DevOps

DevOps is a methodology which integrates the roles of Development (Development and Testing) and Operations teams by automating the process with the help of tools. It will help us to overcome the challenges in Dev and Ops teams in Agility model.

CD 🡪CI🡪CT🡪CD🡪CM

* Continuous Development: GIT
* Continuous Integration: Jenkins
* Continuous Testing: Selenium
* Continuous Deployment: Docker/Kubernetes or Puppet/Ansible
* Continuous Monitoring: Nagios

DevOps Life Cycle -

**Plan – Coding – Building – Testing – Release – Deploy – Operations – Monitoring**

Plan – Which tools are required?

Coding – Tools like Git for Version Control System of the code

Building – Build the code using tools like Maven

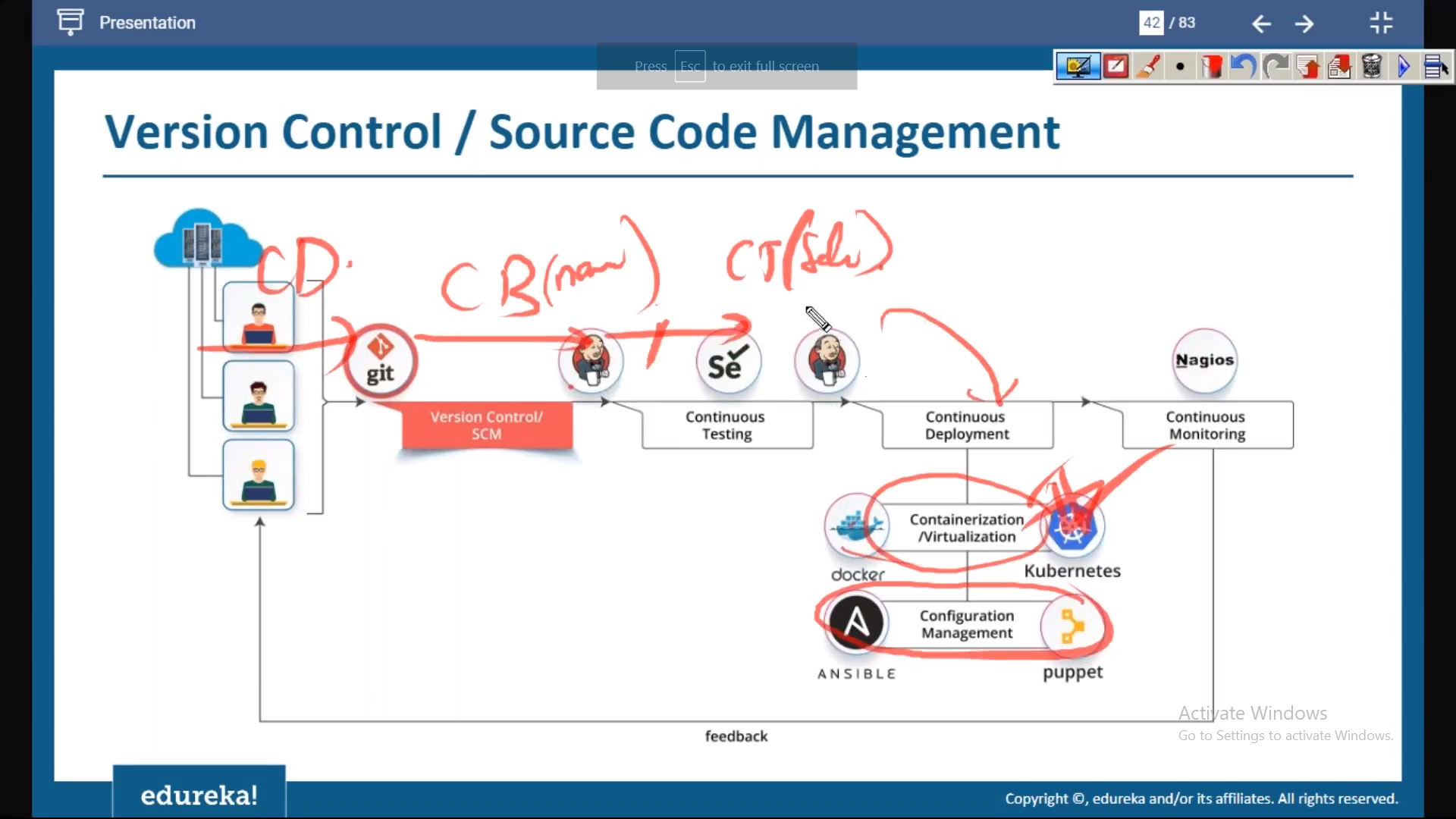
Testing – Test the code using tools like Selenium

