DEPI project

DevOps Track — GIZ1_SWD1_S3d

Automated Deployment Pipeline with Jenkins and Docker Prepared By

Mohamed Waleed Mohamed Abdelfattah — 21044904
Mahmoud Tarek Medany — 21026143
Esraa Shaaban — 21032361
Taqwa Elsayed Mohammed Fahmy — 21025433
Magdy Ayman Abu Bakr — 21003567
Omnia magdy abdelnaby — 21027962

Supervised By
Mohamed Shokry

Documented in LaTeX By
Mahmoud Tarek Medany
Mohamed Waleed Mohamed Abdelfattah

2024

Contents

0.1 Project Flow Diagram

Failing at any stage in this flow leads to the end of the pipeline directly (failing email notification).

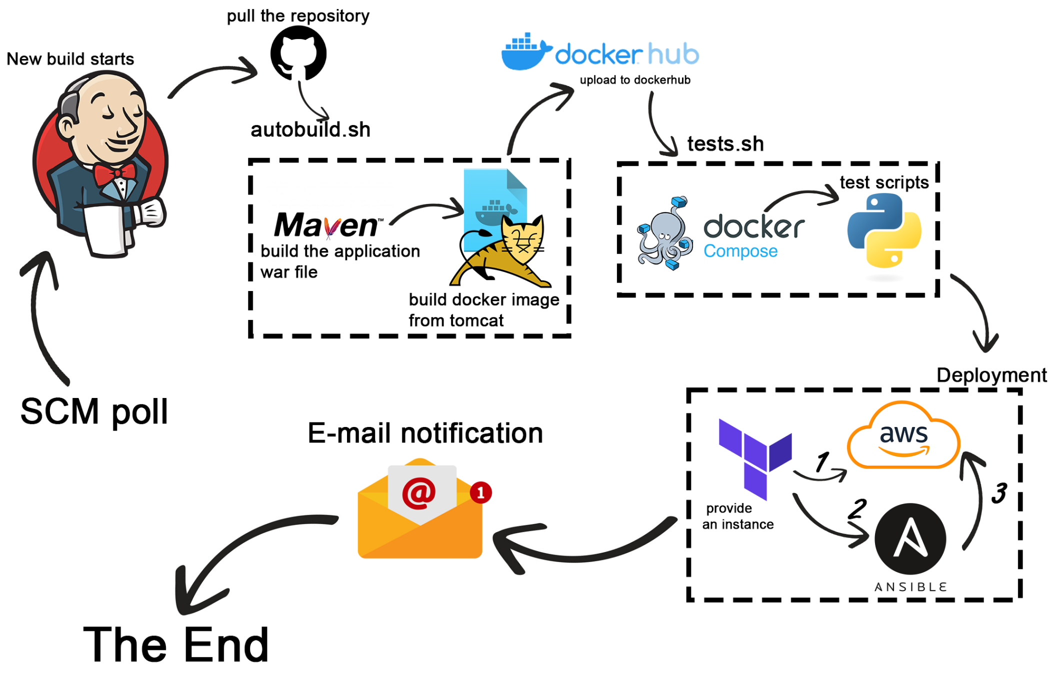


Figure 1: Project Flow diagram

1 Project Overview

The objective is to create an **automated CI/CD pipeline** for deploying a sample web application using **Jenkins, Docker**, and **Ansible**. The pipeline will handle continuous integration, automated testing, and deployment to a cloud environment.

1.1 Key Technologies

- **Jenkins**: Continuous integration and automation server to manage pipeline stages.
- **Docker**: Containerizing the application and managing its deployment.
- **Ansible**: Configuration management tool to automate deployment steps.
- **GitHub**: Source control for managing application code.
- **Cloud Environment (optional)**: AWS to host the final application.

1.2 Application

For the application we picked **Pet clinic** popular application. our colleague **Esraa Shaaban** volunteered to manage the **Repository**

1.3 Pipeline Workflow

1. **Code Build (Continuous Integration)**:
    - **Trigger**: Jenkins is triggered upon a new commit requests.
    - **Action**: Jenkins pulls the latest code from a Git repository and builds a Docker image.
    - **Testing**: Unit tests are run during the build process to ensure code quality.
    - **Notify**: Notification with the state of pipeline after it's done
2. **Push to Docker Registry**:
    - **Image Storage**: The Docker image is pushed to Docker Hub or a private registry for easy access and deployment.
3. **Automated Deployment (Continuous Delivery)**:
    - **Ansible Playbooks**: After the image is pushed, Ansible is used to deploy the application to a cloud server.
    - **Cloud Setup**: The playbooks configure and start the Docker container on the target cloud environment.
4. **Optional Kubernetes Integration**:
    - **Container Orchestration**: Kubernetes can be optionally configured to orchestrate the deployment of multiple Docker containers across nodes, ensuring scaling and load balancing.

1.4 Key Deliverables

- Jenkins and Docker setup with a basic working pipeline.
- Jenkins jobs for continuous integration (build, test, and Dockerize the app).
- Ansible playbooks for automated deployment.
- Documentation of the entire CI/CD pipeline process.

