An academic look at

KUBERNETES AUTO-SCALING

State-of-the-art
Approaches
Challenges
Innovation

HORIZONTAL POD AUTO-SCALING (HPA)

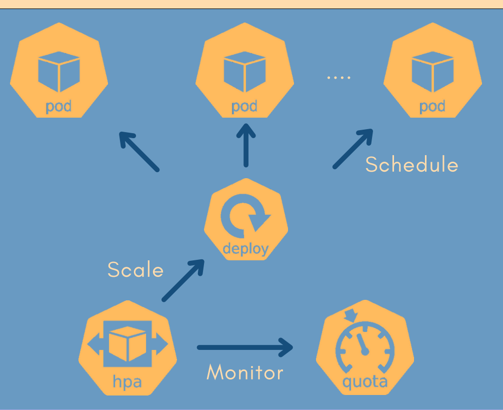
The most basic & common form of auto-scaling provided natively by Kubernetes.

Paper: *Agnostic Approach for Microservices Autoscaling in Cloud Applications*

Author: Khaleq et al.

K8S HPA ≃ HORIZONTAL SCALING

Kubernetes HPA varies slightly from the generic Horizontal scaling concept known outside its ecosystem. Instead of varying the number of nodes to scale in or out, applications in Kubernetes are scaled by scheduling or removing Pods, which are the building block of our systems and the unit containing our virtualized instances.



In its most basic form, all HPA does is periodically monitor resources (CPU, Memory) quotas configured. It will then call the controller to scale in or out, matching the observed metrics and following the specified scaling behaviour.

```yaml
type: Resource
resource:
  name: cpu
  target:
    type: Utilization
    averageUtilization: 60
```

```yaml
behavior:
  scaleDown:
    policies:
    - type: Pods
      value: 4
      periodSeconds: 60
    - type: Percent
      value: 10
      periodSeconds: 60
```

CUSTOM METRICS

Custom Metrics are a complimentary feature to HPA and other auto-scaling techniques.

Paper: Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration

Author: Nguyen et al.

Custom Metrics are widely used nowadays. It provides for a lot of flexibility in the configuration of auto-scaling. Pretty much anything can be fed to the Kubernetes API as a metric. The API allows us to define & declare metrics and consume data from external & internal sources, such as the Kubernetes Metrics Server.

```yaml
type: Object
object:
  metric:
    name: http_requests
    selector: {matchLabels:
{verb: GET}}
```

```yaml
- type: External
  external:
    metric:
      name: queue_messages_ready
      selector:
        matchLabels:
          queue: "worker_tasks"
    target:
      type: AverageValue
      averageValue: 30
```

By using adapters, we can easily transfer data from Prometheus, for example, to any federated monitoring solution such as Stack driver and by that improving accessibility and metrics management.

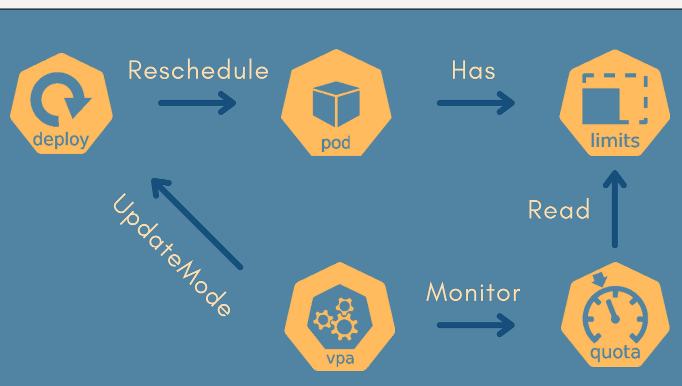
```yaml
spec:
      containers:
      - name: app
      ....
      - name: prometheus-to-sd
        image: gcr.io/google-containers/prometheus-
to-sd:v0.5.0
        command: ["/monitor"]
        args:
        - --source=:http://localhost:8080
        - --stackdriver-prefix=custom.googleapis.com
        - --pod-id=$(POD_ID)
        - --namespace-id=$(POD_NAMESPACE)
```

VERTICAL POD AUTO-SCALING (VPA)

Kubernetes VPA is far from the generic notion of Vertical Scaling. The feature itself is still in development. Also, due to its nature, VPA is a platform specific feature, meaning that you might not be able to use it depending on your cloud provider or environment.

Paper: Exploring Potential for Non-Disruptive Vertical Auto Scaling and Resource Estimation in Kubernetes

Author: Rattihalli et al.



