**Kubernetes**

Features - availability, scalibility or high performance, disaster recovery

Basic components-

1. Node and pod -

* Node is simple server,physical or virtual machine.
* Pod - smalledst unit of k8s. abstraction over container. Creates layers or running environment on top of container.so k8s can replace the container technology(eg. Docker) without interacting with it. **Pod is meant to run one application container inside of it. Multiple containers when there is one main container and other side containers are there.**
* Each pod gets **internal** ip adress and container doesn’t get any ip, pods communicate through ip.also through that ip pods communicate with database.
* Pods failure is common so when it happens database also fails/crashes. And when the new db craeted new ip creates and it is not easy to work with new ip. So to solve this **service** is used.

1. Service and ingress:

* Service- permanent ip address. Service knows which pod to commnicate with by identifying the labels of the pod.
* pod and db will have their own service.lifecycle of service and pod not connected.
* Ingress - the request first goes to ingress and then to service.Ingress acts like a smart router: it receives external HTTP(S) requests and forwards them to the appropriate internal services based on rules you define. It allows for host-based and path-based routing (e.g., example.com/app1 to Service A, example.com/app2 to Service B).

1. Configmap and secret:

* Configmap - has external configuration of the app like services or urls of db. Solves the problem of url change of db. So at time of change just change it in config map and then the pod willg get this data as usual. No need to re-build the application.
* **don’t put credentials in configmap**
* Secrete - used to store the credentials or secrete data in base 64 encoded.

1. Volumes -

* if the pod restart the data will vanish, so it uses volumes. Volumes attaches physical storage on your harddrive to pod. That storage could be - local, remote storage( cloud). **k8s doesn’t manage data persistance, it’s the users job to do that**

1. Deployment and staefulset- blueprint for pods

* **Replication - allows to aviod downtime when pod dies or some another issue.**
* In practice you work with deployments not with pods. In deployments you can specify no of replicas. Abstraction of pads( layers over pods).
* Cant replicate db using deployment.**statefulset -** take cares of inconsistancies of reading and writing the data through replicas of pods to the data storage.
* Db are hosted ususally out of k8s as it is difficult to work with deployment and statefulset together.
* **Nodes-** does the actual work. Contains multiple pods.

3 processes must be installed on every node -

1. Container runtime - eg. Docker
2. Kubelet - interacts with container runtime and node. It starts the node with container inside. Assigning resources to from node to container.
3. Kube proxy - responsible to forward requests from services to pods.

* Master node -

1. Api server - cluster gateway. Gatekeeper of authentication.
2. Scheduler - start the application pod in one of the worker nodes.it knows where to put the new pod in which node as per the resource utilization of the node.
3. Control manager - detect the died pod, cluster state changes, tries to recover the cluster state. For this makes requsets to scheduler.
4. Etcd - key value store(cluster brain) . saves cluster changes.

**Minikube and kubctl demonstration and practice-**

Aftter using the command - start minikube it shows warning as - ! Failing to connect to https://registry.k8s.io/ from inside the minikube container

This means minikube is not able to connect to k8s registry from container within which makes difficult to pull the images, but docker resolves that issue. But in future instead of docker other service like helm will also try to pull the images but if it fails pre pull the required images in minikube like this -   
 minikube image load nginx

minikube image load redis

minikube image load busybox

* Creating pods - actually you don’t create the pods directly you interact with the abstraction layer named ‘deployment’.

To create deployment you need image. For now lets use nginx image I.e. nginx deployment.

>>Kubctl create deployment NAME --image=image [--dry-run] [options]

>>Kubectl create deployment nginx-depl --image=nginx

Output of above command - deployment.apps/nginx-depl created

>>Kubectl get deployment

NAME READY UP-TO-DATE AVAILABLE AGE

nginx-depl 0/1 1 0 34s

Its not ready yet

>>kubectl get pod  
  
 NAME READY STATUS RESTARTS AGE

nginx-depl-68c944fcbc-zjw84 1/1 Running 0 14m  
  
now both deployment and pod are ready

**Replica set**

**>>kubectl get replicaset**

**NAME DESIRED CURRENT READY AGE**

**nginx-depl-68c944fcbc 1 1 1 20m**

In development no need to do the replicaset, as it is managed by the kubernetis. You will interact with deployment and below that everything will be taken care by kubernetes.