Release – Release the code using tools like Jenkins

Deploy – Deploy the code using the tools like

Operations –

Monitoring -

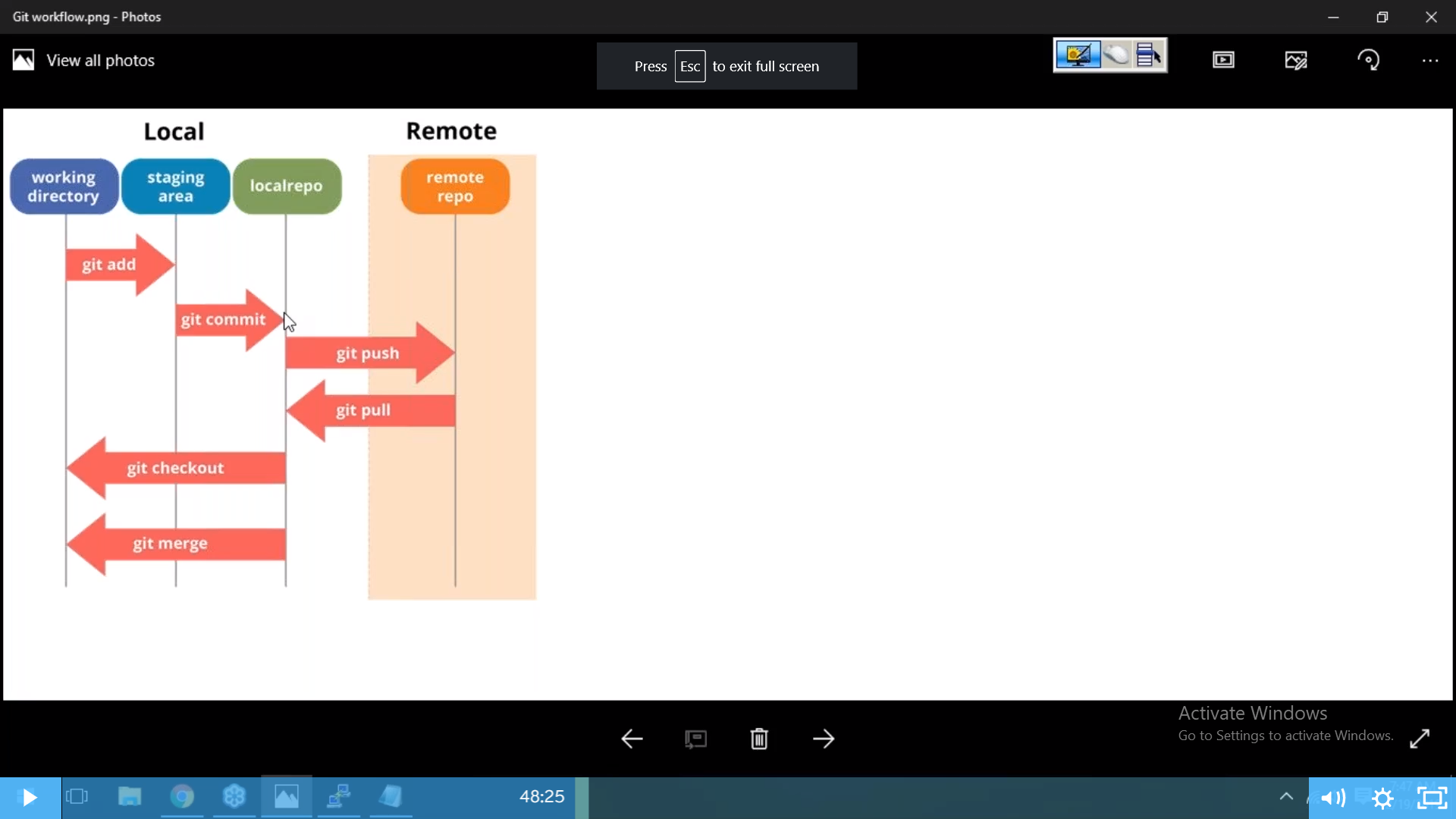


Explanation for above –

1. Developers commit the code in Git 🡪 Continuous Development
2. Jenkins would run the jobs for Building the code (using Build tools like Maven) 🡪 Continuous Building/Integration
3. Once the Build is ready, Jenkins would run the jobs for testing the code (tools like Selenium) 🡪 Continuous Testing
4. Once the testing is completed and no bugs reported, Jenkins would run the job for deploying the code on production environment using the tools like below – 🡪 Continuous Deployment
   1. Docker (Containerization for creating the packages of the code) – Here applications would be first deployed into Containers.
      1. Kubernetes – For managing the Containers.
   2. Deploy the packaged code on servers automatically using configuration management tools like ANSIBLE/Puppet
5. Once the code is deployed on production environment, monitor the code using tools like Nagios 🡪 Continuous Monitoring

Git&Github

***GIT Architecture as below -***



**Git Commands as below -**

git init

git status

git add <file name>

git commit -m “<message>”

git ls-files

git show

git commit -a -m “<message>”

git log –oneline 🡪 Shows all commits done with ID

git diff <filename>

git diff --staged <filename>

git rm <filename> 🡪 removes file from both working directory and local repository

girt rm –cached >filename> removed the file from local repository only. The file will remain in working directory

vim .gitignore

git revert <IDNAME>

git reset –hard <IDNAME> 🡪 makes the ID specified as Header and previous logs and files are lost in local repository

git branch 🡪 lists out the branches in the repo.

git checkout -b <BranchName> 🡪 creates and switch to new branch

git checkout <branchname> 🡪 switch to new branch

git merge <sourcebranch> <destinationbranch> 🡪 merge files from one branch to another branch

