Hi Everyone this is Deekshith, Welcome to my channel

today I will discuss about liveness and readiness probe in kubernetes

prerequisite to watch this video is you need basics of docker and kuberenetes

**now we will see need for Probes**

What are these liveness probe and readiness probes? Why do we need them?

Pod is a collection of 1 or more docker containers. It is an atomic unit of scaling in Kubernetes.

Pod has a life-cycle with multiple phases. For example, When we deploy a pod in the Kubernetes cluster, Kubernetes has to start from scheduling the pod in one of the nodes in the cluster, pulling the docker image, starting a container and ensuring that containers are ready to serve the traffic etc!

As I have already mentioned Pod is the collection of docker containers, in order to accept any incoming request, all the containers must be ready to serve the requests! So it will take some time – usually within a minute & but it mostly depends on the application. So, as soon as we send a deployment request, our application is not ready to serve! Also, we know that software will eventually fail! Anything could happen.For example, a memory leak could lead to OOM error in few hours/days. In that case, Kubernetes has to kill the pod when the pods are not working as expected and reschedule another pod to handle the load on the cluster.

Livenesss and Readiness probes are the tools to monitor the pods & its checks health and take appropriate actions in case of failure.

These probes are optional for a Pod deployment and should be configured under a container section in the deployment file if required.

To demonstrate these probes behavior, I have docker image which has simple spring boot application packaged init.

* It has 2 end points.
  + - **/health** simply sleeps for the given time and responds with a message how long it was sleeping
    - **/processing-time/{time}** – it sets new time for sleeping for any future **/health** requests

The idea for the above requests to simulate some request processing time or to simulate server hanging behaviour

Processing time is 0 by default.

In my dockerfile there is a env variable called START\_DELAY to simulate slow server start time

* Now, Lets try to access the application within 60 seconds.
  + If you are within the cluster, access like this (if not, use your machine ip:nodeport)

wget -qO- http://probe-demo:8080/health

Copy

Screenshot from 2019-08-27 21-41-50

* We get connection refused error as server itself would have not started during this time.
* After 60 seconds, same request shows below response which is expected

Basically Kubernetes thinks that App is ready as soon as it enters the running status. But in our case, it takes more than 60 seconds. This is a problem during deployment. So we need some kind of health check and app should be considered as ready only when the health checks are passing. That is where readiness probe comes into picture.

If you see even exactly after 60 seconds, the pods are NOT in ready status. This is because readinessProbe’s initial delay and it waits for the readiness probe health check to pass.

* The pods become available after few seconds after the initial delay as it passes the health check.
* Now lets issue this command. We have 2 pods. one of the pods will receive this request and its request processing time becomes 30 seconds (below time in milli seconds)

wget -qO- http://probe-demo:8080/processing-time/30000

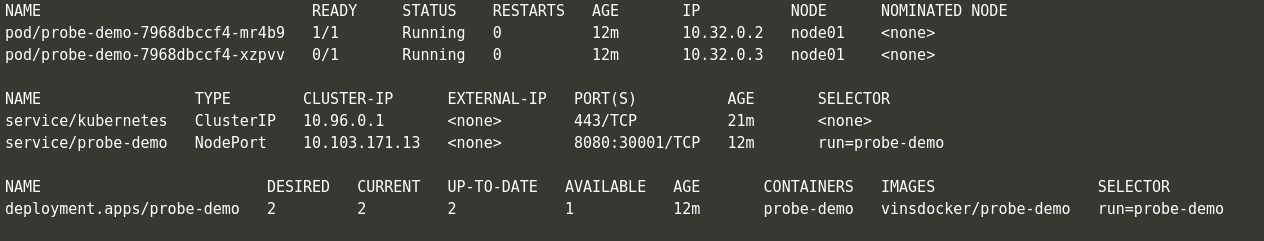
Copy

* If you issue couple of requests, you can see one request takes 30 seconds to respond

wget -qO- http://probe-demo:8080/health

Copy

* If you check the Kubernetes status, we can see that one of the pods become unavailable.



