International Journal for Modern Trends in Science and Technology, 6(12): 294-300, 2020

ISSN: 2455-3778 online
DOI: https://doi.org/10.46501/IJMTST061255
Available online at: http://www.ijmtst.com/vol6issue12.html

Introduction to Containerization

Raghav Goel[1] | Dr. Bhoomi Gupta[2]

[1]UG student, IT, Maharaja Agrasen Institute Of Technology, Delhi, India.
[2]Asst. Prof., IT, Maharaja Agrasen Institute Of Technology, Delhi, India.

To Cite this Article
Raghav Goel and Dr. Bhoomi Gupta, "Introduction to Containerization", *International Journal for Modern Trends in Science and Technology*, 6(12): 294-300, 2020.

Article Info
Received on 10-November-2020, Revised on 02-December-2020, Accepted on 06-December-2020, Published on 14-December-2020.

ABSTRACT

Are you a software engineer/developer/coder or maybe even a tech enthusiast who is thinking of agility, parallel development and reducing cost. In the early twentieth century, we witnessed the rise of Service Oriented Architecture (SOA), which is a software architecture pattern that allows us to construct large-scale enterprise applications that require us to integrate multiple services, each of which is made over different platforms and languages through a common communication mechanism, where we write code and multiple services talk to each other's for a business use case, but sometimes we end up with one big monolithic code base whose maintenance becomes difficult.
Nowadays clients are using cloud and paying for on-demand services without effectively utilizing resources. These problems invite micro-services.
In this paper, I am going to discuss how one should use scale application in a production environment and local machine

KEYWORDS: *SOA, monolithic, micro-services*

INTRODUCTION

Let's see what is SOA?

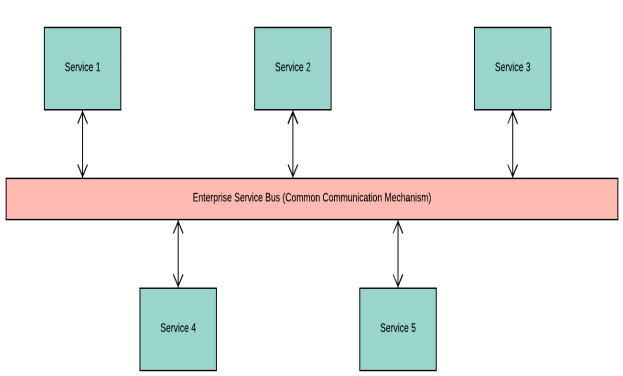


Figure 1

Key Points

1. SOA is preferred for large-scale software products such as enterprise applications
2. SOA focuses on integrating multiple services in a single application rather than emphasizing on modularizing the application
3. The common communication mechanism used for interaction between various services is called Enterprise Service Bus (ESB)

What is Monolithic Architecture?

For example in an online travel shopping company website multiple devices can access most sites, so they have various user interfaces for laptop and mobile views.

Also multiple services are running with each other to ensure the regular functioning of application services are like account creation, displaying travel plan catalog, building and validating your shopping cart, generating bill, order confirmation, payment mechanism etc.

In a monolithic application, all these services run under a single application layer which looks like

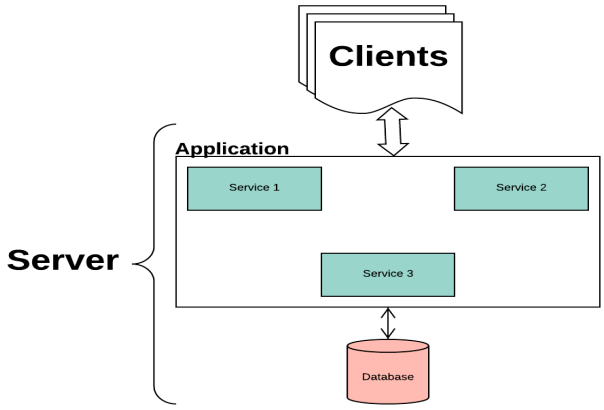


Figure 2

Drawbacks

1. If the application grows in size with the increase in the number of services offered which might become staggering for developers to maintain the application codebase
2. Every change requires developers to rebuild the entirety of the application, which wastes resources.
3. As customer base increases, we will have more requests to process, which will require more resources. So, it is essential to build products that can scale.

