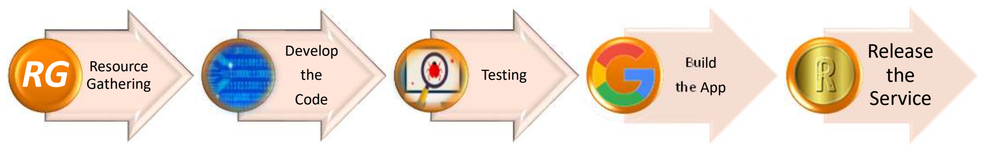
*DevOps*

NIPUNA TECHNOLOGIES

## What is Devops

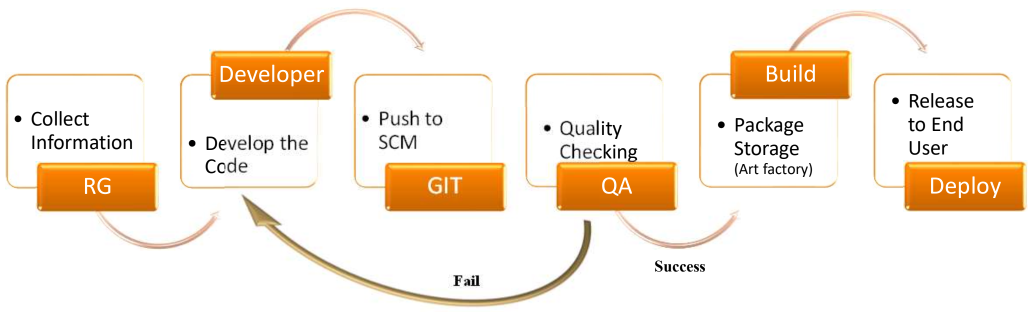
Resource gathering → Develop the code → Test → Build → Release

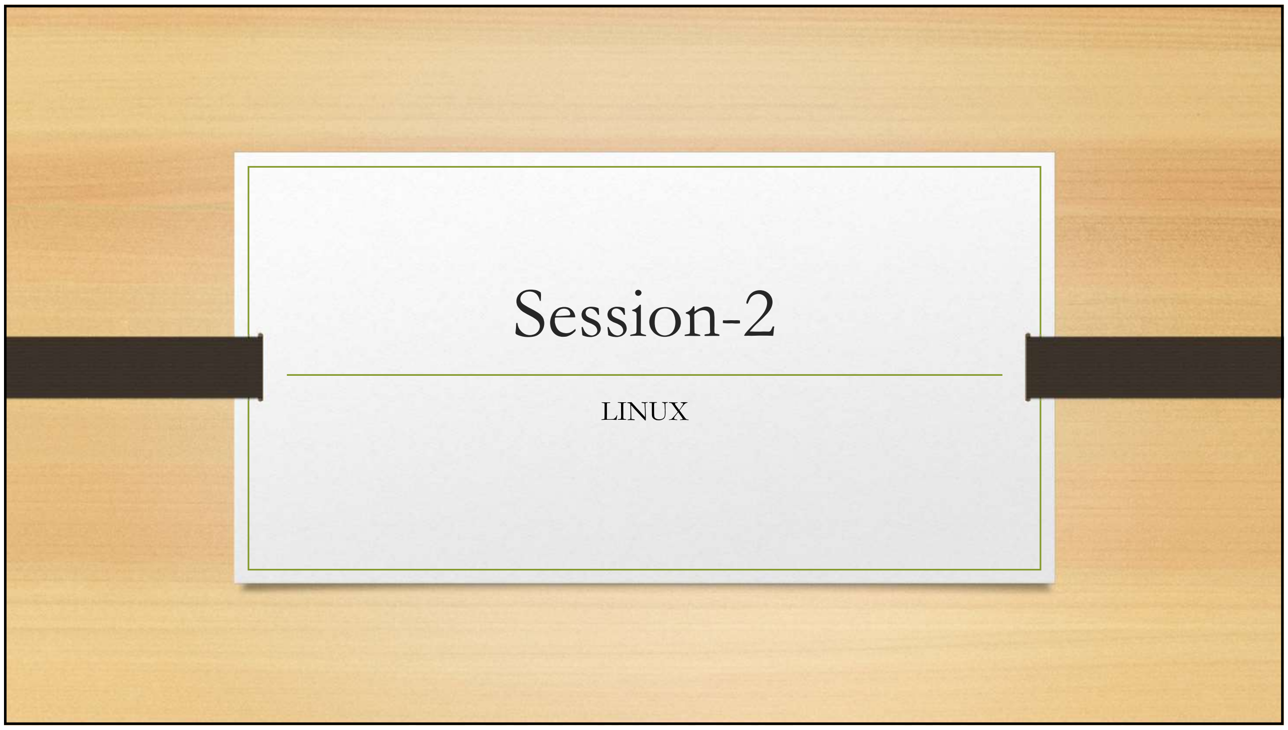


*DevOps*

NIPUNA TECHNOLOGIES

## Devops workflow





Session-2

LINUX

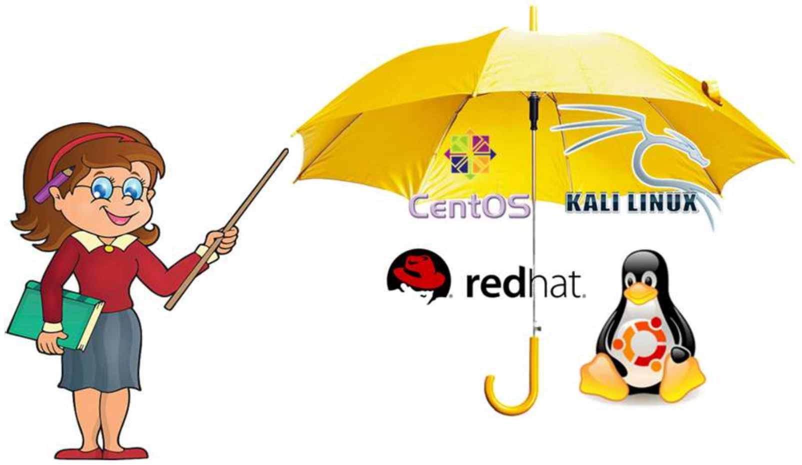
DevOps

NIPUNA TECHNOLOGIES

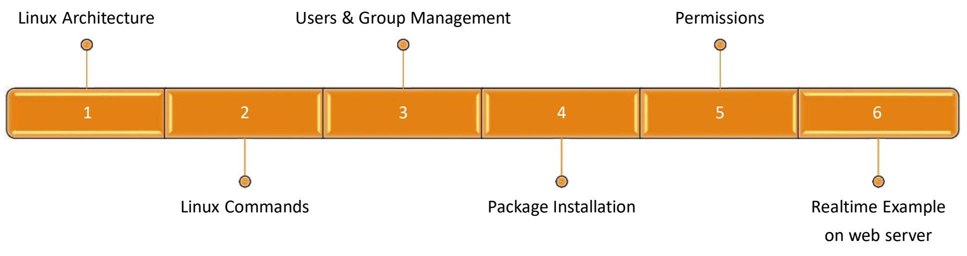
Linux

OPERATING SYSTEM

## What is Linux ?



## About Linux



*DevOps*

NIPUNA TECHNOLOGIES

## Linux Architecture

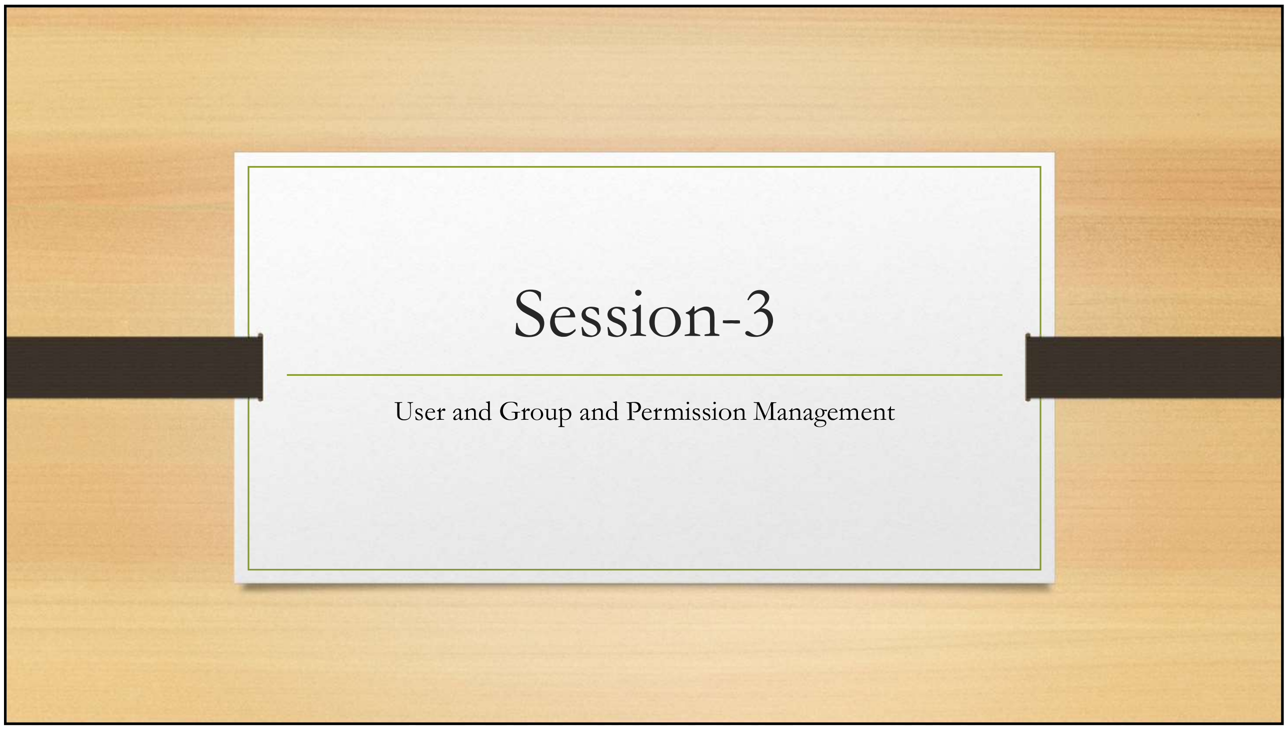
- Linux is Typical terminal interface
- User create a file ie. 1.txt
- Shell is the interface
- Kernel is hardware and it will check and run Hardware and provide output

*DevOps*

NIPUNA TECHNOLOGIES

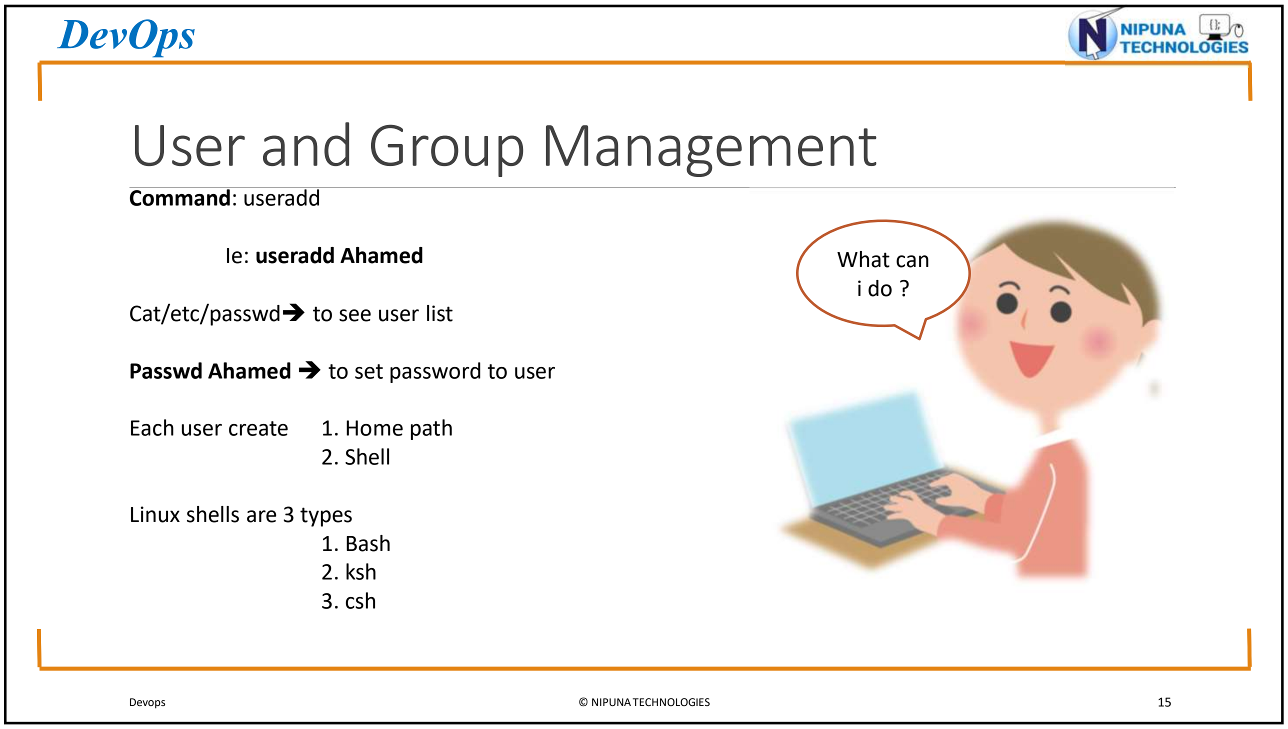
## Linux Commands

| Command | Command | Command |
|---|---|---|
| touch | top | uname –r |
| mkdir | ps –ef | vi/nano/gedit |
| ls –l | ps aux | cd test |
| echo | \| (pipe) | # ➔ super user,$➔ normal user |
| move | pwd | man touch |
| cp | cd | history |
| cat | cd.. | date |
| cat > | who am i | time |
| cat >> | find | clear |
| rm | uname –a | |



