### Workflow Variables

DOCKER\_DRIVER: overlay2

This variable sets the Docker storage driver to overlay2. Overlay2 is a modern storage driver for Docker, which provides better performance and scalability compared to older storage drivers like aufs.

DOCKER\_TLS\_CERTDIR: "/certs"

This variable specifies the directory where Docker will look for TLS certificates. The "/certs" directory is used to store TLS certificates for secure communication between Docker client and server.

CI\_REGISTRY: docker.io

This variable sets the registry to docker.io, which is the Docker Hub. It tells Docker to use Docker Hub as the container registry for pushing and pulling images.

CI\_REGISTRY\_IMAGE: [docker.io/htheva/eduprodigi](http://docker.io/htheva/eduprodigi)

This variable defines the image name and repository path in Docker Hub. The image will be tagged and stored in docker.io/htheva/eduprodigi.

### Workflow Stages

The workflow consists of four stages: build, test, push, and deploy.

#### 1. Build Stage

yaml

Copy code

build\_image:

stage: build

image: docker:20.10.7-dind

services:

- docker:20.10.7-dind

script:

- docker build -t $CI\_REGISTRY\_IMAGE:$CI\_COMMIT\_REF\_SLUG edu-prod-REAL/.

only:

- main

Stage Name: build

Image: docker:20.10.7-dind: Uses the Docker-in-Docker (dind) image to run Docker commands within the CI pipeline.

Services: docker:20.10.7-dind:Enables Docker daemon to run alongside the CI job, allowing Docker commands to be executed.

Script: docker build -t $CI\_REGISTRY\_IMAGE:$CI\_COMMIT\_REF\_SLUG edu-prod-REAL/.

Builds a Docker image from the edu-prod-REAL directory.

Tags the image with the repository path and commit reference slug, which ensures each build is uniquely tagged.

Only: main: Ensures that this job runs only for the main branch.

#### 2. Test Stage

yaml

Copy code

test\_image:

stage: test

image: node:18

script:

- cd edu-prod-REAL

- npm install

- npm test

only:

- main

Stage Name: test

Image: node:18

Uses the Node.js version 18 image to run Node.js commands.

Script:

cd edu-prod-REAL

Navigates to the edu-prod-REAL directory.

npm install

Installs the project dependencies defined in package.json.

npm test

Runs the tests defined in the project.

Only: main:Ensures that this job runs only for the main branch.

#### 3. Push Stage

yaml

Copy code

push\_image:

stage: push

image: docker:20.10.7-dind

services:

- docker:20.10.7-dind

script:

- echo $DOCKER\_PASSWORD | docker login -u $DOCKER\_USERNAME --password-stdin

- docker push $CI\_REGISTRY\_IMAGE:$CI\_COMMIT\_REF\_SLUG

only:

- main

Stage Name: push

Image: docker:20.10.7-dind:Uses the Docker-in-Docker (dind) image to run Docker commands within the CI pipeline.

Services: docker:20.10.7-dind:Enables Docker daemon to run alongside the CI job, allowing Docker commands to be executed.

Script:echo $DOCKER\_PASSWORD | docker login -u $DOCKER\_USERNAME --password-stdin:Logs in to Docker Hub using the username and password stored in CI/CD variables.

docker push $CI\_REGISTRY\_IMAGE:$CI\_COMMIT\_REF\_SLUG

Pushes the built Docker image to Docker Hub, using the tag created in the build stage.

Only: main:Ensures that this job runs only for the main branch.

#### 4. Deploy Stage

yaml

Copy code

deploy\_image:

stage: deploy

image: docker:20.10.7-dind

services:

- docker:20.10.7-dind

script:

- chmod +x edu-prod-REAL/deploy.sh

- edu-prod-REAL/deploy.sh

only:

- main

Stage Name: deploy

Image: docker:20.10.7-dind

Uses the Docker-in-Docker (dind) image to run Docker commands within the CI pipeline.

Services: docker:20.10.7-dind

Enables Docker daemon to run alongside the CI job, allowing Docker commands to be executed.

Script:

chmod +x edu-prod-REAL/deploy.sh

Makes the deploy.sh script executable.

edu-prod-REAL/deploy.sh

Executes the deploy.sh script to deploy the application.

Only: main

Ensures that this job runs only for the main branch.

This workflow ensures that the application is built, tested, pushed to Docker Hub, and deployed automatically when changes are made to the main branch. Each stage is dependent on the successful completion of the previous stage, ensuring a smooth CI/CD pipeline

**Dockerfile**

FROM node:14

This line specifies the base image to use for the Docker image. In this case, it uses the official Node.js version 14 image from Docker Hub. This image comes with Node.js and npm pre-installed.

WORKDIR /app

This line sets the working directory inside the container to /app. Any subsequent commands will be run in this directory. If the directory doesn't exist, it will be created.

