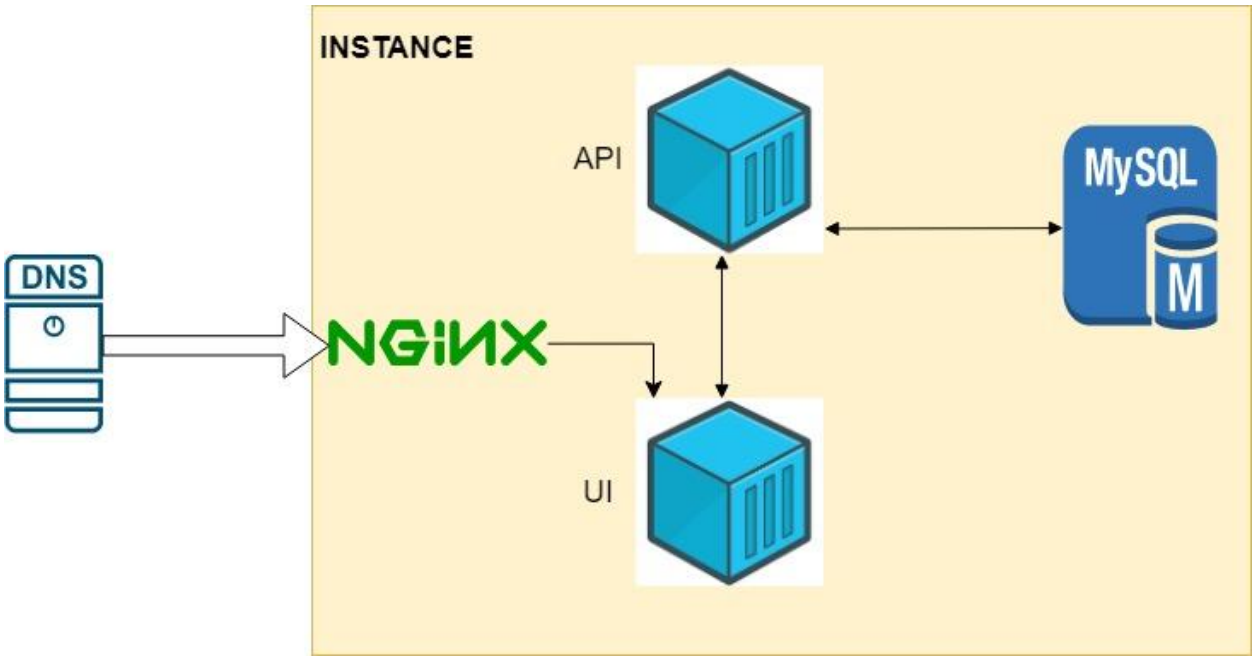
Milagro qa1 qa2 qa3-mig Autodeployment

Previously we were using qa, N1, N2 servers. But now it is renamed to qa1, qa2, qa3.

qa = qa1
N1= qa2
N2 = qa3-mig

Here we will be using single pipeline for all three servers. We have created qa1, qa2, qa3-mig branches in both ui and api. All three environments have same server architecture as below:



- Lets start with domain names of these servers:
    1. qa1 :- *.milagro.ca, api.milagro.ca
    2. qa2 :- qa2api.milagro.ca, qa2tenant.milagro.ca, qa2mbm.milagro.ca
    3. qa3-mig :- mig.milagro.ca, migapi.milagro.ca, migmbm.milagro.ca
- In the architecture there are two docker containers running. One is for ui and one for api.
- UI is published on 4000 port and api is on 3000.
- Nginx and mysql server is not running in docker container. It is installed on host.

Step-1 : creating 3 servers

For creating servers follow below steps:

1. Log in into aws account.
2. Search for aws lightsail servic. Once you open service you will find option ‘create instance’ . click on it.

3. After step select the appropriate specification and like os, instance plan, ssh key pair, then launch instance.
4. Make sure that 22, 80, and 443 ports are open.

Follow same steps for all three servers.

Step-2: Nginx configuration

To install nginx on server use below commands:
#apt-get update
#apt-get install nginx

Once installation is complete check whether nginx is active with below command:
#systemctl status nginx

For qa1 :
There are domains *.milagro.ca, api.milagro.ca.
For each domain nginx configuration is needed in /etc/nginx/sites-enabled/ .

