Jenkins

The problem with that approach is that the image we created is good and we can push it to dockerHub and use it, but what if I change

the project code? Then this change will be only on a local machine, it will not be reflected in the image. Then I have to delete old image

and build new one (or do new version). In the real life whenever we change the code we have to rebuild the image, and in a real life we

work in a team and if a few developers work on one project they are not gonna rebuild project, build tge image, push the image ..., instead

once they change the code they will push it to gitHub repository the Jenkins came up, it will pull the repository to remote machine and

creates the project jars (mvn clean package), it will build the docker image, run the tests, push the image to dockerHub. Its very useful

all you need is just work on project and push it to repo. In this tutorial we will run Jenkins as a container of Docker! For this lets

create directory: C:\for\_experements\docker\_experements\jenkins

Jenkins use file directory to pack everything whatever it uses so it can be reused (its like a history). And since the jenkins is a container

we better use volume mapping to store it locally (for future), if this directory delete locally and try to rerun jenkins you will be doing all

the setup again (with username, password...).

yaml of jenkins:

version: "3"

services:

jenkins:

image: jenkins/jenkins:lts

user: root # to instract the docker that we are gonna run the jenkins as a root with inafe privilage to do volume mapping

ports:

- 8080:8080

- 50000:50000

volumes:

- ./jenkins\_files/master:/var/jenkins\_home # we store inf. that container use, for future use

environment:

- JAVA\_OPTS="-Dhudson.model.DirectoryBrowserSupport.CSP="

Будучи терминалам в директории где расположен jenkins yaml file выполним: docker-compose up.

NODE CONFIGYRATION

Node or slave it is the machine to wich jenkins master will deligate the work! The nodes can be multiple. You need to know the requriments

for the node since its going to run our framwork image, its need to have: java, docker, gid, maven. In that tutorial we are gonna treat

our machine as a node, since it has everithing we need. But if we would have one more computer we could connect it to the master and use it

as a node.

In order to create node we need to use jenkins UI, go to localhost:8000

Dashbord->set up an agent->enter node name->in\_a\_remote\_root\_directory this is folder for node files need to indicate the directory where you

want to keep the files related to the node work->launch\_method choose "Launch agent by connecting it to the controller" it means that node will

automatically connect to the controller->save

At this point the configuration part for the node is ready. Master is now aware that it has a node.But its not connectec yet. To connect to

the node click on it, choose appropriate way (mac, windows...). The principe is to download the jar (first line) for node and run it(multiple

java comand). You have to be in a node directory (to keep things organised) in order to download the jar and then run it. So we need to keep

2 terminals open, one for jenkins image (master), second for his node.

JENKINS JOB

Whatever the tast we want to do wia Jenkins it a Jenkins job. We might create n namber of job, If you know how to execute the comand via cmd,

Jenkins can do it for you.

To create job click + sign enter a neme, choose the tipe, lets use Freestyle project, click build steps, choose acording to OS, enter the ins-

truction in the field->save. Go tp Dashbord, you should see the job presented, click on it, choose 'Build now'. After its done, click on the

result and choose Console output in order to see the logs.

PIPLENE

It is the file in wich we write the comands, that jenkins will execute, one by one, it is written in a domain specific language. It dosnot have

exstention, something similar to Dockerfile.

Jenkinsfile\_pipline - это как список команд которые должны быть выполнены jenkinsom, идея в том чтобы не прописывать

каждую команду а зделать типа список, что за чем, также есть возможность указывать переменные.

Всё наченается в главной секции называющейся piplene, например:

pipline{

any agent <-- любой node(slave) может выполнять следующие далее инструкции, это можно перенастроить на какойто отдельный. Каждый node имеет

конфигуацию, где можно установить/прописать лэйблы. В job-е в jenkins file we indicate not 'any agent', but:

agent{

label 'указываем лэйбл того нода который мы хотим чтобы выполнил этот файл'

}

stages{ <-- блок где расположены stages

stage('stage\_1'){

...здесь команды которые надо выполнить, для удобочитаемости их можно разбить по этим самым stage-ам

}

stage('stage\_2'){

...

}

}

post { <--optional block (что-то типа finally в java где закрываются ресурсы, такоеже назначение и здесь)

...что-то зделать в конце выполнения всей программы

}

}

So lets create jenkinsFile:

pipeline{

agent any

stages{

stage('stage-1'){

steps{

echo "stage-1"

echo "doing mvn clean"

echo "doing mvn package"

}

}

stage('stage-2'){

steps{

echo "stage-2"

echo "building docker image"

}

}

stage('stage-3'){

steps{

echo "stage-3"

echo "pushing docker image"

}

}

}

post{

always{

echo "post section"

}

}

}

Then select new job,give the name, choose pipeline, scrol down and paste the code that we write in a jenkinsfile,save it, press 'build now'.

OVERRIDE VARIABLSE IN JENKINSFILE

Sometimes we may want to override some variables that we use in a file.

pipeline{

agent any

environment {

NUMBER = 3 <-- this is something as global variable. All the stages can refer to this variable.

}

stages{

stage('stage-1'){

steps{ <-- steps is kind of comands that we execute

echo "stage-1"

echo "doing mvn clean"

echo "doing mvn package"

echo "NUMBER = ${NUMBER}" <-- example how to use variable

}

}

stage('stage-2'){

environment {

NUMBER = 100 <--we override variable

}

steps{

echo "stage-2"

echo "building docker image"

echo "NUMBER = ${NUMBER}" <-- printing new value of variable

}

}

stage('stage-3'){

steps{

echo "stage-3"

echo "pushing docker image"

echo "NUMBER = ${NUMBER}" <-- in this case the value vill be original = 3, because blocks of code works as in java

} and we overrided the value in stage #2 not in #3

}

}

post{

always{

echo "post section"

}

}