Question 1: Creating a Single Container Pods

a. Create a pod with the name of kplabs-nginx.

b. The pod should be launched from an image of mykplabs/kubernetes:nginx

c. The name of the container should be mycontainer

Answer:

apiVersion: v1

kind: Pod

metadata:

name: kplabs-nginx

spec:

containers:

- name: mycontainer

image: mykplabs/kubernetes:nginx

Question 2: Commands and Arguments

Create a pod with the name of kplabs-cmdargs. The pod should be launched from an image of busybox. The name of the container should be cmdcontainer. Both the container image's CMD and ENTRYPOINT instruction should be overridden.

The container should start with sleep command and argument of 3600

Answer:

apiVersion: v1

kind: Pod

metadata:

name: kplabs-cmdargs

spec:

containers:

- name: cmdcontainer

image: busybox

command: ["sleep"]

args: ["3600"]

Reference Documentation:

https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/

Question 3: Exposing Ports for PODS

Create a pod with the name of kplabs-ports. The pod should be launched from an image of nginx The name of the container should be nginx. Expose Port 80 for the POD.

Answer:

apiVersion: v1

kind: Pod

metadata:

name: nginx-ports

spec:

containers:

- image: nginx

name: nginx-ports

ports:

- containerPort: 80

Reference Documentation:

https://kubernetes.io/docs/concepts/services-networking/connect-applications-service/

Question 4: Arguments

Create a pod named kplabs-logging

The Pod should have a container running from the nginx image with the following arguments:

- /bin/sh

- -c

- >

i=0;

while true;

do

echo "$i: $(date)" >> /var/log/1.log;

echo "$(date) INFO $i" >> /var/log/2.log;

i=$((i+1));

sleep 1;

done

Once POD is created, connect to the POD and verify the contents of /var/log/1.log and /var/log/2.log

Answer:

Step 1: Create the POD

apiVersion: v1

kind: Pod

metadata:

name: kplabs-logging

spec:

containers:

- image: nginx

name: logging-container

args:

- /bin/sh

- -c

- >

i=0;

while true;

do

echo "$i: $(date)" >> /var/log/1.log;

echo "$(date) INFO $i" >> /var/log/2.log;

i=$((i+1));

sleep 1;

done

Step 2: Check the Logs

kubectl exec -it kplabs-logging bash

tail /var/log/1.log

tail /var/log/2.log

-----------------------------------------

Question 1: Labels

Create a pod named kplabs-label. The pod should be launched from nginx image. The name of container should be nginx-container. Attach following label to the pod.

env=production

app=webserver

Answer to Question 1: Labels

apiVersion: v1

kind: Pod

metadata:

name: kplabs-label

labels:

env: production

app: webserver

spec:

containers:

- name: nginx-container

image: nginx

----------------------------------------------------

Question 2: Deployments

Create a deployment named kplabs-deployment. The deployment should be launched from nginx image. The deployment should have three replicas. The selector should be based on the label of app=nginx

Answer to Question 2: Deployments

apiVersion: apps/v1

kind: Deployment

metadata:

labels:

app: nginx

name: kplabs-deployment

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- image: nginx

name: kplabs-deployment

--------------------------------------------------------

Question 3: Deployments - Rolling Updates and Rollbacks

Create a deployment named kplabs-updates. The deployment should be launched from nginx image. There should be two replicas. Verify the status of the deployment. As part of rolling update, update the image to nginx2:alpine. Verify the status of deployment. Perform a rollback to the previous version. Verify the status of deployment.

Answer to Question 3: Rolling Updates and Rollbacks

Step 1: Create a Deployment

apiVersion: apps/v1

kind: Deployment

metadata:

labels:

app: nginx

name: kplabs-updates

spec:

replicas: 2

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- image: nginx

name: kplabs-updates

Step 2: Perform Rolling Update

kubectl set image deployment kplabs-updates kplabs-updates=nginx2:alpine --record=true

Step 3: Verify Status of Deployment

This can be verified in multiple ways, here are some of the approaches

kubectl rollout status deployment kplabs-updates

kubectl get pods

In the output of kubectl get pods command, you will see that new POD as part of deployment has failed to launch with the rror of ImagePullBackOff

Step 4: Rollback to the Previous Version

kubectl rollout undo deployment kplabs-updates

---------------------------------------------------------------

Question 4: Labels and Selectors

Create a deployment named kplabs-selector. The pods should be launched from nginx image.The pods should only be launched in a node which has a label of disk=ssd. Observe the status of deployment. Add the appropriate label to the worker node and then observe the status of the deployment.