K8S VPA ≠ VERTICAL SCALING

VPA in Kubernetes consists of editing resources requests and limits of Pods on the fly and therefore removes the need to specify them manually. It keeps track of resource usage, and then decides to increase or decrease the configured values for newly scheduled Pods, or for existing ones in some cases. The update process still requires to reschedule the Pod for now, but it is planned to have an *UpdateMode* without this constraint implemented at some point.

VPA configuration is still pretty similar to HPA. Custom Metrics are used in the same way, and instead of specifying a replicas number bounds, we specify resource bounds and target.

```yaml
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: my-vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind:       Deployment
    name:       my-rec-deployment
  updatePolicy:
    updateMode: "Off"
```

```yaml
recommendation:
    containerRecommendations:
    - containerName: my-container
      lowerBound:
        cpu: 536m
        memory: 262144k
      target:
        cpu: 587m
        memory: 262144k
      upperBound:
        cpu: 27854m
        memory: "545693548"
```

CLUSTER AUTO-SCALING

Cluster auto-scaling is closer to the conventional Horizontal Scaling known outside K8s ecosystem. Being directly bound to your system's infrastructure, it is a platform specific feature. In fact, in most cases, this would be configured directly through your cloud provider interface.

Paper: An Experimental Evaluation of the Kubernetes Cluster Autoscaler in the Cloud

Author: Tamiru et al.

As you may have guessed, Cluster auto-scaling allows for the dynamic addition and removal of nodes from a Kubernetes cluster. It also offers additional configurations, such as providing differentiated node pools and a selection strategy to fit into more specific use-cases. It also comes with a profile configuration option, allowing for cost optimization, for example.

```
$ gcloud container clusters create example \
    --num-nodes 2 \
    --zone us-central1-a \
    --enable-autoscaling --min-nodes 1 --max-nodes 4
```

```
$ gcloud beta container clusters update example \
--autoscaling-profile optimize-utilization
```

MULTIDIMENSIONAL POD AUTO-SCALING (MPA)

Now this is where we start truly getting out of the beaten tracks. MPA is still super early in its development stage. Even finding documentation for it might be a hustle in some cases. At this point, I would not consider it of much use with all the limitations it has. It is, however, a very interesting concept that might have a lot of potential later in its road map.

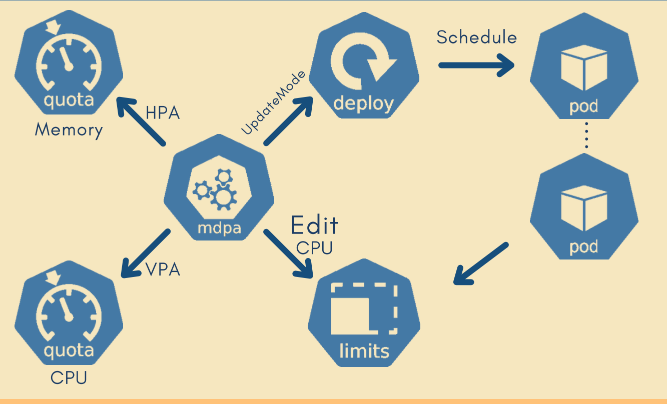
Paper: Adaptive scaling of Kubernetes pods

Author: Balla et al.

K8S MPA = VPA || HPA

The idea behind MPA isn't that complicated. It is basically an auto-scaling technique, allowing for both horizontal AND vertical auto-scaling simultaneously. However, HPA being coupled to Pod limits to trigger scaling and VPA editing these same limits, encountering conflicts and scaling misbehaviour, is inevitable. MPA dodges the issue by limiting the type of resource each dimension gets to work with.

```yaml
apiVersion: autoscaling.gke.io/v1beta1
kind: MultidimPodAutoscaler
metadata:
  name: autoscaler
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: app
```



For now, MPA manages CPU though its vertical axis and memory through its horizontal. It is however planned to have an implementation solving this limitation soon enough.

```yaml
  goals:
    metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 60
```

```yaml
  constraints:
    global:
      minReplicas: 1
      maxReplicas: 5
    containerControlledResources: [ memory ]
    container:
    - name: '*'
      requests:
        minAllowed:
          memory: 1Gi
        maxAllowed:
          memory: 2Gi
  policy:
    updateMode: Auto
```

MED MOUINE
E: MOHAMED.MOUINE.2@ULAVAL.CA