Kubernetes

What it is?

It is the most popular container orchestration tool now days.

- If the there are 100’s of container run in a one environment it hard to manageable.
- Using orchestra with some automatic feature to help.
- That’s where container orchestra comes.
- It help to manage the container.
    - Run the application as container.
    - Find the existing layers for container from other images.
    - Deploy the containers into server,laptop,cloud.

What problem it solve:

- Rise of the Micro services the needs of the containers needs get peak.
    - **Micro-services :** It is a tool that run single application in a single isolated environment called container.
- Then manage those containers via scripts not possible at all.

It offer :

- Manage all the containers with the high availability.
- High response.
- Disaster Recovery.

Architecture

- There is at-least one **Master** node In a kubernetes **cluster.**
- Can have more than one Master.
    - Inside this there are several process running to manage the cluster.
        - **API Server** → It is the entry point to the cluster.

        - **Controller Manager** → It track the cluster what happening on it.
        - **Scheduler** → It ensures the pods placement and scheduling the container based on the workload.
        - **etcd** → key - value storage It contain all the current state of the container inside the node.
    - **virtual network** → It is the process that help to connect the worker nodes to the master node.
        - It create the individual nodes into a single machine.
- And many **worker** node in a cluster.
    - **kubelet** → It is a process that run inside every worker node to manage the cluster connections.
    - Inside the worker node the containers are running.

Pods,Service & Ingress

Pod

In a node there is a smallest unit in k8s .

- It is the layer of the container.
- Inside the pod can run more than one container.
- In K8s it only interact with kunbernetes layers.

Each pod have the **IP Address** it connect with the ip address inside the application. the container can die easily.

That where service comes,

services

- It create the **static IP** address for the each pod.
- It not connected to the life cycle of the pod.
- For application [external_Service] → `http://node_ip:port_number` .
    - It difficult to connect the url with ip address every time.
    - then ingress come. for secure protocol and domain name.

- For database [internal_service] → `http://app_ip:db_ip`.

Ingress

It just folded to the **service ip** address with the **domain name url ,**

If the connection is established it will goes with the ingress then service.

example : http://my-app.com → It is ingress.

http://128.9.9.0:7070 → It is service url.

ConfigMap

It is the component that help to config the application build file.

Secret

It is same as the configmap but it only store the credentials like password.

It is base-64 encoded format.

Volumes

It is a component that help to store the cluster data on hardware.

- Because K8s doesn't care about the data inside the cluster.
- So as a developer we need to persist the data on external drives it will remote or local.

Deployment

What if the application die inside the cluster. so the user can't connect the application

> that's why the cluster will cloned in many servers it is reachable through the **service ip**

It is responsible for the cloning the application pods.

- **It create the blue print for the pod .**
- It take the number of the replicates needs.
- It is **abstraction** of the pods.

stateful set

It is replicate the state of the application. It means the database of the application.

- database contain all the state of the application it would not change while the replicate the application.
- It would responsible for the data will same.

Minikube and Kubectl

Minikube

- It is a virtual box that contain nodes on it.
- It runs the **master and worker** processes.
- This virtual box sets inside the laptop and it have pre-installed docker.

kubectl

- It is a **command line** tool that interact with the clusters inside the minikube.
- It works on both local and cloud hyper clusters.
- It send the commands to the **Minikube** it execute the command based on the **kubctl.**

Example : In Master node there is api server the entry point to the application clusters.

If user want to create pod it will help through the CLI.

Installation

We don't need a docker while install the minikube because it contain the docker pre installed inside the minikube.

This is the official website to install the minikube on any os's

https://minikube.sigs.k8s.io/docs/start

start the minikube

- Once installation is done then start the minikube cluster using this command.

🧀 `$ minikube start --driver=virtualbox`

It is start the **minikube** on the virtual box [It help to create the vm on our local machine]

🧀 `$ minikube start`

It runs the minikube on docker container.

