SESSION - 53

We are completed docker file instructions.

--> ADD - in local files and folders to copy the in docker images.

--> COPY - same like add but extra advantage we can download the files from inetrnet.it can directly enter the files into the docker images.

--> ARG - in build time will provide images to docker image. ARG has some speciality it can be the first instruction before to supply the version to the from instruction.  
--> CMD -- container started time it will run. It can be overridden.

--> ENTRYPOINT -- it can not be overridden for better we can use cmd entrypoint together. Entry point you can mention the command cmd can provide the arguments it can always override documents or command line.

--> ENV - environment variables

--> EXPOSE - it will provide the information of the ports open by the container by the user.

--> LABEL - key value pairs

--> ONBUILD - image developer can put something conditions through the onbuilding sections.

--> RUN - runs at the time of image building. Can have multiple run instructions usually we use run instruction to install few packages or to configure the image.

--> USER - which user the container runs, can usually not root users we mentioned user instruction because will use in security purpose.

--> WORKDIR - this instruction we used to mentioned the working directory inside image or container.

### Dockerfile Instructions

**ADD**  
Used to copy local files and folders into the Docker image. It can also handle remote URLs and extract compressed files automatically.

**COPY**  
Similar to ADD, but more predictable and limited to copying local files/directories. It doesn't support auto-extraction of archives or downloading from URLs.

**ARG**  
Defines build-time variables. These can be used during the image build process. ARG can appear before the FROM instruction to define variables like the base image version.

**CMD**  
Specifies the default command to run when a container starts. This can be overridden by the user when launching the container.

**ENTRYPOINT**  
Defines the main command that always runs when the container starts. Unlike CMD, it’s not easily overridden. It’s best to use ENTRYPOINT for the main process and CMD for default arguments.

**ENV**  
Sets environment variables in the container, which can be used by the application or during image build.

**EXPOSE**  
Indicates the ports that the container will listen on at runtime. It's informational only — it doesn't actually publish the port.

**LABEL**  
Adds metadata to the image as key-value pairs (e.g., version, description, maintainer).

**ONBUILD**  
Sets instructions that will be executed when the image is used as a base for another image. Useful for image builders to define preset behaviors.

**RUN**  
Executes commands at build time, typically used to install packages or perform image configuration. Multiple RUN instructions are allowed.

**USER**  
Specifies the user to run the container processes. It's good practice to avoid running containers as the root user for security reasons.

**WORKDIR**  
Sets the working directory for subsequent instructions (RUN, CMD, ENTRYPOINT, etc.). If the directory doesn’t exist, it will be created.

**/docker/docker.tf**

resource "aws\_instance" "docker" {

ami = local.ami\_id

#instance\_type = "t3.micro"

vpc\_security\_group\_ids = [aws\_security\_group.allow\_all\_docker.id]

instance\_type = "t3.medium"

# need more for terraform

root\_block\_device {

volume\_size = 50

volume\_type = "gp3" # or "gp2", depending on your preference

}

user\_data = file("docker.sh")

#iam\_instance\_profile = "TerraformAdmin"

tags = {

Name = "${var.project}-${var.environment}-docker"

}

}

resource "aws\_security\_group" "allow\_all\_docker" {

name = "allow\_all\_docker"

description = "allow all traffic"

ingress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

ipv6\_cidr\_blocks = ["::/0"]

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

ipv6\_cidr\_blocks = ["::/0"]

}

lifecycle {

create\_before\_destroy = true

}

tags = {

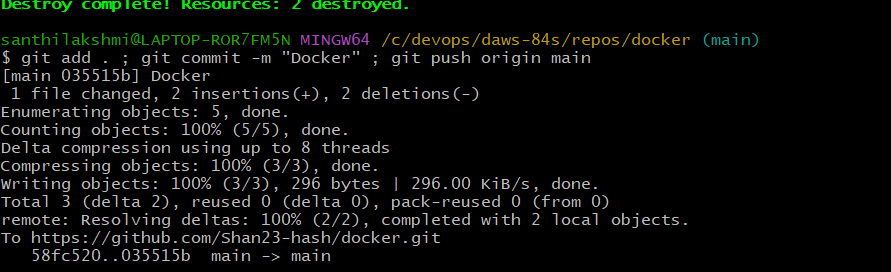
Name = "allow-all-docker"

}

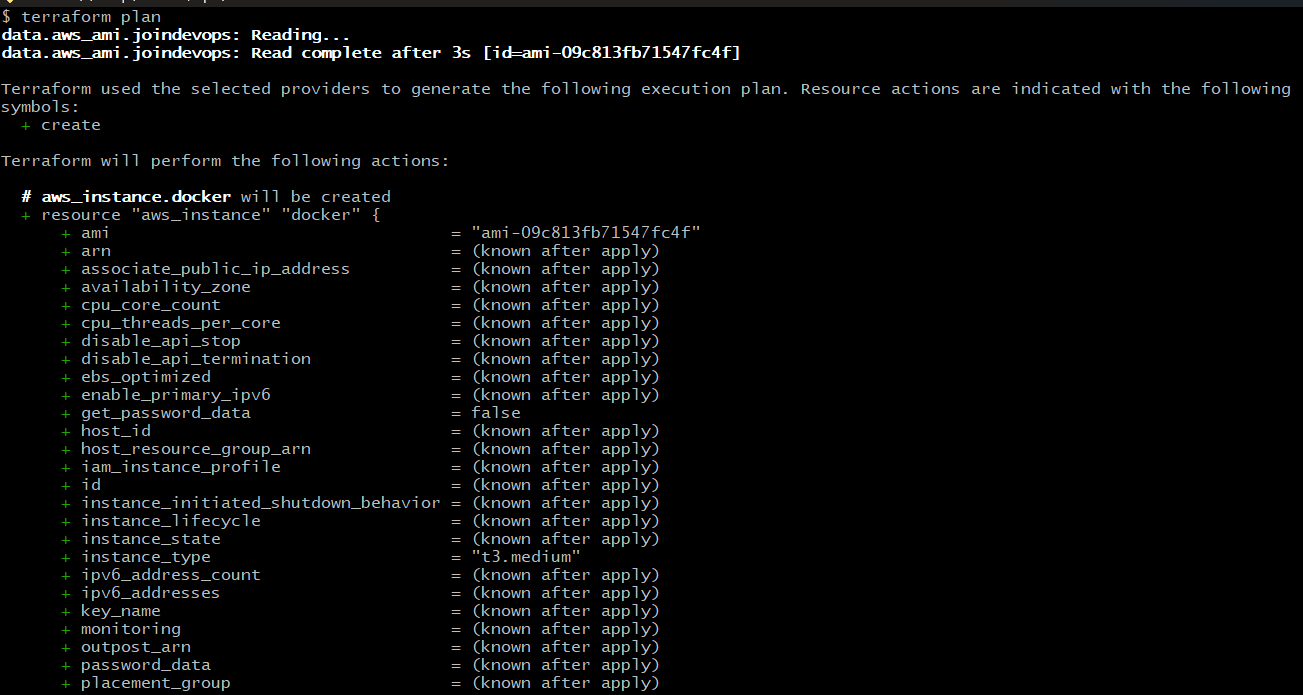
}

**cd /c/devops/daws-84s/repos/docker**

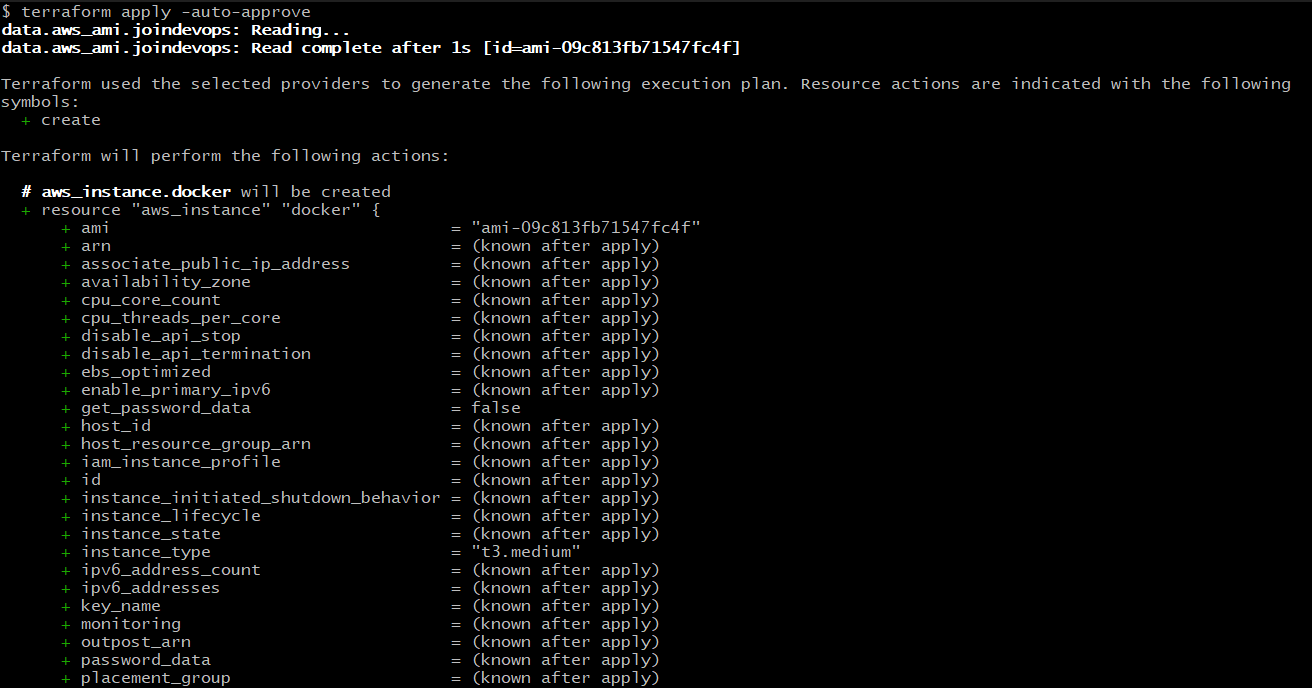
**git add . ; git commit -m "Docker" ; git push origin main**

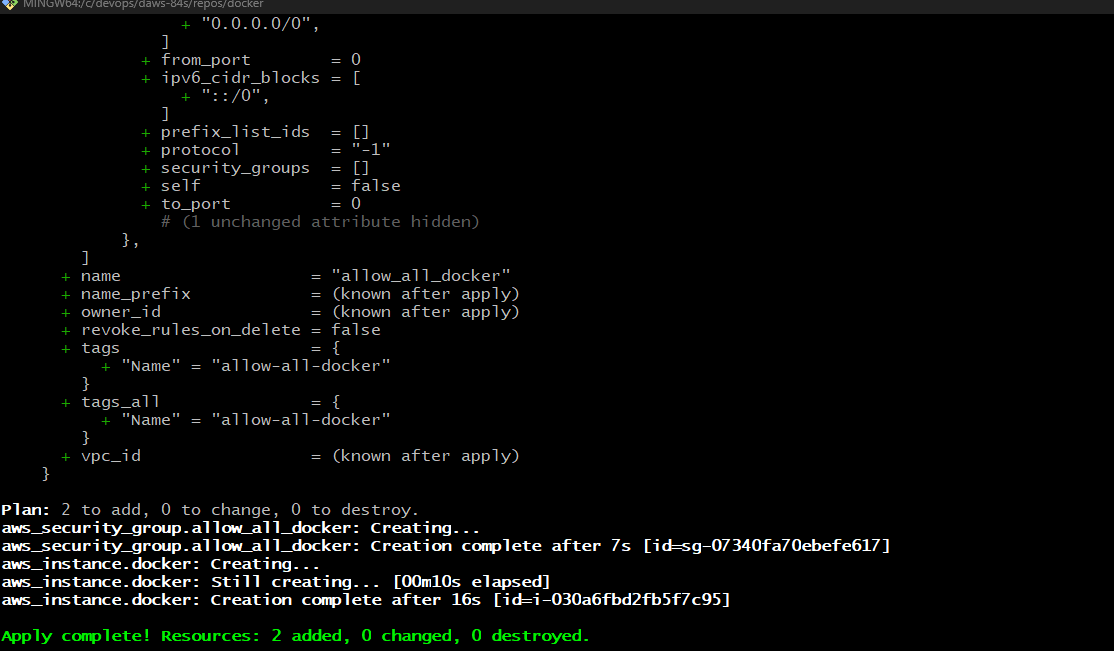


**terraform plan**

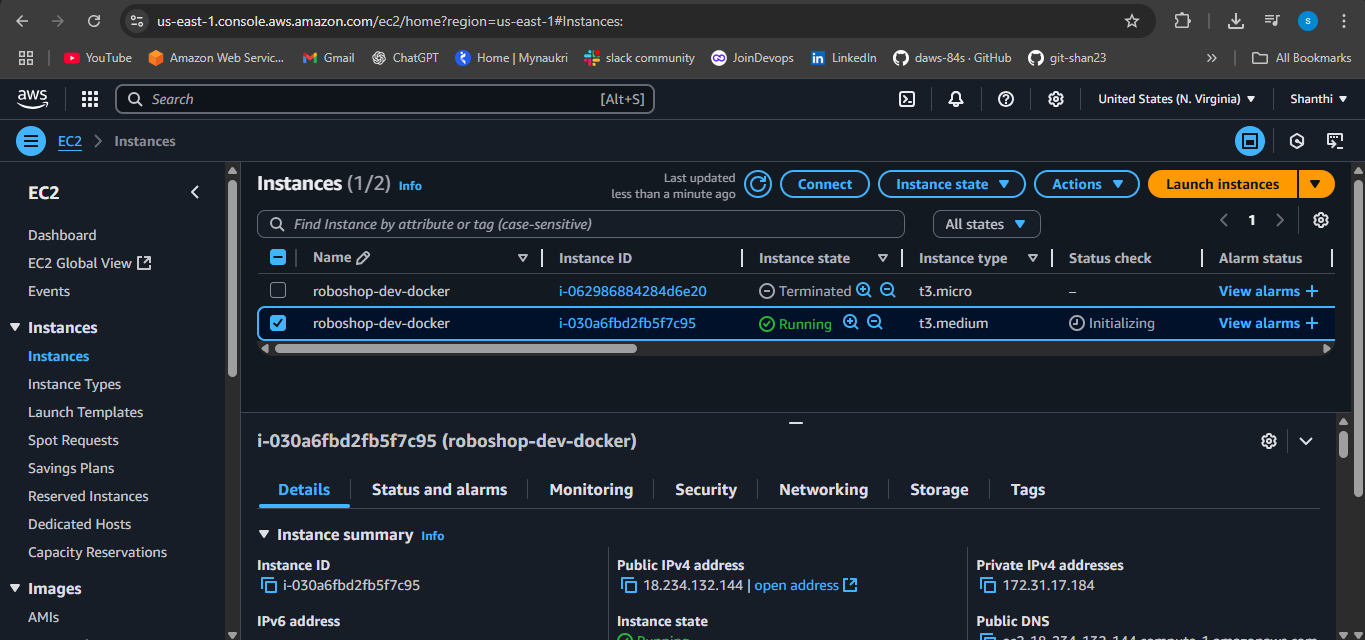


**terraform apply -auto-approve**





Output:



Connect the server



For creating roboshop instances taken t3.medium (big instance)

Because we have many images may be it will not work t3.micro so taken t3.medium.

