Kubernetes Autoscaling: 3 Methods and How to Make Them Great

What is Autoscaling in Kubernetes?

Kubernetes autoscaling is a feature that allows a cluster to automatically increase or decrease the number of nodes, or adjust pod resources, in response to demand. When demand increases, the cluster can add nodes or provide more resources to pods, and when demand decreases, Kubernetes can remove nodes or assign less resources to a pod. This can help optimize resource usage and costs, and also improve performance.

Three common solutions for scaling applications in Kubernetes environments are:

Horizontal Pod Autoscaler (HPA): Automatically adds or removes pod replicas.

Vertical Pod Autoscaler (VPA): Automatically adds or adjusts CPU and memory reservations for your pods.

Cluster Autoscaler: Automatically adds or removes nodes in a cluster based on all pods' requested resources.

This is part of our series of articles about Kubernetes.

In this article, you will learn:

1. Horizontal Pod Autoscaler (HPA)

Horizontal Pod Autoscaler (HPA) automatically adds or removes pod replicas. This makes it possible to automatically manage workload scaling when the level of application usage changes.

which is 15 seconds by default. The flag is: –horizontal-pod-autoscaler-sync-period

After each loop period, the controller manager compares actual resource utilization to the metrics defined for each HPA. It obtains these from either the custom metrics API or, if you specify that auto scaling should be based on resources per pod (such as CPU utilization), from the resource metrics API.

HPA makes use of metrics to determine auto scaling, as follows:

For resource metrics, you can either set a target utilization value or a fixed target.

For custom metrics, only raw values are supported, and you cannot define a target utilization.

For object metrics and external metrics, scaling is based on a single metric obtained from the object, which is compared to the target value to produce a utilization ratio.

Limitations

Avoid using HPA alongside vertical pod autoscaling (VPA) on memory or CPU when evaluating CPU or memory metrics. Additionally, when using a Deployment, you cannot configure HPA on a ReplicaSet or Replication Controller, only on the Deployment itself.

Learn more in our detailed guide to kubernetes replicaset.

Best Practices

Here are two key best practices for efficiently using HPA:

Ensure all pods have resource requests configured—when making scaling decisions, HPA uses the observed CPU utilization values of pods working as part of a Kubernetes controller. This is calculated as a percentage of resource requests made by individual pods. To ensure the data is accurate, you should use all resource request values of all containers.

Prefer custom metrics over external metrics when possible—the external metrics API represents a security risk because it can provide access to a large number of metrics. A custom metrics API presents less risk if compromised, because it only holds your specific metrics.

Use HPA together with Cluster Autoscaler—this allows you to coordinate scalability of pods with the behavior of nodes in the cluster. For example,

2. Vertical Pod Autoscaling (VPA)

The Vertical Pod Autoscaler automatically adds or adjusts CPU and memory reservations for your pods. It uses live usage data to set limits on container resources.

Most containers adhere more closely to their initial requests rather than to upper limit requests. As a result, Kubernetes' default scheduler overcommits a node's memory and CPU reservations. To deal with this, the VPA increases and decreases the requests made by pod containers to ensure actual usage is in line with available memory and CPU resources.

Some workloads can require short periods of high utilization. Increasing request limits by default would entail wasting unused resources, and would limit the nodes that can run those workloads. HPA may help with this in some cases, but in other cases, the application may not easily support distribution of load across multiple instances.

A VPA deployment calculates target values by monitoring resource utilization, using its recommender component. Its updater component evicts pods that must be updated with new resource limits. Finally, the VPA admission controller overwrites the pod resource requests when they are created, using a mutating admission webhook.

Limitations

Updating running pods is still experimental in VPA, and performance in large clusters remains untested. VPA reacts to most out-of-memory events, but not all, and the behavior of multiple VPA resources that match the same pod remains undefined. Finally, VPA recreates pods when updating pod resources, possibly on a different node. As a result, all running containers restart.

Best Practices

Here are two best practices for making efficient use of Vertical Pod Autoscaler:

Avoid using HPA and VPA in tandem—HPA and VPA are incompatible. Do not use both together for the same set of pods, unless you configure the HPA to use either custom or external metrics.

