AKSHIT MITTAL
10/27/23

Build a Docker Jenkins Pipeline to Implement CI/CD Workflow

Description

Demonstrate the continuous integration and delivery by building a Docker Jenkins Pipeline.

Problem Statement Scenario:

You are a DevOps consultant in AchiStar Technologies. The company decided to implement DevOps to develop and deliver their products. Since it is an Agile organization, it follows Scrum methodology to develop the projects incrementally. You are working with multiple DevOps Engineers to build a Docker Jenkins Pipeline. During the sprint planning, you agreed to take the lead on this project and plan on the requirements, system configurations, and track the efficiency. The tasks you are responsible for:

- Availability of the application and its versions in the GitHub
- Track their versions every time a code is committed to the repository
- Create a Docker Jenkins Pipeline that will create a Docker image from the Dockerfile and host it on Docker Hub
- It should also pull the Docker image and run it as a Docker container
- Build the Docker Jenkins Pipeline to demonstrate the continuous integration and continuous delivery workflow

Company goal is to deliver the product frequently to the production with high-end quality.

You must use the following tools:

- Docker: To build the application from a Dockerfile and push it to Docker Hub
- Docker Hub: To store the Docker image
- GitHub: To store the application code and track its revisions
- Git: To connect and push files from the local system to GitHub
- Linux (Ubuntu): As a base operating system to start and execute the project
- Jenkins: To automate the deployment process during continuous integration

Following requirements should be met:

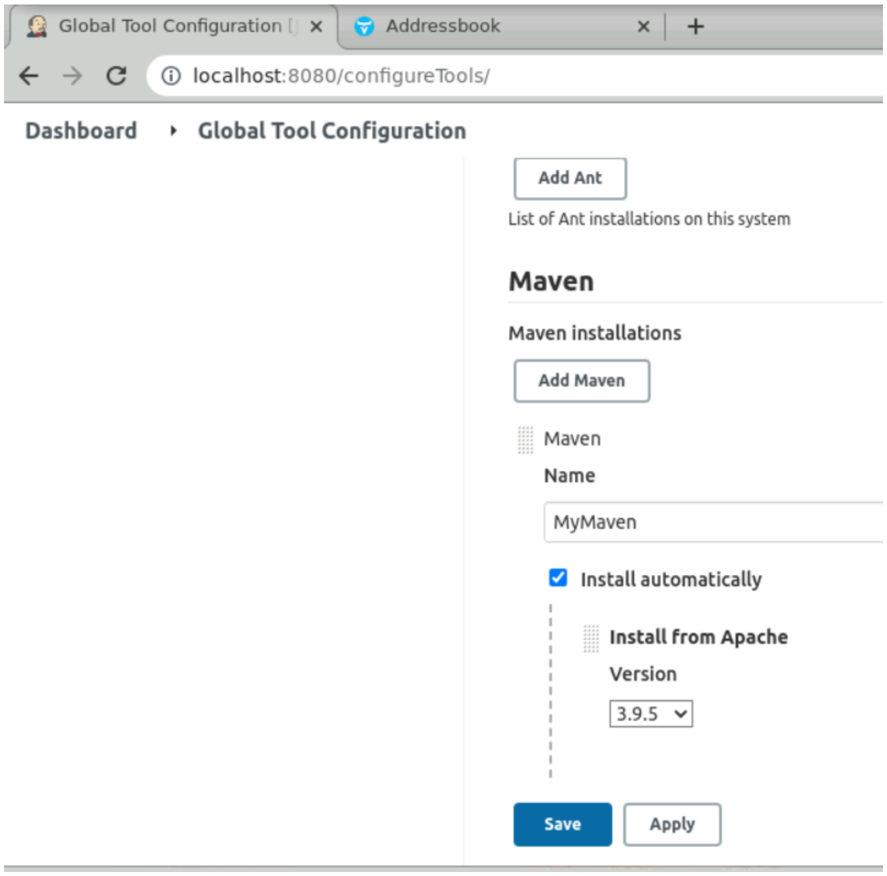
- Document the step-by-step process from the initial installation to the final stage
- Track the versions of the code in the GitHub repository
- Availability of the application in the Docker Hub
- Track the build status of Jenkins for every increment of the project

