PODS AND CONTAINERS IN K8S

1) Config Maps
2) Secret
3) Passing Config maps and Secret using.
   a) Environment Variables
   b) Using mount Volumes
   c) Posix Configmap
   d) From file.
4) Resource Request.
5) Resource limit
6) Liveness probe
7) Startup probe
8) Readiness probe
9) Restart policies (always, Onfailure, Never)
10) 10) multi Containers / Cross Containers.
11) Container initialization. (init Containers)

Page No.
Date

For Hands On :- Dojo video no:- 199

PODS AND CONTAINERS IN KUBERNETES

This is important topic.

1. Managing application Configuration
2. Managing Container Resources.
3. monitoring Container Health.
4. Building Self-Healing Pods.

Managing application Configuration :-

1) Most People already understand "Configuration" or Settings that influence the Operation of an application

2) Kubernetes allow users to pass dynamic Configuration Value to application at Runtime.

3) These Dynamic Configuration help user to Control the application flow.

Config Maps :-

1) Keep the Non-Sensitive Data in Config map, which Can be passed to Container Application.

2) Config Map Store Data in Key-Value format.

3) Config Maps allow you to Separate your Configurations

from your pods & Components.

(means we can have seperate Config map file for each Containers)

4) Config map help to makes Configurations easier to change & manage & prevents hardcoding Configuration data to pod Specifications.

CONFIGMAP Commands :-

1) Config map Via Config file (it will .YAML file).

Kubectl Create Configmap [Name] --from-file [Path of the file.Extensions] --from-file [path of the file with Extension]

Above Command we can use single file or multiple files. (--from-file).

- Via Directory path.

Kubectl Create Configmap [Name] --from-file [Path of Direct

(2) GET Configmap Via CLI

Kubectl get Configmap <Config_map_name] -o Yaml

Page No.
Date

To manage data dynamically we have one more objects called "Secrets"

SECRETS :- are similar to Configmaps but it is for sensitive data (Secret data)

(1) Secrets are similar to Configmap but designed to keep the sensitive data.

(2) Create Secrets from file.

```
Kubectl Create Secret generic db-user-pass --from-file
=./username.txt --from-file=./password.txt
```

Note:- Special character such as $, \, *, & ! require escaping.

(3) Get Secrets :-

Kubectl get Secrets.

(4) Describe Secrets.

Kubectl describe Secrets [Secret_name]

Page No.
Date

Sample Secrets YAML file :-

```
apiVersion : V1
Kind : Secret
metadata :
  Name : mySecret-manifest
  type : Opaque
data :
  Username : xxxvvvzzz-
  password : zzz@@@--
```

PASS CONFIGMAP AND Secrets to Containers.

1. By Environment Variables.
2. Using mount Volumes.
3.

Config-map & Secrets Value in to be passed to Containers. This can be achived by environment variables or mount volumes

1. By Environment Variables :-

User can pass Secrets & Config map to Container Using Environment Variables.

```
Spec:
  Containers:
    - .
    env:
      - Name : Special-Level-Key
      Value From :
        ConfigMapKeyRef :
          Name : Special-Config
          key : Special.how
```

Key hold the Key. to get value & pass to the Environment name.

(2) Mount Volume :-

a) Config mount Volume is another way to pass Config Data & Secrets to Containers.

b) Using this Config Data will be available in files to Container file System.

Syntax :-

```
Volumes:
  - name: Config-Volume
Configmap:
  name: special-config
```

Hands On :-

Lab Video number 221.

(3) Posix Configmap

We have one more way to Pass Configuration properties/ files to Containers. That is posix Configmap.

In env Variable Config map, for each & every property we have in the Config map, we was creating env variables

Suppose we don't want to do such kind of stuff

we don't want to create the env variable for each & every property which is available in my Config map and I want that for each property which I mentioned within my configmap, automatically one env variable will be declared with in my container.

how can we achieve this ? : we need to create a new config_map which is caued posix configmap.

Syntax:-

```
apiversion : v1
Kind    : ConfigMap
metadata:
  name : player-posix-demo
data :
  PLAYER_LIVES: "5"
  PROPERTIES_FILE_NAME : "user-interface-properties"
  BASE_PROPERTIES: "Template1"
  USER_INTEREFACE_PROPERTIES : "Dark".
```

NOTE:- In posix Configmap..

