

Course- BTech/BCA/B.Sc: (B.Tech)

Course Code- ESE460L

Year- III (VI Sem)

Date- 10-02-2022

Type- Core/Elective (Elective)

Course Name: DevOps Engineering Practices

Semester- Even/Odd (Even)

Batch- B1-B14

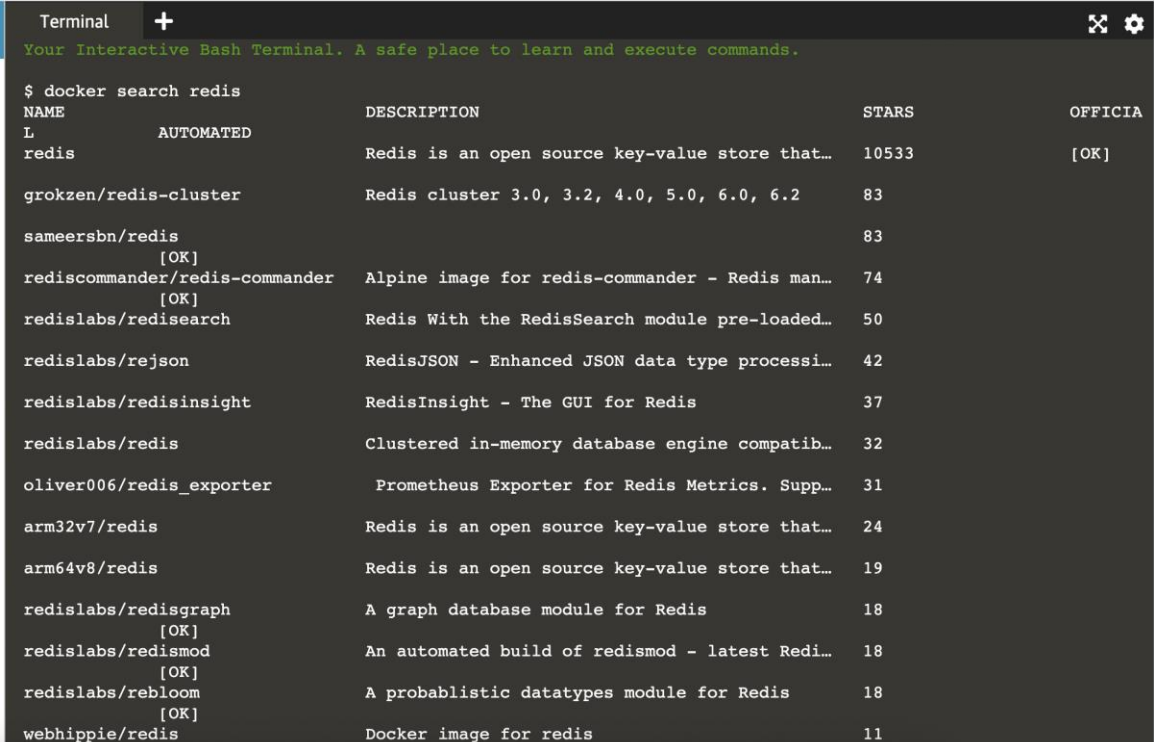
A- Type- Lab Assignment (Week 4, Lab 4)

Objectives:

- 1) Creating and running an instance in detached mode
- 2) Deploy static pages using Docker (Theoretical Background and Practical) (45)
- 3) Dockerizing node.js application (Theoretical Background and Practical) (45)

Create a running container

- a) Docker Images start from a base image. The base image should include the platform dependencies required by your application For example, to find an image for *Redis*, you would use `docker search redis`.



```
Terminal +
Your Interactive Bash Terminal. A safe place to learn and execute commands.

$ docker search redis
NAME                                DESCRIPTION                                STARS                                OFFICIAL
L                                  AUTOMATED
redis                               Redis is an open source key-value store that... 10533                                [OK]
grokzen/redis-cluster               Redis cluster 3.0, 3.2, 4.0, 5.0, 6.0, 6.2    83
sameersbn/redis                     83
[OK]
rediscommander/redis-commander      Alpine image for redis-commander - Redis man... 74
[OK]
redislabs/redisearch                Redis With the RedisSearch module pre-loaded... 50
redislabs/rejson                    RedisJSON - Enhanced JSON data type processi... 42
redislabs/redisinsight              RedisInsight - The GUI for Redis              37
redislabs/redis                     Clustered in-memory database engine compatib... 32
oliver006/redis_exporter             Prometheus Exporter for Redis Metrics. Supp... 31
arm32v7/redis                       Redis is an open source key-value store that... 24
arm64v8/redis                       Redis is an open source key-value store that... 19
redislabs/redisgraph                A graph database module for Redis             18
[OK]
redislabs/redismod                  An automated build of redismod - latest Redi... 18
[OK]
redislabs/rebloom                   A probabilistic datatypes module for Redis    18
[OK]
webhippie/redis                     Docker image for redis                         11
```

