Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2021,8(6):63-69

Research Article ISSN: 2394-658X

Dynamic Resource Management in K8s: Leveraging Autoscaling and KEDA

Venkata Sasidhar (Sasi) Kanumuri

Email id - sasinrt@gmail.com

ABSTRACT

This article explores maximizing resource utilization and cost efficiency in Kubernetes installations by integrating autoscaling and Kubernetes Event-Driven Autoscaler (KEDA). This article presents powerful tools that can help maximize resource usage and cost-effectiveness in Kubernetes deployments. It explores the individual functionalities of each approach and demonstrates their synergistic potential through practical scenarios and best practices. The paper identifies critical advantages of KEDA compared to traditional metrics-based autoscaling, highlighting its ability to handle diverse event-driven workloads. Beyond traditional metrics-based scaling, KEDA emerges as a game-changer, enabling tailored scaling for diverse event-driven workloads. By integrating these methodologies, entities can attain a comprehensive resource management strategy, enabling scalability, performance, and cost-efficiency in dynamic cloud settings.

Key words: Autoscaling, KEDA, Kubernetes, Event-Driven Scaling, container Cost Optimization, Resource Management, Scalability, Performance, container autoscaling, HPA, VPA

INTRODUCTION

Kubernetes is used in containerized deployments to coordinate runtime and application delivery and enable horizontal scaling between nodes.

On the other hand, Kubernetes maintains effective resource utilization, which comes with complicated issues. Because containerized environments are inherently dynamic, unpredictable workloads might result in either over- or under-provisioning of resources. This equates to unnecessary spending or possible performance bottlenecks.

To address these concerns, autoscaling is a crucial strategy for dynamically orchestrating resource allocation. Autoscaling algorithms enable automated adjustments of pod replicas based on predefined metrics, ensuring performance consistency amidst fluctuating workloads.

Although Horizontal Pod Autoscalers (HPAs) are the cornerstone of Kubernetes autoscaling, their inability to handle heterogeneous workloads with various scaling triggers stems from their dependence on preset metrics. At this point, the powerful event-driven scaling solution provided by Kubernetes Event-Driven Autoscaler (KEDA) comes into play. By integrating with various event sources beyond traditional resource metrics, KEDA unlocks granular control over scaling behavior tailored to the specific requirements of event-driven applications.

KUBERNETES AUTOSCALING: EFFICIENT RESOURCE MANAGEMENT IN DYNAMIC ENVIRONMENTS

The dynamic nature of cloud computing requires using flexible and adaptive application infrastructure and integrating robust autoscaling techniques. Here's where Kubernetes enters the scene. It is a popular platform for container orchestration that makes managing containerized apps easier. However, to maximize resource use, efficient autoscaling strategies are required. Kubernetes Autoscaling offers a dynamic platform that can adapt to real-time changing needs, improving scalability, effectiveness, and affordability.

A. NEED FOR AUTOSCALING IN DYNAMIC CLOUD ENVIRONMENTS: AVOIDING BOTTLENECKS AND ENSURING BUSINESS CONTINUITY

One of the biggest challenges in cloud-native deployments is maintaining constant performance and business continuity in the face of varying demands. Static infrastructure frequently fails to keep up with traffic spikes caused by marketing campaigns, seasonal trends, or unanticipated events, which results in service interruptions and unhappy users. Autoscaling has become vital to contemporary cloud orchestration in handling these changing situations.

Traditional solutions involve manual scaling, either vertically by increasing the capacity of existing servers or horizontally by adding more servers to the fleet. While both options have their place, vertical scaling can become cumbersome with complex clusters like relational databases. More importantly, horizontal scaling offers distinct advantages:

[1]. **Load Distribution:** Horizontal scaling alleviates pressure on individual instances by distributing processing across multiple servers, ensuring smoother performance and responsiveness.

[2]. **High Availability and Fault Tolerance:** Introducing redundancy through additional servers enhances overall service availability, minimizing downtime risks in case of server failures.

