Task 1:

Create a API Locally :

Created the API in node JS and have hosted that application on docker.
Created a Server.js File in which the complete API code with mentioned endpoints are written.

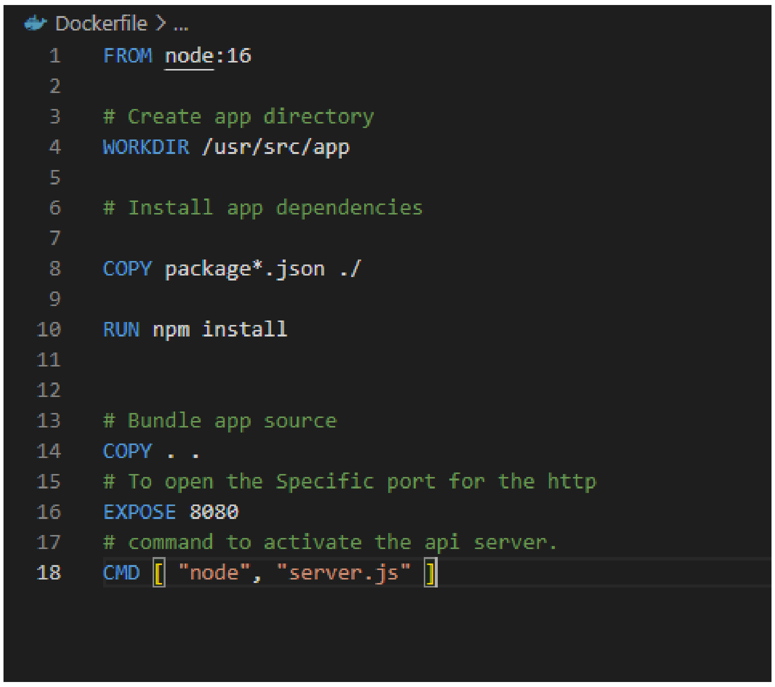
Dockerize the app with minimal foot print, preferably using multi stage building
Create a script which will allow

1. **install dependencies and build the api locally.**

1. **create Docker image - Docker tags either can be given as an argument or determine based on git hash**

1. **Create a README.MD file to describe how to run the app in local environment.**

Solution :
Created a Docker file :



Dockerfile > ...

```
FROM node:16

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies

COPY package*.json ./

RUN npm install


# Bundle app source
COPY . .
# To open the Specific port for the http
EXPOSE 8080
# command to activate the api server.
CMD [ "node", "server.js" ]
```

Command run to build the docker image in local system:
`docker build . -t hardik/node-docker-app`

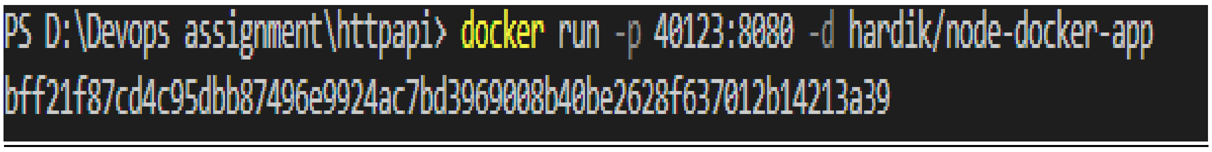
A docker image is created successfully .

Images list can be viewed using the command :

`docker images`

Command to run the docker using our own created image with the Ports (40123 of docker and 8080 of image used)

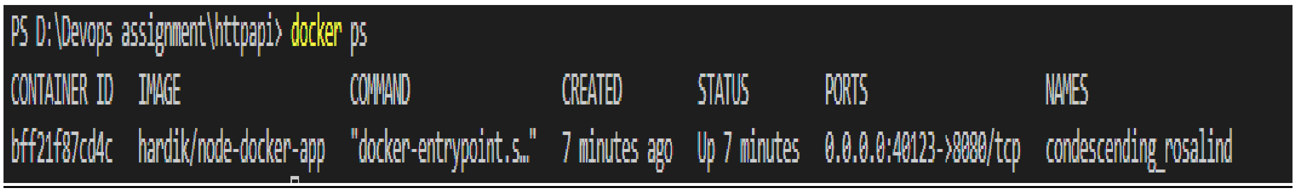
```
docker run -p 40123:8080 -d hardik/node-docker-app
```



```
PS D:\Devops assignment\httpapi> docker run -p 40123:8080 -d hardik/node-docker-app
bff21f87cd4c95dbb87496e9924ac7bd3969008b40be2628f637012b14213a39
```

