# **Deployment Part 1– Git versioning and Dockerfile**

**Git versioning for the first time creation**

1. Git installation
2. Create a Git repository
3. Checking git version in VS Code terminal :

* Activate venv
* git version

1. Initializes a new Git repository : git init
2. Create a .gitignore file and list the things which has to be excluded from the git repository
3. Creating a branch : git branch -M main
4. Connecting with git repo: git remote add origin https://github.com/Nihal124/RAG-MEDICAL-CHATBOT.git
5. Stage changes to a file to be committed except those in the .gitignore : git add .
6. To commit staged changes with a descriptive message : git commit -m “commit”

Note(If you are getting the error) : git commit -m “commit” Author identity unknown

git config --global user.name "Mahammad Nihal"

git config --global user.email [mahammadnihaln@gmail.com](mailto:mahammadnihaln@gmail.com)

1. Upload committed changes to a remote Git repository - git push origin main

**Git versioning while working**

1. git add .
2. git commit -m “commit”
3. git push origin main

**Dockerfile creation**

# **Deployment Part 2– Jenkins setup for CI-CD Deployment**

1. Install the **Docker Desktop**
2. Open the terminal and navigate to this folder : cd custom\_jenkins
3. Make sure **Docker Desktop is running in the background**, then build the image: docker build -t jenkins-dind .
4. Run Jenkins container (all together at once) :

docker run -d ^

--name jenkins-dind ^

--privileged ^

-p 8080:8080 ^

-p 50000:50000 ^

-v /var/run/docker.sock:/var/run/docker.sock ^

-v jenkins\_home:/var/jenkins\_home ^

jenkins-dind

1. Access Jenkins Dashboard:

[http://localhost:8080](http://localhost:8080/)

1. To get the password for Jenkins:  
   docker logs jenkins-dind

Note - If the password isn’t visible, run:

docker exec jenkins-dind cat /var/jenkins\_home/secrets/initialAdminPassword

1. Paste the password in the Jenkins Dashboard and choose the Install suggested plugins
2. If you want you can create the user name and password or skip as admin

Note - (As you have skipped the username creation) username will be : admin

1. Install Python inside Jenkins container (In vs code in the virtual environment and in the

cd custom\_jenkins : **(Run the code one by one)**

docker exec -u root -it jenkins-dind bash

apt update -y

apt install -y python3

python3 --version

ln -s /usr/bin/python3 /usr/bin/python

python --version

apt install -y python3-pip

exit

1. Restart the Jenkins container : docker restart jenkins-dind
2. Open Jenkins Dashboard and login again

# **Deployment Part 3– Github Integration with Jenkins**

1. Open Github > Profile > Settings (Open Settings in a new tab) > Developer settings > Personal access tokens > Tokens (classic) > Generate new token > Generate new token(classic) > login with the password > In the note section give any name (for ex: rag) > (give two permissions ) **repo and admin:repo\_hook** > Generate token
2. Go to Jenkins dashboard > Manage Jenkins (settings) > Credentials > global > Add Credentials > enter the git hub user name > for password copy the access token and paste it > id can be anything (for example github-token) > description also give the same > create token
3. Go back to the Jenkins dashboard > New Item > Enter item name (for example: RAG CHATBOT MEDICAL) > select Pipleline > click Ok > Drag down to get the pipeline section > for the Definition select Pipeline script for SCM > for SCM select Git > for Repository URL, go to the github copy the URL from the code section and paste it > for Credentials select the Github credentials in the dropdown that is previously passed > for Branch change it main or keep it master based on Github repository branch > apply > save
4. Go to the pipeline syntax > for Sample Step select the checkout : Check out from version control > for Repository URL, go to the github copy the URL from the code section and paste it > for Credentials select the Github credentials in the dropdown that is previously passed > for Branch change it main or keep it master based on Github repository branch > click on Generate Pipeline Script
5. Copy the content of the Jenkinsfile > create a new file in the in the vscode inside project folder with the name Jenkinsfile > paste the content > Have to execute Jenkins content step by step (refer the video) > contro + / (shortcut for commenting down the code ) > Comment out the environment, build scan push stage, Deploy to AWS runner >remove the checkout section in the code and replace it with one you are getting from Jenkins dashboard (from generate pipeline script section ) – Save the changes
6. Now have to upload this to the github > open the vs code terminal and activate the environment and you should be in the main project folder (if you are inside any other folder just provide exit text ) >

git init

git add .

git commit -m “commit”

git push origin main

1. Go to Jenkins dashboard > click on the pipeline ( for example RAG CHATBOT MEDICAL ) > click on build now

# **Deployment Part 4– Build scan with AquaTrivy & Push to AWS ECR**

**Note : Make sure that you have created setup.py file and installed in the environment using**

Pip install -e .

