**Imagine you have a Kubernetes cluster with multiple node pools, each serving different purposes — like:**

* **prod apps on high-performance nodes**
* **dev apps on cost-effective nodes**
* **east and west regional workloads**
* **Separate nodes for Python, Java, or AI/ML workloads**

**You want control over where pods land, and Node Affinity gives you that.**

✅ What is Node Affinity?

Node Affinity lets you tell the Kubernetes scheduler:

*"I want this pod to run only on nodes that have specific labels"*

Node Affinity lets you constrain which nodes your pods are eligible to be scheduled on based on labels.

kubectl label node aks-workerpool-47159840-vmss000002 env=prod region=west

kubectl label node aks-workerpool-47159840-vmss000003 env=dev region=east

Think of it like saying:

*"Only run my app on nodes that are SSD-backed" or "Only use GPU nodes"*.

**How Does Kubernetes Decide Where to Schedule Pods?**

**The Kubernetes Scheduler goes through several steps to decide where a pod should land.**

**1. Filter (Predicates Phase)**

**The scheduler removes nodes that don’t match requirements like:**

✅ Enough CPU/memory  
✅ NodeSelector / Node Affinity  
✅ Tolerates taints  
✅ Required topologySpreadConstraints

ML Model –(nodeAffinity – required : node-type=ml)

Java batch – 100GB

Node 1 : GPU -🡪 ML Model ( node-type=ml)

Node 2: java app ( 120GB, 20 CORE) (node-ty= batch)

Node 3 : Python app( 100core, 20GB RAM)

Node Affinity

* Required (Hard limit)
* Optional (Soft limit)

kubectl get nodes --show-labels

**kubectl label nodes gke-gke-cluster-default-pool-226c9a8a-41g7 node-type=javapp env=prod**

kubectl label nodes gke-gke-cluster-default-pool-226c9a8a-mnbk node-type=pythonapp env=pre-prod

apiVersion: apps/v1

kind: Deployment

metadata:

name: springboot-app

labels:

app: springboot-app

spec:

replicas: 2

selector:

matchLabels:

app: springboot-app

template:

metadata:

labels:

app: springboot-app

spec:

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: env

operator: In

values:

- prod

containers:

- name: springboot-app

image: sumanth17121988/appjava:1

ports:

- name: metrics-port

containerPort: 8881

**Breakdown:**

* requiredDuringSchedulingIgnoredDuringExecution: Pod won’t be scheduled unless the rule is satisfied.

matchExpressions:

* key: must match a label key on the node.
* operator: like In, NotIn, Exists, etc.
* values: values to match for that key.

====-=

✅ 1. **Required (Hard) Node Affinity** – Must run on env=prod

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: env

operator: In

values:

- prod

🧠 **Use case**: Critical production apps must only run on trusted nodes.

**🧠 Explanation:**

* **requiredDuringSchedulingIgnoredDuringExecution** = hard rule.
* Pod will **only be scheduled** on nodes where env=prod.
* If such nodes are not available, **pod stays pending**.

✅ 2. **Multiple Match Conditions** – Must run on nodes with env=prod AND region=west

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: env

operator: In

values:

- prod

- key: region

operator: In

values:

- west

You want apps to be deployed **only** on nodes that are:

* Production (env=prod)
* In West region (region=west)

✅ 3. **✅ 3. Preferred Node Affinity: Pod should prefer but not forced**

affinity:

nodeAffinity:

preferredDuringSchedulingIgnoredDuringExecution:

- weight: 100

preference:

matchExpressions:

- key: region

operator: In

values:

- west

✅ 4. **Avoid Certain Nodes**: Don’t schedule on env=dev nodes

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: env

operator: NotIn

values:

- dev

Alright! Let’s break down **Taints and Tolerations in Kubernetes** using a **"house lock and key" analogy** and then look at an example using a **Java application Docker image**.

👉 **Taints are used to *protect* a node (or group of nodes) from getting *unwanted pods*.**  
✅ You use them to **block everything by default**, and only allow **specific, trusted pods** using tolerations.

Even if you taint a node (e.g., app=javapp:NoSchedule) and use toleration in the pod, Kubernetes is NOT guaranteed to schedule the pod on that tainted node.

**🔥 Example: You have a GPU node pool for ML workloads**

kubectl taint nodes gpu-node-pool gpu=true:NoSchedule

This means:

❌ "Don’t allow **any pod**, unless it explicitly tolerates gpu=true:NoSchedule."

A screenshot of a computer program

AI-generated content may be incorrect.