#cd /etc/nginx/sites-enabled/
#touch ui-qa1-milagro
#touch api-qa1-milagro

Then using vim add below content to ui-qa1-milagro.

```
server {

        server_name *.milagro.ca;
        gzip on;


        location / {
        proxy_pass http://localhost:4000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
#        add_header Access-Control-Allow-Origin *;
        client_max_body_size 16384m;

                }


}
```

Similarly , add below content to api-qa1-milagro

```
server {

        server_name api.milagro.ca;
        gzip on;

        location / {
        proxy_pass http://localhost:3000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
#        add_header Access-Control-Allow-Origin *;
        client_max_body_size 16384m;

         }

}
```

For qa2 :
There are three domains for qa2.
qa2tenant.milagro.ca, qa2mbm.milagro.ca, qa2api.milagro.ca

So
#cd /etc/nginx/sites-enabled/
#touch ui1-qa2-milagro
#touch ui2-qa2-milagro
#touch api-qa2-milagro

This way three nginx configuration files are created.

ui1-qa2-milagro for qa2tenant.milagro.ca
ui2-qa2-milagro for qa2mbm.milagro.ca
api-qa2-milagro for qa2api.milagro.ca

Add below content to ui1-qa2-milagro

```
server {
        server_name qa2tenant.milagro.ca;
        gzip on;
        location / {
```

```
        proxy_pass http://localhost:4000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
#        add_header Access-Control-Allow-Origin *;
        client_max_body_size 16384m;

                }
}
```

Similarly add below content to ui2-qa2-milagro

```
server {
        server_name qa2mbm.milagro.ca;
        gzip on;
        location / {
        proxy_pass http://localhost:4000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
#        add_header Access-Control-Allow-Origin *;
        client_max_body_size 16384m;

                }

}
```

Then add below content to api-qa2-milagro

```
server {
        server_name qa2api.milagro.ca;
        gzip on;

        location / {
        proxy_pass http://localhost:3000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
#        add_header Access-Control-Allow-Origin *;
        client_max_body_size 16384m;

                }

}
```

Milagro qa1 qa2 qa3-mig Autodeployment

For qa3-mig :

There are three domains for qa3-mig.

mig.milagro.ca, migapi.milagro.ca, migmbm.milagro.ca

So now,

```
#cd /etc/nginx/sites-enabled/
#touch mig.milagro.ca
#touch migapi.milagro.ca
#touch migmbm.milagro.ca
```

ui1-qa3-milagro for mig.milagro.ca
ui2-qa3-milagro for migapi.milagro.ca
api-qa3-milagro for migmbm.milagro.ca

Add below content to ui1-qa3-milagro

```
server {
        server_name mig.milagro.ca;
        gzip on;


        location / {
        proxy_pass http://localhost:4000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
#        add_header Access-Control-Allow-Origin *;
        client_max_body_size 16384m;

                }

}
```

Add below content to ui2-qa3-milagro

```
server {
        server_name migmbm.milagro.ca;
        gzip on;


        location / {
        proxy_pass http://localhost:4000;
        proxy_set_header Host $host;
```

```
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
#       add_header Access-Control-Allow-Origin *;
        client_max_body_size 16384m;
          }

}
```

Add below content to api-qa3-milagro

```
server {
        server_name migapi.milagro.ca;
        gzip on;

        location / {
        proxy_pass http://localhost:3000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
#       add_header Access-Control-Allow-Origin *;
        client_max_body_size 16384m;

                }

}
```

Step-2: SSL implementation

- As all nginx configuration is done, to apply ssl to all domains, we need certbot installed in the server.
- To install cerbot use below link: **https://www.freecodecamp.org/news/the-nginx-handbook/#how-to-configure-ssl-and-http-2**
- Once cerbot is installed use below commands to apply SSL:
  #sudo certbot --nginx -d api.milagro.ca —for subdomians
  #sudo certbot --server https://acme-v02.api.letsencrypt.org/directory -d *.koders.xyz --manual --preferred-challenges dns-01 certonly —for wild domains

- Using above methods ssl can be applied on domains.

Step-3: docker, mysql 5.7 and gitlab-runner installation

Milagro qa1 qa2 qa3-mig Autodeployment

- For installing docker: https://docs.docker.com/engine/install/ubuntu/
- For installing mysql 5.7: **https://www.devart.com/dbforge/mysql/how-to-install-mysql-on-ubuntu/**
- For installing gitlab-runner:

#curl -L "https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh" | sudo bash
#sudo apt install gitlab-runner

Step-4: Configuring gitlab-runner on servers

1. Each environment have two jobs:
    a. Build of docker image, pushing it to repository
    b. Running same image to deploy application.
2. Both UI and API will have above two jobs.
3. So, Every environment will have two runners each for ui and api.