Use VPA together with Cluster Autoscaler—VPA might occasionally recommend resource request values which exceed available resources. This can result in resource pressure and cause pods to go into a pending status.

3. Cluster Autoscaler

The cluster autoscaler automatically adds or removes nodes in a cluster based on all pods' requested resources. It seeks unschedulable pods and tries to consolidate pods that are currently deployed on only a few nodes. It loops through these two tasks constantly.

Unschedulable pods are a result of inadequate memory or CPU resources, or inability to match an existing node due to the pod's taint tolerations (rules preventing a pod from scheduling on a specific node), affinity rules (rules encouraging a pod to schedule on a specific node), or nodeSelector labels. If a cluster contains unschedulable pods, the autoscaler checks managed node pools to see if adding a node may unblock the pod. If so, and the node pool can be enlarged, it adds a node to the pool.

The autoscaler also scans a managed pool's nodes for potential rescheduling of pods on other available cluster nodes. If it finds any, it evicts these and removes the node. When moving pods, the autoscaler takes pod priority and PodDisruptionBudgets into consideration.

When scaling down, the cluster autoscaler allows a 10-minute graceful termination duration before forcing a node termination. This allows time for rescheduling the node's pods to another node.

Limitations

Cluster autoscaler only supports certain managed Kubernetes platforms—if your platform is not supported, consider installing it yourself. Cluster autoscaler does not support local PersistentVolumes. You cannot scale up a size 0 node group for pods requiring ephemeral-storage when using local SSDs.

Best Practices

Here are two best practices for making efficient use of Cluster Autoscaler:

Ensure resource availability for the Cluster Autoscaler pod—you can do that by defining a minimum of one CPU for resource requests made to the cluster autoscaler pod. It is critical to ensure that the node running the cluster autoscaler pod has enough resources. Otherwise, the cluster autoscaler might become non responsive.

Ensure all pods have a defined resource request—to function correctly, the cluster autoscaler needs a specified resource request. This is because the

Other Kubernetes Scaling Mechanisms

There are other methods you can use to scale workloads in Kubernetes. Here are two common methods:

DaemonSets—used to deploy background services across all pods in a selected set. Learn more in our guide to Kubernetes Daemonset.

ReplicaSets—used to create a specified quantity of identical pods. Learn more in our guide to Kubernetes ReplicaSets.

Automating Kubernetes Infrastructure with Spot

For users that don't want to take this DIY approach to scaling cluster infrastructure, Spot's serverless container engine, Ocean, does all the work for you. Spot Ocean provides autoscaling that reads requirements of pending pods in real-time. This pod-driven autoscaling serves three goals:

Continuously check for unschedulable pods and find optimal nodes for it to run

Ensure that frequent scaling pods won't have to wait for new instances to launch

Ensure that there are no underutilized nodes in the cluster

With Cluster Autoscaler, users need to have a good understanding of their container needs. Spot Ocean, however, monitors events at the Kubernetes API server and dynamically allocates infrastructure based on container requirements (CPU, memory, networking) while honoring specific labels, taints, and tolerations. An out-of-the-box solution, Spot Ocean users don't have to configure or maintain individual scaling groups, nor are they restricted from using mixed instance types or multiple AZ's in a scaling group.

Learn More About Kubernetes Autoscaling

Understanding Kubernetes Cluster Autoscaler: Features, Limitations and Alternatives

There are different tools and mechanisms for scaling applications and provisioning resources in Kubernetes. Kubernetes's native horizontal and vertical pod autoscaling (HPA and VPA) handle scaling at the application level. However when it comes to the infrastructure layer, Kubernetes doesn't carry out

Read more: Understanding Kubernetes Cluster Autoscaler: Features, Limitations and Alternatives

Kubernetes Deployment: The Basics and 4 Useful Deployment Strategies

Kubernetes provides capabilities that help you efficiently orchestrate containerized applications. This is mainly achieved through the automation of provisioning processes. A Kubernetes Deployment enables you to automate the behavior of pods. Instead of manually maintaining the application lifecycle, you can use Deployments to define how behavior is automated. Learn how a Kubernetes Deployment works, considerations for using it, and four useful strategies including blue-green and canary deployment.