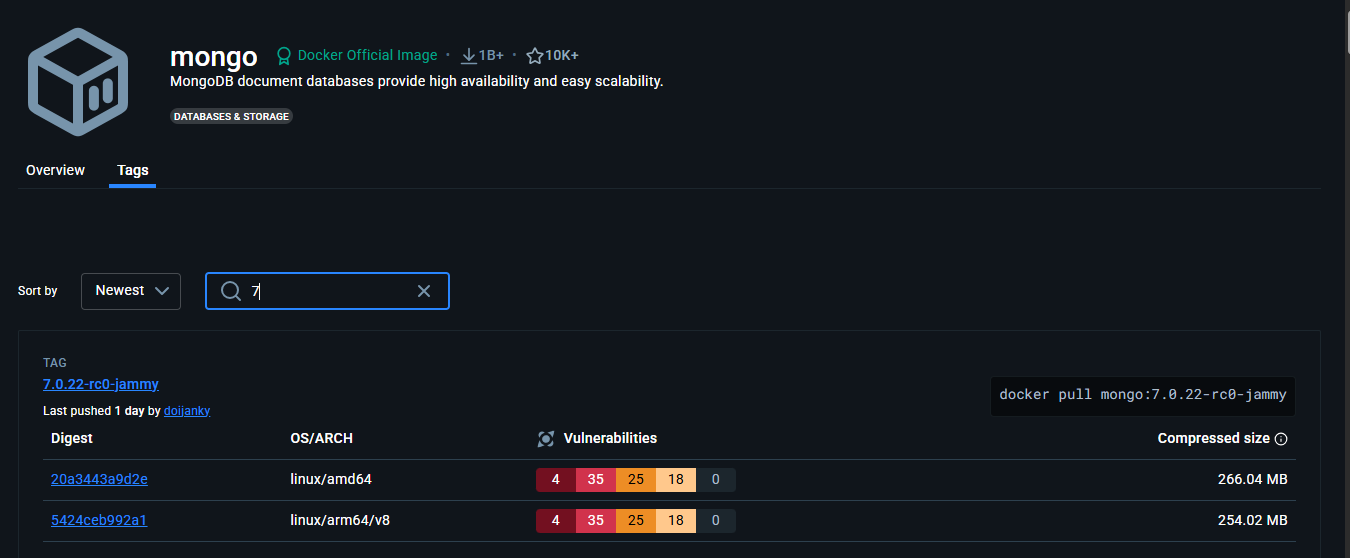
We have instruction right,we have to do docker containerization

First we have to setup mongodb in databases

Mongodb - 7 version

Or mongodb offcial image from hub -- which one is better

If we have office image take that one - we can’t do better than offcial



If I taken like this it can come total mongodb

Create a directory **roboshop-docker**

**mongodb/Dockerfile**

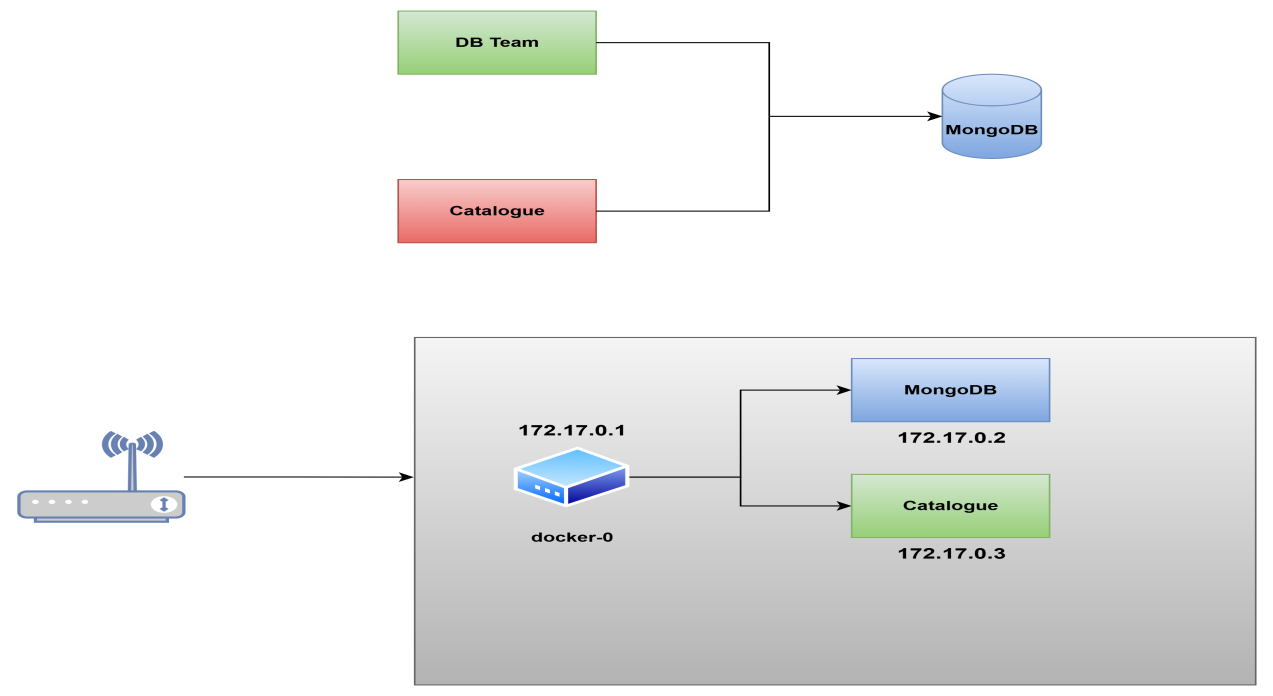
FROM mongo:7.0

--> no need to do install and start that one automatically it will run that is the image.

In mondodb we have to keep default for categories.

Already we are discussed there is a database team.

**Diagram**



Say this is a mongodb server

Usually database team also load data to this.

We are loaded with catalogue.

But original responsibility is database team.

Database team not there so we did.

**Initializing a fresh instance**

When a container is started for the first time it will execute files with extensions .sh and .js that are found in /docker-entrypoint-initdb.d. Files will be executed in alphabetical order. .js files will be executed by mongosh (mongo on versions below 6) using the database specified by the MONGO\_INITDB\_DATABASE variable, if it is present, or test otherwise. You may also switch databases within the .js script.

This location like that you keep means static pic docker will automatically will pickup and run.

We saw in catalogue for loading some scripts.

That one also we can load directly from here.

Download catalogue load.

**https://roboshop-artifacts.s3.amazonaws.com/catalogue-v3.zip**

This is catalogue in this we have db script

Java or shell you can keep in one location docker will load that one automatically.

we are keeping in script for allocation.docker will run automatically.

Copy and paste it in **master-datajs**

//

// Products

//

db = db.getSiblingDB('catalogue');

db.products.insertMany([

    {sku: 'Watson', name: 'Watson', description: 'Probably the smartest AI on the planet', price: 2001, instock: 2, categories: ['Artificial Intelligence']},

    {sku: 'Ewooid', name: 'Ewooid', description: 'Fully sentient assistant', price: 200, instock: 0, categories: ['Artificial Intelligence']},

    {sku: 'HPTD', name: 'High-Powered Travel Droid', description: 'Traveling to the far reaches of the Galaxy? You need this for protection. Comes in handy when you are lost in space', price: 1200, instock: 12, categories: ['Robot']},

    {sku: 'UHJ', name: 'Ultimate Harvesting Juggernaut', description: 'Extraterrestrial vegetation harvester', price: 5000, instock: 10, categories: ['Robot']},

    {sku: 'EPE', name: 'Extreme Probe Emulator', description: 'Versatile interface adapter for hacking into systems', price: 953, instock: 1, categories: ['Robot']},

    {sku: 'EMM', name: 'Exceptional Medical Machine', description: 'Fully automatic surgery droid with exceptional bedside manner', price: 1024, instock: 1, categories: ['Robot']},

    {sku: 'SHCE', name: 'Strategic Human Control Emulator', description: 'Diplomatic protocol assistant', price: 300, instock: 12, categories: ['Robot']},

    {sku: 'RED', name: 'Responsive Enforcer Droid', description: 'Security detail, will gaurd anything', price: 700, instock: 5, categories: ['Robot']},

    {sku: 'RMC', name: 'Robotic Mining Cyborg', description: 'Excellent tunneling capability to get those rare minerals', price: 42, instock: 48, categories: ['Robot']},

    {sku: 'STAN-1', name: 'Stan', description: 'Observability guru', price: 67, instock: 1000, categories: ['Robot', 'Artificial Intelligence']},

    {sku: 'CNA', name: 'Cybernated Neutralization Android', description: 'Is your spaceship a bit whiffy? This little fellow will bring a breath of fresh air', price: 1000, instock: 0, categories: ['Robot']}

]);

// full text index for searching

db.products.createIndex({

    name: "text",

    description: "text"

});

// unique index for product sku

db.products.createIndex(

    { sku: 1 },

    { unique: true }

);

Which instruction I can use here - copy or add

**mongodb/Dockerfile**

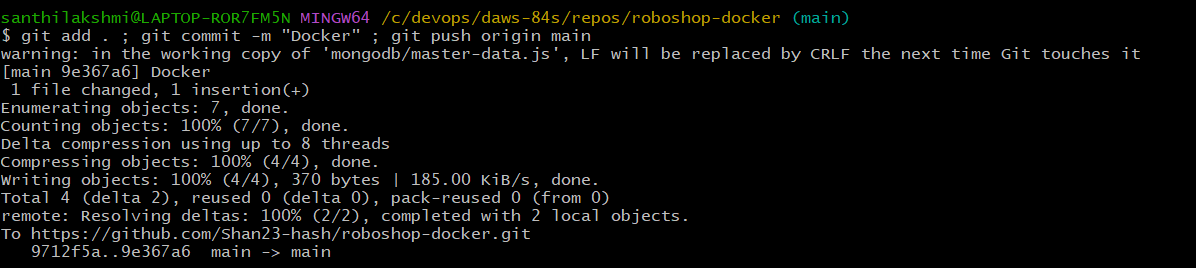
FROM mongo:7.0

COPY \*.js /docker-entrypoint-initdb.d

Simple this all queries will run automatically no need to run from catalogue directly you can run here .

These all will come from image in by default data also.

**git add . ; git commit -m "Docker" ; git push origin main**



What is the command to build the image.

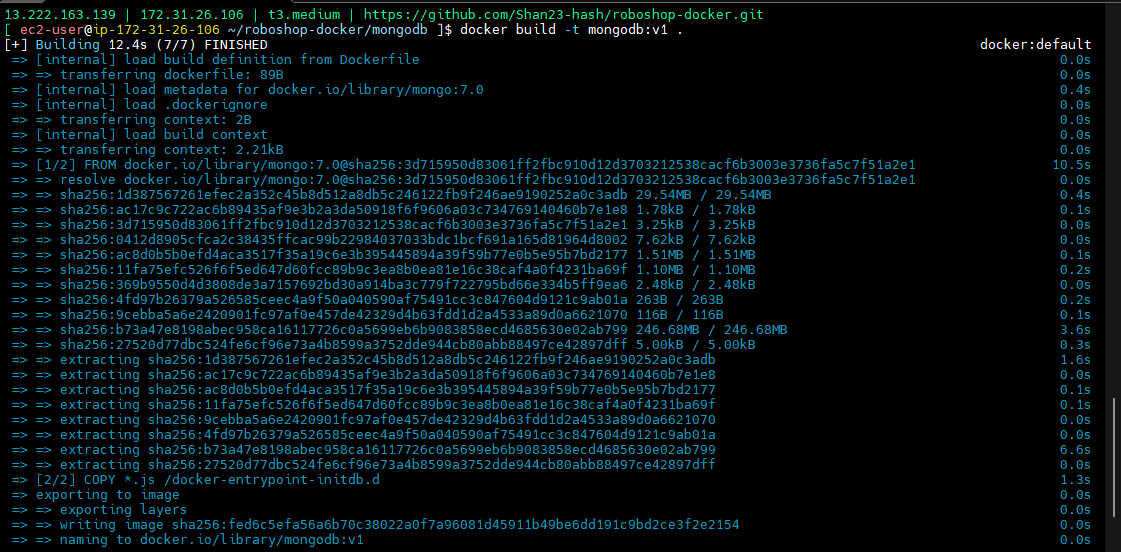
**git clone <https://github.com/Shan23-hash/roboshop-docker.git>**

