Ques 6 : Configure Taints and Tolerants.

SOLN:

Taint (Definition):

A **taint** is a property that you apply to a Kubernetes node to **prevent pods from being scheduled** onto it unless those pods explicitly **tolerate** the taint.

Toleration (Definition):

A **toleration** is applied to a pod to **allow** (or tolerate) the scheduling of the pod onto a node that has a **matching taint**.

Taints and tolerations work together to ensure that pods are only scheduled onto appropriate nodes. Taints act as a **barrier**, and tolerations **grant exceptions** to that barrier.

Layman Explanation:

Imagine a **restaurant (your Kubernetes cluster)** with many **tables (nodes)**.

- Some tables have **"Reserved" signs (taints)** saying:
  "Only VIPs (special pods) can sit here."
- A regular customer (normal pod) walks in.
  He sees the "Reserved" sign and **doesn't sit there**.
- A VIP customer (pod with toleration) walks in.
  He **has permission (toleration)** and can sit at the reserved table.

So:

- **Taint = puts a restriction on a node.**
- **Toleration = allows specific pods to ignore that restriction and get scheduled on that node.**

Use Case in Real Life:

Suppose you have:

- **GPU Nodes**: These are expensive and meant only for ML workloads.
- You **taint** them to prevent general pods from using them.

- You **add toleration** to ML pods so they can go to those nodes.

Concept Diagram :

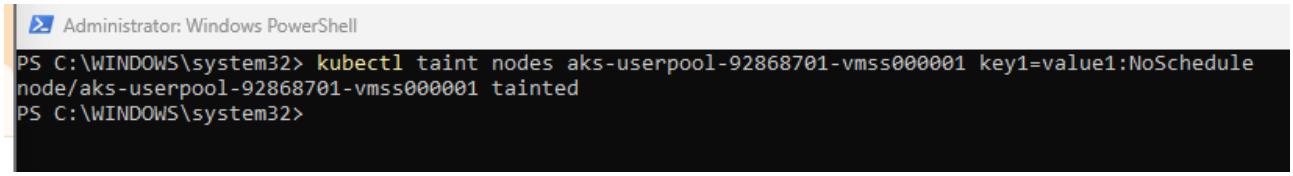
```
[NODE A - Tainted: key=gpu:NoSchedule]
      |
      |--> POD 1 [No toleration] ❌ Will not schedule
      |
      |--> POD 2 [Tolerates gpu:NoSchedule] ✅ Scheduled here

[NODE B - No Taint]
      |
      |--> POD 1 ✅ Will schedule
      |--> POD 2 ✅ Will also schedule
```

Taint = Node says *"Don't schedule pods on me unless you tolerate me."*

Toleration = Pod says *"I can handle that taint — let me run here."*

Step 1: Add a Taint to a Node

kubectl taint nodes aks-userpool-92868701-vmss000000 key1=value1:NoSchedule



Step 2: Apply a Pod with a Toleration

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deploy
spec:
  replicas: 2
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
```

```
    labels:
      app: myapp
  spec:
    tolerations:
    - key: "role"
      operator: "Equal"
      value: "frontend"
      effect: "NoSchedule"
    containers:
    - name: myapp-container
      image: prateek2004/my-frontend
      ports:
      - containerPort: 80
      livenessProbe:
        httpGet:
          path: /
          port: 80
        initialDelaySeconds: 15
        periodSeconds: 10
      readinessProbe:
        httpGet:
          path: /
          port: 80
        initialDelaySeconds: 5
        periodSeconds: 5
```

VERIFY:

kubectl get pods -o wide

It should land on `aks-userpool-92868701-vmss000000`

```
PS C:\WINDOWS\system32> kubectl get pods -o wide
>>
NAME                          READY   STATUS    RESTARTS   AGE     IP             NODE
NOMINATED NODE   READINESS GATES
myapp-deploy-cc75b64d-qcjjj   1/1     Running   0          7m25s   10.244.2.68    aks-userpool-92868701-vmss000000
<none>           <none>
myapp-deploy-cc75b64d-ql2h2   1/1     Running   0          7m15s   10.244.2.181   aks-userpool-92868701-vmss000000
<none>           <none>
PS C:\WINDOWS\system32>
```

In real production systems, taints are used to **segregate workloads**. For example, GPU-heavy nodes may be tainted so only pods with GPU requirements (and tolerations) run there. This prevents general workloads from using expensive resources. In my setup, I tainted one node and configured tolerations in the pod to target it — simulating a dedicated node scheduling scenario.