init containers:

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

if anyone ask in inetrview: we should ans yes we are using for checking the db connecttion, we have db and backend application.

before starting backend we have an init container that will check the db connection,

if connection is success then only the backend container will start. otherwise it wont start.

init containers

ephemeral volumes

emptyDir

hostPath

Liveness probe

readiness probe

Init containers

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

1. Init containers run before the main containers run. It can be one or many

2. Init containers should be completed one by one.

3. If init containers fail main containers will not run.

Init containers always run to completion.

4. to wait until other applications are ready, to prepare some configuration for main container.

DB and backend both are started at a time. backend is ready before DB started successfully. backend is getting failed because no connection to DB.

we can have init container for backend. so before backend starts init container should be checking if DB is ready or not.

docker best practices

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

1. without root user

2. use volumes

3. use official images

4. use small images, dont keep unnecessary installations

5. use multi stage builds

6. use health checks

7. use custom network

8. reduce imaging layers

9. use autoscaling

readiness probe

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

we can define when our container is ready.

for backend we can say when port 8080 opened, we can say

it is ready

liveness probe

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

health check

once container is ready, then only we can do health check

auto healing

liveness proble will happen frequently, whenever it finds

something is wrong it will auto restart the container.

ephemeral volumes

emptyDir

hostPath

static provisioning and dynamic provisioning --> perm volumes

ephemeral volumes

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

1. pod is running and generating log. does it generate log after terminated?

/var/log/nginx/access.log

/var/log/nginx/error.log

ephemeral volumes --> emptyDir

1. nginx --> main container

2. filebeat --> sidecar

you should somehow mount the volume/storage to both containers.

by default k8 considers volumes as emptyDir.

docker run -d -v /:/spam nginx

daemon set

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

daemonset will make sure a pod runs in every node. why this is useful?

we have to access underlying node logs and push them to elastic search for log monitoring. if we run deamonset in kubernetes,

it make sure a pod is automatically created when new node is added.

replicaset:

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

it will create multiple pods besed on what you have defined in the code.

ex: you have given 3 replicas and post that you have changed the image from nginx:latest to nginx:stable-otel

you have updated the file but replicaset wont update the pods to the lastest changes.

this in only responsible to maintain the number of pods which you have mentioned in the code.

if you deleted any of the pod intentionally or unintentionally automatically this replicaset will create a pod in the obesence of old one.

pod can update with your new code but there will be downtime to go and create a new pod.

deployment set:

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

you have given 3 replicas and post that you have changed the image from nginx:latest to nginx:stable-otel

you have updated the file but deployment set can update the pods to the lastest changes.

any changes in code first it will create 1 pod and then it will terminate the old one. at one point of time old and new pods will be available.

it will do the same process for defined replicas

deployment is a subset of replicaset

if you deleted any of the pod intentionally or unintentionally automatically this replicaset will create a pod in the obesence of old one.

deployment set:

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

used for stateless application i.e ususlly used for frontend and backend

statefulset:

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

used for stateful application. which should have some storage. this will give the pods numerical range

in statefulset:

1. pods are created with -0,-1 in statefulset, because statefulset should keep the identity

2. pods in deplyment are created at a time, but pods in statefulset will be created in orderly manner

\*\*\*statefulset must have headless service, what is headless service

a service which will not have cluster ip is called headless

Deployment vs Statefulset

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

1. stateful applications like DB

2. PV and PVC are mandatory for statefulset

3. Orderly provisioning of pods happens.

4. pods keep their identity like name.

5. We must create headless service.

6. every pod should have its own storage. so PV and PVC should be created for every pod.

1. Install EBS drivers

2. Check EC2 Role permissions

3. create storage class

4. create statefulset

mysql-mysql-0

pvc-name-statefulset-name-0

most impotant point for interview

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if deployment, nslookup of service gives, service IP address

if statefulset, nslookup of headless service gives us IP address of all pods.