COPY package.json ./\*

This line copies package.json and package-lock.json files from the local machine to the current working directory in the container (/app). This is done to ensure that the dependencies specified in these files are installed.

RUN npm install

This line runs npm install inside the container to install all the dependencies listed in package.json. By doing this in a separate step, Docker can cache the installed dependencies if the package.json and package-lock.json files haven't changed, speeding up subsequent builds.

COPY . .

This line copies the rest of the application code from the local machine to the current working directory in the container (/app). This includes all source files and assets needed for the application.

EXPOSE 3000

This line specifies that the container will listen on port 3000 at runtime. This is the port that the application will use to accept incoming connections.

ENV PORT 3000

This line sets an environment variable PORT to 3000. This can be used within the application to reference the port number, allowing for easier configuration.

CMD ["npm", "start"]

This line specifies the command to run when the container starts. In this case, it runs npm start, which typically starts the Node.js application as defined in the package.json file.

Deployment Script

bash

Copy code

#!/bin/bash

# Variables

DOCKER\_IMAGE=htheva/eduprodigi:latest

CONTAINER\_NAME=eduprodigi\_container

# Stop and remove existing container if it exists

if [ $(docker ps -q -f name=$CONTAINER\_NAME) ]; then

docker stop $CONTAINER\_NAME

docker rm $CONTAINER\_NAME

fi

# Pull the latest image from Docker Hub

docker pull $DOCKER\_IMAGE

# Run the container

docker run -d -p 3000:3000 --name $CONTAINER\_NAME $DOCKER\_IMAGE

#!/bin/bash

This line specifies that the script should be run using the Bash shell.

Variables

DOCKER\_IMAGE=htheva/eduprodigi:latest

This variable defines the Docker image to use. The :latest tag indicates that the latest version of the image should be pulled.

CONTAINER\_NAME=eduprodigi\_container

This variable defines the name to give to the running container.

Stop and remove existing container if it exists

if [ $(docker ps -q -f name=$CONTAINER\_NAME) ]; then

This line checks if there is a running container with the name specified in CONTAINER\_NAME. If there is, it proceeds to stop and remove it.

docker stop $CONTAINER\_NAME

This line stops the running container.

docker rm $CONTAINER\_NAME

This line removes the stopped container, freeing up the name for a new container.

Pull the latest image from Docker Hub

docker pull $DOCKER\_IMAGE

This line pulls the latest version of the specified Docker image from Docker Hub.

Run the container

docker run -d -p 3000:3000 --name $CONTAINER\_NAME $DOCKER\_IMAGE

This line runs a new container in detached mode (-d), mapping port 3000 of the host to port 3000 of the container (-p 3000:3000), giving the container the name specified in CONTAINER\_NAME, and using the image specified in DOCKER\_IMAGE

**Deploy.sh**

1. #!/bin/bash:This line is known as the shebang. It specifies that the script should be executed in the Bash shell.
2. Variables
   * DOCKER\_IMAGE=htheva/eduprodigi:latest
     + This variable defines the Docker image to use. It specifies the repository htheva/eduprodigi and the tag latest. The latest tag usually points to the most recent version of the image.
   * CONTAINER\_NAME=eduprodigi\_container
     + This variable defines the name that will be assigned to the running container. Naming the container makes it easier to manage and reference later.
3. Stop and remove existing container if it exists
   * if [ $(docker ps -q -f name=$CONTAINER\_NAME) ]; then
     + This line checks if there is a running container with the name specified in CONTAINER\_NAME. The docker ps -q -f name=$CONTAINER\_NAME command returns the container ID if a container with that name is running.
   * docker stop $CONTAINER\_NAME
     + This command stops the running container named eduprodigi\_container. Stopping the container is necessary before removing it to ensure that no resources are being used.
   * docker rm $CONTAINER\_NAME
     + This command removes the stopped container named eduprodigi\_container. Removing the container frees up the name for a new container and clears any associated resources.
4. Pull the latest image from Docker Hub
   * docker pull $DOCKER\_IMAGE
     + This command pulls the latest version of the specified Docker image from Docker Hub. By pulling the latest image, you ensure that you are running the most recent version of your application.
5. Run the container
   * docker run -d -p 3000:3000 --name $CONTAINER\_NAME $DOCKER\_IMAGE
     + This command runs a new container in detached mode (-d), which means it runs in the background.
     + -p 3000:3000 maps port 3000 of the host machine to port 3000 of the container. This allows external access to the application running inside the container on port 3000.
     + --name $CONTAINER\_NAME assigns the name eduprodigi\_container to the new container.
     + $DOCKER\_IMAGE specifies the image to use for the container, in this case, htheva/eduprodigi:latest.

### Summary

The deploy.sh script automates the deployment process by stopping and removing any existing container with the specified name, pulling the latest version of the Docker image from Docker Hub, and running a new container with the latest image. This ensures that the deployed application is always up-to-date and minimizes downtime by quickly replacing the old container with a new on