Tags given to each Environment is as below:

For qa1:

- ui-qa1-milagro
- api-qa1-milagro

For qa2:

- ui-qa2-milagro
- api-qa2-milagro

For qa3-mig:

- ui-qa3mig-milagro
- Api-qa3mig-milagro

Like above tags will be given.
To register runner:

- Gitlab runner register
    - To register gitlab runner
    - Run '**#sudo gitlab-runner register**' on the server then go to this link (depending upon the project) https://gitlab.vivantcorp.com/milagro/milagro-ui/-/settings/ci_cd

- Fill in the requirement gitlab instance URL, token, Description, tag, Executor (Ubuntu:latest)
- Once it is configured you can see the runner in Runners (https://gitlab.vivantcorp.com/utiliko/utiliko-api/-/settings/ci_cd)
- Give tags like discussed above.

Now runners are registered.
Now open /etc/gitlab-runner/config.toml and make sure below lines are there;

```
privileged = true
volumes = ["/cache", "/var/run/docker.sock:/var/run/docker.sock"]
concurrent = 2
```

concurrent = 2 , it means that at a time two jobs can be run.

Step-4: About .gitlab-ci.yml

Here is .gitlab-ci.yml which is used to define jobs to be run on server.
For all the three server have common cicd file, which will have total six jobs.
For each Environment two jobs are required:

1. Build docker image
2. Running docker image on desired port.

UI will be running on 4000 and api will be on 3000.

```
image: "docker:19.03.15"
services:
  - docker:19.03.15-dind
.only-qa1:
  only:
    - qa1
.only-qa2:
  only:
    - qa2
.only-qa3-mig:
  only:
    - qa3-mig
```

This GitLab CI snippet defines a Docker image and specifies a Docker service to use in a CI/CD pipeline. The image being used is "docker:19.03.15", which is likely a specific version of the Docker software that is needed for the pipeline.

Milagro qa1 qa2 qa3-mig Autodeployment

The "docker:19.03.15-dind" service is specified, which stands for "Docker in Docker". This allows the pipeline to spin up Docker containers inside of a Docker container, which can be useful for testing or building Docker images.

There are also three "only" directives, which limit the job to run only for specific branches or tags that match the specified criteria. Specifically, the job will only run on branches or tags that are labeled as "qa1", "qa2", or "qa3-mig", respectively. This can help to ensure that the job is only run in specific environments or stages of development or testing.

```
stages:
  - build
  - deploy
```

"Build" is a common stage in CI/CD pipelines where the source code is compiled, dependencies are installed, tests are run, and artifacts are generated. This stage is responsible for creating the application package or binary that can be deployed to a desired environment.

"Deploy" is the stage where the application is deployed to a production environment or any other environment as specified. This stage can involve various tasks, such as uploading the application artifacts to a cloud platform, deploying containers to Kubernetes or other container orchestration systems, or pushing code to servers.

```
qa1-build:
  variables:
    ENV_TYPE: qa
  extends: .only-qa1
  stage: build
  tags:
    - milagro-ui-qa-1
  script:
    - docker system prune -a --volumes -f
    - docker login "$CI_REGISTRY_URL" -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD"
    - docker build --pull -t "${CONTAINER_IMAGE}-v2-${ENV_TYPE}" --build-arg ENV_TYPE=${ENV_TYPE} .
    - docker tag "$CONTAINER_IMAGE-v2-${ENV_TYPE}" "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:${CI_PIPELINE_ID}"
    - docker push "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:${CI_PIPELINE_ID}"
    - docker tag "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:${CI_PIPELINE_ID}" "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"
    - docker push "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"
```

Note: Please zoom and understand the pipeline code. For above code font is kept small to keep longest command in one line.

This is a GitLab CI/CD pipeline configuration snippet for building a Docker container image in a QA environment. Here is what each section of the snippet does:

qa1-build: This is the name of the job that will be created in the GitLab CI/CD pipeline.