Microservice The Rescue

Monolithic architecture drawbacks can be overcome with Microservice architecture which focuses on modularizing the application by dividing it into smaller standalone services that can be built, deployed, scaled and even maintained independently of other existing services or the application itself as a whole. These independent services are called microservices

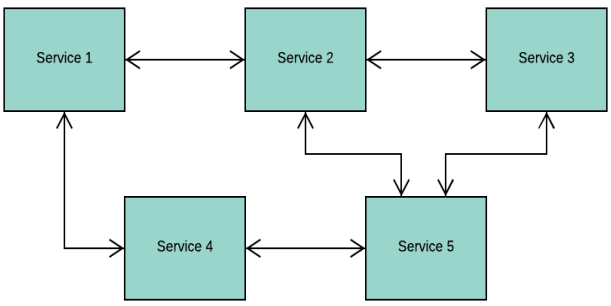


Figure 3

Highlights

1. Microservice Architecture is considered to as specialization of SOA. Or SOA can be considered to be a superset of Microservices Architecture

Advantages

1. Introduces the philosophy of Separation of Concerns and ensures Agile Development of software applications
2. The independent nature of microservices open doors for following benefits:
   - Reduces complexity by allowing developers to break into small teams, each of which builds/maintains many services
   - Easy maintenance by allowing flexibility to incrementally update/upgrade the technology stack for many services, rather than the entire application in a single point in time
3. Also a fully automated deployment mechanism can be build for ensuring individual service deployments, service management and autoscaling of the application

METHODOLOGY

Evolution Of Technologies

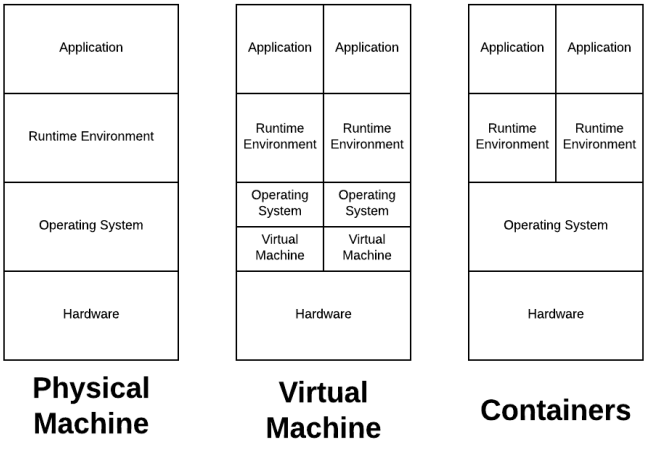


Figure 4

The first picture shows a physical machine. Typically, when we build and deploy applications, we use the resources provided by our host. But if we want to scale the application? At some point, you might want another hardware server and as the number keeps increasing, so does your cost and other resources like hardware and energy consumptions.

In the second diagram, VMs have their guest OS which runs over a single physical machine or host

OS which allow to run multiple applications without needing installing numerous physical machines.

Though VMs make software more accessible to maintain and reduced costs, more optimization was still possible.

These problems led to the next innovation: containerization. Unlike virtual machines which were more operating system specific, containers are application specific, making them far lighter. This lead to two things:

1. Multiple containers can run on a single VM. In either case, it solves application related problems
2. Containerization is not in competition with Virtualization, but rather a complementary factor to further optimize IT infrastructure

So Why do we use Docker?

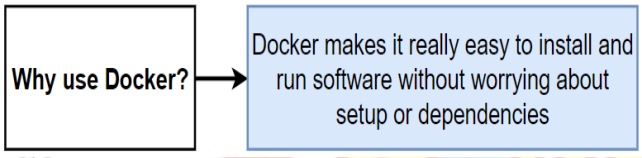


Figure 5

It is containerization technology. "Docker containers wrap a piece of software in a complete file system that contains everything needed to run: code, runtime, system tools, system libraries — anything that can be installed on a server which guarantees that the software will always run the same, regardless of its setup or dependencies."