2 Dockerizing the application

This Section processes are done by Mohamed Waleed Mohamed

to ease the image building process I made a shell script

autobuild.sh:

```
#!/bin/bash

echo "MAVEN.."
mvn package

echo "running docker build .."
docker build -t depi_petclinic .
```

this shell script to be called in jenkins pipeline to build the image

2.1 Building the image

Dockerfile:

- -

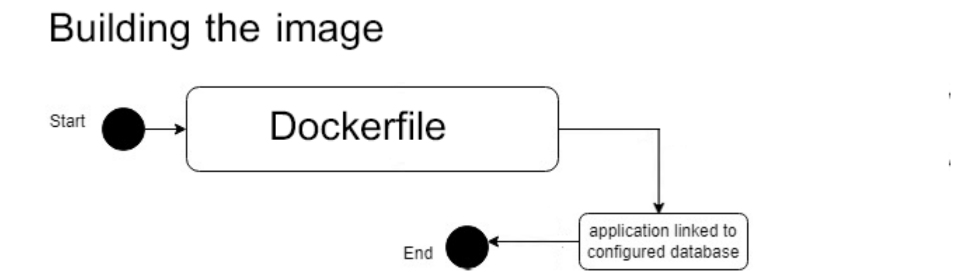
```
FROM tomcat:9.0.91

# copy the main WAR file
COPY ./target/spring-petclinic-2.3.1.BUILD-SNAPSHOT.war /usr/local/tomcat/webapps/ROOT.war

# expose port 8080
EXPOSE 8080

# run Tomcat for a short time and then stop it to initialize the application
RUN catalina.sh run & sleep 5 && catalina.sh stop


#
COPY ./properties_configuration_mw/application.properties

/usr/local/tomcat/webapps/ROOT/WEB-INF/classes/application.properties
#
COPY ./properties_configuration_mw/application-mysql.properties

/usr/local/tomcat/webapps/ROOT/WEB-INF/classes/application-mysql.properties

# start Tomcat
CMD ["catalina.sh", "run"]
```

- -

After running maven the building process starts.
To build the image it deploys the generated WAR file in tomcat container.
Dockerfile will try to access special database configurations if exists.

The configuration files are necessary for the build project to allow the image database to be linked to Mysql,Redis or whatever configured database. However they are not necessary to run the image itself

2.2 Running the image



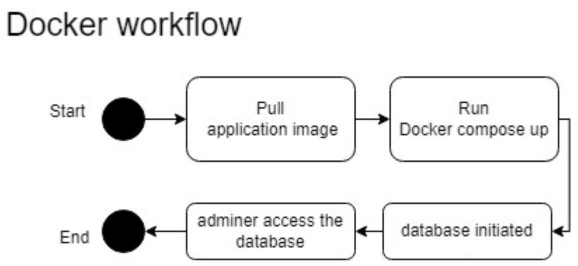


Figure 2: e.g.: using Ansible to run the image

After pushing the image to dockerhub we will have to configure docker-compose.yml file to manage the image ports and database volume (so data won't be lost)

docker-compose.yml:

```
services:
  mysql:
    image: mysql:5.7
    container_name: petsql
    ports:
      - "3306:3306"
    environment:
      - MYSQL_DATABASE=petclinic
      - MYSQL_ROOT_PASSWORD=
      - MYSQL_ALLOW_EMPTY_PASSWORD=true
      - MYSQL_USER=petclinic
      - MYSQL_PASSWORD=petclinic
      - MYSQL_DATABASE=petclinic
    volumes:
      - "./conf.d:/etc/mysql/conf.d:ro"  #was like this in documentaion
      - "./database_init_mw:/docker-entrypoint-initdb.d"
      - "./savedata_mw:/var/lib/mysql"

  adminer:
    image: adminer
    depends_on:
      - mysql
    container_name: petadminer
    ports:
      - "5051:8080"

  petclinic:
    image: mohamedwaleed77/depi_petclinic
    container_name: petapp
    depends_on:
      - mysql
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://mysql:3306/petclinic
      SPRING_DATASOURCE_USERNAME: petclinic
      SPRING_DATASOURCE_PASSWORD: petclinic
    ports:
      - "5050:8080"
```

mysql : The database, initiated in /savedata_mw directory and saved in /savedata_mw directory.
adminer : to acess the database manually.
petclinic : the application itself.

- **SPRING_DATASOURCE_URL**: Sets the database connection URL as `jdbc:mysql://mysql:3306/petclinic`, where:
  - `mysql` is the service name (Docker resolves this to the MySQL container's IP).
  - `3306` is the MySQL default port.
  - `petclinic` is the database name.
- **SPRING_DATASOURCE_USERNAME**: Specifies the username for the database connection as `petclinic`, matching the `MYSQL_USER` in the MySQL service.
- **SPRING_DATASOURCE_PASSWORD**: Specifies the password for the database connection as `petclinic`, matching the `MYSQL_PASSWORD` in the MySQL service.

After that the application will be available at URL: **http://localhost:5050/petclinic**

3 Testing

This Section processes are done by Mahmoud Tarek Medany

To make it easier for Jenkins team to test the application behaviour I wrote the following scripts.

