Univerzitet u Sarajevu
Elektrotehnički fakultet
Predmet: Administracija računarskih mreža
Godina studija: 3 - ciklus I

Laboratorijska vježba 7 – DevOps (Gitlab Runners)

Cilj vježbe:

Upoznavanje studenata sa pravljenjem kontinualne protočne strukture za build, test, stage i deploy aplikacije.

Vježba je bazirana na: https://gitlab.com/kibrovic/gitlab-task/-/tree/arm?ref_type=heads

Prije izrade vježbe potrebno je:

1. Kreirati račun na GitLab-u.
2. Napraviti repozitori s fork-om Classroom reservation Website
3. Konfigurisati GitLabRunner na lokalnom računaru:
    1. U sklopu repozitorija otići na Settings → CI/CD → Runners
    2. Pratiti uputu za instalaciju GitLab Runner-a i registraciju istog.
        1. Omogućiti i grupne runner-e – potrebno je ostaviti podatke kartice radi sprečavanja mine-anja coin-ova.
        2. Kreirati svoj runner (docker+machine ili shell)
4. Kompletirati terraform-task

Zadaci:

1. Modificirati js/db.js da koristi environment varijable:
    a. MYSQL_USER
    b. MYSQL_PASSWORD
    c. MYSQL_DB_HOST
2. Kreirati Dockerfile za Node.js aplikaciju
3. Kreirati ECS definicije taska ecs/frontend-task.json
4. Kreirati ECS definiciju taska ecs/database-task.jso
5. Kreirati GitLab CI/CD pipeline

Detalji izrade:

1. Dockerfile

- Kreirati dockerfile za Node aplikaciju. Buildati image te ga deployati na docker hub. Pozivati isti u daljoj izradi:

```
FROM node:19-alpine

EXPOSE 8080

ADD . /app
WORKDIR /app/js
RUN npm ci

ENTRYPOINT ["node", "index.js"]
```

2. Iako su kreirani servisi i taskovi na prošloj vježbi, potrebno je proći kroz korake kreiranje servisa i taskova pomoću AWS console (opcionalno)

- Definisati task za bazu:

    - Koristiti MySql image
    - Mapirati portove 3306 na 3306
    - Postaviti environment varijable: MYSQL_DATABASE: DBWT19, MYSQL_USER, MYSQL_PASSWORD i MYSQL_ROOT_PASSWORD
    - Konfigurisati CPU/Memory Limits
    - Definisati placement constraint za deployanje unutar privatnog subneta
    - Koristiti LabRole
- Definisati frontend task
    - Koristiti deployani image
    - Mapirati kontejnere 80
    - Postaviti MYSQL_DB_HOST na privatnu ip adresu EC2 instances i MYSQL_USER i MYSQL_PASSWORD
    - Konfigurisati CPU/Memory Limits
    - Definisati placement constraint za deployanje unutar javnog subneta
    - Koristiti LabRole

3. Konfiguracija Gitlab CI/CS

- Kreirati ecs/frontend-task.json (kopirati konfiguraciju iz konzole iz prethodnog koraka):

```
{
 "containerDefinitions": [
  {
   "portMappings": [
    {
     "hostPort": 80,
     "protocol": "tcp",
     "containerPort": 8080
    }
   ],
   "cpu": 256,
   "environment": [
    {
     "name": "MYSQL_DB_HOST",
     "value": "${MYSQL_DB}"
    },
    {
     "name": "MYSQL_USER",
     "value": "${MYSQL_USER}"
    },
    {
     "name": "MYSQL_PASSWORD",
     "value": "${MYSQL_PASSWORD}"
    }
   ],
   "memory": 512,
   "image": "${REPOSITORY_URL}",
   "name": "frontend-task"
  }
  ],
  "placementConstraints": [
    {
      "type": "memberOf",
      "expression": "attribute:ecs.subnet-id in [${PUBLIC_SUBNET_ID}]"
    }
  ],
  "family": "frontend-task",
  "taskRoleArn": "${CI_AWS_TASK_EXECUTION_ROLE}",
  "executionRoleArn": "${CI_AWS_TASK_EXECUTION_ROLE}"
}
```

