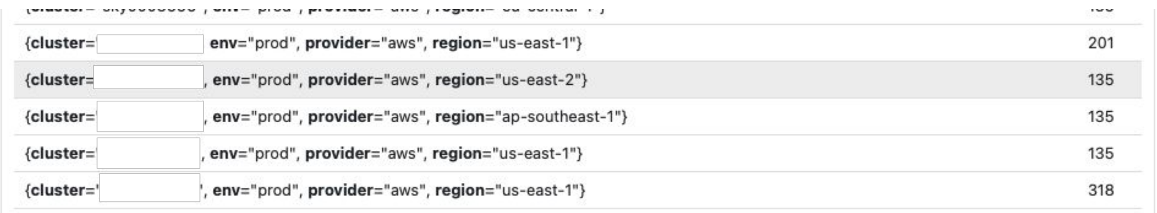


```
count by (cluster, env, provider, region) (container_memory_working_set_bytes)
```

☑ Use Deduplication ☐ Use Partial Response

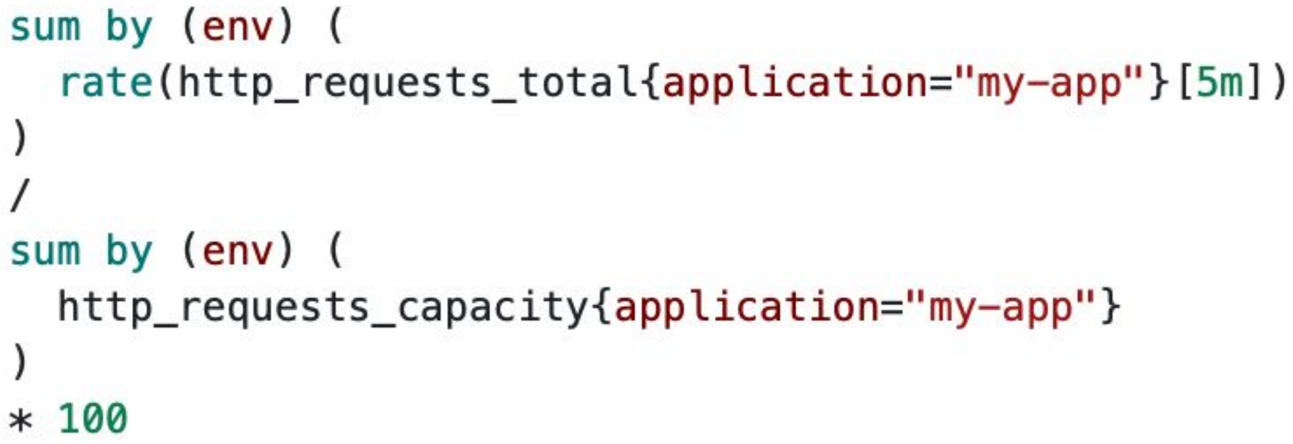
```
{cluster="[redacted]", env="prod", provider="gcp", region="us-west2"}         195
{cluster="[redacted]", env="prod", provider="gcp", region="us-central1"}      347
{cluster="[redacted]", env="prod", provider="gcp", region="us-central1"}      179
{cluster="[redacted]", env="prod", provider="gcp", region="us-central1"}      156
{cluster="[redacted]", env="prod", provider="gcp", region="europe-west1"}     156
{cluster="[redacted]", env="prod", provider="gcp", region="us-central1"}      401
{cluster="[redacted]", env="prod", provider="gcp", region="us-east1"}         483
{cluster="[redacted]", env="prod", provider="gcp", region="us-east1"}         157
```

……..



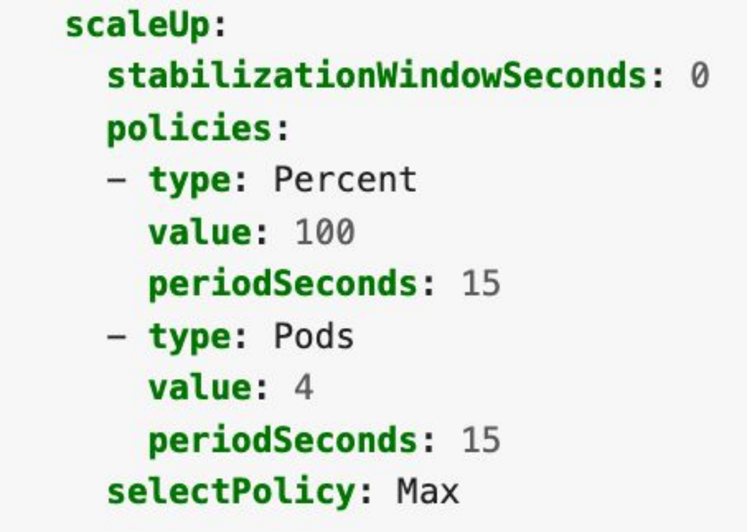
```
{cluster=[redacted] env="prod", provider="aws", region="us-east-1"}           201
{cluster=[redacted], env="prod", provider="aws", region="us-east-2"}          135
{cluster=[redacted], env="prod", provider="aws", region="ap-southeast-1"}     135
{cluster=[redacted], env="prod", provider="aws", region="us-east-1"}          135
{cluster="[redacted]', env="prod", provider="aws", region="us-east-1"}        318
```

first: The metric



```
sum by (env) (
  rate(http_requests_total{application="my-app"}[5m])
)
/
sum by (env) (
  http_requests_capacity{application="my-app"}
)
* 100
```

second: The step



```yaml
scaleUp:
  stabilizationWindowSeconds: 0
  policies:
  - type: Percent
    value: 100
    periodSeconds: 15
  - type: Pods
    value: 4
    periodSeconds: 15
  selectPolicy: Max
```

third: The traffic

It depends…

Service mesh…

Smart CDN…

maybe you don’t need an LB?

Do you need it?

… maybe not

The End.
QUestions?