## Session-3

User and Group and Permission Management



DevOps

NIPUNA TECHNOLOGIES

## User and Group Management

**Command**: useradd

Ie: **useradd Ahamed**

Cat/etc/passwd➔ to see user list

**Passwd Ahamed** ➔ to set password to user

Each user create 1. Home path
2. Shell

Linux shells are 3 types
1. Bash
2. ksh
3. csh

Devops © NIPUNA TECHNOLOGIES 15

## User and Group Management

1. I can change my Home path
   ie: usermod –home/tmp Ahamed

2. I can change my shell
   ie: usermod –shell/bin/ksh Ahamed

3. I can set Password Expiry date
   ie: usermod –e 2021-11-10 Ahamed

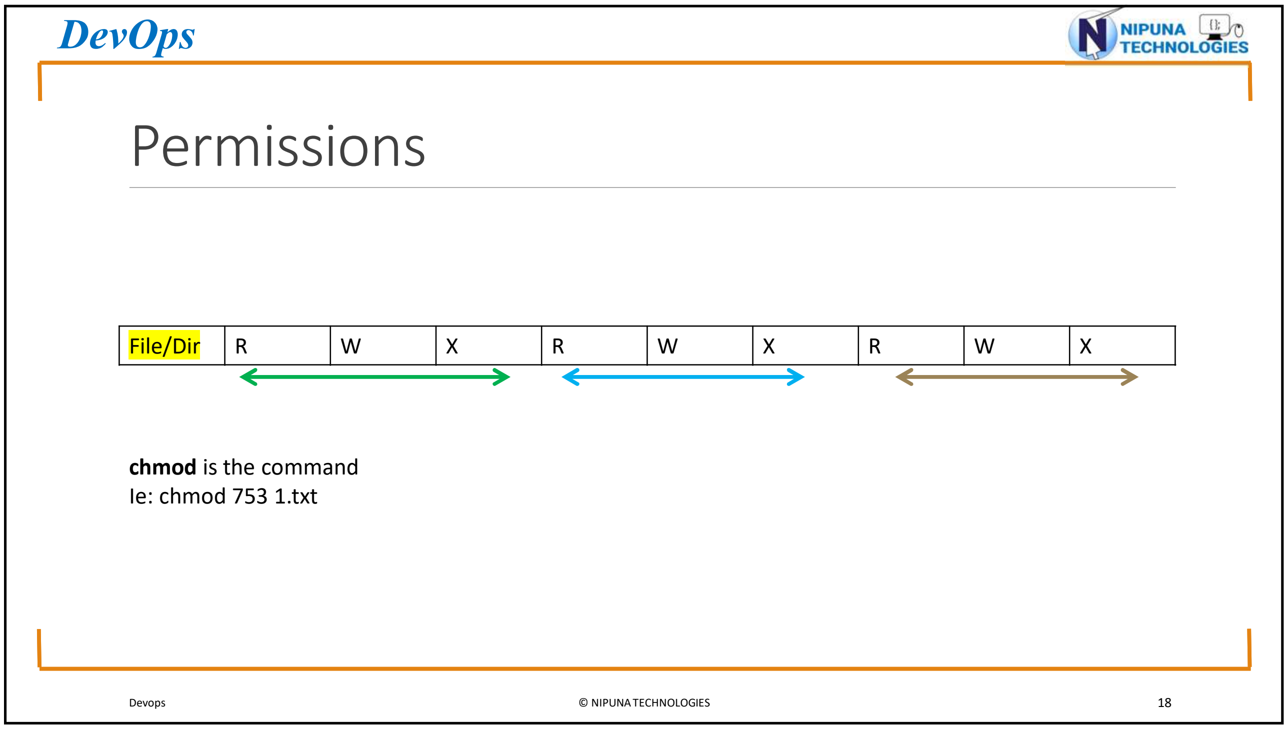
   change -l ➔ to check

## Permissions

- Absolute permissions
- Symbolic permissions

**Permissions**

| | | |
|---|---|---|
| Read | -r | -4 |
| Write | -w | -2 |
| Execute | -x | -1 |



DevOps

NIPUNA TECHNOLOGIES

## Permissions

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| File/Dir | R | W | X | R | W | X | R | W | X |

**chmod** is the command
Ie: chmod 753 1.txt

Devops © NIPUNA TECHNOLOGIES 18

## Session-4

Creating Linux in aws & Connect Linux

*DevOps*

## Creating Linux in aws

- Create aws Account in http://www.aws.amazon.com
- Login to aws console using aws (root/iam) account
  - Go to Services
    - Compute
      - EC2
- EC2 Dashboard
  - Instance
    - Launch instance

  Amazon Elastic Compute Cloud - User Guide for Linux Instances

*DevOps*

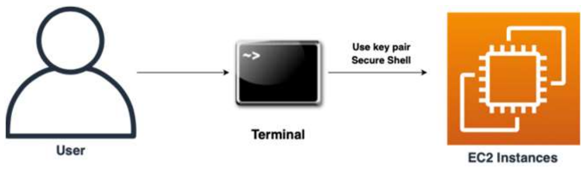
## Connect Linux Server

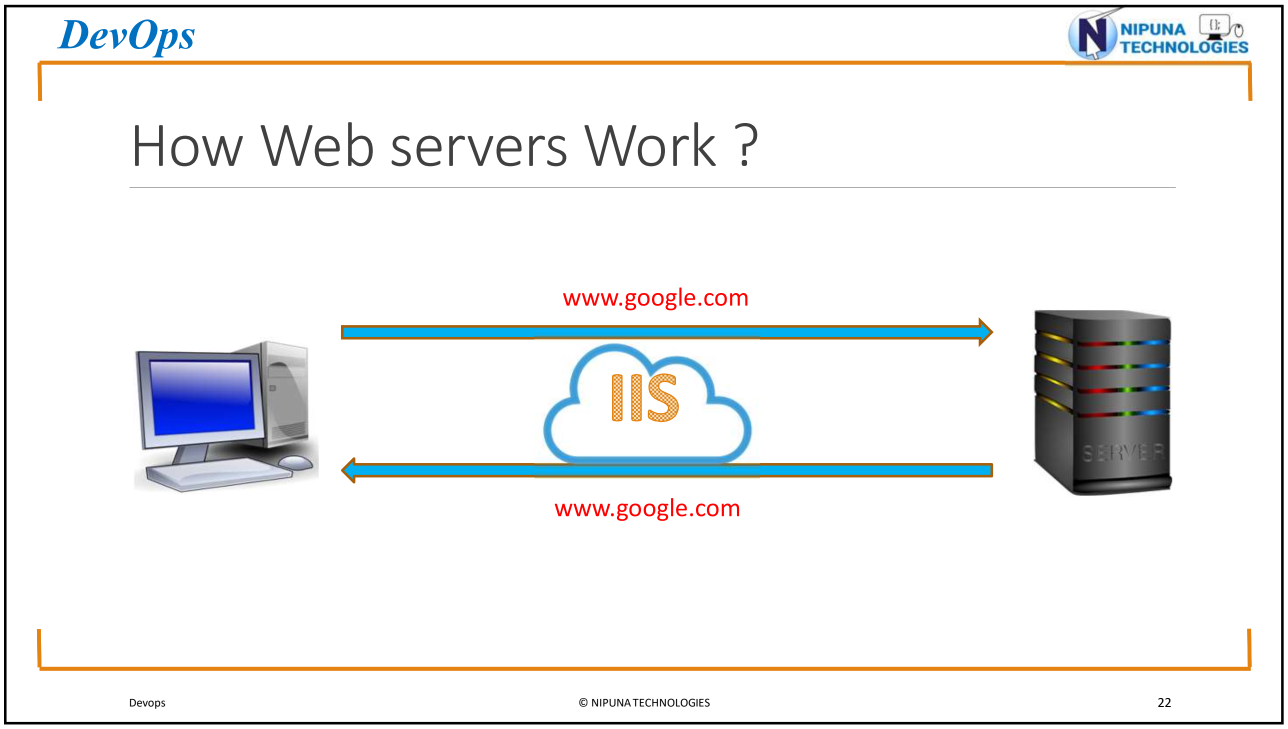
- Download and install **gitbash** (Terminal) in your windows computer
- Open gitbash
  - Type “ssh –I <pemfile name> <username>@<instance public-ip>

    ie: ssh –I linux.pem ec2-user@3.17.185.14

    $sudo su

    #(now in super user mode)

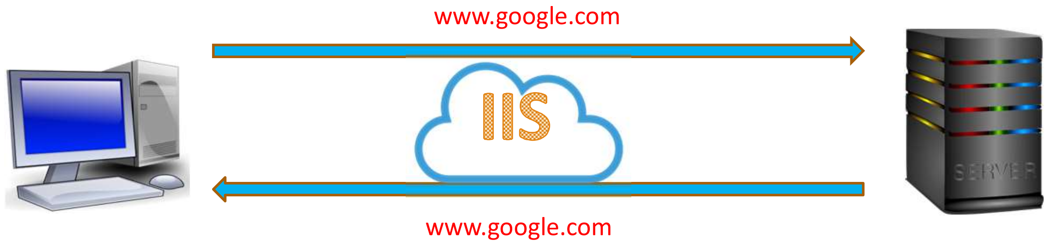




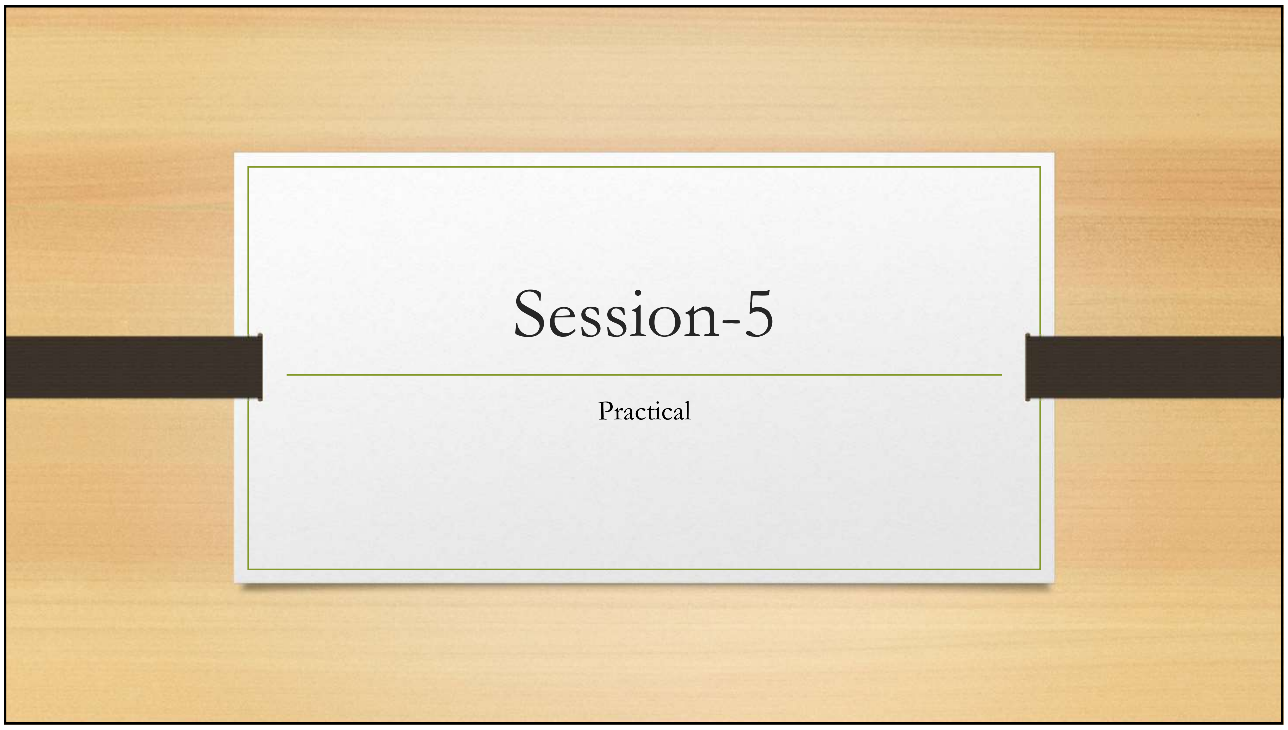
*DevOps*

NIPUNA TECHNOLOGIES

## How Web servers Work ?

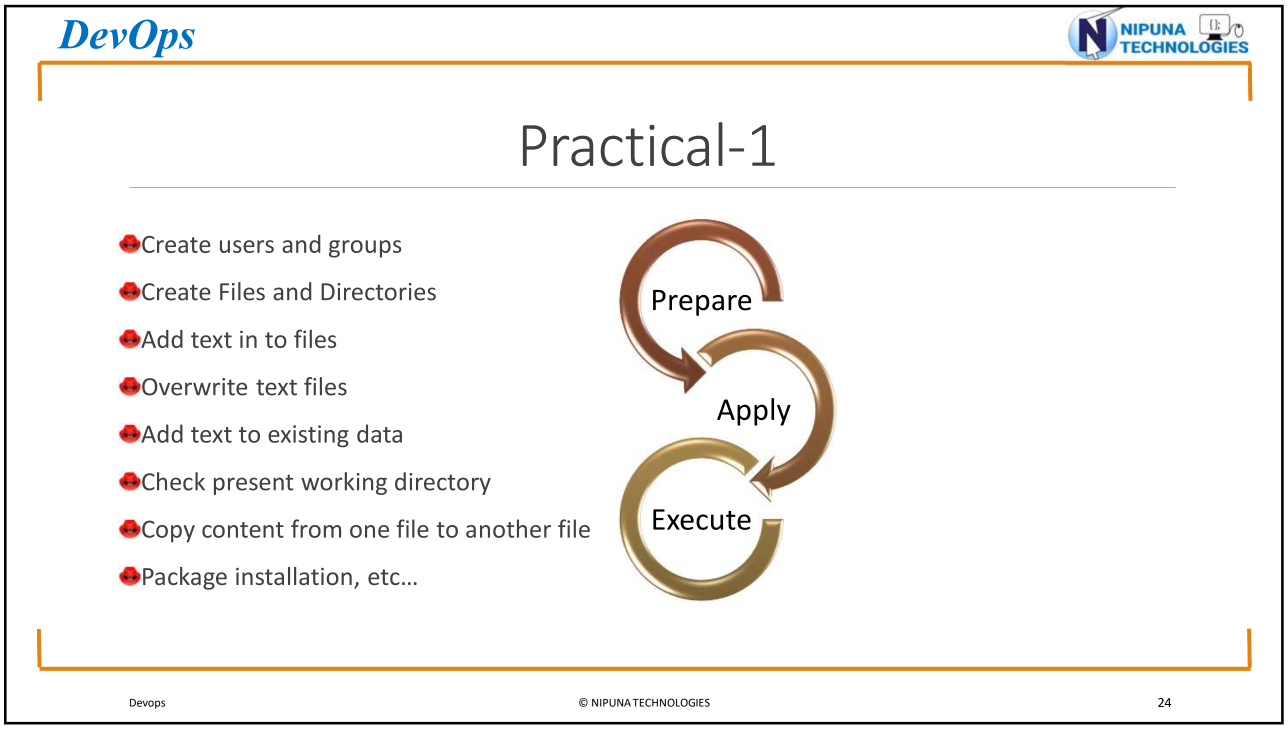


Devops © NIPUNA TECHNOLOGIES 22



## Session-5

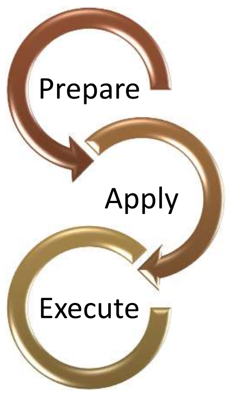
Practical



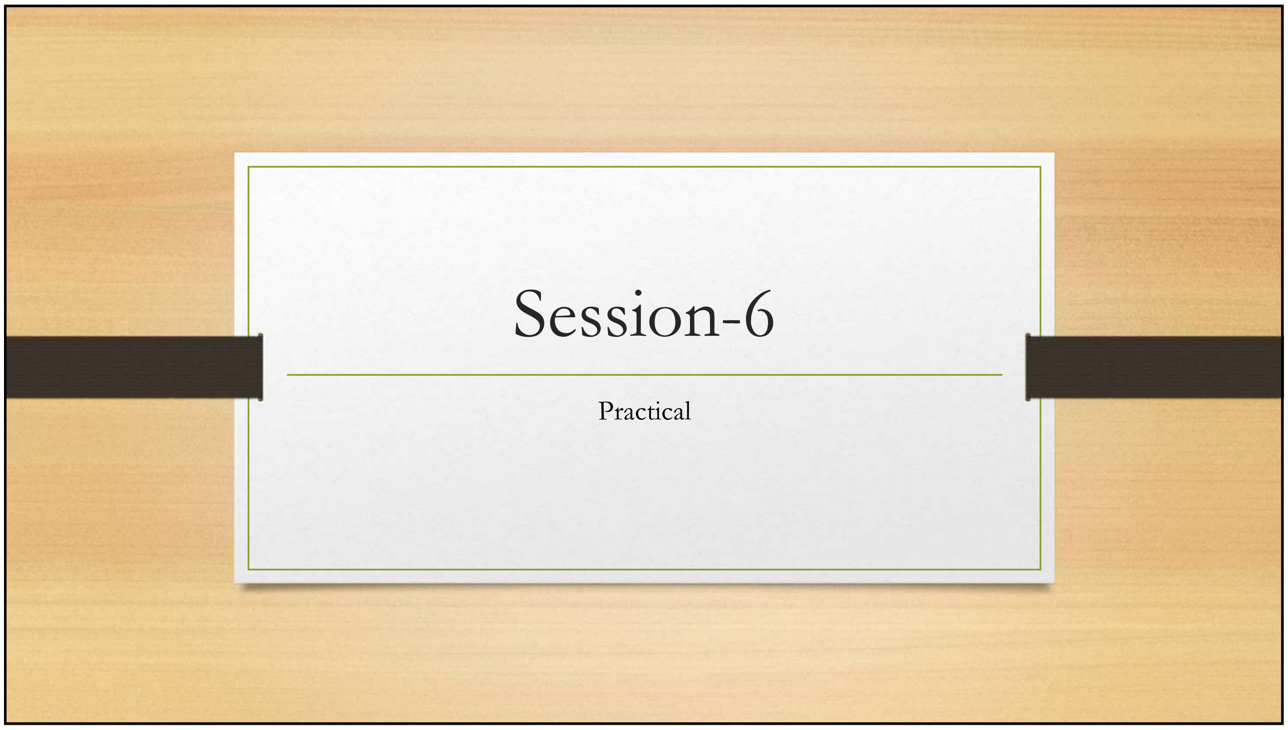
DevOps

## Practical-1

- Create users and groups
- Create Files and Directories
- Add text in to files
- Overwrite text files
- Add text to existing data
- Check present working directory
- Copy content from one file to another file
- Package installation, etc…