1. Install Trivy in Jenkins Container : (run one by one inside custom\_jenkins)

cd custom\_jenkins

docker exec -u root -it jenkins-dind bash

apt install -y

**curl -LO https://github.com/aquasecurity/trivy/releases/download/v0.62.1/trivy\_0.62.1\_Linux-64bit.deb**

dpkg -i trivy\_0.62.1\_Linux-64bit.deb

trivy --version

exit (don’t exit now if you are continuing with the installation of AWS CLI)

1. Install AWS CLI inside Jenkins container :

docker exec -u root -it jenkins-dind bash

apt update

apt install -y unzip curl

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

./aws/install

aws --version

exit

1. After exit Restart the Jenkins container : docker restart jenkins-dind
2. Install AWS plugins in Jenkins :

Go to Jenkins dashboard > Manage Jenkins > Plugins > Available plugins > search for AWS >

Install AWS Credentials and AWS Web Services SDK :: All

1. Restart the Jenkins container : docker restart jenkins-dind
2. Create an IAM user in AWS (After login)

Go to AWS Console – Search and open the IAM – Users – Create user (for ex:rag-medical) – Next – select Attach policies directly (search for EC2 and select AmazonEC2containerregistryfullaccess) – Next – Click on Create user

1. Open create user – Click on create access key – select the Command line interface – Next – Click on Create access key
2. Add AWS credentials to Jenkins

Go to Jenkins dashboard – Manage Jenkins – Credentials - global – Add credentials – select AWS credentials in the drop down – id and description (for ex: aws-token) – provide access key and secret key – create

1. Create an ECR Repository

Open AWS Console – Search for ECR – Open in new tab – create repository – name ( for ex: my repo) ( make sure it is mutable) – create

1. In VS Code, open Jenkins file and uncomment the environment and build scan push stage (shortcut for uncomment – ‘control’ + ‘/’) – replace the info in the environment (AWS\_REGION and ECR\_REPO) - replace the info in the build scan push stage ( credentialsId) – Save the changes
2. Now have to upload this to the github > open the vs code terminal and activate the environment and you should be in the main project folder (if you are inside any other folder just provide exit text ) >

git init

git add .

git commit -m “commit”

git push origin main

1. Go to Jenkins dashboard > click on the pipeline ( for example RAG CHATBOT MEDICAL ) > click on build now
2. Fix Docker Daemon Issues (If Any)

If you encounter Docker socket permission issues, fix with: ( run one by one inside cd custom\_jenkins)

docker exec -u root -it jenkins-dind bash

chown root:docker /var/run/docker.sock

chmod 660 /var/run/docker.sock

getent group docker

# If group 'docker' exists, skip next line

usermod -aG docker jenkins

exit

docker restart jenkins-dind

( Go to Jenkins dashboard and build )

# **Deployment Part 5–Deployment to AWS Runner**

1. IAM User Permissions

Go to AWS Console > IAM > Users > Open the user created by you (Ex: rag-medical) > Add permissions > add permissions > attach policies directly > Click the check box of AWSAppRunnerFullAccess > Next > Add permission

1. In AWS Console – Search for Appr runner– Open in new tab – create an app runner service – repository type – Container registry -Provider – Amazon ECR –Container Image URL - Browse for the ECR Repo or paste the ECR Repo URL – Deployment trigger keep manual or automatic based on the requirement – ECR access role – Create new service role – Next – Service name – give the service name you have used in Jenkins file – add environment variable – name – env variable name used in .env file (ex: OPENAI\_API\_KEY) – Environment variable value (paste the access key without quotes) – Port (which one the flask app is using for ex : 5000) – Networking – Public endpoint – Public access – Next – Create and deploy
2. Uncomment the Deploy to app runner part in Jenkins file – save and push the files to git

git init

git add .

git commit -m “commit”

git push origin main

1. Go to Jenkins dashboard and build the pipline