Docker for Windows/ Mac

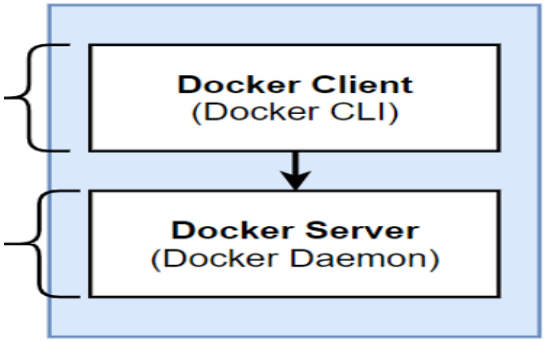


Figure 6

Docker Client -*Tool that we are going to issue commands to.*

Docker Server *- Tool that is responsible for creating images, running containers, etc*

The New Way is to deploy containers based on OS-level virtualization rather than hardware virtualization. These containers are isolated from each other and from the host they have their own file systems..

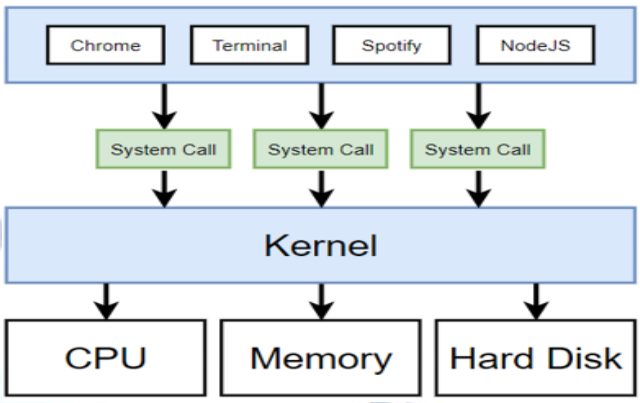


Figure 7

- Chrome, Terminal, spotify, NodeJS are Processes running on your computer.
- **System calls –** These are the calls that running programs issues request to kernel to interact with piece of hardware.

Representation of all Proesses running on the computer

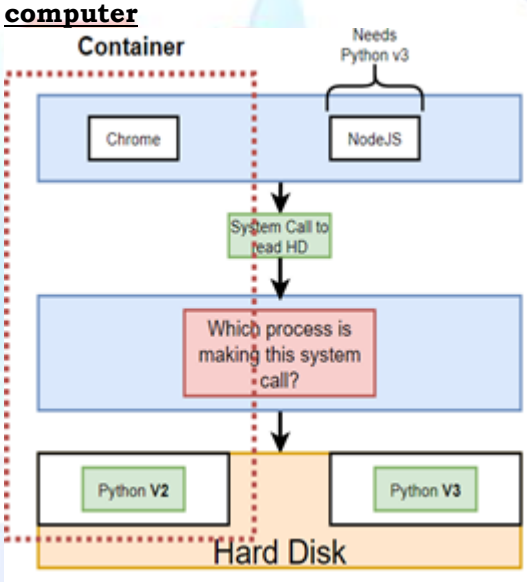


Figure 8

Representation of a container

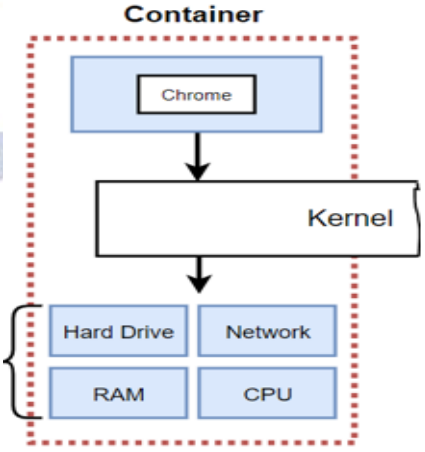


Figure 9

Hard Drive, Network, RAM, CPU are the portions of each made available to process

What is a Docker Image?

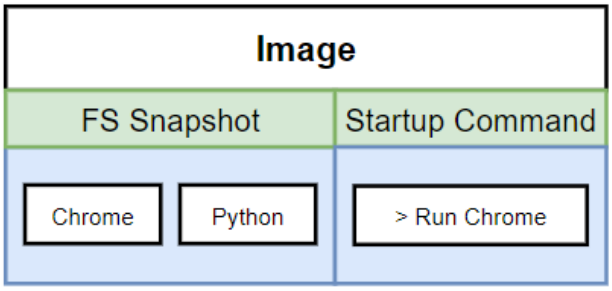


Figure 10

A Docker Image consists of two portions, one is the File System Snapshhot and the other one is the Startup command which can be anything.