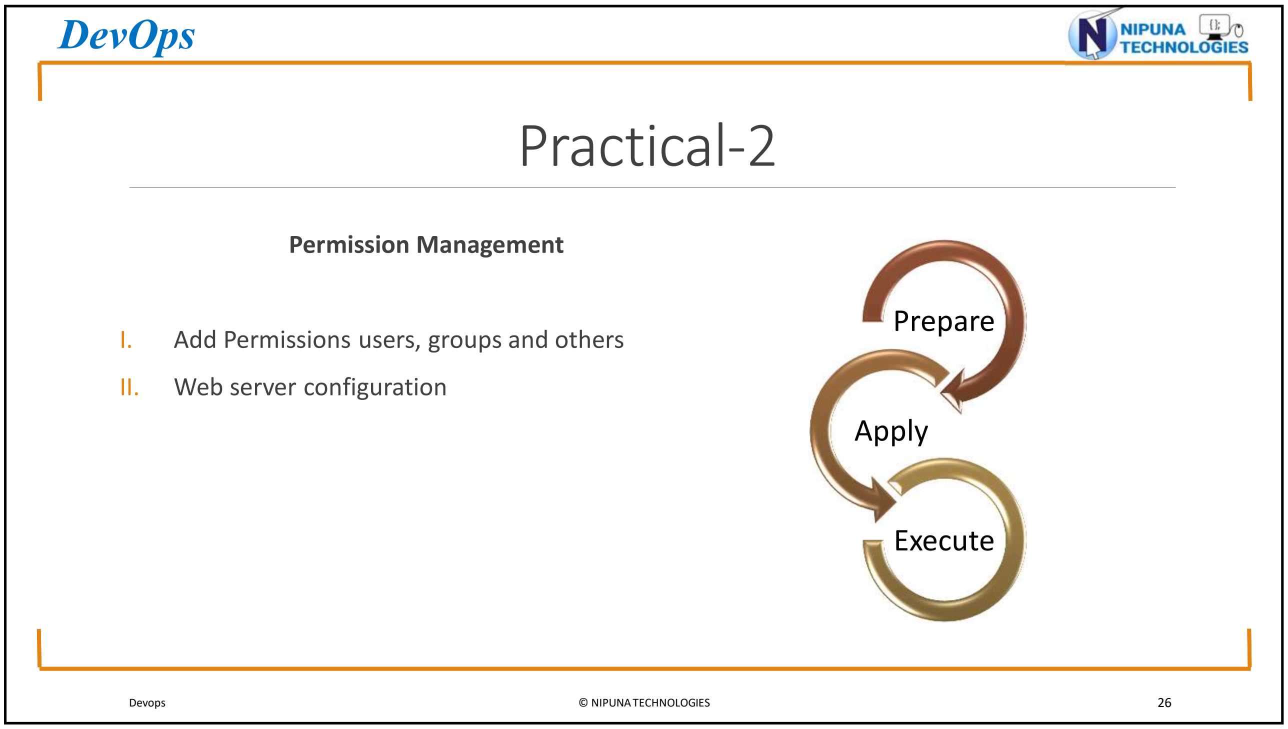


Devops © NIPUNA TECHNOLOGIES 24



## Session-6

Practical



DevOps

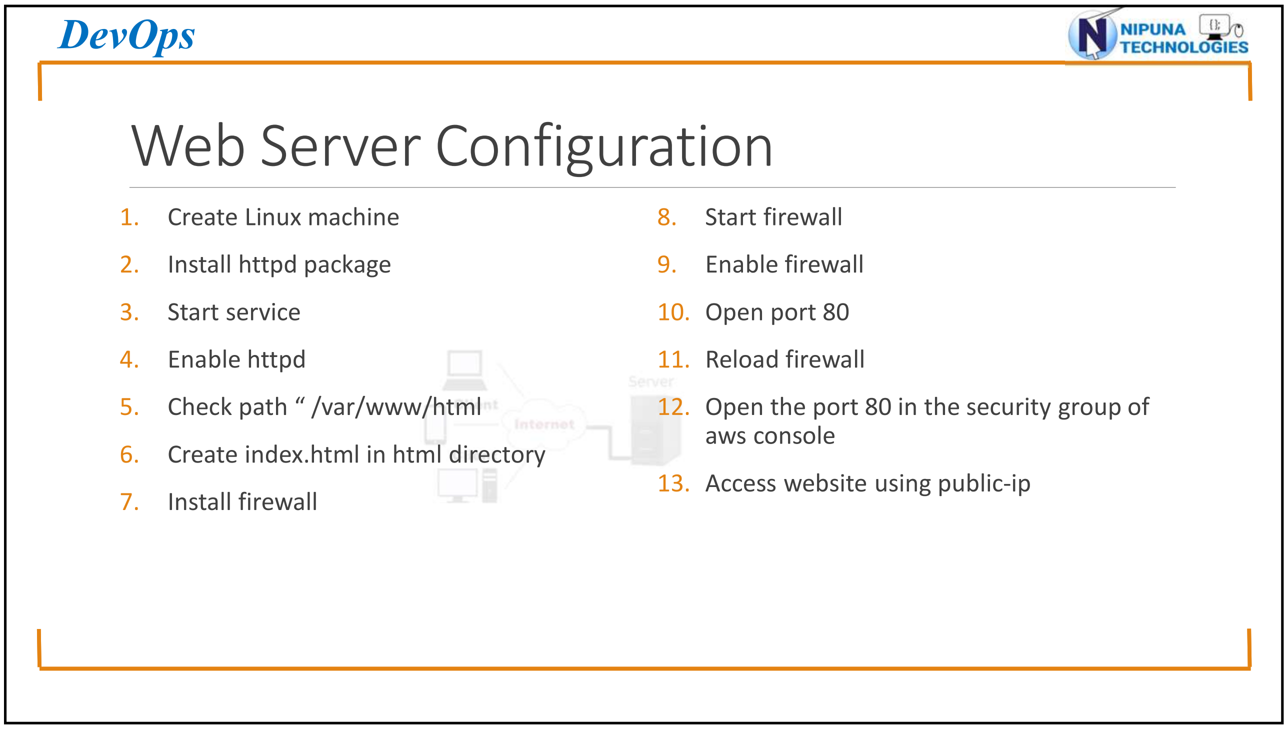
NIPUNA TECHNOLOGIES

## Practical-2

**Permission Management**

I. Add Permissions users, groups and others
II. Web server configuration

Devops © NIPUNA TECHNOLOGIES 26



DevOps

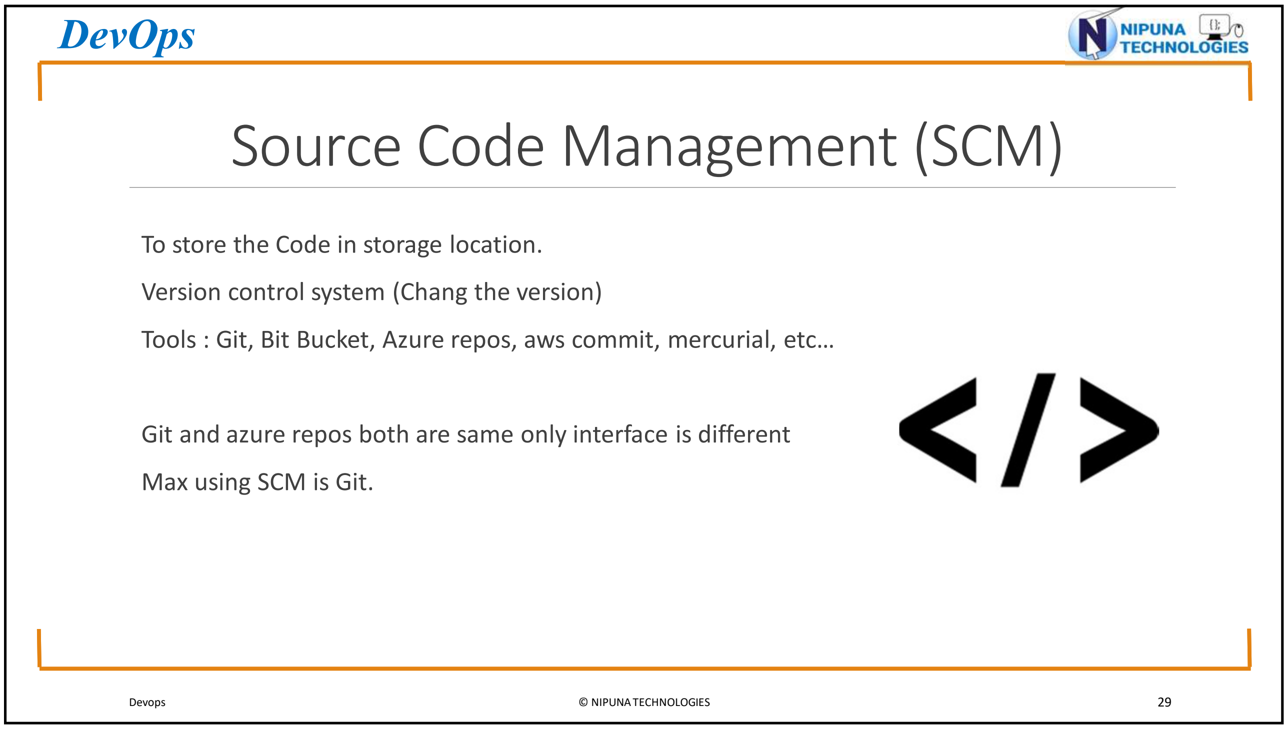
NIPUNA TECHNOLOGIES

## Web Server Configuration

1. Create Linux machine
2. Install httpd package
3. Start service
4. Enable httpd
5. Check path " /var/www/html
6. Create index.html in html directory
7. Install firewall
8. Start firewall
9. Enable firewall
10. Open port 80
11. Reload firewall
12. Open the port 80 in the security group of aws console
13. Access website using public-ip

# Session-7

Source Code Management



## Source Code Management (SCM)

To store the Code in storage location.

Version control system (Chang the version)

Tools : Git, Bit Bucket, Azure repos, aws commit, mercurial, etc…

Git and azure repos both are same only interface is different

Max using SCM is Git.

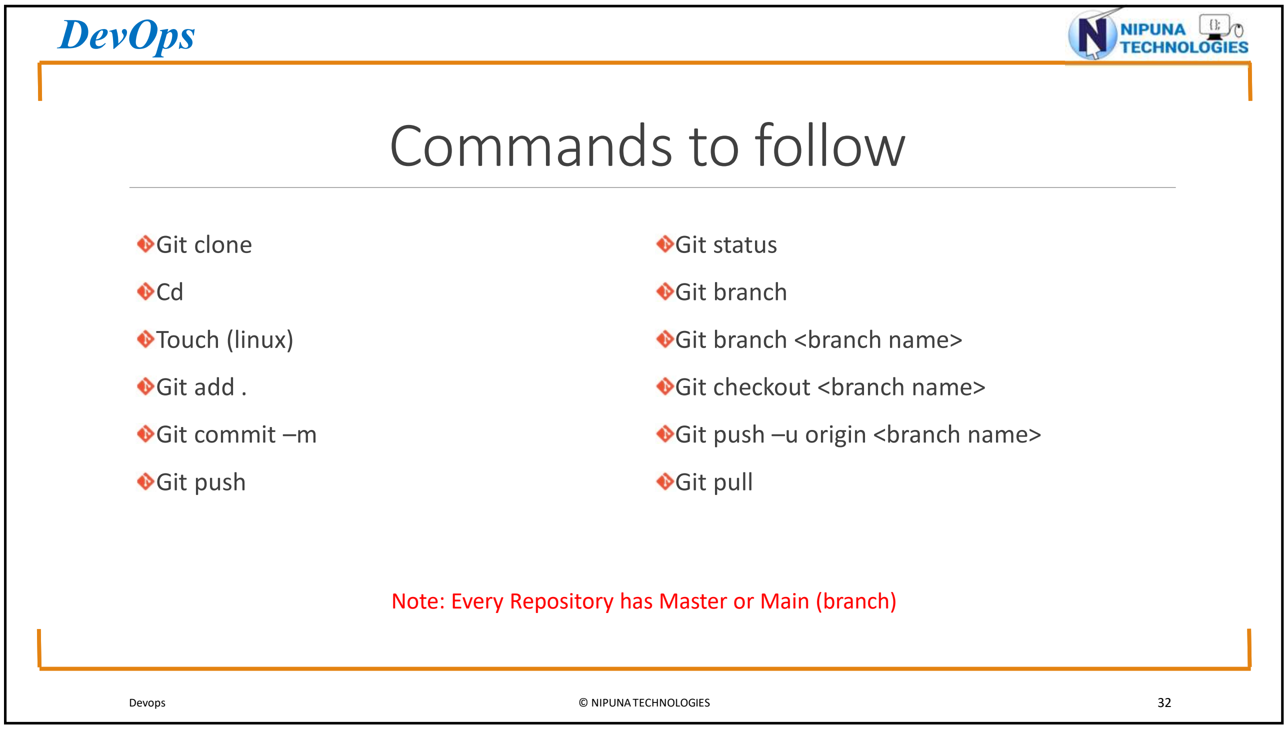
## Git

Git Repository

- Git Local Repository
  - It's Created in Local machine
- Git Remote Repository
  - It's Hosted in centralized

## Git
http and ssh url's

- Git clone
- Create folder
  - using 'cd' command
- Create files
  - With respect project requirement
- Stagging area (publish)
  - #git add . (to add all files)
  - #git add 1.txt (to add only one file
- Verification
  - #git status
- Commit
  - #git commit –m "file description"
- Push
  - #git push