(a) Key shouldbe in Capital.
(b) One to one mapping. no multiple mapping./multiple values
(c) we "_" we underscore to separate out the words
(d) all the properties which you will define in posix Config map container will automatically create one env variable for that property..
(e) we also need to define new pod yaml file with uniform

```
apiVersion: v1
kind: Pod
metadata:
  name: Configmap-posix-demo
Specs:
  Containers:
    - name: Configmap-posix
      image: anshuldimaps/Kubernetes-web2:0.6
      ports:
        - ContainerPort: 8080
      envFrom:
        - ConfigMapRef:
            name: player-posix-demo
```

Command:-

1. Save the both in Syntax in diffrent file each.
2. Command to Create posix-Configmap.Yml:

   Kubectl apply -f posix-Configmap.Yml
3. Kubectl get Configmap
4. Kubectl describle posix-Configmap.
5. Kubectl apply -f Configmap-posix
6. Kubectl describle posix-Configmap.
7. Kubectl get pods.

Page No.
Date

Go To Container & Executed Commands.

Kubectl exec ~~pod~~ Configmap-posix-demo --it --bin/bash

Now we are inside the Container. for performing sh. use -it --sh.

(2) ls

(4) printenv.

Section 36 (224. Lab) - Udemy.

ConfigMap & Secret "from file."

(a) Create Secret from file.

(5) Create ConfigMap from file.

(3) Setup Nginx pod to Authenticate from Secret.

(1) Update the Container.

Sudo apt-get update.

(2) apt install apache2-utils.

(3) htpasswd -c .htpassd user.

given new password & Remember it.

(4) ls -a

we notice hidden file .htpasswd is Created.

(5) by using the file above, we can Create Secret.

Kubectl Create Secret generic saf-htpasswd
--from-file .htpasswd.

(6) Kubectl get Secrets.

Once we Created Secrets using hidden file, we don't need hidden file anymore, we can remove the file

rm -rf .htpasswd.

(7) for creating Configmap using the file : for Example take nginx.file.

Kubectl Create Configmap nginx-Config-file
--from-file nginx.conf.

(8) Next Create pod using mount volume.

Section 16:- (225 - Lab) - udemy.

MANAGE CONTAINER IN KUBERNETES.

manage Container Resources.

(1) We will see how we can limit the resource uses on Containers & how we can limit the resource used on Kubernetes nodes.

(a) Resource Request.
(b) Resource limit.

Resource Request :- is something which will define that how much resource a particular Container can request from Kubernetes cluster.

Resource limit :- basically a limit on a Container which you can define in Kubernetes.

Page No
Date

Resource Request :-

(1) Resource Request allow user to define a Resource limit, user expect a Container to use.

(2) Kube Scheduler will mange Resource Request and avoid Scheduling on node, which don't have Enough Resources.

(3) Note:- Containers are allowed to use more or less than the Request Resource.
Resource Request is to manage the Scheduling Only.

(4) Memory is measure in Bytes. User is allowed to define in megaByte as well.

(5) Requests for CPU resources are measured in CPU units. 1 vCPU means 1000 CPU unit.

For Ex:-
```
apiVersion : v1
Kind: Pod
metadata:
  name: frontend
Spec:
  Containers:
  - name: app
    image: nginx
    resources:
      Requests:
        memory: "64 mi"
        cpu: "250m"
```

Page No.
Date

Resource Limit :-

① Resource Limits used to limit the Container's resource use.
(means we are imposing to use that much resource itself).

② Limits are imposed at Run Time Container.

```
apiVersion: V1
Kind : Pod
metadata:
  name: frontend
Spec:
  Containers:
  - name: app
    image : nginx
    Resources:
      limits:
        memory : "128mi"
        cpu : "500m"
```

If Container is reaching the cpu limit, it means my Container is reaching the half of CPU or 500 cpu unit. Then Kubernetes will throttle the process & it will keep the Container running.

But if my Container is using that much of memory & it will try to exceed in that case, Kubernetes will kill that Container & restart that Container as per your restart policy

Demo :-

① cd to directory you have stored YAML file.

cd Demo-Yaml.