These benefits become crucial as deployments scale, encompassing hundreds or thousands of servers and microservices. Manual scaling in such environments becomes untenable and error-prone. This is where autoscaling shines. Autoscaling platforms can adjust resource allocation in real-time by leveraging pre-defined utilization metrics such as CPU, memory, network traffic, or queue lengths. This dynamic scaling ensures:

[1]. **Seamless Response to Fluctuations:** Autoscaling smoothly adds resources during spikes in traffic, avoiding service interruptions and performance bottlenecks. On the other hand, it reduces during quiet times to maximize resource use and save expenses.

[2]. **Adaptive Infrastructure:** Autoscaling guarantees an appropriately sized infrastructure by continuously adjusting to fluctuating demands, supplying the resources required for peak performance without needless spending.

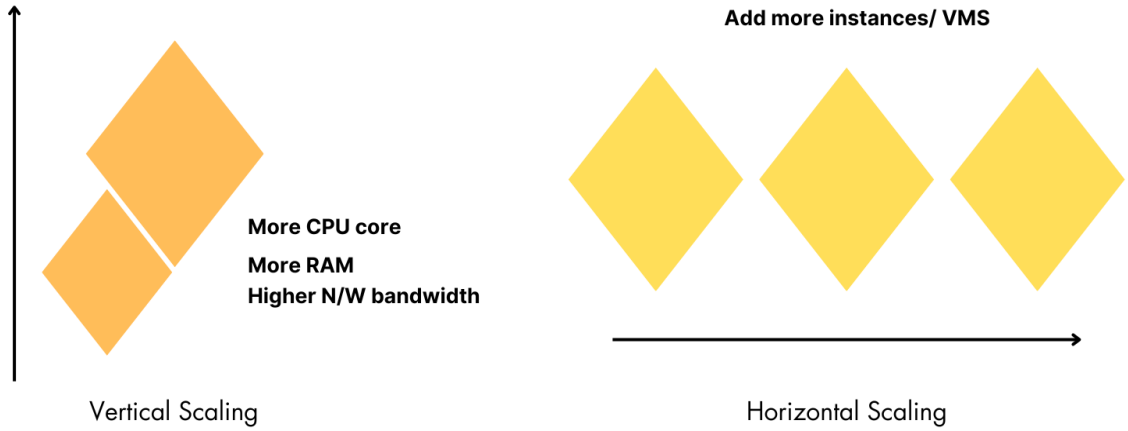

Figure 1

Autoscaling is no longer just a convenience in the dynamic and constantly changing world of cloud computing—it is now essential for modern, large-scale installations to achieve cost-efficiency, ensure business continuity, and guarantee predictable performance. Organizations may build a robust and responsive infrastructure that easily adjusts to changing demands by utilizing autoscaling, guaranteeing a great user experience and commercial success.

B. BENEFITS

Embracing Kubernetes Autoscaling unlocks a plethora of advantages for your cloud deployments:

[1]. **Cost Optimization:** Autoscaling dynamically adjusts the number of nodes or pods based on actual demand, eliminating unnecessary resource provisioning and minimizing idle resource costs.

[2]. **Performance Enhancement:** Continuous high performance for applications is ensured by automatically scaling up resources during peak loads and scaling down during idle periods. This guarantees that applications always have the necessary resources for optimal performance.

[3]. **Unparalleled Scalability:** Horizontal scaling, a key element of autoscaling, allows seamless addition or removal of resources as needed, ensuring that infrastructure adapts to changing demands.

[4]. **Operational Efficiency:** Autoscaling frees up significant resources for operations and development by automating the whole process, enabling them to focus on key business objectives.

C. AVAILABLE TOOLS

Kubernetes Autoscaling offers a spectrum of tools to cater to diverse scaling needs:

[1]. **Cluster Autoscaler:** This intelligent tool ensures efficient node utilization across the cluster by automatically scaling the cluster size up or down based on pod resource demands, preventing resource waste, and optimizing node usage.