- b) Using the search command, we can identify that the *Redis* Docker Image is called *redis* and wants to run the *latest* release. Because *Redis* is a database, we want to run it as a background service while we do other operations, to complete this step, launch a container in the background running an instance of Redis based on the official image. The Docker CLI has a command called *run* which will start a container based on a Docker Image. The structure is *docker run <options> <image-name>*. By default, Docker will run a command in the foreground. To run in the background, the option *-d* needs to be specified.

```
docker run -d redis
```

```
$ docker run -d redis
050521788e259c3e976a45502c2119632c3d20557b2a36da4d96ab3de5504b9f
$ docker run -d redis
b8e07c39f0199c9790be9fb4c4f1eb375562ef908a75640b6573c8d8f08e2447
$
```

- c) The launched container is running in the background, the `docker ps` command lists all running containers, the image used to start the container and uptime

```
$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS
b8e07c39f019   redis     "docker-entrypoint.s..." About a minute ago Up About a minute 6379/tcp
050521788e25   redis     "docker-entrypoint.s..." 2 minutes ago  Up 2 minutes  6379/tcp
zealous_borg
```

- d) The command `docker inspect <friendly-name|container-id>` provides more details about a running container, such as IP address. The command `docker logs <friendly-name|container-id>` will display messages the container has written to standard error or standard out.

Deploy static pages using Docker (Theoretical Background and Practical) (45)

Steps:

a) Create Dockerfile

Docker Images start from a base image. The base image should include the platform dependencies required by your application

Our base image is the Alpine version of Nginx. This provides the configured web server on the Linux Alpine distribution

```
: FROM nginx:alpine
COPY . /usr/share/nginx/html
#The first line defines our base image.
#The second line copies the content of the current directory into a particular location inside the container.
```

b) Build Dockerimage

The Dockerfile is used by the Docker CLI *build* command. The *build* command executes each instruction within the Dockerfile.

The build command takes in some different parameters. The format is *docker build -t <build-directory>*. The *-t* parameter allows you to specify a name for the image and a tag.

```
$ docker build -t webserver-image:v1 .
Sending build context to Docker daemon 3.072kB
Step 1/2 : FROM nginx:alpine
---> bef258acf10d
Step 2/2 : COPY . /usr/share/nginx/html
---> afac3cca4bfb
Successfully built afac3cca4bfb
Successfully tagged webserver-image:v1
$
```

Docker images can be listed by:

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
webserver-image	v1	afac3cca4bfb	33 seconds ago	23.4MB
nginx	alpine	bef258acf10d	2 weeks ago	23.4MB
redis	latest	4760dc956b2d	3 years ago	107MB
ubuntu	latest	f975c5035748	3 years ago	112MB
alpine	latest	3fd9065eaf02	4 years ago	4.14MB

```
$
```

c) Run

The built Image can be launched in a consistent way to other Docker Images.

Eg:

-p <host-port>:<container-port>.

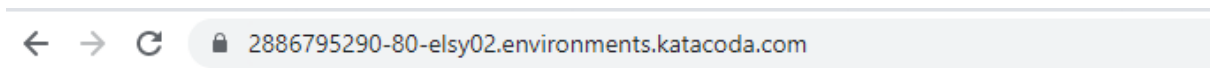
```
docker run -d -p 80:80 webserver-image:v1
```

```
$ docker run -d -p 80:80 webserver-image:v1
a87a9af6f25eeccf2711f2ec0f1af1a361bbde9623ac8ea3994994e35947d2fd
docker: Error response from daemon: driver failed programming external connectivity on endpoint a
gitated_stallman (670786b3353e8c105c7310c689685b965654648901b09d025eac3e3e4c928b84): Bind for 0.0
.0.0:80 failed: port is already allocated.
$ curl docker
<h1>Hello World</h1>
$
```

Html code:

Dockerfile	index.html	x
1	<h1>Hello World</h1>	
2		

Browser:



Hello World

Dockerizing node.js applications:

This scenario explores how to deploy a Node.js application within a container.