1. PV --> representing physical storage

2. PVC --> Claiming the storage

3. SC --> automatic creation of volume and PV based on the claim from PVC

Daemon set:

---------

it make sure pod should run on each and every node.

if new node is added into cluster daemon set automatically spin up the pods in new node.

if node is deleted then daemon set will delete the pod.

its is useful for collecting server logs and pushing the logs for elastic search

ex: in your manifest file you have not given number of replicas

if you add daemon set in the manifest you have only 1 pod, still pod is getting created in number of nodes a/c to your count of nodes.

this is admin activity, have to find your pods in kube-system namespace

zeal vora-->aws security specialist

imagepullpolicy:

node have nginx:latest imagepullpolicy

you have updated the image as nginx:alpha

so if nodes should pull the changes each time means we have to select imagepullpolicy as always

Taint:

-----------

kubectl taint nodes ip-192-168-12-185.ec2.internal project=expense:NoSchedule

kubectl label nodes ip-192-168-12-185.ec2.internal project=expense

Tolerations are applied to pods. Tolerations allow the scheduler to schedule pods with matching taints.

Tolerations allow scheduling but don't guarantee scheduling: the scheduler also evaluates other parameters as part of its function.

Affinity:

affinity = like

1. schedule -- scheduler schedules the pod

2. execution --pod should run

preferredDuringSchedulingIgnoredDuringExecution --> soft

requiredDuringSchedulingIgnoredDuringExecution --> hard

ip-192-168-42-153.ec2.internal --> tained with project exepense

labelled also with project exepense

1. Node effinity

requiredDuringSchedulingIgnoredDuringExecution --> pod will not run even you set affinity.

we need to apply tolerations. So in this case until you apply toleration pod will be in pending state

preferredDuringSchedulingIgnoredDuringExecution --> scheduler tries the asked node,

but it cant schedule because tainted node. since it is only preferred it can schedule on to differnt node

2. Pod affinity:

K8-Architectre

-----

control plane/master component

node component

Master: have kube API server, kube scheduler, kube controll manager, etcd

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

1>this is main component.API server responding to all user requests.

it runs on port http 443

ex: when we execute kubectl get pods that request go to api server and this communicate with k8

s

2>responsible to schedule the pods. it will check multiple things while scheduling.

like taint, tolerances, affinity, underlying server memory, cpu consumption etc

3>controll manager:

1.replica controller component:>>replicaset>>is one of the component of control manager:if you delete the pod, it will automatically create the pod automatically

2. node controller: checks the nodes all the time, whether they are ready,able to schedule or not.

if any node is down it will make sure another node is created and shifts the pods to different node.

3. Service account controller: creates service account for every namespace created. basically under internal authentication.

4>etcd:

entire k8s cluster data in etcd. it is very important to take frequent backup.

if cloud, aws team will responsible for this.

Node components:

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

1. kubeproxy:-->on behalf of someone> it intercepts evry incoming and outgoing request of node. kube-proxy maintains network rules on nodes.

it runs on evry node.

2 .kubelet-->it works as a agent. it connects nodes to control plane.an agent that runs on each node in the cluster.

it pulls the info from control plane and runs the pods.

3. container runtime: it is on everynods. it basically runs the images into container.

containerd is the runtime usually.

add-ons:

kube-dns:provide dns to the prods. http://backend:8080. kubedns provides ip address to the service when one pod wants to connect with other pod.

networking plugins: VPC CNI(container network interface)default network inside eks.we have multiple nodes and this vpc cni gives the netork to the nodes.

Vm should have aws credetails:

1. aws configure

2. go iam and click on users copy the access key and secreat key

3. need to install eksctl

curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl\_$(uname -s)\_amd64.tar.gz" | tar xz -C /tmp

sudo mv /tmp/eksctl /usr/local/bin

eksctl version

4.need to install kubectl

curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.0/2024-05-12/bin/linux/amd64/kubectl

chmod +x ./kubectl

sudo mv kubectl /usr/local/bin

kubectl version --client

5. kubectl version

namespace: isolated project space where you can created resources related to your project

POd: pod is a smallest deploybale unit in kubernates.

Container is deploable unit docker.

Pod vs container:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

a pod containes multiple containers. container inside pod share seme network and storage.

kubernates services:

1. load balancer

2. service mesh(cluser ip , node port)

cluster ip:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. first create a pod with set of labels

2. create a service and give the same labels of which pod you want to attach to service.

3, create a pod and service.

if your use curl http://nginx it wont out side of cluster, but inside the cluster your on with ay pod able to acess the curl http://nginx.com/amazon-eks/1

nodeport:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. first create a pod with set of labels

2. create a service. for nodeport , you have to define under spec type:NodePort and give the same labels of which pod you want to attach to service.