Milagro qa1 qa2 qa3-mig Autodeployment

variables: This section defines an environment variable called ENV_TYPE and sets its value to "qa". This variable will be used later in the script to set the container image tag. This variable is also passed to dockerfile to deploy qa environments.

extends: This section allows this job to inherit the configuration defined in another job that is labeled with the .only-qa1 tag.

stage: This section specifies the stage of the pipeline in which this job will run. In this case, it is set to "build".

tags: This section specifies the runner tags that are required to run this job.

script: This section contains the commands that will be executed when this job runs. The commands in this script perform the following steps:

a. docker system prune: This command cleans up any unused Docker containers, images, and volumes.

b. docker login: This command logs in to the GitLab container registry using the CI_REGISTRY_USER and CI_REGISTRY_PASSWORD variables.

c. docker build: This command builds the Docker image using the Dockerfile located in the current directory. The --pull option is used to ensure that the latest version of the base image is pulled. The --build-arg option is used to set the ENV_TYPE argument to "qa".

d. docker tag: This command tags the newly created image with a unique tag using the CI_PIPELINE_ID variable.

e. docker push: This command pushes the newly created image to the GitLab container registry. Two tags are pushed for the image: the unique tag generated using the CI_PIPELINE_ID variable and a tag called "latest".

```
qa2-build:
  variables:
    ENV_TYPE: n1
  extends: .only-qa2
  stage: build
  tags:
    - milagro-ui-qa-2
  script:
    - docker system prune -a --volumes -f
    - docker login "$CI_REGISTRY_URL" -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD"
    - docker build --pull -t "${CONTAINER_IMAGE}-v2-${ENV_TYPE}" --build-arg ENV_TYPE=${ENV_TYPE} .
    - docker tag "$CONTAINER_IMAGE-v2-${ENV_TYPE}" "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:${CI_PIPELINE_ID}"
    - docker push "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:${CI_PIPELINE_ID}"
    - docker tag "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:${CI_PIPELINE_ID}" "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"
    - docker push "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"
```

Milagro qa1 qa2 qa3-mig Autodeployment

This is another GitLab CI/CD pipeline configuration snippet for building a Docker container image, but this time it is for the QA2 and environment n1. Here is what each section of the snippet does:

qa2-build: This is the name of the job that will be created in the GitLab CI/CD pipeline.

variables: This section defines an environment variable called ENV_TYPE and sets its value to "n1". This variable will be used later in the script to set the container image tag.

extends: This section allows this job to inherit the configuration defined in another job that is labeled with the .only-qa2 tag.

stage: This section specifies the stage of the pipeline in which this job will run. In this case, it is set to "build".

tags: This section specifies the runner tags that are required to run this job.

script: This section contains the commands that will be executed when this job runs. The commands in this script perform the same steps as the previous example, including cleaning up any unused Docker containers, logging in to the GitLab container registry, building the Docker image, tagging the image with a unique tag and "latest", and pushing the image to the registry.

Note that both snippets are very similar, but they differ in the values assigned to the ENV_TYPE variable, the name of the job, and the runner tags. This is because they represent different environments (QA1 and QA2) and they require different runner tags to run.

```
qa3-mig-build:
  variables:
    ENV_TYPE: n2
  extends: .only-qa3-mig
  stage: build
  tags:
    - qa3-mig-ui-milagro
  script:
    - docker system prune -a --volumes -f
    - docker login "$CI_REGISTRY_URL" -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD"
    - docker build --pull -t "${CONTAINER_IMAGE}-v2-${ENV_TYPE}" --build-arg ENV_TYPE=${ENV_TYPE} .
    - docker tag "$CONTAINER_IMAGE-v2-${ENV_TYPE}" "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:${CI_PIPELINE_ID}"
    - docker push "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:${CI_PIPELINE_ID}"
    - docker tag "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:${CI_PIPELINE_ID}" "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"
    - docker push "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"
```

This is another GitLab CI/CD pipeline configuration snippet for building a Docker container image, but this time it is for the n2 environment in the qa3-mig branch. Here is what each section of the snippet does:

Milagro qa1 qa2 qa3-mig Autodeployment

qa3-mig-build: This is the name of the job that will be created in the GitLab CI/CD pipeline.

variables: This section defines an environment variable called ENV_TYPE and sets its value to "n2". This variable will be used later in the script to set the container image tag.

extends: This section allows this job to inherit the configuration defined in another job that is labeled with the .only-qa3-mig tag.

stage: This section specifies the stage of the pipeline in which this job will run. In this case, it is set to "build".

tags: This section specifies the runner tags that are required to run this job.

script: This section contains the commands that will be executed when this job runs. The commands in this script perform the same steps as the previous examples, including cleaning up any unused Docker containers, logging in to the GitLab container registry, building the Docker image, tagging the image with a unique tag and "latest", and pushing the image to the registry.