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

**docker login -u shan2324**

**cd ~/roboshop-docker/mongodb**

**docker build -t mongodb:v1 .**



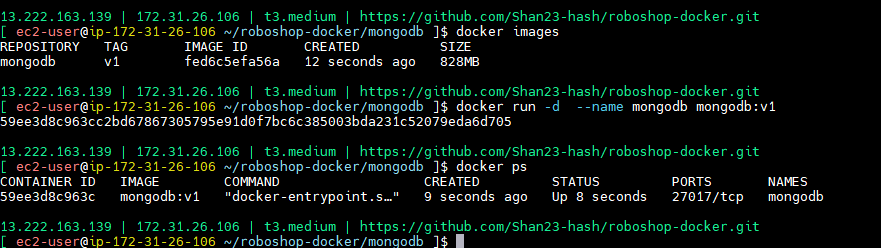
**docker images**

**docker run -d --name mongodb mongodb:v1**

-p : for external exposing we can

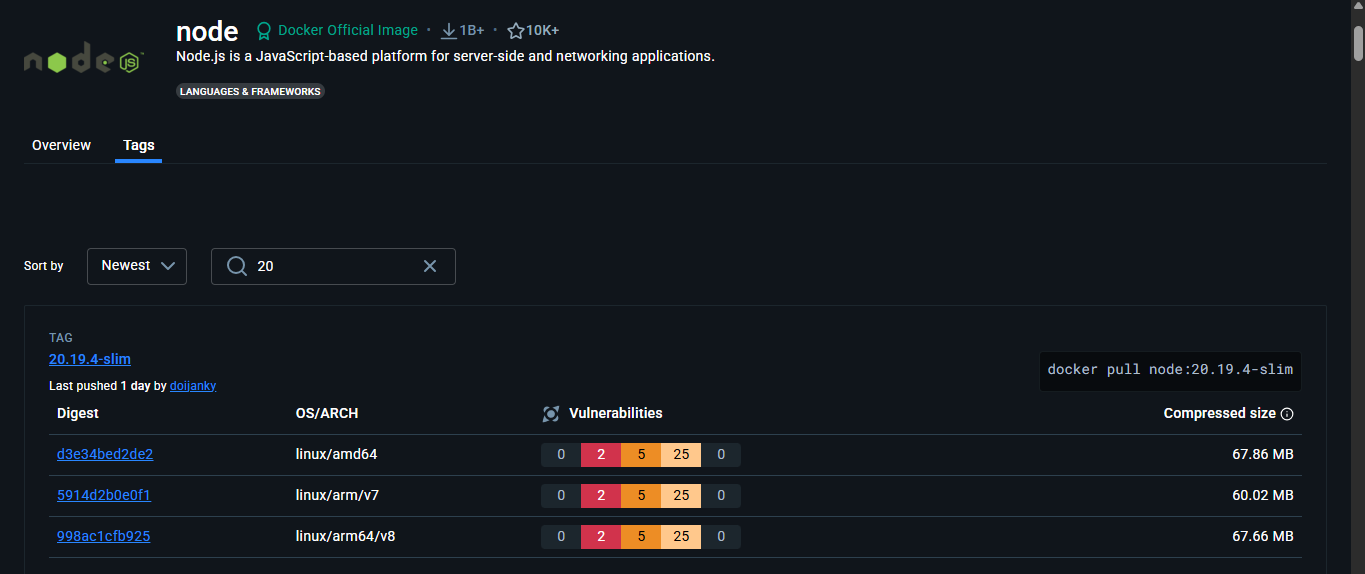
But here we are not do mongodb so ne need -p

**docker ps**



**Catalogue**

* In catalogue one running application from VM while migrating application to containerization. Will change like micro services.
* I can take directly take node js image. We are taking dirctly means underline os no need.
* Application is running or not that is the matter.
* Nodejs - taken their base os - Debain/ubuntu - on that installed node js.
* Debain/ubuntu + node js
* No need to run us node js. Wherever you will do that taking image only node js, what is there in base os we don’t want
* Require we can check no problem but we need node js whether application is running or not.
* We are taking official images so not a problem.
* We are leave os - its increased most responsibilities.
* If you selected os that configuration is o big task.
* Docker will do this heavy thing.
* Here node js is there we need 20 version.
* Here already we have 20



We have many version:20

**package.json**

{

  "name": "catalogue",

  "version": "1.0.0",

  "description": "product catalogue REST API",

  "main": "server.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1"

  },

  "author": "SteveW",

  "license": "Apache-2.0",

  "dependencies": {

    "body-parser": "^1.18.1",

    "express": "^4.15.4",

    "mongodb": "^4.7.0",

    "pino": "^5.10.8",

    "express-pino-logger": "^4.0.0",

    "pino-pretty": "^2.5.0",

    "@instana/collector": "^1.132.2"

  }

}

**server.js**

const instana = require('@instana/collector');

// init tracing

// MUST be done before loading anything else!

instana({

    tracing: {

        enabled: true

    }

});

const { MongoClient, ObjectId } = require('mongodb');

const bodyParser = require('body-parser');

const express = require('express');

const pino = require('pino');

const expPino = require('express-pino-logger');

const logger = pino({

    level: 'info',

    prettyPrint: false,

    useLevelLabels: true

});

const expLogger = expPino({

    logger: logger

});

// MongoDB

let db;

let collection;

let mongoConnected = false;

const app = express();

app.use(expLogger);

app.use((req, res, next) => {

    res.set('Timing-Allow-Origin', '\*');

    res.set('Access-Control-Allow-Origin', '\*');

    next();

});

app.use((req, res, next) => {

    let dcs = [

        "asia-northeast2",

        "asia-south1",

        "europe-west3",

        "us-east1",

        "us-west1"

    ];

    let span = instana.currentSpan();

    span.annotate('custom.sdk.tags.datacenter', dcs[Math.floor(Math.random() \* dcs.length)]);

    next();

});

app.use(bodyParser.urlencoded({ extended: true }));

app.use(bodyParser.json());

app.get('/health', (req, res) => {

    var stat = {

        app: 'OK',

        mongo: mongoConnected

    };

    res.json(stat);

});

// all products

app.get('/products', (req, res) => {

    if (mongoConnected) {

        collection.find({}).toArray().then((products) => {

            res.json(products);

        }).catch((e) => {

            req.log.error('ERROR', e);

            res.status(500).send(e);

        });

    } else {

        req.log.error('database not available');

        res.status(500).send('database not available');

    }

});

// product by SKU

app.get('/product/:sku', (req, res) => {

    if (mongoConnected) {

        // optionally slow this down

        const delay = process.env.GO\_SLOW || 0;

        setTimeout(() => {

            collection.findOne({ sku: req.params.sku }).then((product) => {

                req.log.info('product', product);

                if (product) {

                    res.json(product);

                } else {

                    res.status(404).send('SKU not found');

                }

            }).catch((e) => {

                req.log.error('ERROR', e);

                res.status(500).send(e);

            });

        }, delay);

    } else {

        req.log.error('database not available');

        res.status(500).send('database not available');

    }

});

// products in a category

app.get('/products/:cat', (req, res) => {

    if (mongoConnected) {

        collection.find({ categories: req.params.cat }).sort({ name: 1 }).toArray().then((products) => {

            if (products) {

                res.json(products);

            } else {

                res.status(404).send('No products for ' + req.params.cat);

            }

        }).catch((e) => {

            req.log.error('ERROR', e);

            res.status(500).send(e);

        });

    } else {

        req.log.error('database not available');

        res.status(500).send('database not available');

    }

});

// all categories

app.get('/categories', (req, res) => {

    if (mongoConnected) {

        collection.distinct('categories').then((categories) => {

            res.json(categories);

        }).catch((e) => {

            req.log.error('ERROR', e);

            res.status(500).send(e);

        });

    } else {

        req.log.error('database not available');

        res.status(500).send('database not available');

    }

});

// search name and description

app.get('/search/:text', (req, res) => {

    if (mongoConnected) {

        collection.find({ '$text': { '$search': req.params.text } }).toArray().then((hits) => {

            res.json(hits);

        }).catch((e) => {

            req.log.error('ERROR', e);

            res.status(500).send(e);

        });

    } else {

        req.log.error('database not available');

        res.status(500).send('database not available');

    }

});

// set up Mongo

async function mongoConnect() {

    try {

        const mongoURL = process.env.MONGO\_URL || 'mongodb://mongodb:27017/catalogue';

        const client = await MongoClient.connect(mongoURL, { useNewUrlParser: true, useUnifiedTopology: true });

        db = client.db('catalogue');

        collection = db.collection('products');

        mongoConnected = true;

        logger.info('MongoDB connected');

    } catch (error) {

        mongoConnected = false;

        logger.error('ERROR', error);

        setTimeout(mongoLoop, 2000);

    }

}

// mongodb connection retry loop

function mongoLoop() {

    mongoConnect().catch((e) => {

        logger.error('ERROR', e);

        setTimeout(mongoLoop, 2000);

    });

}

mongoLoop();

// fire it up!

const port = process.env.CATALOGUE\_SERVER\_PORT || '8080';

app.listen(port, () => {

    logger.info('Started on port', port);

});

**catalogue/Dockerfile**

FROM node:20

WORKDIR /opt/server

COPY package.json .

COPY \*.js .

RUN npm install

ENV MONGO="true" \

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

CMD ["node","server.js"]

--> We need to create a one directory for application.

--> location you use for to download the extra-content

--> download the code and install dependencies ofcourse systemctl it will not work.

--> systemctl we can directly.

--> already we have catalogue code package.json and server.js. Paste it in catalogue.

--> No need DB

--> this is applicatio code.

--> here we have one file but big projects 20 -30 os it may be there.

--> but everything in json and server.js/

--> copy package.json inti this current directory.

--> copy server.js

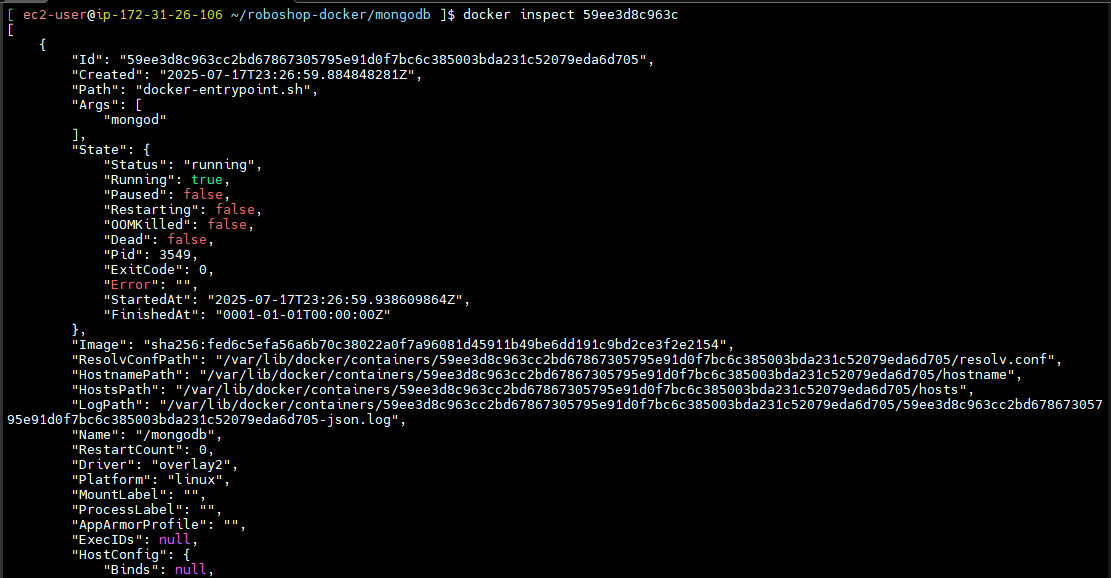
--> install dependencies. Insytalling command is npm install. Which instruction I have to use RUN npm install.

--> dependencies installation also completed.

--> mentioned environment setup.

--> env instruction, multiple env need means take back slash \

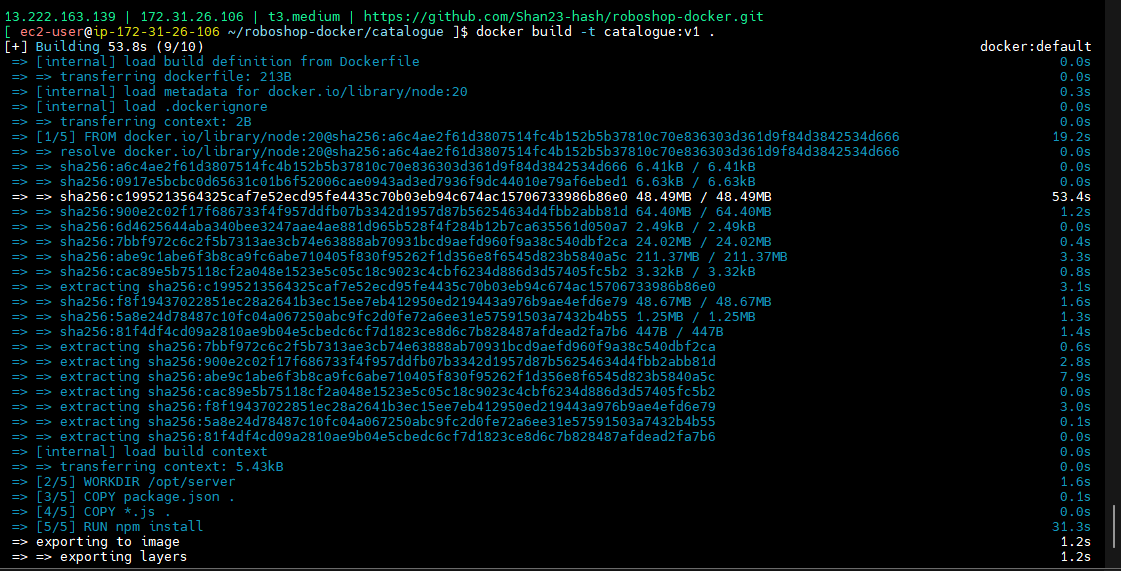
**docker inspect 59ee3d8c963c**



--> how can I start the container - cmd

--> one application containerzation is very easy.