SOLUTION

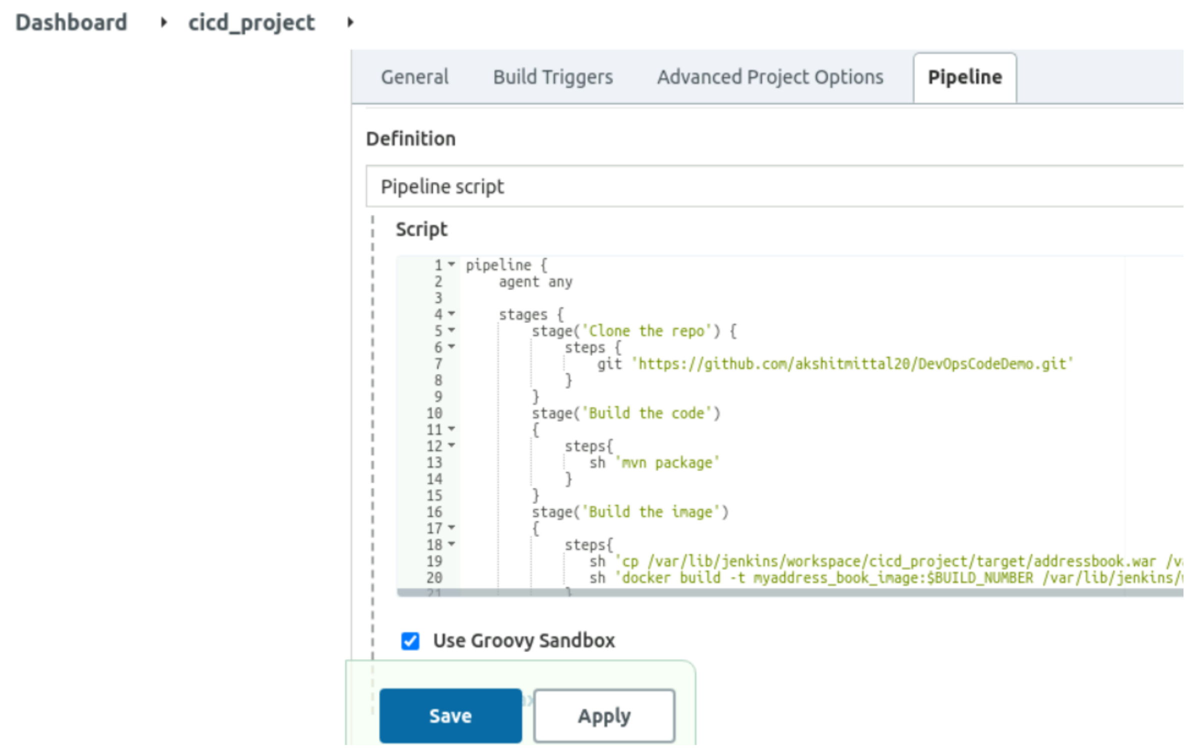
Workflow-:

→ Install maven plugins on Jenkins
→ Cone the repo → build the code artifact .war file → Create the image → CI
→ Push the image to docker hub Registry → Run the image to create the container running → CD
→ Set up the webhooks and trigger via jenkins to github → CICD