>>**kubectl exec -it mongo-dpl-77d6b5dd-xskwv -- bin/bash** - gets the terminal of the container inside of that pod whose name is specified.this lets you see the inside data of the container.

>>kubectl delete deployment mongo-dpl

deployment.apps "mongo-dpl" deleted

>>kubectl apply -f config-file.yml  
kubectl will execute the actions written in that file, like - creating deployment if the metadata name is different otherwise it will update the exisiting one.

**Yaml configuration file in kubernetes**

* **3 parts** of k8s confg file -

1. Metadata - name of the component
2. Specification - configuration to be applied for that component
3. Status - automatically generated and edited by kubernetes. Kubernetes will compare the desired and actual state of the component, if deosnt match it will take the necessary actions. Its called self healing factor of kubernetes. Kubernetes gets the status information from etcd.

* Template for pods in YAML - inside the spec of the deployment

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.16

ports:

- containerPort: 8080

This is the confg file for pod. Image of the pod, name of the pod and what at port

* Connecting components - labels and selectors and ports
* Metadata contains labels and spec contains selectors . labels give key value pair to component like deployment, pod. Eg. app: nginx
* Deployment knows which pods belongs to it with the help of selector as it builds connection between those two. The selector tell deployment “Watch and manage only the Pods that match this label — but only if I created them.”
* In service.yml configuration port is where the service component is accessible and target port is the port of container to which the service forwards the request.
* To debug the yaml file -

>>kubectl get deployment nginx-deployment -o yaml

It shows the status form etcd.

* >> kubectl delete -f .\nginx-deployment.yaml  
  deletes the yaml file and corresponding deploment.
* >> kubectl get all - shows all running components

**Complete application setup with kubernetes components**

Mongo express and mongo db

1. Creating mongo db pod. For that need to create an internal service. So only the components in the same cluster can talk to it.
2. Creating mongo express deployment- needs two things

* db url to connect to db
* db user and password to authenticate.

And we can pass these data to express through the confg file i.e. deployment.yaml

Creating config map for db url and secrete for credentials and then referecing both in yaml file.

1. After this setup we need mongo express to accessible through browser. For that need to create external service that will allow external request to talk to pod. URL contains - ip address of node and port of the external service.

Browser request flow - mongo express browser -> mongo express external service -> mongo express pod -> mogo db internal service -> mongodb pod -> secrete for authentication

Step-1) create mongo db deployment -

Note - to copy text from vim editor - click v to enter visual mode -> select the text -> type “+y -> then paste it anywhere you like

To paste into vim - open vim editor -> go to the desired line -> type “+p

apiVersion: apps/v1

kind: Deployment

metadata:

name: mongodb-deployment

labels:

app: mongodb

spec:

replicas:1

selector:

matchLabels:

app: mongodb

template:

metadata:

labels:

app: mongodb

spec:

containers:

- name: mongodb

image: mongo

ports:

- container: 27017

env:

- name: MONGO\_INITDB\_ROOT\_USERNAME

value:

- name: MONGO\_INITDB\_ROOT\_PASSWORD

Value

Step-2) creating secret component for the credentials of db - we have to create secret before the deployment if we are referencing it in deployment confg file.