Creating and Running a Container from an Image

docker	run	<image name>	command!

Docker – reference the Docker client
Run – Try to create and run a container
<image name> - Name of the image to use for this container
Command – Default command override.

Representation of all processes running on computer and Docker image

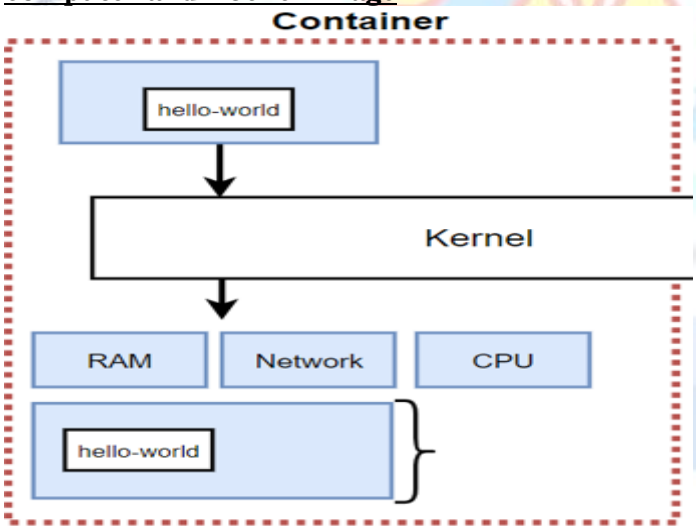


Figure 11

At the top of container the hello-world program is the Running process and at bottom the hello-world is the hard drive segment of this process.

Generate List all running containers

To generate the list of all the containers in the running processes we use the command "docker ps" where ps stands for process status.

docker	ps

Whereas, the Docker run command is the combination of the 2 commands. i.e.

Docker run = docker create + docker start

Create a Container	docker	create	<image name>

Figure 12

Create – tries to create the container. ***<image-name>*** - Name of the image to use for this container
Start – tries to start the container.
<container id> - gives the ID of the container to the start

Start a Container	docker	start	<container id>

Get logs from a container

Docker logs <container id>

Execute an additional command in a container

docker	exec	-it	<container id>	<command>

Figure 13

Exec – it is used to run another command.
-it flag – it is used to provide input to the container.

bash	powershell	zsh	sh

These are the Command Processors generally used.

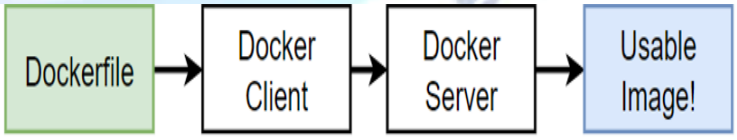
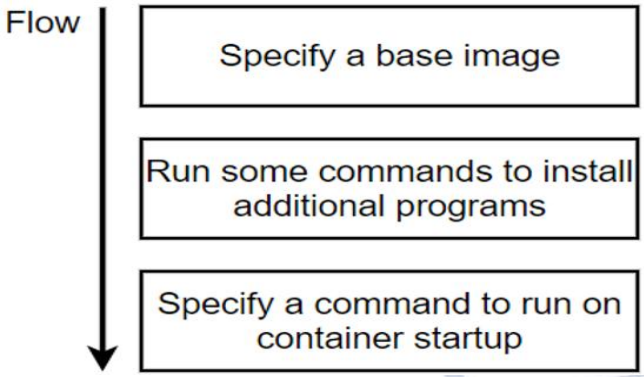


Figure 14

"Dockerfile" – It is the configuration used to define how our containers should behave.

For Create a DockerFile

For creating a dockerfile these three steps are followed in the same order.



What is a Redis?

Redis is an open source key-value store that functions as a data structure server.

Creating a Redis image?