② vi resource-limit-pod.Yaml.

```
apiVersion: V1
Kind : Pod
metadata :
   name : frontend-1
Spec :
   Containers :
   - name : app
     image : alphine
     Command : ["sleep" ; "3600"]
     Resources :
       Requests :
         memory : "64mi"
         CPU : "250m"
```

③ To Execute Top to ~~Execute~~ Check the cpu available in cluster.

④ Kubectl apply -f resource-limit-pod.Yaml.

⑤ Kubectl get pods -o wide.

Page No.
Date

Section 36 (226 Chapter)

Monitor Containers in Kubernetes.

Manage Container Health Checks.

(a) why/what Container health.
(b) Liveness probe
(c) Startup probe
(d) Readiness probe
(e) Hands On Demonstration.

Container health:-

(1) Kubernetes is feature Rich, & provide number of features to monitor the Containers.

(2) Active Monitoring Helps K8s to decide the Container State & Auto Restart in Case of Container failure.

(1) Liveness probe :-

(1) The first health check type available in the Communities is Called Liveness probe.

(2) liveness probe will help to determine in the State of your Container.

(3) It will make sure that the Container is running or not.

→ Considered

④ By default K8s is down, if the Container process is stop, K8s restart Container Automatically. It is also depend on restart policy configured.

⑤ User can execute Two types of Liveness probes

· Run Command in Container :- Once you execute a

① Command with in a Container & if that Command status is successful, then K8s make sure that the Container is in the running state.

② Periodic HTTP Health Check :- if your Container is some kind of web application. Then you can put the HTTP health check within your Container, which will hit the health check stub API & Verify the status.

```
livenes probe:
  exec:
    command:
      - some command here -
  initialDelaySeconds : 5
  periodSeconds : 5
```

Initial Delay Seconds :- How long to wait before sending a probe after a Container starts.

Period Seconds :- How often a probe will be sent.

Page No.
Date

Liveness via HTTP Request manifest

```
livenessProbe:
  httpGet:
    path: /health.html
    port: 8080
    httpHeaders:
    - name: Custom-Header
      value: Awesome
  initialDelaySeconds: 3
  periodSeconds: 3
  timeoutSeconds: 1
```

STARTUP PROBE :-

① Setting up Liveness probe is very Trickly with Application which have Long Startup Time.

Sometimes we have some application that could be the DB application that could be some heavy Software which have a very long Startup time.

Generally the legacy Software have a very long Startup time & if you are going to put the liveness probe on that kind of Container which have the longest Startup time. Then you should be very precise about the initial time delay because you don't know that the Container will start in 10 minutes or 30 minutes.
Sometimes that prediction is not work.

So Startup probe will help us to Execute with the legacy application which has longest Startup time.

A Startup probe run at the Container Startup and Stop running Once Container Succes.

if a Startup probe is a part of your Container definition or Container manifest (xml) then Startup probe Execute Once & Stop Once it is Successful.

(2) Once the Startup probe has Passed Once, the liveness probe takes over to provide a fast response to Container deadlocks.

note:- if we have both Startup & liveness probe in manifest (xml), it Conside first Startup & if it is Successful it never execute Again, then it will Start with liveness probe.

(4) Once Container is Restorted., the very first time it will pick Startup probe will execute & the liveness probe.

Manifest or definitions of Startup Probe

```
StratupProbe:
  http Get:
    Path: /health.html
    Port: 8080
  failure threshold: 30
  Period Seconds: 10
```

Application will have a maximum of 5 minutes (30 * 10 = 300s) to finish its Startup.

Readiness Probe :-

1) Readiness is used to detect if a Container is ready to accept traffic

2) The running State of your Container doesn't guarantee that your application is ready to accept the traffic

for ex:-
Generaly the front end application Containers Come up very Quickly But backend application like Database take time to Comeup.

if we Start Sending the traffic on the front end Container, then the user will start getting the errors.
- frontend application will try to talk the backend application that is still down.

So although my Container is up that is ready to accept the traffic, but my Complete application, My applications as a whole is not ready to Consume the traffic.

in that Case we need another probe which is Called Readiness.