git stash -> to move some work in progress files to temporary shelf in order to not to commit the files in working directory of those.

git stash list 🡪 shows the stash ID

git stash pop <stashID> removes the stash and make it available in working directory again as files to be added/committed

git stash apply <stashID> keep the stash available and make it available in working directory again as files to be added/committed

git stash clear 🡪 Deletes all the stashes

or

git stash drop <StashID>

git stash -p 🡪 partially stash the files

git rebase <branch name> 🡪 it will not create the new commit as HEAD from parent branch in b2 branch and sits linear for easy understanding of log history of commits

example:

Branch Master –

Commit B

Commit A

New branch is created as b2 from master, which has the commits as below –

Commit B

Commit A

Now, new commits were made on both the branches i.e. master and b2. New commits look like as below in both the branches –

Branch master –

* Commit C
* Commit B
* Commit A

Branch b2 –

* Commit D
* Commit B
* Commit A

If we apply git rebase master (assume that you are in b2 branch) the log history on b2 branch looks like as below –

* Commit D
* Commit C
* Commit B
* Commit A

Note: If we do merge i.e. git merge master b2, then Commit C will be shown as <HEAD> in b2 branch.

**Github** 🡪 It is an online version control system to host git like local repositories.

git remote add origin <https://github.com/rajeshsrinivasa1981/newrepo1.git> (it will get connected to remote repo)

git push origin <branchname> (it will ask user credentials for pushing the files from local repo to remote repo)

git pull origin <branchname> 🡪 It pulls the changes from remote to local (fetch + merge)

git clone <https://github.com/rajeshsrinivasa1981/newrepo2.git> --> creates the new repository locally for the first time from remote.

git fetch origin 🡪 it shows verbose of if any changes are not present in locally when compared to remote.