The credential values arent plain text but base 64 encoded. So convert thos into base64 -  
[Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes("username"))  
[Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes("pass123"))

apiVersion: v1

kind: Secret

metadata:

name: mongodb-secret

type: Opaque

data:

mongo-root-username: dXNlcm5hbWU=

mongo-root-password: cGFzczEyMw==

>> kubectl get pod -o wide

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

mongodb-deployment-6d9d7c68f6-qmvcr 1/1 Running 0 29m 10.244.0.12 minikube <none> <none>

Step-3) apply the secret file - kubectl apply -f .\mongo-secret.yaml

Create reference of the secret file inside the depl yaml file -

env:

- name: MONGO\_INITDB\_ROOT\_USERNAME

valueFrom:

secretKeyRef:

name: mongodb-secret

key: mongo-root-username

- name: MONGO\_INITDB\_ROOT\_PASSWORD

valueFrom:

secretKeyRef:

name: mongodb-secret

key: mongo-root-password

Apply the deploy.yaml - kubectl apply -f .\mongodb\_deployment.yaml

Note - for debugging of pod use >> kubectl describe pod pod name

step-4) create internal service configuration - so other components can talk to mongodb pod

Usually the service confg is included in same deploy.yaml file

In yaml file --- represents separate docunment starting

---

apiVersion: v1

kind: Service

metadata:

name: mongodb-service

spec:

selector: helps to connnect the service to the pod

app: mongodb

ports:

- protocol: TCP

port: 27017 service port

targetPort: 27017 container or pod port

The service port and target port can be different.

>>kubectl describe service - check the service configuration

>>>> kubectl get pod -o wide - check the ip address is same in both ervice ocnfig and pod

Step-5) creating mongo express deployment and service. and external configuration where the url of db will be added.

Mongo express requires -

1. Which database to connect? - mongodb address/ internal service. env variable is mongodb\_server
2. Which credentials to authenticate? env variables - ME\_CONFIG\_MONGODB\_ADMINUSERNAME, ME\_CONFIG\_MONGODB\_ADMINPASSWORD

apiVersion: apps/v1

kind: Deployment

metadata:

name: mongo-express

labels:

app: mongo-express

spec:

replicas: 1

selector:

matchLabels:

app: mongo-express

template:

metadata:

labels:

app: mongo-express

spec:

containers:

- name: mongo-express

image: mongo-express

ports:

- containerPort: 8081

env:

- name: ME\_CONFIG\_MONGODB\_ADMINUSERNAME

valueFrom:

secretKeyRef:

name: mongo-secret

key: mongo-root-username

- name: ME\_CONFIG\_MONGODB\_ADMINUSERPASSWORD

valueFrom:

secretKeyRef:

name: mongo-secret

key: mongo-root-password

- name: ME\_CONFIG\_MONGODB\_SERVER

value:

Step-6) creating a configmap file in which the url of the db will be there so that all the components can use it.

apiVersion: v1

kind: ConfigMap

metadata:

name: mongodb-configmap

data:

database\_url: mongodb-service

>>kubectl apply -f mongo-config.yaml

Step-7) adding environmental variables inside mongoexpress.yaml  
 env:

- name: ME\_CONFIG\_MONGODB\_ADMINUSERNAME

valueFrom:

secretKeyRef:

name: mongodb-secret

key: mongo-root-username

- name: ME\_CONFIG\_MONGODB\_ADMINUSERPASSWORD

valueFrom:

secretKeyRef:

name: mongodb-secret

key: mongo-root-password

- name: ME\_CONFIG\_MONGODB\_SERVER

valueFrom:

configMapKeyRef:

name: mongodb-configmap

key: database\_url

Step-8) >>kubectl logs mongo-express-76787747bf-7kv2m

Waiting for mongodb-service:27017...

No custom config.js found, loading config.default.js

Welcome to mongo-express 1.0.2

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

Mongo Express server listening at http://0.0.0.0:8081

←[31mServer is open to allow connections from anyone (0.0.0.0)←[39m

←[31mbasicAuth credentials are "admin:pass", it is recommended you change this in your config.js!←[39m

Step-9) access mongo express from a browser. For that external service is needed-

Same as intertnal service this external servcice usually created in mongoexpress-deployment.yaml

apiVersion: v1

kind: Service

metadata:

name: mongo-express-service

spec:

selector:

app: mongo-express

type: LoadBalancer

ports:

- protocol: TCP

port: 8081

targetPort: 8081

nodePort: 30000

Type : loadbalancer - assigns service an external ip address and so accepts externla requests

