**SESSION - 60**

--> We are discussed kubernates resources.

--> 1. **namespace**

--> name space is an isolated project space in the kubernates cluster where you can create your resources and control them that specific like VPC in aws.

--> 2. **POD**

--> POD is the smallest deployment in the kubernates, a pod can have one or multiple containers. All the pods inside the containers share same network space and storage.

--> 3. **Multi-containers**

--> 4. **Labels**

--> Labels are selectors you can use labels to select other resources inside kubernates.

--> 5.**Annotations**

--> Annotations are not going to use selectors they are select external resources outside of the kubernates cluster and you can mention the information about the external tools.

--> 6. **Resources**:

--> You can limit the resources to your cart so that will not go rock we will not occupy to much underline host resources. Two types are 1. request 2. limit. Soft and hard limit.

--> 7. **environments**

--> You can mention environment variables configure map you can mention environment as config map you can refer that.

--> 8. **secret**

--> encoded values

--> 9. **Sevices**

--> mainly services, for pod to pod communication and load balancing we will use service. Services use selectors to select the pod.

--> 3 types of services

1. Cluster ip
2. Node Pod
3. Load balancer

--> Cluster IP: Cluster ip is internal to the cluster

--> Node pod : Node pod is if you want to expose your application to the outside world you can use node pod and load balancer. Node open the port in the Kubernates node and specifically we want one node we can tel not randomly you have an range. We can mention port.

--> eks node pode range -- check in google

--> the default port range used for nodeport services is 30000-32767

--> So you can mention this range in any port. Specifically if you want.

--> Load balacer: load balacer works in cloud environment service can create load balancer automatically. And then we have seen the replica set if you want particular nuber of replicas in your POD you can use replica set. Again selectors are labels here.

--> 10. **Deployment**:

--> Whenever there is a version change in image you can make you use of deployment it will create new replicaset it will create the containers create pods with new version and it will remove the old pods with old version.

--> use all resources and create our roboshop project in kubernates

--> create a directory roboshop

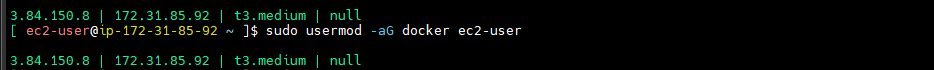
--> First we have to create the namespace

1. Build the image
2. Configure image

--> Build the image we were completed in docker.

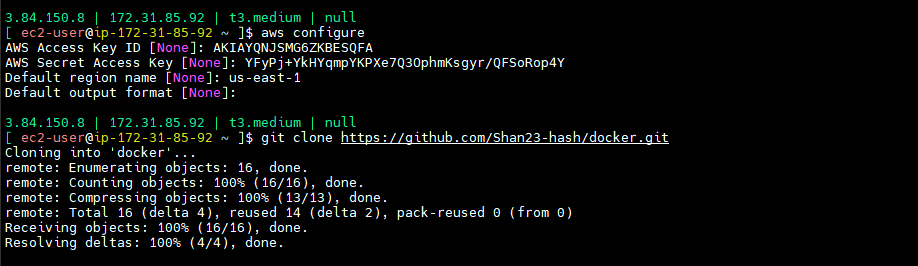
--> that images connect to kunernates pods.

**--> sudo usermod -aG docker ec2-user**

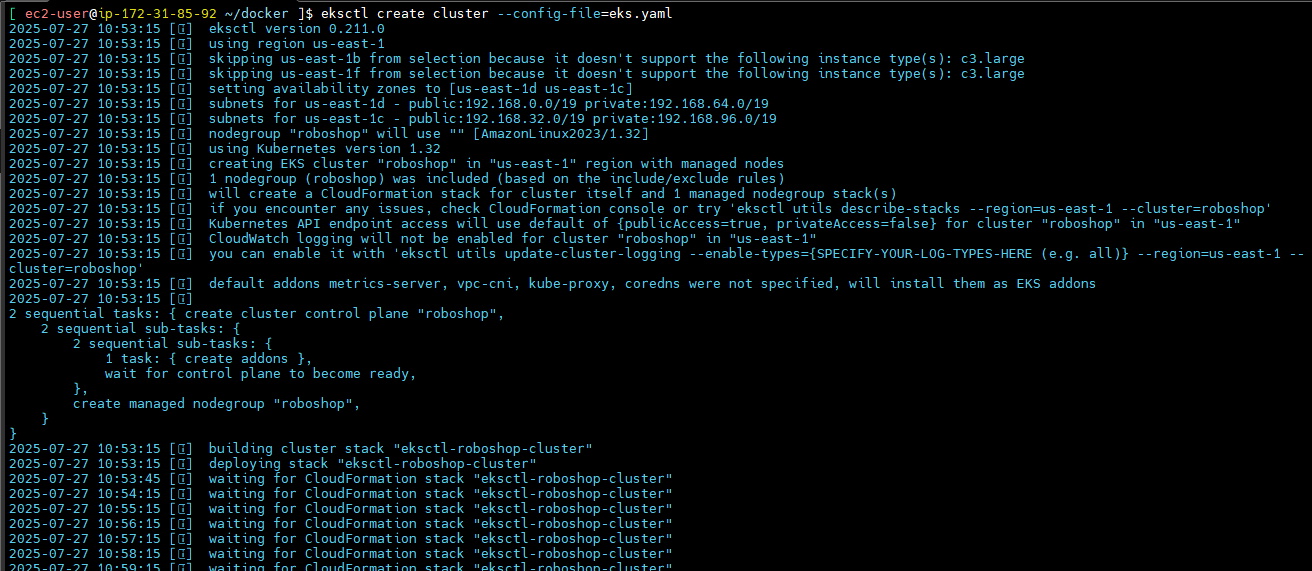


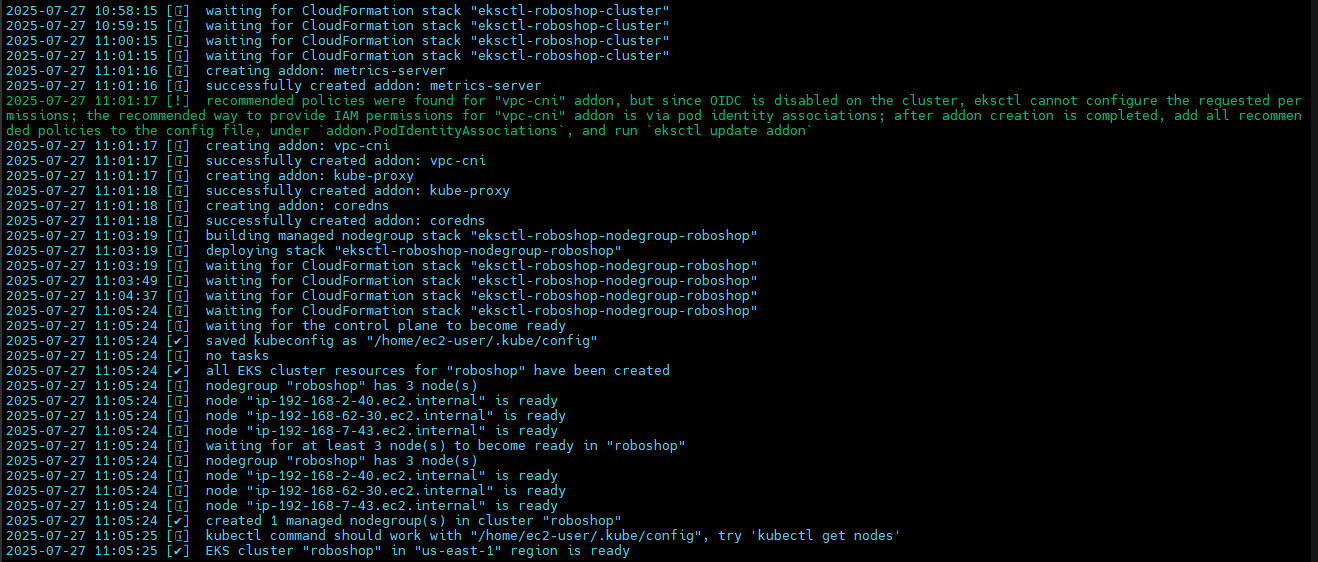
**--> aws configure**

**--> git clone <https://github.com/Shan23-hash/docker.git>**



**--> eksctl create cluster --config-file=eks.yaml**





**01-namespace.yaml**

apiVersion: v1

kind: Namespace

metadata:

name: roboshop

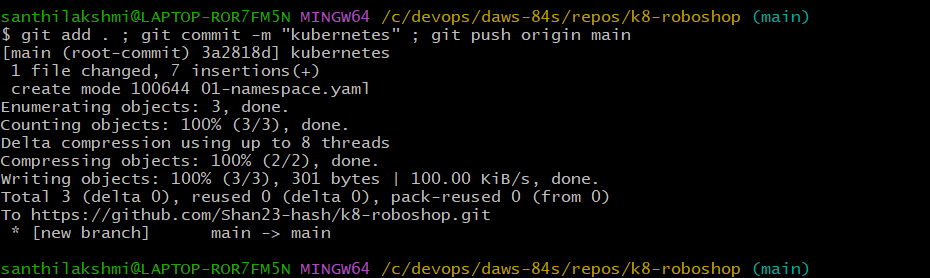
labels:

project: roboshop

environment: dev

--> /c/devops/daws-84s/repos/k8-roboshop

--> git add . ; git commit -m "kubernetes" ; git push origin main



--> Pull the code

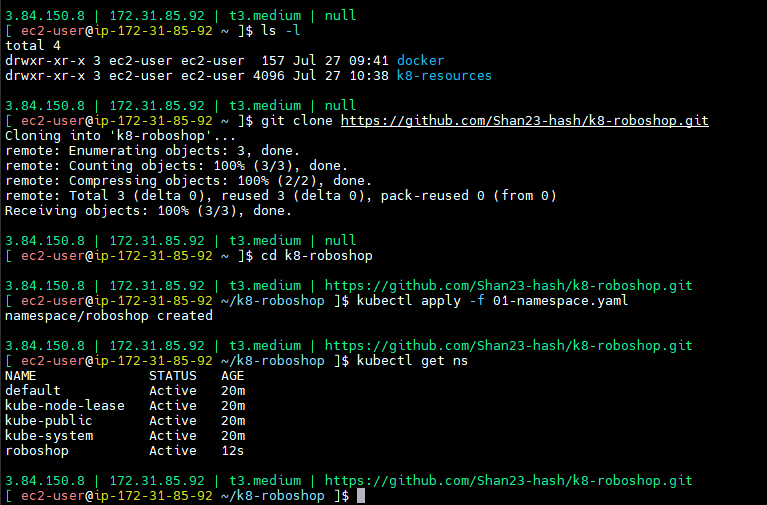
--> **git clone https://github.com/Shan23-hash/k8-roboshop.git**

--> kubectl get nodes

--> **cd ../k8-roboshop**

**--> kubectl apply -f 01-namespace.yaml**

**--> kubectl get ns**



--> POD is a subset of deployment, if you create a deplyment you have a advantage.

