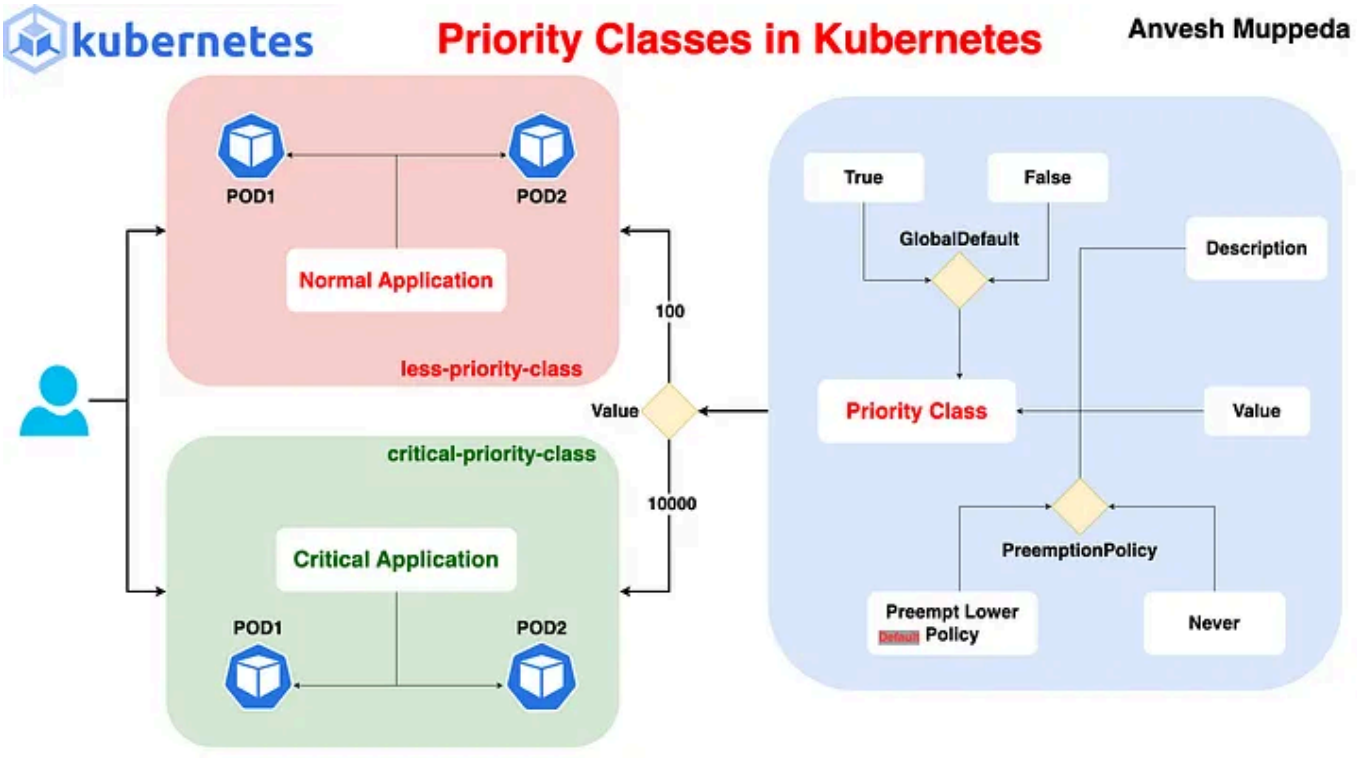
☸ A Hands-On Guide to Kubernetes Priority Classes 🛠️

⇢ PriorityClasses in Kubernetes with a practical example

Anvesh Muppeda Follow 6 min read · May 27, 2024

35 2



Kubernetes Pod Priority and Preemption by Anvesh Muppeda

Kubernetes is a powerful orchestration tool for containerized applications, but managing resource allocation and ensuring critical workloads get priority can be challenging. This is where Kubernetes

PriorityClasses come in. In this blog, we'll explore what PriorityClasses are, why they're important, and walk through a practical example to illustrate their use.

. . .

