Kubernetes from Basic to Advanced

kubernetes

Session #03
Scheduling

kubernetes

Session Contents

- Probes
- Init Containers
- Affinity
- Taints & Tolerations

kubernetes

Probes

Probes

- Kubernetes uses probes to have a better understanding about how pods are behaving
- Exists 3 types of probes: liveness, readiness and startup
- All of them are used by `kubelet` to get important understand about workload running inside the pod
- Probe can bes run using a command, HTTP request, TCP request or gRPC request

kubernetes

Liveness Probes

- Liveness probes are used to know when to restart a container
- For example, liveness probes could catch a deadlock, where an application is running, but unable to make progress
- Restarting a container in such a state can help to make the application more available despite bugs.

kubernetes

Readiness Probes

- Readiness probes are used to know when a container is ready to start accepting traffic
- A Pod is considered ready when all of its containers are ready but workload may be not ready to receive requests
- One use of this signal is to control which Pods are used as backends for Services. When a Pod is not ready, it is removed from Service load balancers.
- Used during rolling update to only make new pods available after readiness probe successed

kubernetes

Startup Probes

- Startup probes are used to know when a container application has started
- If such a probe is configured, it disables liveness and readiness checks until it succeeds, making sure those probes don't interfere with the application startup
- This can be used to adopt liveness checks on slow starting containers, avoiding them getting killed by the kubelet before they are up and running.

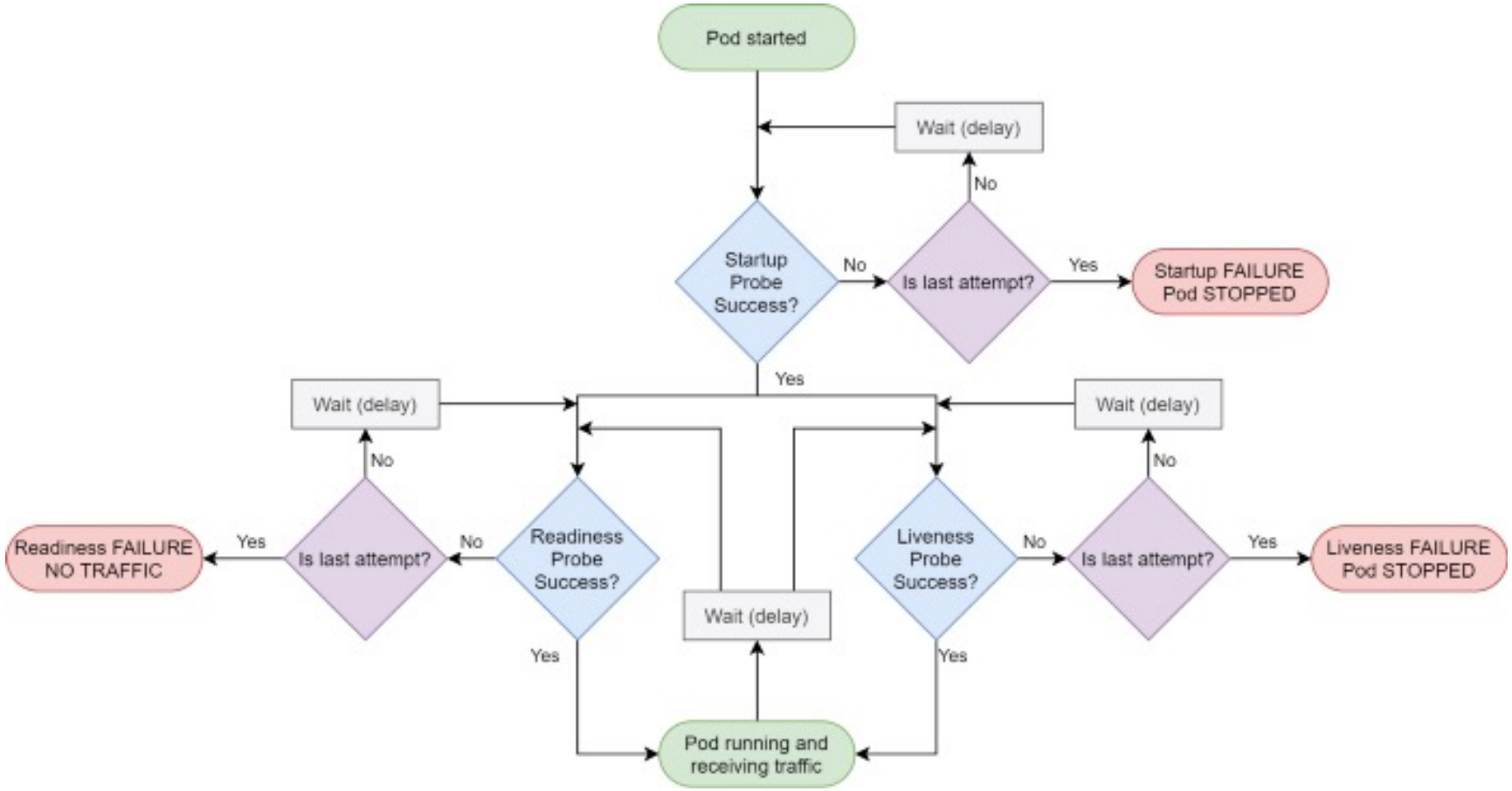
kubernetes

How to tune probes?