3.if you want to go with custom port then have to give the cutomeport at end of your code.

4.create a pod and service.

5. check kubectl get pods -o wide for where the pod has been created in cluster nodes

6. go to that node and go to security gruop, edit the rules

7. select cutom tcp and enter the port number and expose to internet.

8.when you check with node public ip and port number , you can able to access it.

1.Pod:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

kind: Pod

apiVersion: v1

metadata:

name: nginx

labels:

name: frontend

project: expense

component: frontend

environment: dev

spec:

containers:

- name: nginx

image: nginx

2.cluserip:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

apiVersion: v1

kind: Service

metadata:

name: nginx

spec:

selector:

name: frontend

project: expense

component: frontend

environment: dev

ports:

- protocol: TCP

port: 80 #service-port

targetPort: 80 #target-port

#if your mentioning only service and attaching the service to pod then

this is called cluster ip, here pods are communicating only inside cluser through cluser ip

3. Nodeport:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

apiVersion: v1

kind: Service

metadata:

name: nginx-np

spec:

type: NodePort

selector:

name: frontend

project: expense

component: frontend

environment: dev

ports:

- protocol: TCP

port: 80 #service-port

targetPort: 80 #target-port

nodePort: 32760

session 58/59

kubernetes volumes:

2 types of volumes

EBS:Elastic blob storage

EFS:elastic file system

google drive works on NFS protocol. (network file sharing)

1. static provisioning

2. dynamic provisioning

static provisioning

\_\_\_\_\_\_\_\_\_\_\_\_

EBS:(elastic block storage)

1.

volume create at 10 mins

need to install csi drive in the cluster, as cluster shuold connect for ebs volume know

2.

kubectl apply -k "github.com/kubernetes-sigs/aws-ebs-csi-driver/deploy/kubernetes/overlays/stable/?ref=release-1.32"

kubectl get ns

kubectl get pods -n kube-system (you must see the admin details, as csi controller info post installing csi drivers)(kube system have admin things)

3.

your nodes should have access to connect with ebs volume. ebscsi policy need to attach to the eks cluster nodes

4.

wrappers:

you can represent ebs volume inside k8s with a resource called "persistent volume". this is eqivalent to ebs storage inside k8 cluster. this is admin level

PVC:

persistent volume claim. Pods should claim i want the volume.pods can claim the volume through resource called pvc.this is user level

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

important note: if you stopped the cluster and when you craete again the cluster nodes will change and the nodes have not the of the role is ebscsidriver policy.

which we have to attach the policy again whenever nodes get changes.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

k8s-access modes:

read-write-once: we cant connect hard disk to multiple devices at a time, similarly ebs also works as hard disk so access mode read-write-once

Dynamic provisioning:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1.

volume create at 10 mins

need to install csi drive in the cluster, as cluster shuold connect for rbs volume know

2.

kubectl apply -k "github.com/kubernetes-sigs/aws-ebs-csi-driver/deploy/kubernetes/overlays/stable/?ref=release-1.32"

3.

someone on behalf of you should create volume and equivalent pv in k8s. this can be done by storage class in k8s

retain, recycle, delete

4.

static provisioning:

kid-->mother-->father-->money

pod-->pvc-->pv-->storage

dynamic provioning:

kid-->mother-->e wallets -->money

pod-->pvc-->storage class --> storage

ebs vs efs

\_\_\_\_\_\_\_\_\_\_\_\_

1. ebs is like hard disk, volume and instance should be in one AZ

2. efs is through nfs protocal , it can be anywhere

3. ebs volume is static, it cant increase by its won

4. efs is completely dynami, it can automatically expand. which is suitable for DB applications like mysql. ebs have less latency.

ebs is good for db like mysql as data no need to travel , volume and pod will be same network or AZ.

efs is good for images and videos

EFS:

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

a. Static provisoning:

1. need to create efs volume in aws account

2. allow ec2 instaces on nfs protocol in efs sg

( means have to copy the cluster nodes sg id and add the inbound rule of your efs sg on nfs protocol and paste your sg id)

3. install drivers , efs drivers

4. check iam access on ec2 instances

installation of efs driver:

kubectl kustomize \

"github.com/kubernetes-sigs/aws-efs-csi-driver/deploy/kubernetes/overlays/stable/ecr/?ref=release-2.0" > private-ecr-driver.yaml

kubectl apply -f private-ecr-driver.yaml

5. add the policy of efscsi

LB:>node on particular port-->node should allow traffic from LB --> nodeport -->cluster IP-->Pod

b. dynamic provision :

