********** Topic **********

**********MAVEN**********

==========
Introduction
==========

-> Java Programs (.java file) contains source code.
-> We can't execute .java files directley.
-> Java Programs should be converted into Machine understandable format to Execute.
-> We need to compile java source code into byte code using java compiler . (javac)

Ex: javac Demo.java

-> When we compile java code it will create .class file.
-> We need to execute .class file to run the java program.

Ex: java Demo

-> When we run java program using java command, JVM will start and it will execute java program

Note: JVM stands for Java Virtual Machine

-> JVM will convert byte code into machine understandable code.
-> Java project contains several java programs (.java files).
-> We need to compile project source code into byte code.
-> When we compile project source code we will get .class files.
-> To deploy java project, we will package all .class files as JAR file or WAR File.

JAR : Java Archieve
WAR : Web Archieve

-> Standalone java projects will be packaged as a JAR file

Note: Stadalone applications means the project which runs only in one computer

Ex: notepad, calculator....

-> Web Applications will be packaged as WAR file
Note: Multiple users can access web applications at a time

Ex: gmail, facebook, flipkart, naukri etc....

==========

Build Tools

==========

-> Build tools are used to automate application build process

a) Compile The Source Code
b) Download Required Dependencies (Ex: springboot, hibernate, junit, log4j, kafka...)
c) Execute Unit Test Cases (JUnits)
d) Package our application as jar / war

JAR ---> Java Archieve ---> It is package format for java standalone appl'n.

WAR ---> Web Archieve ---> It is package format for java web appl'ns.

-> Instead of doing the above build steps manually, we can take the help of Build Tools to automate that process.

-> We have below build tools for java applications

1) Ant (Outdated)
2) Maven
3) Gradle

==========

Maven

==========

-> Maven is a free and open source software given by Apache Organization
-> Maven s/w is developed using Java programming language
-> Maven is used to perform Build Automation for java projects
-> Maven is called as Java Build Tool

Note: Maven is used as a build tool for java projects.

==========

What we can do using maven

==========

1) We can create initial project folder structure using maven
2) We can download "project dependencies" using maven
(ex : springboot, hibernate, kafka, redis, email, log4j, junit, security...)

-> To develop one java project we will use several frameworks like spring, hibernate etc along with Java

-> We need to download those frameworks and we should add to our java project

-> These frameworks we are using in our project are called as our project dependencies

-> Instead of we are downloading dependencies, we can tell to maven s/w to download dependencies

Note: Required dependencies we will add in "pom.xml" file then maven s/w will download them

-> pom stands for project object model

-> When we create maven project then pom.xml file will be created automatically
-> pom.xml will act as input file for maven software

3) We can compile project source code using maven
4) We can package java project as jar or war file using maven

==========
Maven Installation
==========

1) Download and install Java software

-> When we install java we will below 2 things

a) JDK (Java Development Kit)
b) JRE (Java Runtime Environment)

-> JDK contains set of tools to develop java programs
-> JRE contains platform/environment which is used to run java programs

Link To Download Java :
https://www.oracle.com/in/java/technologies/javase/javase8-archive-downloads.html

2) Set JAVA_HOME in Environment Variables (System Env Variables)

User Environment Variables: Specific to particular account which logged in our PC.
System Envrionment Variables : For All Accounts

Note: Environment Variables will be used by operating system
JAVA_HOME = C:\Program Files\Java\jdk1.8.0_202

3) Set Path for JAVA (Go to System Env Variables -> Env Variables -> System Variables -> Select Path and Click on Edit then add JDK path like below)
Path = C:\Program Files\Java\jdk1.8.0_202\bin

4) Verify Java installation by executing below command in "Command Prompt"
> java -version

Note: It should dipslay java version which we have installed
5) Download Maven software from Apache website

Link To download Maven : https://maven.apache.org/download.cgi
File Name : apache-maven-3.8.5-bin.zip (Binary Archive)

6) Extract Maven Zip file -> Copy extracted maven folder and paste it in "C" drive

7) Set MAVEN_HOME in System Environment Variables

MAVEN_HOME = C:\apache-maven-3.8.5

8) Set Path for Maven in System Environment Variables

Path : C:\apache-maven-3.8.5\bin

9) Open Command Prompt and verify Maven Installaton using below command

> mvn -version

==========

Maven Terminology

==========

archetype
groupId
artifactId
packaging

-> **Archteype** represents what type of project we want to create
=> maven-archetype-quickstart : It represents java standalone application
=> maven-archetype-webapp : It represents java web application

Note: Maven providing 1500+ archetypes

-> **groupId** represents company name or project name

-> **artifactId** represents project name or project module name

-> **packaging** represents how we want to package our java application (jar or war)

==========

Creating stand-alone application using maven

==========

1) Create one folder for maven practise
2) Open Command prompt from that folder
3) Execute below command to create maven project

>> mvn archetype:generate -DgroupId=in.ashokit -DartifactId=01-Maven-App -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

