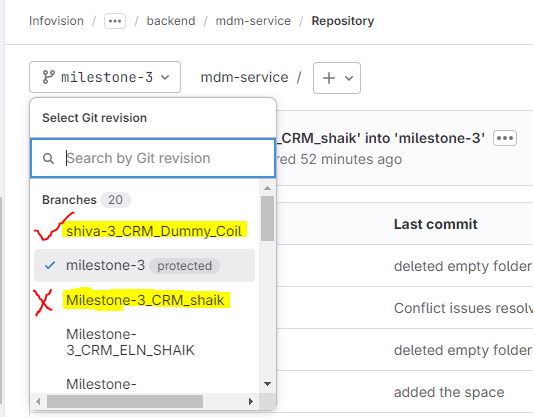
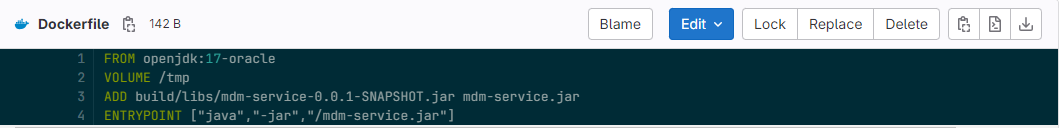
* The development code is pushed to the individual branches of developers on the GitLab repository.
* Ensure that branch names do not start with the prefix "milestone," as all milestone branches are protected.



* When the code is ready Dev team opens a merge request to merge their branches with milestone branch, once it get approved.
* Clone the microservice into local and checkout to running milestone branch and do gradle assemble to get the JAR file under build/libs path. We need JAR file to specify in docker file and gitlab yaml file.
* Set up a pipeline by adding a docker file which contains instructions for building a Docker image of your application.



* Create the .gitlab-ci.yml file which defines the CI/CD pipeline, including stages like build and build image.
* Push the changes to gitlab by using the below commands:

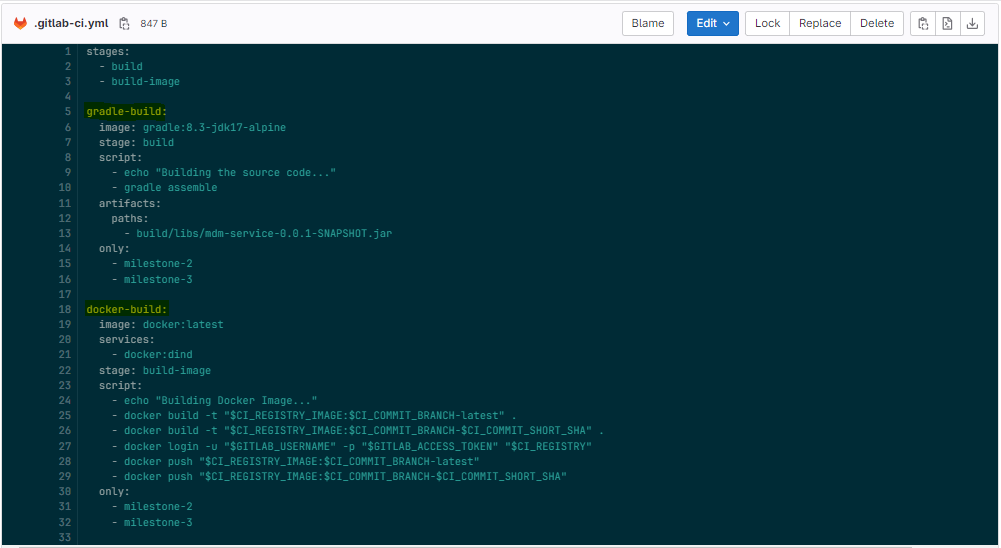
git branch – to verify the branch in where we are planning to push

git add . – to add all the files into the current branch

git commit -m “commit message”

git push

* GitLab CI/CD pipelines are automatically triggered whenever code/changes is pushed to the milestone branch
* When a push event occurs, GitLab reads the .gitlab-ci.yml file and starts executing the defined pipeline stages and jobs, , such as building the Docker image using the Docker file.



* After building the image, the image is pushed to GitLab Container Registry.