- Kreirati ecs/database-task.json:

```
{
  "containerDefinitions": [
    {
      "portMappings": [
        {
          "hostPort": 3306,
          "protocol": "tcp",
          "containerPort": 3306
        }
      ],
      "cpu": 256,
      "environment": [
        {
          "name": "MYSQL_DATABASE",
          "value": "DBWT19"
        },
        {
          "name": "MYSQL_USER",
          "value": "${MYSQL_USER}"
        },
        {
          "name": "MYSQL_PASSWORD",
          "value": "${MYSQL_PASSWORD}"
        },
        {
          "name": "MYSQL_ROOT_PASSWORD",
          "value": "${MYSQL_ROOT_PASSWORD}"
        }
      ],
      "memory": 512,
      "image": "mysql",
      "healthCheck": {
        "retries": 10,
        "command": [
          "CMD-SHELL",
          "mysqladmin ping -h localhost --password=${MYSQL_ROOT_PASSWORD}"
        ],
        "timeout": 10,
        "interval": 30,
        "startPeriod": 5
      },
      "name": "database"
    }
  ],
  "placementConstraints": [
    {
      "type": "memberOf",
      "expression": "attribute:ecs.subnet-id in [${PRIVATE_SUBNET_ID}]"
    }
  ],
  "family": "database-task",
  "taskRoleArn": "${CI_AWS_TASK_EXECUTION_ROLE}",
  "executionRoleArn": "${CI_AWS_TASK_EXECUTION_ROLE}"
}
```

- **Napomena: Unutar postojećih zamijenjene su konfiguracije s environment varijablama**
- Dodati GitLab tajne unutar Settings->CI/CD -> Variables (**pripaziti da ne bude protected)**:
  - AWS_ACCESS_KEY_ID
  - AWS_DEFAULT_REGION
  - AWS_SECRET_ACCESS_KEY
  - AWS_SESSION_TOKEN
  - DOCKER_PASSWORD
  - DOCKER_USER
  - MYSQL_PASSWORD
  - MYSQL_ROOT_PASSWORD
  - MYSQL_USER
- Kreirati .gitlab-ci.yml za konfiguraciju:
  - Definisati dva stage-a: build i deployment
  - Build treba da koristi Docker-in-Docker image
  - Deploy treba da koristi AWS optimized GitLab image
  - Build stage ima jedan job build_frontend:
    - Login na docker
    - Buildanje aplikacije
    - Publish na docker hub (u slučaju izmjena)
  - Deploy ima dva job-a deploy_frontend i deploy_database:
    - Zamijenjuju environment varijable unutar ecs/*.json template-a sa stvarnnim vrijednostima
    - Registruje novu definiciju task-a
    - Ažurira ECS servis s najnovijim definicijama taska
  - Svaki job ima svoje varijable definisane

- Po želji se mogu dodati pravila kada se pokreće pipeline
-

```
stages:
  - build
  - deploy

.docker: &docker
  image: docker:20.10-git
  services:
    - docker:20.10-dind
  before_script:
    - echo "Docker login..."
    - docker login -u $DOCKER_USER -p $DOCKER_PASSWORD

.build_docker_script: &build_docker_script
  <<: *docker
  script:
    - echo "Building image..."
    - docker build -t $REPOSITORY_URL:$CI_COMMIT_SHORT_SHA .
    - echo "Tagging image..."
    - docker tag $REPOSITORY_URL:$CI_COMMIT_SHORT_SHA $REPOSITORY_URL:latest
  after_script:
    - echo "Pushing image..."
    - docker push $REPOSITORY_URL:$CI_COMMIT_SHORT_SHA
    - docker push $REPOSITORY_URL:latest

.deploy_ecs_script: &deploy_ecs_script
  image: ${CI_TEMPLATE_REGISTRY_HOST}/gitlab-org/cloud-deploy/aws-ecs:latest
  script:
    - apt-get update -y && apt-get install -y gettext
    - envsubst < ecs/$CI_AWS_ECS_TASK_DEFINITION_FILE >
$CI_AWS_ECS_TASK_DEFINITION_FILE
    - echo "Registering new task definition..."
    - aws ecs register-task-definition --region "${AWS_DEFAULT_REGION}" --cli-input-json
file://$CI_AWS_ECS_TASK_DEFINITION_FILE
    - echo "Updating the service..."
    - aws ecs update-service --force-new-deployment --region "${AWS_DEFAULT_REGION}" --
cluster "${CI_AWS_ECS_CLUSTER}" --service "${CI_AWS_SERVICE}"  --task-definition "$
{CI_AWS_TASK_DEFINITION}" --desired-count 1


build_frontend:
  <<: *build_docker_script
  stage: build
  variables:
    REPOSITORY_URL: sbecirovic1/arm-frontend-2024

deploy_frontend:
  before_script:
    - export PUBLIC_SUBNET_ID="$(aws ec2 describe-subnets --filters
"Name=tag:Name,Values=Public" --query "Subnets[*].SubnetId" --output text)"
```

```
    - export MYSQL_DB="$(aws ec2 describe-instances --filters
"Name=tag:Name,Values=PrivateServer" --query 'Reservations[*].Instances[*].PrivateIpAddress' --
output text)"
    - export CI_AWS_TASK_EXECUTION_ROLE="$(aws iam get-role --role-name $
{CI_AWS_TASK_EXECUTION_ROLE_NAME} --query Role.Arn --output text)"
  <<: *deploy_ecs_script
  stage: deploy
  variables:
    REPOSITORY_URL: sbecirovic1/arm-frontend-2024
    CI_AWS_ECS_CLUSTER: arm_ecs_cluster
    CI_AWS_SERVICE: frontend-service
    CI_AWS_ECS_TASK_DEFINITION_FILE: frontend-task.json
    CI_AWS_TASK_DEFINITION: frontend-task
    CI_AWS_TASK_EXECUTION_ROLE_NAME: LabRole

deploy_database:
  <<: *deploy_ecs_script
  before_script:
    - export PRIVATE_SUBNET_ID="$(aws ec2 describe-subnets --filters
"Name=tag:Name,Values=Private" --query "Subnets[*].SubnetId" --output text)"
    - export CI_AWS_TASK_EXECUTION_ROLE="$(aws iam get-role --role-name $
{CI_AWS_TASK_EXECUTION_ROLE_NAME} --query Role.Arn --output text)"
    - echo "Scaling down database service..."
    - aws ecs update-service --region "${AWS_DEFAULT_REGION}" --cluster "$
{CI_AWS_ECS_CLUSTER}" --service "${CI_AWS_SERVICE}" --desired-count 0
  stage: deploy
  variables:
    REPOSITORY_URL: mysql
    CI_AWS_ECS_CLUSTER: arm_ecs_cluster
    CI_AWS_SERVICE: database-service
    CI_AWS_ECS_TASK_DEFINITION_FILE: database-task.json
    CI_AWS_TASK_DEFINITION: database-task
    CI_AWS_TASK_EXECUTION_ROLE_NAME: LabRole
    MYSQL_DATABASE: DBWT19
```