Testing Cluster Autoscaler (CA)

Scale out:

We'll create a simple single replica pod with a service/ app and then ratchet up the number of replicas to a very high number to see the scale-out behavior. If the load/ # of replicas increases, HPA will create new replicas, for which there may or may not be enough space in the cluster. If there are not enough resources, CA will try to bring up some nodes so that the HPA-created pods have a place to run.

```
# CA triggers a scale-out event when there's a significant increase in # of pods to accommodate the current state needs.
$ kubectl scale deployment xyz-microservice –replicas=100

# Watch the pods running state with 100 replicas
$ kubectl get pods -o wide
# CA will create new nodes to accommodate 100 pods
```

Scale in:

If the load decreases, HPA will stop some of the replicas. As a result, some nodes may become underutilized or empty, and then CA will terminate such unneeded nodes.

```
# CA triggers a scale-in event when there's a significant decrease in # of pods to accommodate the current state needs.
$ kubectl scale deployment xyz-microservice –replicas=1

# Watch the pods running state with 1 replica
$ kubectl get pods -o wide
# CA will delete nodes to accommodate 1 pod.
```

[1]. Horizontal Pod Autoscaler (HPA): HPA focuses on individual microservices, monitors memory or CPU usage, and modifies the number of pod replicas in response, ensuring workloads always have the right resources for the required performance levels.

How does HPA scaling work?

Let's assume you have a deployment name xyz-microservice, and you want to scale it based on the CPU utilization (to be more specific, scale up or down on the # of pod replicas to maintain an average CPU utilization of 70%). You can use the command below to enable HPA on your xyz-microservice deployment:

```
$ kubectl autoscale deployment xyz-microservice --cpu-percent=70 --min=1 --max=8
```

The common question is, 'My pods are at max (8) now, but I still have more CPU utilization; how do I deal with this situation?'

HPA tries to create a pod at this point, but it might encounter an error like 'insufficient CPU.' To solve this, a pod that failed to be created triggers a cluster autoscale event, adding more nodes to the EKS cluster. When the load is down on microservice (i.e., CPU utilization < 50%), HPA will prune pods conservatively and optimize the # of pods; this helps with scaling and saves costs.

[1]. KEDA (Kubernetes Event-Driven Autoscaling): By going beyond conventional CPU/memory measurements, KEDA enables autoscaling based on events. It integrates with HPA, enabling scaling based on custom metrics like queue length or application events, offering greater flexibility and control.

Installation

KEDA offers helm chart-based installation.

```
$ helm repo add KEDAcore https://KEDAcore.github.io/charts
$ helm repo update
$ kubectl create namespace KEDA
$ helm install KEDA KEDAcore/KEDA --namespace KEDA
```

When you install KEDA, it creates four custom resources:

[1]. scaledobjects.KEDA.sh
[2]. scaledjobs.KEDA.sh

[3]. triggerauthentications.KEDA.sh
[4]. clustertriggerauthentications.KEDA.These custom resources enable you to map an event source (and the authentication to that event source) to a Deployment, StatefulSet, Custom Resource, or Job for scaling.

D. Optimization Strategies

Kubernetes Autoscaling has many advantages, but realizing its full potential requires careful application:

[1]. **Workload Understanding:** Accurate evaluation of application resource needs and consumption trends enables well-informed scaling decisions and autoscaling tool configuration.
[2]. **Gradual Implementation:** Begin with essential tools like Cluster Autoscaler and HPA. As familiarity grows, explore advanced options like KEDA for customized scaling scenarios.
[3]. **Continuous Monitoring:** Keep a close eye on the autoscaling setup's performance. Examine data, pinpoint opportunities for enhancement, and adjust settings to guarantee the best possible use of resources and program efficiency.
[4]. **Schedule Non-critical Workloads Off-peak**: Analyze your application usage patterns and consider scheduling non-critical workloads or batch jobs to run during off-peak hours when resource demand is lower. This can be achieved through various approaches, including K8s Cron Jobs or external schedulers. Scaling down resources during these periods reduces costs.

Adopting Kubernetes Autoscaling and strategically deploying its capabilities can help organizations unleash cloud infrastructure that dynamically adjusts to their demands and delivers agility, scalability, and cost-effectiveness for contemporary applications.