* This is because readinessProbe continues to monitor the health of the pods even after it becomes available after the first 60 seconds. If it does not respond as we had configured, it removes the pod from the service. So that the below request does not go to the pod which is not responding.

wget -qO- http://probe-demo:8080/health

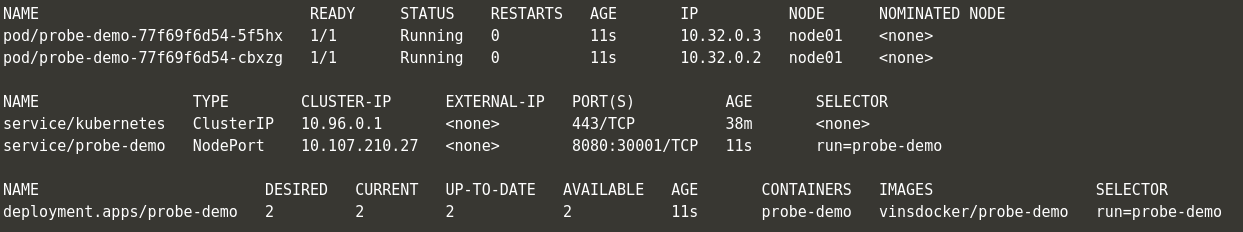
Copy

*Note: The readiness probe is used to remove the pod from the service. But it does not kill the pod. Pod is continued to be in running status even though it is not responding as we expected.*

But we might want to take some action for the pods which are causing issues like this. In this case, I might want to restart pod.

This we can achieve it through livenssprobe

* If you check the Kubernetes objects status, within 11 seconds, the pods are shown as Available. But actually they are NOT ready to accept any traffic as we know this already due to the initial delay we had set!



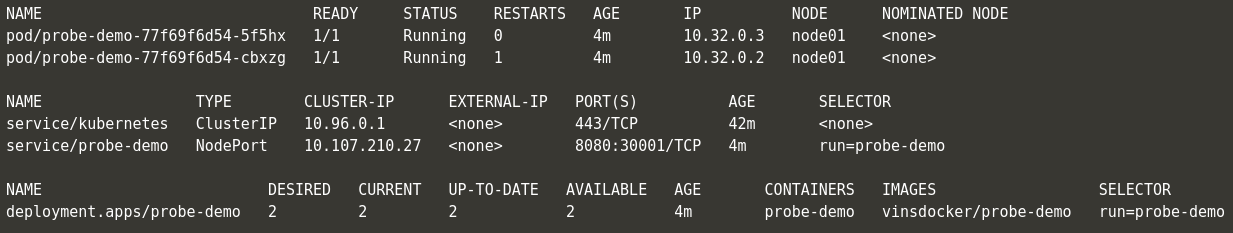
*Note: Liveness probe can NOT be used to make a pod unavailable in case of issues.*

* Lets send a request to simulate slow processing time.

wget -qO- http://probe-demo:8080/processing-time/30000

Copy

* We know that one of the pods which receives the request will not respond for 30 seconds. So livenessProbe checks will start failing.
* Whenever the liveness probe check fails, the pods are restarted. If you see the restart count has increased to 1 from 0 for one of the pods.



* However it does not change the Available count. That is, if the pod restarts the initial start delay we had set in the container will once again make the pod be actually unavailable for 60 seconds.

*Note: Both readiness probe and liveness probe seem to have same behavior. They do same type of checks. But the action they take in case of failures is different. Readiness Probe shuts the traffic from service down. so that service can always the send the request to healthy pod whereas the liveness probe restarts the pod in case of failure. It does not do anything for the service. Service continues to send the request to the pods as usual if it is in ‘available’ status.*

**Summary:**

After seeing Readiness probe and Liveness probe behaviors, It is recommended to use both probes in the deployment yaml as shown below. The reason being, in case of any pod failure or during deployment, with the help of the probes, we would be sending all the requests only to a healthy pods and pods would be restarted immediately in case of failure.