On the footmaps of the three steps mentioned, a redis image can be craeted in the following manner. It is exactly similar to creating a Dockerfile shown as below

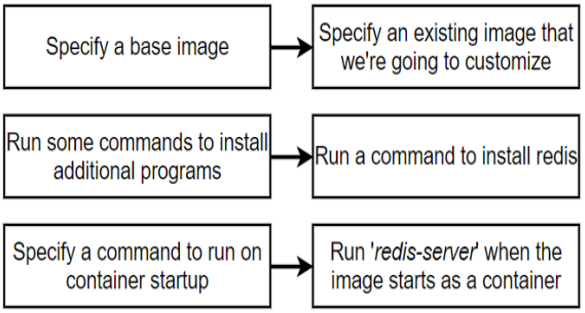


Figure 15

To create a redis image from the terminal -

```
FROM alpine
RUN apk add --update redis
CMD ["redis-server"]
```

code 1

One could simply ask "why did we use alpine as a base image?"

Hence it can be corelated with the fact that, why do we use Windows, MacOS, or Ubuntu? Since they come with a preinstalled set of programs that are useful to you!

Tagging an Image

docker	build	-t	.

"-t <name>" – Tage the image

. Specifies the directory of files/folder to use for the build process

How to create a simple web server using NodeJS?

It can be created in the 5 steps below in the same sequence.

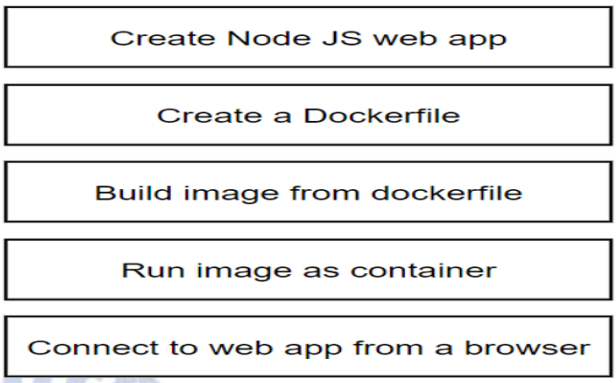


Figure 16

These are simple 5 steps which leds us to create a simple web server using NodeJS.

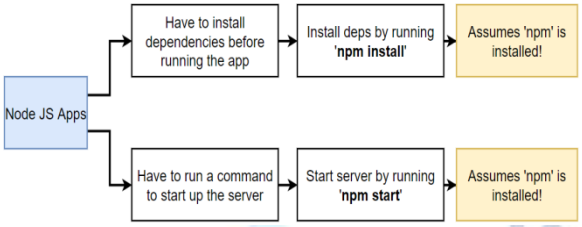
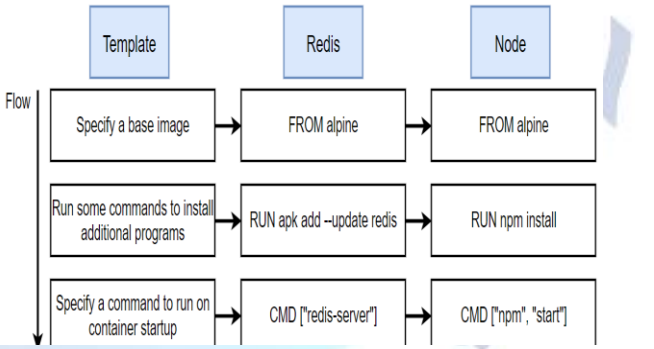


Figure 17

This chart illustrates that with Nodejs apps we have to install the dependencies before running the app with command "npm install" and to start up the server with the command "npm start"

Representation relationship between the Template, Redis and NodeJs



This shows that the a Redis server or a Node server can be created with the similar steps as shown above

Docker Commands in the code editor

```
# Specify a base image
FROM node:alpine

WORKDIR /usr/app

# Install some depenendencies
COPY ./package.json ./
RUN npm install
COPY ./ ./

# Default command
CMD ["npm", "start"]
```

code 2

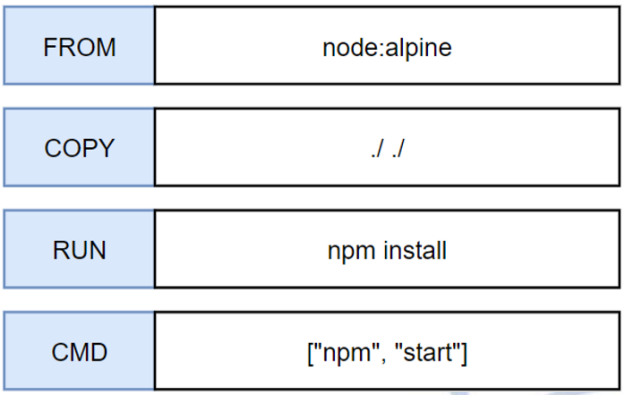
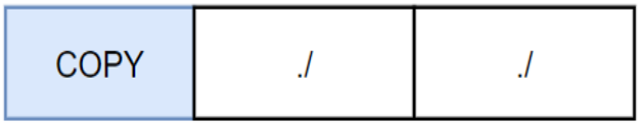