The environment is configured with access to a personal Docker instance, and the code for a default Expressjs application is in the working directory. To view the code use `ls` and `cat <filename>` or use the editor.

The machine name Docker is running on is called *docker*. If you want to access any of the services then use *docker* instead of `localhost` or `0.0.0.0`.

Steps:

a) Base Image

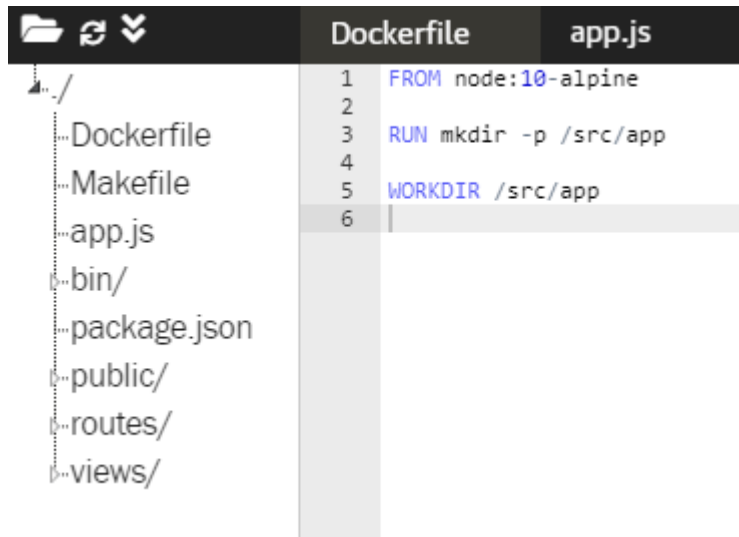
The image for Node 10.0 is `node:10-alpine`. This is an Alpine-based build which is smaller and more streamlined than the official image.

Alongside the base image, we also need to create the base directories of where the application runs from. Using the `RUN <command>` we can execute commands as if they're running from a command shell, by using `mkdir` we can create the directories where the application will

execute from. In this case, an ideal directory would be `/src/app` as the environment user has read/write access to this directory.

b) Define Base Environment

Set the *FROM* `<image>:<tag>`, *RUN* `<command>` and *WORKDIR* `<directory>` on separate lines to configure the base environment for deploying your application.

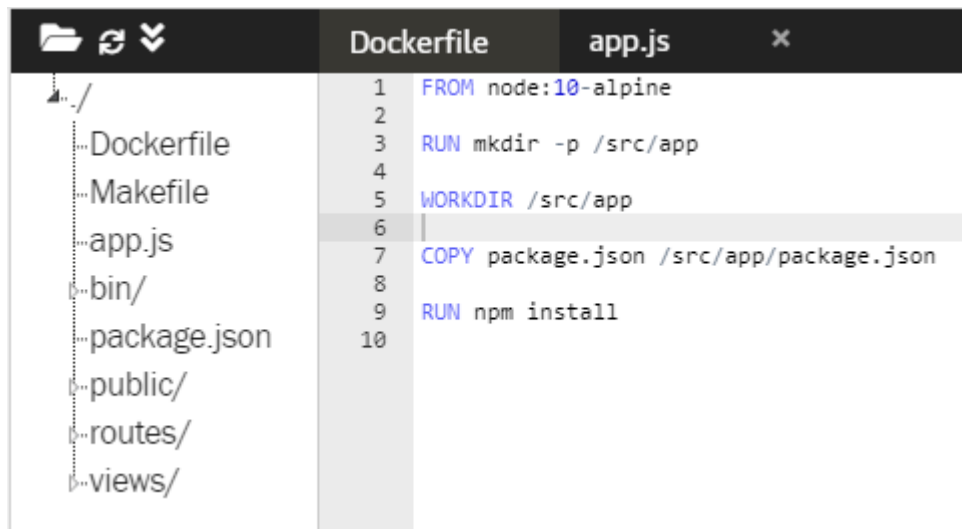


c) NPM Install

The next stage is to install the dependencies required to run the application. For Node.js this means running NPM install.

With NPM we only want to re-run `npm install` if something within our `package.json` file has changed. If nothing has changed then we can use the cache version to speed up deployment. By using *COPY* `package.json <dest>` we can cause the *RUN* `npm install` command to be invalidated if the `package.json` file has changed. If the file has not changed, then the cache will not be invalidated, and the cached results of the `npm install` command will be used.

