Docker and Kubernetes

docker introduction

docker run

```
docker run helloworld
docker run busybox echo hi
docker run busybox ls
```

docker run is combination of creating {start} and running (run) an container

```
docker create hello-world``
docker start <container-id>  ##  doesnt show output by default
docker start -a <container-id>`
```

docker start can also be used to used to restart stopped container

If we forget "-a" with start then we can print output using

```
docker logs <container-id>
docker stop <container-d>       ## issues sigterm for processes
docker kill <container-id>      ## issues sigkill for processes
```

ps command

```
docker ps       ## currently running docker containers
docker ps --all         ## list of all container ever created
```

deleting container

```
docker system prune ## deletes all stopeed container
```

interactive terminal

installing redis server using docker

```
docker run redis        ## starts a redis server
```

How to attached redis-cli to this server??

```
docker exec -it <container-id> <command>        ## -it allows to provide
input to container
```

-i to accept input -t to show std out in formatted manner

```
docker exec -it <id> redis-cli
```

to get shell for container

```
docker exec -it <id> sh
```

to create a new container with shell

```
docker run -it <image> sh
```

creating docker file

- specify base image
- run commands to install additional programs
- specify commands to run at startup

Make a directory creating a docker file -- create file dockerfile

```
### content of docker file
# use existing image
FROM alpine

# install packages
RUN apk add --update redis

# run command
CMD ["redis-server"]
```

execute following command

```
docker build .
```

modifying and rebuilding existing image modify dockerfile and run

```
docker build .
```

Tagging an Image

Tagging is a way to make my custom image more memorizable for next runs. Format of tag

```
<yourdockerid>/<projectname>:version
```

```
docker build -t <tagName> . # <dot> says to use cwd for build
docker run <yourdockerid>/<projectname>
```

creating docker image manually without docker file

This is not recommended way -- just for understanding

```
docker run -it alpine sh
## this opens custom shell on new machine
apk add --update redis

## then open a new window to add initial run command
docker commit -c 'CMD ["redis-verver"]' <container-id>
```

simple nodejs project with docker

create a simple nodejs project with package.json and index.js. Create docker file in same directory with contect below:

```
#specify base image
FROM node:alpine

## change directory to working directory in docker filesystem
WORKDIR /usr/myApp

## copy package.json first to workdir
COPY ./package.json .
#then run
RUN npm install
```

```
## then copy other files. this is done later because change in js file
later will invoke npm install as cache will be invalid
COPY ./ .

## start script
CMD ["npm", "start"]
```

then run following in terminal

```
docker build -t <myname>/<projectname> .
## then run docker with port mapping
docker run -p <hostportnumber>:<vmportnumber> <imageTagId>
```

Multicontainer local application using docker compose

We can create two container shown below, where one container is running node application which is using redis database to store and fetch values.

![app1](Screenshot_2019-04-19_10-12-04.png =500x)

But how can these container talk to each other?? To solve this comes the docker compose

docker compose

- seperate cli installed with docker
- used to start and manage multiple containers at same time
- Avoid repetitive code

This is how docker-compose.yml looks like

![docker_compose.yml](./docker-compose_yml.png =300x)

To use docker-compose

- create docker-compose.yml in nodejs project folder
- address redis-server saying name "radisServer" as named in docker-compose.yml file
- Now docker-compose.yml file looks as shown below:

```
version: '3'                # version of docker
services:                   # declaring containers
    redisServer:            # name of radis container
        image: 'redis'      # just run using image
    node-app:
        build: .            # use dockerFile in curr folder to start build
and run this app
        ports: 8081:8081    # port mapping
```

Fire docker-compose using

```
docker-compose up
# to rebuild
docker-compose up --build
```

To stop the running container

```
docker-compose down
```

To see docker compose instances running

```
docker-compose ps # from dir docker compose was fired
```

Restart policies

if container closes accidently restart policies are used. By default restart policy is "no".

![docker_compose.yml](./restart_policy.png =400x)

Docker flow for development

![prod](./prod.png =500x)

Lets create a react project first

```
npm install -g create-react-app
create-react-app myFrontendApp

npm run start # to start app in react server for development
npm run test # to run test
npm run build # to create build for production
```

Instead of copy files to container, linking is a much better idea so that local changes rebuild server running react application

```
docker run -p 3000:3000 -v /app/node_modules -v $(pwd):/app <image_id>
```

Now we will have two docker files

- DockerFile.dev for development run
- dockerFile for production and deployment

create docker file for development in myFrontEndApp folder with name DockerFile.dev

```
#specify base image
FROM node:alpine

## change directory to working directory in docker filesystem
WORKDIR /usr/myApp

## copy package.json first to workdir
COPY ./package.json .
#then run
RUN npm install

## then copy other files. this is done later because change in js file
later will invoke npm install as cache will be invalid
COPY ./ .   # can be deleted as this is linked to original dir. Keeping
this means no harm

## start script
CMD ["npm", "run", "start"]
```

running docker during development

```
docker build -f DockerFile.dev .
```

lets use docker-compose.yml to build instead of building using command line

```
version: '3'                # version of docker
services:                   # declaring containers
    web:                    # name this container
        build: .            # needed as dockerfile name is custom
            context: .      # path of dockerfile
            dockerfile: DockerFile.dev
        ports:
            - 8081:8081    # port mapping
        volumes:
            - /app/node_modules
            - .:/app
```

running test during development

```
docker run -it <container-id> npm run test
```

integrating test running part to docker-compose

```
version: '3'                # version of docker
services:                   # declaring containers
    web:                    # name this container
        build: .            # needed as dockerfile name is custom
            context: .      # path of dockerfile
            dockerfile: DockerFile.dev
        ports:
            - 8081:8081    # port mapping
        volumes:
            - /app/node_modules
            - .:/app
    test:                    # name this container
        build: .            # needed as dockerfile name is custom
            context: .      # path of dockerfile
            dockerfile: DockerFile.dev
        ports:
            - 8081:8081    # port mapping
        volumes:
            - /app/node_modules
            - .:/app
        command: ["npm", "run", "test"]
```

so overall flow we developed above looks like:

![./testflow.png](testflow.png =500x)

Docker flow for production

Overall flow in development mode looks like:

![./devenv.png](devenv.png =500x)

and desired flow in production mode should be:

![./prodenv.png](prodenv.png =500x)

To achive above production enviroment steps are:

![./prod_bld_run_phase.png](prod_bld_run_phase.png =500x)

So based on this overall create a file DockerFile which will be used for production environment

```
FROM node:alpine as builder
WORKDIR '/app'
COPY package.json .
RUN npm install
COPY . .

FROM nginx
COPY --from=builder /app/build /usr/share/nginx/html
```

Now that production environment is setup we need to setup travis_ci by linking it with github and beanstalk. So that whenever a code is changed in master branch travis_ci runs tests which on passing deploys code to elastic beanstalk

To achieve this follow lecture in section 7

Developing multicontainer application

This is a simple app to compute fibonacci of a given number and store previous results in redisdb and queries in posgres. Overall app looks like