Kubernetes from Basic to Advanced

Session #06
HPA & Probes

kubernetes

Session Contents

- Autoscaling
- Probes
- Init Containers

kubernetes

Autoscaling

Motivation

- Kubernetes can handle several replicas of the same pods
  - ReplicaSets handle replication
  - Services handle load balancing between them
- However, if the demand of a service starts to grow, the number of replicas deployed may be not sufficient to handle requests
- Number of replicas can be changed manually but it’s not scalable
- Kubernetes have a HorizontalPodAutoscaller (HPA) object to handle scalability of a Deployment automatically

kubernetes

Horizontal Pod Autoscaler

- Horizontal scaling means that the response to increased load is to deploy more Pods
- HPA defines a minimum and maximum number of replicas
- If the load increases, and the number of Pods is below the configured maximum, the HPA instructs the Deployment to scale up
- If the load decreases, and the number of Pods is above the configured minimum, the HPA instructs the workload resource to scale down
- HPA uses a control loop with a definedinterval (default is 15 seconds) to check if some change is needed

kubernetes

Horizontal Pod Autoscaler

- To make the decision about scaling, HPA uses metrics about pods resources (CPU, Memory) utilization
- Metric target can be set as a percentage or raw value (preferable)
- Percentage value: controller calculates the utilization value as a percentage of the equivalent resource request
- Raw value: metric values are used directly
- HAP uses a mean of the utilization or the raw value across all targeted Pods, and produces a ratio used to scale the number of desired replicas.

```
desiredReplicas = ceil[currentReplicas * ( currentMetricValue / desiredMetricValue )]
```

kubernetes

Horizontal Pod Autoscaler

- HAP uses a mean of the utilization or the raw value across all targeted Pods, and produces a ratio used to scale the number of desired replicas.
- Algorithm uses the following formula

```
desiredReplicas = ceil[currentReplicas * ( currentMetricValue /
                    desiredMetricValue )]
```

- If metrics cannot be gathered from one pod, is considered as using 0% for scale up and 100% for scale down

HPA Metrics

- HPA can use 3 types od metrics to make the decision to scale up/down: per-pod resource metrics, custom metrics and external metrics
- Per-pod resource metrics: metrics gathered by native Metrics Server, like CPU, Memory and, GPU (near future)
- Custom metrics: metrics gathered by metrics scrappers plugin installed on the cluster, like Prometheus. For instance, you can autoscale your pods based on number of requests.
- External metrics: metrics that can be gathered from external resources using an additional plugin like Prometheus. For instance, you can autoscale your pods based on queued messages on a message queue.

kubernetes

HPA with Multiple Metrics

- You may specify more than one metric to be analyzed by HPA
- HPA makes the calculation for each metric and then select the biggest replica number from those calculations
- HPA can use multiple metrics from different sources

Other types of autoscaling

- Vertical Pod Autoscaler: adjusts the resource requests and limits of a container
- Cluster Autoscaler: adjusts the number of nodes of a cluster
- Both tools are defined and maintained by Kubernetes community but not available on a vanilla Kubernetes cluster
- VPA is fully implemented by Kubernetes community code
- Cluster Autoscaler needs to have specific implementation from nodes provider (and really hard to implement in on-prem cluster with bare metal ☺)

kubernetes

Demo | HPA

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.

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

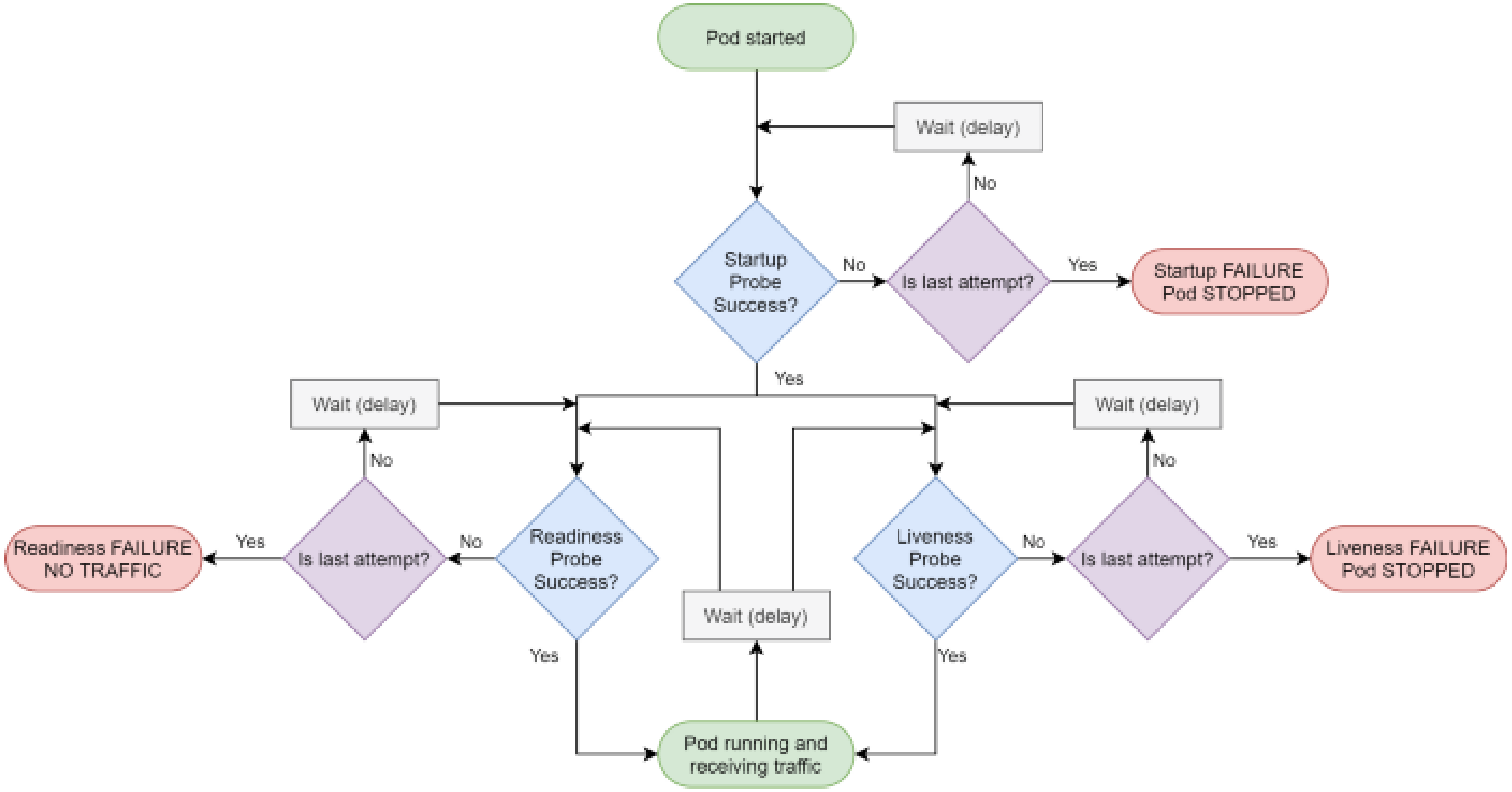
Startup Probes

- 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.

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

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.
- If a Pod's init container fails, the kubelet repeatedly restarts that init container until it succeeds
- This approach differs from probes since you can execute tasks on a separate container

kubernetes

Demo | Init Containers

Questions?

kubernetes

Lab #05: HPA