Docker compose is an orchestration tool for docker that allows you to define a set of containers and their interdependencies in the form of a YAML file. Once the yaml file is created , we can use the docker-compose commands to create the whole application stack. Besides this we can track the application output and various other things.

The docker-compose is a tool for defining and running multi-container application. In this we create a compose file to configure our application services. Then by using a single command we create all images necessary for the container, Start the container. We just need to access the application running inside the container

Using Compose is basically a three-step process.

1. Define your app's environment with a Dockerfile so it can be reproduced anywhere.
2. Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.
3. Lastly, run docker-compose up and Compose will start and run your entire app.

1.Let's write a basic example of running a java application inside the container. Below is the directory structure of my first docker compose example

```
[puppet@root$:/test/docker]$  tree docker-compose/
docker-compose/
├── compose
│   ├── docker-compose.yml
│   ├── ping
│   └── PingPong
│       ├── Dockerfile
│       └── PingPong.java
└── PingPong
```

2. Lets write our java code for the PingPong.java class

```
[puppet@root$]$  cat PingPong.java
import java.io.IOException;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpHandler;
import com.sun.net.httpserver.HttpServer;

public class PingPong {

        public static void main(String[] args) throws Exception {
        HttpServer server = HttpServer.create(new InetSocketAddress(19090), 0);
        server.createContext("/ping", new MyHandler());
        server.setExecutor(null);
        server.start();
        }

        static class MyHandler implements HttpHandler {
        @Override
        public void handle(HttpExchange t) throws IOException {
        String response = "pong\n";
        t.sendResponseHeaders(200, response.length());
        OutputStream os = t.getResponseBody();
        os.write(response.getBytes());
        os.close();
        }
```

```
        }
}
```

The above java code is a simple HTTP listener which responds to the requests made on the port 19090 with a response. Once the user sends a request called "ping" , the above code responds with a reponse of "pong".

3.Besides the PingPong.java class we have the Dockerfile with contents as,

```
[puppet@root$]$  cat Dockerfile
FROM java:8
MAINTAINER Jagadish Manchala <jagadishm@example.com>
WORKDIR /
COPY PingPong.java /
RUN javac PingPong.java
EXPOSE 19090
CMD ["java","PingPong"]
```

The instructions are self explanatory. We are using a Java 8 based container. The work directory is / and we are moving the PingPong.java class to this contianer location "/". We also compiling the java class and running the code by exposing the 19090 port.

4. Now come out of the directory and we can have the compose file as,

```
[puppet@root$]$  cat docker-compose.yml
version: '2'
services:
 PingPong:
        build: ./PingPong
        image: pingpong:1
        ports:
        - "5000:19090"
        volumes:
        - ../PingPong:/code
```

We start off with the line "version: '2'", which tells Docker Compose we are using the new Docker Compose syntax. We define a single service called pingpong, which runs from an image called pingpong:1. It exposes a single port 5000 on the docker host that maps to port 19090 inside the container.

The "build: ./PingPong"

We've added a single line "build: ./PingPong" to the helloworld service. It instructs Docker Compose to enter the compose/PingPong directory, run a docker build there, and tag the resultant image as pingpong:1.

Now if you tried to run this as-is, using "docker-compose up", docker could complain that it couldn't find pingpong:1. That's because it's looking on the docker hub for a container image called pingpong:1. We haven't created it yet. So now, let's add the recipe to create this container image.

Now run the **docker-compose up --force-recreate** command and we can see below output as,

```
[puppet@root$:/test/docker/docker-compose/compose]$  docker-compose up --force-recreate
Creating network "compose_default" with the default driver
Building PingPong
Step 1 : FROM java:8
---> d23bdf5b1b1b
```

```
Step 2 : MAINTAINER Jagadish Manchala <jagadishm@example.com>
---> Running in 278b9e78e8ac
---> db3eeedd3862
Removing intermediate container 278b9e78e8ac
Step 3 : WORKDIR /
---> Running in cbfa16414694
---> 2ee46f20f539
Removing intermediate container cbfa16414694
Step 4 : COPY PingPong.java /
---> dc039d682eca
Removing intermediate container 2771d925c2d9
Step 5 : RUN javac PingPong.java
---> Running in 792a5fbfdc57
---> 3e9a9ddac467
Removing intermediate container 792a5fbfdc57
Step 6 : EXPOSE 19090
---> Running in 2efa5017a79e
---> 8164eb703ff4
Removing intermediate container 2efa5017a79e
Step 7 : CMD java PingPong
---> Running in ef7347c01dee
---> bd15166ab1a7
Removing intermediate container ef7347c01dee
Successfully built bd15166ab1a7
WARNING: Image for service PingPong was built because it did not already exist. To rebuild this image you must use `docker-compose build` or `docker-compose up --build`.
Creating compose_PingPong_1
Attaching to compose_PingPong_1
```

Now in the above output , we have the docker-compose command waiting after the Attaching to compose_*** command. At this point when we see the docker images , we can see a pingpong image created. And When we run the "docker ps" command we can see the container running as below,

```
[puppet@root$:/test/docker/docker-compose/compose]$  docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
078f287da46c        pingpong:1          "java PingPong"     About a minute ago
STATUS              PORTS                    NAMES
Up About a minute   0.0.0.0:5000->19090/tcp  compose_PingPong_1
```

Now we can test the Java Code running inside the Container using

```
[puppet@root$:/test/docker/docker-compose/compose]$  wget http://localhost:5000/ping
--2017-02-21 08:13:22--  http://localhost:5000/ping
Resolving localhost (localhost)... ::1
Connecting to localhost (localhost)|::1|:5000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5
Saving to: 'ping'

100%[=======================================================
=====================================================>] 5       --.-
K/s   in 0s

2017-02-21 08:13:22 (1.52 MB/s) - 'ping' saved [5/5]
```

Or we can take a browser and type the URL as "http://localhost:5000/ping"

To stop the application, simply type Ctrl-C at the terminal prompt and Docker Compose will stop the container and exit. You can go ahead and change the code in the PingPong directory, add new code or modify existing code, and test it out using "docker-compose up" again.

To run it in the background: docker-compose up -d.
To tail the container standard output: docker-compose logs -f.
To down the container: docker-compose down -v
To Force recreate: docker-compose up --force-recreate