*DevOps*

## Commands to follow

- Git clone
- Cd
- Touch (linux)
- Git add .
- Git commit –m
- Git push
- Git status
- Git branch
- Git branch <branch name>
- Git checkout <branch name>
- Git push –u origin <branch name>
- Git pull

Note: Every Repository has Master or Main (branch)

## Session-8

Source Code Management

git

DevOps

NIPUNA TECHNOLOGIES

## Git Practical

Create git account in github.com

Loin in to github account

Create your own Repository

Clone your Repository in your git server

Create files in local Repository and push in to Remote Repository

Check files in your Remote Repository

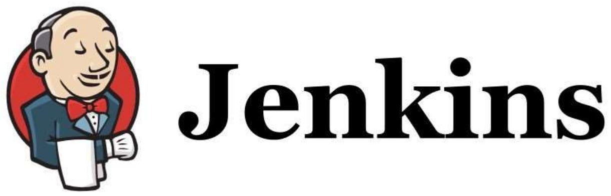
Devops © NIPUNA TECHNOLOGIES 34

## Session-9

Jenkins

*DevOps*

## Continuous Integration (CI)



*DevOps*

## Agenda

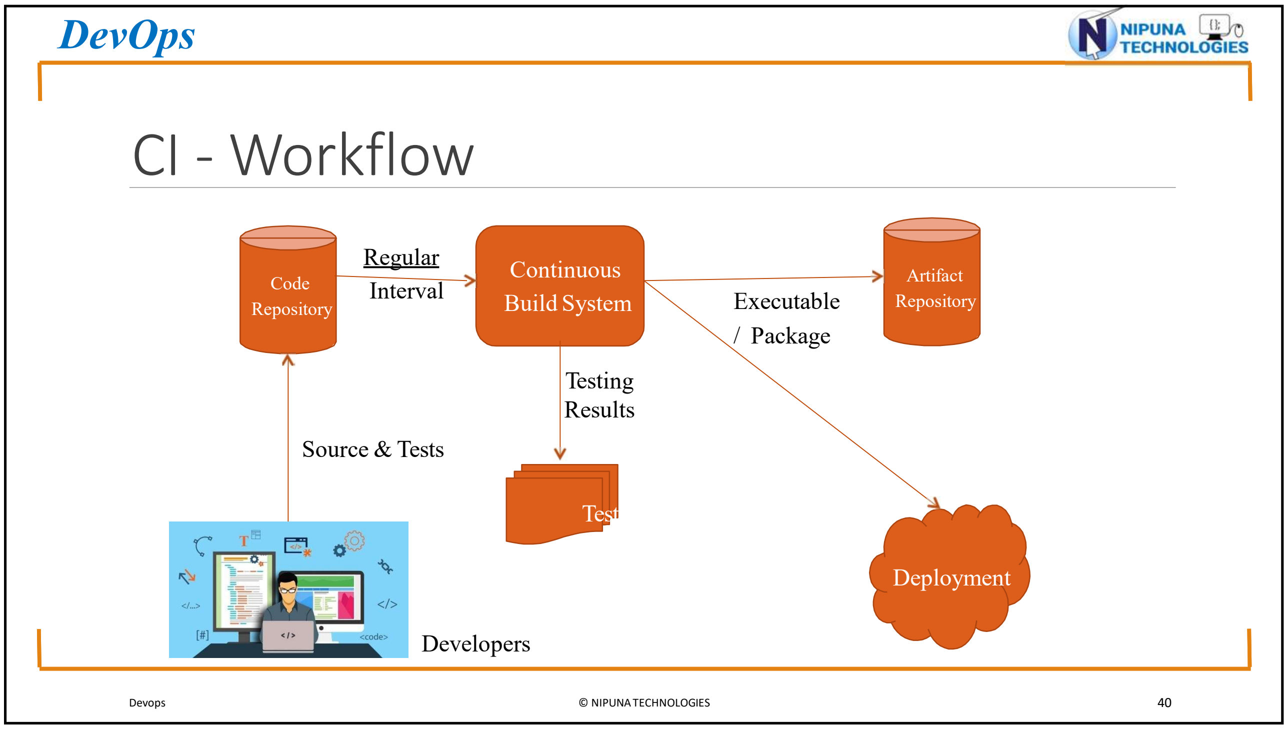
- ❑ Continuous Integration (CI)
  - ❑ What is it?
  - ❑ What are the benefits?
  - ❑ Continuous Build Systems
- ❑ Jenkins
  - ❑ What is it?
  - ❑ Where does it fit in?
  - ❑ Why should I use it?
  - ❑ What can it do?
  - ❑ How does it work?
  - ❑ Where is it used?
  - ❑ How can I get started?
- ❑ Putting it all together
- ❑ Conclusion

## CI - Defined

"Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily-leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible"

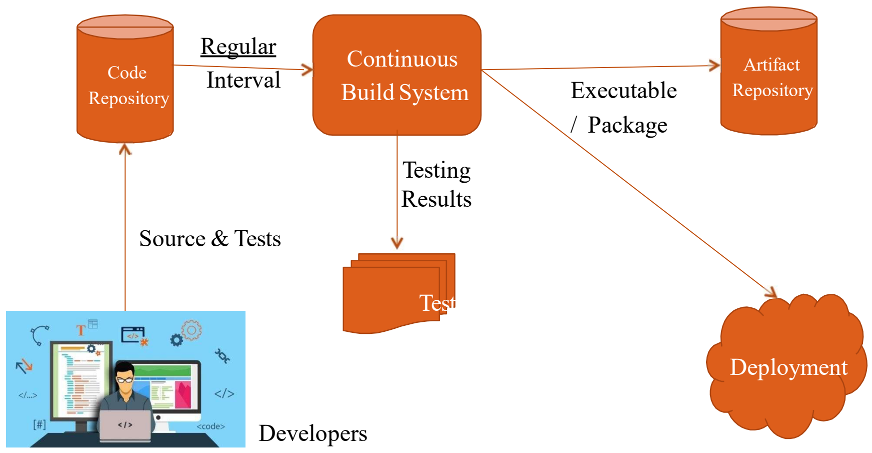
## CI – What does it really mean?

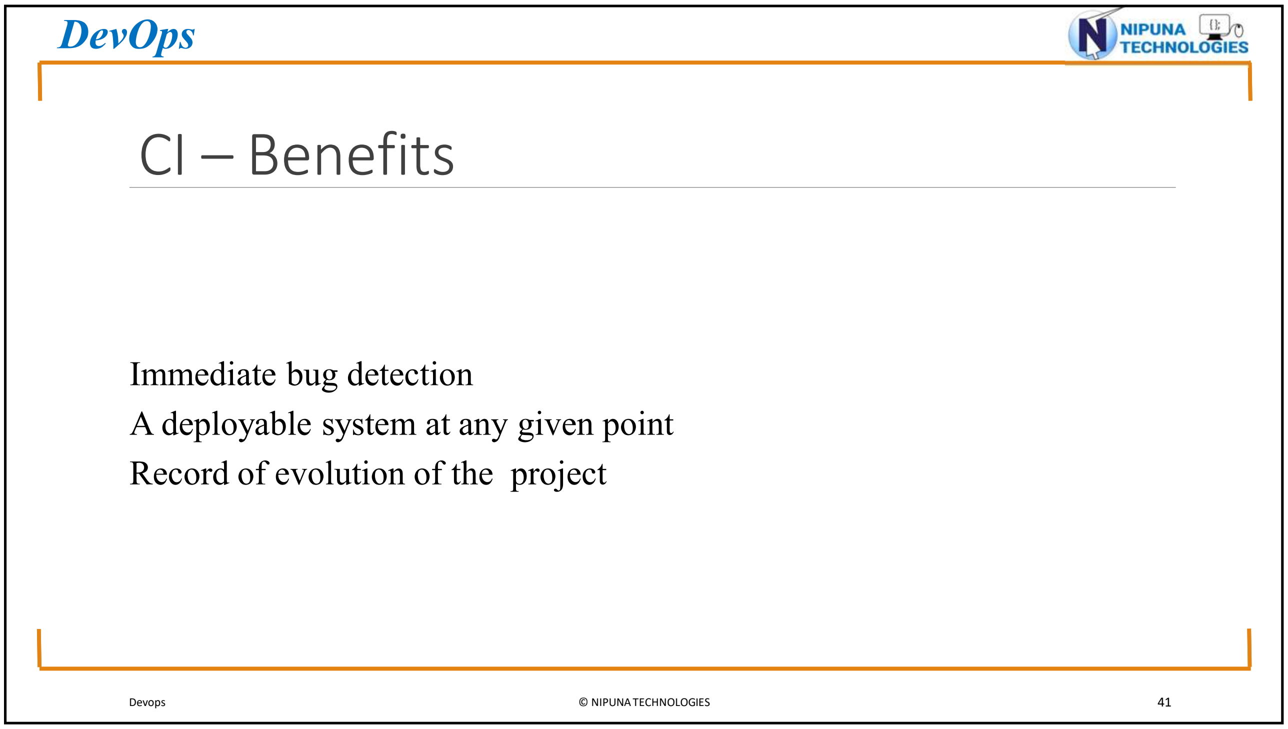
- At a regular frequency (ideally at every commit), the system is:
  - Integrated
    - All changes up until that point are combined into the project
  - Built
    - The code is compiled into an executable or package
  - Tested
    - Automated test suites are run
  - Archived
    - Versioned and stored so it can be distributed as is, if desired
  - Deployed
    - Loaded onto a system where the developers can interact with it



DevOps

## CI - Workflow





DevOps

## CI – Benefits

Immediate bug detection

A deployable system at any given point

Record of evolution of the project

## CI – The tools

- Code Repositories
  - SVN, Mercurial, Git
- Continuous Build Systems
  - **Jenkins**, Bamboo, Cruise Control
- Test Frameworks
  - JUnit,Cucumber, CppUnit
- Artifact Repositories
  - Nexus, Artifactory, Archiva

## Jenkins

Branched from Hudson

Java based Continuous Build System

Runs in servlet container

  Glassfish, Tomcat

Supported by over 400 plugins

  SCM, Testing, Notifications, Reporting, Artifact Saving,Triggers, External Integration

Under development since 2005

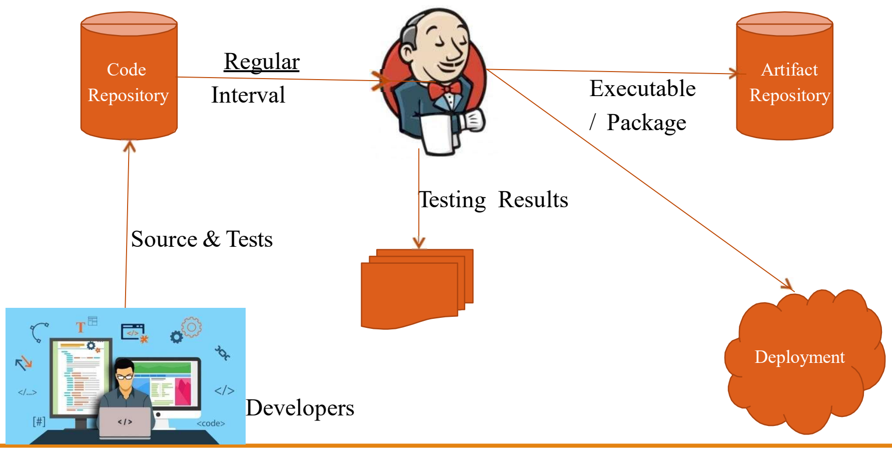
http://jenkins-ci.org/

## Jenkins - History

2005 - Hudson was first release by Kohsuke Kawaguchi of Sun Microsystems

2010 – Oracle bought Sun Microsystems
Due to a naming dispute, Hudson was renamed to Jenkins
Oracle continued development of Hudson (as a branch of the original)

## Jenkins – Fitting in



## Why Jenkins? Flexibility!

Jenkins is a highly configurable system by itself

The additional community developed plugins provide even more flexibility

By combining Jenkins with Ant, Gradle, or other Build Automation tools, the possibilities are limitless

## Why Jenkins? Free/OSS

Jenkins is released under the MIT License

There is a large support community and thorough documentation

It's easy to write plugins

Think something is wrong with it? You can fix it!

## What can Jenkins do?

Generate test reports

Integrate with many different Version Control Systems

Push to various artifact repositories

Deploys directly to production or test environments

Notify stakeholders of build status

…and much more

## How Jenkins works - Setup

- When setting up a project in Jenkins, out of the box you have the following general options:
  - Associating with a version control server
  - Triggering builds
    - Polling, Periodic, Building based on other projects
  - Execution of shell scripts, bash scripts, Ant targets and Maven targets
  - Artifact archival
  - Publish JUnit test results and Javadocs
  - Email notifications
- As stated earlier, plugins expand the functionality even further

# Session-10

Jenkins

## How Jenkins works - Building

- Once a project is successfully created in Jenkins, all future builds are automatic
- Building
  - Jenkins executes the build in an executer
    - By default, Jenkins gives one executor per core on the build server
  - Jenkins also has the concept of slave build servers
    - Useful for building on different architectures
    - Distribution of load

DevOps

NIPUNA TECHNOLOGIES

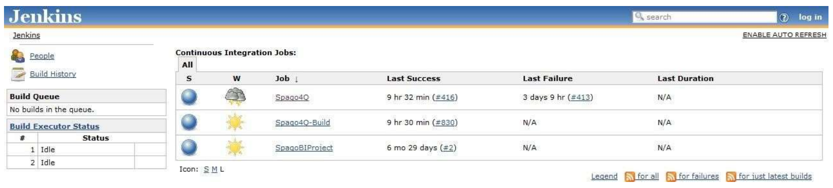
## How Jenkins works - Reporting

- Jenkins comes with basic reporting features
  - Keeping track of build status
    - Last success and failure
    - “Weather” – Build trend
- These can be greatly enhanced with the use of pre-build plugins
  - Unit test coverage
  - Test result trending
  - Find bugs, Checkstyle, PMD

DevOps

NIPUNA TECHNOLOGIES

## Jenkins by example – Main Page



Build Queue

No builds in the queue.

