Pod autoscaling using Keda

Last updated by | Zoltan Toth | Aug 30, 2024 at 10:34 AM GMT+2

KEDA

Note: Open source installation using Helm is not recommended if you want to install using helm you can follow below procedure but in case of any issues reach KEDA official github page

The *Kubernetes-based Event Driven Auto-Scaler* will automatically scale a resource in a Kubernetes cluster based on a scale trigger: KEDA will monitor the event source, and feed that data to Kubernetes to scale the resource out/in accordingly, leveraging standard Kubernetes components (e.g. HPA) and extending the existing functionality without overwriting or duplicating components.

Contents

- KEDA
  - How KEDA works
    - Scaler
    - ScaledObject
    - CustomResourceDefnitions
  - Installation and Un-installation
    - Install using AKS add-on (Recommended)
    - Helm chart
  - Usage
    - ScaledObject
      - Example:
      - Testing Auto-Scaling with Event Scaler
    - ScaledJobs
    - Authentication
  - External Scalers
  - Troubleshooting
    - Keda pods are in crashloop/ healthprobe failures post inst...
    - Access logging and telemetry
    - Long running executions
    - Manually uninstall everything
  - Deprecated features and changes on KEDA 2.14.0+ on AKS...
  - Further readings
  - Sources

How KEDA works

KEDA integrates with multiple Scalers (event sources) and uses Custom Resources (CRDs) to define the required/desired scaling behavior and parameters. `Deployments` and `StatefulSets` are the most common way to scale workloads with KEDA.

It allows us to define the Kubernetes `Deployment` or `StatefulSet` that we want KEDA to scale based on a scale trigger.

KEDA will monitor that service and based on the events that occur it will automatically scale our resource out/in accordingly. Behind the scenes, KEDA monitors the event source and feeds that data to Kubernetes and the HPA (Horizontal Pod Autoscaler) to drive the rapid scale of a resource.

KEDA integrates with multiple Scalers (event sources) and uses Custom Resources (CRDs) to define the required/desired scaling behavior and parameters. Deployments and StatefulSets are the most common way to scale workloads with KEDA.

It allows us to define the Kubernetes Deployment or StatefulSet that we want KEDA to scale based on a scale trigger. KEDA will monitor that service and based on the events that occur it will automatically scale our resource out/in accordingly. Behind the scenes.

KEDA monitors the event source and feeds that data to Kubernetes and the HPA (Horizontal Pod Autoscaler) to drive the rapid scale of a resource.

Scaler

KEDA uses a Scaler to detect if a deployment should be activated or deactivated (scaling) which in turn is fed into a specific event source. Here we will use Cron event scaler to demonstrate Auto Scaling. We will scale out our application for some duration and then later scale it back to normal count.

Cron scaler is especially useful when there is a known pattern in the application usage like it’s used during weekdays and not used during weekends.

ScaledObject

ScaledObject is deployed as a Kubernetes CRD (Custom Resource Definition) which defines the relationship between an event source to a specific workload (i.e. Deployment, StatefulSet) for scaling.

For details and updated information see KEDA's concepts page.

CustomResourceDefnitions

Upon installation, KEDA creates the following custom resources to enable one to map an event source to a Deployment, StatefulSet, Custom Resource or Job for scaling:

- `scaledobjects.keda.sh`
- `scaledjobs.keda.sh`
- `triggerauthentications.keda.sh`

ScaledObjects represent a mapping between an event source (e.g. Rabbit MQ) and any K8S resource (Deployment, StatefulSet or Custom) defining the `/scale` subresource; *ScaledJobs* specifically represent a mapping between an event source and a Kubernetes Job.

TriggerAuthentication are referenced by ScaledObjects and ScaledJobs when they need to access authentication configurations or secrets to monitor the event source.

KEDA also creates the following deployments after the installation:

- `keda-operator`
- `keda-operator-metrics-apiserver`

The *operator* acts as an agent which activates, regulates and deactivates the scaling of K8S resources defined in a ScaledObject based on the trigger events.