Answer to Question 4: Labels and Selectors

Step 1: Create a Deployment based on a specification

apiVersion: apps/v1

kind: Deployment

metadata:

name: kplabs-selector

spec:

replicas: 1

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- image: nginx

name: kplabs-updates

nodeSelector:

disktype: ssd

Step 2: Verify the status of deployment:

Run the kubectl get pods command to see if pod is launched. With our use-case, it will not be launched and would be in the "Pending" state.

Step 3: Add Label the worker node as part of your k8s cluster

kubectl label node pool-hrhwxjzi6-kcru disktype=ssd

Once you have added a label to the node, you will observe that the Pod is now in the "Ready" state.

-------------------------------------------------------

Question 5: CronJob

Create a job named kplabs-job. The job should run every minute and should print out the current date.

Answer to Question 5: CronJob

CronJob manifest can be referenced from the official documentation.

https://kubernetes.io/docs/tasks/job/automated-tasks-with-cron-jobs/

We have used the template from the documentation and modified it according to the requirement of the question.

Step 1: Create the CronJob

apiVersion: batch/v1beta1

kind: CronJob

metadata:

name: kplabs-job

spec:

schedule: "\*/1 \* \* \* \*"

jobTemplate:

spec:

template:

spec:

containers:

- name: hello

image: busybox

args:

- /bin/sh

- -c

- date

restartPolicy: OnFailure

Step 2: Verify if POD is created.

It will take a minute for POD to be created. Verify if POD is created and also check the logs of the pod to check if date is displayed.

kubectl get pods

kubectl logs [pod-name]

Output:

-----------------------------------------------------------

Question 6: CronJob

Create a job named kplabs-cron. The job should run every minute and should run following command "curl kplabs.in/ping". Terminate the container within 10 seconds if it does not run.

Answer to Question 6: CronJob

apiVersion: batch/v1beta1

kind: CronJob

metadata:

name: kplabs-cron

spec:

schedule: "\*/1 \* \* \* \*"

jobTemplate:

spec:

activeDeadlineSeconds: 10

template:

spec:

containers:

- name: hello

image: busybox

command: ["curl", "kplabs.in/ping"]

restartPolicy: OnFailure

Reference Document:

A manifest template with activeDeadlineSeconds parameter for cronjob is already present within the documentation. Here is a link for the same.

https://kubernetes.io/docs/concepts/workloads/controllers/jobs-run-to-completion/

--------------------------------------------------

Question 7: Deployment Configuration

Create a deployment named kplabs-configuration. The deployment should have 3 replicas of nginx image. Once the deployment is created, verify the maxSurge and maxUnavailable parameters. Edit the the maxUnavailable to 0 and maxSurge to 30% on the live deployment object. Once those two parameters are modified, change the image of the deployment to nginx:alpine. Make sure to use the record instruction on rolling updates.

Answer to Question 7: Deployment Configuration.

Step 1: Create the Deployment.

apiVersion: apps/v1

kind: Deployment

metadata:

labels:

run: kplabs-configuration

name: kplabs-configuration

spec:

replicas: 3

selector:

matchLabels:

run: kplabs-configuration

template:

metadata:

labels:

run: kplabs-configuration

spec:

containers:

- image: nginx

name: kplabs-configuration

Step 2: Verification

To verify the maxSurge and maxUnavailable parameters, run the following command:

kubectl get deployment kplabs-configuration -o yaml

Step 3: Modify the Parameters

kubectl edit deployment kplabs-configuration

Edit the maxUnavailable to 0 and maxSurge to 30%

Set 4: Set New Image;

kubectl set image deployment kplabs-configuration kplabs-configuration=nginx:alpine --record

-----------------------------------------------------

Question 1: Service

Create a deployment named kplabs-service. The deployment should have three replicas and the image should be based on nginx. Create a service based on NodePort. The service port should be 8080. Website should be accessible from port 32001 from all hosts.