Figure 18

FROM, COPY, RUN, CMD are the instructions telling the docker server what to do and adjacent to it are the arguments to the instruction.

Copy command



"COPY ./" - path to folder to copy from on your machine relative to build context.
"COPY ./ ./" – place to copy stuff to inside the conatiner

Port Mapping : Docker Run with Port Mapping

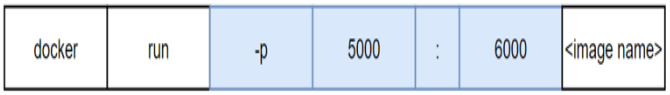
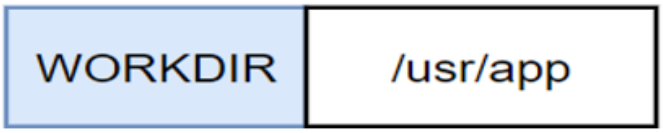


Figure 19

5000 : it is the port which route the incoming requests to this port on the local host to.
6000 : this is the port inside the container



WORKDIR – It stands for work directory
/usr/app – Any following command will be executed relative to this path in the container.

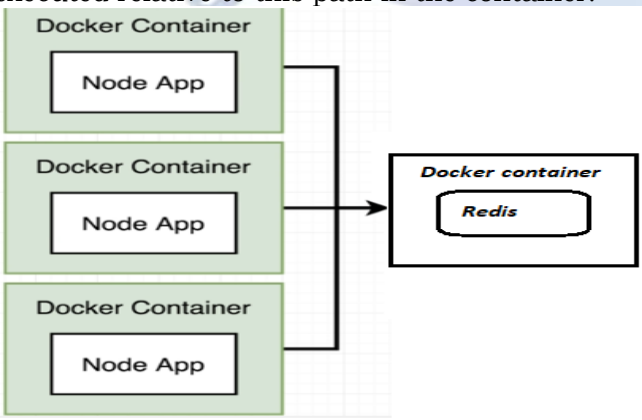


Figure 20

This diagram illustrates that 3 docker containers which are interconnected with each other are connected to one docker conatiner which is the redis server. In this way we can scale the complete application.

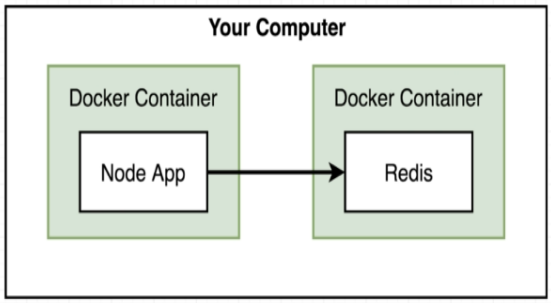


Figure 21

It signifies that there are two docker containers in the computer, which perform the work of Node App and Redis Server as shown below.

What is Docker compose?

In simple steps docker-compose can be understood as below in the same steps

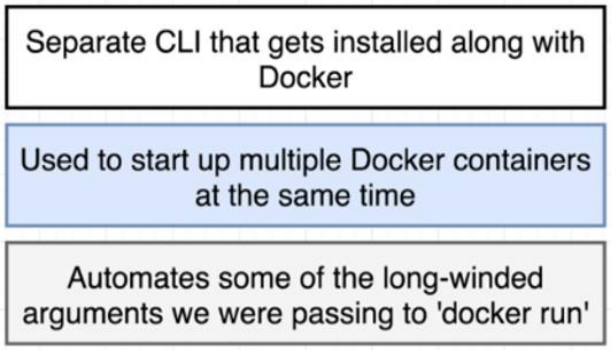


Figure 22

Docker compose is the separate CLI that gets installed along with docker which is used to startup multiple docker containers at the same time. It also automates some if the long winded arguments which we passed to docker-run.