3) Readiness probe we Can define end to end health Check Status. which will verify your application End-to-end health Check.

if Readiness probe is a part of your manifest, then no traffic will be sent to a Pod or Container until the first Readiness probe will be Success.

Here is the manifest of the readiness problem.

Sometimes application might need to load large data or Configuration files during Startup, or depend on external services after Startup.

No Traffic will be sent to a pod until Container pass the Readiness Probe.

```
Readiness Probe:
  exec:
    Command:
    - cat
    - /tmp/healthy
  initialDelaySeconds: 5
  periodSeconds: 5
```

(1) Configuration for HTTP Readiness probes also remains identical to liveness probes.

(2) Readiness & liveness probes can be used in parallel for the same Container.

Section 36 (227 Lab: Liveness & Startup probe).

manifest for liveness proble.

```
apiVersion: v1
Kind: Pod
metadata:
  name: liveness-probe
Spec:
  Containers:
    - name: liveness
      image: k8s.gcr.io/busybox
      args:
        - /bin/sh
        - -c
        - touch /tmp/healthcheck; Sleep 60;
          rm -rf /tmp/healthcheck; Sleep 600
      LivenessProbe:
        exec:
          Command:
            - cat
            - /tmp/healthCheck
          initialDelaySeconds: 5
          PeriodSeconds: 5
```

Page No.
Date

① cd to directory where you have kept manifest (Yml files)

1) cd YmlScripts.

② vi liveness-probe.yml.

Copy the manifest written in Back page.
then Save & Close.

③

④ Kubectl apply -f liveness-probe.yml.

⑤ Kubectl get pods -o wide

⑥ Kubectl describle <name of the pod>

In manifest (Yml file) we have touch /Created the /tmp/healthcheck file and made to sleep for 60, and then remove the file & then sleep for 600.

In liveness probe we are looking for /tmp/healthCheck file periodicaly, As we have removed the file, after 60 second. liveness probe look for that file periodicaly after 5 second, if it not find the file, it will await for the delay time & restart the Container again.

By this we are Validating Success case & failure case.

Page No.
Date

manifest for liveness for http:

```
apiversion : v1
kind : Pod
metadata :
   name : liveness-probe-http
Spec :
   Containers :
     - name : liveness-nginx
       image : k8s.gcr.io/nginx
       livenessProbe :
         httpGet :
           Path : /
           Port : 80
         initialDelaySeconds : 3
         PeriodSeconds : 3
```

Repeat same steps from 1 TO 5
Note:- file name will be different.

To verify Execute Curl Command.

Curl <ip of container>.

Startup probe for http

```
apiversion: v1
Kind: Pod
metadata:
    name: liveness-probe-http
Spec:
  Containers:
    - name: liveness-nginx
      image: k8s.gcr.io/nginx
      StartupProbe:
        httpGet:
          path: /
          port: 80
      failureThreshold: 30
      PeriodSeconds: 10
```

① cd to the directory.

cd YmlScript

② Vi StratupScript.yml.

Copy the above manifest
Save & close

③ Kubectl apply -f StratupScript.yml.

④ Kubectl get pod -o wide

5. Kubectl describe <pod-name>

Wait. Period is Period x threshold.
10 x 30 = 300.

Section 36 [Lab 228]

Lab: Liveness & Readiness probe.

manifest :- for http. (it Contain both liveness & Readness)

```
apiVersion: v1
kind: Pod
metadata:
  name: hc-probe-http
Spec:
  Containers:
    - name: liveness-nginx
      image: k8s.gcr.io/nginx
      livenessProbe:
        httpGet:
          Path: /
          Port: 80
        initialDelaySeconds: 3
        PeriodSeconds: 3
      readinessProbe:
        httpGet:
          Path: /
          Port: 80
        initialDelaySeconds: 3
        PeriodSeconds: 3
```

Page No.
Date / /

Execution Steps are same only filename will be different & pod name will be different.

SECTION 36: [229]

SELF HEALING PODS IN Kubernetes.

Container Restart policies:-

The Container Restart policy is always a part of K8s.

Even if you are not defining the Container, Restart policy explicitly in your manifest, but given it is always have a default Container Restart policy.