--> so that automatically you will get replicas set.

--> if incase by mistakenly delete the pod it will update and create. pod have any problem it will recreate. You can increase the no of replicas.

--> you can change the image.

--> just apply the manifest and update then will automatically remove the old image.

--> Old container to create the new pod.

--> we should create an pod we can go for deployment.

--> with mongodb have depency catalogue and user.

--> we have to run cluster ip and service.

--> because it’s purely internal job.

**mongodb/manifest.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

name: mongodb

namespace: roboshop

# deployment labels

labels:

component: mongodb

project: roboshop

tier: database

spec:

replicas: 1

# These are the labels replica set use to create pod replicas, this should match pod labels

selector:

matchLabels:

component: mongodb

project: roboshop

tier: database

# This is pod definition

template:

metadata:

labels:

component: mongodb

project: roboshop

tier: database

spec:

containers:

- name: mongodb

image: shan2324/mongodb:v1

---

apiVersion: v1

kind: Service

metadata:

name: mongodb

namespace: roboshop

labels:

component: mongodb

project: roboshop

tier: database

spec:

selector:

component: mongodb

project: roboshop

tier: database

ports:

- protocol: TCP

port: 27017 # service port

targetPort: 27017 # container port

--> replicas will work for state less.

--> databases we should not keep on in kubernates.

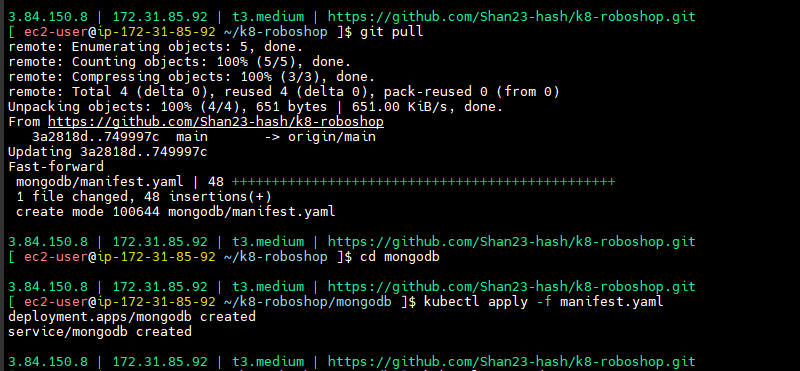
--> replica set labels and pod labels

--> specially I have to give name space roboshop.

--> **~/k8-roboshop/mongodb**

**--> git pull**

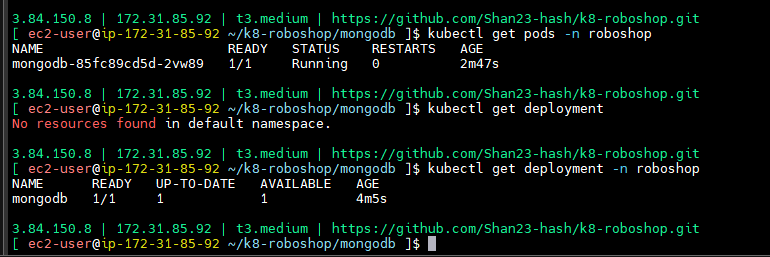
--> **kubectl apply -f manifest.yaml**



**--> kubectl get pods --all-namespaces**

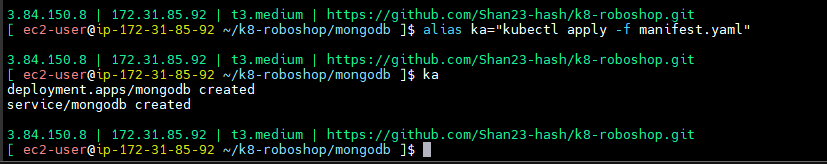
**--> kubectl get pods -n roboshop**

**--> kubectl get deployment -n roboshop**



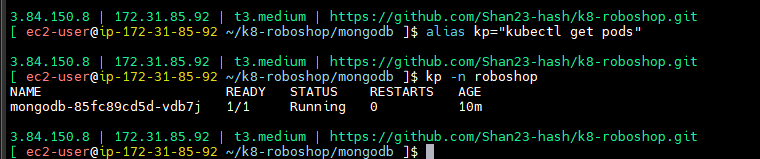
**--> alias ka="kubectl apply -f manifest.yaml"**

**--> ka**



**--> alias kp="kubectl get pods"**

**--> kp -n roboshop**



--> Install kubens

--> install k9s

**--> sudo git clone https://github.com/ahmetb/kubectx /opt/kubectx**

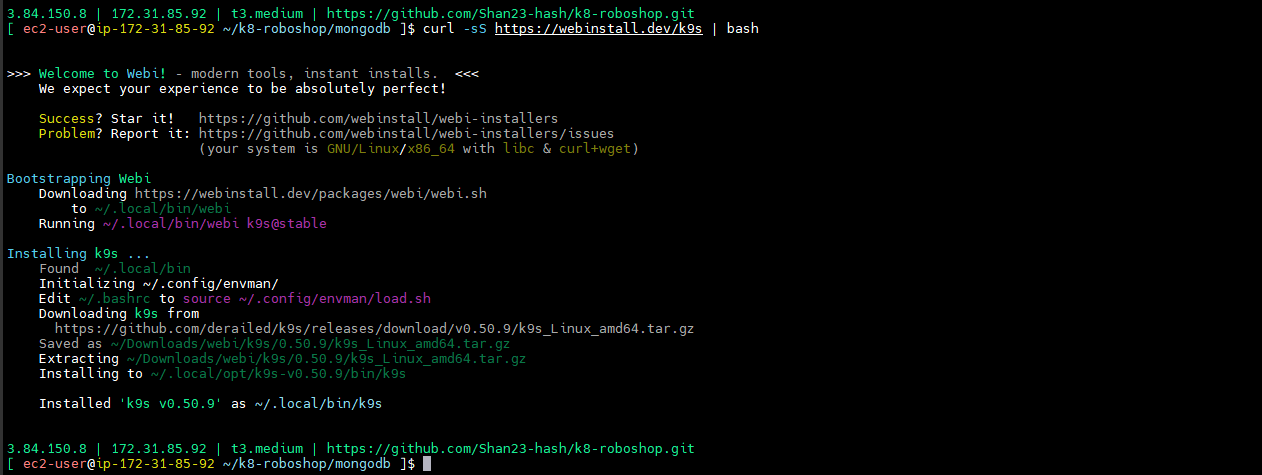
**--> sudo ln -s /opt/kubectx/kubectx /usr/local/bin/kubectx**