1. Login into the Jenkins. Install maven tool on Jenkins



2. Select the pipeline in new project section. Give pipeline name and start writing the pipeline



Definition

Pipeline script

Script

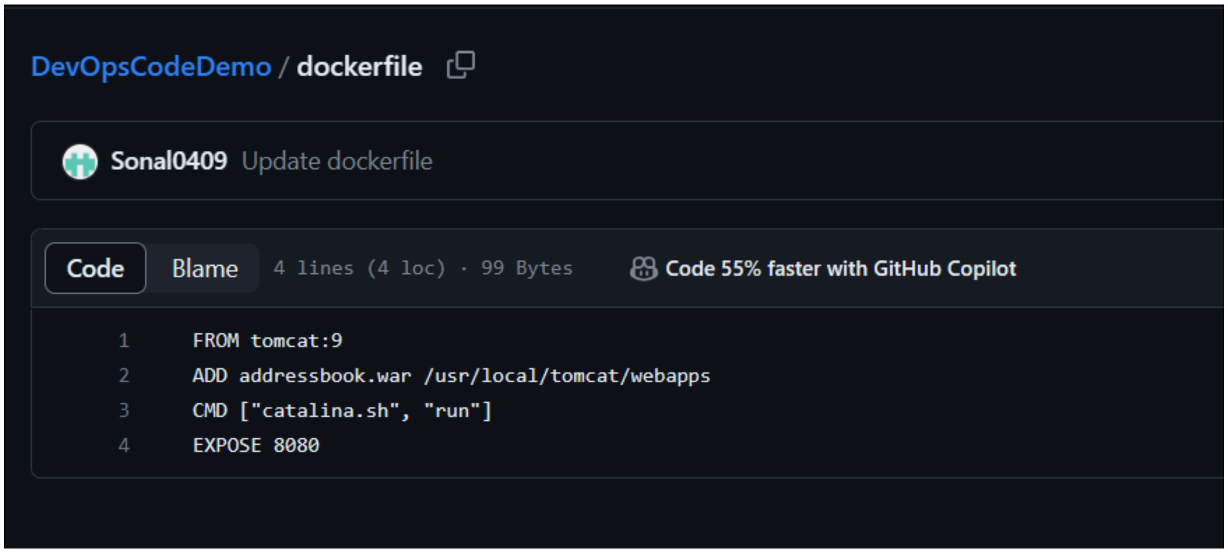
```
pipeline {
    agent any

    stages {
        stage('Clone the repo') {
            steps {
                git 'https://github.com/akshitmittal20/DevOpsCodeDemo.git'
            }
        }
        stage('Build the code')
        {
            steps{
               sh 'mvn package'
            }
        }
        stage('Build the image')
        {
            steps{
               sh 'cp /var/lib/jenkins/workspace/cicd_project/target/addressbook.war /v
               sh 'docker build -t myaddress_book_image:$BUILD_NUMBER /var/lib/jenkins/
```

☑ Use Groovy Sandbox

3. Give Jenkins permission to run docker commands through command : #chmod 777 /var/run/docker.sock , in your CLI

4. The pipeline syntax:

→ Clone the repo – Clone the repository of your code available on GitHub. Make sure it contains the docker file too to create the image.


DevOpsCodeDemo / dockerfile
Sonal0409 Update dockerfile
Code
Blame
4 lines (4 loc) · 99 Bytes
Code 55% faster with GitHub Copilot

```
FROM tomcat:9
ADD addressbook.war /usr/local/tomcat/webapps
CMD ["catalina.sh", "run"]
EXPOSE 8080
```