- `initialDelaySeconds`: Number of seconds after the container has started before startup, liveness or readiness probes are initiated. Defaults to 0 seconds.
- `periodSeconds`: How often (in seconds) to perform the probe. Default to 10 seconds. Minimum value is 1.
- `timeoutSeconds`: Number of seconds after which the probe times out. Defaults to 1 second. Minimum value is 1.
- `failureThreshold`: After a probe fails failureThreshold times in a row, Kubernetes considers that the overall check has failed

kubernetes

Probes workflow



kubernetes

Demo | Probes

Init Containers

Init Containers

- A Pod can have multiple containers running apps within it, but it can also have one or more init containers, which are run before the app containers are started
- Init containers are exactly like regular containers, except:
  - Init containers always run to completion.
  - Each init container must complete successfully before the next one starts.
- This approach differs from probes since you can execute tasks on a separate container

kubernetes

Init Containers: Failures

- If a Pod's init container fails, the kubelet repeatedly restarts that init container until it succeeds
- If a pod fails and need to be restarted, the init container is not executed again

kubernetes

Demo | Init Containers

Affinity

Motivation

- Kubernetes runs a Scheduler to automatically selected best node to run a pod based on some specific metrics
- But sometimes makes sense to give additional information to help scheduler to decide
- Scheduler will use those information to select all nodes that respect the restriction shared, and after select the best one where to schedule the pods

kubernetes

nodeName

- `nodeName` is the more direct form to select a node
- If present, Scheduler don't do any calculation and tries to directly schedule the pod
- This property overrules any other related with scheduling
- Some limitations
  - If the named node does not exist, the Pod will not run, and in some cases may be automatically deleted.
  - If the named node does not have the resources to accommodate the Pod, the Pod will fail and its reason will indicate why, for example OutOfmemory or OutOfcpu.
  - Node names in cloud environments are not always predictable or stable.

nodeSelector

- `nodeSelector` is the simplest recommended form of node selection constraint.
- You can add the **`nodeSelector`** field to your Pod specification and specify the node labels you want the target node to have
- Kubernetes only schedules the Pod onto nodes that have each of the labels you specify
- With this approach you are free from all limitation from `nodeName` approach

kubernetes

Affinity and anti-affinity

- Affinity and anti-affinity works on a similar way as `nodeSelector` but expands the types of constraints you can define
- `nodeSelector` only selects nodes with all the specified labels. Affinity/anti-affinity gives you more control over the selection logic.
- You can define rules that are mandatory or preferable
- You can define affinity based on Pods running on nodes instead of nodes properties

kubernetes

Affinity and anti-affinity

- There are two types of node affinity:
  - `requiredDuringSchedulingIgnoredDuringExecution`: The scheduler can't schedule the Pod unless the rule is met. This functions like nodeSelector, but with a more expressive syntax.
  - `preferredDuringSchedulingIgnoredDuringExecution`: The scheduler tries to find a node that meets the rule. If a matching node is not available, the scheduler still schedules the Pod.
- `IgnoredDuringExecution` means that if the node labels change after Kubernetes schedules the Pod, the Pod continues to run.

kubernetes

Node Affinity Weight

- You can specify a weight between 1 and 100 for each instance of the `preferredDuringSchedulingIgnoredDuringExecution` affinity type
- When the scheduler finds nodes that meet all the other scheduling requirements of the Pod, the scheduler iterates through every preferred rule that the node satisfies and adds the value of the weight for that expression to a sum
- Nodes with the highest total score are prioritized when the scheduler makes a scheduling decision for the Pod

Pod Affinity

- Inter-pod affinity and anti-affinity require substantial amount of processing which can slow down scheduling
- This amount is impacted from number of rules to process
- Preferable rules have a bigger computing cost than required ones
- Nevertheless, Kubernetes community states that you should only see impact on clusters larger than several hundred nodes

kubernetes

Pod Affinity and Topology Key

- Topology key is mandatory when defining Pod Affinity rules
- This property specify a label to be present on every node to be used to identify them on an unique way
- On on-prem clusters, `kubernetes.io/hostname` is commonly used
- But on cloud cluster you use different keys, like region where node is located, a zone inside a region or any additional information

kubernetes

Demo | Affinity

Taints & Tolerations

Motivation

- Affinity and node selectors define how a pod can “select” a node to schedule its containers
- But from Node perspective, could have some reasons for a node to “repel” a pod from being
- To have that behavior, a node can be tainted (marked, contaminated) and to only schedule pods that tolerate that taint

kubernetes

Taints & Tolerations

- Taints (one or more) are applied to the nodes
- Tolerations are applied to pods allowing the scheduler to schedule pods with matching taints
- Taints and tolerations work together to ensure that pods are not scheduled onto inappropriate nodes
- By default, nodes don’t have any taint. You need to add it manually

kubernetes

Taints & Tolerations

- Add a taint

```
kubectl taint nodes node1 key1=value1:NoSchedule
```

- Remove a taint

```
kubectl taint nodes node1 key1=value1:NoSchedule-
```

kubernetes

Taints & Tolerations: Effects

- When you create a taint you can define 3 effects: `NoSchedule`, `PreferNoSchedule or NoExecute`
- `NoSchedule`, only allow to schedule pods with toleration
- `PreferNoSchedule`, is a soft version of NoSchedule. Pod can be scheduled even without toleration if there is any other node to use
- `NoExecute`, when taint is added any pod running on that node without toleration will be evicted

kubernetes

Use Cases

- Dedicated Nodes: If you want to dedicate a set of nodes for exclusive use by a particular set of users/pods. Is used to have system pods on specific nodes without user pods
- Nodes with Special Hardware: In a cluster where a small subset of nodes have specialized hardware (for example GPUs), it is desirable to keep pods that don't need the specialized hardware off of those nodes
- Taint based Evictions: A per-pod-configurable eviction behavior when there are node problems, which is described in the next section

Demo | Taints & Tolerations

Questions?

kubernetes

Lab #03: Scheduler

kubernetes