**--> sudo ln -s /opt/kubectx/kubens /usr/local/bin/kubens**

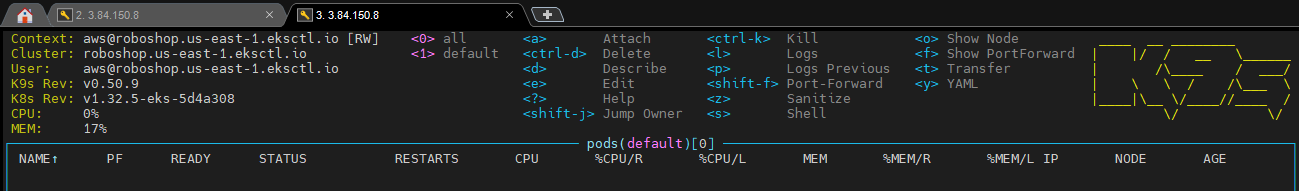
--> k9s is command line

--> k9s install

--> **curl -sS https://webinstall.dev/k9s | bash**



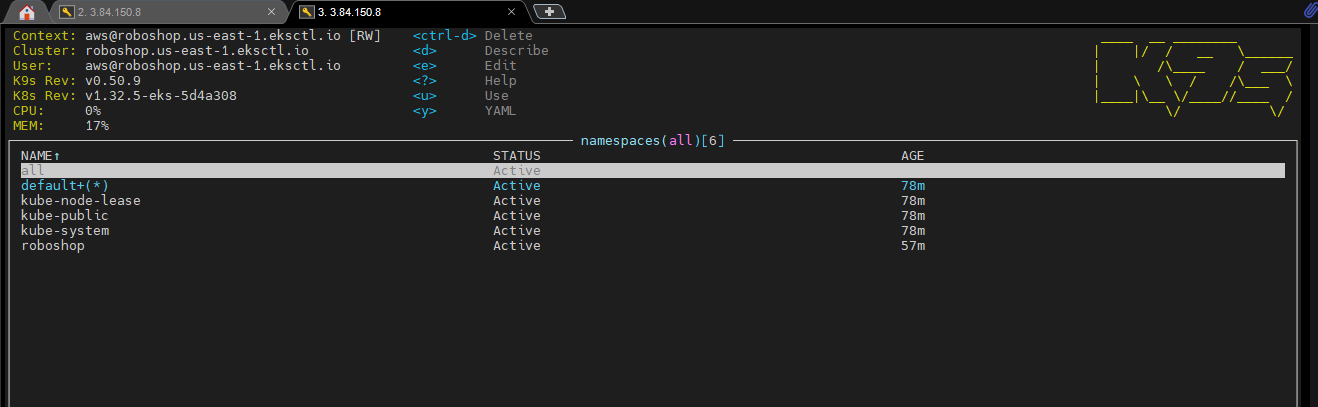
--> Take duplicate session and give k9s



--> shift+; -- give deployments

--> configmap

--> namespace

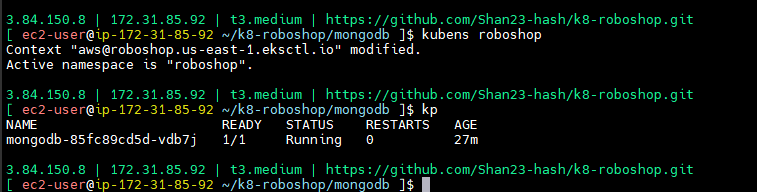


--> if you got any error use this coomad **:ns roboshop**

--> if you want totally exit --> ctrl+c

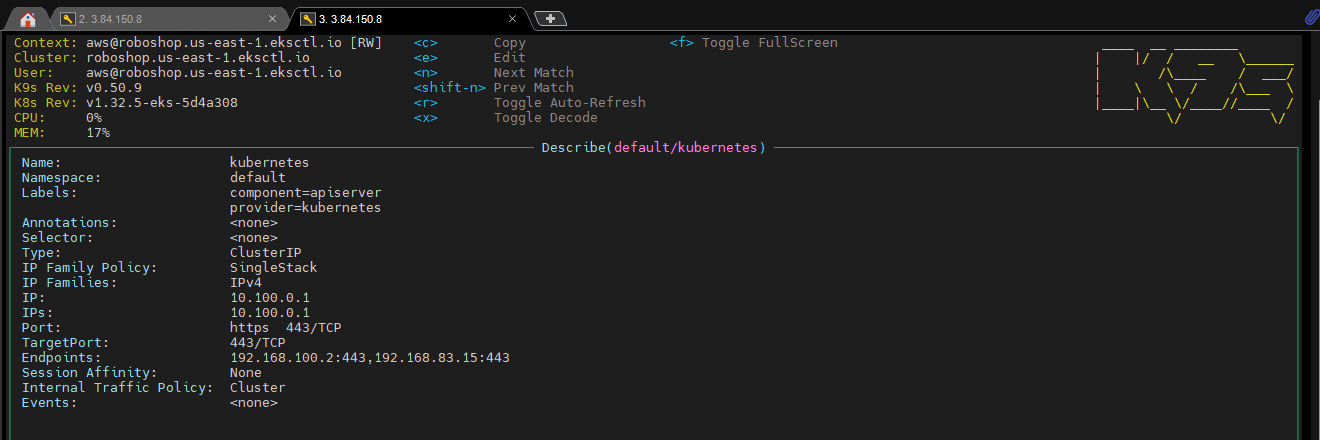
-**-> kubens roboshop**

**--> kp**



--> sevices in mondodb - I already added

**--> k9s**



**Catalogue**

**--> catalogue/manifest.yaml**

apiVersion: v1

kind: ConfigMap

metadata:

name: catalogue

namespace: roboshop

labels:

component: catalogue

project: roboshop

tier: app

data:

MONGO: "true"

MONGO\_URL: "mongodb://mongodb:27017/catalogue"

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: catalogue

namespace: roboshop

# deployment labels

labels:

component: catalogue

project: roboshop

tier: app

spec:

replicas: 1

# These are the labels replica set use to create pod replicas, this should match pod labels

selector:

matchLabels:

component: catalogue

project: roboshop

tier: app

# This is pod definition

template:

metadata:

labels:

component: catalogue

project: roboshop

tier: app

spec:

containers:

- name: catalogue

image: shan2324/catalogue:v1

envFrom:

- configMapRef:

name: catalogue

---

apiVersion: v1

kind: Service

metadata:

name: catalogue

namespace: roboshop

labels:

component: catalogue

project: roboshop

tier: app

spec:

selector:

component: catalogue

project: roboshop

tier: app

ports:

- protocol: TCP

port: 8080 # service port

targetPort: 8080 # container port

--> catalogue also deployment instaed of mongodb

--> I’m going for replacing

--> port no 8080

--> image my own image -- this public image so it will pull

--> which resource have to configure

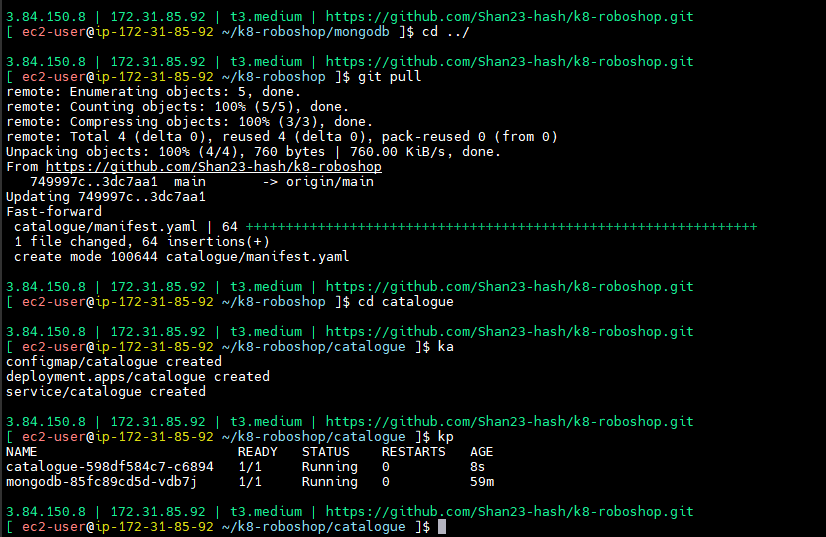
--> yestarday we discussed configure

**--> git pull**

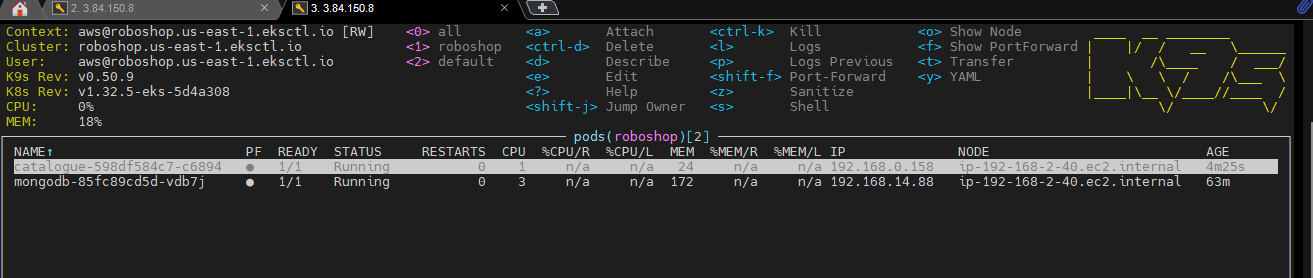
**--> cd catalogue**

**--> ka**

**--> kp**

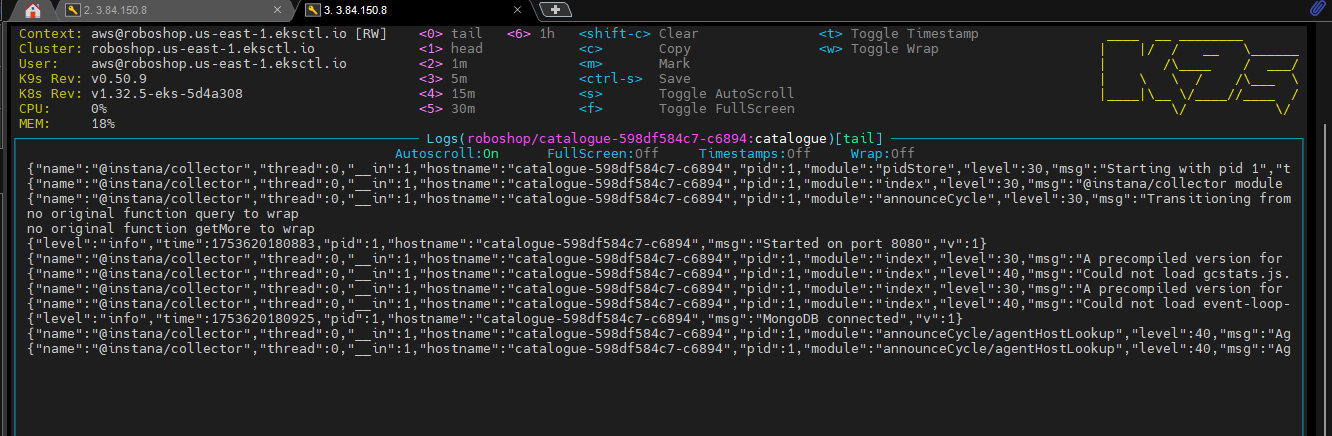


**ns roboshop**



--> How can I check whether catalogue is running or not

--> if you type “l” you will get logs



esc+shell -- if you you will get shell

--> curl not there.