1. Always
2. OnFailure
3. Never.

Restart policies :-

1. Kubernetes have Capability to Auto Restart the Containers when they fails.

2. Restart Policies Customize the K8s Container Restart behaviour & you can choose when to Restart the Containers.

   (Based On Business need, or need behaviour change in Restart).

(3) K8s have three Restart Policies - Always, onFailure & Never.

ALWAYS Restart policy :-

1) Always is default restart policy in K8s. if you are not defining restart policy explicitly within your pod manifest, whenever we are not defining any restart policy, always restart policy will be applicable on your pod & Container.

2) X with the always restart policy will be applicable on your pod & Container.

(3) in always restart policy, there is a drawback that it will always restart your Container even if the Container Completed Successfully.

(4) always restart policy, make the Container restart every time whenever the Container will stop or Container will exit successfully.

(5) always restart policy is only recommended for the Container that you want, that Container should always be in the Running state, but if you are creating some intermittent Containers which are specific to a particular job & after the job complete that Container will exit out. Always restart policy is definitely not recommended for these kind of Containers.

Page No.
Date

To Customize the behaviour :- we have another Restart policy called "ON FAILURE"

(1) On failure is only works if the Container process Exits Out with the error code.

So Suppose you design the Container & you want that Container will always Restart. if that Container Exits Out with failure Reason. or there could be any kind of failure in a Container & that Container will Exit Out.

So we can apply Onfailure.

(2) It also works if Container liveness probe determine the Container unhealthy. it will restart the Container.

We can use this policy on the application that needs to be run Successfully and then Stops.
So if we want some Containers that ~~S~~ Container will Execute Successfully & then it will Stop.

So there is a One time job of that particular Container we can apply the On failure Restart policy on that Container because due to any Reason. if Container will Exit Out due to any error in b/w and the job is not Completed. Then. Kubernetes will auto Restart that Container & it will Complete their job.

Page No.
Date

NEVER Policy :-

always : Restart.

Opposite of ~~OnFailure~~ Policy.

(1) Never allow Container to never restart even the Container liveness probe failed.

(2) use this for application that Run Only Once & never automatically restarted.

if you have a specific need that whatever the job I'm doing within a Container that is Only One time Executable, it doesn't matter that job Completed Successfully or not, but that job is just One time Executable.
we will apply the never Restart policy with that Particular Container.

LAB :- 230 Section 36.

Manifest :-

```
apiVersion : V1
Kind : Pod
metadata :
   name : Restart-always-pod.
Spec :
   RestartPolicy : Always.
   Container :
      - name : APP
        image : alpine
        Command : ["Sleep", "20"]
```

Page No.
Date

Execution Steps are same only difference is filename & podname. as mentioned manifest.

as we have given Sleep for 20 seconds our pod will exit then pod will be restarted automatically.

Kubectl get pod -o wide.

(2) On failure manifest

```
apiVersion: v1
Kind: Pod
metadata:
  name: Onfailure-always-pod.
Spec:
  RestartPolicy: OnFailure
  Containers:
    - name: app
      image: alpine
      Command: ["Sleep", "20"]
```

Execution Steps are same

Kubectl get pod -o wide.

To Verify the if it Restart or note, change the manifest Command Section.

Command: ["Sh", "-c", "Sleep 20", "dummy Comands"]

when it Execute dummy Command, it will exit with error

Page No.
Date

Never :-

Never manifest :-

```
apiVersion: v1
kind: Pod
metadata:
  name: never-always-pod
Spec:
  RestartPolicy: Never
  Containers:
    - name: app
      Image: alphine
      Command ["Sleep", "20"]
```

Execution steps are same

Section. 36 [231]

Creating multi Container Pods.

1. What is multi Container Pod.
2. Cross Container Communication.
3. multi Container Example.
4. Hands On..

Page No.
Date

Multi Container pods

(1) Kubernetes pods can have single or multiple containers in the pod.

till now we have seen single container in a pod.

(2) In multi container pods, containers share the resources like n/w and storage, also can communicate on Local Host.

(3) Note:- Best practice is to keep the containers in seperate pods, untill we would like containers will share the resources.

So if there is a need to share the resources b/w the containers, then definitely you can create the multiple container within a single port.