Build Executor Status

| # | Status |
|---|---|
| 1 | Idle |
| 2 | Idle |

Continuous Integration Jobs:

All

| S | W | Job ↓ | Last Success | Last Failure | Last Duration |
|---|---|---|---|---|---|
| ● | | Spago4Q | 9 hr 32 min (#416) | 3 days 9 hr (#413) | N/A |
| ● | | Spago4Q-Build | 9 hr 30 min (#830) | N/A | N/A |
| ● | | SpagoBIProject | 6 mo 29 days (#2) | N/A | N/A |

The main page provides a summary of the projects

- Quick view of
  - What’s building (“No builds in the queue”)
  - Build Executor Status (both “Idle”)
  - Status of the projects

DevOps

NIPUNA TECHNOLOGIES

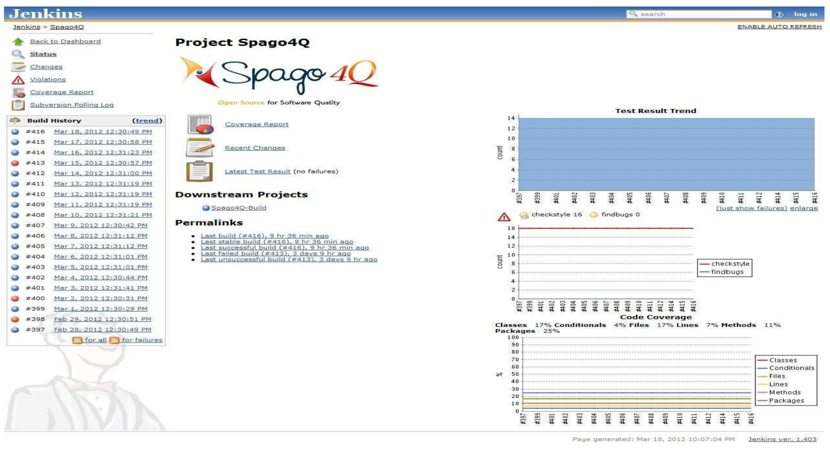
## Jenkins by example – Project Status

- Project status pages provide more details about a given project
  - The status of the last several builds
  - Charting (depending on plugins)
  - Dependencies

DevOps

NIPUNA TECHNOLOGIES

## Jenkins by example – Project Status



Jenkins » Spago4Q

Back to Dashboard
Status
Changes
Violations
Coverage Report
Subversion Polling Log

**Build History** (trend)

| | |
|---|---|
| #416 | Mar 18, 2012 12:30:49 PM |
| #415 | Mar 17, 2012 12:30:58 PM |
| #414 | Mar 16, 2012 12:31:23 PM |
| #413 | Mar 15, 2012 12:30:57 PM |
| #412 | Mar 14, 2012 12:31:00 PM |
| #411 | Mar 13, 2012 12:31:19 PM |
| #410 | Mar 12, 2012 12:31:19 PM |
| #409 | Mar 11, 2012 12:31:19 PM |
| #408 | Mar 10, 2012 12:31:21 PM |
| #407 | Mar 9, 2012 12:30:42 PM |
| #406 | Mar 8, 2012 12:31:12 PM |
| #405 | Mar 7, 2012 12:31:12 PM |
| #404 | Mar 6, 2012 12:31:01 PM |
| #403 | Mar 5, 2012 12:31:01 PM |
| #402 | Mar 4, 2012 12:30:44 PM |
| #401 | Mar 3, 2012 12:31:41 PM |
| #400 | Mar 2, 2012 12:30:31 PM |
| #399 | Mar 1, 2012 12:30:29 PM |
| #398 | Feb 29, 2012 12:30:51 PM |
| #397 | Feb 28, 2012 12:30:49 PM |

for all  for failures

**Project Spago4Q**

Coverage Report
Recent Changes
Latest Test Result (no failures)

**Downstream Projects**

Spago4Q-Build

**Permalinks**

- Last build (#416), 9 hr 36 min ago
- Last stable build (#416), 9 hr 36 min ago
- Last successful build (#416), 9 hr 36 min ago
- Last failed build (#413), 3 days 9 hr ago
- Last unsuccessful build (#413), 3 days 9 hr ago

checkstyle 16  findbugs 0

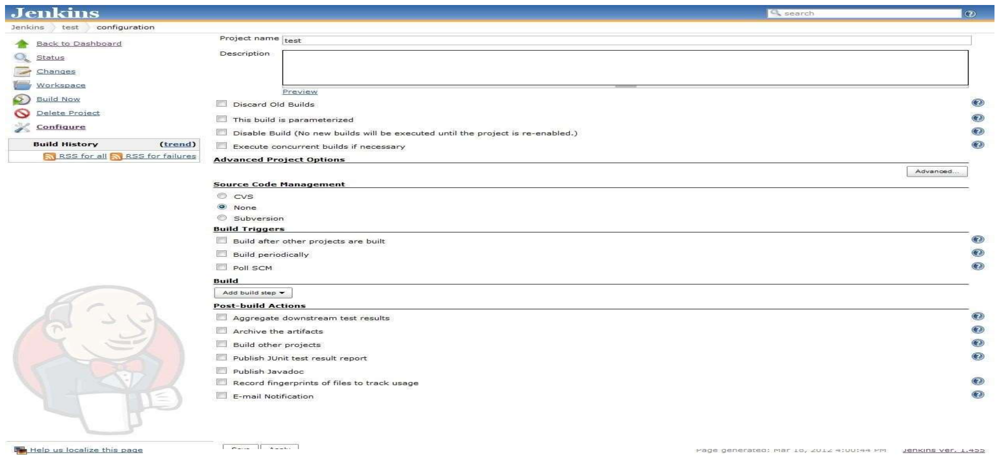
**Code Coverage**

Classes 17% Conditionals 4% Files 17% Lines 7% Methods 11% Packages 25%

Page generated: Mar 18, 2012 10:07:04 PM  Jenkins ver. 1.403

*DevOps*

## Jenkins by example – New Project



Project name: test

Description:

Preview

- [ ] Discard Old Builds
- [ ] This build is parameterized
- [ ] Disable Build (No new builds will be executed until the project is re-enabled.)
- [ ] Execute concurrent builds if necessary

**Advanced Project Options**

**Source Code Management**

CVS [✓ None] Subversion

**Build Triggers**

- [ ] Build after other projects are built
- [ ] Build periodically
- [ ] Poll SCM

**Build**

**Post-build Actions**

- [ ] Aggregate downstream test results
- [ ] Archive the artifacts
- [ ] Build other projects
- [ ] Publish JUnit test result report
- [ ] Publish Javadoc
- [ ] Record fingerprints of files to track usage
- [ ] E-mail Notification

*DevOps*

## Enhancing Jenkins

- Jenkins plugin system can enable a wide range of features including (but certainly not limited to)
- SCM
  - Mercurial, Git, Subversion
- Testing
  - Selenium,Windmill,TestLink
- Notifications
  - IRC,Twitter, Jabber
- Reporting
  - Doxygen, PMD, Findbugs
- Artifact Saving
  - Artifactory, Amazon S3, SCP
- Triggers
  - Jabber, Directory Watchers
- External Integration
  - GitHub, Bugzilla, JIRA
- And most importantly – The CI Game
  - A points based game where developers compete against each other to develop the most stable, well- tested code

## Who uses Jenkins?

LinkedIn
ebay
CARFAX
VEHICLE HISTORY REPORTS
NETFLIX
NASA
MICHELIN
BOEING
mozilla
FOUNDATION

## Running Jenkins yourself

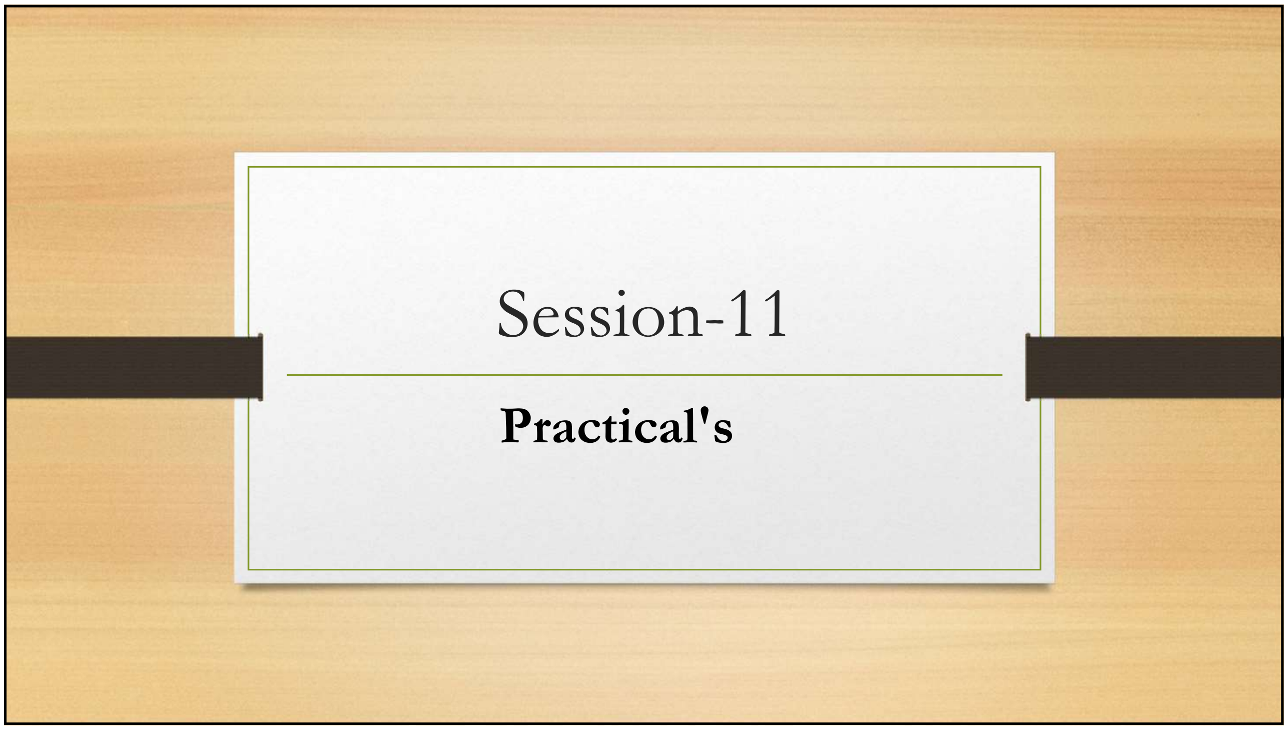
- ❑ Jenkins is packaged as a WAR, so you can drop it into whichever servlet container you prefer to use Jenkins comes pre-packaged with a servlet if you just want a light-weight implementation Native/Supported packages exist for
- ❑ Windows,
- ❑ Ubuntu/Debian,
- ❑ Redhat/Fedora/CentOS,
- ❑ Mac OSX,
- ❑ OpenSUSE,
- ❑ FreeBSD,
- ❑ OpenBSD,
- ❑ Solaris/OpenIndiana,
- ❑ Gentoo

## Running Jenkins yourself – Updates

- Jenkins has two release lines
- Standard releases
  - Weekly bug fixes and features
- Long-Term Support releases
  - Updates about every 3 months Uses a “Stable but older” version from the standard release line Changes are limited to backported, well-tested modifications

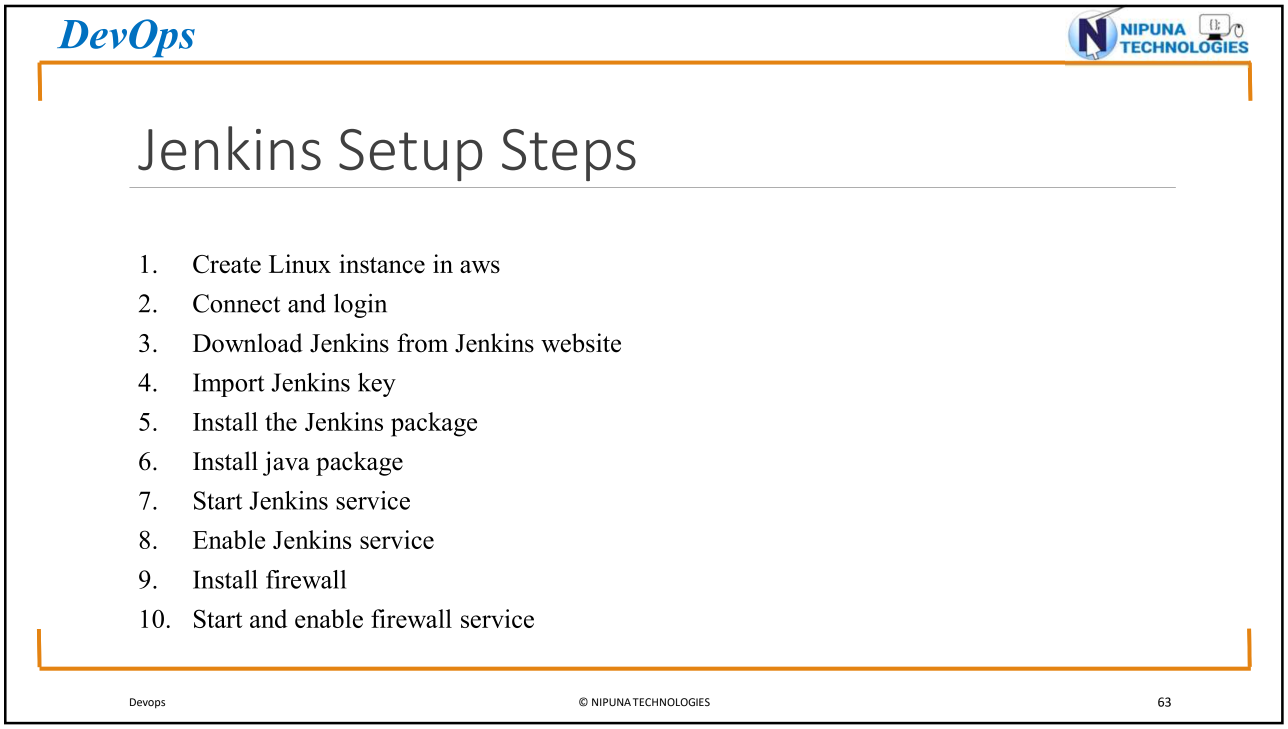
## Conclusion

- Continuous integration is a necessity on complex projects due to the benefits it provides regarding early detection of problems
- A good continuous build system should be flexible enough to fit into pre-existing development environments and provide all the features a team expects from such a system
- Jenkins, a continuous build system, can be an integral part of any continuous integration system due to it’s core feature set and extensibility through a plugin system



## Session-11

**Practical's**



*DevOps*

NIPUNA TECHNOLOGIES

## Jenkins Setup Steps

1. Create Linux instance in aws
2. Connect and login
3. Download Jenkins from Jenkins website
4. Import Jenkins key
5. Install the Jenkins package
6. Install java package
7. Start Jenkins service
8. Enable Jenkins service
9. Install firewall
10. Start and enable firewall service

Devops © NIPUNA TECHNOLOGIES 63

*DevOps*

NIPUNA TECHNOLOGIES

## Jenkins Setup Steps

11. Open port '8080' and 'http' service using firewalls
12. Access the Jenkins dashboard with the help of public-ip of your Jenkins mechine ie: http://public-ip:8080
13. Unlock the Jenkins using "cat" key & copy on the dashboard to unlock
14. Go with the option of suggested plugins in the Jenkins
15. Create user account to access the Jenkins
16. Access the Jenkins dashboard with the help of http://public-ip:8080

*DevOps*

NIPUNA TECHNOLOGIES

## Jenkins Setup in Linux

**Practical**

1. wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
2. rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key
3. yum install epel-release
4. sudo amazon-linux-extras install epel -y
5. amazon-linux-extras install java-openjdk11 -y
6. yum install jenkins -y
7. systemctl start jenkins
8. systemctl enable jenkins
9. systemctl status jenkins

## Jenkins Setup in Linux

10. yum install firewall* -y
11. systemctl start firewalld
12. systemctl enable firewalld
13. systemctl status firewalld
14. firewall-cmd --zone=public --add-port=8080/tcp --permanent
15. firewall-cmd --zone=public --add-service=http --permanent
16. firewall-cmd --reload
17. firewall-cmd --list-all
18. Open browser-->type <publicip>:8080-->login with username and passwoard.
19. copy /var/lib/jenkins/secrets/initialAdminPassword --->path
20. in Linux--> #cat /var/lib/jenkins/secrets/initialAdminPassword Copy code and paste into jenkins browser and install plugins.