--> but how I can check

--> saparate part you can write debug

**debug/Dockerfile**

FROM almalinux:9

RUN dnf install telnet net-tools iputils -y

RUN useradd roboshop

USER roboshop

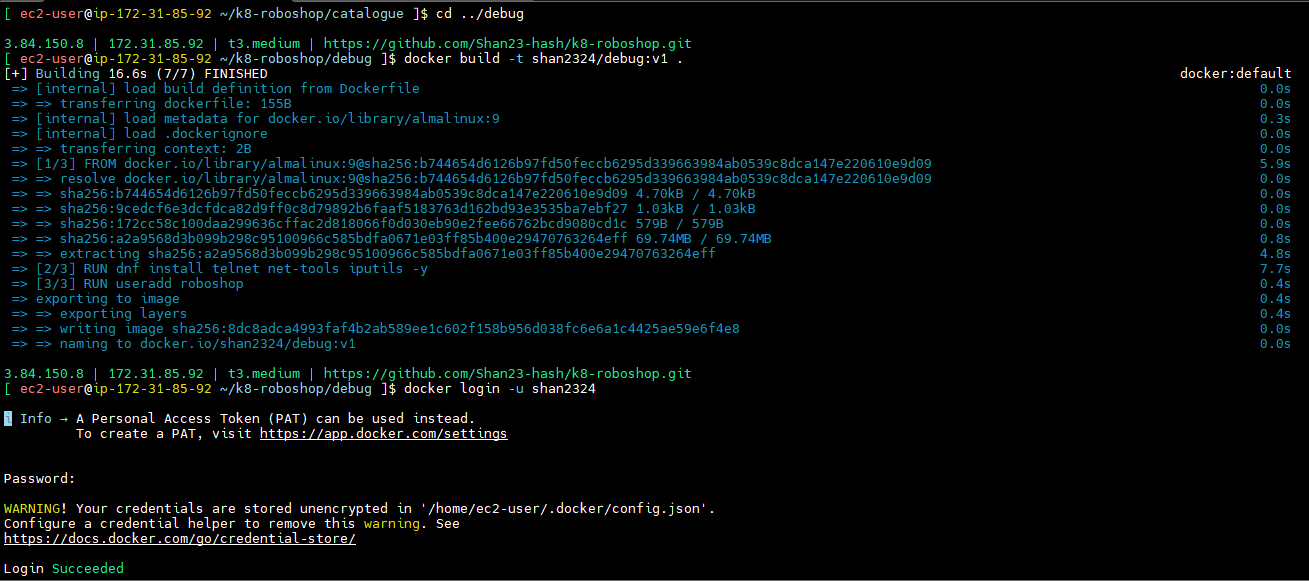
CMD ["sleep","100000"]

--> if incase completed that 1lak second you can run again

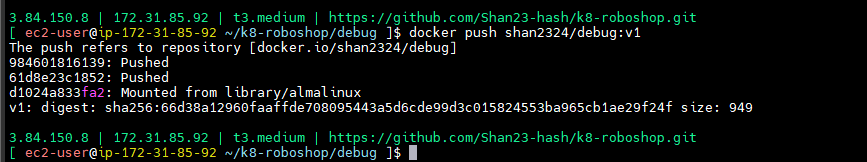
**--> cd ../debug**

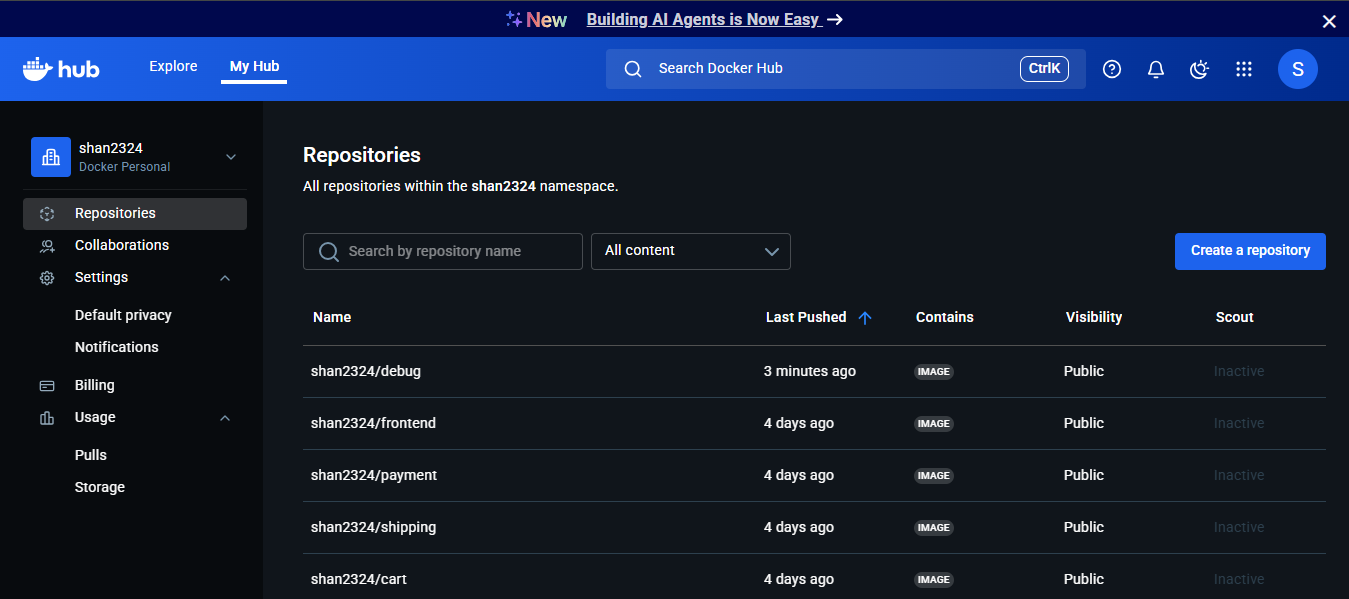
**--> docker build -t shan2324/debug:v1 .**

**--> docker login -u shan2324**



**--> docker push shan2324/debug:v1**





**debug/manifest.yaml**

apiVersion: v1

kind: Pod

metadata:

name: debug

namespace: roboshop

spec:

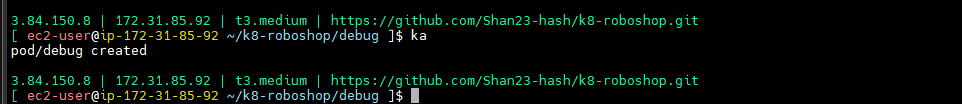
containers:

- name: debug

image: shan2324/debug:v1

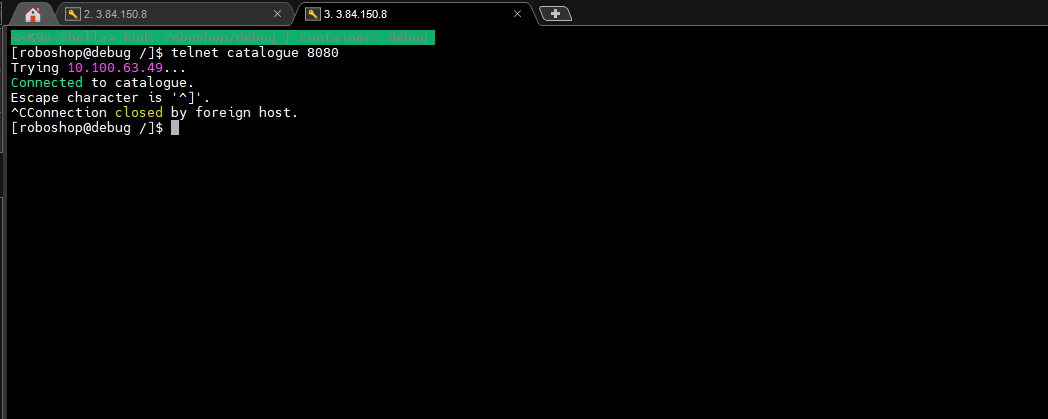
--> push and pull the code

--> **ka**



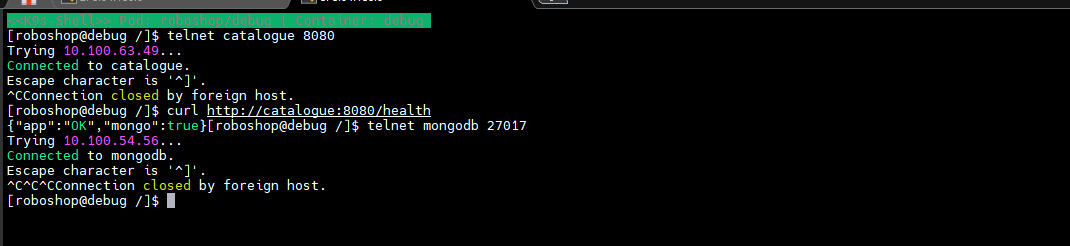
--> Go to duplicate session--> select debug --> esc+s --> shell script

--> [roboshop@debug /]$ **telnet catalogue 8080**



**--> curl <http://catalogue:8080/health>**

**--> telnet mongodb 27017**



**REDIS**

**redis/manifest.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

name: redis

namespace: roboshop

# deployment labels

labels:

component: redis

project: roboshop

tier: database

spec:

replicas: 1

# These are the labels replica set use to create pod replicas, this should match pod labels

selector:

matchLabels:

component: redis

project: roboshop

tier: database

# This is pod definition

template:

metadata:

labels:

component: redis

project: roboshop

tier: database

spec:

containers:

- name: redis

image: redis:7.0

---

apiVersion: v1

kind: Service

metadata:

name: redis

namespace: roboshop

labels:

component: redis

project: roboshop

tier: database

spec:

selector:

component: redis

project: roboshop

tier: database

ports:

- protocol: TCP

port: 6379 # service port

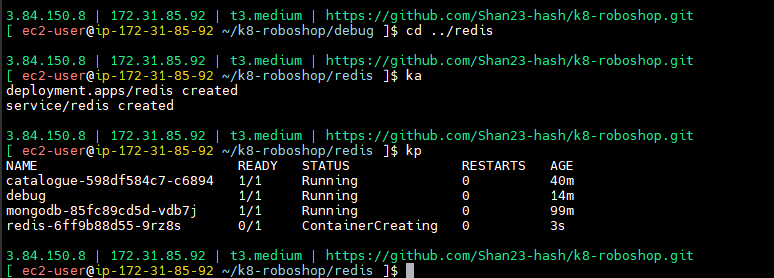
targetPort: 6379 # container port

--> for redis we can take directly

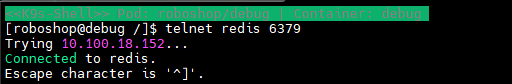
**--> cd ../redis**

**--> ka**

**--> kp**

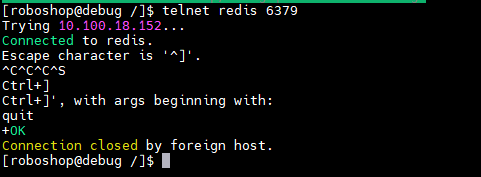


**--> telnet redis 6379**



**--> Ctrl+]**

**--> quit**



**USER**

**user/manifest.yaml**

apiVersion: v1

kind: ConfigMap

metadata:

name: user

namespace: roboshop

labels:

component: user

project: roboshop

tier: app

data:

MONGO\_URL: "mongodb://mongodb:27017/users"

REDIS\_URL: "redis://redis:6379"

MONGO: "true"

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: user

namespace: roboshop

# deployment labels

labels:

component: user

project: roboshop

tier: app

spec:

replicas: 1

# These are the labels replica set use to create pod replicas, this should match pod labels

selector:

matchLabels:

component: user

project: roboshop

tier: app

# This is pod definition

template:

metadata:

labels:

component: user

project: roboshop

tier: app

spec:

containers:

- name: user

image: shan2324/user:v1

envFrom:

- configMapRef:

name: user

---

apiVersion: v1

kind: Service

metadata:

name: user

namespace: roboshop

labels:

component: user

project: roboshop

tier: app

spec:

selector:

component: user

project: roboshop

tier: app

ports:

- protocol: TCP

port: 8080 # service port

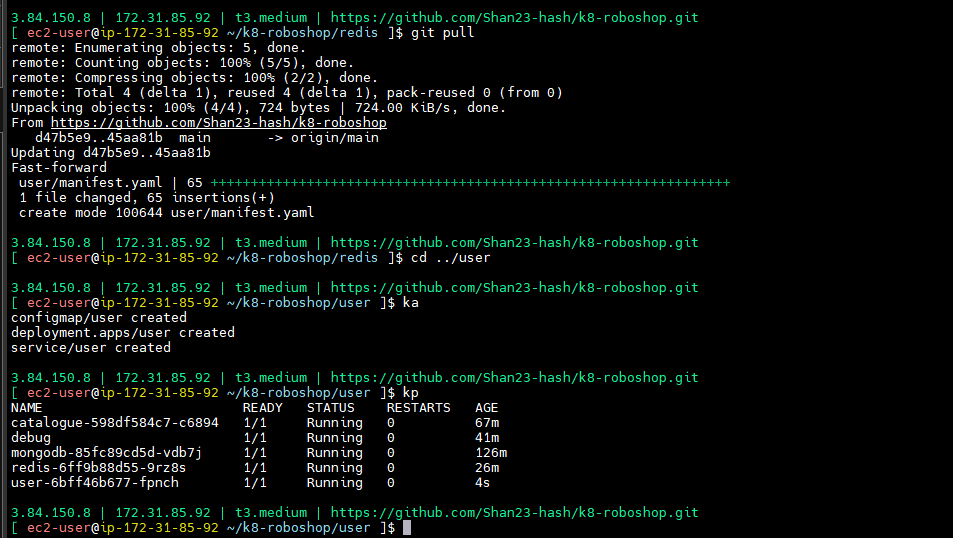
targetPort: 8080 # container port

--> push and pull the code

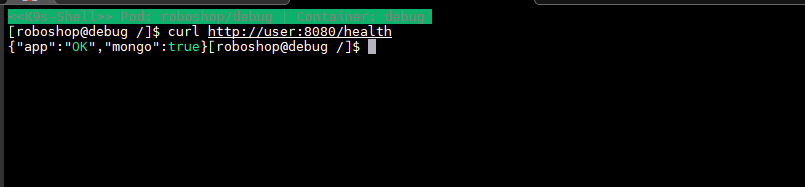
**--> cd ../user**

**--> ka**

**--> kp**



**--> curl <http://user:8080/health>**



**CART**

**cart/manifest.yaml**

apiVersion: v1

kind: ConfigMap

metadata:

name: cart

namespace: roboshop

labels:

component: cart

project: roboshop

tier: app

data:

REDIS\_HOST: "redis"

CATALOGUE\_HOST: "catalogue"

CATALOGUE\_PORT: "8080"

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: cart

namespace: roboshop

# deployment labels

labels:

component: cart

project: roboshop

tier: app

spec:

replicas: 1

# These are the labels replica set use to create pod replicas, this should match pod labels

selector:

matchLabels:

component: cart

project: roboshop

tier: app

# This is pod definition

template:

metadata:

labels:

component: cart

project: roboshop

tier: app

spec:

containers:

- name: cart

image: shan2324/cart:v1

envFrom:

- configMapRef:

name: cart

---

apiVersion: v1

kind: Service

metadata:

name: cart

namespace: roboshop

labels:

component: cart

project: roboshop

tier: app

spec:

selector:

component: cart

project: roboshop

tier: app

ports:

- protocol: TCP

port: 8080 # service port

targetPort: 8080 # container port

--> change very carefully

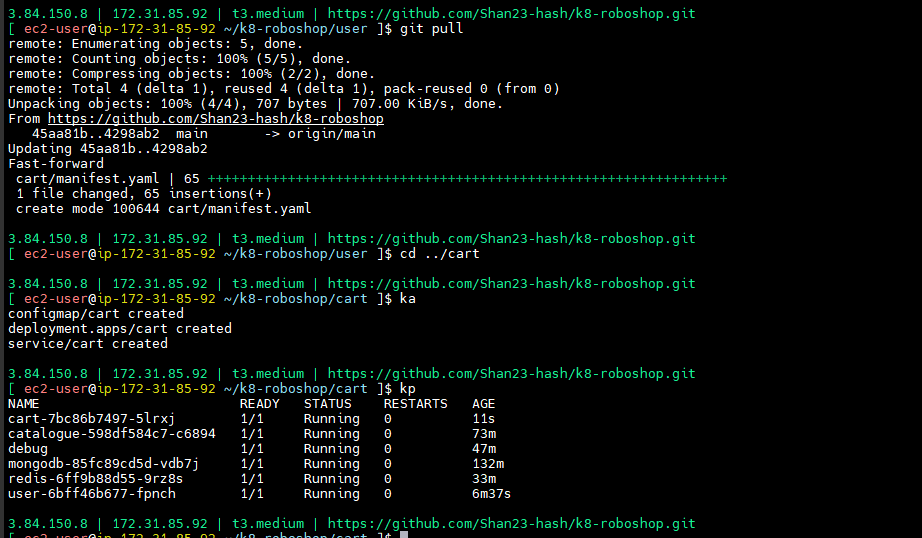
--> git pull and push

**--> git pull**

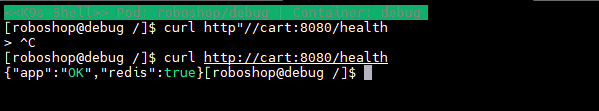
**--> cd ../cart**

**--> ka**

**--> kp**



**curl http://cart:8080/health**



**MYSQL**

**mysql/manifest.yaml**

apiVersion: v1

kind: Secret

metadata:

name: mysql

labels:

component: mysql

project: roboshop

tier: database

type: Opaque

data:

MYSQL\_ROOT\_PASSWORD: "Um9ib1Nob3BAMQ=="

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: mysql

namespace: roboshop

# deployment labels

labels:

component: mysql

project: roboshop

tier: database

spec:

replicas: 1

# These are the labels replica set use to create pod replicas, this should match pod labels

selector:

matchLabels:

component: mysql

project: roboshop

tier: database

# This is pod definition

template:

metadata:

labels:

component: mysql

project: roboshop

tier: database

spec:

containers:

- name: mysql

image: shan2324/mysql:v1

envFrom:

- secretRef:

name: mysql

---

apiVersion: v1

kind: Service

metadata:

name: mysql

namespace: roboshop

labels:

component: mysql

project: roboshop

tier: database

spec:

selector:

component: mysql

project: roboshop

tier: database

ports:

- protocol: TCP

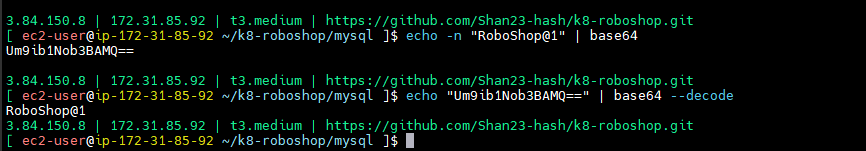
port: 3306 # service port

targetPort: 3306 # container port

--> push and pull the code

**--> echo -n "RoboShop@1" | base64**

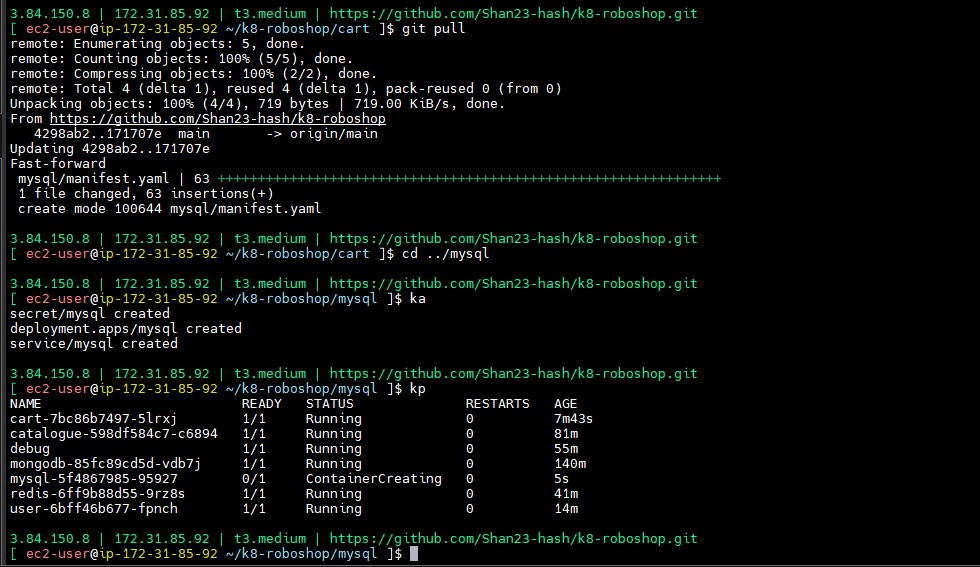
**--> echo "Um9ib1Nob3BAMQ==" | base64 --decode**