**docker build -t catalogue:v1 .**

****

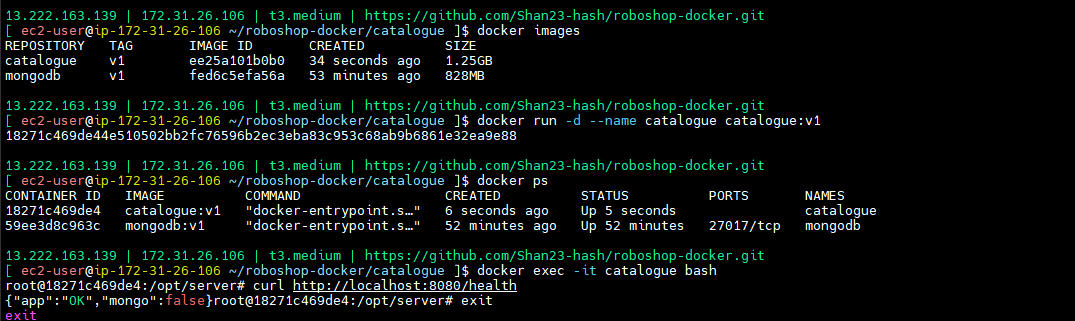
**docker images**

**docker run -d --name catalogue catalogue:v1**

**docker ps**

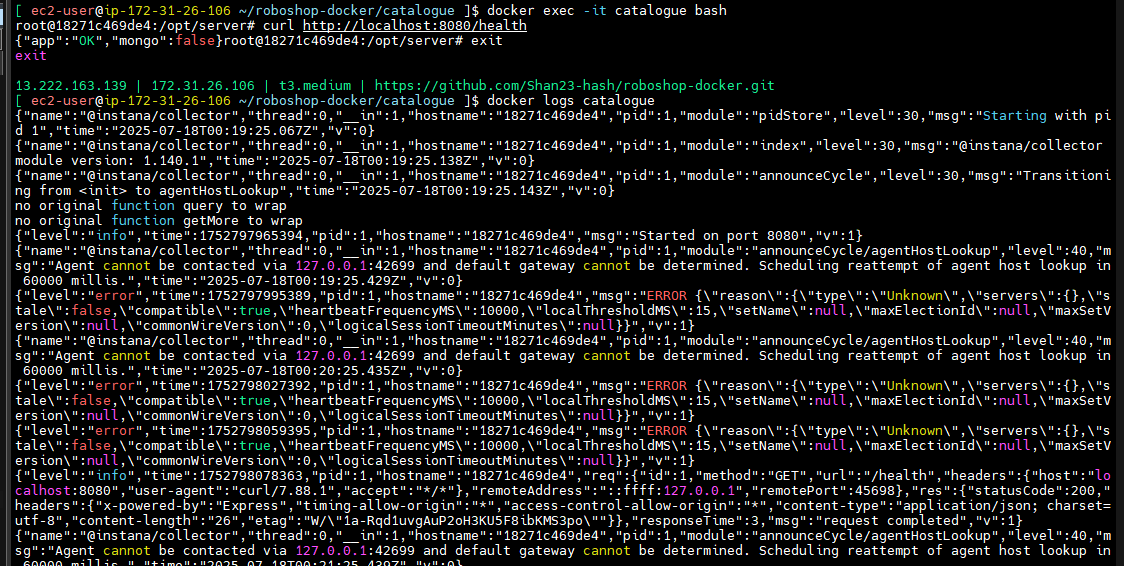
**docker exec -it catalogue bash**

**curl http://localhost:8080/health**



Mongodb not connecting check logs

**docker logs catalogue**



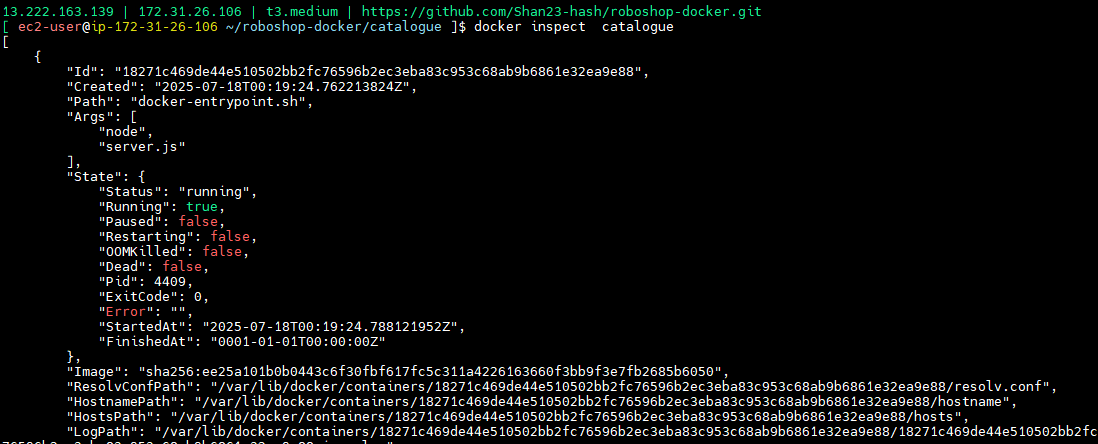
--> catalogue was downloded but mongodb not connected.

--> while downloading docker docker will create a internt.

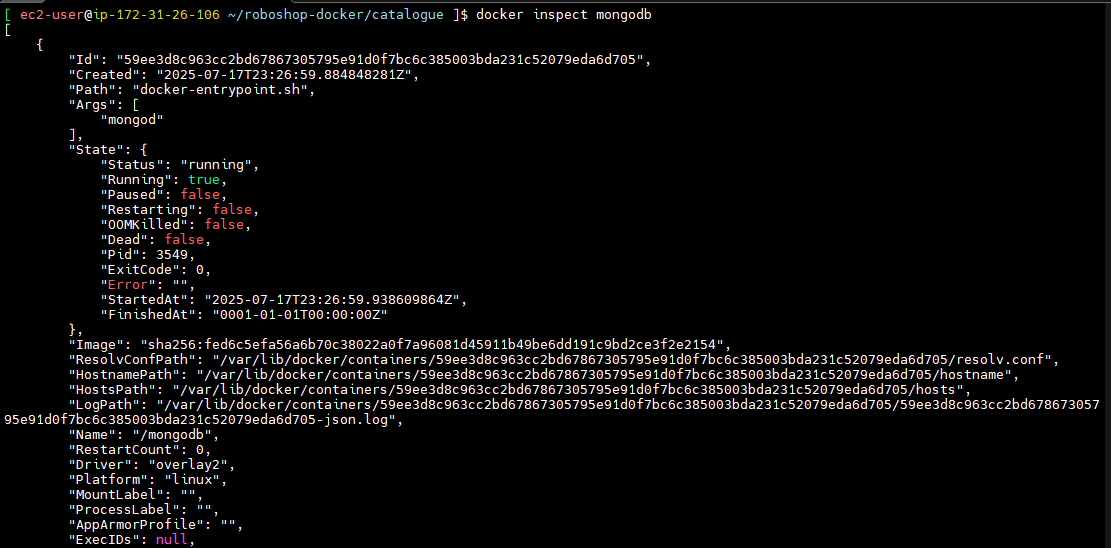
--> docker will create internet on virtual interface.

-- for running every container this one will provide ip address.

**docker inspect catalogue**



**docker inspect mongodb**



--> gate way means modem,

--> everything in same series.

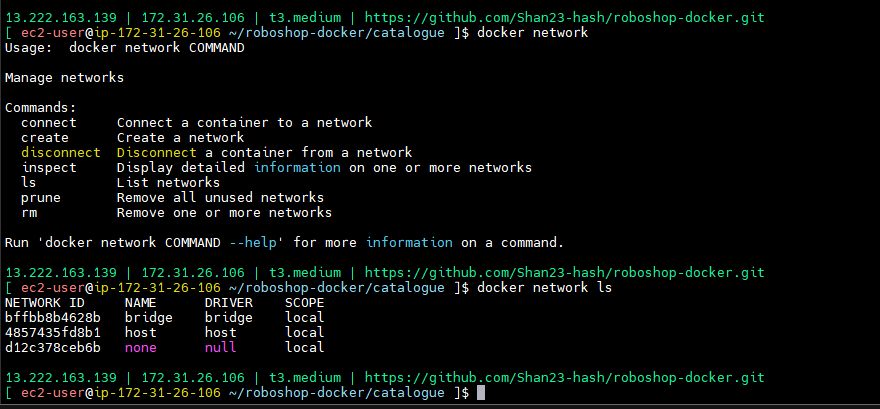
--> mondb and catalogue both are in same net work but they are unable to connect.

--> docker is using default network.

--> in docker we have networks.

**docker network**

**docker network ls**



--> docker by default bridge network.

--> bridge - docker saparately by create virtual interface ip address allocate from mongo db and catalogue.

--> docker is sitting in between the components.

--> sitting like a bridge.

--> if you selected host ip address give from modem

--> directly it will connect host.

--> by default docker creates a bridge network whenever you install docker inside a any server.

--> here already created one bridge network.

--> host network also created . docker uses bridhe network whenever you create a container.

--> this bridge was created default.

--> disadvantage is if you are using default bridge network it will not communicate one ip to another ip.

--> docker said create always saparate network.

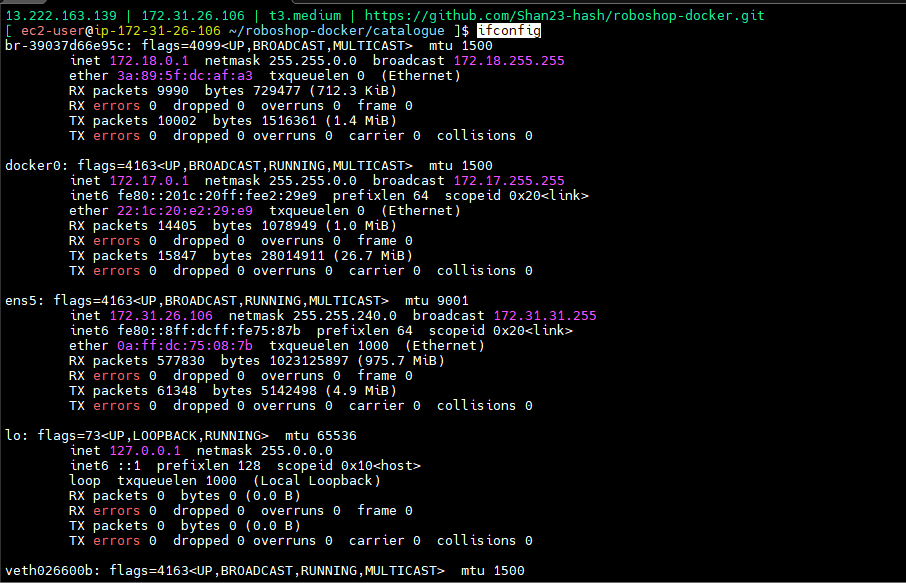
**docker network create roboshop ( for roboshop)**



**-->** expense project means - for expense project create saparately.

--> just like vpc.like non vpc.

**ifconfig**



--> one bridge network was created. 172.18.0.1 series.

--> now if you used this network, this is customized network so in side compounds it will communicate one to one.

--> docker have 2 types of network

1, bridge and 2, host

--> host means directly using host network.

--> bridge means docker creates separate network interface and assign the ip address to containers.

--> docker default bridge network can’t communicate betwwn containers, docker always suggest to create custom bridhe network.

--> docker network create <name>

--> these all in default network. There in bridge network

--> connect that cretaed roboshop network

--> Docker will handle internal dns

**docker ps**

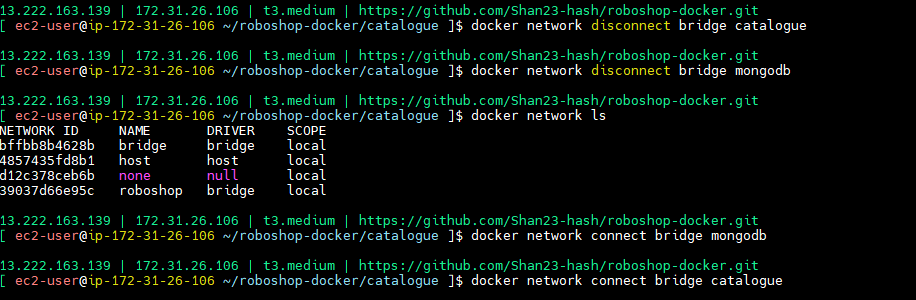
**docker network disconnect bridge catalogue**

**docker network disconnect bridge mongodb**

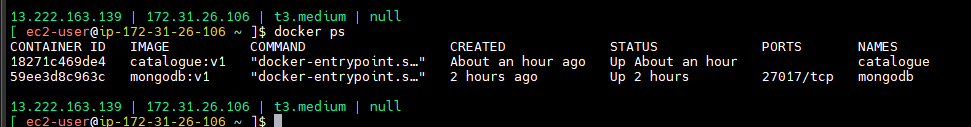
**docker network ls**

**docker network connect bridge mongodb**

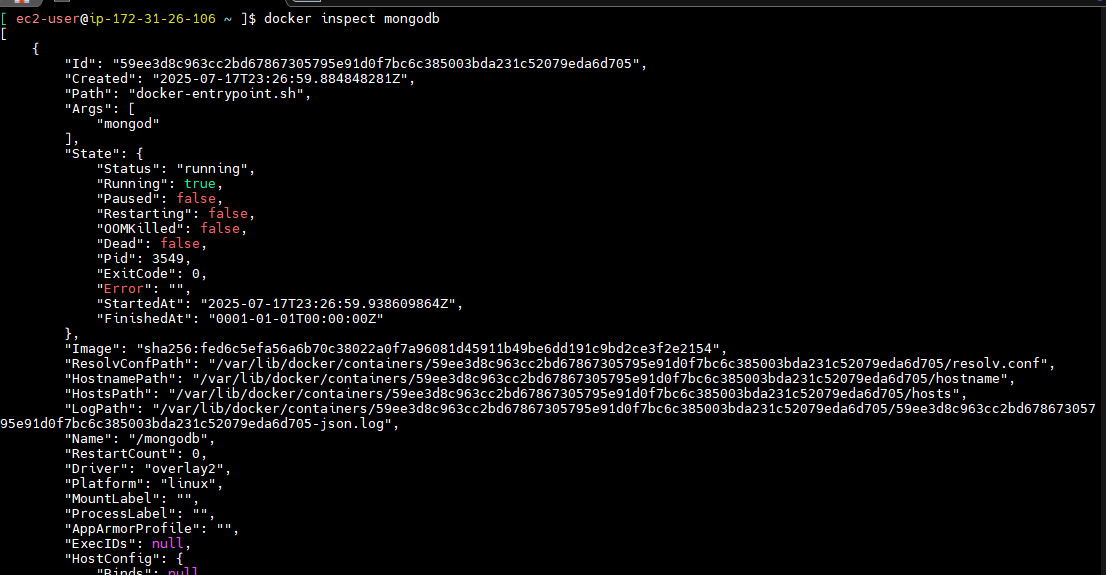
**docker network connect bridge catalogue**