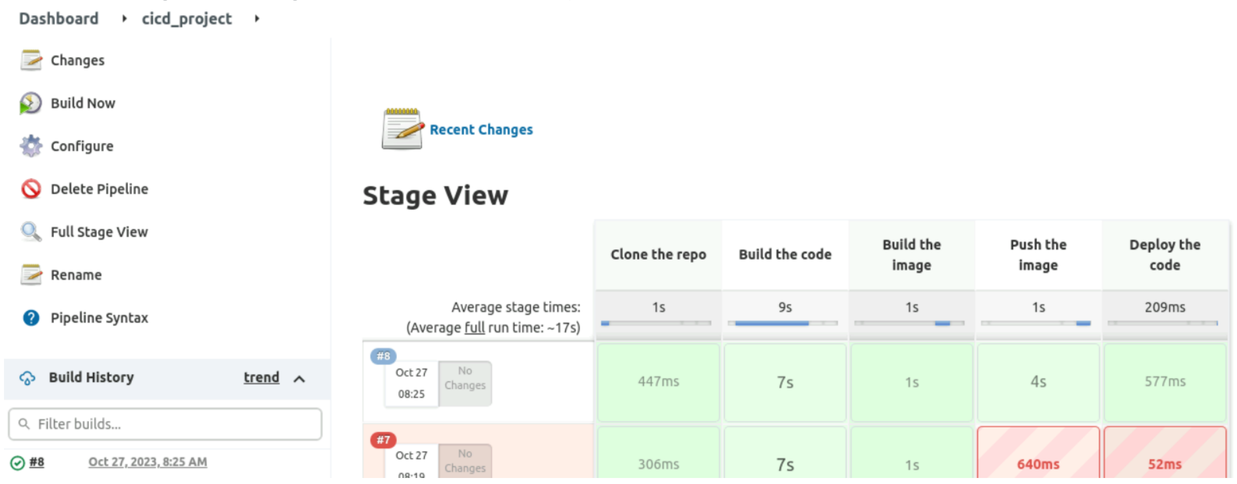
→ Build the code – we use maven tool for building the code artifact.

→ Push the image – We will first copy the .war file path which is build by maven previously and paste it into the folder where our dockerfile is present.
Docker file is already present in the directory of project here, hence we copy the .war file there. We can see all the path of the created files in the workspace section of Jenkins or in the logs in Jenkins.
After that we push the image to the docker hub public registry after logging in. Please note, we need to tag the image name with the username of docker hub, to push it to our repo in docker hub. Also please note, the $ BUILD NUMBER tag after image name define the image number after it is build. This is used when there is continuous CICD happening and multiple images and containers are getting generated. $buildNumber helps us to identify the image

→ Deploy the code → Running the image in the detached environment to create a container.

```
pipeline {
    agent any

    stages {
        stage('Clone the repo') {
            steps {
                git 'https://github.com/akshitmittal20/DevOpsCodeDemo.git'
            }
        }
        stage('Build the code')
        {
            steps{
               sh 'mvn package'
            }
        }
        stage('Build the image')
        {
            steps{
               sh 'cp /var/lib/jenkins/workspace/cicd_project/target/addressbook.war /var/lib/jenkins/workspace/cicd_project '
               sh 'docker build -t myaddress_book_image:$BUILD_NUMBER /var/lib/jenkins/workspace/cicd_project'
            }
        }
         stage('Push the image')
        {
            steps{
               sh 'docker login -u akshitmittal20 -p Am@151220'
               sh 'docker tag myaddress_book_image:$BUILD_NUMBER akshitmittal20/myaddress_book_image:$BUILD_NUMBER'
               sh 'docker push akshitmittal20/myaddress_book_image:$BUILD_NUMBER'
            }
        }
        stage('Deploy the code')
        {
            steps {
                sh 'docker run -d -P akshitmittal20/myaddress_book_image:$BUILD_NUMBER'
            }
        }
    }
}
```

5. Save the pipeline and click on Build now to execute ever stage step and you may also see logs clicking on the build history latest build number → output

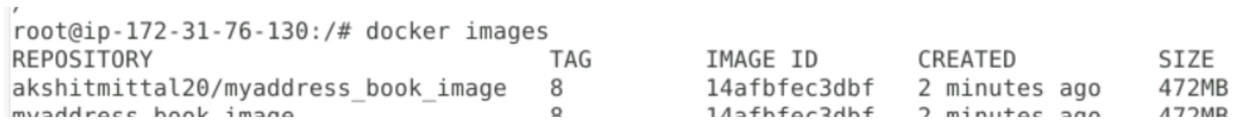


Stage View

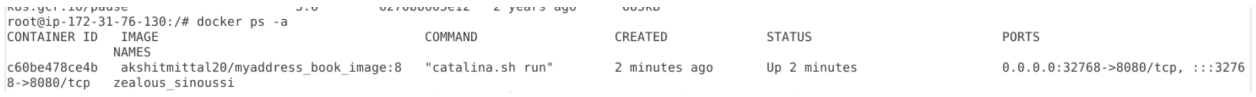
	Clone the repo	Build the code	Build the image	Push the image	Deploy the code
Average stage times: (Average full run time: ~17s)	1s	9s	1s	1s	209ms
#8 Oct 27 08:25 No Changes	447ms	7s	1s	4s	577ms
#7 Oct 27 08:19 No Changes	306ms	7s	1s	640ms	52ms

6. You will be able to successfully see the image created for your build and container

running with the port number details.