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

1. installi efs csi drivers

2. check ec2 nodes shouls have efs csi role permission

3. storage EFS volume

4. check efs sg allowing ec2 sg

5 create sc

implementing expese with volumes\_\_\_\_\_\_\_\_\_

1. install ebs csi drivers

2 . ec2 nodes permisions

Admin team created sc and name space, drivers inastalled and permissions given to node.

session 58/59

kubernetes volumes:

2 types of volumes

EBS:Elastic blob storage

EFS:elastic file system

google drive works on NFS protocol. (network file sharing)

1. static provisioning

2. dynamic provisioning

static provisioning

\_\_\_\_\_\_\_\_\_\_\_\_

EBS:(elastic block storage)

1.

volume create at 10 mins

need to install csi drive in the cluster, as cluster shuold connect for ebs volume know

2.

kubectl apply -k "github.com/kubernetes-sigs/aws-ebs-csi-driver/deploy/kubernetes/overlays/stable/?ref=release-1.32"

kubectl get ns

kubectl get pods -n kube-system (you must see the admin details, as csi controller info post installing csi drivers)(kube system have admin things)

3.

your nodes should have access to connect with ebs volume. ebscsi policy need to attach to the eks cluster nodes

4.

wrappers:

you can represent ebs volume inside k8s with a resource called "persistent volume". this is eqivalent to ebs storage inside k8 cluster. this is admin level

PVC:

persistent volume claim. Pods should claim i want the volume.pods can claim the volume through resource called pvc.this is user level

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

important note: if you stopped the cluster and when you craete again the cluster nodes will change and the nodes have not the of the role is ebscsidriver policy.

which we have to attach the policy again whenever nodes get changes.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

k8s-access modes:

read-write-once: we cant connect hard disk to multiple devices at a time, similarly ebs also works as hard disk so access mode read-write-once

Dynamic provisioning:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1.

volume create at 10 mins

need to install csi drive in the cluster, as cluster shuold connect for rbs volume know

2.

kubectl apply -k "github.com/kubernetes-sigs/aws-ebs-csi-driver/deploy/kubernetes/overlays/stable/?ref=release-1.32"

3.

someone on behalf of you should create volume and equivalent pv in k8s. this can be done by storage class in k8s

retain, recycle, delete

4.

static provisioning:

kid-->mother-->father-->money

pod-->pvc-->pv-->storage

dynamic provioning:

kid-->mother-->e wallets -->money

pod-->pvc-->storage class --> storage

ebs vs efs

\_\_\_\_\_\_\_\_\_\_\_\_

1. ebs is like hard disk, volume and instance should be in one AZ

2. efs is through nfs protocal , it can be anywhere

3. ebs volume is static, it cant increase by its won

4. efs is completely dynami, it can automatically expand. which is suitable for DB applications like mysql. ebs have less latency.

ebs is good for db like mysql as data no need to travel , volume and pod will be same network or AZ.

efs is good for images and videos

EFS:

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

a. Static provisoning:

1. need to create efs volume in aws account

2. allow ec2 instaces on nfs protocol in efs sg

( means have to copy the cluster nodes sg id and add the inbound rule of your efs sg on nfs protocol and paste your sg id)

3. install drivers , efs drivers

4. check iam access on ec2 instances

installation of efs driver:

kubectl kustomize \

"github.com/kubernetes-sigs/aws-efs-csi-driver/deploy/kubernetes/overlays/stable/ecr/?ref=release-2.0" > private-ecr-driver.yaml

kubectl apply -f private-ecr-driver.yaml

5. add the policy of efscsi

LB:>node on particular port-->node should allow traffic from LB --> nodeport -->cluster IP-->Pod

b. dynamic provision :

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

1. installi efs csi drivers

2. check ec2 nodes shouls have efs csi role permission

3. storage EFS volume

4. check efs sg allowing ec2 sg

5 create sc

implementing expese with volumes\_\_\_\_\_\_\_\_\_

1. install ebs csi drivers

2 . ec2 nodes permisions

Admin team created sc and name space, drivers inastalled and permissions given to node.