**You want your pods (springboot-app) to be spread evenly across nodes in your Kubernetes cluster, instead of being scheduled unevenly (like 3 pods on one node and 1 on another).**

**🧠 Why This Happens**

By default, the Kubernetes scheduler does not guarantee even spreading across nodes. It looks at:

* Resource availability
* Node affinity/taints
* And other scheduling constraints...

Unless you explicitly instruct it to spread pods, it might schedule multiple pods on the same node.

**✅ Solution: topologySpreadConstraints**

Kubernetes offers a feature called **topology spread constraints** which lets you control how pods should be **distributed across "topologies"** (like nodes or zones).

apiVersion: apps/v1

kind: Deployment

metadata:

name: springboot-app

labels:

app: springboot-app

spec:

replicas: 4 # Updated replicas for high availability

selector:

matchLabels:

app: springboot-app

template:

metadata:

labels:

app: springboot-app

spec:

topologySpreadConstraints:

- maxSkew: 1

topologyKey: kubernetes.io/hostname

whenUnsatisfiable: ScheduleAnyway

labelSelector:

matchLabels:

app: springboot-app

containers:

- name: springboot-app

image: sumanth17121988/appjava:1

ports:

- name: metrics-port # Named port for Prometheus

containerPort: 8881 # Application container port

A screenshot of a chat

AI-generated content may be incorrect.

You’re running a **Spring Boot app** in Kubernetes and want to ensure:

1. Pods are **not bunched up on one node** (for high availability)
2. Some pods **stay running** during maintenance like upgrades or draining

**🧠 What is Pod Disruption Budget (PDB)?**

A **PDB** protects your application by telling Kubernetes:

❗ “Don’t evict too many pods of this app at the same time.”

**Pod Disruption Budget (PDB) – "Don't evict too many pods at once"**

Kubernetes upgrades or node maintenance might **evict pods**.

PDB is like a safety guard: “Hey! At least **4 pods** should always stay up!”

**🤔 Why do you need a PDB?**

During:

* ✅ Node upgrades
* ✅ Node draining (e.g. scaling down)
* ✅ Manual evictions

Kubernetes **may remove pods temporarily**.

If too many pods go down together, your app could:

* Crash
* Get overwhelmed by traffic
* Fail health checks
* **🛡️ What does a PDB protect from?**

| **Disruption Type** | **Protected by PDB?** |
| --- | --- |
| Node drain | ✅ Yes |
| Node upgrade | ✅ Yes |
| Manual eviction | ✅ Yes |
| Pod crash (self-inflicted) | ❌ No — that’s not voluntary |

**🎯 How does it work?**

You can define:

* minAvailable: Minimum number of pods that **must stay running**
* or
* maxUnavailable: Maximum number of pods that **can be taken down at once**

📝 **Choose one — not both!**

**✅ Real Example**

Let’s say you have:

* 6 replicas of a Spring Boot app
* You want at least 4 pods always running

apiVersion: policy/v1

kind: PodDisruptionBudget

metadata:

name: springboot-app-pdb

spec:

minAvailable: 2

selector:

matchLabels:

app: springboot-app

Kubernetes will now **refuse to evict more than 2 pods**.

**kubectl get pdb**

kubectl drain aks-appnode-50911972-vmss000002 --ignore-daemonsets --delete-emptydir-data

kubectl uncordon aks-appnode-50911972-vmss000002

kubectl get pods -o wide -l app=springboot-app

Confirm that the 15 pods are **evenly spread across nodes** with env=prod.

kubectl get pods -o wide -l app=springboot-app

**🧠 Outcome:**

* If the node has ≤13 pods, **drain will work**
* If the node has >13 pods, you’ll see

 During drain, K8s starts evicting pods **one by one**.

 After 5 pods are evicted, only **10 are still available** (meets PDB).

 Trying to evict the 6th pod would **drop availability to 9** → ❌ PDB violation.

 Kubernetes retries eviction **every few seconds**.

 Meanwhile, earlier-evicted pods get **rescheduled and become Ready** on other nodes.

 Now, **available pod count is again ≥ 10** → ✅ K8s continues eviction.

A screenshot of a white box

AI-generated content may be incorrect.