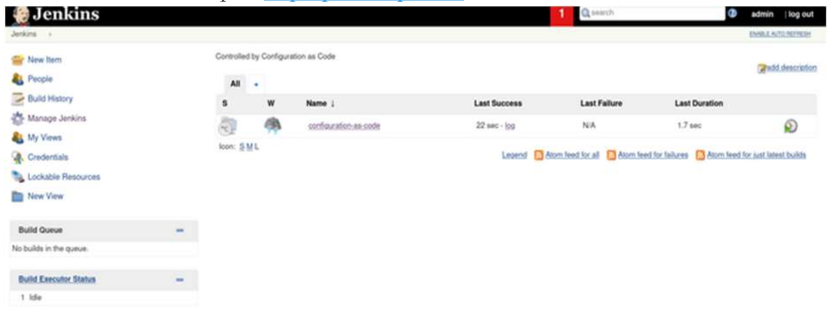
## Session-12

**Jenkins website Options**

## Jenkins - Practical

**Jenkins website Options**

1. Access the Jenkins dashboard with the help of http://public-ip:8080
   1. New item
   2. People
   3. Build history
   4. Manage Jenkins
   5. Security
   6. Status information
   7. Trouble shooting
   8. Tools & Actions
   9. My View, etc..



| S | W | Name ↓ | Last Success | Last Failure | Last Duration |
|---|---|---|---|---|---|
| | | configuration-as-code | 22 sec - log | N/A | 1.7 sec |

## Session-13

**Free Style project**

## Jenkins - Practical

**Free Style project -1**

**Part-1 Git Server**

```
0  git clone https://github.com/Devops-SDA/site.git
1  ls -l
2  cd site
3  ls -l
4  git add .
5  sit status
6  git commit -m "sample website files"
7  git push
```

## Jenkins - Practical

**Part-2 Jenkins Dashboard**

1. Go to GIT dashboard → settings → personal token settings → select all permissions → Generate Token.
2. Go to Jenkins dashboard → Create New item [test] → OK.
3. Source code management → Select Git → copy ulr from github and paste under url
4. Click on Add → Jenkins → enter detail (username & passwd) → Apply → OK.
5. Build now

DevOps

NIPUNA TECHNOLOGIES

## Jenkins - Practical

**Part -3 Linux Server**

1. #cd /var/lib/Jenkins/workspace
2. #ls –l
3. #cd test
4. #ls -l
5. Go to Jenkins → configure → source code management (Git)→ build environment → buils → execute shell → command → type: →
6. Apply → save. And check in work space

```
mkdir test1
cd test
touch 1.txt 2.txt 3.txt
Echo “this is test file”
```

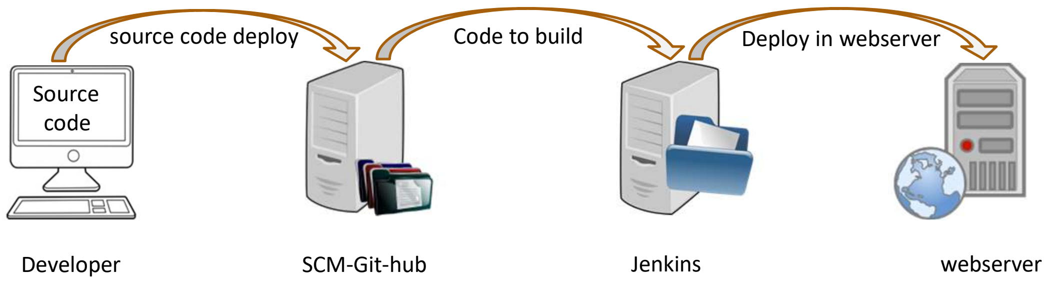
## Session-14

**Free Style project**

DevOps

## Jenkins - Practical

**Website Deployment Project-2**



DevOps

## Website Deployment Project-2 Steps

**Part-1**

1. Create 1 Repository in git-hub

2. Store code in git-hub Repository

**Part-2**

1. Create Linux machine & configure web-server

2. Create user-account in web server

3. Modify root ownership permissions to our user account to the following directories "var, www, html"

4. Connection between web-server and Jenkins will be establish by ssh password less authentication

NIPUNA TECHNOLOGIES

## Website Deployment Project-2 Steps

5. In web-server go to "/etc/ssh/sshd_config" and enable the password authentication "yes"

5. Login to Jenkins machine do the password authentication same as web-server and restart sshd service

6. In Jenkins machine create ssh key using "ssh-keygen"

7. Copy id-rsa-pub key from Jenkins to web-server with the help of ssh-config-id command

8. Verify the connection between Jenkins and web-server with the help of ssh command

NIPUNA TECHNOLOGIES

## Website Deployment Project-2 Steps

**Part-3**

1. Login Jenkins dashboard in the internet

2. We will install the "publish over ssh" plugins, go to manage Jenkins - manage plugins - search for publish over ssh

3. Go to the configure section of manage Jenkins and store the web-server details in the configure system section

4. Go to the net item section and create the new-job of free-style project

    a. Grab the code from SCM

    b. Deploy the code to the web-server machine

    c. Access the website using http://publlic-ip

## Jenkins - Practical

**Website Deployment Project-2**

Part-1 **Git Server**

```
0  git clone https://github.com/Devops-SDA/site.git
1  ls -l
2  cd site
3  ls -l
4  git add .
5  sit status
6  git commit -m "sample website files"
7  git push
```

## Jenkins - Practical

Part-2 **Webserver**

```
1  Create webserver
2  useradd sda
3  passwd sda
4  cd /var
5  ls -l
6  cd www
7  ls -l
8  chown sda:sda html
9  ls -l
10  cd ..
11  chown sda:sda www
12  cd ..
13  chown sda:sda var
14  ls -l
15  clear
16  vi /etc/ssh/sshd_config
17  systemctl restart sshd
18  ifconfig
```

DevOps

## Jenkins - Practical

Part-3 **Jenkins server**

1 vi /etc/ssh/sshd_config

2 systemctl restart sshd

3 ssh-keygen

4 cd .ssh/

5 ls -l

6 ssh-copy-id sda@172.31.42.130

7 ssh sda@172.31.42.130

DevOps

## Jenkins - Practical

Part-4 **Jenkins website**

1 login in to Jenkins account (http://public-ip:8080)

2 Goto Manage Jenkins ---> manage plugins ---> install plugins (publish over ssh)

3 Configuration ---> publish ---> over ssh ---> In path to key ---> /root/.ssh/id_rsa

4 Add ---> ssh server ---> Name: webserver ---> host:172.31.42.130 ---> user: sda ---> Remote Directory:/var ---> password authentication ---> password:Abc@123 ---> test config ---> Apply ---> Save

5 In Dashboard ---> New Item ---> name: webserver ---> Freestyle ---> OK ---> ⊙ Git ---> in url:http://github.com/devops-sda/webserver.git ---> build ---> send files publish over ssh ---> source file: **/*.html ---> remote Directory: www/html ---> select post build ---> ssh arctic.. ---> source file: **/*.html ---> remote Directory: www/html ---> Apply ---> Save

6 In Dashboard ---> Build.

## Session-15

**Maven project**

**Using Groovy script**

*DevOps*

NIPUNA TECHNOLOGIES

## Pipeline Groovy Script

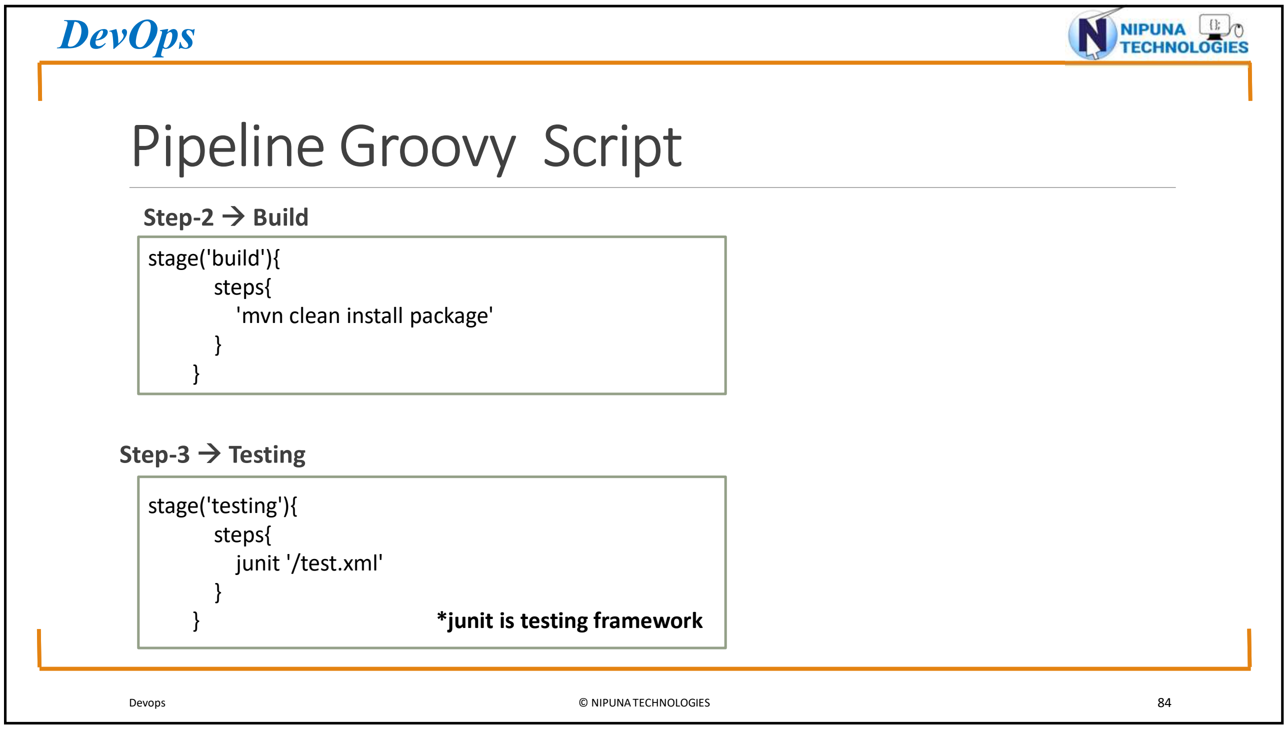
1. Git→ Step-1
2. Build → Step-2
3. Testing → Step-3
4. Artifactory → Step-4
5. Deployment → Step-5

**Beginning statement**

```
pipeline {
   agent any
   stages{
```

**Step-1→ Git**

```
stage('scm-git'){
         steps{
            git ::'https://github.com/Devops-SDA/test.git'
         }
      }
```



DevOps

NIPUNA TECHNOLOGIES

## Pipeline Groovy Script

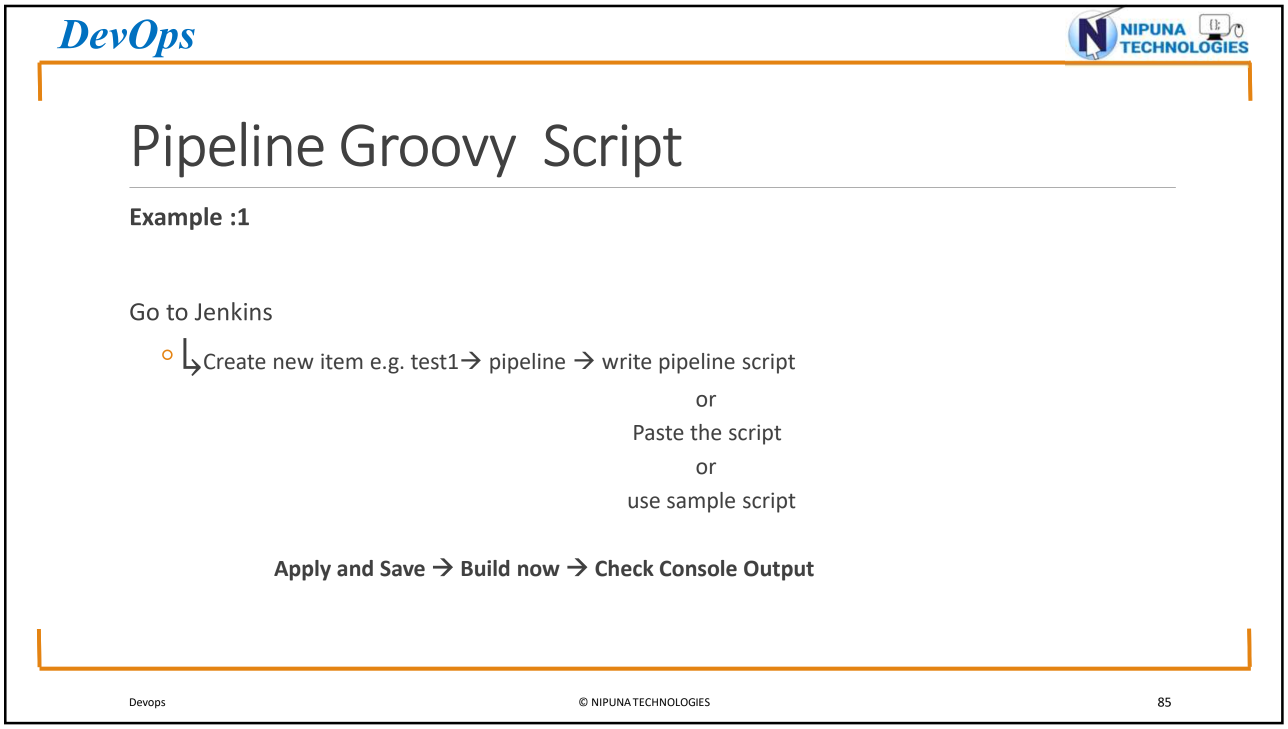
**Step-2 → Build**

```
stage('build'){
        steps{
           'mvn clean install package'
        }
     }
```

