Candidate: Nawordth Rodrigo

GH Repository : https://github.com/KNrodrigo/rodrigo_swipejobs

Updated and optimized files are located in the above repo.

Q1.

In the docker file we see that the maximum and initial heap size of the JVM is set to 12 gigabytes of memory, which is substantially large for a simple hello world application. Usually, to avoid out of memory issues, where the application cannot start, it is recommended to use a higher memory request in the container definition than the allocated heap size.

In the deployment manifest it is true that the resource requests for memory are of 2gi, which is much less than the initial heap size mentioned in the Dockerfile. It can also be assumed that the given memory allocation could have been a typo in either of the files. However, the point to be noted here is that the allocated container request memory should typically be at least the same size of the initial heap size for the given java application. In this case we could potentially see an out of memory exception / memory insufficient error. Therefore, we need to ensure our containers have sufficient memory to handle overall memory requirement of our java app, as well as other memory needs of the container.

In Kubernetes the resource request and limits are usually calculated depending on certain conditions such as any SLA, SLO, memory, CPU etc of a particular service. Developers and testers usually determine the needed memory and or CPU for a particular pod to run the service.

Based on my understanding here is what the two parameters mean.

Memory Request: Memory request is the amount of memory that the container will be allocated when scheduled within a pod. When we specify a memory request the kube scheduler determines what node should the pod be placed to. It should also be noted that the cluster node usually needs to have enough available memory resources to cater at least the resource memory request for a specific pod, else the pod by default can’t be scheduled, unless we have provided other polices such as pod priorities and etc.

The memory limit : memory limit is the maximum amount of memory that a container can be allocated, provided that there is enough memory in the cluster nodes. This parameter is usually advisable to be provided to make sure that no particular container overuses memory allocations in a cluster node.

Q2.

In the Dockerfile.

Instead of pulling the latest tomcat version, it could be better to use a specific version of tomcat to make sure images are consistent in different environments and builds. Also as mentioned earlier the heap size is too large, we could optimize the docker image by making this value smaller. Apart from that I feel that the other commands of this docker file are generally acceptable.

In the Kubernetes manifest file

Issues

The spec.selector.matchLabels is missing and we need to add it and to make sure it matches the spec.template.metadata.labels, (pod template specifications)

We also need to create the respective volume for the volume mount, for example Persistent Volume and using a PVC in the specific namespace the pod is created.

Also under spec.template.spec.containers , Kubernetes requires us to provide a list of containers with at least one container spec. The syntax is wrong in the YAML file as it misses a hypen (-) in the container spec to denote that it's a list.

Under the container probes the port is specified as http , however we need to give a valid port number. Since the container port is 8080 we can safely give 8080 provided that /health is accessible and present as a http get request.

Improvements

As an improvement I also notice that the period seconds given to test readiness and liveness is 3600 which is one hour. I would feel its much better to bring this down to a number like 60 seconds to measure health more often and thereby improving the availability and reliability of your application. This will also make sure the pods are ready within 60 seconds after initial delay second is lapsed. And if for some reasons health checks start to fail, Kubernetes would take down the pod (restart or remove from deployment) so that customers requests don't land on unhealthy pods.

Additionally, I also see that the image pull policy is set to Never since the image is built locally and tagged to devops: test. In a prod system environment we can improve this by publishing the image to a remote repository and updating the image pull policy to a different attribute such as always.

As for another improvement it would be ideal to also add CPU resource request and limits so that the container doesn't overuse node CPU resources.

Also, I did only see deployment kind, if we need to allow users to access our application we need to also expose a service for those set of pods such as nodeport, clusterIP or etc so that our application is accessible within or outside of our cluster

Q3.

As mentioned above in Question 1. It is recommended that the container memory requirements (environment requirements) is larger or at least the same size of the heap memory requirements. So, every time a modification is made to the docker file it is advisable to update the Kubernetes manifest file accordingly. If it was only to be modified in one of the two files, we could potentially see out of memory exceptions or even sometimes a waste of memory resources.

To automate this, we could use tools such as Ansible or Jenkins, I have done the below way to overcome a situation like this in my previous roles.

Here is what we can do.

1. Create a playbook to manage the k8s deployment manifests and docker file configurations.
2. We could use jinja templating to refer configurations that would be dynamic in nature.
3. In this case we can make use of Ansible variables to manage the memory configuration.

Once a developer needs to update the memory configuration, they can run an ansible playbook directly or through Jenkins (recommended) to simultaneously update the two files. (Docker file and K8s Deployment file). I have provided the updated two files in the respective GH repository.

Q4.

Since, this is a hello world application, I would assume that we might not need 12gib of initial heap size memory, So I have gone and reduced it to 512 and 2048 mib respectively for the initial and final heap size respectively.

By doing this I am able to keep the container memory allocation to 2048 mib (2Gi) as it is making sure that the application has enough heap size to run on the container.

I would not stop from here, we could introduce some monitoring capabilities to measure the JVM usage within the docker container. And we could set alarms if it reached a pre determined value lets say (75%) of max heap size. If so then the developer/devops engineer can decide to increase the value of the memory allocation using the automation described in Q3.

By doing this we also make sure that we don't spin up unnecessary resources in a cluster due to inefficient memory management, this in turn would save us compute cost resources.

Q5.

To answer this question I will reflect back on a real world scenario I did and this was what we did at one of my previous roles.

Every service can have its own config map template. But however, values in this configmap template that were dynamic were referenced as variables.

We had ansible playbooks that will do pre validation and population of these variables into these configmaps manifest and other k8s workload manifest files. A Jinja template for the manifest files was used where variables were given placeholders that would be referenced from a central location (could be ansible variables or a cloud store such as AWS parameter store) .

By doing it this way we were able to manage a consistent sharing of variables across the vast amount of microservices when these microservices were built , validated and deployed using CICD pipelines.

Everything that had code was centrally updated to a GH repository so that teams could work on together.

Hence the same approach could also be used to answer this question.

Please also note that there are other tools out there which could potentially be a better approach. However I did want to answer this question reflecting back on my own experience.

Q6.

Updated in the GH Repo file (q6.py)

Outcome and Testing

I have installed a minikube cluster on my local ubuntu VM and tried creating the docker image and the Kubernetes deployment. I was successful in creating the docker image however i was unsuccessful in creating the deployment mainly due to health check failures on /health on the original deployment due to a 404. Hence in prod environments we need to ensure we give the correct path or pattern..

However just for testing and out of curiosity i changed the path to / which then i was successful in creating the deployment.

NOTE: Due to local VM shortages I could not create PV and PVC and for simplicaity, I used a simple method of using host path to mount a local path /etc/timezone to the container instead of the application properties for obvious reasons as this file was missing .

Necessary logs and screenshots are attached in the GH repo.

THANKYOU

RODRIGO