The *metrics apiserver* exposes rich event data, like queue length or stream lag, to the Horizontal Pod Autoscaler to drive the scale out. It is then up to the resource to consume such events directly from the source.

KEDA offers a wide range of triggers (A.K.A. *scalers*) that can both detect if a resource should be activated or deactivated and feed custom metrics for a specific event source. The full list of scalers is available here .

Installation and Un-installation

Install using AKS add-on (Recommended)

Keda addon for AKS is now GA, its recommended to use KEDA addon as we get official support from Microsoft in case of any issues

You need to have use ARM/bicep apiversion > 2023-11-1 and azcli > 2.44.0

Ref: https://learn.microsoft.com/en-us/azure/aks/keda-deploy-add-on-cli

azcli:

```
#new cluster
az aks create \
  --resource-group myResourceGroup \
  --name myAKSCluster \
  --enable-keda

#existing cluster
az aks update --resource-group myResourceGroup --name myAKSCluster --enable-keda
```

Or using below code in your `akscluster.bicep` file

```
...
    workloadAutoScalerProfile: {
      keda: {
        enabled: true
      }
    }
...
```

For more details about `arm` / `bicep` templates refer:

https://learn.microsoft.com/en-us/azure/templates/microsoft.containerservice/managedclusters

Helm chart

```
# Installation.
helm repo add kedacore https://kedacore.github.io/charts \
 && helm repo update kedacore \
 && helm upgrade -i keda kedacore/keda \
      --namespace keda --create-namespace

# Uninstallation.
helm uninstall keda --namespace keda \
 && kubectl delete namespace keda
```

Usage

One can just add a resource to their deployment using the Custom Resource Definitions KEDA offers:

- **ScaledObject** for Deployments, StatefulSets and Custom Resources
- **ScaledJob** for Jobs

ScaledObject

For details and updated information see KEDA's Scaling Deployments, StatefulSets and Custom Resources page.

The ScaledObject Custom Resource definition is what defines how KEDA should scale one's application and what the triggers (A.K.A. scalers) are. The full list of scalers is available here :

