Kubernetes

Resource Limits

In a Kubernetes cluster, there are various types of pods and resources running, and these pods consume server resources like RAM and CPU. In some cases, if you want to restrict a pod from consuming excessive resources, such as memory or CPU, you can set resource limits. With resource limits, you can specify the amount of resources a pod or deployment is allowed to request and consume.
In the deployment manifest below, you can see that I have specified limits and requests for the container's resource usage..

```
kind: Deployment
apiVersion: apps/v1
metadata:
 name: slam-app
spec:
 selector:
   matchLabels:
     app: slam-app
 template:
   metadata:
     labels:
       app: slam-app
   spec:
     containers:
     - name: slambook-app
       image: amittashok/slambook-cicd
       resources:
         limits:  #define limit of resource from server
           memory: "128Mi"
           cpu: "500m"
         requests:  # request from define limits
```

```
          memory: "56Mi"
          cpu: "100m"
      ports:
      - containerPort: 3000
        targetPort: 3000
```

Probes

In some cases, when you don’t want to manually check if an application, pod, or container is running, you can use probes to automate this process. Kubernetes provides three types of probes:

1. Readiness Probe

Indicates whether the pod is ready to handle traffic or requests. If the readiness probe fails, the pod is removed from the Service’s endpoints and won’t receive traffic until it becomes ready again.

💡 **Use Case**:

- Ensuring that only healthy pods are part of the load balancer or Service.
- For applications that take some time to initialize but can start serving traffic after certain pre-conditions are met.

2. Liveness Probe

Indicates whether the pod is running and functioning properly. If the liveness probe fails, Kubernetes will restart the container to restore its functionality.

💡 **Use Case**:

- Detecting and recovering from application deadlocks or crashes.
- Ensuring the application remains responsive over time.

3. Startup Probe

Indicates whether the application within the container has started successfully. It is useful for containers with a slow initialization process. Once the startup probe succeeds, Kubernetes switches to using readiness and liveness probes (if configured).

💡 **Use Case**:

- Applications with long startup times, such as those requiring complex initialization or external dependencies.
- Ensuring Kubernetes doesn't restart a container unnecessarily during its startup phase.

Taints and Tolerations

Taints

Let's consider a scenario where an important pod is running on a specific worker node, and you don't want any other pods to be deployed on that node. In such cases, you can use Taints. Once you apply a taint to a worker node, no other pods will be scheduled on it unless they have a matching toleration.

You can apply taint by command

```
kubectl taint nodes worker-node-1  key=prod:NoSchedule
```

Remove taint from worker node

```
kubectl taint nodes worker-node-1  env=prod:NoSchedule-
```

Toleration

If you want to deploy a pod on a tainted worker node, you need to specify a Toleration in the deployment manifest. This allows the pod to be scheduled on that specific worker node despite the taint.

Thank you for today! We have covered **resource limits**, **probes**, and also explored **taints and tolerations** in Kubernetes.