Jenkins

Advantages of continuous integration –

* Improves Quality: Improves quality by running multiple unit tests and analyzing any static code
* Increases Productivity: Automating building of code saves lot of time, thereby increasing productivity
* Reduces Risk: Eliminate the risk of potential human errors by automating the test

Various CI tools are – Jenkins, Gitlab CI, CodeShip, Bamboo, TeamCity, Travis CI

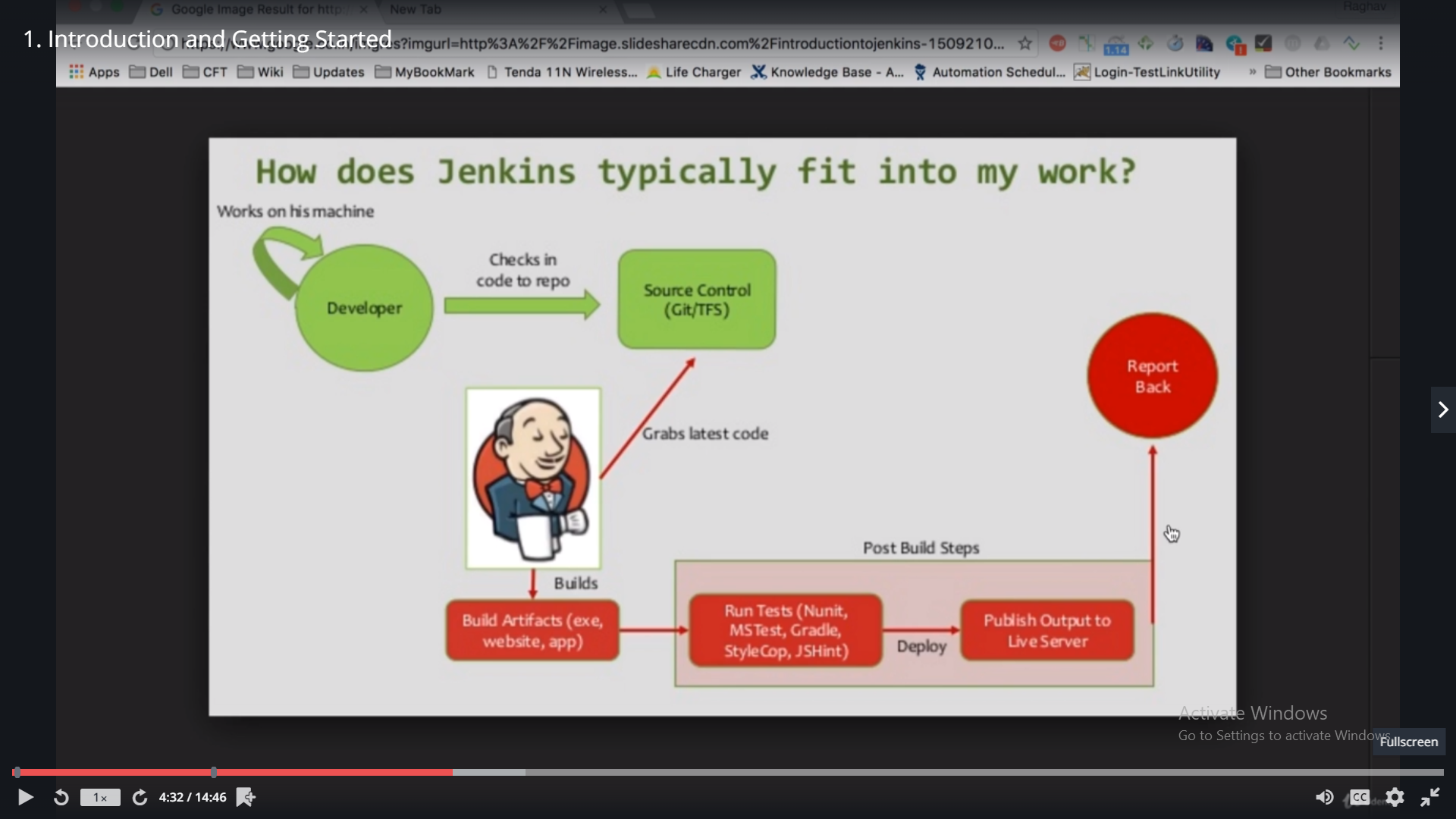
Note: Jenkins is the market leader among all various tools available in the market.