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: {{ scaledObject.name }}
spec:
  scaleTargetRef:
    apiVersion: {{ targetResource.apiVersion }}    # optional; defaults to 'apps/v1'
    kind:       {{ targetResource.Kind }}          # optional; defaults to 'Deployment'
    name:       {{ targetResource.Name }}          # mandatory; the target resource must reside in the same
    envSourceContainerName: {{ container.name }}   # optional; defaults to the target's '.spec.template.spe
  pollingInterval: 30                              # optional; defaults to 30 seconds
  cooldownPeriod:  300                             # optional; defaults to 300 seconds
  minReplicaCount: 0                               # optional; defaults to 0
  maxReplicaCount: 100                             # optional; defaults to 100
  advanced:                                        # optional
    restoreToOriginalReplicaCount: false           # optional; defaults to false
    horizontalPodAutoscalerConfig:                 # optional
      behavior:                                    # optional; modifies the HPA's default scaling behavior
        scaleDown:
          stabilizationWindowSeconds: 300
          policies:
          - type: Percent
            value: 100
            periodSeconds: 15
  triggers: []                                     # mandatory; list of the triggers (= scalers) which will
```

Custom Resources are scaled the same way as Deployments and StatefulSets, as long as the target Custom Resource defines the `/scale` subresource .

When a ScaledObject is already in place and one first creates the target resource, KEDA will immediately scale it to the value of the `minReplicaCount` specification and will then scale it up according to the triggers.

The `scaleTargetRef` specification references the resource KEDA will scale up/down and setup an HPA for, based on the triggers defined in `triggers` . The resource referenced by `name` (and `apiVersion` and `kind`) must reside in the same namespace as the ScaledObject.

`envSourceContainerName` specifies the name of the container inside the target resource from which KEDA will retrieve the environment properties holding secrets etc. If not defined, KEDA will try to retrieve the environment properties from the first Container in the resource's definition.

`pollingInterval` is the interval to check each trigger on. In a queue scenario, for example, KEDA will check the `queueLength` every `pollingInterval` seconds, and scale the resource up or down accordingly.

`cooldownPeriod` sets how much time to wait after the last trigger reported active, before scaling the resource back to `minReplicaCount`. This only applies after a trigger occurs **and** when scaling down to a `minReplicaCount` value of 0: scaling from 1 to N replicas is handled by the Kubernetes Horizontal Pod Autoscaler.

`minReplicaCount` is the minimum amount of replicas KEDA will scale the resource down to. If a non default value (> 0) is used, it will not be enforced, meaning one can manually scale the resource to 0 and KEDA will not scale it back up. However, KEDA will respect the value set there when scaling the resource afterwards.

`maxReplicaCount` sets the maximum amount of replicas for the resource. This setting is passed to the HPA definition that KEDA will create for the target.

`restoreToOriginalReplicaCount` specifies whether the target resource should be scaled back to original replicas count after the ScaledObject is deleted. The default behavior is to keep the replica count at the number it is set to at the moment of the ScaledObject's deletion.

If running on Kubernetes v1.18+, the `horizontalPodAutoscalerConfig.behavior` field allows the HPA's scaling behavior to be configured feeding the values from this section directly to the HPA's behavior field.

Example:

I am taking Cron scaler as a trigger event. The above ScaledObject Custom Resource is used to define how KEDA will scale our application and what the triggers are. The minimum replica count is set to 0 and the maximum replica count is 5.

you can find list of other scalers here

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: gatekeeper-deployment
  namespace: gatekeeper-system
spec:
  scaleTargetRef:
    apiVersion:    {{ targetResource.apiVersion }}         # Optional. Default: apps/v1
    kind:          {{ targetResource.Kind }}               # Optional. Default: Deployment
    name:          {{ targetResource.Name }}               # Mandatory. Must be in the same namespace as th
  pollingInterval:  5                                      # Optional. Default: 5 seconds
  cooldownPeriod:   300                                    # Optional. Default: 300 seconds
  minReplicaCount:  0                                      # Optional. Default: 0
  maxReplicaCount:  5                                      # Optional. Default: 100
  fallback:                                                # Optional. Section to specify fallback options
    failureThreshold: 3                                    # Mandatory if fallback section is included
    replicas: 1                                            # Mandatory if fallback section is included
  advanced:                                                # Optional. Section to specify advanced options
    restoreToOriginalReplicaCount: true                    # Optional. Default: false
    horizontalPodAutoscalerConfig:                         # Optional. Section to specify HPA related optio
      name: {{ name_of_HPA }}                          # Optional. Default: keda-hpa-{scaled-object-name}
      behavior:                                            # Optional. Use to modify HPA's scaling behavior
        scaleDown:
          stabilizationWindowSeconds: 600
          policies:
            - type: Percent
              value: 100
              periodSeconds: 15
  triggers:
  - type: cron
    metadata:
      # Required
      timezone: Asia/Kolkata  # The acceptable values would be a value from the IANA Time Zone Database.
      start: 25 * * * *       # Every hour on the 30th minute
      end: 35 * * * *         # Every hour on the 45th minute
      threshold: '10'
```

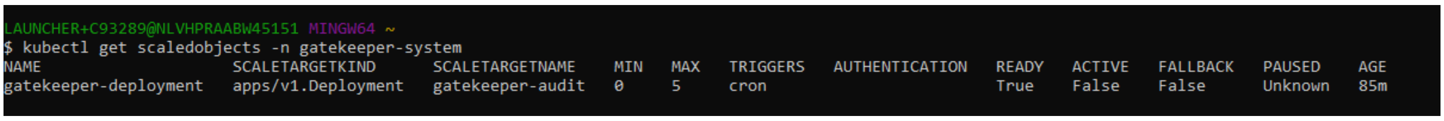
Once the `ScaledObject` is created, the KEDA controller automatically syncs the configuration and starts watching the deployment nginx created above. KEDA seamlessly creates a HPA (Horizontal Pod Autoscaler) object with the required configuration and scales out the replicas based on the trigger-rule provided through ScaledObject (in this case it is Cron expression).