Answer:

-------

Step 1: Create the deployment

Command:

kubectl run kplabs-service --image=nginx --replicas 3

apiVersion: apps/v1

kind: Deployment

metadata:

labels:

run: kplabs-service

name: kplabs-service

spec:

replicas: 3

selector:

matchLabels:

run: kplabs-service

template:

metadata:

labels:

run: kplabs-service

spec:

containers:

- image: nginx

name: kplabs-service

Step 2: Create a NodePort Service

apiVersion: v1

kind: Service

metadata:

name: myservice

labels:

run: myservice

spec:

type: NodePort

ports:

- port: 8080

targetPort: 80

nodePort: 32001

protocol: TCP

selector:

run: kplabs-service

-------------------------------------------------------------

Question 2: Troubleshoot Service

Apply the following manifest file in your Kubernetes environment:

https://github.com/zealvora/myrepo/blob/master/demo-files/troubleshoot-service.yaml

Verify if you are able to access website by referencing to the service IP address from a busybox pod. If it's not working, fix the issue so that the website is downloadable when following command is ran: wget [SERVICE-IP]:8080

Answer:

-------

Step 1: Download the manifest file and create the objects accordingly:

kubectl apply -f troubleshoot-service.yaml

Step 2: Create a Busybox Pod for Testing

kubectl run busybox --image=busybox --restart=Never --command sleep 3600

Step 3: Verify if you are able to download the website page

kubectl exec -it busybox sh

wget [SERVICE-IP]:8080

In this scenario, you should get connection refused error.

Step 4: Fix the Issue

If you will observe the service endpoints, they are associated with a deployment named "kplabs-fix". This deployment is running based on nginx image and is running on a default port of 80. However within the service, the targetPort is 8080. This is causing the issue. Change the targetPort to 80.

kubectl edit service fix-service

Once the issue has been fixed, run the wget command once again and everything should work as expected. The index.html file will be downloaded.

-----------------------------------------------------

Question 3: Namespace

Create a pod named kplabs-namespace. The pod should be part of namespace kplabs.

The pod should make use of redis image. Expose port 6379.

Answer:

--------

Step 1: Create the namespace

kubectl create namespace kplabs-namespace

Step 2: Run the pod in the given namespace with exposed port

kubectl run redis-pod --image=redis --restart=Never --port=6379 -n kplabs-namespace

--------------------------------

Question 4: Service Account

Create a new service account named kplabs. Launch a new pod named kplabs-sa from nginx image. The pod should be launched from the kplabs service account. Verify whether the token has been mounted inside the pod.

Answer:

-------

Step 1: Create a new Service Account

kubectl create serviceaccount kplabs

Step 2: Create a Pod with service account

kubectl run kplabs-sa --image=nginx --serviceaccount "kplabs" --restart=Never

--------------------------------------

Question 5: Deployments and Service Account

Create a deployment named deployment-sa. The deployment should have 2 replicas of nginx image. After the deployment has been created, check the service account associated with the pods. Modify the deployment so that all pods shall use service account of kplabs.

Answer:

-------

Step 1: Create a Deployment

kubectl run nginx-sa --image=nginx --replicas 2

Step 2: Associate the new service account with deployment.