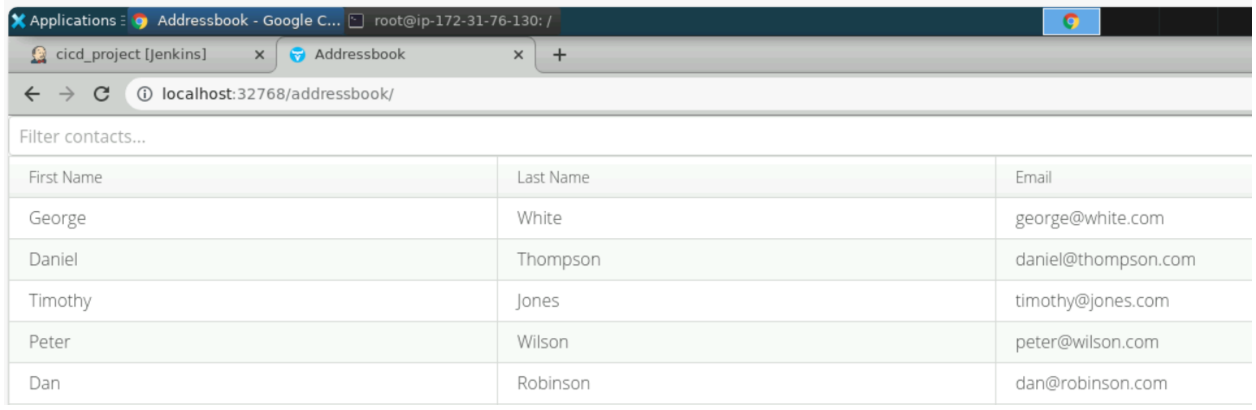


```
root@ip-172-31-76-130:/# docker images
REPOSITORY                          TAG      IMAGE ID       CREATED          SIZE
akshitmittal20/myaddress_book_image   8        14afbfec3dbf   2 minutes ago    472MB
myaddress_book_image                8        14afbfec3dbf   2 minutes ago    472MB
```



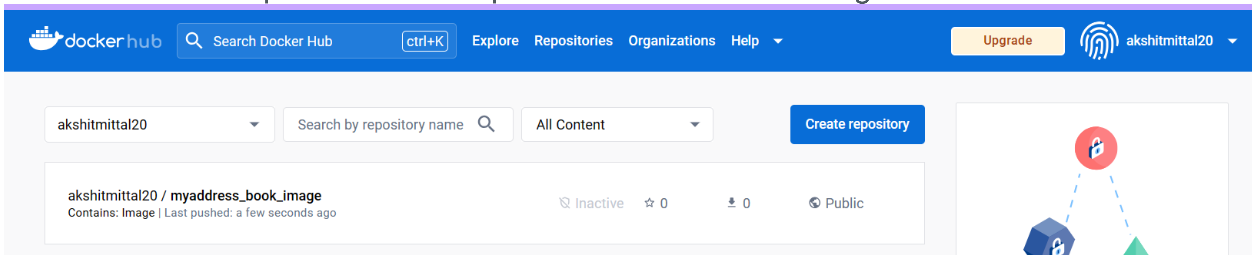
```
root@ip-172-31-76-130:/# docker ps -a
CONTAINER ID   IMAGE                                COMMAND              CREATED         STATUS         PORTS
               NAMES
c60be478ce4b   akshitmittal20/myaddress_book_image:8   "catalina.sh run"    2 minutes ago   Up 2 minutes   0.0.0.0:32768->8080/tcp, :::3276
8->8080/tcp    zealous_sinoussi
```

You will be able to see the code running on port number, details mentioned in the container console. Please note, the /addressbook/ tag in the URL is mandatory for running any java application Provide the war executable file name at the end.