**🔥 Real-World Analogy**

You have a VIP lounge (node), and it has a **“VIP only”** sign (taint).  
Only people with a **VIP pass** (toleration) can enter.

**🌟 Analogy: House Lock & Key**

* **Imagine each house (Node)** in your neighborhood is like a **Kubernetes Node**.
* Sometimes, a house (Node) wants to **allow only specific guests (Pods)**.
* So, the owner of the house **puts a special lock on the door** (this is called a **Taint**).
* Now, only guests with a **matching key** (this is called a **Toleration**) can enter that house.

So:

* **Taint** = Lock (placed on a Node to repel Pods unless they have a matching key)
* **Toleration** = Key (attached to a Pod, allowing it to enter Nodes with the matching lock)

**💡 Why Taints & Tolerations?**

You use **Taints** to **protect nodes from running any random Pods**. Maybe some nodes have high memory and are only meant for Java applications, or maybe they're GPU nodes reserved for AI tasks. So, you taint those nodes and only allow specific pods (with tolerations) to be scheduled there.

**🛑 What Does “Tainting a Node” Mean in Kubernetes?**

A **taint** is like a **warning sign or rule** that you apply to a node to say:

“⚠️ Don’t schedule any pod on me unless the pod is **explicitly allowed**!”

In technical terms:

* A **taint** **repels pods** from being scheduled on the node.
* Only pods with a matching **toleration** are allowed.

**🎯 Why Do We Need Tolerations?**

Use cases:

* Run **special workloads** only on **special nodes** (e.g., GPU, high-memory, Spot nodes)
* Prevent regular pods from being scheduled on **critical** or **unstable** nodes
* Implement **workload separation** (e.g., production vs development)

**🛠️ Real-World Example: Java App with Docker Image**

Let’s say you have a node reserved for **Java applications only** (e.g., javapp: true). You want to run a Docker container that runs a Java application

**👨‍🔧 Step 1: Taint the Node (Node Level Restriction)**

You taint the node to **repel all Pods unless they tolerate this taint**:

kubectl taint nodes node1 app=javapp:NoSchedule

kubectl taint nodes <node-name> key=value:effect

This says:

This node now has a **lock** (app=javapp) with effect NoSchedule

"Hey node1, don’t allow any Pod to run here **unless** it has a toleration for app=javapp."

**📦 Step 2: Create Pod with JavaApp Docker Image and Toleration**

**Tolerations allow a pod to "tolerate" the taint and be scheduled on nodes with it.**

Here’s a simple Pod YAML that runs a Java app using a Docker image and has the **correct toleration** (key to the lock):

apiVersion: v1

kind: Pod

metadata:

name: java-app

spec:

containers:

- name: java-container

image: myrepo/javapp:latest

ports:

- containerPort: 8080

tolerations:

- key: "app"

operator: "Equal"

value: "javapp"

effect: "NoSchedule"

**✅ Use Cases**

* Reserve nodes for certain apps (Java, Python, GPU, memory-heavy).
* Keep critical apps away from regular nodes.
* Avoid accidental deployment to sensitive nodes.

**✅ When to Use Taints & Tolerations?**

* You have a **special-purpose node** (high memory, GPU, security-sensitive)
* You want to **block everything except specific workloads**
* Great for **multi-tenant clusters**, production nodes, GPU machines, or JVM-heavy nodes

**📝 Step 2: Pod YAML with Both Node Affinity & Toleration**

apiVersion: v1

kind: Pod

metadata:

name: nginx-prod-taint-affinity

spec:

containers:

- name: nginx

image: nginx

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: kubernetes.azure.com/agentpool

operator: In

values:

- prod

tolerations:

- key: "env"

operator: "Equal"

value: "prod"

effect: "NoSchedule"

❌ What if You Remove tolerations?

apiVersion: v1

kind: Pod

metadata:

name: nginx-no-toleration

spec:

containers:

- name: nginx

image: nginx

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: kubernetes.azure.com/agentpool

operator: In

values:

- prod

kubectl apply -f nginx-no-toleration.yaml

Expected outcome:  
❌ Pod stays in Pending due to taint.

**❓What if you remove nodeAffinity and only keep toleration?**

**✅ Result: The pod can be scheduled on any node in the cluster — tainted or untainted — but only if it tolerates the taint on the tainted node.**

apiVersion: v1

kind: Pod

metadata:

name: nginx-toleration-only

spec:

containers:

- name: nginx

image: nginx

tolerations:

- key: "env"

operator: "Equal"

value: "prod"

effect: "NoSchedule"

**🔎 Observations:**

* If an **untainted node exists** (like appnode), the pod may land **there**, not on prod.
* If **only prod nodes are tainted**, this pod **will be scheduled on prod** because it tolerates the taint.
* You **cannot guarantee** pod will run on prod node pool **unless** you **also add nodeAffinity**.