This can be done with the help of kubectl edit command and add the "serviceAccountName: kplabs" under the template->spec section.

kubectl edit deployment nginx-sa

Here is a sample manifest of the real deployment

apiVersion: apps/v1

kind: Deployment

metadata:

annotations:

deployment.kubernetes.io/revision: "2"

creationTimestamp: "2020-01-04T00:23:57Z"

generation: 2

labels:

run: kplabs-sa

name: kplabs-sa

namespace: default

resourceVersion: "318110"

selfLink: /apis/apps/v1/namespaces/default/deployments/kplabs-sa

uid: ba658fc3-403e-4350-9bad-b6b2f78880d1

spec:

progressDeadlineSeconds: 600

replicas: 2

revisionHistoryLimit: 10

selector:

matchLabels:

run: kplabs-sa

strategy:

rollingUpdate:

maxSurge: 25%

maxUnavailable: 25%

type: RollingUpdate

template:

metadata:

creationTimestamp: null

labels:

run: kplabs-sa

spec:

containers:

- image: nginx

imagePullPolicy: Always

name: kplabs-sa

resources: {}

terminationMessagePath: /dev/termination-log

terminationMessagePolicy: File

dnsPolicy: ClusterFirst

restartPolicy: Always

schedulerName: default-scheduler

securityContext: {}

serviceAccount: kplabs

serviceAccountName: kplabs

terminationGracePeriodSeconds: 30

status:

availableReplicas: 2

conditions:

- lastTransitionTime: "2020-01-04T00:24:10Z"

lastUpdateTime: "2020-01-04T00:24:10Z"

message: Deployment has minimum availability.

reason: MinimumReplicasAvailable

status: "True"

type: Available

- lastTransitionTime: "2020-01-04T00:23:57Z"

lastUpdateTime: "2020-01-04T00:40:54Z"

message: ReplicaSet "kplabs-sa-564f6b576c" has successfully progressed.

reason: NewReplicaSetAvailable

status: "True"

type: Progressing

observedGeneration: 2

readyReplicas: 2

replicas: 2

updatedReplicas: 2

---------------------------------------------------------------------

Question 6: Failed Deployments

Download and Apply the Following Manifest File to your k8s:

https://github.com/zealvora/myrepo/blob/master/demo-files/troubleshoot-deployment.yaml

There is failed deployment within your cluster. Find and fix it.

Answer:

------

Step 1: Download and Apply the Provided manifest file

kubectl apply -f troubleshoot-deployment.yaml

Step 2: Find all the deployments across all the namespaces

kubectl get deployment --all-namespaces

In the output of above command, you will see one deployment where none of the pods are in ready state (0/2). The deployment name is troubleshoot-deployment and resides in newkplabs namespace.

When you inspect the properties of deployment, you will see there is typo in the name of nginx image, it is "ninx". Edit the deployment and add the right name.

kubectl edit deployment troubleshoot-deployment -n newkplabs

After you have modified and added the right image, re-run the kubectl get deployment command for all namespaces to ensure that all the pods are up (2/2)

----------------------------------------------------------------

Question 1: Resource Quotas

Create a pod named kplabs-quota. The pod should have following configuration:

a. Should run with nginx image.

b. It should use maximum of 512 MiB of memory.

c. It should use maximum of 2 core CPU.

d. The POD should require a minimum of 128 MiB of memory before it is scheduled.

Answer:

-----------

Step 1: Create a basic POD manifest file based on given specification.

kubectl run kplabs-quota --image=nginx --restart=Never --dry-run -o yaml > quotas.yaml

Step 2: Add the Quota according to the number specified.

Reference Documentation:

https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/

apiVersion: v1

kind: Pod

metadata:

labels:

run: kplabs-quota

name: kplabs-quota

spec:

containers:

- image: nginx

name: kplabs-quota

resources:

requests:

memory: "128Mi"

limits:

memory: "512Mi"

cpu: "2"

-----------------------------------------------------

Question 2: Secrets

Create a secret named kplabs-secret. The secret should have content where user=admin and pass=12345. Create a pod from the nginx image. Mount the secret as environment variables in the pod. The username should be available as DB\_USER and password should be available as DB\_PASSWORD inside the pod

Answer to Question 2 - Secrets

Step 1 - Create Secret from Literal Value

kubectl create secret generic kplabs-secret --from-literal=user=admin --from-literal=pass=12345

Step 2: Create a POD and Mount Secret as Environment Variable

apiVersion: v1

kind: Pod

metadata:

name: kplabs-secret-pod

spec:

containers:

- name: mycontainer

image: nginx

env:

- name: DB\_USER

valueFrom:

secretKeyRef:

name: kplabs-secret

key: user

- name: DB\_PASSWORD

valueFrom:

secretKeyRef:

name: kplabs-secret

key: pass

restartPolicy: Never

Reference Documentation:

https://kubernetes.io/docs/concepts/configuration/secret/

----------------------------------------------------------------------

Question 1: Probes

Create a POD from nginx image. Pod should named kplabs-probe. The pod should be created in such a way that if the application inside is not responding to HTTP requests made on port 8080, then Kubernetes should restart the POD.