Testing Auto-Scaling with Event Scaler

Once the `ScaledObject` is created for an application with a relevant trigger, KEDA will start monitoring our application. We can check the status of `ScaledObjects` using the below command

```
kubectl get scaledobjects -n <namespace>
```

Example:



```
LAUNCHER+C93289@NLVHPRAABW45151 MINGW64 ~
$ kubectl get scaledobjects -n gatekeeper-system
NAME                   SCALETARGETKIND      SCALETARGETNAME    MIN   MAX   TRIGGERS   AUTHENTICATION   READY   ACTIVE   FALLBACK   PAUSED    AGE
gatekeeper-deployment  apps/v1.Deployment   gatekeeper-audit   0     5     cron                        True    False    False      Unknown   85m
```

and to check autoscaling events/logs you can check keda operator pod logs

```
kubectl logs -f keda-operator-xxxx-xxxx -n keda --tail=50
```

ScaledJobs

The ScaledJob Custom Resource definition is what defines how KEDA should scale a Job and what the triggers (*scalers*) are. The full list of scalers is available here .

Instead of scaling up the number of replicas, KEDA will schedule a single Job for each detected event. For this, a ScaledJob is primarily used for long running executions or small tasks being able to run in parallel in massive spikes like processing queue messages.

For details and updated information see KEDA's Scaling Jobs page.

```
apiVersion: keda.sh/v1alpha1
kind: ScaledJob
metadata:
  name: {{ scaledJob.name }}
spec:
  jobTargetRef:
    parallelism: 1                              # [max number of desired pods](https://kubernetes.io/docs/
    completions: 1                              # [desired number of successfully finished pods](https://k
    activeDeadlineSeconds: 600                  # specifies the duration in seconds relative to the startT
    backoffLimit: 6                             # specifies the number of retries before marking this job
    template:                                   # describes the [job template](https://kubernetes.io/docs/
  pollingInterval: 30                           # optional; defaults to 30 seconds
  successfulJobsHistoryLimit: 5                 # optional; how many completed jobs should be kept as hist
  failedJobsHistoryLimit: 5                     # optional; how many failed jobs should be kept as history
  envSourceContainerName: {{ container.name }}  # optional; defaults to the target's '.spec.JobTargetRef.t
  maxReplicaCount: 100                          # optional; defaults to 100
  scalingStrategy:
    strategy: "custom"                          # optional; which Scaling Strategy to use; defaults to 'de
    customScalingQueueLengthDeduction: 1        # optional; a parameter to optimize custom ScalingStrategy
    customScalingRunningJobPercentage: "0.5"    # optional; a parameter to optimize custom ScalingStrategy
  triggers: []                                  # list of the triggers (= scalers) which will spawn jobs
```

`pollingInterval` is the interval in seconds KEDA will check each trigger on.

`successfulJobsHistoryLimit` and `failedJobsHistoryLimit` specify how many *completed* and *failed* jobs should be kept, similarly to Jobs History Limits; it allows to learn what the outcome of the jobs are.
The actual number of jobs could exceed the limit in a short time, but it is going to resolve in the cleanup period. Currently, the cleanup period is the same as the Polling interval.

`envSourceContainerName` specifies the name of container in the target Job from which KEDA will retrieve the environment properties holding secrets etc. If not defined, KEDA will try to retrieve the environment properties from the first Container in the target resource's definition.

`maxReplicaCount` is the max number of Job Pods to be in existence within a single polling period. If there are already some running Jobs others will be created only up to this numbers, or none if their number exceeds this value.

`scalingStrategy` is one from *default*, *custom*, or *accurate*. For details and updated information see this PR .