4) Once project created then verify project folder structure

```
01-Maven-App
    - src
        - main
            -java
        - test
            -java
    - pom.xml
```

src/main/java : Application source code (.java files)
src/test/java : Application Unit Test code (.java files)
pom.xml : Project Object Model (Maven configuration file)

5) We can add dependencies in pom.xml file
6) We can find maven dependencies in www.mvnrepository.com website
7) Add below dependency in pom.xml file

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.3.23</version>
</dependency>
```

8) Execute maven goal

> mvn compile

==========

How maven will downoad dependencies

==========

-> Maven will download dependencies using repository

-> In Maven we have 3 types of repositories

1) Central Repository

2) Remote Repository

3) Local Repository

-> central repository is maintaining by apache organization

-> every company will maintain their own remote repository (Ex: Nexus, JFrog)

-> Local repository will be created in our system (Location : C://users/<uname>/.m2)

-> When we add dependency in pom.xml, maven will search for that dependency in local repository. If it is available it will add to project build path.

-> If dependency not available in local repository then maven will connect to Central Repository or Remote Repository based on our configuration.

Note: By default maven will connect with central repository. If we want to use remote reposiotry then we need to configure remote repository details.

Note: Every software company will maintain their own remote repositories (Ex: Nexus / JFrog)

==========

Configuring Remote Repository

==========

```
<repositories>
      <repository>
         <id>id</id>
         <url>jfrong-repo-url/</url>
      </repository>
</repositories>
```

==========

Maven Goals

==========

-> To perform project build activities maven prorvided several goals for us

- clean
- compile
- test
- package
- install

-> clean goal is used to delete target folder

-> compile goal is used to compile project source code. Compiled code will be stored in target folder.

compile

.java ----------> .class

-> test goal is used to execute unit test code of our application (junit code)[test goal= resource+compile+test]

-> package goal is used to generate jar or war file for our application based on packaging type available in pom.xml file.

Note: jar or war file will be created in target folder.

-> install goal is used to install our project as a dependency in maven local repository.

Note: Every maven goal is associated with maven plugin. When we execute maven goal then respective maven plugin will execute to perform the operation.

Syntax : mvn <goal-name>

Note: We need to execute maven goals from project folder where pom.xml is present.

==========

Creating web application using maven

==========

>> mvn archetype:generate -DarchetypeArtifactId=maven-archetype-webapp -DgroupId=in.ashokit -DartifactId=flipkart_app -DinteractiveMode=false

==========

Working with Maven Using IDE (STS / Eclipse

==========

-> Every IDE will have Maven plugin by default (no need to install)

-> Using IDE we can create maven project directley

-> Create Standalone project using IDE (quick-start archetype)

-> Create Web application using IDE (web-app archetype)

-> Right Click on Project -> Build Path -> Configure Build Path -> Libraries -> Select JRE -> Edit -> Configure JDK

Note: If JDK not displaying, then add JDK using 'Installed-JRES' button (we have to select jdk folder from our Program Files where we have installed java)

Note: For web application we need to add 'servlet-api' dependency in pom.xml because every jsp page has to convert into servlet.

```
<dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>4.0.1</version>
        <scope>provided</scope>
</dependency>
```

-> Right click on Project -> Run As -> Maven Build -> Enter Maven Goals (Ex: clean package)

==========

What is Exclusion In Maven

==========

-> Exclusion is used to remove un-wanted child dependencies from build path

Ex: When we add 'spring-context' we are getting other dependencies also like 'spring-core', 'spring-beans', 'spring-aop' etc..

-> I want to use 'spring-context' but i don't want 'spring-aop' in build path then we can exclude 'spring-aop' from 'spring-context' like below

```
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.3.23</version>
        <exclusions>
            <exclusion>
                <groupId>org.springframework</groupId>
                <artifactId>spring-aop</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
```

==========

Maven Multi - Module Project

==========

-> A project contains collection of Modules like below

1) User Module
2) Admin Module
3) Reports Module
5) Payment Module
6) Products Module etc....

-> It is not recommended to develop all the modules code in single project. Maintenence will become very difficult

-> To simplify the development we will use 'Maven-Multi-Module' concept

-> In Maven Multi Module, We will create project with Parent - Child relation

-> One Parent Project can have multiple child projects

-> Child Projects are called as 'Maven Modules'

-> When we execute Maven Goal in Parent Project then it will execute that goal in all the child projects also.

Note: Parent Project Packaging type should be 'POM'

==========

Creating Maven Multi Module Project in IDE

==========

1) Create Maven Project with Packaging Type as 'pom'
2) Right Click on the Project -> New -> Select Others -> Search For Maven Module -> Create the Module
3) After Module got created, check parent project pom.xml and child project pom.xml

==========
Parent Project pom.xml looks like below
==========

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

	<modelVersion>4.0.0</modelVersion>
	<groupId>com.flipkart</groupId>
	<artifactId>Flipkart_App</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<packaging>pom</packaging>

	<modules>
		<module>ADMIN</module>
		<module>REPORTS</module>
	</modules>

</project>
```