Continuous Integration in Agile: Develop the code – Build Automation – Test Automation – Manual Deployment - Feedback/Report (Without DevOps in Agile)

Continuous Integration in DevOps: Develop the code – Automate the Build – Automate the Test – Automate the Deployment – Feedback/Report

* Jenkins is a Java application
* It is used for Continuous Integration and Continuous Delivery

Let’s say if developers are committing the code to repository multiple times in a day, at the end of the day Jenkins will try to build the code and the Build fails. It is very difficult to find out which code has caused the error occurring in the build generation. During this scenario Jenkins comes into picture - Whenever developer commits the code to Version Control Systems tools (Like Git/TFS), Jenkins will automatically grab the latest code and trigger the build. If the build gets failed, we would come to know which code has caused the error. If the Build gets successful, then Jenkins will automatically run the units/functional tests to ensure the no code break/functional defects encountered in the build and sends the report back to the team. We would come to know if the build has any defects or not immediately once the code is committed to VCS. This process is called Continuous Integration and Continuous Delivery.

* 

Download Jenkins.war from - <https://jenkins.io/download/>

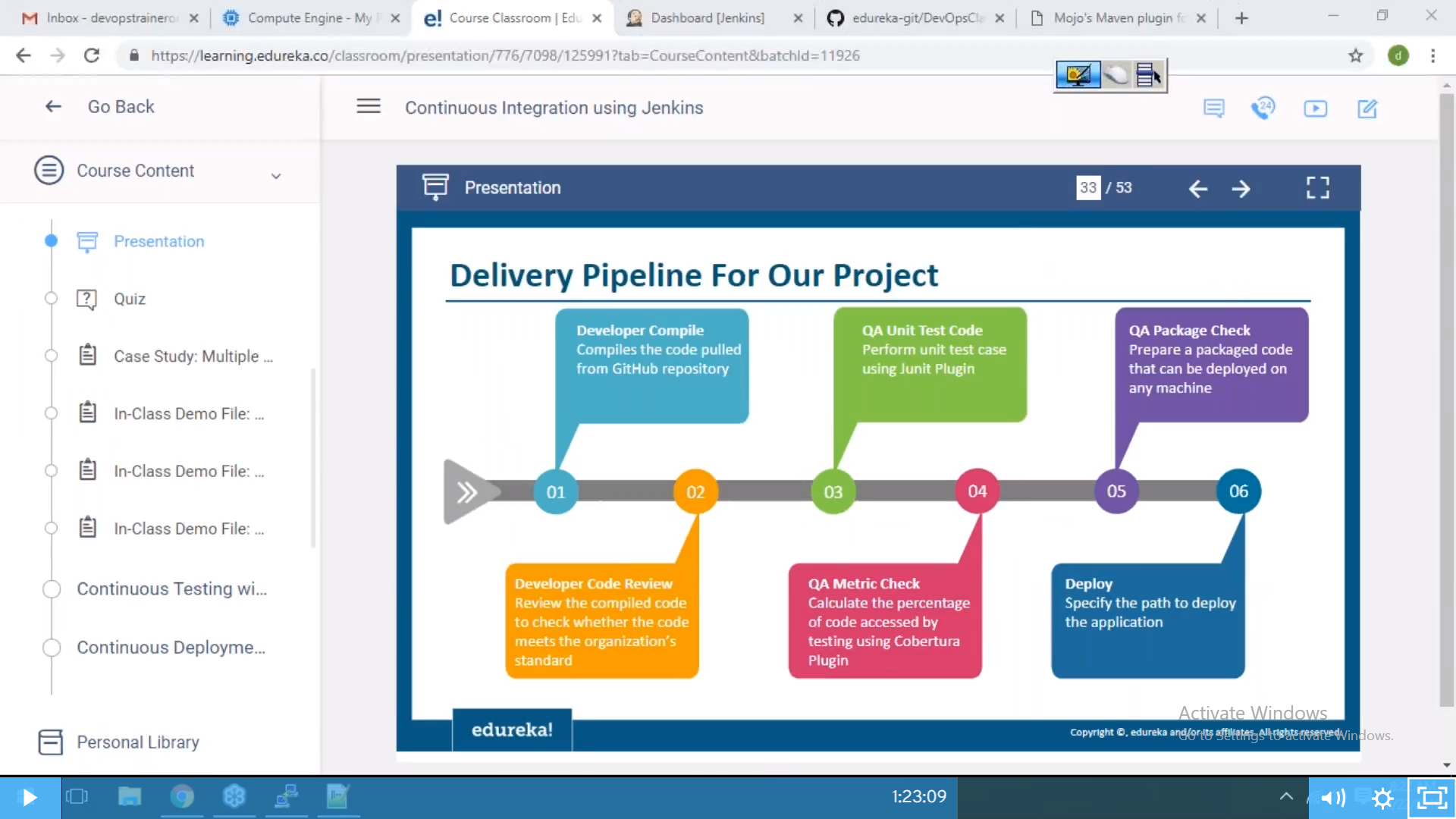
Go to command prompt and go to the location where “jenkins.war” file is located.