First Name	Last Name	Email
George	White	george@white.com
Daniel	Thompson	daniel@thompson.com
Timothy	Jones	timothy@jones.com
Peter	Wilson	peter@wilson.com
Dan	Robinson	dan@robinson.com

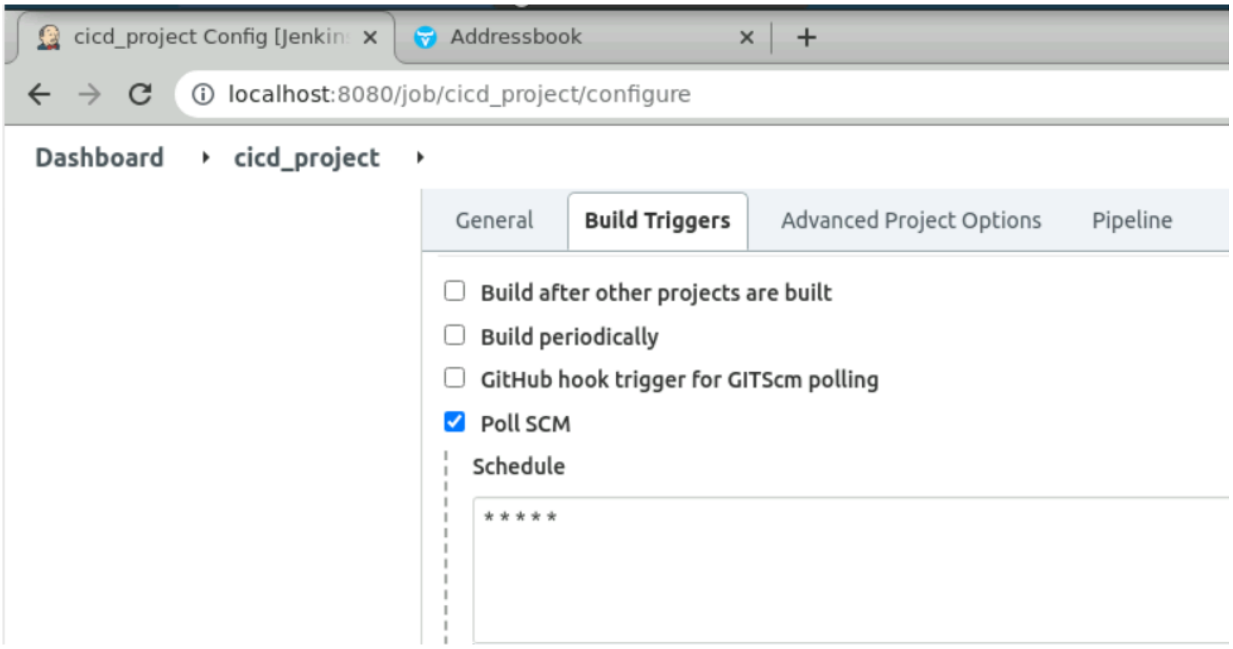
The docker hub repo will also be updated with the latest image.



akshitmittal20 / myaddress_book_image
Contains: Image | Last pushed: a few seconds ago
Inactive ☆ 0 ⤓ 0 Public

7. You can also set up the webhooks triggers in the Jenkins for github, so that if new commit happens in the git repo, the pipeline is automatically triggered. This will create the CICD continuous workflow.

In the Poll SCM trigger section, We will give * * * * * to specify 1 minute timer for Jenkins. That means Jenkins will check for a new update/commit after every 1 minute.



- [ ] Build after other projects are built
- [ ] Build periodically
- [ ] GitHub hook trigger for GITScm polling
- [x] Poll SCM

Schedule

```
* * * * *
```

This implements our CI-CD workflow with Jenkins, Maven, and Docker.