**--> cd ../mysql**

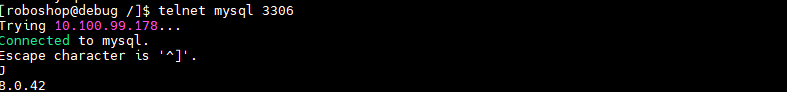
**--> ka**

**--> kp**



--> esc+s

--> **telnet mysql 3306**



--> mysql shell script

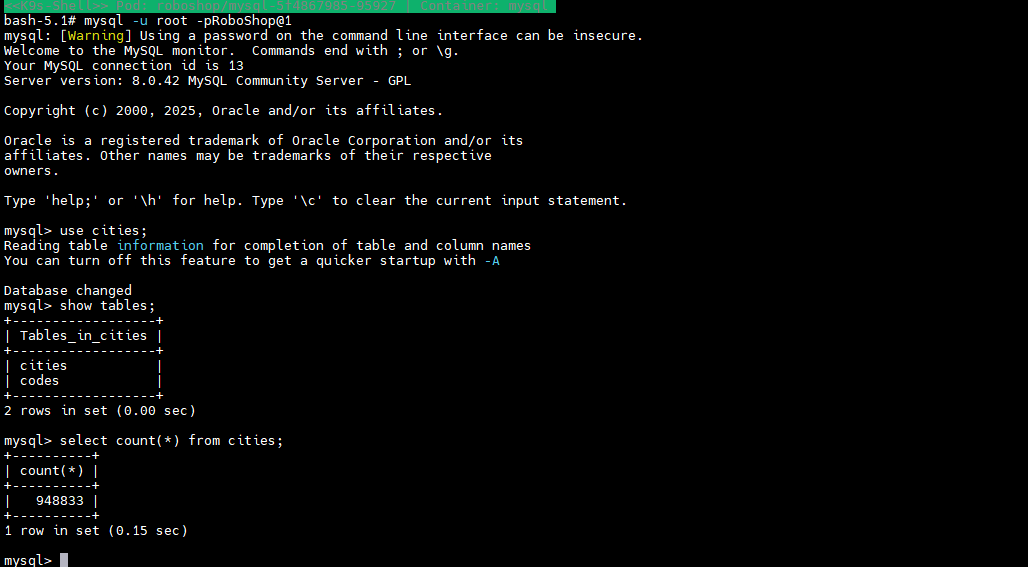
--> esc+s

**--> mysql -u root -pRoboShop@1**

**--> use cities;**

**--> show tables;**

**--> select count(\*) from cities;**



**SHIPPING**

**shipping/manifest.yaml**

apiVersion: v1

kind: ConfigMap

metadata:

name: shipping

namespace: roboshop

labels:

component: shipping

project: roboshop

tier: app

data:

CART\_ENDPOINT: "cart:8080"

DB\_HOST: "mysql"

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: shipping

namespace: roboshop

# deployment labels

labels:

component: shipping

project: roboshop

tier: app

spec:

replicas: 1

# These are the labels replica set use to create pod replicas, this should match pod labels

selector:

matchLabels:

component: shipping

project: roboshop

tier: app

# This is pod definition

template:

metadata:

labels:

component: shipping

project: roboshop

tier: app

spec:

containers:

- name: shipping

image: shan2324/shipping:v1

envFrom:

- configMapRef:

name: shipping

---

apiVersion: v1

kind: Service

metadata:

name: shipping

namespace: roboshop

labels:

component: shipping

project: roboshop

tier: app

spec:

selector:

component: shipping

project: roboshop

tier: app

ports:

- protocol: TCP

port: 8080 # service port

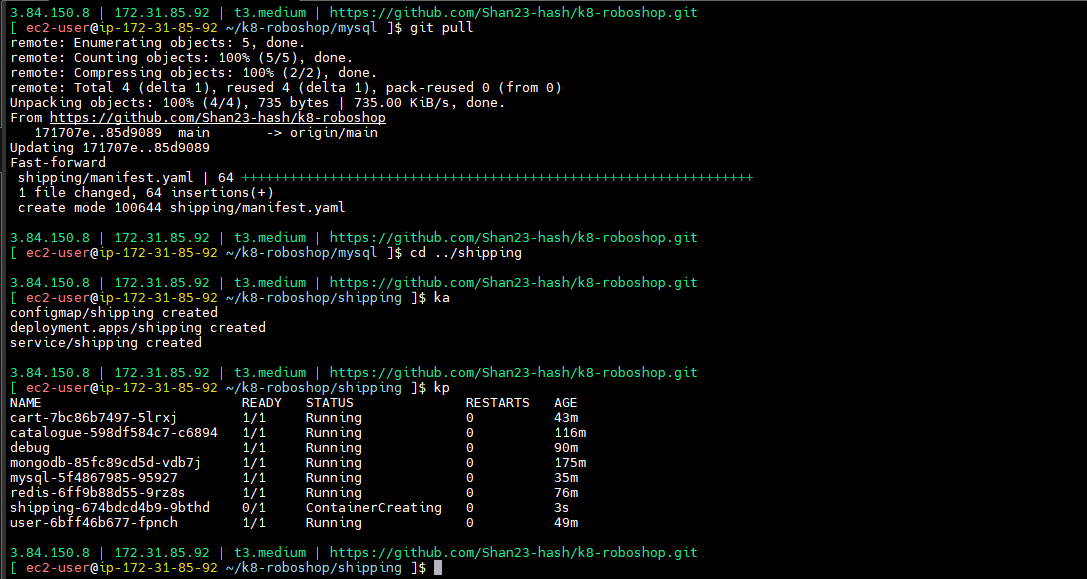
targetPort: 8080 # container port

--> Push and pull the code

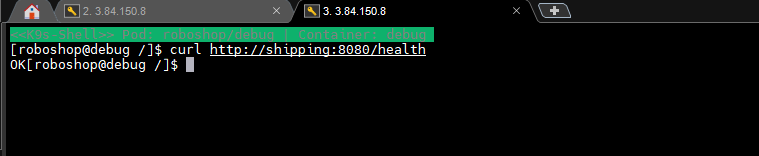
**--> cd ../shipping**

**--> ka**

**--> kp**

****

**--> curl <http://shipping:8080/health>**



--> if mysql is not running shipping will not work.

**RABBITMQ**

**rabbitmq/manifest.yaml**

apiVersion: v1

kind: Secret

metadata:

name: rabbitmq

labels:

component: rabbitmq

project: roboshop

tier: database

type: Opaque

data:

RABBITMQ\_DEFAULT\_USER: "cm9ib3Nob3A="

RABBITMQ\_DEFAULT\_PASS: "cm9ib3Nob3AxMjM="

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: rabbitmq

namespace: roboshop

# deployment labels

labels:

component: rabbitmq

project: roboshop

tier: database

spec:

replicas: 1

# These are the labels replica set use to create pod replicas, this should match pod labels

selector:

matchLabels:

component: rabbitmq

project: roboshop

tier: database

# This is pod definition

template:

metadata:

labels:

component: rabbitmq

project: roboshop

tier: database

spec:

containers:

- name: rabbitmq

image: rabbitmq:3

envFrom:

- secretRef:

name: rabbitmq

---

apiVersion: v1

kind: Service

metadata:

name: rabbitmq

namespace: roboshop

labels:

component: rabbitmq

project: roboshop

tier: database

spec:

selector:

component: rabbitmq

project: roboshop

tier: database

ports:

- protocol: TCP

port: 5672 # service port

targetPort: 5672 # container port

--> port no is 5642.

**--> echo -n "roboshop" | base64**

**--> echo -n "roboshop123" | base64**

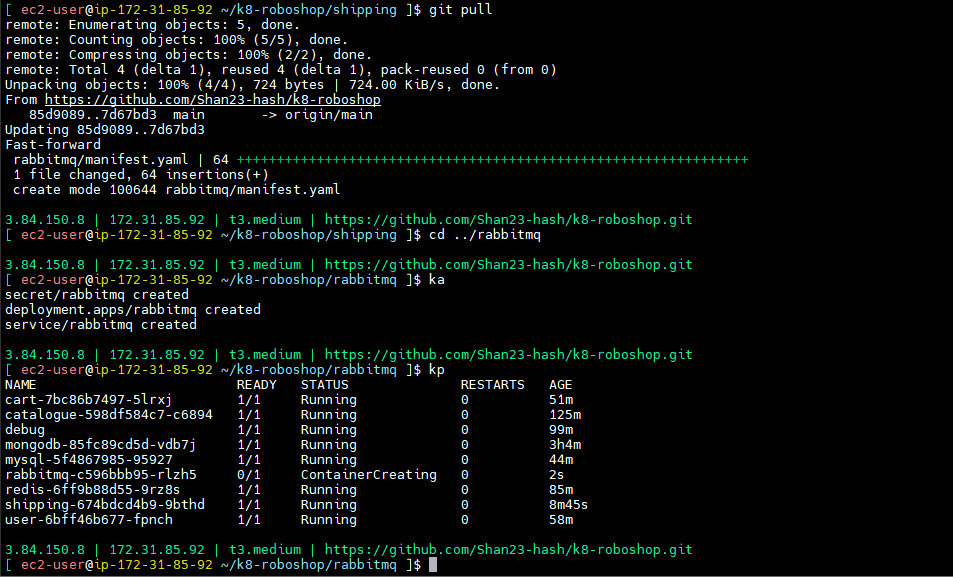


**--> Push and pull the code**

**--> cd ../rabbitmq**

**--> ka**

**--> kp**

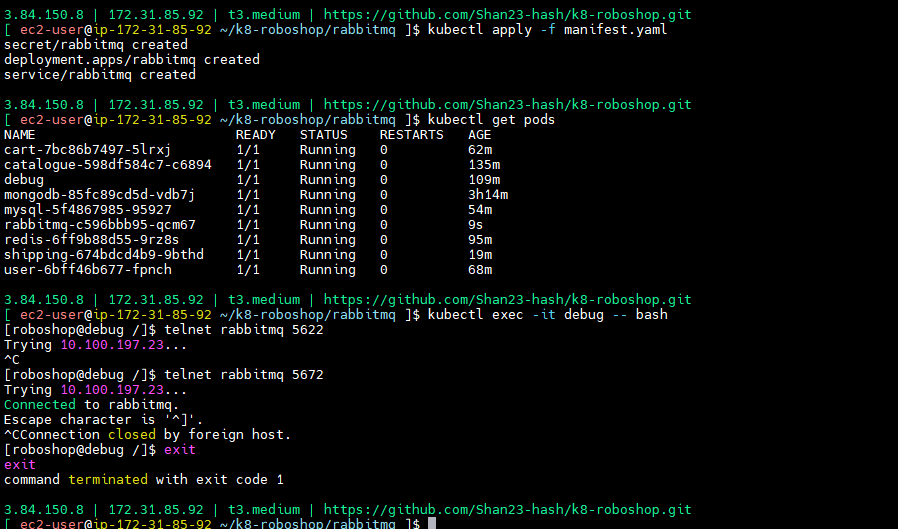


**--> kubectl apply -f manifest.yaml**

**--> kubectl get pods**

**--> kubectl exec -it debug -- bash**

**--> telnet rabbitmq 5672**



**PAYMENT**

**payment/manifest.yaml**

apiVersion: v1

kind: ConfigMap

metadata:

name: payment

namespace: roboshop

labels:

component: payment

project: roboshop

tier: app

data:

CART\_HOST: "cart"

CART\_PORT: "8080"

USER\_HOST: "user"

USER\_PORT: "8080"

AMQP\_HOST: "rabbitmq"

---

apiVersion: v1

kind: Secret

metadata:

name: payment

labels:

component: payment

project: roboshop

tier: app

type: Opaque

data:

AMQP\_USER: "cm9ib3Nob3A="

AMQP\_PASS: "cm9ib3Nob3AxMjM="

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: payment

namespace: roboshop

# deployment labels

labels:

component: payment

project: roboshop

tier: app

spec:

replicas: 1

# These are the labels replica set use to create pod replicas, this should match pod labels

selector:

matchLabels:

component: payment

project: roboshop

tier: app

# This is pod definition

template:

metadata:

labels:

component: payment

project: roboshop

tier: app

spec:

containers:

- name: payment

image: joindevops/payment:v1

envFrom:

- configMapRef:

name: payment

- secretRef:

name: payment

---

apiVersion: v1

kind: Service

metadata:

name: payment

namespace: roboshop

labels:

component: payment

project: roboshop

tier: app

spec:

selector:

component: payment

project: roboshop

tier: app

ports:

- protocol: TCP

port: 8080 # service port

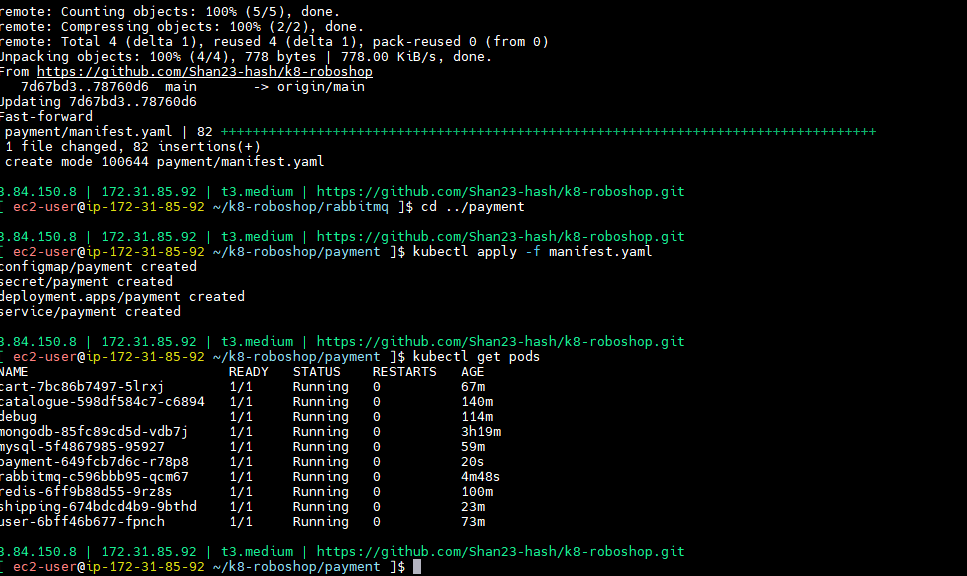
targetPort: 8080 # container port

--> Push and pull the code

**--> cd ../payment**

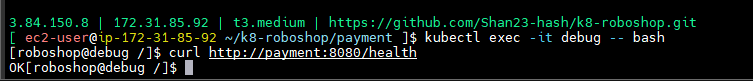
**--> kubectl apply -f manifest.yaml**

**--> kubectl get pods**

****

**--> kubectl exec -it debug -- bash**

**--> curl <http://payment:8080/health>**



**FRONTEND**

--> change all the compounends

--> config flag,services

--> port no 80.

--> type:load balancer

--> remove config flag.

--> git push and pull the code

**--> kubectl apply -f manifest.yaml**

**--> kubectl get pods**

**--> Kubectl logs** **frontend-5f76d444d8-449d6**

--> error is non root user

--> there is something called system port range - check in google

--> 0 to 1023 allocated to system port.

--> http 80

--> this is reserved for common

--> How to change nginx port

--> in nginx.conf have to change port

**Interview question**

**--> How to change the port no of nginx?**

Ans)) in /etc/nginx/nginx.conf you can do changes and restart the server.

--> if I changed this one again need to build the image

--> this is also configuration we have one option in kuberbates.

--> we can also use files as sonfigmap and mount to pod.

--> file is physical thing /etc/nginx/nginx.conf we can mount an file.

--> nginx conf as config map -- check in Google

--> so here you can write file also

--> needed configuration noted like this -- check in code.

--> adjust the configuration file

--> /etc/nginx/nginx.conf -- mention 8080, check remain thing where correct or not.

--> mention volumes and mount it, read you can mention -- true

--> push and pull the code -- again got error

--> see the vedio for understanding frontend code.

**frontend /manifest.yaml**

apiVersion: v1

kind: ConfigMap

metadata:

name: nginx-conf

namespace: roboshop

labels:

component: frontend

project: roboshop

tier: web

data:

nginx.conf: |

user nginx;

worker\_processes auto;

error\_log /var/log/nginx/error.log notice;

pid /run/nginx.pid;

include /usr/share/nginx/modules/\*.conf;

events {

worker\_connections 1024;

}

http {

log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" "$http\_x\_forwarded\_for"';

access\_log /var/log/nginx/access.log main;

sendfile on;

tcp\_nopush on;

keepalive\_timeout 65;

types\_hash\_max\_size 4096;

include /etc/nginx/mime.types;

default\_type application/octet-stream;

include /etc/nginx/conf.d/\*.conf;

server {

listen 8080;

listen [::]:8080;

server\_name \_;

root /usr/share/nginx/html;

include /etc/nginx/default.d/\*.conf;

error\_page 404 /404.html;

location = /404.html {

}

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

}

location /images/ {

expires 5s;

root /usr/share/nginx/html;

try\_files $uri /images/placeholder.jpg;

}

location /api/catalogue/ { proxy\_pass http://catalogue:8080/; }

location /api/user/ { proxy\_pass http://user:8080/; }

location /api/cart/ { proxy\_pass http://cart:8080/; }

location /api/shipping/ { proxy\_pass http://shipping:8080/; }

location /api/payment/ { proxy\_pass http://payment:8080/; }

location /health {

stub\_status on;

access\_log off;

}

}

}

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: frontend

namespace: roboshop

# deployment labels

labels:

component: frontend

project: roboshop

tier: web

spec:

replicas: 1

# These are the labels replica set use to create pod replicas, this should match pod labels

selector:

matchLabels:

component: frontend

project: roboshop

tier: web

# This is pod definition

template:

metadata:

labels:

component: frontend

project: roboshop

tier: web

spec:

containers:

- name: frontend

image: shan2324/frontend:v1

volumeMounts:

- name: nginx-conf

mountPath: /etc/nginx/nginx.conf

subPath: nginx.conf

readOnly: true

volumes:

- name: nginx-conf

configMap:

name: nginx-conf

items:

- key: nginx.conf

path: nginx.conf

---

apiVersion: v1

kind: Service

metadata:

name: frontend

namespace: roboshop

labels:

component: frontend

project: roboshop

tier: web

spec:

type: LoadBalancer

selector:

component: frontend

project: roboshop

tier: web

ports:

- protocol: TCP

port: 80 # service port

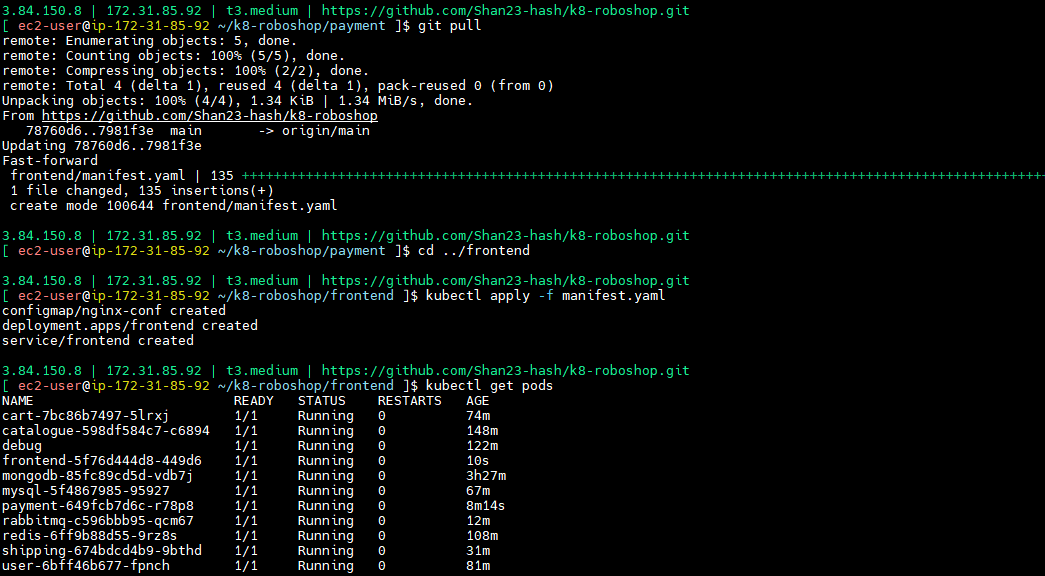
targetPort: 8080 # container port

--> push and pull the code

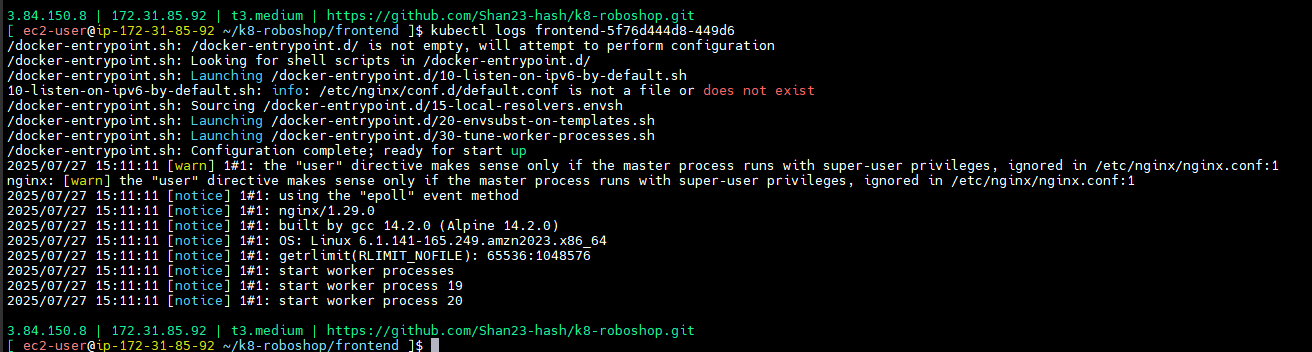
**--> cd ../frontend**

**--> kubectl apply -f manifest.yaml**

**--> kubectl get pods**

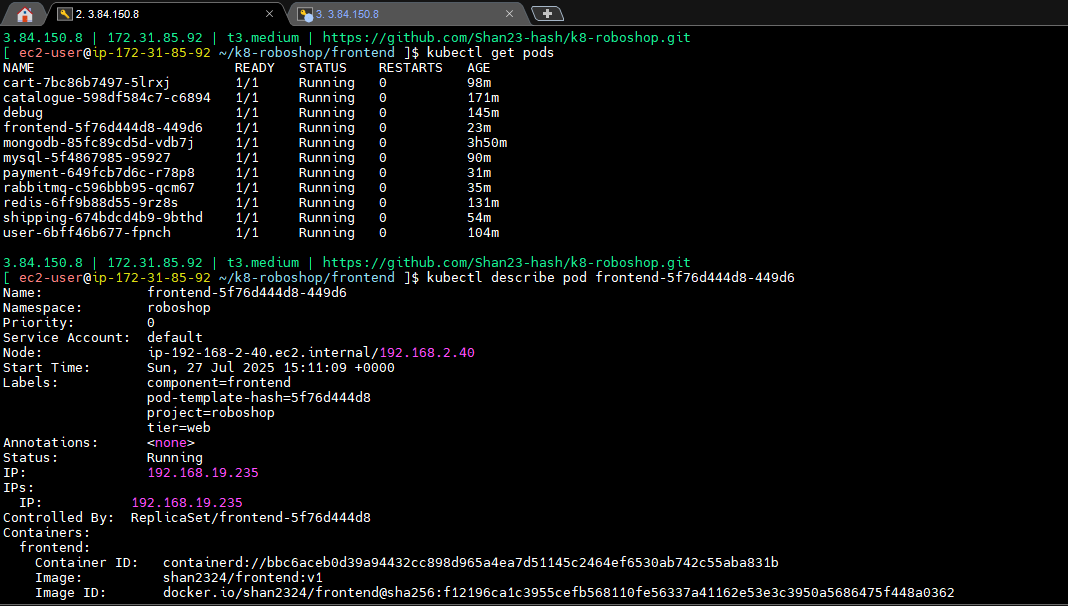


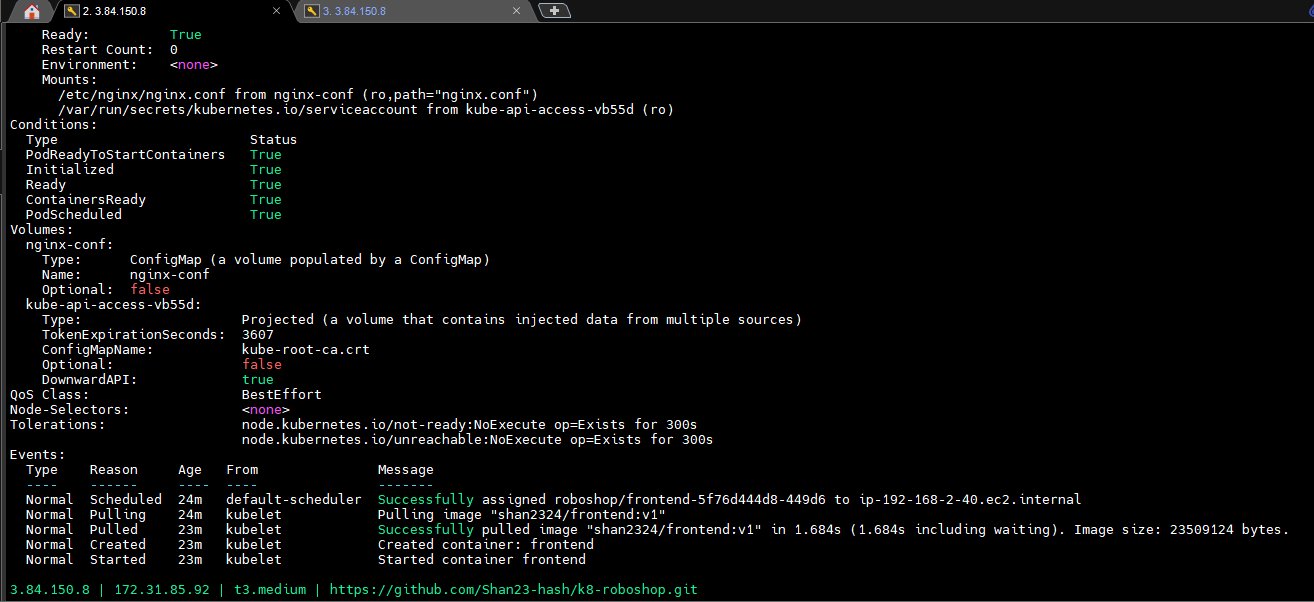
**--> kubectl logs frontend-5f76d444d8-449d6**



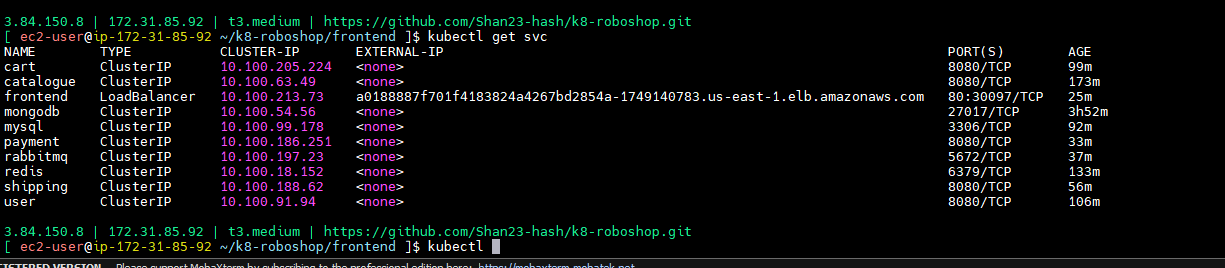
**--> kubectl get pods**

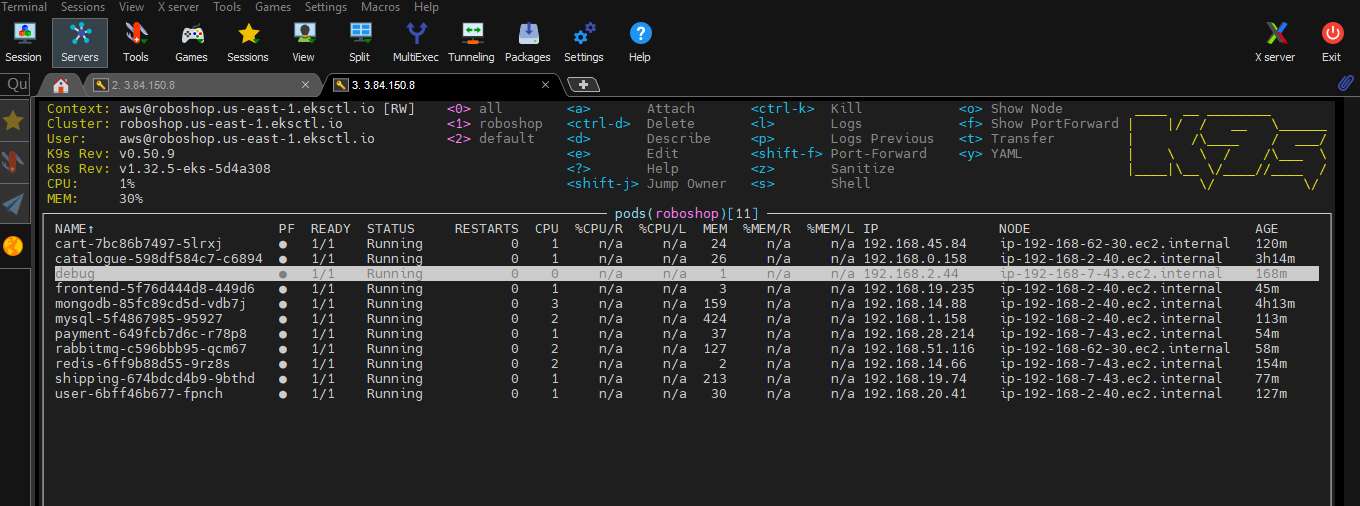
**--> kubectl describe pod frontend-5f76d444d8-449d6**





**--> kubectl get svc**





**--> <http://a0188887f701f4183824a4267bd2854a-1749140783.us-east-1.elb.amazonaws.com/>**