**Step-3 → Testing**

```
stage('testing'){
        steps{
           junit '/test.xml'
        }
     }
```

***junit is testing framework**



DevOps

NIPUNA TECHNOLOGIES

## Pipeline Groovy Script

**Example :1**

Go to Jenkins

- ↳Create new item e.g. test1→ pipeline → write pipeline script

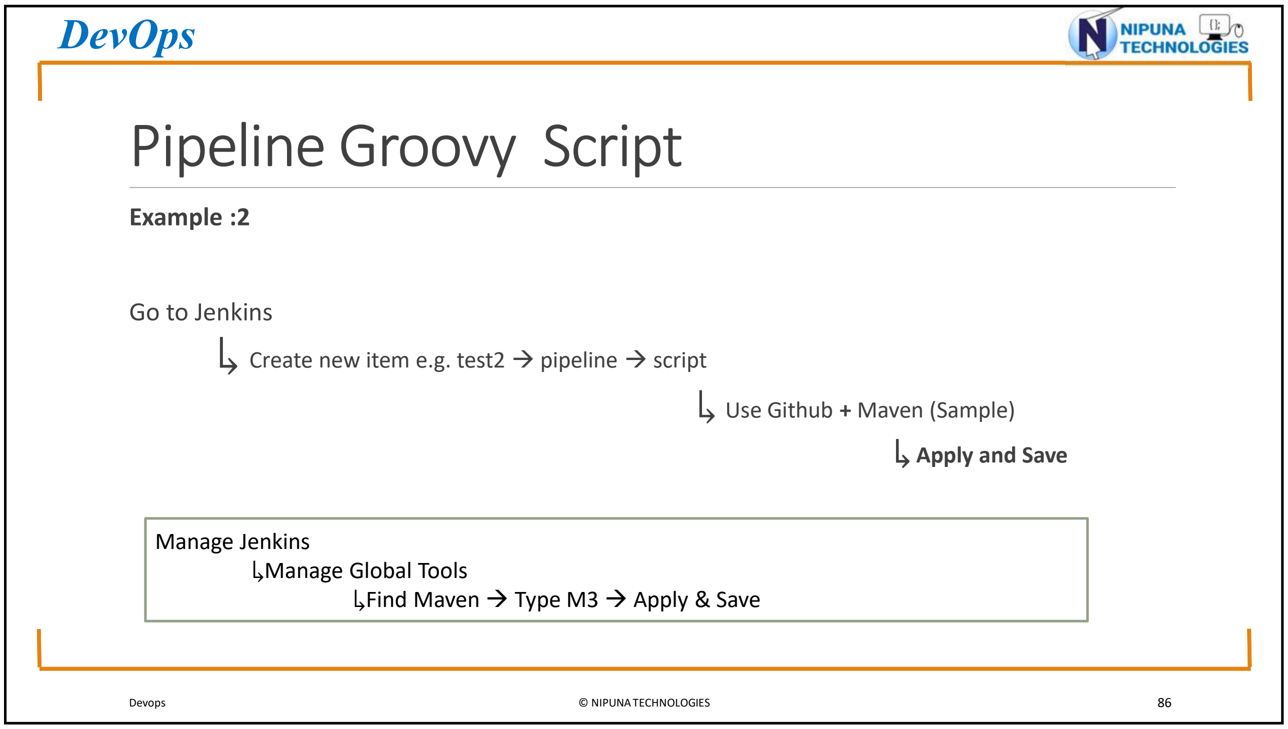
  or

  Paste the script

  or

  use sample script

**Apply and Save → Build now → Check Console Output**



*DevOps*

NIPUNA TECHNOLOGIES

## Pipeline Groovy Script

**Example :2**

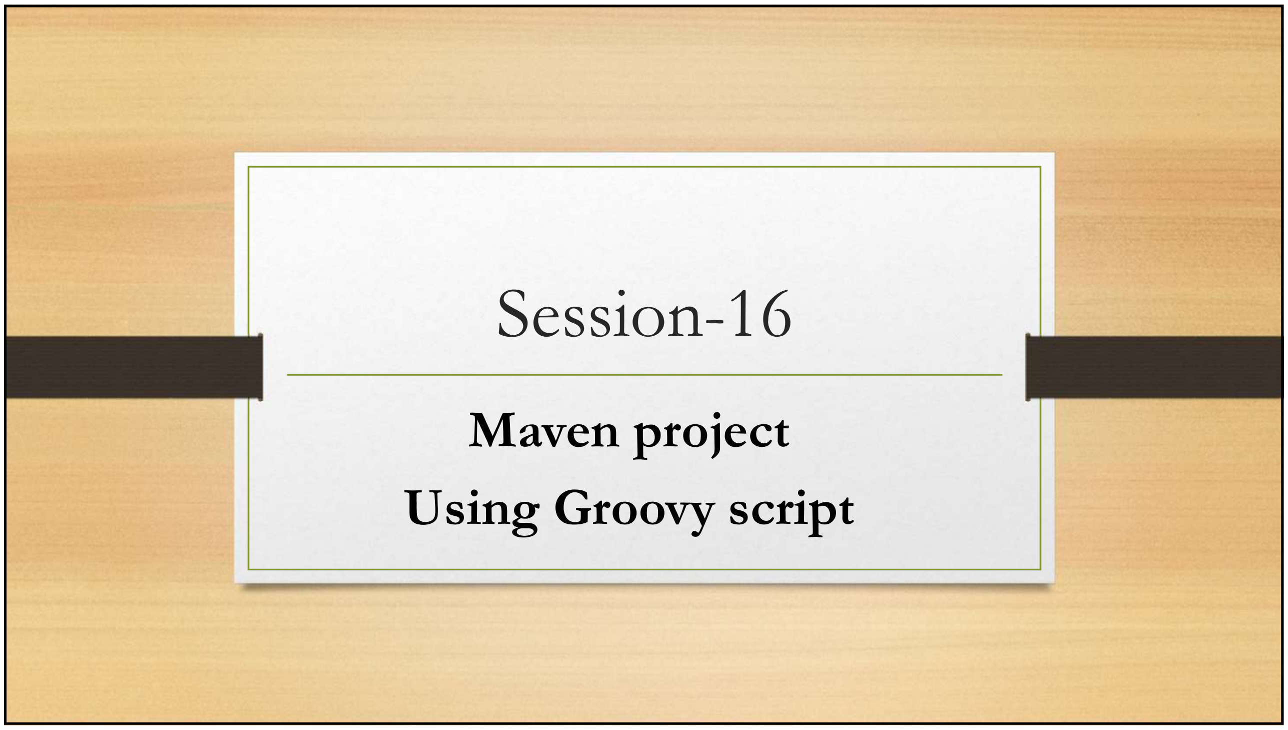
Go to Jenkins

↳ Create new item e.g. test2 → pipeline → script

↳ Use Github + Maven (Sample)

↳ **Apply and Save**

Manage Jenkins
↳Manage Global Tools
↳Find Maven → Type M3 → Apply & Save



## Session-16

**Maven project**

**Using Groovy script**

*DevOps*

NIPUNA TECHNOLOGIES

## Webserver pipeline Groovy  Script

```
pipeline {
 agent any

 stages{
  stage("scm"){
   steps{
    git url: 'https://github.com/devops-srv/webserverpipeline_B23.git'
   }
  }
  stage('Archiving the Artifacts'){
   steps{
    archiveArtifacts '*/.html'
   }
  }
  stage('Build'){
   steps{
    sshPublisher(publishers:[sshPublisherDesc(configName:'webserver', [sshTransfer(excludes: '', execCommand: '', execTimeout: 120000, flatten: true,
makeEmptyDirs: false, noDefaultExcludes: false, patternSeparator: '[, ]+', remoteDirectory: '/var/www/html', remoteDirectorySDF: false, removePrefix: '',
sourceFiles: '*/.html')], usePromotionTimestamp: false, useWorkspaceInPromotion: false, verbose: true)])
   }
  }
 }
}
```

## Session-17

**Docker**

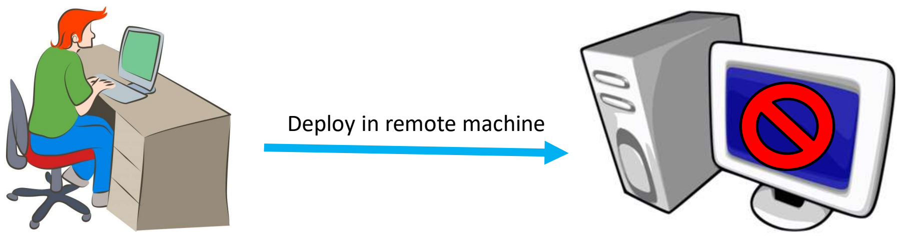
docker

1. Docker is micro services Technology
2. Why Docker
3. Performance
4. Storage

docker

## Docker

In Monolithic



Developer
Develop the Code & Test
It's working

Not executing
Expected outcome

## Topics

1. Docker Images
2. Docker hub
3. Docker file
4. Docker Container
5. Docker workflow
6. Docker Network
7. Real time Project work on maven project for web application deployment

# Session-18

## Docker

*DevOps*

NIPUNA TECHNOLOGIES

## Topics

1. Docker Images
2. Docker hub
3. Docker file
4. Docker Container
5. Docker workflow
6. Docker Network
7. Real time Project work on maven project for web application deployment

## Docker

***commands to create docker container***

docker run –dit --name test_container <test image>   <to create test container>

docker ps

docker ps –a

docker stop ‘container ID’

docker start ‘container ID’

docker rm ‘container ID’

docker exec –it ‘container ID /bin/bash

## Docker

***Docker image commands***

docker pull imagename

docker images

docker rmi image name/ image id

docker built –t image name

***Docker hub commands***

docker tag testimage username/imagename

docker images

docker push username/imagename

docker login

## Docker

**Dockerfile commands**

vi dockerfile

docker built –t imagename

**Docker container commands**

docker run –dit –name test_container <test image> <to create test container>

docker ps

docker ps –a

docker stop 'container ID'

docker start 'container ID'

docker rm 'container ID'

docker exec –it 'container ID /bin/bash

## Docker

**Workflow-1**

Go to Docker hub → image pull → Docker images → Docker container → Access the application

**Workflow-2**

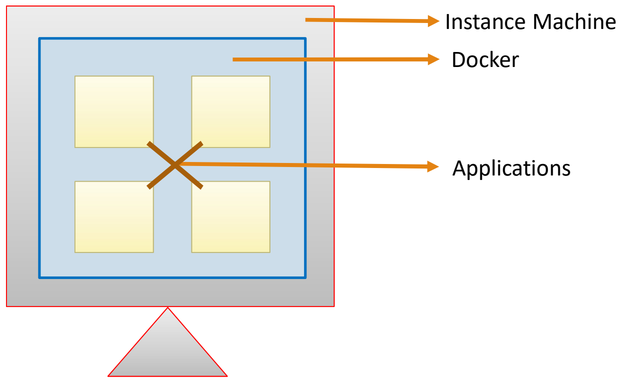
Docker file → Build image → Docker images → publish on Docker hub → Docker container → Access Application.

## Docker

**Docker Network**

Bridge Network

Host Network



## Session-19

**Docker**

## Practical

***Install Docker in Linux***

- create linux instance and connect
- login to machine

```
#yum install docker –y
#systemctl start docker
#systemctl enable docker
#systemctl status docker
#docker --version
```

## Practical

### CREATE DOCKER IMAGE-1

```
#docker pull ubuntu
#docker run –dit --name test_container ubuntu
#docker ps
#docker run –dit ubuntu
#docker ps
```

### CREATE DOCKER IMAGE-2

```
#docker exec –it <docker id> /bin/bash
#mkdir dhoni
#cd dhoni
#touch 1.txt 2.txt 3.txt
#ls –l
#exit
```

## Practical

*Cleanup all images and containers*

#docker ps

#docker stop <image id>

#docker ps

#docker ps –a

#docker rm <image id>

#docker ps –a

#docker images

## Docker

***PRACTICAL-1 INSTALL LINUX AND LOGIN***

#sudo su -

#vi Dockerfile

#Press 'I'

#**FROM ubuntu**

#**RUN apt-get update**

#**RUN apt-get install nginx –y**

#**EXPOSE 80**

#**COPY index.html /var/www/html**

#**cmd ["nginx" "-g", "daemon off;"]**

Esc:wq! To save

***PRACTICAL-1 INSTALL LINUX AND LOGIN***

#cat > index.html

#docker build –t webserver_image **.**

#docker images

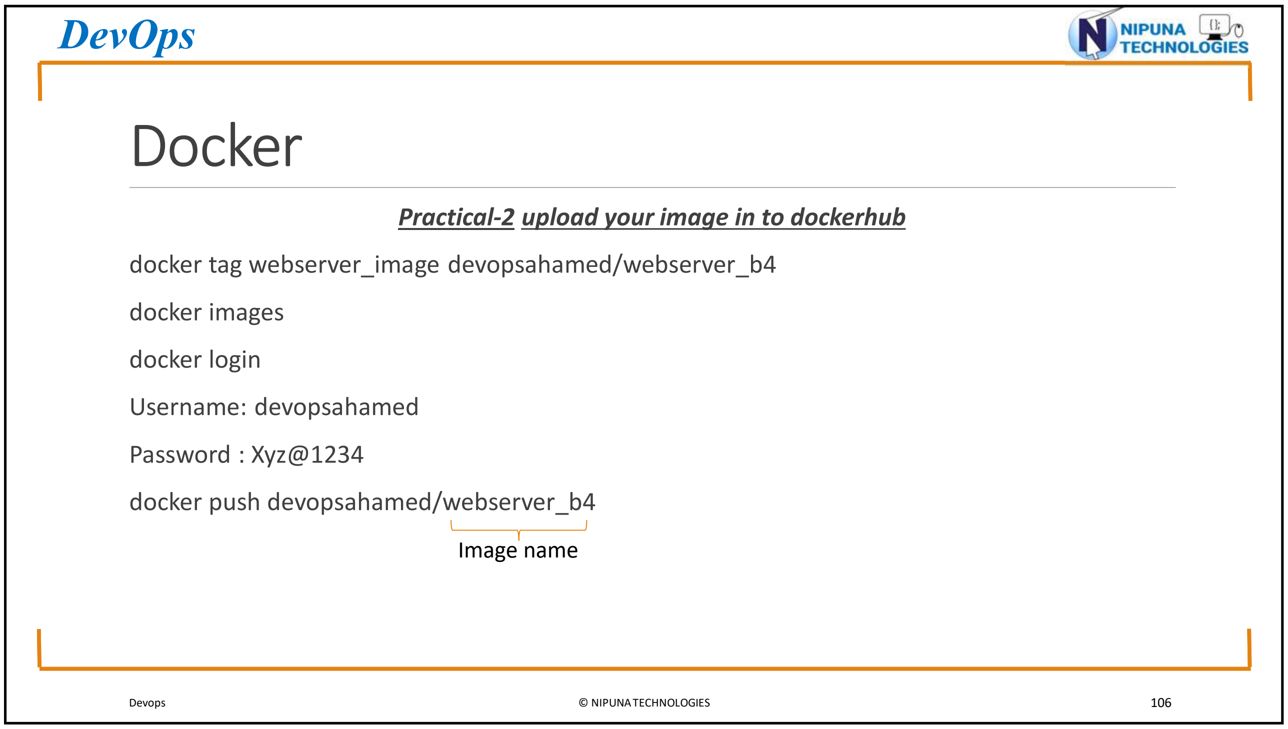
#docker run –dit –name webserver_container –p 1234:80 webserver_image