Run the following command – “java -jar Jenkins.war”

Once the installation is complete, the following message is displayed - Jenkins is fully up and running

Demo Delivery Pipeline for DevOps project –

***Compile Job – Code Review Job – Unit Test Job – MetricCheck job (Code coverage analysis) - Package***



Permissions -

Matrix based authorization: Provide permissions to the user based on their role.

Project based permissions – Based on the jobs

Email notifications – Sends email notifications based on the status of the jobs

Scheduling –

* timer : Depending on the timer set
* pollscm – Github, If any commit happens in Github, systems triggers the job in Jenkins for compiling or testing or deploying the code. (cd /var/lib/Jenkins/workspace/job<1 or 2 or 3> )
* pipeline:

Maven is a Build automation tool which is used to automate different phases of the project. (compile, code review, analysis, test)

If the project has to be executed through **Maven** you require POM.xml file. POM contains version details, executable file details, additional file details etc.. POM is like an Heart for the Maven to execute the code.

Below are the goals for the Jenkins job –

* compile – goal for compile
* -P metrics pmd:pmd – goal for code review (Program Mistake Detector)
* test – goal for unit test cases
* cobertura:cobertura -Dcobertura.report.format=xml
* package
* Master – Slave concept in Jenkins – If we want to run the build in different environments/different machines we create the slave to the master by creating the jobs in master machine.

Note: Slave machines should have the necessary tools installed except Jenkins i.e. Git, Java, Maven etc.…

* Pipeline as a code –

Docker

***File 🡪 Images 🡪 Container (From files we create images, from images we create containers)***

* sudo apt-get -y update - To get the latest updates
* sudo apt-get install -y docker - To install the docker
* sudo service docker start – Start the docker
* sudo adduser ec2-user (This will add the user)
* sudo usermod -a -G docker <ec2-user> (This will add the user to the group.Logout and Login once again so that sudo need not be given time when you type docker commands)
* sudo service docker stop – Stop the docker
* sudo apt-get remove docker – Uninstall the docker
* Basic:
* docker version: This gives information about docker client and server.
* docker -v or docker --version: This gives the version of the docker
* docker info: Detailed information about docker (i.e. # of containers – running, stopped, paused, # of images etc..)
* docker --help: >docker images --help>
* docker login:
* Images –
* docker images
* docker pull
* docker rmi
* Containers
  + docker ps
  + docker run
  + docker start <container ID>: This will start the container
  + docker stop <container ID>: This will stop the container
* System
  + docker stats
  + docker system df
  + docker system prune