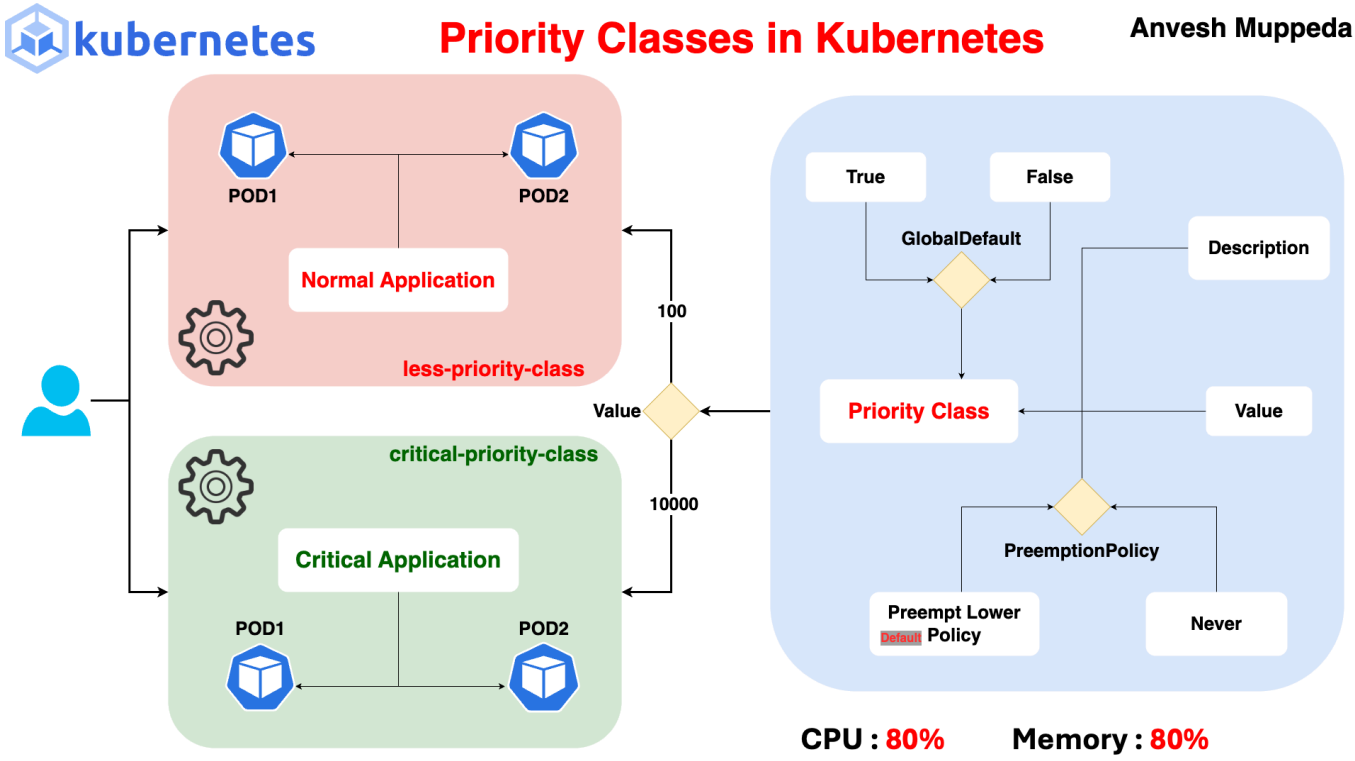
What are PriorityClasses?

Definition and Purpose

PriorityClasses in Kubernetes allow you to assign different priorities to your pods. Higher priority pods are scheduled first and, in cases of resource contention, can preempt (evict) lower priority pods. This ensures that critical applications continue running even under resource constraints.

Key Concepts

- **Priority**: A numerical value assigned to a `PriorityClass`. Higher numbers indicate higher priority.
- **Preemption**: The process of evicting lower priority pods to make room for higher priority ones.
- **Non-preempting PriorityClasses**: A type of PriorityClass that does not cause preemption.



Animated Kubernetes Pod Priority and Preemption by Anvesh Muppeda

. . .

Why Use PriorityClasses?

Use Cases

1. **Critical Workloads**: Ensure that critical applications (like monitoring or logging) are always running.
2. **Resource Optimization:** Efficiently manage and allocate resources under high load conditions.
3. **Quality of Service:** Maintain service levels for different applications by prioritizing more important ones.

. . .

