Red Hat Summit

Connect

# KEDA + OpenShift = Custom Metrics Autoscaler

Patrik Plachý
patrik.plachy@redhat.com
Senior Solution Architect @ Red Hat CEE
Nov 2022

Red Hat | intel.

## Agenda

- What is KEDA?
- KEDA concepts
- Demo
- KEDA vs Knative

## What is KEDA?

- Project aims to make **K**ubernetes **E**vent **D**riven **A**utoscaling dead simple
- Started as a partnership between Red Hat and Microsoft (Feb 2019)
- Donated into CNCF as a Sandbox project (Mar 2020)
- KEDA 2.0 brought major redesign (Nov 2020)
- Promoted to **CNCF Incubation** project (Aug 2021)
- KEDA **2.8** has been recently released (Aug 2022)
- https://keda.sh

# KEDA users

Alibaba Cloud | AMPLIFY | Apache Airflow | ASTRONOMER

baseten | BoldDesk by Syncfusion | Chattermill | CISCO

cloudtrade | Coralogix | Docplanner Tech | FedEx

fission | freepikcompany | HM Courts & Tribunals Service | INTERCEPT EMPOWERING AZURE

KENDAXA | KPMG | Meltwater | METAMAZE Intelligent Document Processing

Microsoft | PENINSULAR RESEARCH OPERATION | PUREFACTS

qonto | reddit | Roadwork | Schréder HYPERION

SELDON | STILINGUE | SwissLife | Teamleader

workiva | zapier

Red Hat | intel

## KEDA concepts

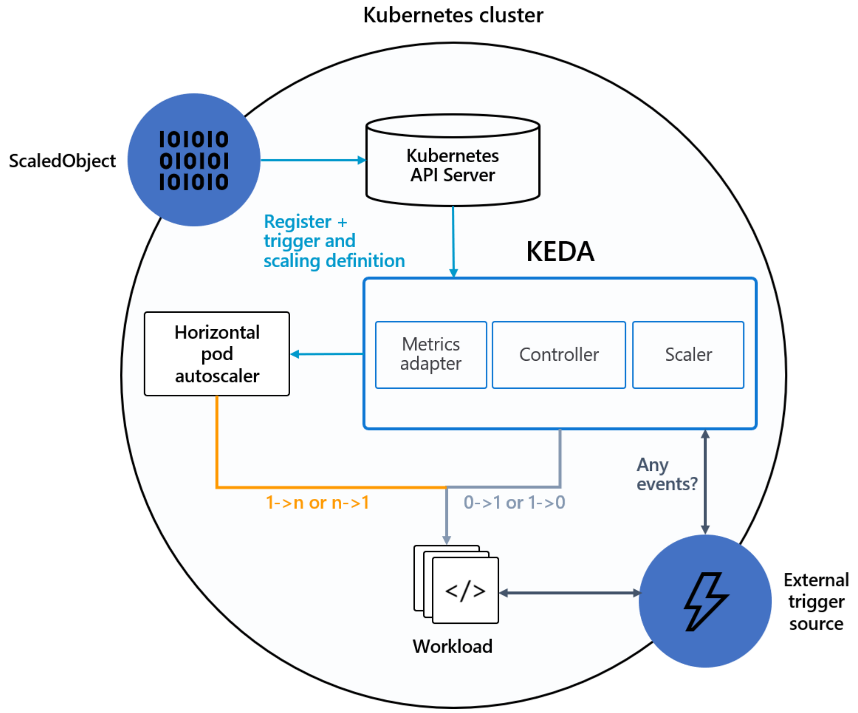
KEDA

- Automatically scale Kubernetes Deployments, Jobs & Custom Resources
- Provides **50+** built-in scalers, but users can build own external scalers
  - Kafka, Prometheus, RabbitMQ, AWS services, Azure Services,...
- Scale resources based on **events** in the target scalers, eg. messages in Kafka topic
- KEDA **does not** manipulate the data, just scales the workload
- Installation through OLM Operator or Helm

https://cloud.redhat.com/blog/custom-metrics-autoscaler-on-openshift

# KEDA concepts & architecture

- KEDA is built on top of Kubernetes
- Use ScaledObject/ScaledJob to define scaling metadata
- Manages workloads to scale to 0
- Registers itself as k8s Metric Adapter
- Provides metrics for Horizontal Pod Autoscaler (HPA) to scale on