KEDA: UNVEILING EVENT-DRIVEN AUTOSCALING IN KUBERNETES

Traditional Horizontal Pod Autoscalers (HPAs) in Kubernetes primarily rely on predefined metrics like CPU or memory utilization to trigger scaling decisions. While effective for resource-bound applications, HPAs lack the versatility required for event-driven workloads with diverse scaling triggers. At this point, the Kubernetes ecosystem undergoes a paradigm change toward event-driven autoscaling thanks to the ground-breaking Kubernetes Event-Driven Autoscaler (KEDA).

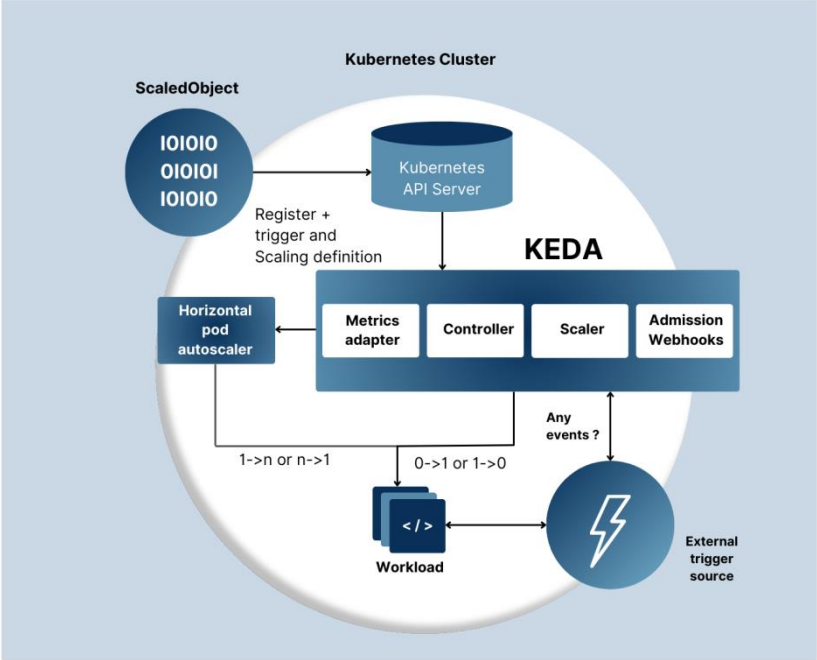

Figure 2: KEDA Architecture

A. KEDA: A FOUNDATION FOR DYNAMIC SCALING

At its core, KEDA distinguishes itself by leveraging **event sources** as the driving force behind autoscaling decisions. These event sources can encompass various elements, including:

[1]. **Message Queues:** KEDA seamlessly integrates with popular message queues like RabbitMQ, Apache Kafka, and Amazon SQS, dynamically scaling pods based on queue length or backlog size.

[2]. **Serverless Functions:** Applications utilizing serverless functions can leverage KEDA for autoscaling based on function invocations or concurrent executions.

[3]. **Custom Metrics:** KEDA empowers users to define custom event sources using Prometheus metrics or other data sources, enabling tailored scaling behavior for unique application requirements.

KEDA utilizes scalers to translate these events into scaling actions. These modular components act as interpreters, deciphering event data and translating it into concrete decisions regarding pod replica counts. KEDA boasts a rich library of built-in scalers for various event sources while also enabling the creation of custom scalers for specific needs.

Furthermore, KEDA introduces the concept of custom resources to configure and manage scaling behavior. These resources define the mapping between an event source, a scaler, and the target Kubernetes deployment or job to be scaled. This configuration-driven approach offers flexibility and scalability, allowing users to manage diverse autoscaling scenarios within a unified framework.