Learn more in our detailed guide to kubernetes deployment strategy.

Kubernetes StatefulSets: Scaling and Managing Persistent Applications

A Kubernetes StatefulSet is a workload API resource object. There are several built-in Kubernetes workload resources, each designed for certain purposes. StatefulSets are designed to help you efficiently manage stateful applications. Learn how Kubernetes StatefulSets can help you define, scale, and manage persistent applications on Kubernetes.

Learn more in our detailed guide to kubernetes statefulset.

Kubernetes ReplicaSet: Kubernetes Scalability Explained

A ReplicaSet (RS) is a Kubernetes object that ensures there is always a stable set of running pods for a specific workload. The ReplicaSet configuration defines a number of identical pods required, and if a pod is evicted or fails, creates more pods to compensate for the loss. Learn how Kubernetes ReplicaSets work, discover 3 types of Kubernetes replication, and see a quick tutorial on creating a deployment with a ReplicaSet.

Read more: Kubernetes ReplicaSet: Kubernetes Scalability Explained

Kubernetes Daemonset: A Practical Guide

DaemonSet is a Kubernetes feature that lets you run a Kubernetes pod on all cluster nodes that meet certain criteria. Every time a new node is added to a cluster, the pod is added to it, and when a node is removed from the cluster, the pod is removed. When a DaemonSet is deleted, Kubernetes removes all the pods created by it. Learn how DaemonSets work, how to perform common operations

Read more: Kubernetes DaemonSet: A Practical Guide

6 Kubernetes Deployment Strategies: Roll Out Like the Pros

A Kubernetes Deployment allows you to declaratively create pods and ReplicaSets. You can define a desired state, and a Deployment Controller continuously monitors the current state of the relevant resources, and deploys pods to match the desired state. It plays a central role in Kubernetes autoscaling. Understand how Kubernetes Deployment strategies, and discover advanced strategies like ramped slow rollout, controlled rollout, blue/green and canary deployments.

Learn more in our detailed guide to kubernetes deployment tools.

See Additional Guides on Key CI/CD Topics

Together with our content partners, we have authored in-depth guides on several other topics that can also be useful as you explore the world of Kubernetes.

CI/CD

Authored by Spot.io

- [Guide] What Is CI/CD? (Continuous Integration/Continuous Deployment)
- [Guide] CI/CD Tools: Key Features and 10 Tools You Should Know
- [Report] 2023 State of CloudOps
- [Product] Spot Ocean CD | Cloud Native Continuous Delivery

Kubernetes Architecture

Authored by Komodor

- What Are Kubernetes Services? The Ultimate Guide
- Kubernetes Dashboard: Quick Guide and 4 Great Alternatives
- Ultimate Guide to Kubernetes Operators and How to Create New Operators

Kubernetes Performance

Authored by Interl Tiber

- Kubernetes VPA: How It Works, HPA vs. VPA & 7 Best Practices
- Cluster Autoscaler Basics, Pros and Cons & 8 Best Practices

flexera | Spot

Related posts

Kubernetes StatefulSets: Scaling & Managing Persistent Apps

Run an ECS Cluster on Spot Instance in 3 Steps

Kubernetes Cluster Autoscaler: The Basics and a More Powerful Alternative

8 Kubernetes Deployment Strategies

Kubernetes Autoscaling: 3 Methods and How to Make Them Great

Kubernetes ReplicaSet: Kubernetes Scalability Explained

Why You Need Kubernetes Deployment Tools and 5 Tools to Know

Kubernetes Daemonset: A Practical Guide

Email us
info@flexera.com

Company Solutions

flexera | Spot

Partners

Commitment management

Cloud services for MSPs

Amazon Web Services

Microsoft Azure

Google Cloud Platforms

Products

Eco

CloudCheckr

Ocean

Ocean for Apache Spark

Elastigroup

Spot Security

Resources

Support

Documentation

Product News

Case Studies

Blog

Privacy Policy | Terms and Conditions