Answer 1: Probes

---------------

We need to make use of HTTP method for Liveness Probe since restart is required.

apiVersion: v1

kind: Pod

metadata:

name: kplabs-probe

spec:

containers:

- name: liveness

image: nginx

livenessProbe:

httpGet:

path: /

port: 8080

initialDelaySeconds: 3

periodSeconds: 3

------------------------------------

Question 2: Probes

Create a POD named newprobe. Pod should run from nginx image. The Pod should run with arguments defined below. Create a probe which checks if file on that path /tmp/myfile exists. If it does not exists, the POD should be restarted.

- /bin/sh

- -c

- touch /tmp/myfile; 3600

Answer 2: Probes

-----------------

apiVersion: v1

kind: Pod

metadata:

name: newprobe

spec:

containers:

- name: liveness

image: nginx

args:

- /bin/sh

- -c

- touch /tmp/myfile; sleep 3600

livenessProbe:

exec:

command:

- cat

- /tmp/myfile

initialDelaySeconds: 3

periodSeconds: 3

-------------------------------------------

Question 3: Troubleshooting Probes

Apply the following manifest file in your K8s enviornement.

https://github.com/zealvora/myrepo/blob/master/demo-files/troubleshoot-liveness.yaml

Await for Next 30 seconds. Observe that the pod named liveness-http keeps on restarting. endpoint. Check and fix the issue.

Answer:

----------

Step 1: Download and create objects as part of YAML file

kubectl apply -f troubleshoot-liveness.yaml

After you create the objects, you will notice that a pod named liveness-http keeps on restarting. On observation, you will see that it is running based on nginx image but liveness probe is configured on port 8080. This is the misconfiguration. To fix it, edit the configuration.

kubectl edit pod liveness-http

Note: You cannot edit probe of live object. Editing and saving will create a new file in /tmp. Once that file is created, delete the existing pod and create new pod from YAML file created in /tmp directory

kubectl delete pod liveness-http

kubectl apply -f /tmp/kubectl-edit-m64ur.yaml

----------------------------------------------------

Question 4: Events

Store all the events associated with the pod liveness-http and store it to /tmp/podlog.txt. Make sure to use the kubectl command and do not modify anything in the output.

Answer:

-------------

kubectl get events --field-selector involvedObject.name="liveness-http"

Reference Documentation:

https://kubernetes.io/docs/tasks/debug-application-cluster/events-stackdriver/

-------------------------------------

Question 1 - ConfigMap

Create a configmap named kplabs-config which contains all the contents of that file.

course: kubernetes 2020

instructor: zeal

type: certification

Mount the configmap to a pod named configmap-pod based on nginx image in such a way that all contents are available at /etc/config/kplabs.config

Answer 1: ConfigMap

----------------------

Step 1: Create a file named kplabs.config and add all the contents specified in the question.

Step 2: Create a new configmap from that file

kubectl create configmap kplabs-config --from-file ./kplabs.config

Step 3: Create a POD with ConfigMap Mounted

apiVersion: v1

kind: Pod

metadata:

name: configmap-pod

spec:

containers:

- name: configmap-container

image: nginx

volumeMounts:

- name: config-volume

mountPath: /etc/config

volumes:

- name: config-volume

configMap:

name: kplabs-config

restartPolicy: Never

-----------------------------------------------

Question 2 - PV and PVC

Create a persistent volume with the name kplabs-pv. The size should be 2Gi and hostpath should be /tmp/mydata. It should have access mode of ReadWriteOnce

Create a persistent volume claim that will make use of the PV created earlier.