Node port - must be between 30000 - 32767

Port for external ip address to open

Port you need to put into browser

>>kubectl get service  
mongo-express-service LoadBalancer 10.111.21.159 <pending> 8081:30000/TCP 12s

mongodb-service ClusterIP 10.107.183.215 <none> 27017/TCP 6h9m

Here the mongodb-service has type of cluster ip which is default for internal service. The difference is cluster ip will give the service an internal ip add, loadbalancer will also give service an internal ip add but also an external ip add

Step-10) >>minikube service mongo-express-service

|-----------|-----------------------|-------------|---------------------------|

| NAMESPACE | NAME | TARGET PORT | URL |

|-----------|-----------------------|-------------|---------------------------|

| default | mongo-express-service | 8081 | http://192.168.49.2:30000 |

|-----------|-----------------------|-------------|---------------------------|

\* Starting tunnel for service mongo-express-service.

|-----------|-----------------------|-------------|------------------------|

| NAMESPACE | NAME | TARGET PORT | URL |

|-----------|-----------------------|-------------|------------------------|

| default | mongo-express-service | | http://127.0.0.1:49706 |

|-----------|-----------------------|-------------|------------------------|

\* Opening service default/mongo-express-service in default browser...

! Because you are using a Docker driver on windows, the terminal needs to be open to run it.

This shows the NodePort URL where your service is exposed:

http://192.168.49.2:30000

This is the Minikube VM's IP (since Minikube runs inside a VM or container).

Port 30000 is the NodePort that maps to the internal container port 8081.

You could open this in your host browser, but only if Minikube is running with a VM driver that allows direct IP access (e.g., VirtualBox or Hyper-V). In your case, you're on Docker for Windows, which doesn't allow this kind of direct access.

Since you're on Docker driver (on Windows), Minikube starts a tunnel — it forwards traffic from a localhost port (here, http://127.0.0.1:49706) to the internal Minikube service.

✅ This URL works in your browser: http://127.0.0.1:49706

It’s a temporary tunnel — it only works while your terminal is open and running this command.

Step-11) in the browser when you create a database the process is as follows -

Mongo express external service -> mongo express pod -> mongodb internal service -> mongo db pod

**Namespace**

Virtual cluser inside kubernetes cluster. Kubernetes gives bydefault namesapces at the time of clustre creation.

>>kubernetes get namespace

1. kube-system - shouldn’t modify anything in it
2. kube-public - contains publically accessible data - configmap
3. Kube-node-lease- heartbeat of nodes. Each node has associated lease object in namespace. Determines the node availability
4. Default - resource you create are located here. It is use to create resources at the begging if you havent create any namespace.

* Need of namespace-

1. Imagine you have only default namespace and you create your resources in that namespace. So for multiple resources you can create different namespcaes. Grouping resources in namespaces inside the cluster.
2. Many teams (conflict)- if two teams working on same application then it will create conflict so creating different namespace for the application.
3. Resource sharing - different production verions of application can use same nginx-ingress controller and elastic stack
4. Each team has their own isolated environment in terms of name space. So none can interfere others environment and also there can be limitation introduced for the resources of each namespace

Characteristics of namespace-

1. Each namespace must define its own configmap and secret. Resource that can be shared is service.
2. Components cant be created in namespace, they live globally in cluster - volume, node  
   >>kubectl api-resources --namespace false

* Creating components in namespace -

>> kubectl apply -f configmap.yaml --namespace=my-namespace

**Or**

provide namespace in yaml file -

apiVersion: v1

kind: ConfigMap

metadata:

name: mysql-configmap

namespace: my-namespace

data:

db\_url: mysql-service.database

But for this need to create namespace first- >> kubectl create namespace my-namespace

>> kubectl get configmap -n my-namespace- because the configmap is in specified different namespce now and not in the default namespace of the kubernetes.

* To use kubens in powershell -

Go to: https://github.com/ahmetb/kubectx/releases

Download kubens.exe from the latest release.

Place it in a folder in your system PATH (like C:\Windows\System32 or add a custom folder to PATH).