Add Dockerfile lines



d) Configuring Application

After we've installed our dependencies, we want to copy over the rest of our application's source code. Splitting the installation of the dependencies and copying out source code enables us to use the cache when required.

e) Deploy Application

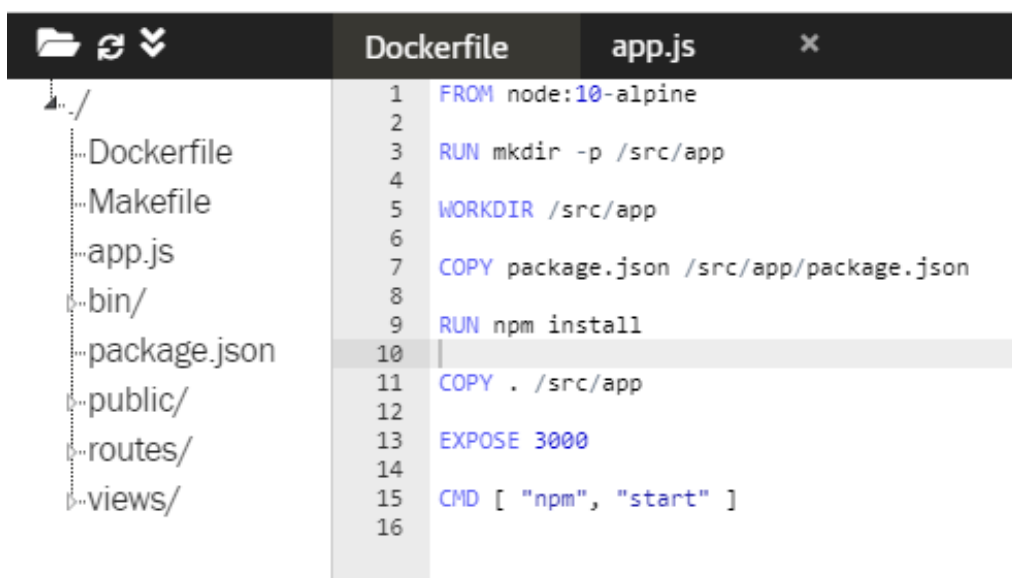
Create the desired steps in the Dockerfile to finish the deployment of the application.

We can copy the entire directory where our Dockerfile is using *COPY*. *<dest dir>*.

Once the source code has been copied, the ports the application requires to be accessed is defined using *EXPOSE* *<port>*.

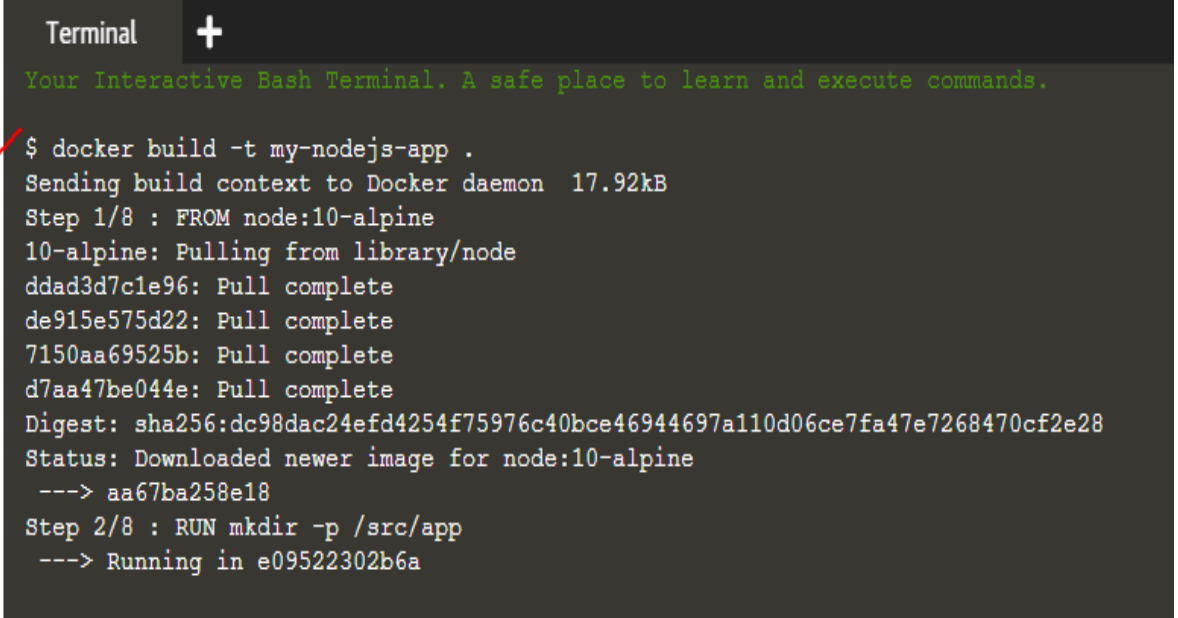
Finally, the application needs to be started.