Create a Pod named kplabs-pv-pod. The POD should have the volume mounted at /mydata directory.

---------------------------------------

Question 3 - Security Context

Create a POD named busybox-security. The pod should run command "sleep 3600". The primary process in POD should run with UID of 1000 and GID of 2000 all contents of volume any mounted volume should have the group ID of 3000.

Answer 3: Security Context

-------------------------

apiVersion: v1

kind: Pod

metadata:

name: busybox-security

spec:

securityContext:

runAsUser: 1000

runAsGroup: 2000

fsGroup: 3000

containers:

- name: busybox-container

image: busybox

command: [ "sleep","3600" ]

-----------------------------------------------------------

Question 4 - Secrets and Environment Variables

Andrew works as a database administrator and has generated set of credentials that will be used by the application to connect to the database. Instead of giving the credentials to developers to hard-code in their application, he has requested security team to create a secret and mount it as environment variable to the application containers.

Answer 4: Secrets and Environment Variables

-------------------------------------------

Step 1: Create a secret

kubectl create secret generic db-creds --from-literal=user=dbreadonly --from-literal=pass=myDBPassword#%

Step 2: Create a POD with Nginx Image and Mount The Secret

apiVersion: v1

kind: Pod

metadata:

name: secret-pod

spec:

containers:

- image: nginx

name: secret-pod

env:

- name: DB\_USER

valueFrom:

secretKeyRef:

name: db-creds

key: user

- name: DB\_PASSWORD

valueFrom:

secretKeyRef:

name: db-creds

key: pass

a. Create a secret name db-creds which has following data:

user: dbreadonly

pass: myDBPassword#%

b. Create a pod from nginx image. The pod should be named secret-pod

c. Mount the secret to the POD in such a way that the contents of database user is available in form of DB\_USER environment variable and database password is available in form of DB\_PASSWORD environment variable inside the container.

----------------------------------------

Question 5 - Secrets and Volumes

a. Create a secret name app-creds which has following data:

appuser: dbreadonly

apppass: myDBPassword#%

b. Create a pod based on nginx image with the name of app-pod

c. Mount the secret to the pod so that it is available in the path of /etc/secret

Answer 5 - Secrets and Volumes

-----------------------------

Step 1: Create a Secret

kubectl create secret generic app-creds --from-literal=appuser=dbreadonly --from-literal=apppass=myDBPassword#%

Step 2: Create Manifest Files based on Specification;

apiVersion: v1

kind: Pod

metadata:

name: app-pod

spec:

containers:

- image: nginx

name: app-pod

volumeMounts:

- name: myvolume

mountPath: "/etc/secret"

readOnly: true

volumes:

- name: myvolume

secret:

secretName: app-creds

----------------------------------------------------------------

Practice Test - Ambassador Pattern

Enterprise Corp has a legacy application that was built by the team which no longer exists.

All the members of the team have left the organization. The legacy application only works on the port of 9080. The client does not want to always include port 9080 in their request.

They want that the application should be accessible via port 80. Create an ambassador container based on HAProxy which can handle the scenario. You don't have to learn about HAProxy. All the configuration files are provided.

Create a pod named kplabs-ambassador-pod from the legacy application image. The image is mykplabs/kubernetes:nginx

Create configmap called as kplabs-ambassador-config which has the following data:

global

daemon

maxconn 256

defaults

mode http

timeout connect 5000ms

timeout client 50000ms

timeout server 50000ms

listen http-in

bind \*:80

server server1 127.0.0.1:9080 maxconn 32

Create an ambassador container named haproxy-container from the image of haproxy:1.7

Expose the port 80 from Haproxy container.

Mount the configmap to the haproxy in such a way that HAProxy config is available at the following file:

/usr/local/etc/haproxy/haproxy.cfg

Create a Busybox pod from following pod definition:

apiVersion: v1

kind: Pod

metadata:

name: kplabs-busybox-curl

spec:

containers:

- name: curl-container

image: yauritux/busybox-curl

command: ['sh', '-c', 'while true; do sleep 3600; done']

Verify if you can perform CURL from busybox pod towards the ambassador pod on port 80.

