Scaling pods

Document Level Classification

200

Introduction

Pod scaling is the scaling out of your application instances within the cluster to meet increased demand or scaling in when resources are not needed.

Applications can be scaled both vertically, by increasing the requested CPU and memory, as well as horizontally by increasing the number of pods. Your Substrate cluster supports horizontal scaling using Horizontal Pod Autoscaling (HPA) or Kubernetes Event Driven Architecture (KEDA).

Horizontal Pod Autoscaling (HPA)

If you are using epic-app, HPA is enabled by default. The default behavior is that scaling up is triggered by CPU utilization of 60%.

NOTE: the term "workload" below generally means a `Deployment` or `StatefulSet`. `DaemonSet` isn't included here because they don't scale horizontally as they are normally present on every node in a cluster and having more than one on a node would not help.

Basic pod scaling can be done without extra addons in Kubernetes using a `HorizontalPodAutoscaler` resource. By default, without any additional addons installed, an HPA can scale a workload based on CPU and/or memory. The documentation does a great job describing HPAs, but the gist of it is this: you set the minimum and maximum number of pods allowed and a threshold of resource usage (normally CPU) expressed as a percentage. When the average CPU usage across all pods in your workload is greater than that threshold, the HPA will cause the workload to scale up until it levels off at roughly that target percentage.

Note: The percentage is related to the resource requests on the pod. So if you request `500m` (half a CPU) for each pod, and on average the pods are using `250m`, then you are at 50% utilization.

There are other intracacies to the configuration you can explore that are documented in the above link.

The snippet below from the epic-app values.yaml file shows the default autoscaling parameters. You can set the minimum and maximum number of pods allowed and a threshold of resource usage expressed as a percentage. If you take average CPU usage as the resource metric as shown in the snippet below, when the average CPU usage across all pods is greater than that threshold, HPA will cause the number of pods to scale up until it levels off at that target percentage or hits the pre-defined maximum.

In the default configuration shown below, when the average utilization goes above 60%, HPA scales up the number of pods to 2.

```
autoscaling:
  enabled: true
  cpu:
    enabled: true
    percentage: 60
  memory:
    enabled: false
    percentage: 60
  min_replicas: 1
  max_replicas: 2
```

If you want to configure app to scale up based on memory, here is a snippet you can use that will increase the number of pods to 2 when the average memory utilization is above 50%:

```
autoscaling:
  enabled: true
  cpu:
    enabled: false
    percentage: 60
  memory:
    enabled: true
    percentage: 50
  min_replicas: 1
  max_replicas: 2
```

Kubernetes Event Driven Autoscaler (KEDA)

While it's perfectly fine to use built-in `HorizontalPodAutoscaler` to scale workloads, a lot of times CPU or Memory will not be the best metric to base scaling decisions on. For instance, request count is a better metric for the a lot of HTTP API services. This is where KEDA comes into play as an easy way to integrate other types of metrics in HPA scaling. The best

part is that you can still use CPU or Memory to scale as a backup because Keda accepts multiple "triggers" for scaling decisions.

The epic-app base helm chart supports adding keda autoscalers. Here is an example PR adding it to egs-platform-service in the gamedev environment: https://github.ol.epicgames.net/online-web/egs-platform-service/pull/198

This is grabbing cloudwatch metrics from the loadbalancer itself. KEDA then takes that number and divides it by the current number of pods to determine what the current number of requests per pod is being served by the application. Once it exceeds the configured setting (the setting is per pod) it starts to scale up. Take note that this `ScaledObject` also has a CPU trigger. If that CPU threshold is met before the RequestCount threshold is met, it will still scale up. This will be nice in the case the cloudwatch metrics have an outage or they are unavailable for some other reason.

It's worth mentioning that under the hood, KEDA still creates an HPA. Because HPAs can use external metrics to scale, the KEDA workload becomes the external metrics API that the HPA calls to get current utilization metrics such as the current Request Count or CPU utilization. You can read more about external and custom metrics in Kubernetes here. To sum this up, KEDA becomes an interface to external metrics that makes it easier to implement usage of metrics from a variety of sources.

There is a central page linking to a variety of KEDA documentation to help as well.

KEDA allows you to activate a Kubernetes deployment (no pods to a single pod) and scale your deployment based on events from various event sources. KEDA has several types of scalers (memory, CPU, AWS CloudWatch, etc.) which can be configured to scale your application based on specific metrics from those scalers. You should use KEDA because for several HTTP API services, request count is a better metric to scale on instead of CPU or memory that HPA uses. Keda also accepts multiple "triggers" for scaling decisions so you can still use CPU or Memory to scale as backup.

epic-app allows you to use KEDA to scale your application. For example, in order to scale on the number of requests on Application Load Balancer, you can modify the KEDA `scaled_objects` section of the epic-app values.yaml file.

```
    cloudwatch-alb:
      scaleTargetRef:
        name: deployment-one
      minReplicaCount: 1
      maxReplicaCount: 50
      cooldownPeriod: 5
      idleReplicaCount: 0
      triggers:
      - type: aws-cloudwatch
        metadata:
          identityOwner: operator
          awsRegion: "us-east-1"
          namespace: AWS/ApplicationELB
          dimensionName: LoadBalancer
          dimensionValue: net/dead-dev-teleport-alb/<name of load balan
          metricName: RequestCount
          metricStat: "Sum"
          targetMetricValue: "1000"
          minMetricValue: "0"
```

Vertical Pod Autoscaling (VPA)

Lastly, the least used type of scaling is vertical pod autoscaling which doesn't create more pods to balance load, but instead increases the resources assigned to each pod. This is a good way to scale a `DaemonSet` if it's getting CPU throttled or OOMKilled. It's not safe to use VPA and HPA on the same workload as they will fight with one another.

The controller that manages Vertical Pod Autoscaling isn't included with Kubernetes by default, but worth covering here in case there are

situations where this makes sense for your team. You can read more about VPA on EKS here.

Note: if you are using AWS service specific scalers, you do not need to configure authentication for KEDA, but if you are using non-AWS scalers, you need to configure authentication using Vault. Additional information on how to do this is located https://confluence-epicgames.atlassian.net/wiki/spaces/CE/pages/93488835.