3.1 Testing if application is deployed

isOn.py:

```
import requests

website_url = 'http://localhost:5050/petclinic'
response = requests.get(website_url)
print(f'The website {website_url} is online with status code: {response.status_code}')
```

3.2 Testing database

database.py:

```
import requests
import mysql.connector

# URL of the form
form_url = 'http://localhost:5050/petclinic/owners/new'

# Data to be filled in the form
data = {
    'firstName': 'abcdefgh',
    'lastName': 'ijklmnop',
    'address': 'email@example.com',
    'city': 'Cairo',
    'telephone': '12345',
    'submit': 'Submit'
}

# Create a session and submit the form
session = requests.Session()
response = session.post(form_url, data=data)

print('Form submitted successfully!')

# Database connection details
connection = mysql.connector.connect(
    host='localhost',
    database='petclinic',
    user='petclinic',
    password='petclinic'
)

# Create a cursor object
cursor = connection.cursor()

# Query to check if the entry exists
query = "SELECT * FROM owners WHERE first_name = %s AND last_name = %s"
cursor.execute(query, (data['firstName'], data['lastName']))

# Fetch all matching results
result = cursor.fetchall()

# Delete the entry
print('Entry found in the database:', result)
delete_query = "DELETE FROM owners WHERE first_name = %s AND last_name = %s"
cursor.execute(delete_query, (data['firstName'], data['lastName']))

# Commit the changes to remove the entry
connection.commit()
print('Entry removed from the database.')

# Close cursor and connection
cursor.close()
connection.close()
```

3.3 Shell script to automate testing and install dependinces

tests.sh:

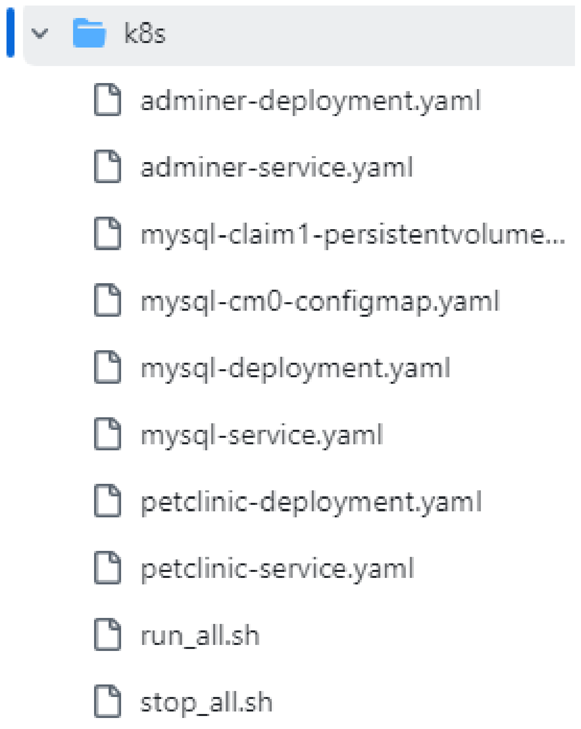
```
#!/bin/bash

apt install python3.6
apt-get install python3-pip
pip3 install mysql-connector-python
pip3 install requests

python3 isON.py
# Check if application deployed
if [ $? -ne 0 ]; then
    echo "Error in isON.py. Stopping execution."
    exit 1  #
fi

python3 database.py
# Check the database behaviour
if [ $? -ne 0 ]; then
    echo "Error in database.py. Stopping execution."
    exit 1
fi

echo "Test scripts executed successfully."
```

4 Kubernetes (kompose)

This Section processes are done by Mohamed Waleed

4.1 Kombose

Kombose is an open-source tool that translate a Docker Compose File to Kubernetes Resources
I used the docker-compose.yml to generate the following kubernetes resources



```
k8s
    adminer-deployment.yaml
    adminer-service.yaml
    mysql-claim1-persistentvolume...
    mysql-cm0-configmap.yaml
    mysql-deployment.yaml
    mysql-service.yaml
    petclinic-deployment.yaml
    petclinic-service.yaml
    run_all.sh
    stop_all.sh
```

Figure 3: Kubernetes Resources (shell scripts written manually)

However i had to edit *-service.yaml files by changing the service type to NodePort. As in Kubernetes it makes service accessible outside the cluster by exposing it on a specific port on each node. Also I wrote shell scripts to automate the process.

4.2 services

petclinic-service.yaml:

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kompose.cmd: kompose convert
    kompose.version: 1.34.0 (cbf2835db)
  labels:
    io.kompose.service: petclinic
  name: petclinic
spec:
  type: NodePort
  ports:
    - name: "5050"
      port: 5050
      targetPort: 8080
      nodePort: 30001
  selector:
    io.kompose.service: petclinic
```

mysql-service.yaml:

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kompose.cmd: kompose convert
    kompose.version: 1.34.0 (cbf2835db)
  labels:
    io.kompose.service: mysql
  name: mysql
spec:
  type: NodePort
  ports:
    - name: "3306"
      port: 3306
      targetPort: 3306
  selector:
    io.kompose.service: mysql
```

adminer-service.yaml:

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kompose.cmd: kompose convert
    kompose.version: 1.34.0 (cbf2835db)
  labels:
    io.kompose.service: adminer
  name: adminer
spec:
  type: NodePort
  ports:
    - name: "5051"
      port: 5051
      targetPort: 8080
      nodePort: 30002
  selector:
    io.kompose.service: adminer
```

4.3 shell scripts:

run_all.sh:

```
#!/bin/bash

# Deploy Kubernetes resources
kubectl apply -f mysql-cm0-configmap.yaml
kubectl apply -f mysql-claim1-persistentvolumeclaim.yaml
kubectl apply -f mysql-deployment.yaml
kubectl apply -f mysql-service.yaml
kubectl apply -f adminer-deployment.yaml
kubectl apply -f adminer-service.yaml
kubectl apply -f petclinic-deployment.yaml
kubectl apply -f petclinic-service.yaml

echo "All Kubernetes resources have been deployed."
```

stop_all.sh:

```
#!/bin/bash

# Delete Kubernetes resources
kubectl delete -f adminer-deployment.yaml
kubectl delete -f adminer-service.yaml
kubectl delete -f mysql-cm0-configmap.yaml
kubectl delete -f mysql-deployment.yaml
kubectl delete -f mysql-service.yaml
kubectl delete -f petclinic-deployment.yaml
kubectl delete -f petclinic-service.yaml

echo "All Kubernetes resources have been deleted."
```

NOTE:
I didn't delete mysql-claim1-persistentvolumeclaim.yaml **to prevent losing data.**

To access the results ((locally)):

Petclinic URL: MINIKUBE_IP:30001/petclinic

Adminer URL: MINIKUBE_IP:30002

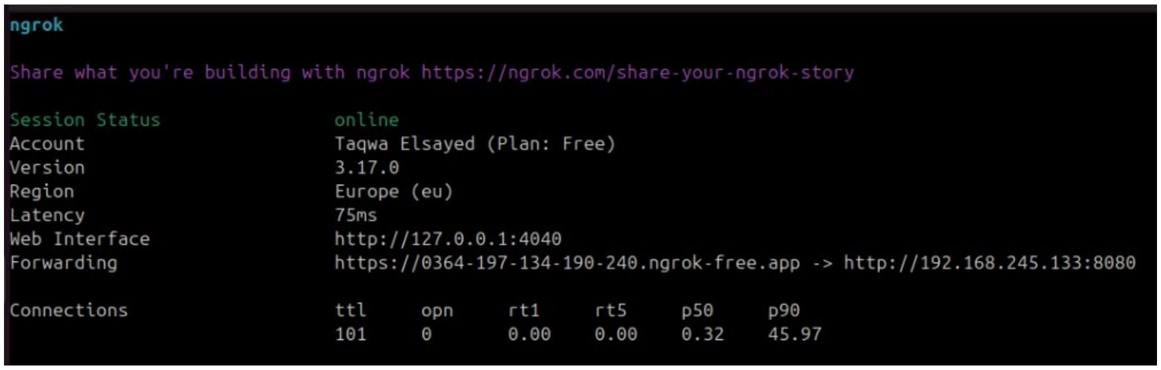
5 Jenkins: Webhooks to trigger Jenkins

This section processes are done by Taqwa Elsayed Mohammed Fahmy

5.1 Ngrok

ngrok is a tool that creates tunnels from local machine to the public internet. This allows to expose a local web server

In order to use webhooks I used ngrok to expose my local jenkins to the internet to receive webhook events.



```
ngrok

Share what you're building with ngrok https://ngrok.com/share-your-ngrok-story

Session Status                online
Account                       Taqwa Elsayed (Plan: Free)
Version                       3.17.0
Region                        Europe (eu)
Latency                       75ms
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://0364-197-134-190-240.ngrok-free.app -> http://192.168.245.133:8080