**Think of Taints as Node-Level Firewalls**

**🧲 Node Affinity – Complete Breakdown (Pod Controls Scheduling)**

Node affinity is the **opposite direction**.

* Instead of nodes saying, “Don’t run here” (like a taint),
* **Pods say**, “I want to run **only on specific nodes**” (based on node labels)

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**🧪 Putting it Together – Real Scenario with Java Docker Image**

**🧱 Step 1: Label and Taint the Node**

**kubectl label node node1 app=javapp**

**kubectl taint node node1 app=javapp:NoSchedule**

**This does two things:**

* **Adds a label to the node (for affinity)**
* **Adds a taint to the node (for repelling Pods)**

**📝 Step 2: Pod YAML with Both Node Affinity & Toleration**

apiVersion: v1

kind: Pod

metadata:

name: nginx-prod-taint-affinity

spec:

containers:

- name: nginx

image: nginx

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: kubernetes.azure.com/agentpool

operator: In

values:

- prod

tolerations:

- key: "env"

operator: "Equal"

value: "prod"

effect: "NoSchedule"

**🟢 This Pod:**

* **Says "I want to run only on nodes labeled app=javapp" → Node Affinity**
* **And "I am allowed to run there even if there's a taint" → Toleration**

Using **both taints/tolerations and node affinity** together is common in real-world production setups, especially when you want to **strictly control scheduling** — i.e., decide both **where a pod *wants* to go** and **where it’s *allowed* to go**.

Let’s break this down with **real-world use cases and examples**, and then show when using **both together** is the best choice.

✅ **Why Use Both Taints/Tolerations and Node Affinity?**

A screenshot of a computer

AI-generated content may be incorrect.

**🧰 Real-Time Use Cases – When to Use Both**

**🧠 Use Case 1: Dedicated JVM Nodes for Java Apps (High Memory)**

**🎯 Problem:**

**You have some nodes optimized for high-memory Java workloads (javapp) — JVM apps cause high memory and GC usage.**

**You want:**

* **Only Java apps to run there**
* **Other pods (Python, Go) to be kept out**
* **Java pods to request those nodes and be allowed there**

**🔧 Solution: Use Node Affinity + Taint/Toleration**

**💻 Node Setup:**

**kubectl label node node-jvm app=javapp**

**kubectl taint node node-jvm app=javapp:NoSchedule**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: javapp**

**spec:**

**containers:**

**- name: app**

**image: myrepo/javapp:latest**

**affinity:**

**nodeAffinity:**

**requiredDuringSchedulingIgnoredDuringExecution:**

**nodeSelectorTerms:**

**- matchExpressions:**

**- key: "app"**

**operator: "In"**

**values:**

**- javapp**

**tolerations:**

**- key: "app"**

**operator: "Equal"**

**value: "javapp"**

**effect: "NoSchedule"**

**✅ Result:**

* **Pod will *only* go to node labeled app=javapp (Node Affinity)**
* **Pod will be *allowed* on that node despite the taint (Toleration)**

**🧠 Use Case 3: Production vs Non-Production Isolation**

**🎯 Problem:**

* **You have separate nodes for production (env=prod) and development (env=dev)**
* **You want:**
  + **Only prod workloads to run on prod nodes**
  + **No test/dev workloads sneak into prod**

**🧩 Solution:**

**Node Setup:**

**kubectl label node node-prod env=prod**

**kubectl taint node node-prod env=prod:NoSchedule**

apiVersion: apps/v1

kind: Deployment

metadata:

name: javapp-deployment

labels:

app: javapp

spec:

replicas: 2

selector:

matchLabels:

app: javapp

template:

metadata:

labels:

app: javapp

spec:

containers:

- name: javapp

image: myrepo/javapp:latest # Replace with your actual Docker image

ports:

- containerPort: 8080

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: "env"

operator: "In"

values:

- prod

tolerations:

- key: "env"

operator: "Equal"

value: "prod"

effect: "NoSchedule"

**kubectl get pods -o wide**

**✅ Why both?**

* **Dev workloads won’t accidentally land on production nodes**
* **Prod workloads are explicitly granted access and forced to run only on prod nodes**