Deployment manages the replicaset - > replicaset manages the pod - > pod is an abstraction of Container

Yaml file configuration:

1. Metadata : name of the component
2. Specification : every kind of activities we want to apply for this component
3. Template(blueprint for pods): Template comes under Specification - > template also has its metadata and specification . so we can say template is configuration inside configuration
4. Status - > automatic by k8s by the help of etcd database which keeps the information for k8s component

Q. How deployment and pod will get connected ?

Ans: in the same deployment.yaml file - > we have metadata for deployment which gives name and labels ( labels is used to matched the deployment and pod in deployment.yaml file) - > like wise we have specification and under specification we have template which defines the pod and its specification - > now under template we have Matchables - > which match the deployment’s label’s key-value pair in specification - > at the same time we can find labels and key value pair under template .

= > Pods get the label through the template blueprint

= > This label is matched by selector

Note : Metadata part contains labels = > and specification part contains selectors . label is sticks to Kubernetes component , in this case template’s metadata for pod and upper part metadata for deployment . so label’s under both component is same key value pair . meaning this deployment is attached with this pod . how deployment knows that pod is attached ? Ans: we will provide matchable under selectors under specification of deployment which confirms or connect pod to deployment .

##################Deployment sample###############

|  |  |  |
| --- | --- | --- |
| Throws error if we did not match the matchable with template labels | Deployment labeles can be different but selectors is responsible to connect pod with deploy(better to put same label for all) | Service.yaml - > now this service will connect with deployment and its pods by its selectors |
| apiVersion: apps/v1  kind: Deployment  metadata:  name: nginx-deployment  labels:  app: nginx  spec:  replicas: 3  selector:  matchLabels:  app: nginx1  template:  metadata:  labels:  app: nginx  spec:  containers:  - name: nginx  image: nginx:1.14.2  ports:  - containerPort: 8080 | apiVersion: apps/v1  kind: Deployment  metadata:  name: nginx-deployment  labels:  app: nginx (this can be any name)  spec:  replicas: 3  selector:  matchLabels:  app: nginx1 (matched to template labels)  template:  metadata:  labels:  app: nginx1  spec:  containers:  - name: nginx  image: nginx:1.14.2  ports:  - containerPort: 8080 | apiVersion: apps/v1  kind: Service  metadata:  name: nginx-service  spec:  selector:  app: nginx1  ports:  -protocol: TCP  Port: 80 (this is for service itself)  targetPort: 8080 (this is for pod)  note: selectors should be matched with selectors of deployment)  if deployment selectors and service selectors does not match then service can not map to deployment and pod . And can not able to forward the external servie to pod) |

= > Complete Demo project

mongoDb and mongoExpress

how it works:

1st create mongoDb pod - > create Internal service (no external request) only component in same cluster can talk not out sider - >

2nd - > create Mongo express pod - > we need to have mongoDb credentials - . it can be done by Env variable (deployment.yaml file)

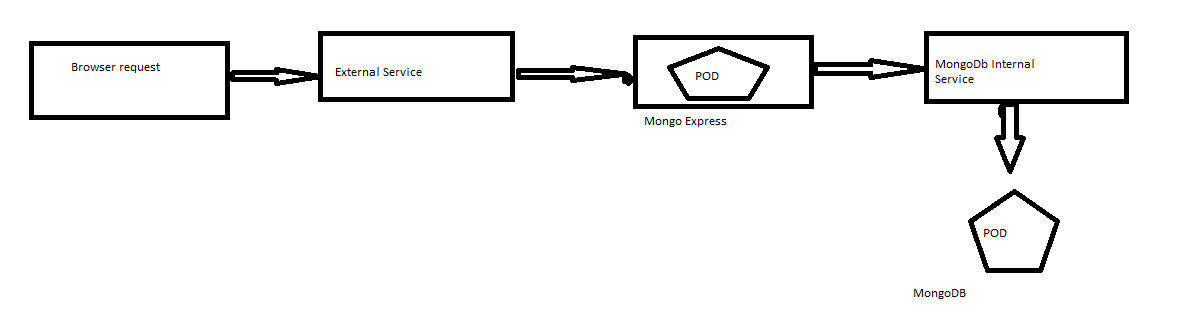
--- -- > we create ConfigMap - > it holds DB url

--- -🡪 We create Secret for credentials

----- > we will give reference of both ConfigMap and Secret to deployment.yaml file

Requirement: after doing this we need to have access mongodb express from web browser for this -- > we will create external Service which will allow web browser talk to mongo express - > url for this will be - > IP address of Node and Port of external Service

Flow :



Nodeport must be between : 30000 to 32767