Connections                   ttl     opn     rt1     rt5     p50     p90
                              101     0       0.00    0.00    0.32    45.97
```

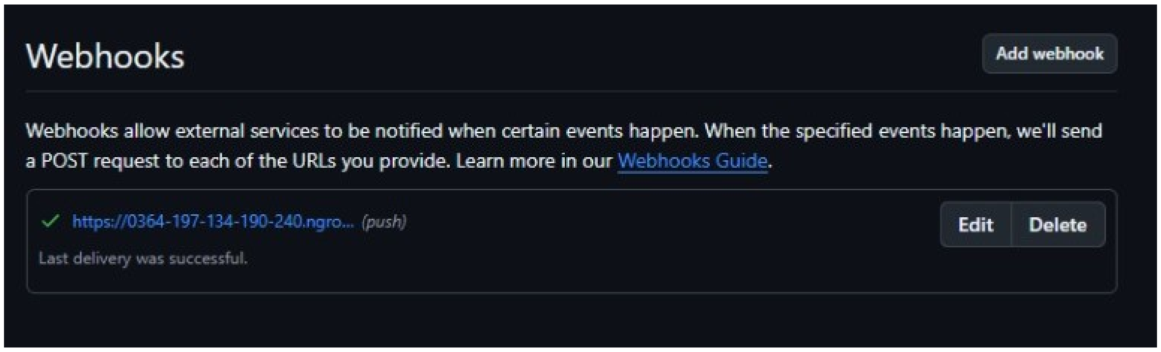
Figure 4: Ngrok

So, in summary:

- Port 8080: This is where Jenkins runs.
- Port 4040: This is where ngrok's web interface runs
- Ngrok URL: This is the publicly accessible URL that routes traffic to jenkins

5.2 Configuring Github Webhook

Moreover I had to configure Github Webhook where
The Payload URL is ngrok URL followed by /github-webhook/
which is a common endpoint used by Jenkins when configured to receive webhooks


Webhooks

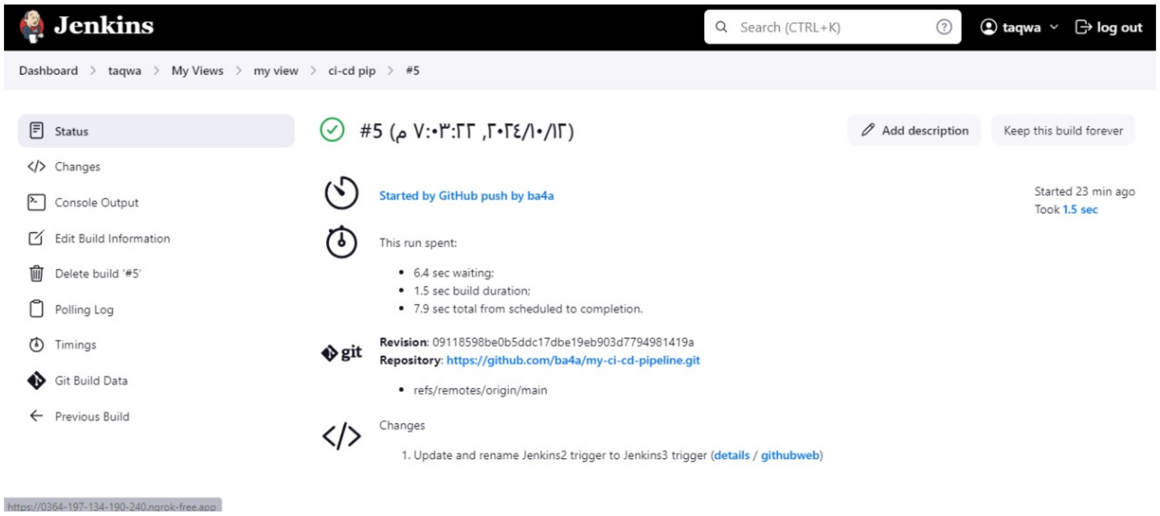
Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our Webhooks Guide.

https://0364-197-134-190-240.ngro... (push)
Last delivery was successful.

Figure 5: Configuring Github Webhook

5.3 Results

I've renamed and edited file that I pushed specifically to test the triggering behaviour.
After doing that the pipeline automatically started a new build.



#5 (١٣/١٠/٢٠٢٤, ٠٧:٠٣:٢٢ م)

Started by GitHub push by ba4a

Started 23 min ago
Took 1.5 sec

This run spent:

- 6.4 sec waiting;
- 1.5 sec build duration;
- 7.9 sec total from scheduled to completion.

Revision: 09118598be0b5ddc17dbe19eb903d7794981419a
Repository: https://github.com/ba4a/my-ci-cd-pipeline.git

- refs/remotes/origin/main

Changes

1. Update and rename Jenkins2 trigger to Jenkins3 trigger (details / githubweb)

https://0364-197-134-190-240.ngrok-free.app

Figure 6: Triggering results

6 Jenkins Build and Test stages in Jenkins

This section processes are done by Omnia Magdy Abdelnaby

6.1 Environment

I've added the following variables to avoid repetitions and make the script more readable:

```
IMG_NAME='petclinic-tomcat:latest'
DOCKER_HUB_USR='mohamedwaleed77'
DOCKER_REPO_NAME='mohamedwaleed77/depi_petclinic:latest'
DOCKER_HUB_TOKEN=credentials('dockerhub')
```

6.2 Build Stage

I used pre-written shell script to build the image.
IMG_NAME variable value is the name of built image :**petclinic-tomcat:latest** which is written like that in (**autobuild.sh**) script
Then I've added the dockerhub token to my jenkins credentials as a secret text. where **DOCKER_HUB_USR** value is the ID of this credential.

I used **DOCKER_HUB_USR** and **DOCKER_HUB_TOKEN** to login into the desired dockerhub account.

Then I tagged the built image to the repository image to push it.

```
stage('Build') {
    steps {
        sh './autobuild.sh'
        sh "echo ${DOCKER_HUB_TOKEN} | docker login -u ${DOCKER_HUB_USR} --password-stdin"
        sh "docker tag ${IMG_NAME} ${DOCKER_REPO_NAME}"
        sh 'docker push ${DOCKER_REPO_NAME}'
    }
}
```

6.3 Testing Stage

I first started the application using docker compose up -d
where -d option is to make the command run in the background instead of making the pipeline stuck at this point.

then I added sleep 60 command to give a chance (60 seconds) for manual testing if wanted.

then I changed the directory to the tests scripts in order to start the pre-written test scripts.

```
stage('Testing') {
        steps {
            sh 'docker compose up -d && sleep 60'
            dir("test_scripts") {
            sh './tests.sh'
        }
    }
}
```

To end this process I've added sh 'docker compose down'

and then removed target directory (which is build by maven but it's not useful anymore at this point)

then I removed savedata_mw directory which is created as a docker volume in docker-compose.yml file.

However it required sudo permissions to remove it as to the system, the directory owner is systemd-coredump. not Jenkins user.