Note that this snippet is very similar to the previous examples, but it differs in the value assigned to the ENV_TYPE variable, the name of the job, and the runner tags. This is because it represents a different environment and it requires a different runner tag to run.

```
qa1-deploy:
  variables:
    ENV_TYPE: qa
  extends: .only-qa1
  stage: deploy
  tags:
    - milagro-ui-qa-1
  script:
    - chmod +x ./ui.sh
    - ./ui.sh &
    - docker login "$CI_REGISTRY_URL" -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD"
    - docker pull "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"
    - docker rm $(docker stop $(docker ps -a -q --filter ancestor="$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest" --format="{{.ID}}")) &
    - docker run -p 4000:80 -d "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"
```

This GitLab CI snippet specifies a job named qa1-deploy that is responsible for deploying the container image built in the qa1-build job. The variables section specifies an environment variable ENV_TYPE with the value qa.

The job extends the .only-qa1 job configuration, which restricts the job to only run on the qa1 branch. The stage parameter is set to deploy, indicating that this job will run in the deployment stage.

Milagro qa1 qa2 qa3-mig Autodeployment

The tags parameter specifies the runner tag for this job, which is milagro-ui-qa-1.
The script section contains a series of shell commands to be executed by the runner.
The first command makes the ui.sh script executable, and the second command runs the script in the background.

The next three commands are responsible for pulling the Docker image from the GitLab container registry, stopping the current running container, and running the new container.
Specifically, the docker login command logs the runner into the GitLab container registry, the docker pull command pulls the latest version of the container image, and the docker rm and docker run commands stop and start a container with the updated image, respectively.

Overall, the qa1-deploy job ensures that the latest version of the container image is pulled from the GitLab container registry and deployed on the server where milagro-ui-qa-1 runner is running.

```
qa2-deploy:
  variables:
    ENV_TYPE: n1
  extends: .only-qa2
  stage: deploy
  tags:
    - milagro-ui-qa-2
  script:
    - chmod +x ./ui.sh
    - ./ui.sh &
    - docker login "$CI_REGISTRY_URL" -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD"
    - docker pull "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"
    - docker rm $(docker stop $(docker ps -a -q --filter ancestor="$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest" --format="{{.ID}}")) &
    - docker run -p 4000:80 -d "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"
```

This GitLab CI script is responsible for deploying the application that was built in the qa2-build stage.

It specifies the deployment environment type as n1 and uses the only-qa2 job template, which defines the GitLab Runner tags that should be used to run the job.

The script starts by setting the execution permission for a script called ui.sh and running it in the background.

Then, it logs into the Docker registry specified in the GitLab CI/CD settings using the credentials stored in the GitLab CI/CD variables.

Next, it pulls the Docker image that was pushed to the registry in the previous qa2-build stage using the latest tag.

After that, it stops and removes any existing containers that are based on the image being deployed, and runs a new container with port 4000 mapped to port 80 of the container, exposing the application to the internet.

```
qa3-mig-deploy:
  variables:
    ENV_TYPE: n2
  extends: .only-mig
  stage: deploy
  tags:
    - qa3-mig-ui-milagro
  script:
    - chmod +x ./ui.sh
    - ./ui.sh &
    - docker login "$CI_REGISTRY_URL" -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD"
    - docker pull "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"
    - docker rm $(docker stop $(docker ps -a -q --filter ancestor="$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest" --format="{{.ID}}")) &
    - docker run -p 4000:80 -d "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"
```

This is a deployment job for the qa3-mig-build stage, which means it will run after the qa3-mig-build job has completed successfully.

The variables section sets the environment type to n2, and the job will run only on runners with the qa3-mig-api-milagro tag.

The script section performs the following steps:

It makes the ui.sh script executable with chmod +x ./ui.sh.
It runs the ui.sh script in the background with ./ui.sh & to start the UI server.
It logs in to the container registry with docker login.
It pulls the latest version of the container image from the registry with docker pull.
It stops and removes any running containers based on the same image with docker rm $(docker stop $(docker ps -a -q --filter ancestor="$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest" --format="{{.ID}}")) &.
It runs a new container based on the latest image, mapping port 4000 on the host to port 80 in the container with docker run -p 4000:80 -d "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest".