To view the running docker/containers :

```
docker ps
```



```
PS D:\Devops assignment\httpapi> docker ps
CONTAINER ID   IMAGE                   COMMAND                  CREATED         STATUS         PORTS                     NAMES
bff21f87cd4c   hardik/node-docker-app  "docker-entrypoint.s…"   7 minutes ago   Up 7 minutes   0.0.0.0:40123->8080/tcp   condescending_rosalind
```

Our application is now active on port: 40123

For the complete automation we can also shell script as below:

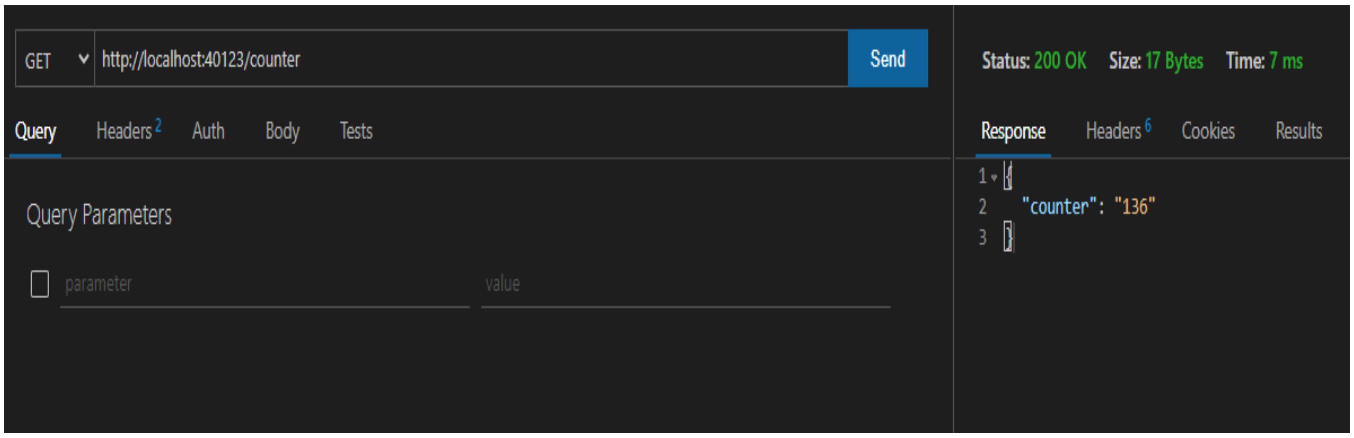
```
#!/bin/bash
sudo apt-get install jq -y #this is the JSON processor

docker build -t nodeTestApp --build-arg GIT_COMMIT=$(git log -1 --format=%h)
docker inspect nodeTestApp | jq '.[].ContainerConfig.Labels'
docker tag nodeTestApp hardik/nodeapp:dockertask
docker push hardik/nodeapp:dockertask
```

We need to add these in DockerFile to show the runtime values.

```
ARG GIT_COMMIT=unspecified
LABEL git_commit=$GIT_COMMIT
```

We tested the API working using the POSTMAN :

1. /counter: ouput format :: {"counter":"123"}
GET will increment the counter by 1 and return the same value.