```
    post {
        success {
            . . .
        }
        failure {
            . . .
        }
        always{
            script{
                 sh 'docker compose down'
                 sh 'rm -rf target && sudo rm -rf savedata_mw'
            }
        }
    }
}
```

7 Ansible and AWS

This section processes are done by Esraa Shaaban

Before running the Ansible playbook, ensure you have the following:

- Ansible installed on your local machine.
- An AWS account with permissions to create EC2 instances and manage security groups.
- Basic knowledge of AWS services and command-line operations.

7.1 Infrastructure Setup

7.1.1 EC2 Instance Configuration

Launch an EC2 Instance:

- Create an EC2 instance using the Ubuntu AMI.
- Choose an instance type (e.g., `t2.micro` for testing).
- Select or create a key pair for SSH access.

7.1.2 Security Group Configuration

Configure the security group to allow inbound traffic on port 5000. This port will be used to access the PetClinic application.

7.1.3 Inventory Configuration

The inventory file contains the details of the EC2 instance where the Docker container will be deployed. Below is the configuration:

```
[ec2]
35.172.137.209 ansible_user=ubuntu ansible_ssh_private_key_file=/home/esraa/deployment-Instance.pem
```

Explanation:

- `35.172.137.209`: Public IP address of the EC2 instance.
- `ansible_user`: The user used for SSH access (typically `ubuntu` for Ubuntu instances).
- `ansible_ssh_private_key_file`: Path to the private key file used for SSH authentication.

7.1.4 Ansible Playbook

The following is the Ansible playbook used for deploying the Docker container:

```
---
- name: Deploy Docker container on Ubuntu EC2
  hosts: ec2
  become: yes
  vars:
    docker_image: "mohamedwaleed77/depi_petclinic:latest"
    container_name: "petclinic_app"
    host_port: 5000
    container_port: 8080
    venv_path: "/opt/ansible-venv"
    ansible_python_interpreter: /usr/bin/python3  # Path for Python interpreter

  tasks:
    - name: Update apt cache
      apt:
        update_cache: yes

    - name: Install dependencies for Docker installation
      apt:
        name:
          - apt-transport-https
          - ca-certificates
          - curl
          - gnupg
          - lsb-release
        state: present

    - name: Install Python3 and venv
      apt:
        name:
          - python3
          - python3-venv
        state: present

    - name: Create a Python virtual environment
      command: python3 -m venv {{ venv_path }}
      args:
        creates: "{{ venv_path }}"

    - name: Install Docker SDK for Python in the virtual environment
      command: "{{ venv_path }}/bin/pip install docker"

    - name: Download Docker installation script
```

```yaml
  command: curl -fsSL https://get.docker.com -o get-docker.sh

- name: Run Docker installation script
  command: sh get-docker.sh

- name: Ensure Docker is running
  service:
    name: docker
    state: started
    enabled: true

- name: Pull the Docker image
  community.docker.docker_image:
    name: "{{ docker_image }}"
    source: pull
    pull:
      platform: amd64

- name: Stop and remove any existing container
  community.docker.docker_container:
    name: "{{ container_name }}"
    state: absent

- name: Run the new Docker container
  community.docker.docker_container:
    name: "{{ container_name }}"
    image: "{{ docker_image }}"
    state: started
    ports:
      - "{{ host_port }}:{{ container_port }}"
```

7.2 Playbook Breakdown

- **Update apt cache:** Updates the package index to ensure the latest packages are available.
- **Install Docker dependencies:** Installs necessary dependencies for Docker installation.
- **Install Python3 and venv:** Installs Python 3 and the `venv` package for creating virtual environments.
- **Create a Python virtual environment:** Sets up a Python virtual environment at the specified path.
- **Install Docker SDK for Python:** Installs the Docker SDK within the virtual environment.
- **Download Docker installation script:** Fetches the Docker installation script from Docker's official website.
- **Run Docker installation script:** Executes the installation script to set up Docker.
- **Ensure Docker is running:** Starts the Docker service and enables it to run at boot.
- **Pull the Docker image:** Pulls the specified Docker image from the Docker registry.
- **Stop and remove existing container:** Stops and removes any previously running container with the same name.
- **Run the new Docker container:** Launches a new Docker container from the specified image and maps the host port to the container port.

7.3 Deployment Steps

7.3.1 Prepare the Environment:

- Ensure the EC2 instance is running with the correct security group settings.
- Verify that the inventory file is correctly configured.

7.3.2 Run the Playbook:

Execute the playbook using the following command:

```
ansible-playbook -i inventory_file_path playbook.yml
```

Replace `inventory_file_path` with the actual path to your inventory file and `playbook.yml` with the name of your playbook.

7.4 Accessing the Application

Once the playbook completes successfully, access the application by navigating to:

```
http://<EC2_Public_IP>:5000
```

in your web browser, replacing `<EC2_Public_IP>` with the actual public IP address of your EC2 instance.

8 Terraform

8.1 features

- **Integrated Pipeline Execution:** Terraform runs inside the Jenkins pipeline to provision infrastructure.
- **AWS Cloud Resources:** EC2 instance is created with necessary security configurations.
- **Seamless Ansible Deployment:** Once provisioned, Ansible deploys the application on the EC2 instance.

8.2 Usage

1. Ensure Terraform and AWS CLI are installed and configured.
2. Define AWS details in `variables.tf`.
3. Run the Jenkins pipeline to automate infrastructure provisioning and deployment.

8.3 Prerequisites

- Terraform CLI installed.
- AWS CLI configured with access credentials.
- Jenkins pipeline configured to trigger Terraform and Ansible jobs.

9 Jenkins Refine

This section are completed by Magdy Ayman Abu Bakr.

9.1 Before running the pipeline:

If you are **NOT** using jenkins from a .war file you have to do the following:

- Add jenkins user to Docker group
- Add jenkins user to sudo group
- Use ”sudo visudo” or ”nano /etc/sudoers” and add NOPASSWD:ALL to sudo group (or jenkins user) as this pipeline requires sudo permissions in some parts.
- Ensure your machine is authorized to access the targeted machines in inventory file.
- Ensure to comment the parts you want to ignore in your process.

If you are using jenkins from a .war file
you have to do the same but for your user instead of jenkins user as jenkins in this case will be runned by your user

9.2 pipeline script

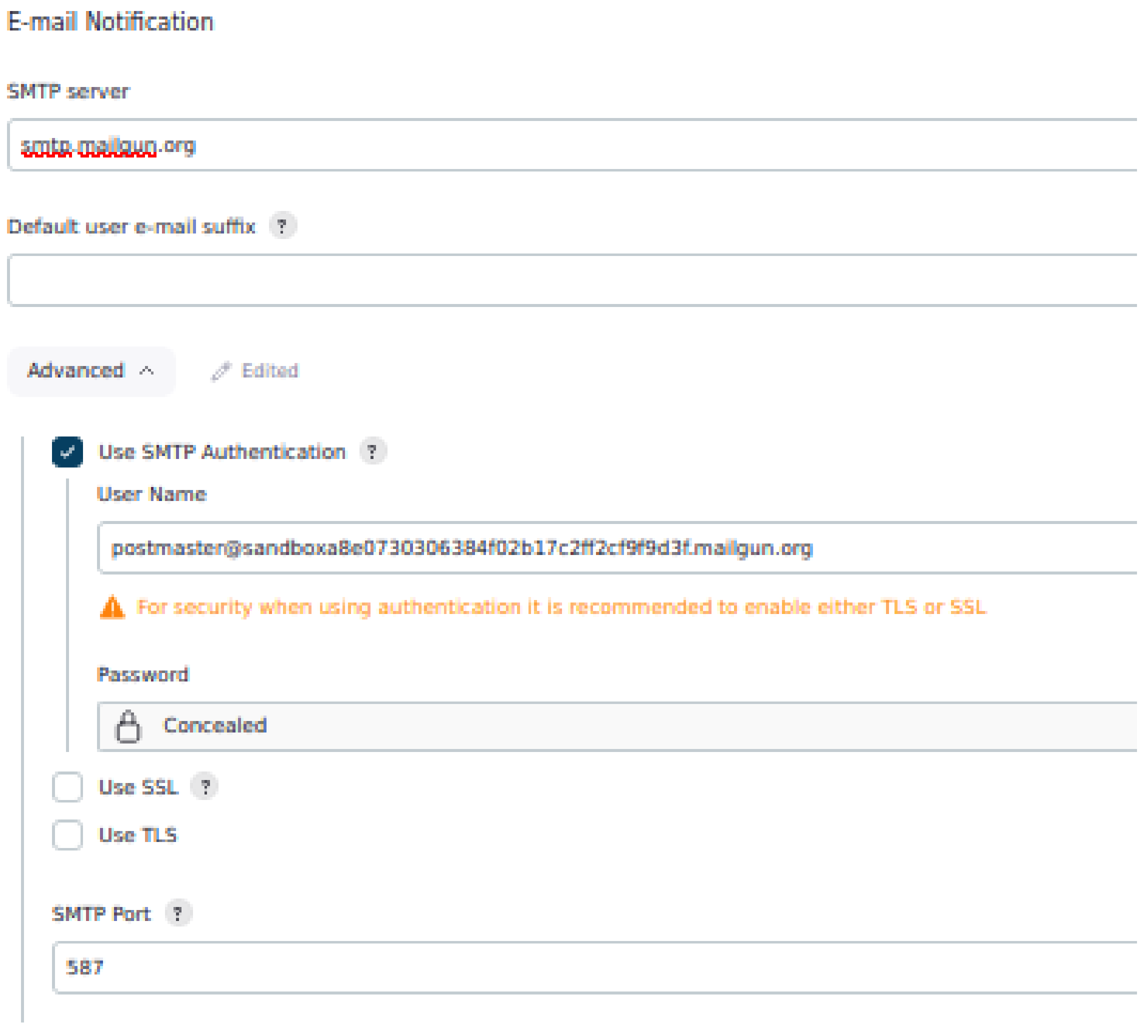
```
pipeline {
agent any
triggers {
    pollSCM('H * * * *') // every 1 hour
}
environment {
    REPO_NAME= 'ci-cd-pipeline-with-jenkins-docker-ansible-aws' //because it's very long
    REPO_URL = "https://github.com/EsraaShaabanElsayed/ci-cd-pipeline-with-jenkins-docker-ansible-aws.git"
    IMG_NAME='petclinic-tomcat:latest'
    DOCKER_HUB_USR='mohamedwaleed77'
    DOCKER_REPO_NAME='mohamedwaleed77/depi_petclinic:latest'
    DOCKER_HUB_TOKEN=credentials('dockerhub')
    NOTI_EMAIL= "example@gmail.com"
}

stages {
    stage('Checkout') {
        steps {
            git url: "${REPO_URL}", branch: 'main'
            sh 'pwd ; ls -la'
        }
    }
    stage('Build') {
        steps {
            sh './autobuild.sh'
            //sh "echo ${DOCKER_HUB_TOKEN} | docker login -u ${DOCKER_HUB_USR} --password-stdin"
            //sh "docker tag ${IMG_NAME} ${DOCKER_REPO_NAME}"
            //sh 'docker push ${DOCKER_REPO_NAME}'
            //commented to save uploading time
        }
    }
    stage('Testing') {
        steps {
            sh 'docker compose up -d && sleep 60'//in case manual testing is wanted.
            dir("test_scripts") {
                sh './tests.sh'
            }
        }
    }
    stage('Deployment') {
        steps {
                sh 'ansible-playbook -i inventory playbook.yml"'
        }
    }
}
```