A screenshot of a computer

Description automatically generated

* Add the port number of the microservice in nginx conf file which is located in frontend/mdm-mfe/devops path.

server {

listen 80 default\_server;

server\_name \_;

return 301 https://$host$request\_uri;

}

server {

client\_max\_body\_size 8M;

large\_client\_header\_buffers 8 64k;

listen 443 ssl;

server\_name localhost;

ssl\_certificate server.crt;

ssl\_certificate\_key server.key;

ssl\_session\_timeout 5m;

ssl\_protocols TLSv1.2 TLSv1.3;

ssl\_ciphers HIGH:!aNULL:!MD5;

ssl\_prefer\_server\_ciphers on;

sendfile on;

default\_type application/octet-stream;

gzip on;

gzip\_http\_version 1.1;

gzip\_disable "MSIE [1-6]\.";

gzip\_min\_length 256;

gzip\_vary on;

gzip\_proxied expired no-cache no-store private auth;

gzip\_types text/plain text/css application/json application/javascript application/x-javascript text/xml application/xml application/xml+rss text/javascript;

gzip\_comp\_level 9;

root /usr/share/nginx/html;

location / {

root /usr/share/nginx/html/crm-base;

index index.html;

}

location /crm-base {

try\_files $uri $uri/ /index.html;

}

location /aas {

proxy\_set\_header X-Real\_IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header Host $http\_host;

proxy\_pass http://localhost:8089;

}

location /mdm {

proxy\_set\_header X-Real\_IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header Host $http\_host;

proxy\_pass http://localhost:8090;

}

location /ums {

proxy\_set\_header X-Real\_IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header Host $http\_host;

proxy\_pass http://localhost:8091;

}

location /ens {

proxy\_set\_header X-Real\_IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header Host $http\_host;

proxy\_pass http://localhost:8093;

}

location /ies-batch {

proxy\_set\_header X-Real\_IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header Host $http\_host;

proxy\_pass http://localhost:8098;

}

location /irs {

proxy\_set\_header X-Real\_IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header Host $http\_host;

proxy\_pass http://localhost:8092;

}

location /config {

proxy\_set\_header X-Real\_IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header Host $http\_host;

proxy\_pass http://localhost:5000;

}

location /ads {

proxy\_set\_header X-Real\_IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header Host $http\_host;

proxy\_pass http://localhost:8088;

}

location /ire {

proxy\_set\_header X-Real\_IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header Host $http\_host;

proxy\_pass http://localhost:8094;

}

location /sms-operation {

proxy\_set\_header X-Real\_IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header Host $http\_host;

proxy\_pass http://localhost:8095;

}

location /ivl-mdm {

proxy\_set\_header X-Real\_IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header Host $http\_host;

proxy\_pass http://localhost:8097;

}

}

* To deploy these microservices in dev/qa servers we created a deployment script to manage deployments that pulls the latest Docker image from the GitLab Container Registry using the appropriate tag (e.g., latest or specific commit ID) and deploys the applications.

#!/bin/bash

########################################

# Put this on a Server

# run chmod +x deploy\_app.sh to make the script executable

# Execute this script: ./deploy\_app.sh appname:$TAG

# Replace the $TAG with the actual Build Tag you want to deploy

########################################

set -e

SERVICE\_NAME=$1

BACKEND\_REGISTRY="registry.gitlab.com/infovision2023/jsw/common-services/backend"

FRONTEND\_REGISTRY="registry.gitlab.com/infovision2023/jsw/common-services/frontend"

# Check for arguments

if [[ $# -lt 1 ]] ; then

echo '[ERROR] You must enter service name to be deployed'

exit 1

fi

CONTAINER\_NAME=$SERVICE\_NAME

if [[ $SERVICE\_NAME == \*":"\* ]]; then

CONTAINER\_NAME=`echo $SERVICE\_NAME | cut -d ':' -f1`

fi

if [[ $CONTAINER\_NAME == "mdm-mfe" ]]; then

CI\_IMAGE="$FRONTEND\_REGISTRY/$SERVICE\_NAME"

elif [[ $CONTAINER\_NAME == "mdm-service" || $CONTAINER\_NAME == "aas-service" || $CONTAINER\_NAME == "ens-service" || $CONTAINER\_NAME == "ums-service" || $CONTAINER\_NAME == "ies-service" || $CONTAINER\_NAME == "irs-service" || $CONTAINER\_NAME == "spring-cloud-config-service" || $CONTAINER\_NAME == "ire-service" || $CONTAINER\_NAME == "sms-operation-service" ]]; then