Lines number 11-15 show the configuration of file






f) Building and Launching Container

To launch your application inside the container you first need to build an image.

A terminal window with a dark background. The title bar says "Terminal" with a plus icon. Below the title bar, it says "Your Interactive Bash Terminal. A safe place to learn and execute commands." in green. The terminal shows the command "\$ docker build -t my-nodejs-app ." followed by the output of the Docker build process. A red checkmark is next to the command line.

```
$ docker build -t my-nodejs-app .  
Sending build context to Docker daemon 17.92kB  
Step 1/8 : FROM node:10-alpine  
10-alpine: Pulling from library/node  
ddad3d7c1e96: Pull complete  
de915e575d22: Pull complete  
7150aa69525b: Pull complete  
d7aa47be044e: Pull complete  
Digest: sha256:dc98dac24efd4254f75976c40bce46944697a110d06ce7fa47e7268470cf2e28  
Status: Downloaded newer image for node:10-alpine  
----> aa67ba258e18  
Step 2/8 : RUN mkdir -p /src/app  
----> Running in e09522302b6a
```



./

Makefile

app.js

bin/

package.json

public/

routes/

views/

Dockerfile

app.js

x

```
1 var express = require('express');
2 var path = require('path');
3 var favicon = require('serve-favicon');
4 var logger = require('morgan');
5 var cookieParser = require('cookie-parser');
6 var bodyParser = require('body-parser');
7
8 var routes = require('./routes/index');
9 var users = require('./routes/users');
10
11 var app = express();
12
13 // view engine setup
14 app.set('views', path.join(__dirname, 'views'));
15 app.set('view engine', 'jade');
16
17 // uncomment after placing your favicon in /public
18 //app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
19 app.use(logger('dev'));
20 app.use(bodyParser.json());
21 app.use(bodyParser.urlencoded({ extended: false }));
22 app.use(cookieParser());
23 app.use(express.static(path.join(__dirname, 'public')));
24
25 app.use('/', routes);
26 app.use('/users', users);
27
28 // catch 404 and forward to error handler
29 app.use(function(req, res, next) {
30   var err = new Error('Not Found');
```

Terminal

+

Your Interactive Bash Terminal. A safe place to learn and execute commands.

g) The command to launch the built image is

```
✓ docker run -d --name my-running-app -p 3000:3000 my-nodejs-app
npm notice created a lockfile as package-lock.json. You should commit this file.
added 78 packages from 74 contributors and audited 79 packages in 17.211s
found 22 vulnerabilities (4 low, 5 moderate, 11 high, 2 critical)
  run `npm audit fix` to fix them, or `npm audit` for details
Removing intermediate container 18dd37f99e9b
---> 6ad9b5bf996a
Step 6/8 : COPY . /src/app
---> 9dab663200e1
Step 7/8 : EXPOSE 3000
---> Running in e64871f594ef
Removing intermediate container e64871f594ef
---> 3aab4948717e
Step 8/8 : CMD [ "npm", "start" ]
---> Running in 2cd13d71af7a
Removing intermediate container 2cd13d71af7a
---> 20bfa98e8786
Successfully built 20bfa98e8786
Successfully tagged my-nodejs-app:latest
$ docker run -d --name my-running-app -p 3000:3000 my-nodejs-app
b15e1cc5012decc0683c21ae1fc591889b8e777cedcab391511a28d27a6b2e3b
$
```

Test Container

You can test the container is accessible using curl. If the application responds then you know that everything has correctly started.

```
✓ $ curl http://docker:3000
<!DOCTYPE html><html><head><title>Express</title><link rel="stylesheet" href="/stylesheets/style.css">
</head><body><h1>Express</h1><p>Welcome to Express</p></body></html>$
```

h) Set the Environment Variables

With Docker, environment variables can be defined when you launch the container. For example with Node.js applications, you should define an environment variable for *NODE_ENV* when running in production.

Using *-e* option, you can set the name and value as *-e NODE_ENV=production*

```
docker run -d --name my-production-running-app -e NODE_ENV=production -p 3000:3000 my-nodejs-app
```

A docker image of node.js application has been created.