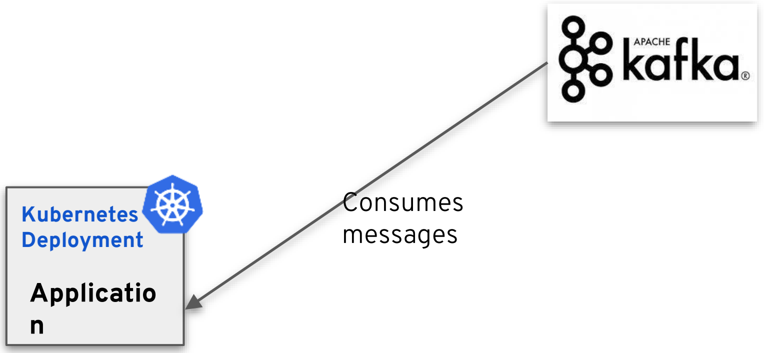
# ScaledObject

- Can target Deployment, StatefulSet or Custom Resource with /scale
- Multiple scalers can be defined as triggers for the target workload
- User can specify HPA related settings to tweak the scaling behavior

```yaml
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
 name: example-so
spec:
 scaleTargetRef:
          name: example-deployment
 minReplicaCount: 0
 maxReplicaCount: 100
 triggers:
 - type: kafka
   metadata:
          bootstrapServers: kafka.svc:9092
          consumerGroup: my-group
          topic: test-topic
          lagThreshold: '5'
```

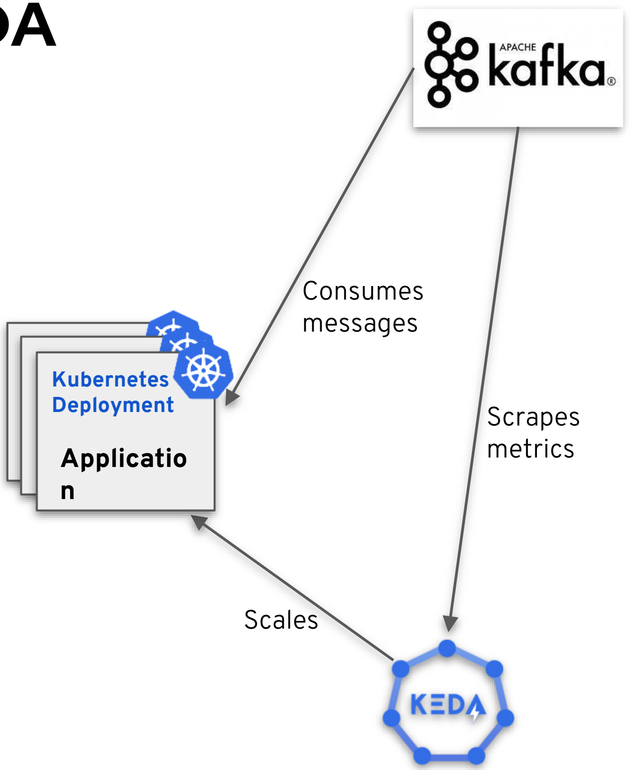
## **Example:**
## Application consuming messages from Kafka topic

- Application is deployed as standard Kubernetes Deployment
- Can be autoscaled only via standard k8s HPA: CPU & Memory
- No event-driven autoscaling



## Example redesigned to utilize **KEDA**

- Application remains the same and is being deployed the same way
- Event-driven autoscaling enabled through KEDA



## KEDA vs. Knative

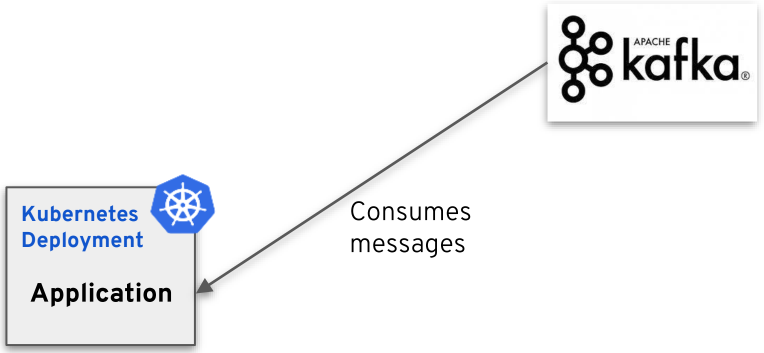
- Operates on standard k8s resources
- Can scale existing deployed apps
- Pull based approach
- Doesn’t manage data delivery
- K8s Horizontal Pod Autoscaler (HPA)
- Focus is on event driven autoscaling

- Operates on Knative Service
- Existing apps must be converted
- Push based approach
- Manages data delivery (Eventing)
- Knative Autoscaler
- Demand-based autoscaling (HTTP)