```
    post {
        success {
            echo 'Deployment succeeded!'
            mail to: "${NOTI_EMAIL}",
            subject: "SUCCESS: Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]'",
            body: "Good news! Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]' has succeeded.\nCheck it here: ${env.BUILD_URL}"
        }
        failure {
            echo 'Deployment failed!'
            mail to: "${NOTI_EMAIL}",
            subject: "FAILURE: Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]'",
            body: "Unfortunately, Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]' has failed.\nCheck it here: ${env.BUILD_URL}"
        }
        always{
            script{
                 sh 'docker compose down'
                 sh 'rm -rf target && sudo rm -rf savedata_mw'
            }
        }
    }
}
```

9.3 Key Changes

- Instead of using Webhooks we used poll SCM which is **less efficient** but more simple and easy to use.
- Added sh 'pwd ; ls -la' to checkout stage to ease the debugging process.
- Set Up Notifications
- Integrated all pipeline scripts to make one final script.

9.4 Setting up SMTP



E-mail Notification

SMTP server: smtp.mailgun.org

Default user e-mail suffix:

Advanced ^ Edited

✓ Use SMTP Authentication

User Name: postmaster@sandboxa8e0730306384f02b17c2ff2cf9f9d3f.mailgun.org

For security when using authentication it is recommended to enable either TLS or SSL

Password: Concealed

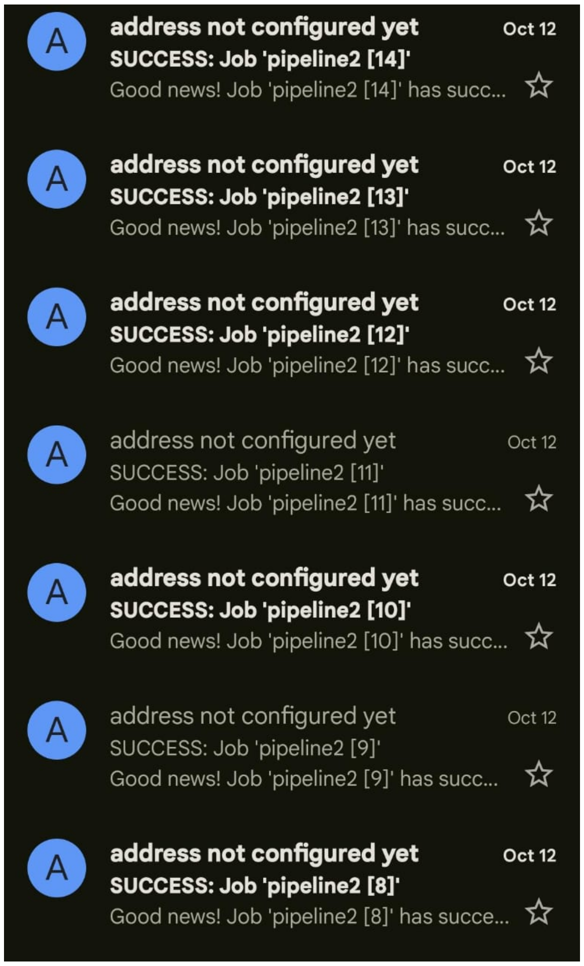
☐ Use SSL

☐ Use TLS

SMTP Port: 587

Figure 7: Setting up SMTP

9.5 Results



```
address not configured yet                    Oct 12
SUCCESS: Job 'pipeline2 [14]'
Good news! Job 'pipeline2 [14]' has succ...

address not configured yet                    Oct 12
SUCCESS: Job 'pipeline2 [13]'
Good news! Job 'pipeline2 [13]' has succ...

address not configured yet                    Oct 12
SUCCESS: Job 'pipeline2 [12]'
Good news! Job 'pipeline2 [12]' has succ...

address not configured yet                    Oct 12
SUCCESS: Job 'pipeline2 [11]'
Good news! Job 'pipeline2 [11]' has succ...

address not configured yet                    Oct 12
SUCCESS: Job 'pipeline2 [10]'
Good news! Job 'pipeline2 [10]' has succ...

address not configured yet                    Oct 12
SUCCESS: Job 'pipeline2 [9]'
Good news! Job 'pipeline2 [9]' has succ...

address not configured yet                    Oct 12
SUCCESS: Job 'pipeline2 [8]'
Good news! Job 'pipeline2 [8]' has succe...
```

Figure 8: Results

10 Tasks

Name	Tasks
Mohamed Waleed Mohamed Abdelfattah — 21044904	Docker,Tomcat,Maven and k8s
Mahmoud Tarek Medany — 21026143	Testing scripts and documentation
Esraa Shaaban — 21032361	Ansible, k8s, Terraform and AWS
Taqwa Elsayed Mohammed Fahmy — 21025433	Jenkins: Triggering and pulling the repo
Magdy Ayman Abu Bakr — 21003567	Jenkins: Deployment stage and Notifications
Omnia Magdy Abdelnaby — 21027962	Jenkins: bulding stage and Testing stage