* Kubens - tool that hleps to change the active namespace

**Namespace in metadata of yaml file should be same in which the service and pod are running.**

**Ingress**

Ingress helps to give the external ip address a proper domain name for the application and secure connection through which the user interacts with the cluster.

The request wil come to ingress then to internal service then to the application pod.doesn’t contain any external service. You would not open the application on ip address and port. You will have my-app ingress for that.

apiVersion: networking.k8s.io/v1beta1

kind: Ingress

metadata:

name: myapp-ingress

spec:

rules:

- host: myapp.com

http:

paths:

- backend:

serviceName: myapp-internal-service

servicePort: 8080

It has routing rules -

Forward requests to internal service - serviceName: myapp-internal-service

host: myapp.com - this is what user enters in browser

Host should be valid domain name and you should map that to the ip address of the node which is the entrypoint

In addition you also need implementation of ingress which is **ingress controller**. IC is another pod running in cluster, which evaluaes the rules in the cluster related to ingfress and redirection.

To install inginex ingress controller on powershell -   
kubectl apply -f <https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.12.2/deploy/static/provider/cloud/deploy.yaml>

>>minikube addons enable ingress - automatically starts the k8s nginx implementation of ingress controller

ingress-nginx-controller-67c5cb88f-pwzh7 1/1 Running 0 20h

When the controller is running Now can create ingress rule the controller can evaluate

* Creating ingress for the mongo exrpess service -

1. Create yaml file for ingress

Express-ingress.yaml -   
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: mongoexpress-ingress

  namespace: default

  # annotations:

  #   nginx.ingress.kubernetes.io/rewrite-target: /

spec:

  rules:

  - host: mongo-express.com

    http:

      paths:

      - path: /

        pathType: Prefix

        backend:

          service:

            name: mongo-express-service

            port:

              number: 8081

>>kubectl get ingress

NAME CLASS HOSTS ADDRESS PORTS AGE

mongoexpress-ingress nginx mongo-express.com 192.168.49.2 80 2m40s

This steps helps browser to know where to send the traffic when the mongo-express.com is visited.

192.168.49.2→ Entry point on your local machine where Ingress is listening (via minikube tunnel)

The ingress controller will evaluate that request as per the rules defined in Express-ingress.yaml

mongo-express-service → Internal Kubernetes service that gets the request forwarded by Ingress

First check the service type of ingress controller,   
>>kubectl get svc -n ingress-nginx  
 for minikube tunnel to work and expose your ingress externally, the service must be of type LoadBalancer — because minikube tunnel is designed to provide external IPs for LoadBalancer services in local clusters. You may see the type as node port or different. In order to change it inload balancer patch it -   
>> kubectl patch svc ingress-nginx-controller -n ingress-nginx --type='json' -p="[{'op':'replace','path':'/spec/type','value':'LoadBalancer'}]"

After this outout of >>kubectl get svc -n ingress-nginx  
ingress-nginx-controller LoadBalancer 10.106.45.63 127.0.0.1 80:31901/TCP,443:30871/TCP 23h  
  
then add the external ip into host file C:\Windows\System32\drivers\etc\hosts - 127.0.0.1 mongo-express.com

Browser (127.0.0.1 mongo-express.com)

↓

127.0.0.1 (minikube tunnel forwards to LoadBalancer svc of ingress-nginx-controller)

↓

ingress-nginx-controller service (ClusterIP 10.106.45.63)

↓

ingress-nginx pod

↓

Ingress rule: routes to mongo-express-service

↓

mongo-express-service (ClusterIP)

↓

mongo-express pod (10.244.0.20)

containers:

      - name: kubernetes-test-app

        image: kubernetes-test-app:latest

        imagePullPolicy: Never  # Tell Kubernetes not to pull from a registry

        # image: vikash95/kubernetes-test-app:latest

In this imagepullpolicy: never will tell the kubernetes to not to search for the image in external repository. So the image will be searched in the local storage only.   
In case image is in another external repository then use the 2nd option which is commneted and comment the forst image option and imagepullpolicy