Answer:

-------

Practice Test Solution - Ambassador Pattern

Step 1: Create a ConfigMap from given configuration file

- Store the contents of config file to haproxy.cfg

- Create ConfigMap from that file.

kubectl create configmap kplabs-ambassador-config --from-file ./haproxy.cfg

Step 2: Create Ambassador POD based on defined specification

apiVersion: v1

kind: Pod

metadata:

name: kplabs-ambassador-pod

spec:

containers:

- name: first-container

image: mykplabs/kubernetes:nginx

- name: haproxy-container

image: haproxy:1.7

ports:

- containerPort: 80

volumeMounts:

- name: config-volume

mountPath: /usr/local/etc/haproxy/haproxy.cfg

volumes:

- name: config-volume

configMap:

name: kplabs-ambassador-config

Step 3: Verify CURL

- Create the POD based on the defined manifest in the question.

- Connect to the Pod

- Find the IP of the Ambsassador POD

kubectl get pods -o wide

- Connect to the Busybox Curl Pod

kubectl exec -it kplabs-busybox-curl sh

- Run the Curl Command

curl [IP-OF-AMBSSADOR-POD]

--------------------------------------------------------------

Practice Test - Adapter Pattern

Question : Adapter Pattern

Create a pod named kplabs-adapter-logging.

The Pod should have a container running from the busybox image with the following arguments:

- /bin/sh

- -c

- >

i=0;

while true;

do

echo "$i: $(date)" >> /var/log/1.log;

echo "$(date) INFO $i" >> /var/log/2.log;

i=$((i+1));

sleep 1;

done

Create and mount a volume under the mount path of /var/log of the first container. The volume should be removed as soon as the pod is deleted.

Create a second container in the pod. It should be launched from the following image.

k8s.gcr.io/fluentd-gcp:1.30

The container should have an environment variable named FLUENTD\_ARGS with following values:

-c /etc/fluentd-config/fluentd.conf

The second container should also have the same volume as the first container mounted on the path of /var/log

The second container should also have a fluentd configuration (mentioned in below configmap) available in the following path:

/etc/fluentd-config/fluentd.conf

Create a ConfigMap object with the name of fluentd-config. The ConfigMap should have the following configuration:

<source>

type tail

format none

path /var/log/1.log

pos\_file /var/log/1.log.pos

tag PHP

</source>

<source>

type tail

format none

path /var/log/2.log

pos\_file /var/log/2.log.pos

tag JAVA

</source>

<match \*\*>

@type file

path /var/log/fluent/access

</match>

Verify if you can see log files with the tag of PHP and JAVA under the following directory

/var/log/fluent/access

Practice Solution - Adapter Pattern

------------------------------------

Step 1: Create ConfigMap

- Create a file named fluentd.conf and add the configuration defined in the question.

- Create a new configmap from the given file.

kubectl create configmap fluentd-config --from-file ./fluentd.conf

Step 2: POD Manifest based on specification

apiVersion: v1

kind: Pod

metadata:

name: kplabs-adapter-logging

spec:

volumes:

- name: log-volume

emptyDir: {}

- name: config-volume

configMap:

name: fluentd-config

containers:

- name: second-container

image: k8s.gcr.io/fluentd-gcp:1.30

env:

- name: FLUENTD\_ARGS

value: "-c /etc/fluentd-config/fluentd.conf"

volumeMounts:

- mountPath: /var/log

name: log-volume

- name: config-volume

mountPath: /etc/fluentd-config

- name: first-container

image: busybox

volumeMounts:

- mountPath: /var/log

name: log-volume

args:

- /bin/sh

- -c

- >

i=0;

while true;

do

echo "$i: $(date)" >> /var/log/1.log;

echo "$(date) INFO $i" >> /var/log/2.log;

i=$((i+1));

sleep 1;

done

Step 3: Verify the Logs

kubectl exec -it kplabs-adapter-logging bash

ls -l /var/log/fluent

-----------------------------------------------------------------------