🧀 `$ minikube status` → it gives this output once properly cluster is started.

```
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

🧀 `$ kubectl version` → It is check the version of the minikube and kubectl.

> 🧀 `$ kubectl version --output yaml` → This give the readable output.

If the server version and the local version is same it will properly installed.

Stop the minikube

🧀 `$ minikube stop` → It stop the minikube clusters.

Workings

It all based on the command line tool **kubectl.**

🧀 `$ kubectl get nodes` → To get the nodes

🧀 `$ kubectl get pods` → It throws the if any pods are here currently.

🧀 `$ kubectl get services` → It give the permanent ip address of the pod.

create the cluster

We can't create the pods directly we can create the **deployments** based on the **image.**

🧀 `$ kubectl create deployment NAME --image=image`

It will help to get the deployment from the docker registry.

`$ kubectl create deployment nginx-depl --image = nginx` → It create the nginx deployment on minikbe cluster.

`$ kubectl get pod` → It create the pods inside the cluster.

🧀 `$ kubectl logs  NAME` → It gives the logs on the deployment.

🧀 `$ kubectl exec -it NAME --bin/bash` → It is execute the terminal of the deployment application.

Apply

It is the command that help to manage the resources of the kubernetes clusters.

- It followed by the file name or the url that have the deployment file of the pod.

🧀 `$ kubectl apply -f [file-name]` → It is the file to configure the pods.

In this file it contain all the details about the pod inside the cluster.

Here is the example to configure the **nginx** pod.

1. First create the deployment using this command.

   `$ kubectl create deployment nginx-depl --image=nginx`

2. Once it will get status get running, using this command to check if it running or not.

   `$ kubectl get pods`

3. Then create the file `deployment config.yaml` file or `deployment config.json` file. then paste the below code.

   ```
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: nginx-deployment
   spec:
     replicas: 3
     selector:
       matchLabels:
         app: nginx
     template:
       metadata:
         labels:
           app: nginx
       spec:
         containers:
         - name: nginx
           image: nginx:1.25
           ports:
           - containerPort: 80
   ```

4. After save the file to create the deployment using this command.

   🧀 `$ kubectl apply -f nginx-config.yaml`

Create the application with K8s Components

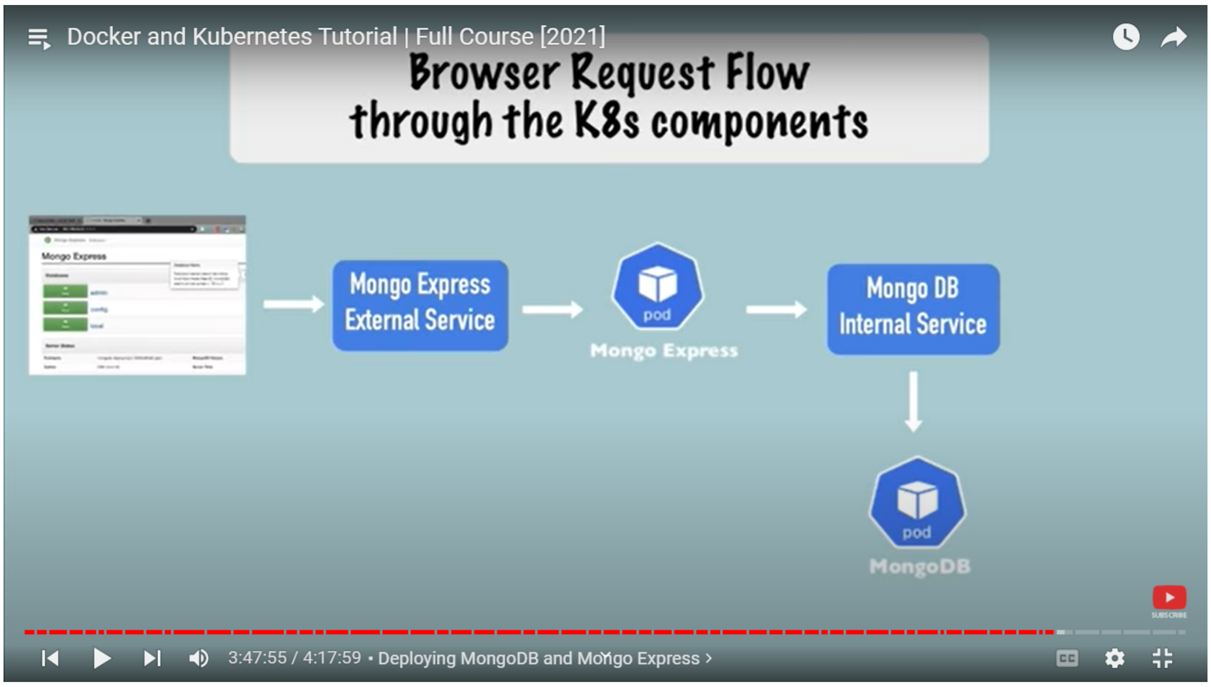
Using **Mongo-Express** and **mongo DB** to create the simple web application set with the help of the K8s components.

Needs:

1. 2 Deployments / Pods → To create the application and data base.
2. 2 Services → To connect the application through the URL.
3. ConfigMap → To build the deployment.yaml file and contain the database url.
4. Secret → For the credentials It contain credentials.

We create the application with no external service

This the basic setup of the application.


Docker and Kubernetes Tutorial | Full Course [2021]
Browser Request Flow through the K8s components
Mongo Express External Service
pod
Mongo Express
Mongo DB Internal Service
pod
MongoDB
3:47:55 / 4:17:59 • Deploying MongoDB and Mongo Express

- It start from the browser send the request to the mongo express external service.
- Then forwarded it to the mongo express pod.
- It convert the request into the internal service.
- It means the mongo db pod.
- And the component take the credential with the request and check and allow to access it.

Config the Mongo DB and internal Service:

Step 1:

create the mongo db configuration yaml file.

```yaml
## this file name is mongo.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
      - name: mongodb
        image: mongo
        ports:
        - containerPort: 27017
        env:
        - name: MONGO_INITDB_ROOT_USERNAME
          valueFrom:
            secretKeyRef:
              name: mongo-secret
              key: mongo-root-username
        - name: MONGO_INITDB_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mongo-secret
              key: mongo-root-password
```

step 2:

create the secret configuration file for the security.

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mongo-secret
type: Opaque
data:
  mongo-root-username: dXNlcm5hbWU=
  mongo-root-password: cGFzc3dvcmQ=
```

Step 3:

Note: In a mongo.yaml file we're referenced to the secret.yaml file for the credentials so It search for the credential inside the cluster.