NAPOMENE:

- envsubst se koristi za zamjenu envirnment varijabli u template za stvarne vrijednosti
- aws-cli se koristi za dobivanje varijabli:
  - aws ec2 describe subnets – za id public i private subnet-a
  - aws ec2 describe-instances – za dobivanje ip adrese privatne ec2 instances
  - aws iam get-role – za dobivanje role-name
- Za definisanje i ažuriranje taskova se koristi:
  - aws register-task-definition
  - aws update-service

Omogućivanje HTTPS:

Kako bi se omogućio https pristup stranici potrebno je uraditi sljedeće (analogno vježbi 5):

1. Promijeniti host port u frontend task-u sa 80 na 8080 i pushati izmjene na repozitorij.

2. Povezati se na javni server. **Napomena: Za povezivanje se koristi privatni ključ para koji je kreiran. Potrebno je isti dodati u ssh agent i onda uraditi povezivanje na javni server na AWS-u.**

3. Instalirati nginx: sudo amazon-linux-extras install -y nginx1

4. Omogućiti i startati isti:

```
sudo systemctl enable nginx
sudo systemctl start nginx
```

5. Generisati ključ (po potrebi kreirati odgovarajuće foldere):

```
 sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout
/etc/pki/nginx/private/server.key -out /etc/pki/nginx/server.crt
```

6. Urediti nginx konfiguraciju */etc/nginx/nginx.conf:*

```
user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;

include /usr/share/nginx/modules/*.conf;

events {
    worker_connections 1024;
}

http {
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile            on;
    tcp_nopush          on;
    tcp_nodelay         on;
    keepalive_timeout   65;
    types_hash_max_size 4096;

    include             /etc/nginx/mime.types;
    default_type        application/octet-stream;

    include /etc/nginx/conf.d/*.conf;

 server {
        listen       80;
        listen       [::]:80;
        server_name  ec2-54-235-225-199.compute-1.amazonaws.com;

        return 302 https://$server_name$request_uri;
    }

# Settings for a TLS enabled server.
#
    server {
        listen       443 ssl http2;
        listen       [::]:443 ssl http2;
        server_name  _;
        root         /usr/share/nginx/html;
#
        ssl_certificate "/etc/pki/nginx/server.crt";
        ssl_certificate_key "/etc/pki/nginx/private/server.key";
        ssl_session_cache shared:SSL:1m;
        ssl_session_timeout  10m;
        ssl_ciphers EECDH+AESGCM:EDH+AESGCM;
        ssl_prefer_server_ciphers on;
 location / {
        proxy_pass http://localhost:8080;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;

        }

    }

}
```

U sklopu konfiguracije omogućen omogućen je redirect s http na https. Pored toga pokrenut je reverse proxy koji proslijeđuje sve poslano na port 8080 na kojem se vrti aplikacija.

Napomena: Ovaj korak se mogao napraviti i kroz ecs zadatak.