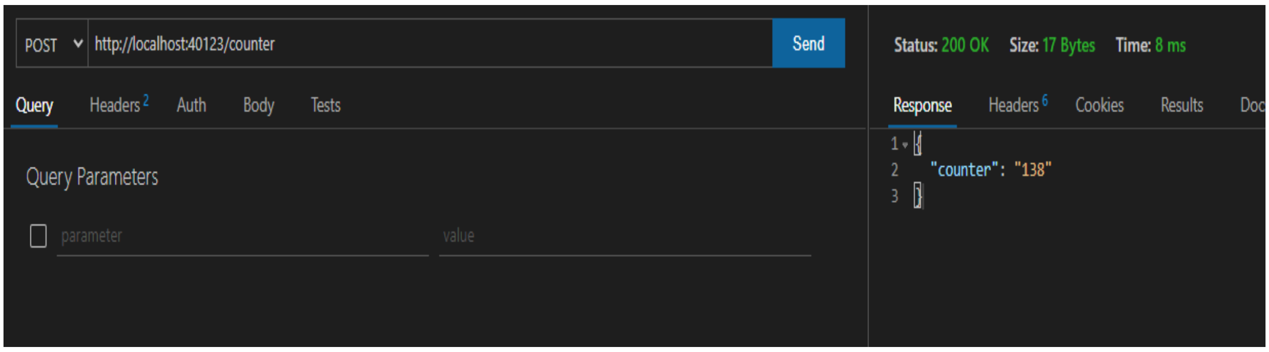


http://localhost:40123/counter

Status: 200 OK   Size: 17 Bytes   Time: 7 ms

```
{
  "counter": "136"
}
```

2. POST will increment the counter by 2 and return the same value.

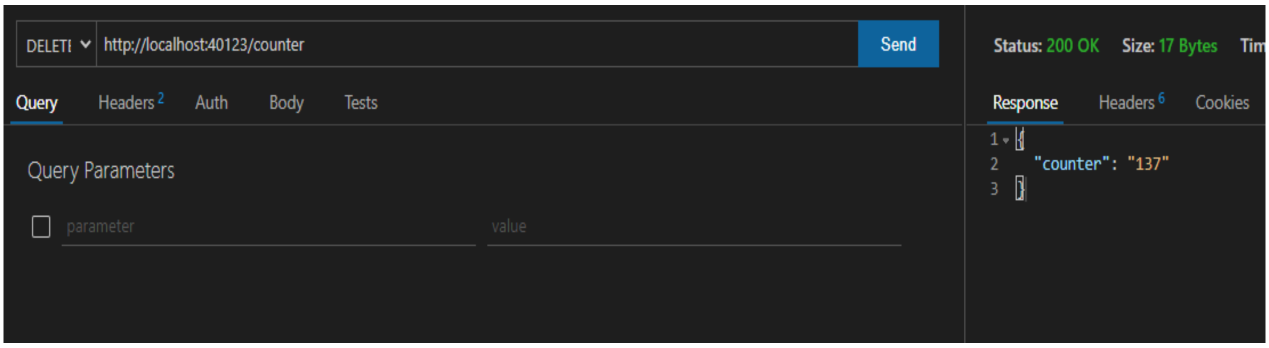


http://localhost:40123/counter

Status: 200 OK   Size: 17 Bytes   Time: 8 ms

```
{
  "counter": "138"
}
```

3. DELETE will decrement the counter by 1 and return the same value.

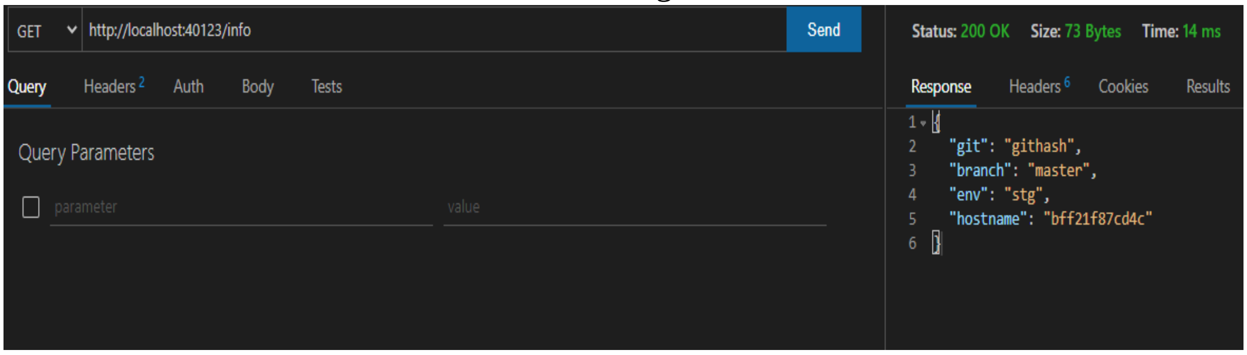


http://localhost:40123/counter

Status: 200 OK   Size: 17 Bytes   Tim

```
{
  "counter": "137"
}
```

4. **/info**: GET will return a JSON with following info :



http://localhost:40123/info

Status: 200 OK   Size: 73 Bytes   Time: 14 ms

```
{
  "git": "githash",
  "branch": "master",
  "env": "stg",
  "hostname": "bff21f87cd4c"
}
```

Design and document CI/CD work flow and code promotion, ex: - dev/qa/stress test/staging/prod etc.

