Using KEDA with Service Metrics

Document Level Classification

300

Introduction

Services that emit metrics using OpenTelemetry or Prometheus can use these metrics for Autoscaling. This is implemented by forwarding a subset of your metrics to a cluster-local AMP workspace that is provisioned with your EKS cluster. Some key points about this implementation:

- We use AMP because it keeps the infrastructure that is driving scaling decisions separate from Epic's other observability systems. This prevents an outage of Chronosphere or New Relic from impacting your apps ability to scale.

- We forward a subset of metrics to AMP to keep costs down. AMP only charges for what we ingest, so sending only metrics used for auto-scaling keeps the cost of the solution to a minimum.
- The AMP workspaces that receive scaling metrics are not available for queries by Grafana - use your standard observability datasources for dashboarding/alerts.

Setup

Sending Metrics

This solution can work with any metrics sent via the OTEL collector in Kubernetes. See Metrics#Howtoemitmetrics for specifics on how to send metrics. In general you will either be:

- emitting metrics directly to the cluster's OpenTelemetry Protocol (OTLP) collector using OTEL SDKs or Auto-instrumentation
- exporting metrics with a prometheus endpoint, which will be picked up when your pod is annotated with **prometheus.io/scrape: true**

Choosing Scaling Metrics

Because we forward only a subset of metrics to the AMP datasource that KEDA queries, we need to know what your scaling metrics are so they can be forwarded properly. By default, we forward anything with a **keda_** prefix, and all **http_server_duration** metrics from the OTEL auto-instrumentation agent, but this can be expanded and customized. The default values are here for reference. To add more metrics to this list, reach out to #ct-obs-support-ext and ask for help updating the metricsRegexAllowList for KEDA. Over time as a standard pattern emerges for scaling metrics, we expect to have these forwarded by default in the OTEL stack.

Self-service Option with Argo

If you're comfortable submitting ArgoCD pull requests for your EKS clusters, you can self-service by submitting a PR and updating **otel_stack.values.collectors.exporters.kedaMetrics.metricsRegexAllowList** with your own list of metrics. Ask in #cloud-ops-support-ext for help with review and applying the config.

```
otel_stack:
  values:
```

```
    collectors:
      exporters:
        kedaMetrics:
          enabled: true
          metricsRegexAllowList:
            - my_custom_metric
            - my_custom_prefix_.+
```

Adding a KEDA ScaledObject to your app

There is more extensive documentation on KEDA here which you are encouraged to read: Pod Autoscaling with HPA and KEDA - for this page we'll focus on a specific example of using KEDA with service metrics.

KEDA is managed using ScaledObject Custom Resources. These are supported in epic-app, so you just need to update your app's values.yaml file and re-deploy to start using KEDA. You can see a practical implementation of this in the csk-apm-demo app. But let's review what a config looks like:

To deploy a similar config to the one below, you will need to look up the AMP workspace URL for your EKS cluster. This is available in the AWS Prometheus Console.

```
epic-app:
  autoscaling:
    keda:
      enabled: true # setting enabled here implicitly disables the defa
      scaledObjects:
        scaler:
          minReplicaCount: 2 # set the minimum number of pods this will
          maxReplicaCount: 50 # set the maxmimum number of pods this wi
          scaleTargetRef:
            apiVersion: apps/v1
```

```
    kind: Deployment # we're targetting a deployment here, if u
    name: csk-apm-demo-gamedev # this is the deployment targete
  advanced:
    horizontalPodAutoscalerConfig:
      behavior:
        scaleDown:
          policies:
            - periodSeconds: 60 # this limits scaling activity
              type: Pods
              value: 1 # we scale down one pod at a time
          # https://kubernetes.io/docs/tasks/run-application/ho
          stabilizationWindowSeconds: 60 # adjust this up if re
        scaleUp:
          policies:
            - periodSeconds: 60
              type: Pods
              value: 2 # we scale up two pods at a time
          stabilizationWindowSeconds: 30
    restoreToOriginalReplicaCount: false # determines whether K
  triggers:
    - type: prometheus
      authenticationRef:
        name: aws
      metadata:
        awsRegion: us-east-1
        serverAddress: "https://aps-workspaces.us-east-1.amazon
        query: 'sum(rate(http_server_duration_milliseconds_coun
        threshold: "20.0" # scale at 20 requests/sec
        identityOwner: operator
        metricType: AverageValue # query is averaged by number
triggerAuthentications: # this is boilerplate that tells KEDA to
  aws:
    podIdentity:
      provider: aws
```

Note

If you are already using an existing HPA to manage your deployment (such as on CPU or Memory), you will need to remove it before deploying a KEDA HPA. This may need to be done manually, as epic-app does not gracefully manage the transition from one HPA to another.