The whole purpose of docker-compose is to make executing docker run easier.

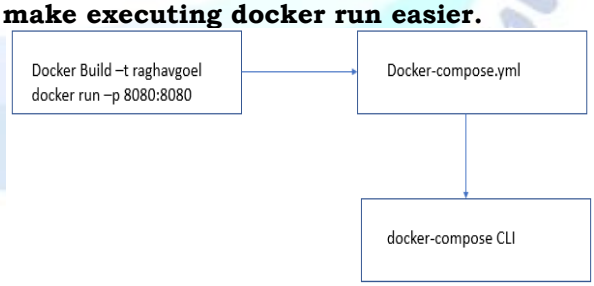


Figure 23

Blueprint of docker-compose.yml file

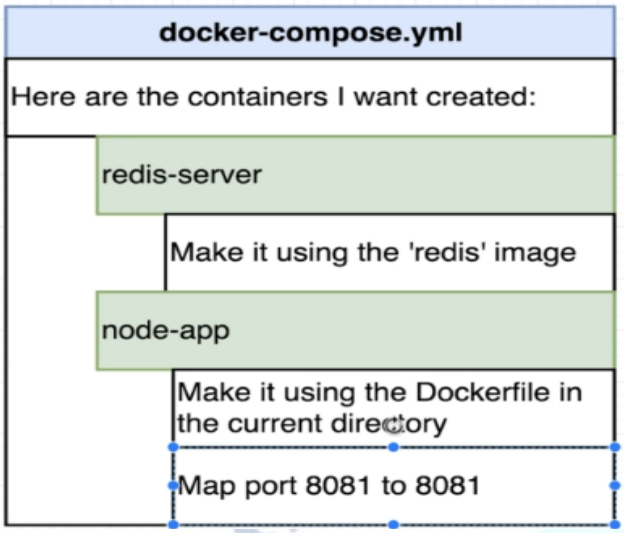


Figure 24

In the docker-compose.yml file I created two containers, redis server and the node-app container and mapped it onto the required port which is 8081:8081.

Results

```yaml
version: '3'
services:
  redis-server:
    image: 'redis'
  web:
    build:
      context: .
      dockerfile: Dockerfile.dev
    ports:
      - '3000:3000'
    volumes:
      - /app/node_modules
  tests:
    build:
      context: .
      dockerfile: Dockerfile.dev
    volumes:
      - /app/node_modules
      - .:/app
    command: ['npm', 'run', 'test']
```

code 3

DISCUSSION

Practical use of docker-compose commands-

Suppose myimage is some any image which we want to run and build.

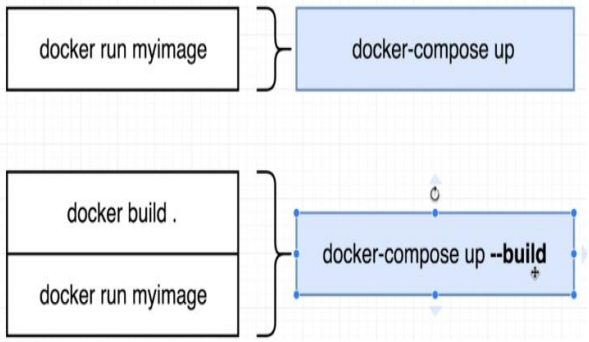


Figure 25

"build . " assumes that we'd dockerfile inside of current working directory

Launch and Stop Docker containers using compose commands

Command : docker-compose up-d

Figure 26

Stop docker conatiner command : docker-compose down

Figure 27

Restart policies in the docker environment –

There are generally 4 restart policies which are no, always, on-failure and unless-stopped. These restart policies are explained below in the tabular form.

"no"	Never attempt to restart this . container if it stops or crashes
always	If this container stops *for any reason* always attempt to restart it
on-failure	Only restart if the container stops with an error code
unless-stopped	Always restart unless we (the developers) forcibly stop it

Figure 28

REFERENCES

https://docs.docker.com/
https://hub.docker.com/?ref=login
https://kubernetes.io/
https://www.docker.com/resources/what-container
https://en.wikipedia.org/wiki/Service-oriented_architecture