CI\_IMAGE="$BACKEND\_REGISTRY/$SERVICE\_NAME"

else

echo '[ERROR] You must enter a valid service name to be deployed [ mdm-mfe | mdm-service | aas-service | ens-service | ums-service | ies-service | irs-service | spring-cloud-config-service | ire-service | sms-operation-service ]'

exit 1

fi

echo "Deploying $CONTAINER\_NAME to Docker Container"

#Check for running container & stop it before starting a new one

if [ $(docker inspect -f '{{.State.Running}}' "$CONTAINER\_NAME") = "true" ]; then

docker stop $CONTAINER\_NAME

fi

# Pull the latest image

echo "Pulling latest image for $CI\_IMAGE"

docker pull $CI\_IMAGE

echo "Starting $CONTAINER\_NAME using Docker Image name: $CI\_IMAGE"

if [[ $CONTAINER\_NAME == "mdm-service" ]]; then

docker run -d --rm=true --network host -e "SPRING\_PROFILES\_ACTIVE=dev" --env-file /root/.devops/dev.list -v /jsw/nextgen-mes/logs/$CONTAINER\_NAME:/var/log -v /jsw/nextgen-mes/sms/sops:/jsw/nextgen-mes/sms/sops --name $CONTAINER\_NAME $CI\_IMAGE

elif [[ $CONTAINER\_NAME == "irs-service" ]]; then

docker run -d --rm=true --network host -e "SPRING\_PROFILES\_ACTIVE=dev" --env-file /root/.devops/dev.list -v /jsw/nextgen-mes/logs/$CONTAINER\_NAME:/var/log -v /jsw/nextgen-mes/reports:/var/reports --name $CONTAINER\_NAME $CI\_IMAGE

else

if [[ $CONTAINER\_NAME != "mdm-mfe" ]]; then

docker run -d --rm=true --network host -e "SPRING\_PROFILES\_ACTIVE=dev" --env-file /root/.devops/dev.list -v /jsw/nextgen-mes/logs/$CONTAINER\_NAME:/var/log -e JAVA\_TOOL\_OPTIONS="-Xms500M -Xmx500M" --name $CONTAINER\_NAME $CI\_IMAGE

else

docker run -d --rm=true --network host -e "SPRING\_PROFILES\_ACTIVE=dev" --env-file /root/.devops/dev.list --name $CONTAINER\_NAME $CI\_IMAGE

fi

fi

# Show the running processes

docker ps

echo " "

echo "#######################################################################################################################"

echo "#######################################################################################################################"

echo " "

* If a new micro service comes up and wants to be deployed, we need to add that service name to this deployment script. That simplifies the deployment process.
* Unless you want to create a mess on the server, first modify the script in GitLab at the following path: jsw/devops/deploy\_scripts/deploy\_app\_dev.sh. Later, you can download the script from GitLab to the server. Each environment has a separate script in this directory, so modify the script according to the specific environment.
* Add any required variables to the property file in GitLab at the following path: jsw/devops/environment\_files/dev.list. Later, you can download the script from GitLab to the server. Each environment has a separate script in this directory, so modify the script according to the specific environment.
* Now, to download the scripts from GitLab to the server, we have written an environment script which is:

#!/bin/bash

# GitLab API URL for the first file

api\_url1="https://gitlab.com/api/v4/projects/infovision2023%2Fdevops/repository/files/environment\_files%2Fdev.list/raw?ref=main"

# Specify the filename for the first file

filename1="dev.list"

# GitLab API URL for the second file

api\_url2="https://gitlab.com/api/v4/projects/infovision2023%2Fdevops/repository/files/deploy\_scripts%2Fdeploy\_app\_dev.sh/raw?ref=main"

# Specify the filename for the second file

filename2="deploy\_app\_dev.sh"

