INFINITE BACK-END DEPLOYMENT:

INSTALLATION OF DOCKER AND DOCKER-COMPOSE ON VIRTUAL MACHINE:

Link for installation steps: https://docs.docker.com/engine/install/ubuntu/

SET UP THE REPOSITORY:

```
sudo apt-get update
sudo apt-get install \
    ca-certificates \
    curl \
    gnupg
```

ADD DOCKER'S OFFICIAL GPG KEY:

```
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/
keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg
```

USE THE FOLLOWING COMMAND TO SET UP THE REPOSITORY:

```
echo \
  "deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg] https://
download.docker.com/linux/ubuntu \
  "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

INSTALL DOCKER ENGINE:

1. Update the `apt` package index:

```
sudo apt-get update
```

2. Install Docker Engine, containerd, and Docker Compose:

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-
plugin
```

Verify that the Docker Engine installation is successful by running the `hello-world` image:

```
sudo docker run hello-world
```

COMMANDS TO CHECK DOCKER AND DOCKER-COMPOSE VERSIONS:

```
Sudo docker — version
Sudo docker compose version
```

For a deployment need to create docker-compose.yml which contains Microservices of an application running on backend virtual machine.

FORMAT OF DOCKER-COMPOSE.YML:

Command To create docker-compose.yml file

```
Sudo vi docker-compose.yml
```

```
version: '3.6'
services:
  mongodb:
    container_name: mongodb
    image: mongo:${MONGODB_VERSION}
    environment:
      MONGO_INITDB_ROOT_USERNAME: ${MONGO_INITDB_ROOT_USERNAME}
      MONGO_INITDB_ROOT_PASSWORD: ${MONGO_INITDB_ROOT_PASSWORD}
    networks:
      - name of network
    ports:
      - ${MONGODB_PORT}:${MONGODB_PORT}
    volumes:
      - ./mongodb:/data/db
  encounters:
    depends_on:
      - mongodb
    networks:
      - name of network
    env_file:
      - ./env/encounters.env
    Image:microservice image address:${ENCOUNTERS_API_VERSION}
    ports:
      - ${ENCOUNTERS_API_PORT}:${ENCOUNTERS_API_PORT}

  emr:
    depends_on:
      - mongodb
    networks:
      - name of network
    env_file:
      - ./env/emr.env
    image: microservice image address:${EMR_API_VERSION}
    ports:
      - ${EMR_API_PORT}:${EMR_API_PORT}

  emrcms:
    networks:
      - name of network
    env_file:
      - ./env/emrcms.env
    image: microservice image address:${EMR_CMS_API_VERSION}
    ports:
      - ${EMR_CMS_API_PORT}:${EMR_CMS_API_PORT}

  infinite:
    depends_on:
      - mongodb
    networks:
      - name of network
    env_file:
      - ./env/infinite.env
    image:microservice image address:${INFINITE_API_VERSION}
    ports:
      - ${INFINITE_API_PORT}:${INFINITE_API_PORT}

networks:
  infinite-network:
    driver: bridge
    name: name of network
```

For a deployment we need to have ENV files of that deploying micro services which contains connection strings of that micro service.

Create env directory and place it all micro service env files in that.

Command to create directory: Sudo mkdir env

Here we are using micro services as:

Encounters.env
Emr.env
Emrcms.env
Infinite.env

And also we are creating .env file which contains variables declared in docker-compose.yml
To avoid hardcoding container PORTS and micro service VERSIONS.
Command to create .env is sudo vi .env

```
MONGODB_VERSION=
MONGODB_PORT=
MONGO_INITDB_ROOT_USERNAME=
MONGO_INITDB_ROOT_PASSWORD=
ENCOUNTERS_API_PORT=
ENCOUNTERS_API_VERSION=
EMR_API_PORT=
EMR_API_VERSION=
INFINITE_API_PORT=
INFINITE_API_VERSION=
EMR_CMS_API_PORT=
EMR_CMS_API_VERSION=
```

We should mention PORTS of micro services as same as env file of micro service and in .env file,
At the same time we should mention database URL for each micro service in that env file.