## Example:

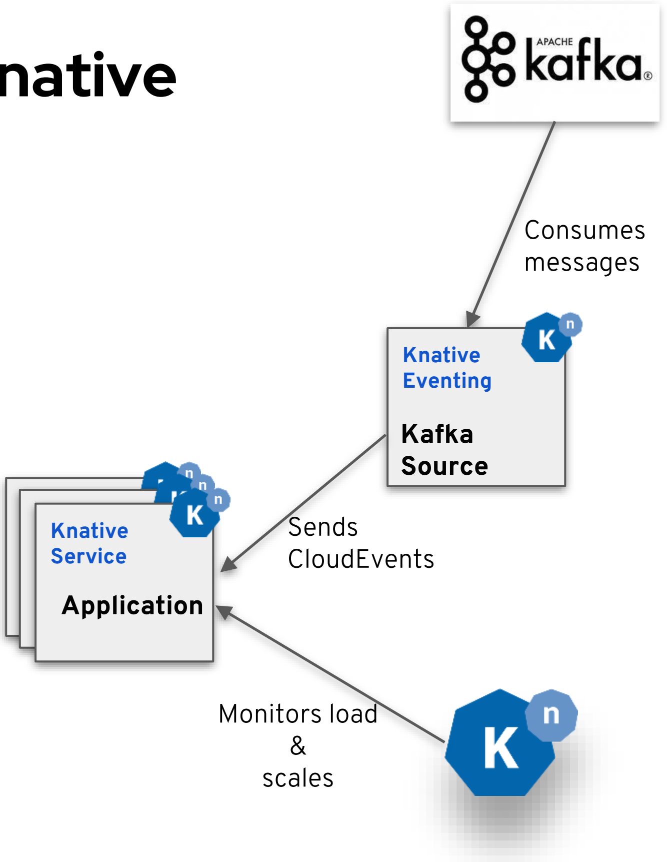
Application consuming messages from Kafka topic

- Application is deployed as standard Kubernetes Deployment
- Can be autoscaled only via standard k8s HPA: CPU & Memory
- No event-driven autoscaling



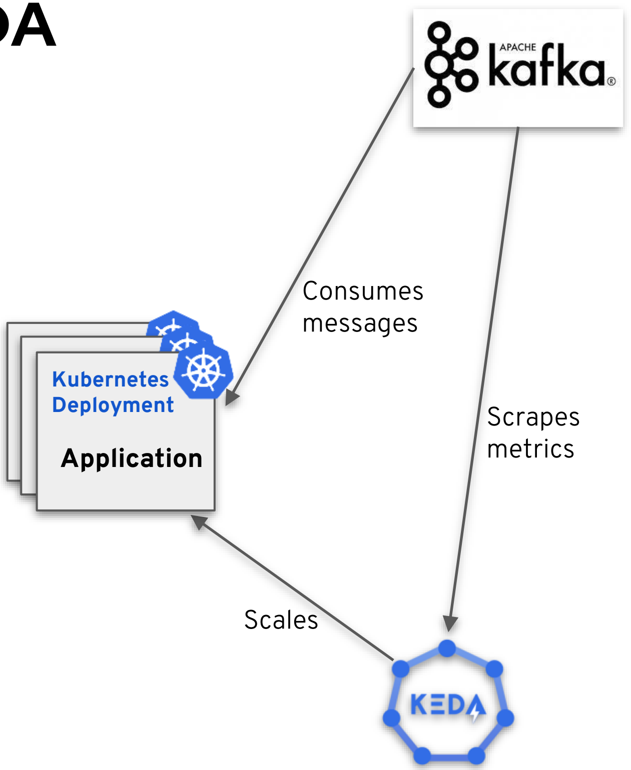
## Example redesigned to utilize **Knative**

- Application needs to be rewritten from Kafka consumer to CloudEvents consumer
- Application needs to be redeployed as Knative Service
- Needs Knative Eventing Kafka Source
- Event-driven autoscaling enabled through Knative Autoscaler