#docker ps

#docker ps –a

→ **In aws instance security details → inbound rule add 1234 as port number**

→ **open browser type http://public-ip:1234**



*DevOps*

## Docker

***Practical-2 upload your image in to dockerhub***

docker tag webserver_image devopsahamed/webserver_b4

docker images

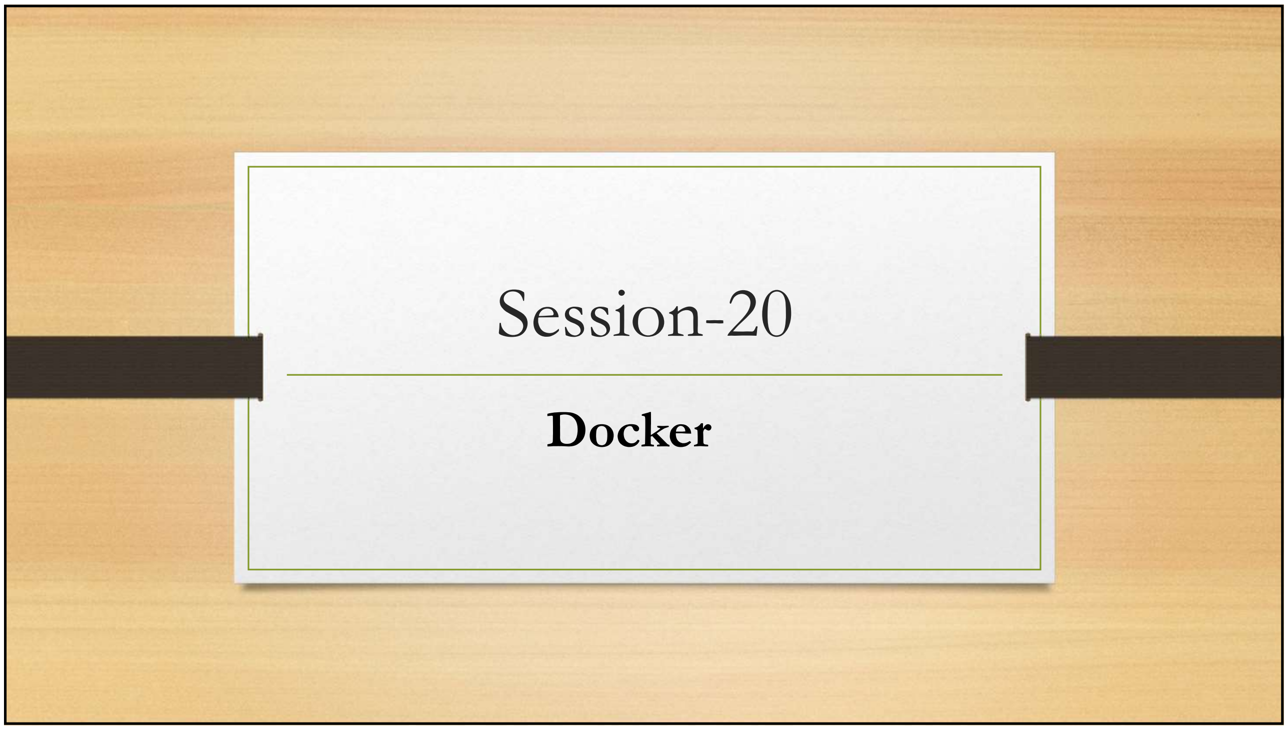
docker login

Username: devopsahamed

Password : Xyz@1234

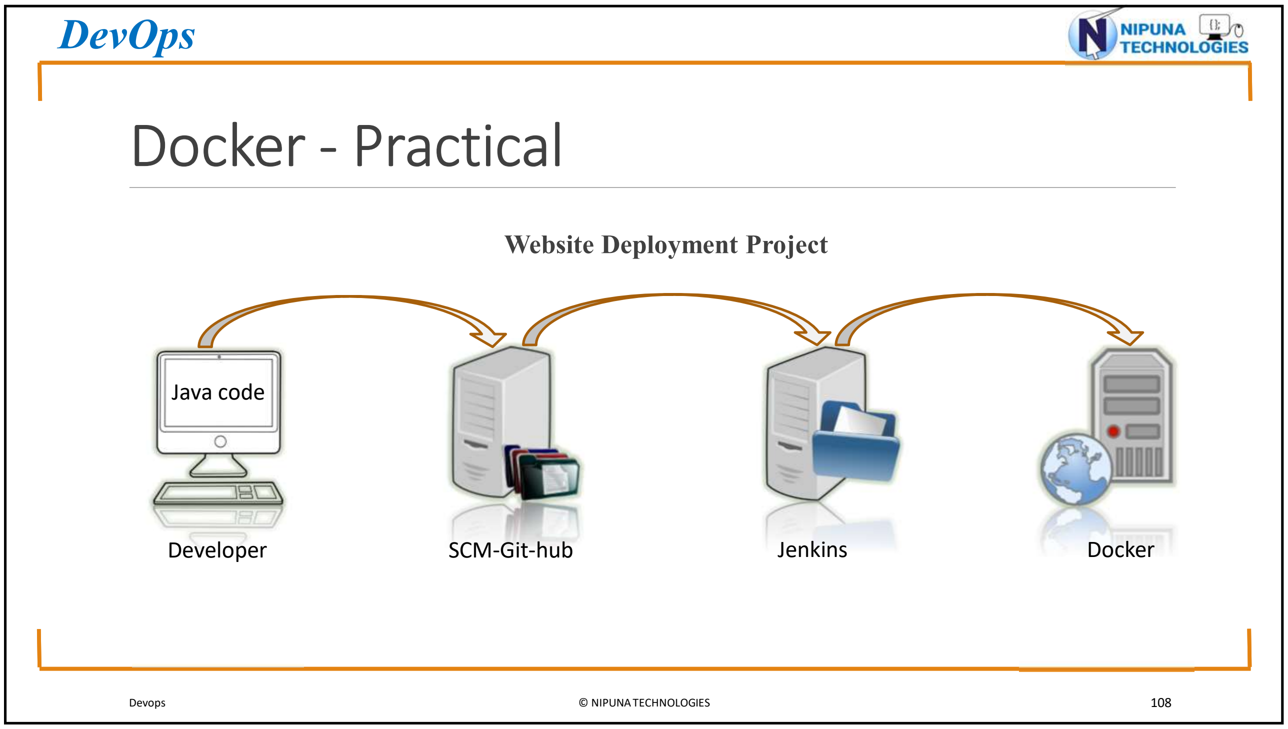
docker push devopsahamed/$\underbrace{\text{webserver\_b4}}_{\text{Image name}}$

Devops © NIPUNA TECHNOLOGIES 106



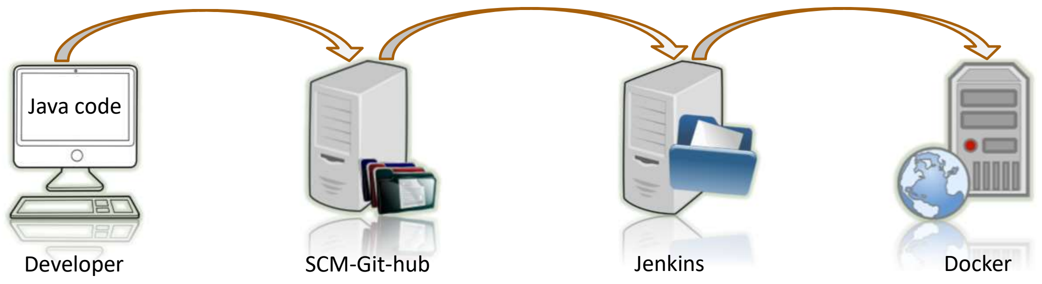
## Session-20

**Docker**

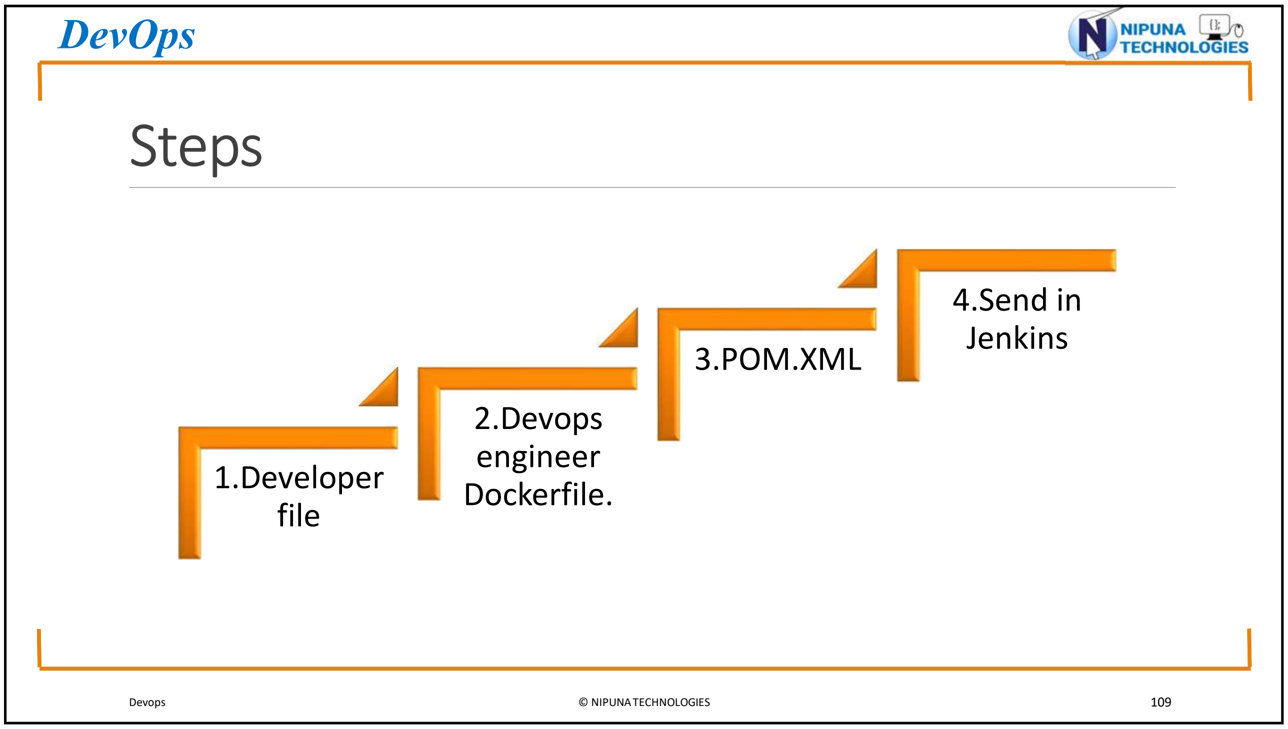


*DevOps*

## Docker - Practical

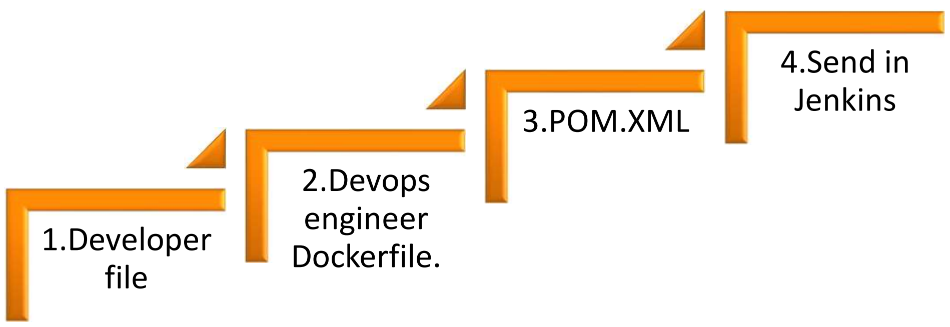

**Website Deployment Project**

Devops © NIPUNA TECHNOLOGIES 108



*DevOps*

## Steps



Devops © NIPUNA TECHNOLOGIES 109

## Docker project steps (Maven)

**Part-1 (Dev & SCM)**

1. Create git repository (private repository)
2. Developer will developing code file in java and store coded file
3. Devops engineer create docker file which is responsible for creating the web application and he will store in git repository
4. Pom.xml will be created by server devs. and it will be stored in git repository

**Part-2 (Docker)**

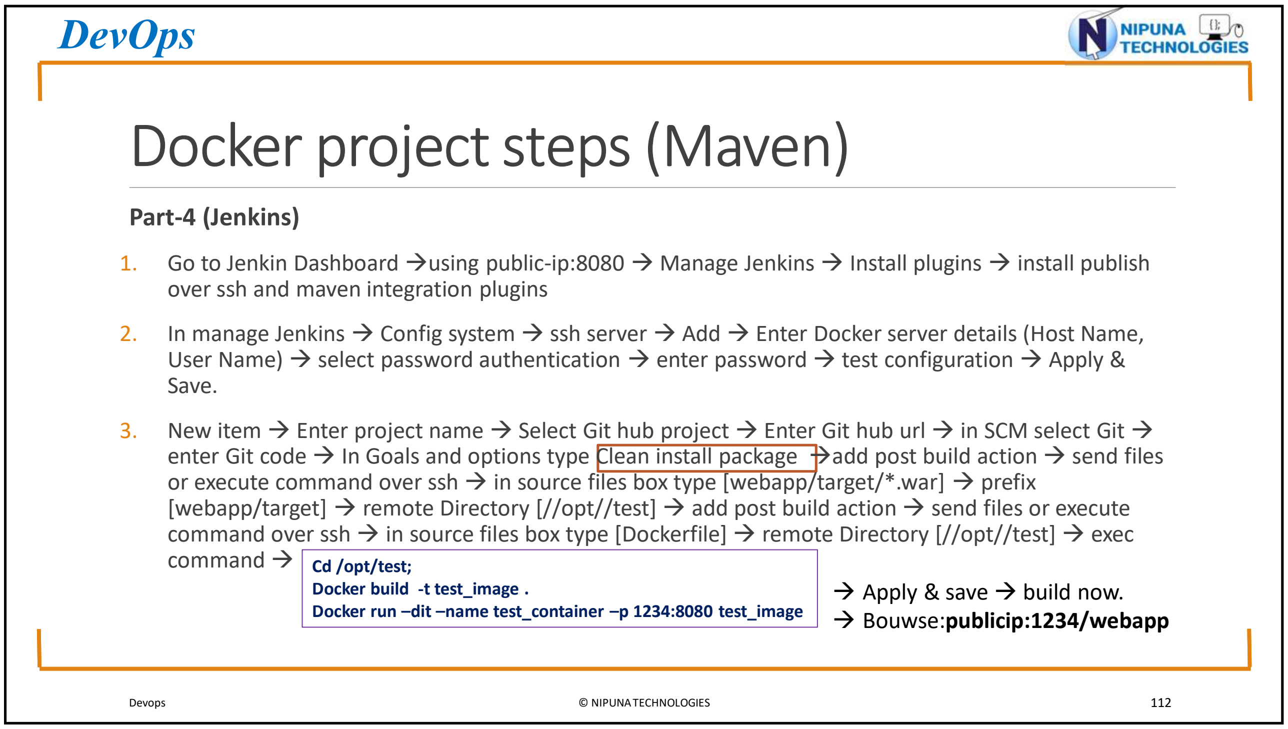
5. Login to Docker machine Create one user account.
6. We will make sure the Docker services are up and running fine
7. We will create one Directory **/opt/test** and this will be used for storing the files coming from Jenkins
8. We will modify ownership permission on the Directory with the Created user account

## Docker project steps (Maven)

9. ls –l | grep docker.sock
10. Usermod –aG docker srv
11. Cat /etc/group | grep docker

**Part-3 (Connectivity between the Jenkins and Docker) → ssh password less authentication**

12. We will login to Docker server and go to the location /etc/ssh/sshd_config and make the service password authentication should be YES
13. Restart the sshd service
14. 12 and 13 points needs to be repeated on Jenkins server (if it is new server)
15. In the Jenkins server generates the ssh key, so two keys will be generated id_rsa, id_rsa_pub
16. Copy the id_rsa.pub key on to the Docker server using ‘ssh-copy-id username>@<ip address>



## Docker project steps (Maven)

**Part-4 (Jenkins)**

1. Go to Jenkin Dashboard →using public-ip:8080 → Manage Jenkins → Install plugins → install publish over ssh and maven integration plugins
2. In manage Jenkins → Config system → ssh server → Add → Enter Docker server details (Host Name, User Name) → select password authentication → enter password → test configuration → Apply & Save.
3. New item → Enter project name → Select Git hub project → Enter Git hub url → in SCM select Git → enter Git code → In Goals and options type Clean install package →add post build action → send files or execute command over ssh → in source files box type [webapp/target/*.war] → prefix [webapp/target] → remote Directory [//opt//test] → add post build action → send files or execute command over ssh → in source files box type [Dockerfile] → remote Directory [//opt//test] → exec command →

```
Cd /opt/test;
Docker build  -t test_image .
Docker run –dit –name test_container –p 1234:8080 test_image
```

→ Apply & save → build now.
→ Bouwse:**publicip:1234/webapp**