So apply the secret.yaml file first and then apply the mongo.yaml file

🧀 `$ kubectl apply -f mongo-secret.yaml`

It create the secret to see that `$ kubectl get secret` .

🧀 `$ kubectl apply -f mongo.yaml`

To check the it running status `$ kubectl get pod`

step 4:

Create Mongo DB **internal Service** config file to connect the application to the database.

```
apiVersion: v1
kind: Service #This is the service type configuration
metadata:
  name: mongo-service #random name for the file
spec:
  selector:
    app: mongodb #It help to connect to pod through the label
  ports:
    - protocol: TCP
      port: 27017 #Service port that from the server
      targetPort: 27017 #container port of thdeployment
```

To check the service is configure `$ kubectl get service` use this.

🧀 `$ kubectl describe service mongo-service`

It shows the IP address the internal mongo-db IP address.

check the mongo-db IP address ,

🧀 `$ kubectl get pod -o wide`

Config the mongo-express and external Service:

Step 1:

Create the express deployment file.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: 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_ADMINPASSWORD
              valueFrom:
                secretKeyRef:
                  name: mongo-secret
                  key: mongo-root-password
            - name: ME_CONFIG_MONGODB_SERVER
              valueFrom:
                configMapKeyRef:
                  name: mongo-configmap
                  key: database_url
```

Step 2:

Crete the configMap file for the application.

```
apiVersion: v1
kind: ConfigMap #type of the configuration
metadata:
  name: mongo-configmap #name of the file
data:
  database_url: mongo-service #reference to the link
```

step 3:

Before we did the db configuration same as we will do for this.

- First apply the configMap file because it is referenced to the application file.

🧀 `$ kubectl apply -f monfo-configmap.yaml`

Check that is deployed or not using this `$ kubectl get configmap`

- Then deploy the **Mango-express** file

🧀 `$ kubectl apply -f mango-express.yaml`

to check `$ kubectl get deploment`

Step 4:

Config the **external service** to connect with the application.

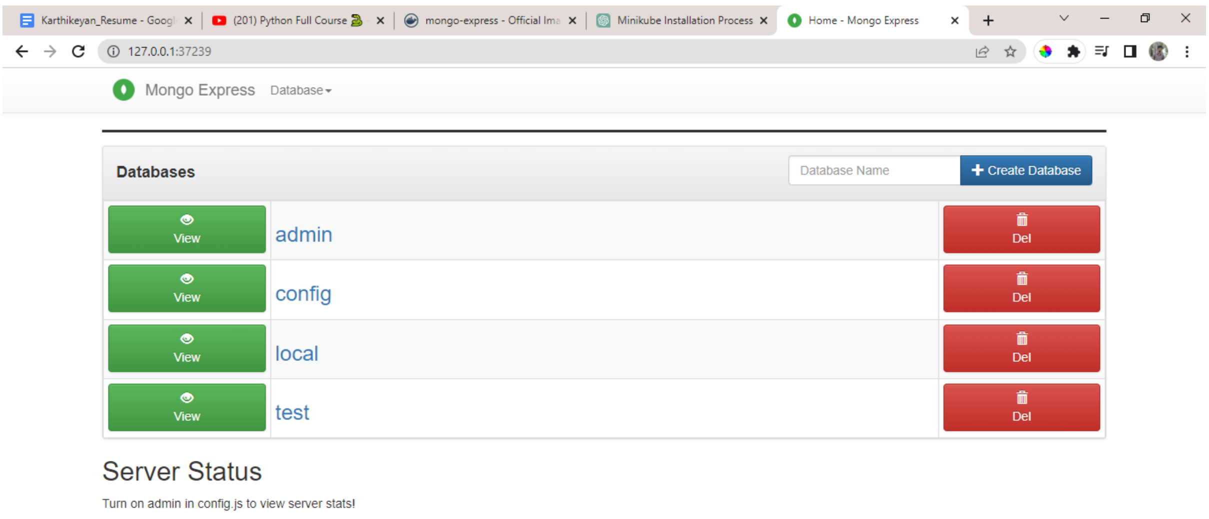
```
apiVersion: v1
kind: Service
metadata:
  name: mongo-express-service
spec:
  selector:
    app: mongo-express
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 8081
      targetPort: 8081
      nodePort: 30000
```

To check service `$ kubectl get service`.

Here is the **type: LoadBalancer** gives the internal IP address and the External IP address also.

That's all now use the application using this command.

```
$ minikube service mongo-express-service
```



127.0.0.1:37239

Mongo Express Database

Databases		
View	admin	Del
View	config	Del
View	local	Del
View	test	Del

Server Status

Turn on admin in config.js to view server stats!

Karthikeyan_Resume - Googl
(201) Python Full Course
mongo-express - Official Ima
Minikube Installation Process
Home - Mongo Express
Database Name
Create Database

This is the final output of the application that running on the port number 37239.