Scaling with KEDA Based on CPU Metrics

A rule can be set up to scale based on CPU utilization inside the container inside the pod. The following is an example manifest file that describes the scaling based on memory utilization with KEDA's scaled objects configuration.

```
apiVersion: KEDA.k8s.io/v1alpha1
kind: ScaledObject
metadata:
  name: {scaled-object-name}
spec:
  scaleTargetRef:
    Name: {deployment-name} # must be in the same namespace as the ScaledObject
    envSourceContainerName: {container-name}  #Optional. Default: deployment.spec.template.spec.containers[0]
  pollingInterval: 30  # Optional. Default: 30 seconds
  cooldownPeriod:  300 # Optional. Default: 300 seconds
  minReplicaCount: 1   # Optional. Default: 0
  maxReplicaCount: 100 # Optional. Default: 100
  fallback:    # Optional. Section to specify fallback options
    failureThreshold: 3    # Mandatory if fallback section is included
    replicas: 6      # Mandatory if fallback section is included
  advanced:   # Optional. Section to specify advanced options
    restoreToOriginalReplicaCount: true/false       # Optional. Default: false
    horizontalPodAutoscalerConfig:  # Optional. Section to specify HPA related options
      behavior:      # Optional. Use to modify HPA's scaling behavior
        scaleDown:
          stabilizationWindowSeconds: 300
          policies:
          - type: Percent
            value: 100
            periodSeconds: 15
  triggers:
  - type: CPU
    metadata:
      type: Utilization # Allowed types are 'Utilization' or 'AverageValue'
      value: "60"
```

B. ADVANTAGES OVER HPA

Compared to HPAs, KEDA presents several significant advantages:

[1]. Enhanced Flexibility: KEDA transcends resource-centric scaling, accommodating event-driven scenarios and various event source types.

[2]. Broader Workload Applicability: Microservices, serverless architectures, and event-driven systems are just a few of the workloads that KEDA can handle.

[3]. Rich Event Source Integration: KEDA makes integration easier and requires less development work by providing out-of-the-box support for various event sources.

C. UNLOCKING REAL-WORLD APPLICATIONS

KEDA empowers practitioners with diverse use cases:

[1]. Message Queue Offloading: Scale microservices consuming messages from queues based on queue length, ensuring timely processing and preventing backlogs.

[2]. Serverless Function Autoscaling: Dynamically adjust serverless function instances based on invocation rate, optimizing cost and performance.

[3]. CI/CD Pipeline Scaling: Adapt build and deployment resources based on pipeline events, improving CI/CD pipeline efficiency.

With KEDA, autoscaling in Kubernetes is revolutionized by switching from resource-centric to event-driven methods. Because of its adaptability, wide range of event source integration, and customization options, it's a priceless tool for growing and maintaining contemporary cloud-native applications. In the ever-changing world of containerized deployments, KEDA is positioned to be a key component in guaranteeing maximum performance and effective resource use as event-driven architectures gain traction.

ORCHESTRATING RESOURCE EFFICIENCY: SYNERGISTIC INTEGRATION OF AUTOSCALING, AUTOSTOPPING, AND KEDA

The intricate interplay of autoscaling, autostopping, and KEDA unveils a potent arsenal for achieving optimal resource utilization and cost efficiency in Kubernetes deployments. This section explores practical scenarios, best practices, and how KEDA complements existing techniques to manage complex workload patterns.

A. SCENARIOS AND BEST PRACTICES

[1]. Production Cluster Autoscaling: Leverage Cluster Autoscaler and Horizontal Pod Autoscalers (HPAs) for a layered approach. Cluster Autoscaler ensures node provisioning based on overall cluster demand, while HPAs fine-tune pod replicas within available nodes.

[2]. Dev/Staging Cluster Optimization: Implement HPA-based autoscaling alongside autostopping during non-business hours using KEDA cron triggers. This minimizes idle resource costs while maintaining availability for development activities.

[3]. Asynchronous microservice Workloads: Employ KEDA for Async microservices scaling based on custom metrics like the number of messages in the queue. Integrate with autostopping during low activity periods to optimize event processing efficiency.

B. KEDA: COMPLEMENTING AUTOSCALING AND AUTOSTOPPING

While HPA excels at resource-centric scaling and autostopping focuses on idle resource deactivation, KEDA bridges the gap by handling diverse event-driven scenarios:

[1]. Complex Workload Patterns: KEDA scales based on events like queue length or function invocations, exceeding the limitations of static resource metrics.