# GitLab Access Token (replace 'YOUR\_ACCESS\_TOKEN' with your actual token)

access\_token="glpat-ycSpgh1UCpLsjfQwFVpb"

# Download the first file using curl with authentication

curl -o "$filename1" --header "PRIVATE-TOKEN: $access\_token" "$api\_url1"

# Download the second file using curl with authentication

curl -o "$filename2" --header "PRIVATE-TOKEN: $access\_token" "$api\_url2"

* Copy this environment script into the server at /root/.devops path and execute it there. This script will download the necessary files, helping to ease the process.
* Now create a .bash\_rc file and under that file write the below line as:

alias deploy='/root/.devops/deploy\_app\_dev.sh'

Adding an alias to your .bashrc file is a way to create shortcuts for frequently used commands. When you write alias deploy='/root/.devops/deploy\_app\_dev.sh' in your .bashrc, you're defining a new command (deploy) that runs the script located at /root/.devops/deploy\_app\_dev.sh whenever you type deploy in the terminal.

* Instead of typing the full path to the script every time, you can simply type deploy.

This saves time and reduces the chance of making a typo.

* If the microservice is not frequently used and needs a one-time deployment or deploys twice a month, we can directly deploy it using the foreground method command. i,e.

docker run -it --rm=true --network host -e "SPRING\_PROFILES\_ACTIVE=dev" --env-file dev.list --name aas-service -v /jsw/nextgen-mes/logs:/var/log registry.gitlab.com/infovision2023/jsw/common-services/backend/aas-service:milestone-3-latest

* Use docker run -d for detachable mode
* To map the volume to all microservices, first create a file path in host directory to store the logs i.e; /jsw/nextgen-mes/logs
* Navigate to gitlab->backend->spring-services-config-library->mdm.yml

Add a logging file path in yaml file

logging:

level:

io:

lettuce:

core:

protocol:

ConnectionWatchdog: OFF

file:

path: /var/log/mdm

This configuration sets the logging levels for different components of the application and specifies that log messages should be written to the file located at /var/log/mdm.

Modify the dockerfile in all services in milestone branch

FROM openjdk:17-oracle

# Add the JAR file to the container

ADD build/libs/mdm-service-0.0.1-SNAPSHOT.jar mdm-service.jar

# Set the log directory inside the container

ENV LOG\_DIR=/var/log

# Create the log directory

RUN mkdir -p $LOG\_DIR

# Set the default command to run your application

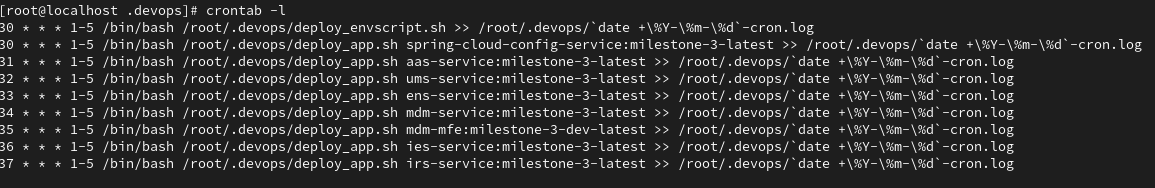
CMD ["java", "-jar", "mdm-service.jar"]

* This Docker file builds a Docker image that includes the necessary environment to run a Java application and sets up a directory for log files. When a container is started from this image, it automatically runs the Java application specified by the JAR file
* After modifying the docker file the pipeline will trigger automatically and build the pipeline that build the image and push that to container registry.
* Before deploying the mdm-service, Modify the deployment script by adding the volume argument in docker run command.
* Attach the volume which is the path of host directory /jsw/nextgen-mes/logs

if [[ $CONTAINER\_NAME == "mdm-service" ]]; then

docker run -d --rm=true --network host -e "SPRING\_PROFILES\_ACTIVE=dev" --env-file /root/.devops/dev.list -v /jsw/nextgen-mes/logs/$CONTAINER\_NAME:/var/log -v /jsw/nextgen-mes/sms/sops:/jsw/nextgen-mes/sms/sops --name $CONTAINER\_NAME $CI\_IMAGE

* We initiated a cron tab which deploys the services for every hour.



* To validate the applications is running or not we check through the browser for both dev and qa environments.