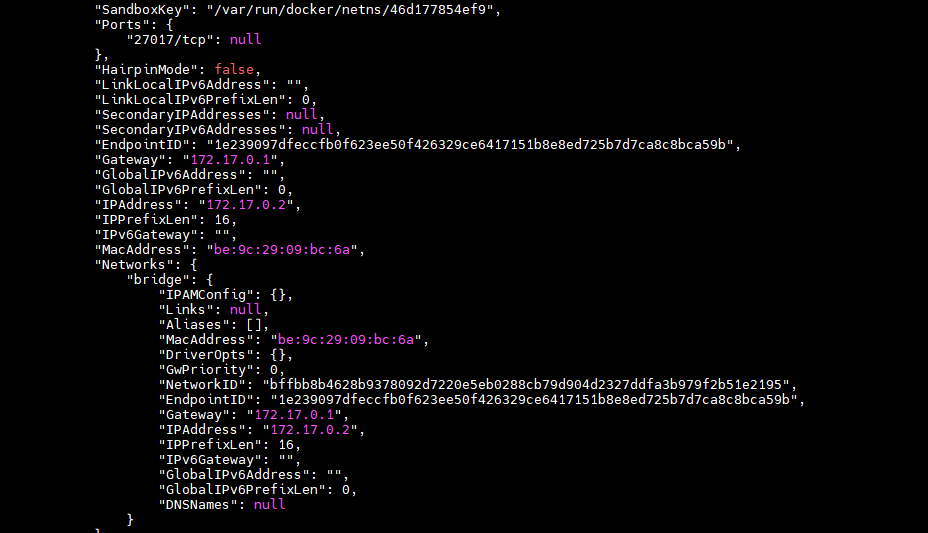


**docker ps**

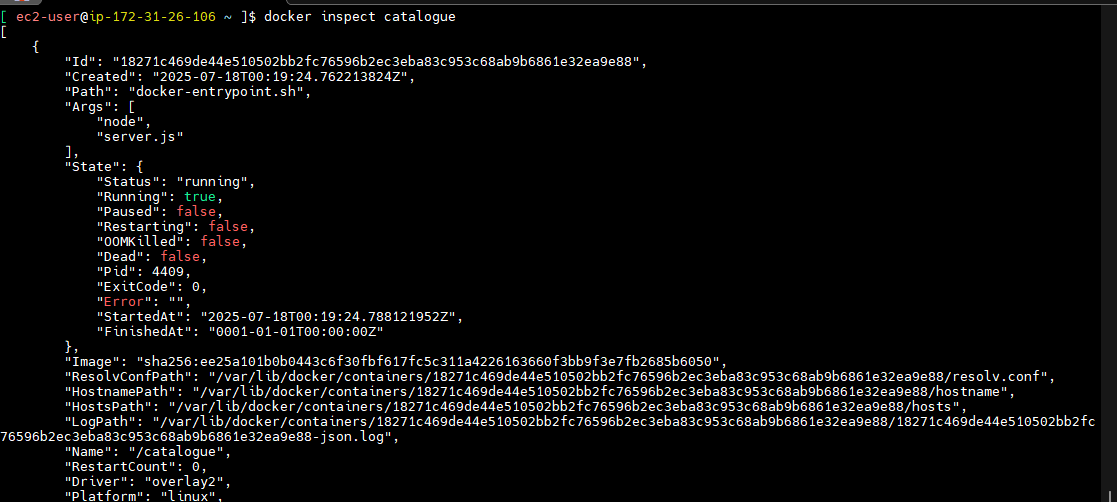


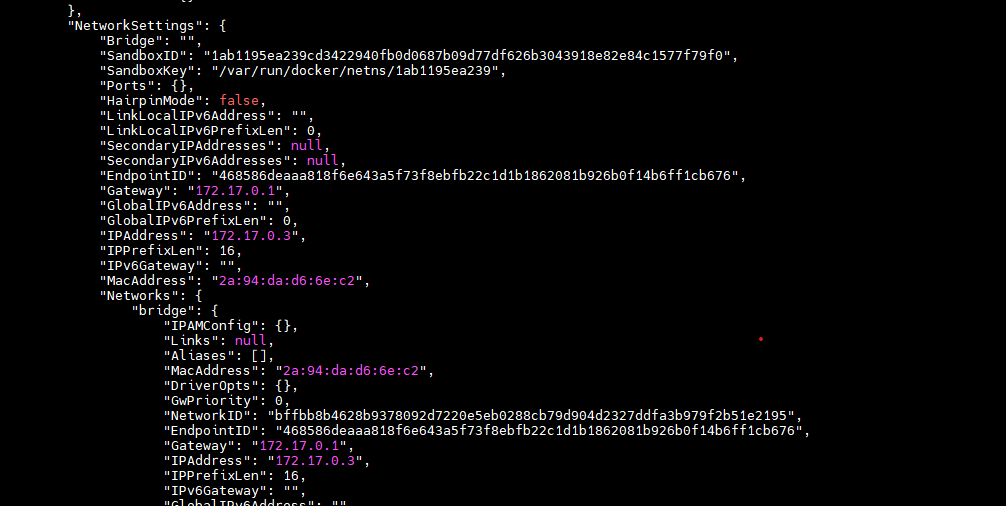
**docker inspect mongodb**





**docker inspect catalogue**





I did not get dns name null only came I given like this

**docker ps**

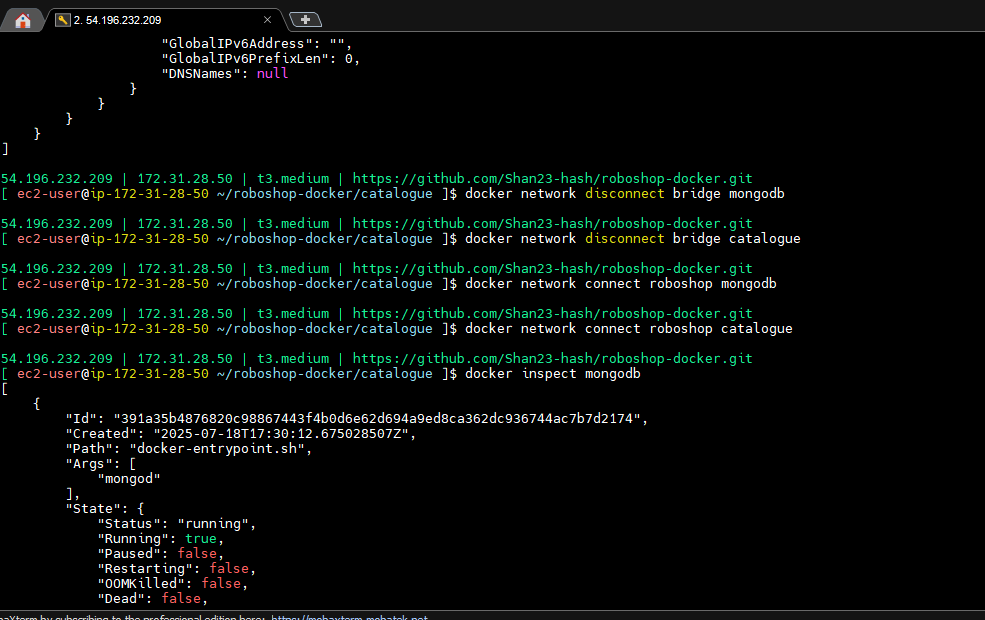
**docker network disconnect bridge mongodb**

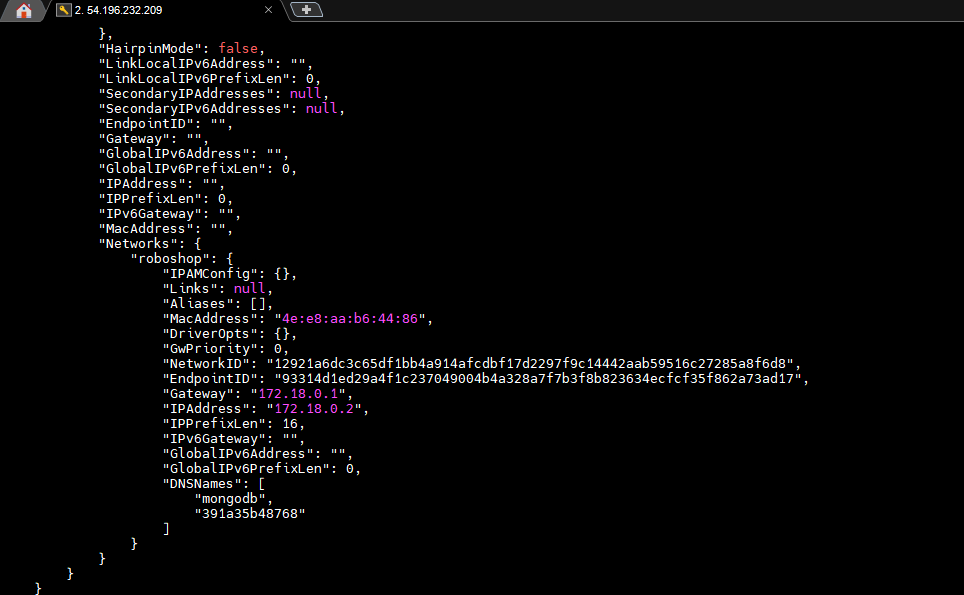
**docker network disconnect bridge catalogue**

**docker network connect roboshop mongodb**

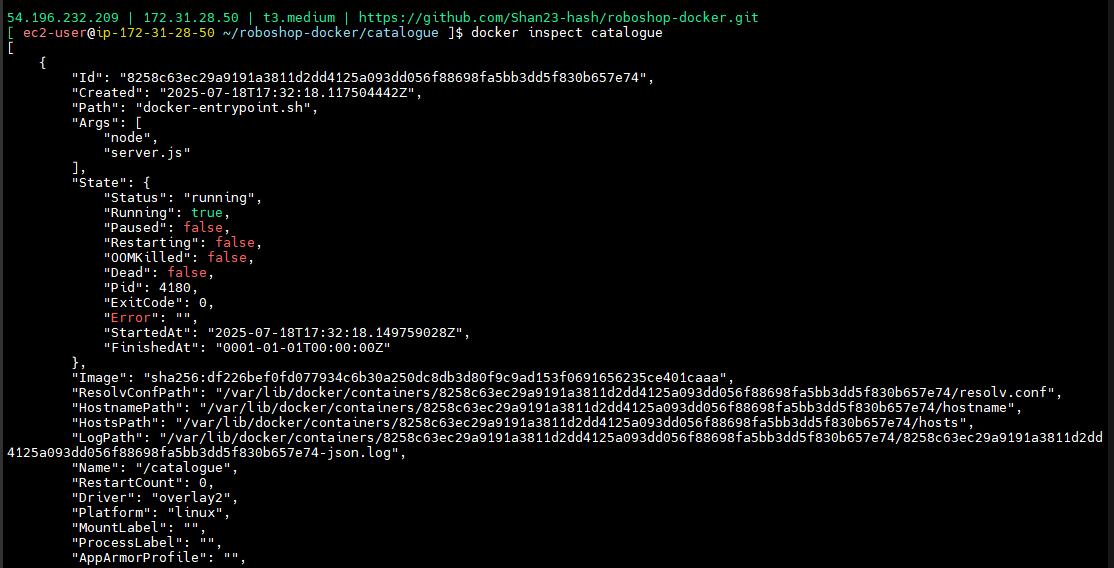
**docker network connect roboshop catalogue**

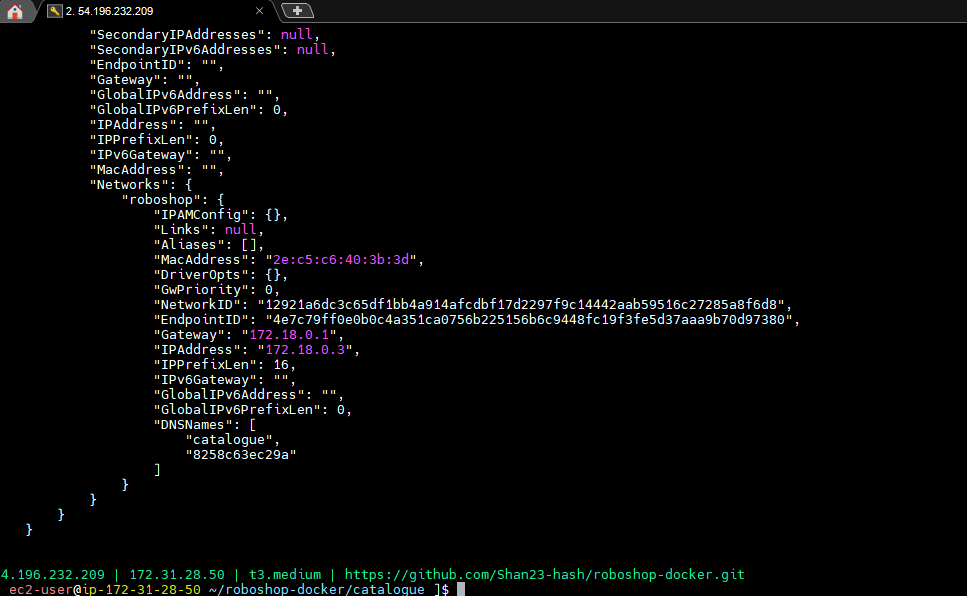
**docker inspect mongodb**





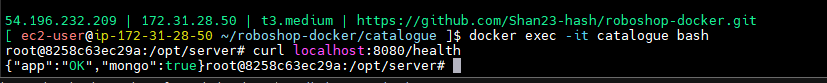
**docker inspect catalogue**





**docker exec -it catalogue bash**

**curl localhost:8080/health**



Now mongodb is connected.

So here we have to use customized network

**REDIS**

We are customizing for redis.

We are not filling data.

No need to write anything. Directly I can write it.