Containers –

* docker run –name <Container Name> <Image name>: This will create a name for the container
* docker ps -a: This will display the list of containers available.
* docker start <container Name/ID>: This will start the container
* docker stop <container Name/ID>: This will stop the container
* docker pause <container name/ID>: This will pause the container
* docker unpause <container name/ID>: This will unpause the container
* docker top <container name/ID>: This will show the process of container
* docker stats < container name/ID >: This will show the memory usage of container
* docker kill < container name/ID >: This will kill the running containers
* docker rm < container name/ID >: This will remove the containers
* docker history <Image Name>: This will show the history of the image
* apt-get install docker – Install the docker
* systemctl start docker – Start the docker
* docker pull <image name> -- It will pull the image to local server
* docker run –name <name of the container> <image name> -- To specify user defined container names
* docker images – Shows the list of images in the docker
* docker ps – shows the list of active containers
* docker ps -a – show the list of all active and inactive containers
* docker push <image name> -- It will push to docker hub (To push you need to have credentials. Authentication required, type **docker login**)
* docker tag <Old image name> <New Image name> -- It will create a copy of old image with a new image name
* Registry/username/repository: tag name – display of an image name (docker.io/rs042424/hello-world:<1>)

***Note: username is optional, if you do not specify tag name system will consider the latest tag name available)***

* docker search <windows> --- will display the list of images available and which has “OK” in column Official are official images.
* Docker run -it <Image Name>: This will go inside container as an interactive terminal mode. In order make this container up and running and exit from this mode (CTRL+P & CTRL+Q)

***Docker attach <container ID> 🡪 This command also will go inside the container.***

***Inside the container if you want to update –***

* *Type “apt-get update”*
* *For vim installation type “apt-get install vim -y”* 
  + Difference between docker run and docker start:
* docker run will always create new containers
* docker start will start the previously existing stopped containers.

Docker Volumes – Docker volumes are used to store the data of containers on Host or share the data between container to Host or Host to Container or Container to Container

**docker run -it --name test1 -v data:/voldir ubuntu**

* -v is volume flag
* data is the volume created on host machine
* voldir is the volume created on container and they both are mapped by using SourceVol:DestVol

docker volume ls 🡪 It will list out the volumes in the host machine

docker volume inspect <data> 🡪 It will display the content in the volume

Creating Images –

Two way of creating images –

1. Create an Image from running container -

Create a container with name “forcommitcontainer” in which it has VIM installed in it. Run the following command in host for creating an image - docker commit forcommitcontainer rs042424/ubuntu:withVim

1. From a Docker File

* Create a directory and create an text file as below sample statements –
  + From ubuntu
  + Maintainer "Rajesh"
  + RUN apt-get update
  + RUN apt-get install vim -y
  + CMD /bin/echo "Hello from ME"
* Run the following command docker build -t rs042424/ubuntu:dockerfile.txt . for creating an image

1. Port forwarding/Exposed ports: docker run -itd -P <image name> (It will generate random port for the host to access the applications in container)

* docker inspect <Image name> 🡪 It should have exposed ports listed i.e. 8080
* docker inspect <container ID> 🡪 to know the host port
* Now access the application in the container on host using the following address in web: Host IP Address:random port generated for the container (<http://54.193.51.153:32768/>)
* If you want to access the application on your own port then run the command for the container generation as docker run -itd -p 9999:8080 <image name> (docker run -itd -p 9999:8080 tomcat) – here 8080 is an exposed port for that image and 9999 is user own port)
* docker export <container name/ID> > <zipfilename.tar> i.e. docker export forcommitcontainer > update.tar (If you want to zip the container in a tar file)
* docker import - <new image> < <zipfilename.tar> i.e. docker import – newimage < update.tar (If you want to create an image from zip file)
* If you want to save an image in tar file use the following command
  + docker save -o <zipfilename.tar> Repository:tagname i.e. docker save -o update1.tar ubuntu:docekrfile
  + docker load <zipfilename.tar> 🡪 It will create an image

1. Docker Toolbox –
   1. Notes –
      * Docker will run only on Linux machines and Windows machines of version 10 and above.
      * If you want to run Docker on Mac or Win(less than 10 version), you need to install Docker Toolbox.
        + Even though docker is installed