PriorityClasses Template

```yaml
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: <priority-class-name>  # Name of the PriorityClass
value: <integer-value>  # Priority value (higher value means higher priority)
globalDefault: <true/false>  # Optional: Set to true if this PriorityClass shou
description: <description-text>  # Optional: Description of the PriorityClass
preemptionPolicy: <PreemptLowerPriority/PreemptNever>  # Optional: Whether Pods
```

Explanation of Fields

- `value`: An integer that represents the priority of the class. Higher values indicate higher priority.
- `globalDefault`: (Optional) A boolean that indicates whether this PriorityClass should be the default for all Pods that do not have a specified priority class. Only one PriorityClass can be set as the global default.
- `description`: (Optional) A string that describes the PriorityClass and its intended usage.
- `preemptionPolicy`: (Optional) Specifies whether Pods with lower priority can be preempted by Pods with this PriorityClass. Valid values are `PreemptLowerPriority` (default) and `Never`.

. . .

Practical Example

Scenario: Managing Resource Contention

Imagine you have a Kubernetes cluster running various workloads. During peak times, you need to ensure that your critical application (a logging service) remains operational while less critical applications (test environments, batch jobs) can be deprioritized.

Preparing Cluster for Testing

I am setting up a cluster for testing purposes. The cluster currently has a single node with a maximum of 1.5 GB of memory. I am testing with two applications: a critical application and a normal application.

Each application has 2 replicas, and each replica requires a minimum of `500 Mi` of memory. Therefore, a total of `2000 Mi` of memory is required. However, with the available memory, we can run a maximum of 2–3 pods, not more than that. As a result, the extra pods will need to remain in a waiting state.

Let's deploy the applications one by one and observe the behavior of the cluster.

Implementation Steps

1. Deploy Critical PriorityClass

Create a file named `high-priority-class.yaml` and add the following content:

```yaml
# PriorityClass for critical applications
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: critical-apps
value: 1000
globalDefault: false
preemptionPolicy: PreemptLowerPriority
description: "Priority class for critical applications."
```

Apply the manifest using `kubectl apply`

```
$ kubectl apply -f high-priority-class.yaml
```

2. Deploy Low PriorityClass

Create a file named `low-priority-class.yaml` and add the following content:

```yaml
# PriorityClass for normal applications
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: normal-apps
value: 100
globalDefault: false
preemptionPolicy: PreemptLowerPriority
description: "Priority class for normal applications."
```

Apply the manifest using `kubectl apply`:

```
$ kubectl apply -f low-priority-class.yaml
```

Now check the newly created priority classes using the below command:

```
$ kubectl get pc critical-apps normal-apps
NAME            VALUE   GLOBAL-DEFAULT   AGE
critical-apps   1000    false            41m
normal-apps     100     false            21m
```

3. Deploy Normal Application

Create a file named `normal-application.yaml` and add the following content:

```
# Deployment for normal application
apiVersion: apps/v1
kind: Deployment
metadata:
  name: normal-app
  labels:
    app: normal-application
spec:
  selector:
    matchLabels:
      app: normal-application
  replicas: 2
  template:
    metadata:
      labels:
        app: normal-application
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        resources:
          requests:
            cpu: 100m
```

```
            memory: 500Mi
          limits:
            cpu: 100m
            memory: 500Mi
      priorityClassName: normal-apps
```

Apply the manifest using `kubectl apply`:

```
$ kubectl apply -f normal-application.yaml
```

First, we will deploy the normal application to test whether our critical application can be deployed when there are not enough resources available.

```
$ kubectl get po
NAME                          READY   STATUS    RESTARTS   AGE
normal-app-5f75988cf6-cqqrf   1/1     Running   0          50s
normal-app-5f75988cf6-tbxhh   1/1     Running   0          50s
```

4. Deploy Critical Application

Create a file named `critical-application.yaml` and add the following content:

```
# Deployment for critical application
apiVersion: apps/v1
kind: Deployment
metadata:
  name: critical-app
  labels:
    app: critical-application
```

```
  spec:
    selector:
      matchLabels:
        app: critical-application
    replicas: 2
    template:
      metadata:
        labels:
          app: critical-application
      spec:
        containers:
        - name: nginx
          image: nginx:latest
          resources:
            requests:
              cpu: 100m
              memory: 500Mi
            limits:
              cpu: 100m
              memory: 500Mi
        priorityClassName: critical-apps
```

Apply the manifest using `kubectl apply`:

```
$ kubectl apply -f critical-application.yaml
```

Now we know that there are no available resources for our critical application replicas to run. Typically, without any priority classes, our critical application replicas would remain in a waiting state. However, since we have used a priority class, our critical application will start running by terminating the normal application replicas, which have a lower priority as defined earlier.