**cd ~/roboshop-docker/catalogue**

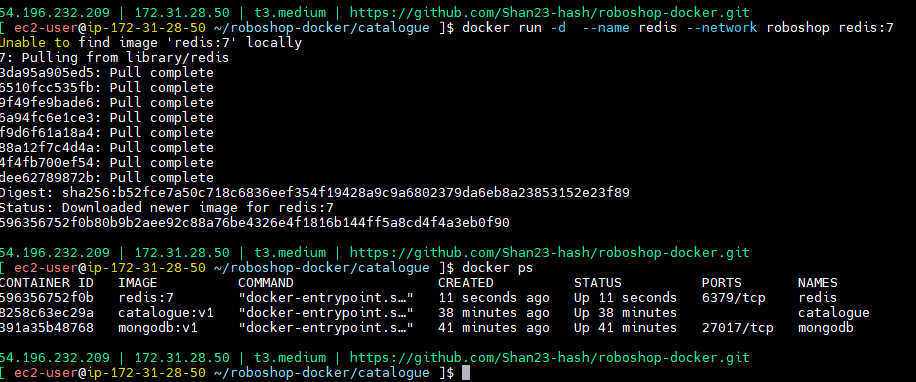
**exit**

**docker run -d --name redis --network roboshop redis:7**

**docker ps**

Of we are not given network that one it will go to network.

If you run this where it will come from image. It’s not there in local so directly it pull from hub.



**USER**

User also node js application.

**user/Dockerfile**

FROM node:20

WORKDIR /opt/server

COPY package.json .

COPY \*.js .

RUN npm install

ENV MONGO="true" \

    REDIS\_URL="redis://redis:6379" \

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

CMD ["node","server.js"]

we have to download user code. Download and extract

**<https://roboshop-artifacts.s3.amazonaws.com/user-v3.zip>**

While doing containerization environment should setup

**user/package.json**

{

  "name": "user",

  "version": "1.0.0",

  "description": "user REST API",

  "main": "server.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1"

  },

  "author": "SteveW",

  "license": "Apache-2.0",

  "dependencies": {

    "body-parser": "^1.18.1",

    "express": "^4.15.4",

    "mongodb": "^4.7.0",

    "redis": "^4.6.4",

    "pino": "^5.10.8",

    "express-pino-logger": "^4.0.0",

    "pino-pretty": "^2.5.0",

    "@instana/collector": "^1.132.2"

  }

}

{

**user/server.js**

const instana = require('@instana/collector');

// init tracing

// MUST be done before loading anything else!

instana({

    tracing: {

        enabled: true

    }

});

const { MongoClient, ObjectId } = require('mongodb');

const { createClient } = require('redis');

const bodyParser = require('body-parser');

const express = require('express');

const pino = require('pino');

const expPino = require('express-pino-logger');

// MongoDB

let db;

let usersCollection;

let ordersCollection;

let mongoConnected = false;

const logger = pino({

    level: 'info',

    prettyPrint: false,

    useLevelLabels: true

});

const expLogger = expPino({

    logger: logger

});

const app = express();

app.use(expLogger);

app.use((req, res, next) => {

    res.set('Timing-Allow-Origin', '\*');

    res.set('Access-Control-Allow-Origin', '\*');

    next();

});

app.use((req, res, next) => {

    let dcs = [

        "asia-northeast2",

        "asia-south1",

        "europe-west3",

        "us-east1",

        "us-west1"

    ];

    let span = instana.currentSpan();

    span.annotate('custom.sdk.tags.datacenter', dcs[Math.floor(Math.random() \* dcs.length)]);

    next();

});

app.use(bodyParser.urlencoded({ extended: true }));

app.use(bodyParser.json());

app.get('/health', (req, res) => {

    const stat = {

        app: 'OK',

        mongo: mongoConnected

    };

    res.json(stat);

});

// use REDIS INCR to track anonymous users

app.get('/uniqueid', async (req, res) => {

    try {

        const r = await redisClient.incr('anonymous-counter');

        res.json({

            uuid: 'anonymous-' + r

        });

    } catch (err) {

        req.log.error('ERROR', err);

        res.status(500).send(err);

    }

});

// check user exists

app.get('/check/:id', async (req, res) => {

    if (mongoConnected) {

        try {

            const user = await usersCollection.findOne({ name: req.params.id });

            if (user) {

                res.send('OK');

            } else {

                res.status(404).send('user not found');

            }

        } catch (e) {

            req.log.error(e);

            res.status(500).send(e);

        }

    } else {

        req.log.error('database not available');

        res.status(500).send('database not available');

    }

});

// return all users for debugging only

app.get('/users', async (req, res) => {

    if (mongoConnected) {

        try {

            const users = await usersCollection.find().toArray();

            res.json(users);

        } catch (e) {

            req.log.error('ERROR', e);

            res.status(500).send(e);

        }

    } else {

        req.log.error('database not available');

        res.status(500).send('database not available');

    }

});

app.post('/login', async (req, res) => {

    req.log.info('login', req.body);

    if (req.body.name === undefined || req.body.password === undefined) {

        req.log.warn('credentails not complete');

        res.status(400).send('name or passowrd not supplied');

    } else if (mongoConnected) {

        try {

            const user = await usersCollection.findOne({

                name: req.body.name,

            });

            req.log.info('user', user);

            if (user) {

                if (user.password == req.body.password) {

                    res.json(user);

                } else {

                    res.status(404).send('incorrect password');

                }

            } else {

                res.status(404).send('name not found');

            }

        } catch (e) {

            req.log.error('ERROR', e);

            res.status(500).send(e);

        }

    } else {

        req.log.error('database not available');

        res.status(500).send('database not available');

    }

});

// TODO - validate email address format

app.post('/register', async (req, res) => {

    req.log.info('register', req.body);

    if (req.body.name === undefined || req.body.password === undefined || req.body.email === undefined) {

        req.log.warn('insufficient data');

        res.status(400).send('insufficient data');

    } else if (mongoConnected) {

        try {

            const user = await usersCollection.findOne({ name: req.body.name });

            if (user) {

                req.log.warn('user already exists');

                res.status(400).send('name already exists');

            } else {

                const r = await usersCollection.insertOne({

                    name: req.body.name,

                    password: req.body.password,

                    email: req.body.email

                });

                req.log.info('inserted', r.result);

                res.send('OK');

            }

        } catch (e) {

            req.log.error('ERROR', e);

            res.status(500).send(e);

        }

    } else {

        req.log.error('database not available');

        res.status(500).send('database not available');

    }

});

app.post('/order/:id', async (req, res) => {

    req.log.info('order', req.body);

    if (mongoConnected) {

        try {

            const user = await usersCollection.findOne({

                name: req.params.id

            });

            if (user) {

                const history = await ordersCollection.findOne({

                    name: req.params.id

                });

                if (history) {

                    const list = history.history;

                    list.push(req.body);

                    await ordersCollection.updateOne(

                        { name: req.params.id },

                        { $set: { history: list } }

                    );

                    res.send('OK');

                } else {

                    await ordersCollection.insertOne({

                        name: req.params.id,

                        history: [req.body]

                    });

                    res.send('OK');

                }

            } else {

                res.status(404).send('name not found');

            }

        } catch (e) {

            req.log.error(e);

            res.status(500).send(e);

        }

    } else {

        req.log.error('database not available');

        res.status(500).send('database not available');

    }

});

app.get('/history/:id', async (req, res) => {

    if (mongoConnected) {

        try {

            const history = await ordersCollection.findOne({

                name: req.params.id

            });

            if (history) {

                res.json(history);

            } else {

                res.status(404).send('history not found');

            }

        } catch (e) {

            req.log.error(e);

            res.status(500).send(e);

        }

    } else {

        req.log.error('database not available');

        res.status(500).send('database not available');

    }

});

// connect to Redis

const redisClient = createClient({

    url: process.env.REDIS\_URL || 'redis://localhost:6379'

});

redisClient.on('error', (e) => {

    logger.error('Redis ERROR', e);

});

redisClient.on('connect', () => {

    logger.info('Redis connected');

});

redisClient.connect();

// set up Mongo

async function mongoConnect() {

    try {

        const mongoURL = process.env.MONGO\_URL || 'mongodb://localhost:27017/users';

        const client = await MongoClient.connect(mongoURL, { useNewUrlParser: true, useUnifiedTopology: true });

        db = client.db('users');

        usersCollection = db.collection('users');

        ordersCollection = db.collection('orders');

        mongoConnected = true;

        logger.info('MongoDB connected');

    } catch (error) {

        mongoConnected = false;

        logger.error('ERROR', error);

        setTimeout(mongoLoop, 2000);

    }

}

function mongoLoop() {

    mongoConnect().catch((e) => {

        logger.error('ERROR', e);

        setTimeout(mongoLoop, 2000);

    });

}

mongoLoop();

// fire it up!

const port = process.env.USER\_SERVER\_PORT || '8080';

app.listen(port, () => {

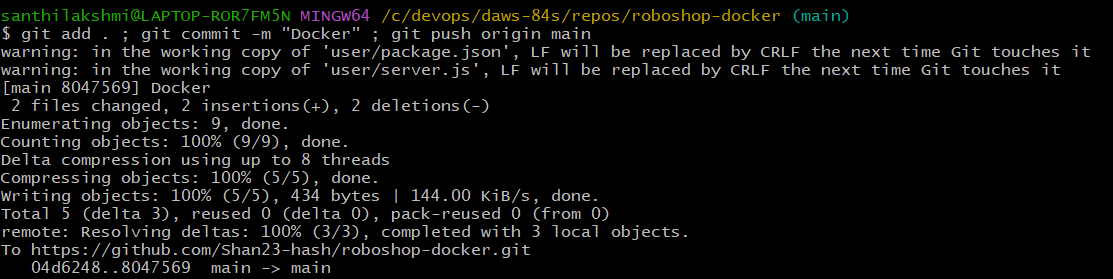
    logger.info('Started on port', port);

});

--> in roboshop we have to see what is the environment REDIS URL.

**/c/devops/daws-84s/repos/roboshop-docker**

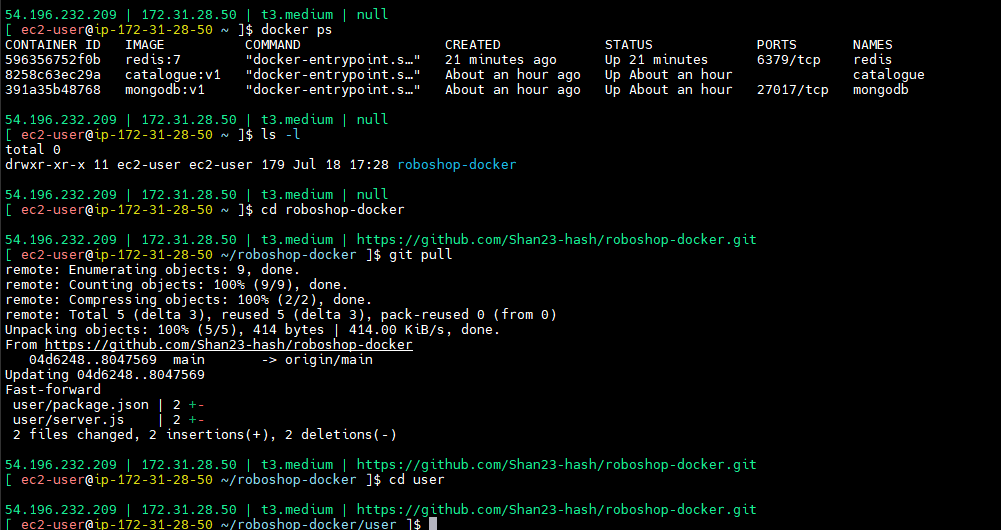
**git add . ; git commit -m "Docker" ; git push origin main**



**cd roboshop-docker**

**git pull**

**cd user**



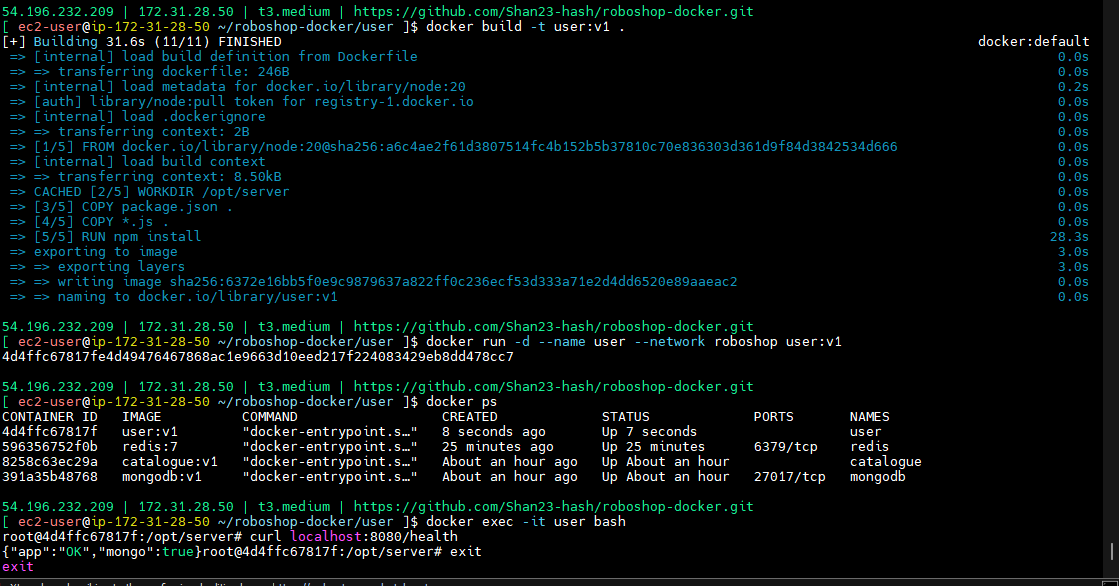
**docker build -t user:v1 .**

**docker run -d --name user --network roboshop user:v1**

**docker ps**

**docker exec -it user bash**

**curl localhost:8080/health --**  for checking whether user is connected mongo are not.



**CART**

<https://roboshop-artifacts.s3.amazonaws.com/cart-v3.zip>

**user/package.json**

{

  "name": "cart",

  "version": "1.0.0",

  "description": "cart REST API",

  "main": "server.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1"