CROSS Container Communication.

(1) Container sharing a pod, can interact with shared resources.

So the first shared resources, the ~~Containers~~ N/w & Containers which are the part of single pod they can communicate to each other on any pod untill unless that pod is exposed to the cluster

The another communication medium is the storage containers can use the same shared Volume (storage) & they can

use the Shared data and they can interact using the Shared data.

Hands On Demo:

① Go To the directory we have yaml file (manifest) Stored.

② Vi multi-container.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: two-containers.
Spec:
  RestartPolicy: OnFailure
  Containers:
    - name: nginx-container
      image: nginx
      VolumeMounts:
        - name: Shared-data.
          mountPath: /usr/share/nginx/html.
    - name: debian-container
      image: debian
      VolumeMounts:
        - name: Shared-data
          mountPath: /pod-data
      Command ["/bin/sh"]
      args:
        ["-c", "echo Hello from the Second >/pod-data
                                          /index.html"]
  Volumes:
    - name: Shared-data.
      emptyDir: {}
```

Container path.

Container path.

Page No.
Date

SECTION 36 [232]

Container initialization in Kubernetes.

1. what/why of Container initialization
2. why it is Done in K8s.
3. Use Case of init Container.
4. Hand On Demon.

what is Container initialization in K8s.

1) init Containers are Specialized Containers that run. before a Container in your pod.

So whenever we are defining the application Container or Containers within my pod & if we are also defining the init Container ~~the~~ alongside with the Containers within my pod, then init Container will always execute before the Containers in a pod manifest.

2) the another feature about the init Container that the init Container will Only execute during the Startup process of your pod

(3) The init Container is designed to execute only once & they will only execute before the main Container or application Container of your pod.

(4) So we can use the init Container for the application setup.

if you want to setup something before the application Container, you can use the Concept of init Container in your K8s.

(5) init Containers can Contain the utility or setup a script that are not present in the image.

(6) User can define N Number of init Container in pod manifest file.

USE CASE :-

So with the help of init Container, we can setup the application right?

(1) The another benefit is if you want to install few packages. you want to get some library you can get this library within the in IT Container.

(2) So by the init Container we are basically controlling the start process of my app Container.

② Second Use case.

if we have DB Server as Backend & front-end application. we can put the Condition that DB should come up first. Successfully then go to front-end..

③ e.g. if my application container needs some data, then we can do that data setting job or data setup job by init Container.

Hands on [Lab 233]

① Go To directory where Yaml file in store.

② Vi init-container.yml:

```
apiVersion: V1
Kind: Pod
metadata:
  name: application-pod
Spec:
  Containers:
  - name: myapp- Container
    image: busybox:1.28
    Command ["sh", "-c", "Echo The app is running! && Sleep 3600"]
  init Containers:
  - name : init-myservice
    image: busybox:1.28
    Command:
      [
        "sh",
        "-c",
```

"until nslookup myservice.$(Cat /var/run/Secrets/
Kubernetes.io/Serviceaccount/namespace).
Svc.cluster.local; do echo waiting for
myservice; Sleep 5; done",

]

- name: init-mydb
image: busybox:1.28
Command:
[
"sh",
"-c",
"until nslookup mydb.$(Cat /var/run/Secrets/
Kubernetes.io/Serviceaccount/namespace).svc.
cluster.local; do echo waiting for mydb;
Sleep 5; done",
]

③ Main Container will not be deployed until init Containers is Successful.

④ as there are no Services Configured in Kluster / K8s init Containers will fail & will be in waiting State.

⑤ Let Configure Services ::

Manifest

```
apiVersion : V1
Kind: Service
metadata:
   name: myservice
Spec:
  Ports:
    - protocol : TCP
      Port : 80
      targetPort: 9376.

---

apiVersion : V1
Kind: Service
metadata
   name : ~~mysservice~~ mydb
Spec:
  ports:
    - protocol : TCP
      port: 80
      targetPort : 9377.
```

⑥ Vi myServicesYaml.yml.

Copy the above manifest

Save.

⑦ Kubectl apply - f <filename>

⑧ To Check init Container status.

Kubectl get - f < Yaml file name>