After deploying the critical application:

```
$ kubectl get po
NAME                              READY   STATUS    RESTARTS   AGE
critical-app-547594cfdd-8hjbg     0/1     Pending   0          2s
critical-app-547594cfdd-fjlbf     0/1     Pending   0          2s
normal-app-5f75988cf6-cqqrf       1/1     Running   0          5m9s
normal-app-5f75988cf6-stwb5       0/1     Pending   0          2s
```

As a result of deploying the critical application, one of the normal application replicas was terminated(preempted) and moved to a pending state. This occurred to free up space for the critical application.

Final status of the applications:

```
$ kubectl get po
NAME                              READY   STATUS    RESTARTS   AGE
critical-app-547594cfdd-8hjbg     1/1     Running   0          86s
critical-app-547594cfdd-fjlbf     1/1     Running   0          86s
normal-app-5f75988cf6-cqqrf       1/1     Running   0          6m33s
normal-app-5f75988cf6-stwb5       0/1     Pending   0          86s
```

Even though the normal application was running first, the critical application terminated the normal application replicas due to the priority classes.

That's it, we have successfully we implemented `PriorityClasses`.

. . .

Conclusion

By using `PriorityClasses` in Kubernetes, you can ensure that critical workloads receive the resources they need even in times of contention. This feature provides a powerful mechanism for maintaining the stability and reliability of essential applications.

. . .

Key Takeaways

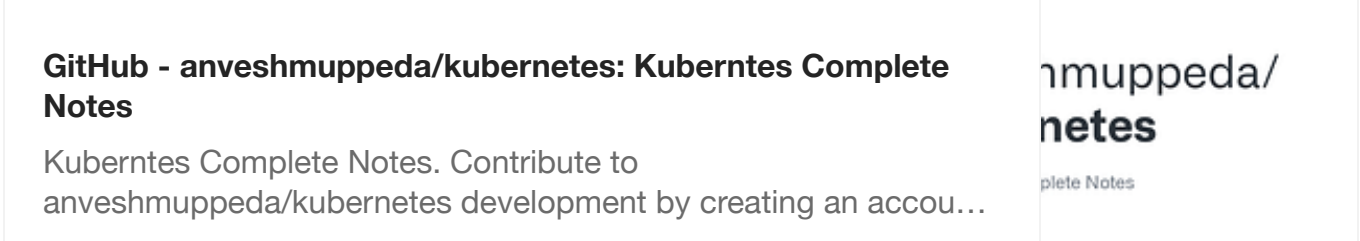
1. **Priority Management:** Efficiently allocate resources using priority values.
2. **Resilient Operations:** Ensure critical applications remain operational under resource constraints.
3. **Flexible Configuration:** Customize priorities according to application needs.

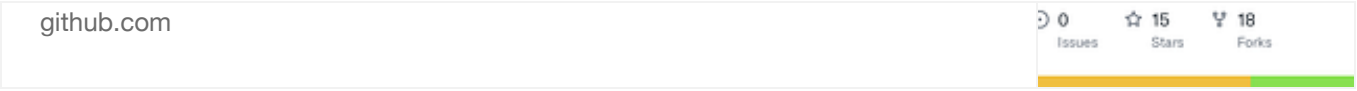
By following this guide, you should have a solid understanding of Kubernetes `PriorityClasses` and how to implement them in your cluster. Happy scheduling!

. . .

Source Code

You're invited to explore our GitHub repository, which houses a comprehensive collection of source code for Kubernetes.





Also, if we welcome your feedback and suggestions! If you encounter any issues or have ideas for improvements, please open an issue on our GitHub repository. 🚀

. . .

Connect With Me

If you found this blog insightful and are eager to delve deeper into topics like AWS, cloud strategies, Kubernetes, or anything related, I'm excited to connect with you on LinkedIn. Let's spark meaningful conversations, share insights, and explore the vast realm of cloud computing together.

Feel free to reach out, share your thoughts, or ask any questions. I look forward to connecting and growing together in this dynamic field!

Happy deploying! 🚀

Happy Kubernetings! ⎈

Kubernetes K8s Kubernetes Cluster Priorityclass DevOps

Written by Anvesh Muppeda

1.2K followers · 11 following

Follow

I'm a Kubernetes developer and cloud architect Certifications: 3x AWS | 2x Kubernetes Connect with me on www.linkedin.com/in/anveshmuppeda