[2]. Granular Control: KEDA's custom scaler and event source capabilities enable tailored scaling behavior for unique workload requirements.

[3]. Event-Driven Autostopping: Combine KEDA with autostopping triggers to intelligently shut down event consumers during low activity periods, further optimizing resource utilization.

C. EXAMPLE: SCALING MICROSERVICES WITH AUTOSTOPPING

Consider a microservice consuming messages from a Kafka topic. Traditional HPA scaling based on CPU might not accurately capture actual processing needs. KEDA, empowered by a Kafka scaler, seamlessly scales pods based on queue length, ensuring timely message processing. During low activity periods, KEDA cron triggers initiate autostopping, deactivating idle pods and minimizing costs. This demonstrates how KEDA augments existing methods for efficient resource management in event-driven scenarios.

Organizations can unlock a paradigm shift in resource management by adopting a synergistic approach that integrates autoscaling, autostopping, and KEDA. This combined strategy empowers practitioners to achieve scalability, cost efficiency, and optimal performance within even the most complex and dynamic Kubernetes deployments. The future of resource management in cloud-native environments will undoubtedly witness KEDA playing a pivotal role in orchestrating efficient and automated scaling decisions, catering to the ever-evolving needs of modern applications.

CONCLUSION

Given the always-changing environment of cloud-native deployments, an adaptable and thoughtful approach to resource management is necessary. This paper has examined the remarkable synergy between autoscaling, autostopping, and KEDA, highlighting the advantages of each technique separately and how they work together to maximize resource use and reduce costs in Kubernetes environments.

A. KEY TAKEAWAYS:

[1]. Embrace Dynamic Scaling: Employ the flexibility of autoscaling tools like HPAs and KEDA to effectively cater to diverse workload patterns, ensuring performance consistency and cost optimization.

[2]. Unleash Event-Driven Control: KEDA empowers you to transcend resource-centric scaling by reacting to real-time events, unlocking tailored scaling behavior for complex and event-driven applications.

[3]. Synergy is Key: The true power lies in seamlessly integrating these tools. Combine autoscaling, autostopping, and KEDA based on your specific needs to achieve a holistic approach to resource management.

B. MOVING FORWARD:

[1]. Continuous Monitoring: Monitor how well your selected setup works, examine statistics, and make configuration adjustments to guarantee the best use of available resources.

[2]. Embrace Experimentation: Look through the always-changing selection of autoscaling and autostopping options to find the one that best suits your changing requirements.

[3]. Community Participation: Give back to open-source initiatives like KEDA, exchange insights, and influence Kubernetes's dynamic resource management development.

By embracing these strategies and fostering a culture of continuous improvement, you can unleash Kubernetes' full potential and empower your cloud-native applications to thrive in a dynamic and resource-conscious environment.

REFERENCES

[1]. "Kubernetes Event-driven Autoscaling." KEDA Documentation. [Online]. Available: https://keda.sh/ . Accessed May, 2021.

[2]. "Horizontal Pod Autoscaler", AWS Documentation. [Online]. Available: https://docs.aws.amazon.com/eks/latest/userguide/horizontal-pod-autoscaler.html, Accessed May, 2021.

[3]. Scaling Kubernetes: Intro to Kubernetes-based event-driven autoscaling (KEDA). May, 2020 [Online]. Available: https://cloudblogs.microsoft.com/opensource/2020/05/12/scaling-kubernetes-keda-intro-kubernetes-based-event-driven-autoscaling/

[4]. "Kubernetes Cluster Autoscaler", KubeCost. [Online]. Available: https://www.kubecost.com/kubernetes-autoscaling/kubernetes-cluster-autoscaler/, Accessed May, 2021.

[5]. "Kubernetes HPA", KubeCost. [Online]. Available: https://www.kubecost.com/kubernetes-autoscaling/kubernetes-hpa, Accessed May, 2021.