Example:

```
apiVersion: v1
kind: Secret
metadata:
  name: rabbitmq-consumer
data:
  RabbitMqHost: <omitted>
---
apiVersion: keda.sh/v1alpha1
kind: ScaledJob
metadata:
  name: rabbitmq-consumer
  namespace: default
spec:
  jobTargetRef:
    template:
      spec:
        containers:
        - name: rabbitmq-client
          image: tsuyoshiushio/rabbitmq-client:dev3
          imagePullPolicy: Always
          command: ["receive",  "amqp://user:PASSWORD@rabbitmq.default.svc.cluster.local:5672", "job"]
          envFrom:
            - secretRef:
                name: rabbitmq-consumer
        restartPolicy: Never
    backoffLimit: 4
  pollingInterval: 10
  maxReplicaCount: 30
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 2
  scalingStrategy:
    strategy: "custom"
    customScalingQueueLengthDeduction: 1
    customScalingRunningJobPercentage: "0.5"
  triggers:
  - type: rabbitmq
    metadata:
      queueName: hello
      host: RabbitMqHost
      queueLength  : '5'
```

Authentication

For details and updated information see KEDA's Authentication page.

External Scalers

For details and updated information see KEDA's External Scalers page.

Troubleshooting

Keda pods are in crashloop/ healthprobe failures post installation.

If you are seeing any healthprobe failure/ crashloop due, it could be due to calico policies. in FSCP 3.0 by deafult all egress and ingress traffic is blocked. you can create allow egress and ingress policies to resolve the issue.

refer official k8s doc for more details on how do we create network policies.
https://kubernetes.io/docs/concepts/services-networking/network-policies/

Access logging and telemetry

Use the logs for the keda operator or apiserver:

```
kubectl logs --namespace keda keda-operator-8488964969-sqbxq
kubectl logs --namespace keda keda-operator-metrics-apiserver-5b488bc7f6-8vbpl
```

Long running executions

There is at the moment of writing no way to control which of the replicas get terminated when a HPA decides to scale down a resource. This means the HPA may attempt to terminate a replica that is deep into processing a long execution (e.g. a 3 hour queue message). To handle this:

- leverage `lifecycle hooks` to delay termination
- use a Job to do the processing instead of a Deployment/StatefulSet/Custom Resource.

Manually uninstall everything

Just run the following:

```
kubectl delete -f https://raw.githubusercontent.com/kedacore/keda/main/config/crd/bases/keda.sh_scaledobjec
kubectl delete -f https://raw.githubusercontent.com/kedacore/keda/main/config/crd/bases/keda.sh_scaledjobs.
kubectl delete -f https://raw.githubusercontent.com/kedacore/keda/main/config/crd/bases/keda.sh_triggerauth
```

and then delete the namespace.

Deprecated features and changes on KEDA 2.14.0+ on AKS 1.30.0+

- General: Remove deprecated `metricName` from trigger metadata section.
  The two impacted Azure Scalers are **Azure Blob Scaler** and **Azure Log Analytics Scaler**.
  If you are using metricName today, please move `metricName` outside of trigger metadata section to `trigger.name` in the trigger section to optionally name your trigger

An example before KEDA 2.14

```
triggers:
- type: any-type
  metadata:
    metricName: "my-custom-name"
```

An example after KEDA 2.14

```
triggers:
- type: any-type
  name: "my-custom-name"
  metadata:
```

e.g: https://keda.sh/docs/2.10/scalers/azure-log-analytics/

- General: Clean up previously deprecated code in **Azure Data Explorer Scaler** about `clientSecret` for 2.13 release, use Azure Managed Workload Identity based authentication instead
- Remove support for Azure AD Pod Identity-based authentication, use Azure Managed Workload Identity based authentication instead

Further readings

- KEDA's concepts
- Authentication
- External Scalers
- Scaling Deployments, StatefulSets and Custom Resources
- Scaling Jobs
- The complete scalers list
- The project's website
- The project's FAQ s

Sources

- KEDA: Event Driven and Serverless Containers in Kubernetes by Jeff Hollan, Microsoft
- The `/scale` subresource
- The ScaledObject specification