  },

  "author": "SteveW",

  "license": "Apache-2.0",

  "dependencies": {

      "body-parser": "^1.18.1",

      "express": "^4.15.4",

      "redis": "^2.8.0",

      "request": "^2.88.2",

      "pino": "^5.10.8",

      "express-pino-logger": "^4.0.0",

      "pino-pretty": "^2.5.0",

      "@instana/collector": "^1.132.2",

      "prom-client": "^11.5.3"

**user/server.js**

const instana = require('@instana/collector');

// init tracing

// MUST be done before loading anything else!

instana({

    tracing: {

        enabled: true

    }

});

const redis = require('redis');

const request = require('request');

const bodyParser = require('body-parser');

const express = require('express');

const pino = require('pino');

const expPino = require('express-pino-logger');

// Prometheus

const promClient = require('prom-client');

const Registry = promClient.Registry;

const register = new Registry();

const counter = new promClient.Counter({

    name: 'items\_added',

    help: 'running count of items added to cart',

    registers: [register]

});

var redisConnected = false;

var redisHost = process.env.REDIS\_HOST || 'redis'

var catalogueHost = process.env.CATALOGUE\_HOST || 'catalogue'

var cataloguePort = process.env.CATALOGUE\_PORT || '8080'

const logger = pino({

    level: 'info',

    prettyPrint: false,

    useLevelLabels: true

});

const expLogger = expPino({

    logger: logger

});

const app = express();

app.use(expLogger);

app.use((req, res, next) => {

    res.set('Timing-Allow-Origin', '\*');

    res.set('Access-Control-Allow-Origin', '\*');

    next();

});

app.use((req, res, next) => {

    let dcs = [

        "asia-northeast2",

        "asia-south1",

        "europe-west3",

        "us-east1",

        "us-west1"

    ];

    let span = instana.currentSpan();

    span.annotate('custom.sdk.tags.datacenter', dcs[Math.floor(Math.random() \* dcs.length)]);

    next();

});

app.use(bodyParser.urlencoded({ extended: true }));

app.use(bodyParser.json());

app.get('/health', (req, res) => {

    var stat = {

        app: 'OK',

        redis: redisConnected

    };

    res.json(stat);

});

// Prometheus

app.get('/metrics', (req, res) => {

    res.header('Content-Type', 'text/plain');

    res.send(register.metrics());

});

// get cart with id

app.get('/cart/:id', (req, res) => {

    redisClient.get(req.params.id, (err, data) => {

        if(err) {

            req.log.error('ERROR', err);

            res.status(500).send(err);

        } else {

            if(data == null) {

                res.status(404).send('cart not found');

            } else {

                res.set('Content-Type', 'application/json');

                res.send(data);

            }

        }

    });

});

// delete cart with id

app.delete('/cart/:id', (req, res) => {

    redisClient.del(req.params.id, (err, data) => {

        if(err) {

            req.log.error('ERROR', err);

            res.status(500).send(err);

        } else {

            if(data == 1) {

                res.send('OK');

            } else {

                res.status(404).send('cart not found');

            }

        }

    });

});

// rename cart i.e. at login

app.get('/rename/:from/:to', (req, res) => {

    redisClient.get(req.params.from, (err, data) => {

        if(err) {

            req.log.error('ERROR', err);

            res.status(500).send(err);

        } else {

            if(data == null) {

                res.status(404).send('cart not found');

            } else {

                var cart = JSON.parse(data);

                saveCart(req.params.to, cart).then((data) => {

                    res.json(cart);

                }).catch((err) => {

                    req.log.error(err);

                    res.status(500).send(err);

                });

            }

        }

    });

});

// update/create cart

app.get('/add/:id/:sku/:qty', (req, res) => {

    // check quantity

    var qty = parseInt(req.params.qty);

    if(isNaN(qty)) {

        req.log.warn('quantity not a number');

        res.status(400).send('quantity must be a number');

        return;

    } else if(qty < 1) {

        req.log.warn('quantity less than one');

        res.status(400).send('quantity has to be greater than zero');

        return;

    }

    // look up product details

    getProduct(req.params.sku).then((product) => {

        req.log.info('got product', product);

        if(!product) {

            res.status(404).send('product not found');

            return;

        }

        // is the product in stock?

        if(product.instock == 0) {

            res.status(404).send('out of stock');

            return;

        }

        // does the cart already exist?

        redisClient.get(req.params.id, (err, data) => {

            if(err) {

                req.log.error('ERROR', err);

                res.status(500).send(err);

            } else {

                var cart;

                if(data == null) {

                    // create new cart

                    cart = {

                        total: 0,

                        tax: 0,

                        items: []

                    };

                } else {

                    cart = JSON.parse(data);

                }

                req.log.info('got cart', cart);

                // add sku to cart

                var item = {

                    qty: qty,

                    sku: req.params.sku,

                    name: product.name,

                    price: product.price,

                    subtotal: qty \* product.price

                };

                var list = mergeList(cart.items, item, qty);

                cart.items = list;

                cart.total = calcTotal(cart.items);

                // work out tax

                cart.tax = calcTax(cart.total);

                // save the new cart

                saveCart(req.params.id, cart).then((data) => {

                    counter.inc(qty);

                    res.json(cart);

                }).catch((err) => {

                    req.log.error(err);

                    res.status(500).send(err);

                });

            }

        });

    }).catch((err) => {

        req.log.error(err);

        res.status(500).send(err);

    });

});

// update quantity - remove item when qty == 0

app.get('/update/:id/:sku/:qty', (req, res) => {

    // check quantity

    var qty = parseInt(req.params.qty);

    if(isNaN(qty)) {

        req.log.warn('quanity not a number');

        res.status(400).send('quantity must be a number');

        return;

    } else if(qty < 0) {

        req.log.warn('quantity less than zero');

        res.status(400).send('negative quantity not allowed');

        return;

    }

    // get the cart

    redisClient.get(req.params.id, (err, data) => {

        if(err) {

            req.log.error('ERROR', err);

            res.status(500).send(err);

        } else {

            if(data == null) {

                res.status(404).send('cart not found');

            } else {

                var cart = JSON.parse(data);

                var idx;

                var len = cart.items.length;

                for(idx = 0; idx < len; idx++) {

                    if(cart.items[idx].sku == req.params.sku) {

                        break;

                    }

                }

                if(idx == len) {

                    // not in list

                    res.status(404).send('not in cart');

                } else {

                    if(qty == 0) {

                        cart.items.splice(idx, 1);

                    } else {

                        cart.items[idx].qty = qty;

                        cart.items[idx].subtotal = cart.items[idx].price \* qty;

                    }

                    cart.total = calcTotal(cart.items);

                    // work out tax

                    cart.tax = calcTax(cart.total);

                    saveCart(req.params.id, cart).then((data) => {

                        res.json(cart);

                    }).catch((err) => {

                        req.log.error(err);

                        res.status(500).send(err);

                    });

                }

            }

        }

    });

});

// add shipping

app.post('/shipping/:id', (req, res) => {

    var shipping = req.body;

    if(shipping.distance === undefined || shipping.cost === undefined || shipping.location == undefined) {

        req.log.warn('shipping data missing', shipping);

        res.status(400).send('shipping data missing');

    } else {

        // get the cart

        redisClient.get(req.params.id, (err, data) => {

            if(err) {

                req.log.error('ERROR', err);

                res.status(500).send(err);

            } else {

                if(data == null) {

                    req.log.info('no cart for', req.params.id);

                    res.status(404).send('cart not found');

                } else {

                    var cart = JSON.parse(data);

                    var item = {

                        qty: 1,

                        sku: 'SHIP',

                        name: 'shipping to ' + shipping.location,

                        price: shipping.cost,

                        subtotal: shipping.cost

                    };

                    // check shipping already in the cart

                    var idx;

                    var len = cart.items.length;

                    for(idx = 0; idx < len; idx++) {

                        if(cart.items[idx].sku == item.sku) {

                            break;

                        }

                    }

                    if(idx == len) {

                        // not already in cart

                        cart.items.push(item);

                    } else {

                        cart.items[idx] = item;

                    }

                    cart.total = calcTotal(cart.items);

                    // work out tax

                    cart.tax = calcTax(cart.total);

                    // save the updated cart

                    saveCart(req.params.id, cart).then((data) => {

                        res.json(cart);

                    }).catch((err) => {

                        req.log.error(err);

                        res.status(500).send(err);

                    });

                }

            }

        });

    }

});

function mergeList(list, product, qty) {

    var inlist = false;

    // loop through looking for sku

    var idx;

    var len = list.length;

    for(idx = 0; idx < len; idx++) {

        if(list[idx].sku == product.sku) {

            inlist = true;

            break;

        }

    }

    if(inlist) {

        list[idx].qty += qty;

        list[idx].subtotal = list[idx].price \* list[idx].qty;

    } else {

        list.push(product);

    }

    return list;

}

function calcTotal(list) {

    var total = 0;

    for(var idx = 0, len = list.length; idx < len; idx++) {

        total += list[idx].subtotal;

    }

    return total;

}

function calcTax(total) {

    // tax @ 20%

    return (total - (total / 1.2));

}

function getProduct(sku) {

    return new Promise((resolve, reject) => {

        request('http://' + catalogueHost + ':' + cataloguePort +'/product/' + sku, (err, res, body) => {

            if(err) {

                reject(err);

            } else if(res.statusCode != 200) {

                resolve(null);

            } else {

                // return object - body is a string

                // TODO - catch parse error

                resolve(JSON.parse(body));

            }

        });

    });

}

function saveCart(id, cart) {

    logger.info('saving cart', cart);

    return new Promise((resolve, reject) => {

        redisClient.setex(id, 3600, JSON.stringify(cart), (err, data) => {

            if(err) {

                reject(err);

            } else {

                resolve(data);

            }

        });

    });

}

// connect to Redis

var redisClient = redis.createClient({

    host: redisHost

});

redisClient.on('error', (e) => {

    logger.error('Redis ERROR', e);

});

redisClient.on('ready', (r) => {

    logger.info('Redis READY', r);

    redisConnected = true;

});

// fire it up!

const port = process.env.CART\_SERVER\_PORT || '8080';

app.listen(port, () => {

    logger.info('Started on port', port);

});

**<https://roboshop-artifacts.s3.amazonaws.com/cart-v3.zip>**

**Downloaded it and extrac.**

**cart/Dockerfile**

FROM node:20

WORKDIR /opt/server

COPY package.json .

COPY \*.js .

RUN npm install

ENV REDIS\_HOST="redis" \

CATALOGUE\_HOST="catalogue" \

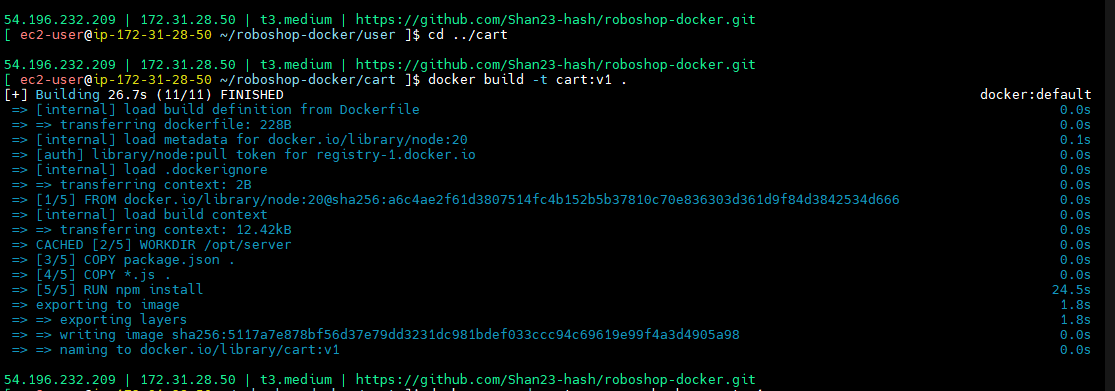
CATALOGUE\_PORT="8080"

CMD ["node","server.js"]