The complete pipeline code will be look like below:

Milagro qa1 qa2 qa3-mig Autodeployment

```
image: "docker:19.03.15"
services:
  - docker:19.03.15-dind
.only-qa1:
  only:
    - qa1
.only-qa2:
  only:
    - qa2
.only-qa3-mig:
  only:
    - qa3-mig
stages:
  - build
  - deploy

qa1-build:
  variables:
     ENV_TYPE: qa
  extends: .only-qa1
  stage: build
  timeout: 3 hours
  tags:
    - milagro-ui-qa-1
  script:
    - docker system prune -a --volumes -f
    - docker login "$CI_REGISTRY_URL" -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD"
    - docker build --pull -t "${CONTAINER_IMAGE}-v2-${ENV_TYPE}" --build-arg ENV_TYPE=${ENV_TYPE} .
    - docker tag "$CONTAINER_IMAGE-v2-${ENV_TYPE}" "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:${CI_PIPELINE_ID}"
    - docker push "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:${CI_PIPELINE_ID}"
    - docker tag "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:${CI_PIPELINE_ID}" "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"
    - docker push "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"

qa2-build:
  variables:
     ENV_TYPE: n1
  extends: .only-qa2
  stage: build
  timeout: 3 hours
  tags:
    - milagro-ui-qa-2
  script:
    - docker system prune -a --volumes -f
    - docker login "$CI_REGISTRY_URL" -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD"
    - docker build --pull -t "${CONTAINER_IMAGE}-v2-${ENV_TYPE}" --build-arg ENV_TYPE=${ENV_TYPE} .
    - docker tag "$CONTAINER_IMAGE-v2-${ENV_TYPE}" "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:${CI_PIPELINE_ID}"
    - docker push "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:${CI_PIPELINE_ID}"
    - docker tag "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:${CI_PIPELINE_ID}" "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"
    - docker push "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"

qa3-mig-build:
  variables:
     ENV_TYPE: n2
  extends: .only-mig
  stage: build
  timeout: 3 hours
  tags:
    - qa3-mig-ui-milagro
  script:
    - docker system prune -a --volumes -f
    - docker login "$CI_REGISTRY_URL" -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD"
    - docker build --pull -t "${CONTAINER_IMAGE}-v2-${ENV_TYPE}" --build-arg ENV_TYPE=${ENV_TYPE} .
    - docker tag "$CONTAINER_IMAGE-v2-${ENV_TYPE}" "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:${CI_PIPELINE_ID}"
    - docker push "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:${CI_PIPELINE_ID}"
    - docker tag "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:${CI_PIPELINE_ID}" "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"
    - docker push "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"

qa1-deploy:
  variables:
     ENV_TYPE: qa
  extends: .only-qa1
  stage: deploy
  tags:
    - milagro-ui-qa-1
  script:
    - chmod +x ./ui.sh
    - ./ui.sh &
    - docker login "$CI_REGISTRY_URL" -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD"
    - docker pull "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"
    - docker rm $(docker stop $(docker ps -a -q --filter ancestor="$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest" --format="{{.ID}}")) &
    - docker run -p 4000:80 -d "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"

qa2-deploy:
  variables:
     ENV_TYPE: n1
  extends: .only-qa2
  stage: deploy
  tags:
    - milagro-ui-qa-2
  script:
    - chmod +x ./ui.sh
    - ./ui.sh &
    - docker login "$CI_REGISTRY_URL" -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD"
    - docker pull "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"
    - docker rm $(docker stop $(docker ps -a -q --filter ancestor="$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest" --format="{{.ID}}")) &
    - docker run -p 4000:80 -d "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"
```

Milagro qa1 qa2 qa3-mig Autodeployment

```
qa3-mig-deploy:
  variables:
    ENV_TYPE: n2
  extends: .only-qa3-mig
  stage: deploy
  tags:
    - qa3-mig-ui-milagro
  script:
    - chmod +x ./ui.sh
    - ./ui.sh &
    - docker login "$CI_REGISTRY_URL" -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD"
    - docker pull "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"
    - docker rm $(docker stop $(docker ps -a -q --filter ancestor="$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest" --format="{{.ID}}")) &
    - docker run -p 4000:80 -d "$CI_REGISTRY_URL/${CONTAINER_IMAGE}-v2-${ENV_TYPE}:latest"
```

The pipeline for API can be made with above explanation.

Milagro qa1 qa2 qa3-mig Autodeployment