After completion of creating docker-compose.yml, env, .env

Comand to start micro services : sudo docker compose up -d

It will denied to pull images from ghcr.io because we do not login to GitHub.

To pull images from ghcr.io we should login to our GitHub for that create classic token in your GitHub and use below commands to login.

```
export CR_PAT=your classic token
echo $CR_PAT | sudo docker login ghcr.io -u (your GitHub username) —password-stdin
```

Command to check docker containers: sudo docker container list

Then map your VM IP to domain name ex: domain.hng.one

Install nginx in your VM as web server for proxy and reverse-proxy services

NGINX INSTALLATION:

LINK for installation steps: https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-22-04

Step 1 – Installing Nginx:

```
sudo apt update
sudo apt install nginx
```

Step 2 – Adjusting the Firewall:

```
sudo ufw app list
sudo ufw allow 'Nginx HTTP’
sudo ufw status
```

Step 3 – Checking your Web Server:

```
systemctl status nginx
```

After installation create config file in nginx for micro services proxy service.
Config file creating path= /etc/nginx/sites-enabled
Config file should be as domain name domain.hng.one

EX:

```
server {
  server_name domain.hng.one;

  # Point upstream to Chatwoot App Server
  set $encounters 127.0.0.1:3001;
  set $emr 127.0.0.1:3002;
  set $cms 127.0.0.1:1337;

  # Nginx strips out underscore in headers by default
  # Chatwoot relies on underscore in headers for API
  # Make sure that the config is set to on.
  underscores_in_headers on;

  location /.well-known {
    alias /var/www/ssl-proof/prohealth/.well-known;
  }

  location /encounters {
    rewrite ^/encounters(.*)$ $1 break;
    proxy_pass http://$encounters;
  }

  location /emr {
    rewrite ^/emr(.*)$ $1 break;
    proxy_pass http://$emr;
  }

  location /cms {
    rewrite ^/cms(.*)$ $1 break;
    proxy_pass http://$cms;
  }



  location / {
      proxy_pass http://$cms;
  }

  listen 80;
}
```

After saving config file restart nginx servicer

Command to restart nginx : sudo systemctl restart nginx

Then use domain name in web browser to see our micro services working or not

SSL CERTIFICATE FOR OUR DOMAIN:

LINK for steps to follow ssl certification: https://certbot.eff.org/

1.Ensure that your version of snapd is up to date:

sudo snap install core; sudo snap refresh core

2.Install Certbot:

sudo snap install --classic certbot

3.Prepare the Certbot command:

sudo ln -s /snap/bin/certbot /usr/bin/certbot

4.Choose how you'd like to run Certbot:

sudo certbot --nginx

Give your mail for ssl renewal

Enable inbound traffic to our application PORTS 80,443 in azure portal then only it will work.

If you have emrcms as a private deployment for that we are using Postgres as a database

In that emrcms micro service env file which contains

Database name
Database username
Database password

We should create the above credentials in azure portal then only the micro service will work.

INFINITE FRONT-END(FIREBASE) DEPLOYMENT

Go to mede-consultation-angular repository

SET ENVIRONMENT.TS

Inside environment.ts for each micro service of our application place that micro service URL

SET PACKAGE.JSON

Inside of package.json set

```
"start:infinite:demo": "ng serve --port 4200 —configuration=infinite-demo",

 "deploy:demo:infinite": "cp ./config/firebase-messaging-sw-play.js ./src/firebase-messaging-sw.js; node --max_old_space_size=4096 ./node_modules/@angular/cli/bin/ng build --configuration=infinite-demo; firebase hosting:channel:deploy infinite-demo --project medeintegra-playground",
```

SET ANGULAR.JSON

```
 "infinite-demo": {
            "fileReplacements": [
             {
               "replace": "src/environments/environment.ts",
               "with": "src/environments/environment.infinite.demo.ts"
             }
           ],


"infinite-demo": {
            "browserTarget": "consultation:build:infinite-demo"
          },
```