--> redis,catalogue host and catalogue port

**cd ../cart**

**docker build -t cart:v1 .**

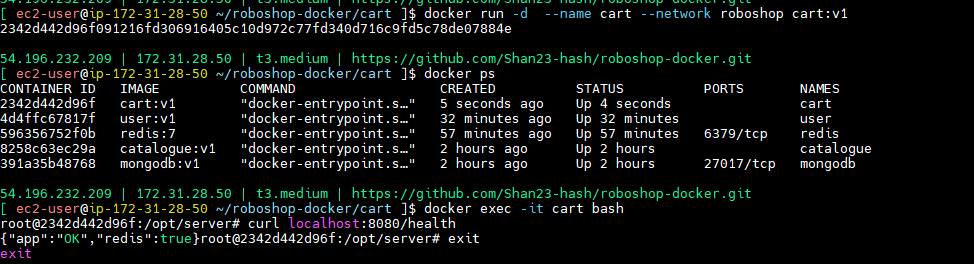


**docker run -d --name cart --network roboshop cart:v1**

**docker ps**

**docker exec -it cart bash**

**curl localhost:8080/health**

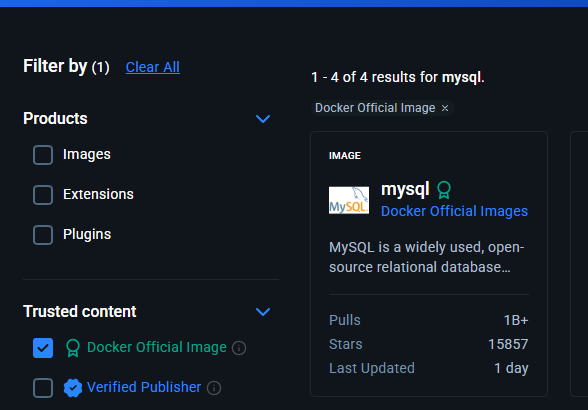


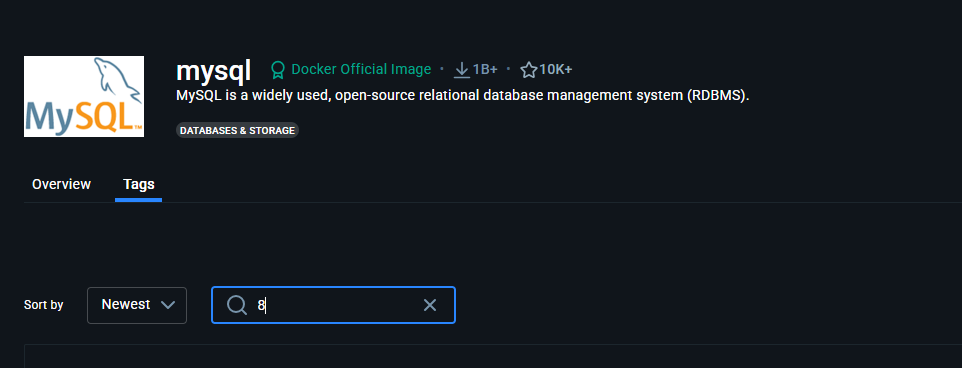
**MYSQL**

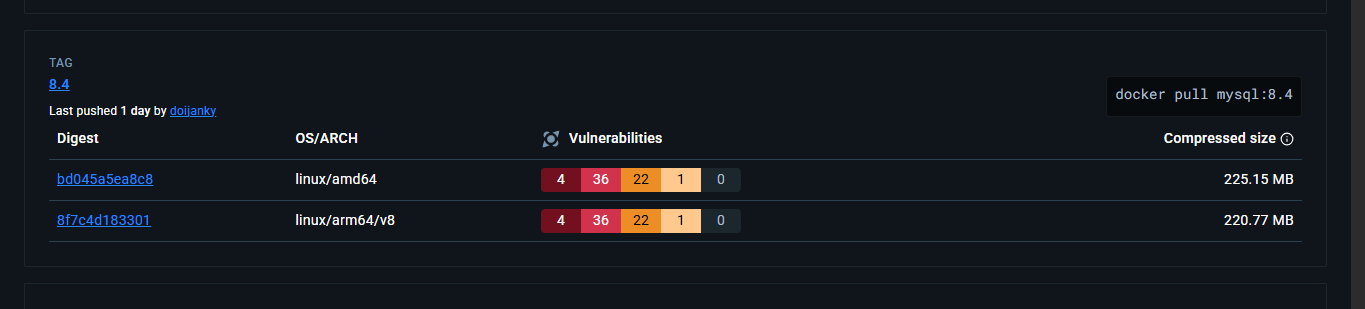
--> Mysql 8 - in mysql also we have to load data.

Go to docker hub image name almost mysql

Search mysql and select docker offcial image







**mysql/Dockefile**

--> this is mysql . have to load dbtm. Scripts we given means they will load.

They wont give root access to application team, so dptm.

--> where we have scripts for loding - there in shipping.

<https://roboshop-artifacts.s3.amazonaws.com/shipping-v3.zip>

**mysql/db/app-user.sql**

CREATE USER IF NOT EXISTS 'shipping'@'%' IDENTIFIED WITH mysql\_native\_password BY 'RoboShop@1';

GRANT ALL ON cities.\* TO 'shipping'@'%';

FLUSH PRIVILEGES;

**mysql/db/master-data.sql**

-- MySQL dump 10.13  Distrib 5.7.20, for Linux (x86\_64)

--

-- Host: localhost    Database: cities

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

-- Server version 5.7.20

CREATE DATABASE IF NOT EXISTS cities;

use cities;

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;

/\*!40101 SET NAMES utf8 \*/;

/\*!40103 SET @OLD\_TIME\_ZONE=@@TIME\_ZONE \*/;

/\*!40103 SET TIME\_ZONE='+00:00' \*/;

/\*!40014 SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0 \*/;

/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;

/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;

/\*!40111 SET @OLD\_SQL\_NOTES=@@SQL\_NOTES, SQL\_NOTES=0 \*/;

--

-- Table structure for table `cities`

--

DROP TABLE IF EXISTS `cities`;

/\*!40101 SET @saved\_cs\_client     = @@character\_set\_client \*/;

/\*!40101 SET character\_set\_client = utf8 \*/;

CREATE TABLE `cities` (

  `uuid` int(11) NOT NULL AUTO\_INCREMENT,

  `country\_code` varchar(2) DEFAULT NULL,

  `city` varchar(100) DEFAULT NULL,

  `name` varchar(100) DEFAULT NULL,

  `region` varchar(100) DEFAULT NULL,

  `latitude` decimal(10,7) DEFAULT NULL,

  `longitude` decimal(10,7) DEFAULT NULL,

  PRIMARY KEY (`uuid`),

  KEY `region\_idx` (`region`),

  KEY `c\_code\_idx` (`country\_code`),

  FULLTEXT KEY `city\_idx` (`city`)

) ENGINE=InnoDB AUTO\_INCREMENT=6223666 DEFAULT CHARSET=latin1;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `codes`

--

DROP TABLE IF EXISTS `codes`;

/\*!40101 SET @saved\_cs\_client     = @@character\_set\_client \*/;

/\*!40101 SET character\_set\_client = utf8 \*/;

CREATE TABLE `codes` (

  `uuid` int(11) NOT NULL AUTO\_INCREMENT,

  `code` varchar(2) DEFAULT NULL,

  `name` varchar(100) DEFAULT NULL,

  PRIMARY KEY (`uuid`),

  UNIQUE KEY `code\_idx` (`code`)

) ENGINE=InnoDB AUTO\_INCREMENT=51 DEFAULT CHARSET=latin1;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Dumping data for table `codes`

--

LOCK TABLES `codes` WRITE;

/\*!40000 ALTER TABLE `codes` DISABLE KEYS \*/;

INSERT INTO `codes` VALUES (26,'au','Australia'),(27,'at','Austria'),(28,'br','Brasil'),(29,'bg','Bulgaria'),(30,'ca','Canada'),(31,'cz','Czech Republic'),(32,'dk','Denmark'),(33,'fi','Finland'),(34,'fr','France'),(35,'de','Germany'),(36,'hu','Hungary'),(37,'in','India'),(38,'it','Italy'),(39,'jp','Japan'),(40,'nl','Netherlands'),(41,'no','Norway'),(42,'pt','Portugal'),(43,'ro','Romania'),(44,'ru','Russia'),(45,'es','Spain'),(46,'se','Sweden'),(47,'ch','Swiss'),(48,'tr','Turkey'),(49,'us','USA'),(50,'gb','Great Britain');

/\*!40000 ALTER TABLE `codes` ENABLE KEYS \*/;

UNLOCK TABLES;

/\*!40103 SET TIME\_ZONE=@OLD\_TIME\_ZONE \*/;

/\*!40101 SET SQL\_MODE=@OLD\_SQL\_MODE \*/;

/\*!40014 SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS \*/;

/\*!40014 SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS \*/;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;

/\*!40111 SET SQL\_NOTES=@OLD\_SQL\_NOTES \*/;

-- Dump completed on 2018-01-29 12:33:04

NOTE : in middle commands are there but I am not added because it’s big thing.

**mysql/db/schema.sql**

CREATE DATABASE IF NOT EXISTS cities;

use cities;

DROP TABLE IF EXISTS `cities`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!40101 SET character\_set\_client = utf8 \*/;

CREATE TABLE `cities` (

`uuid` int(11) NOT NULL AUTO\_INCREMENT,

`country\_code` varchar(2) DEFAULT NULL,

`city` varchar(100) DEFAULT NULL,

`name` varchar(100) DEFAULT NULL,

`region` varchar(100) DEFAULT NULL,

`latitude` decimal(10,7) DEFAULT NULL,

`longitude` decimal(10,7) DEFAULT NULL,

PRIMARY KEY (`uuid`),

KEY `region\_idx` (`region`),

KEY `c\_code\_idx` (`country\_code`),

FULLTEXT KEY `city\_idx` (`city`)

) ENGINE=InnoDB AUTO\_INCREMENT=6223666 DEFAULT CHARSET=latin1;

--> this will create user for shiiping

--> aftercreating user will set password.

--> use cities, schema will create database if does not exist.

--> schema means table creation,insertion and deletion

--> in mysql we selected root password.

--> in dockerhub image also set password - option is there

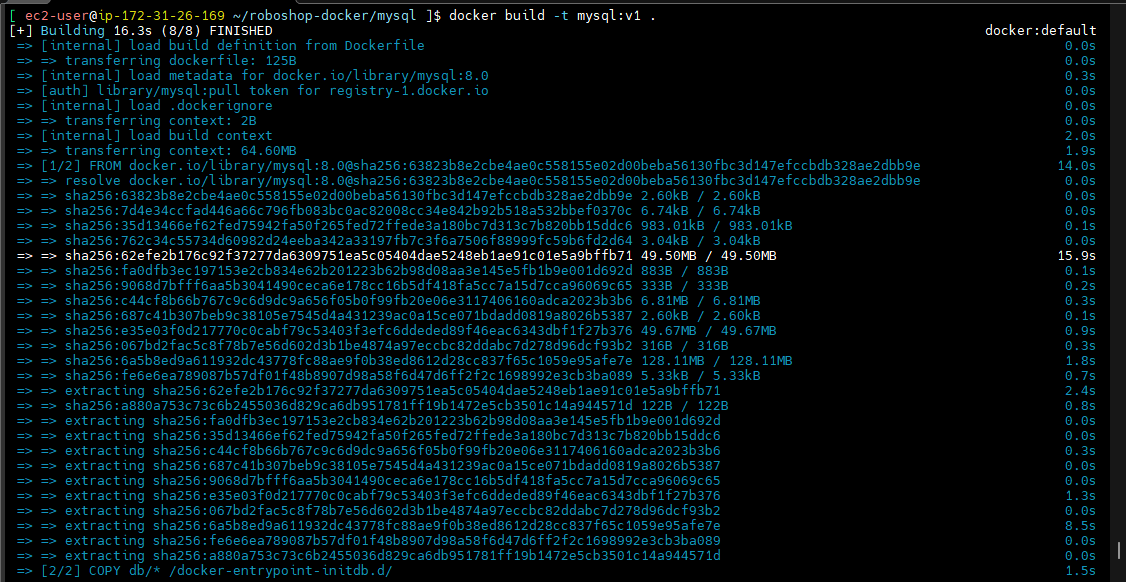
**mysql/Docketfile**

FROM mysql:8.0

ENV MYSQL\_ROOT\_PASSWORD=RoboShop@1

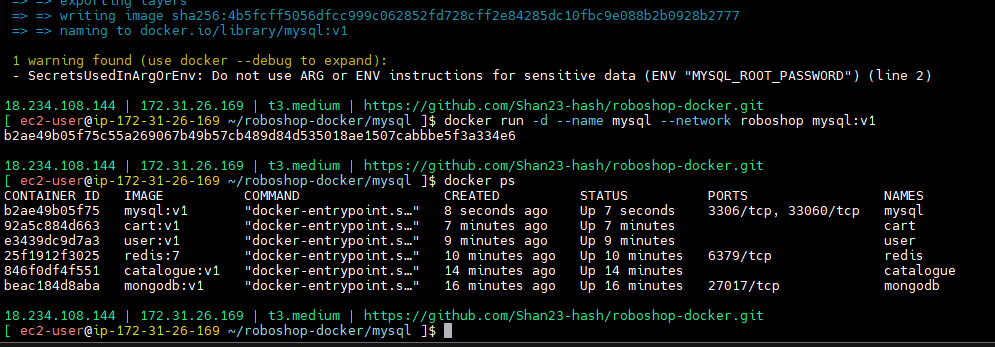
COPY db/\* /docker-entrypoint-initdb.d/

**docker build -t mysql:v1 .**



**docker run -d --name mysql --network roboshop mysql:v1**

**docker ps**



**SHIPPING**

Dowmloas shiiping code -- src folder and pom.xml add in shpping

**shipping/Dockerfile**

FROM maven

WORKDIR /opt/server

COPY pom.xml .

COPY src /opt/server/src

RUN mvn clean package

RUN mv /opt/server/target/shipping-1.0.jar /opt/server/shipping.jar

ENV CART\_ENDPOINT="cart:8080" \

DB\_HOST="mysql"

CMD ["java","-jar","shipping.jar"]

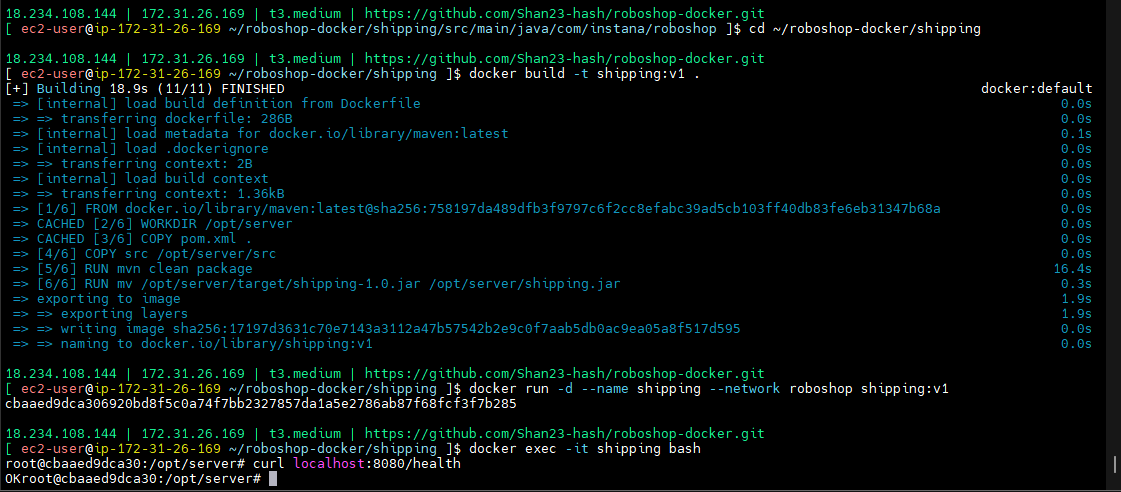
--> while doing shiiping build got many errors please refer vedio.

**docker build -t shipping:v1 .**

**rm -rf src/main/java/com/instana/roboshop/shippingg -**-if you get error use this

**docker build -t shipping:v1 .**

**docker run -d --name shipping --network roboshop shipping:v1**

**docker exec -it shipping bash** 

**docker ps**

**docker images**