## Example redesigned to utilize **KEDA**

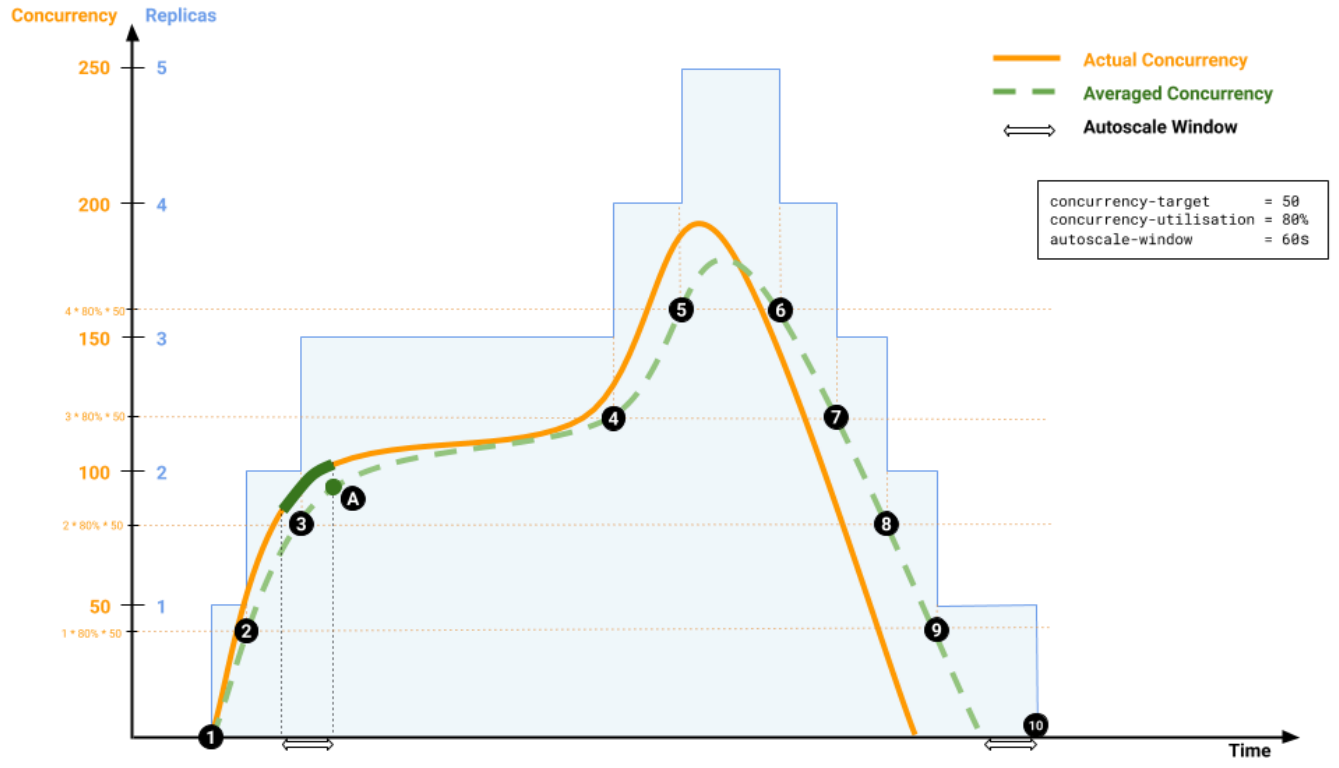
- Application remains the same and is being deployed the same way
- Event-driven autoscaling enabled through KEDA



# Knative Autoscaler Concepts

- Knative Autoscaler scales **Knative Service**, a CR representing the workload, it manages needed Kubernetes resources (Deployment, Service, Ingress,...)
- **Activator** component enables scale to 0
  - Incoming requests are being hold until the app is scaled to 1 replica
- Autoscaler itself has 3 components:
  - **PodAutoscaler Reconciler** – ensures that all components are up to date
  - **Collector** – collect metrics from various sources
  - **Decider** – based on metrics decides how the app should be scaled
    - `want = concurrencyInSystem/targetConcurrencyPerInstance`

## Knative Autoscaler



```
concurrency-target       = 50
concurrency-utilisation  = 80%
autoscale-window         = 60s
```

1. Scale up from 0 to 1 replica on first request.
2. Scale from 1 to 2 replicas if the utilisation 80% of the concurrency target 50 is reached for the **averaged concurrency**.

3 ... 9 Up- and downscaling events when **averaged concurrency** crosses the utilisation threshold counted across the current number of replicas. (2 * 80% * 50 = 80, 3 * 80 % * 50= 120, ….)

10. Scale down to 0 when **averaged concurrency** is going down to 0 for the length of the autoscale window.

A. The **averaged concurrency** is calculated every 2 seconds by averaging **concurrent requests** for the past auto-scale window length (default: 60s)

Red Hat Summit

Connect

# Thank you

linkedin.com/company/red-hat

facebook.com/redhatinc

youtube.com/user/RedHatVideos

twitter.com/RedHat

Red Hat | intel.