Solution :

1. The user creates a branch, make changes, commits his changes to his local branch.

2. For each commit bitbucket webhook triggers a Jenkins build to do a sanity check of the code and publishes success/failure result to bitbucket pull request page.

3. Approvers will review the code and finally merges to the main development branch.

4. Webhook will trigger the Jenkins build job. Jenkins pipeline steps:

a. Pull the latest source code in the build node in the Jenkins workspace.

b. Update the version if any.

c. Compile the source code.

d. Create git tag.

e. Runs source code analysis/code coverage (optional as it takes more time, we can schedule the frequency)

f. Runs unit test cases, publish the report to Jenkins dashboard. If passed then go ky to f, otherwise send an email to specific team about the failure.

g. bundles the binaries/ creates packages/image in order to deploy.

h. artifacts the logs and reports.

i. Deploy the final product in Docker container or deploy to an Ansible host using Ansible.

j. Triggers the stress test, regression test jobs. Collects the reports.

h. Notify user about the test results. If result percentage is above threshold then stage the change or tag it with gold build.*

* This is not CD pipeline as after collecting the result we need manual verification for user acceptance test and release it to production.]

Helm Charts:
I am not aware of Helm charts but I have a tried to find the Solution of the same.
Kindly check this HELM file.

TASK 2:

Automation/Config management: (pick your favorite automation tool like chef, ansible, salt, puppet.)

Write automation scripts to install list of packages: haproxy

Ensure haproxy starts automatically on reboot

Update haproxy config from given repo on demand from given git repo.

Firstly we Require two Servers/VMs/Containers
1: For the Controller Node : Where HA-PROXY server will be configured.
2. For the Target Node : Where the Web Server would be configured for the Load balancing part.

We will be using the ANSIBLE tool for the automation.

Step1: Install the ansible as well as HAproxy on the controller machine.
Step 2: Set up the Hosts List

The Control Node as well as Target Node data needs to be written in hosts file.
Step 3: Ping all the Hosts to check the connection

```
ansible all --list-hosts
```

Step 4: Copy the haproxy.cfg i.e configuration file for the haproxy into the current working directory with name as “haproxy.cfg.j2”.

we will add the loop to check the target nodes

Step 4: We will write the ansible playbook to run the automation script. (myhaproxy.yml)

```
- hosts: controlnode
  tasks:
    - name: "install haproxy"
      package:
        name: haproxy
        state: present
    - name: " configure haproxy.cfg file"
      template:
        src: "/root/ansible_ws/haproxy.cfg.j2"
        dest: "/etc/haproxy/haproxy.cfg"
    - name: " haproxy service start"
      service:
        name: haproxy
        state: restarted

- hosts: targetnode
  tasks:
    - name: "Installing httpd on the target node"
      package:
        name: httpd
        state: present
    - name: "copying data to target node"
      copy:
        dest: /var/www/html/index.html
        content: "Hello World. This Site coming from the target Node"
    - name: "Httpd service start"
      service:
        name: httpd
        state: started
        enabled: yes
- git:
      repo: " https://github.com/hardik23singhal/DevopsAssignment/blob/main/httpapi/Task2/haproxy.cfg.j2"
      dest: "/etc/haproxy/haproxy.cfg"
```

Step 5: Now we need to run the script using the below command :

```
ansible-playbook myhaproxy.yml
```

Step 6:
On control node haproxy configuration file will be like, the file dynamically configured for the ip of the target node of the inventory file.

On the target node httpd server configured successfully.

Step 7 :
We can use the ip of the controlnode to access the webpage running on the target node with help of the load balancer and reverse proxy offered by the haproxy

http://localhost:8080/

We would be able to access the site from the target node as a webpage below :

“Hello World. This Site coming from the target Node” .