==========

Child Project pom.xml looks like below

==========

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.flipkart</groupId>
    <artifactId>Flipkart_App</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </parent>
  <artifactId>REPORTS</artifactId>

</project>
```

-> Right click on Parent Project -> Run as -> Maven Build -> Enter Goals and Execute.

==========

What is Dependency Scope in the Maven

==========

-> 'Scope' represents when that dependency should be used by Maven in Build Process

There are 6 scopes:

compile : This is the default scope, used if none is specified. Compile dependencies are available in all classpaths of a project. Furthermore, those dependencies are propagated to dependent projects.

provided : This is much like compile, but indicates you expect the JDK or a container to provide the dependency at runtime. For example, when building a web application for the Java Enterprise Edition, you would set the dependency on the Servlet API and related Java EE APIs to scope provided because the web container provides those classes. A dependency with this scope is added to the

classpath used for compilation and test, but not the runtime classpath. It is not transitive.

runtime : This scope indicates that the dependency is not required for compilation, but is for execution. Maven includes a dependency with this scope in the runtime and test classpaths, but not the compile classpath.

test : This scope indicates that the dependency is not required for normal use of the application, and is only available for the test compilation and execution phases. This scope is not transitive. Typically this scope is used for test libraries such as JUnit and Mockito. It is also used for non-test libraries such as Apache Commons IO if those libraries are used in unit tests (src/test/java) but not in the model code (src/main/java).

system: This scope is similar to provided except that you have to provide the JAR which contains it explicitly. The artifact is always available and is not looked up in a repository.

import : This scope is only supported on a dependency of type pom in the <dependencyManagement> section. It indicates the dependency is to be replaced with the effective list of dependencies in the specified POM's <dependencyManagement> section. Since they are replaced, dependencies with a scope of import do not actually participate in limiting the transitivity of a dependency.

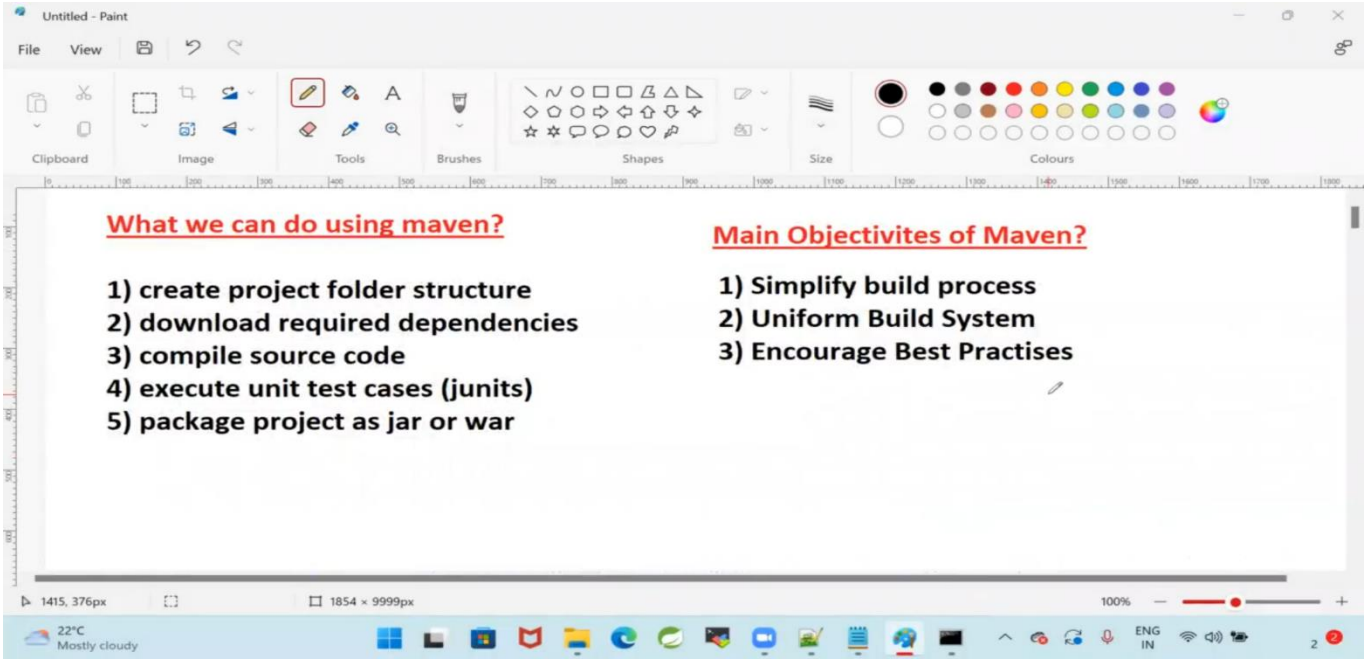
==========

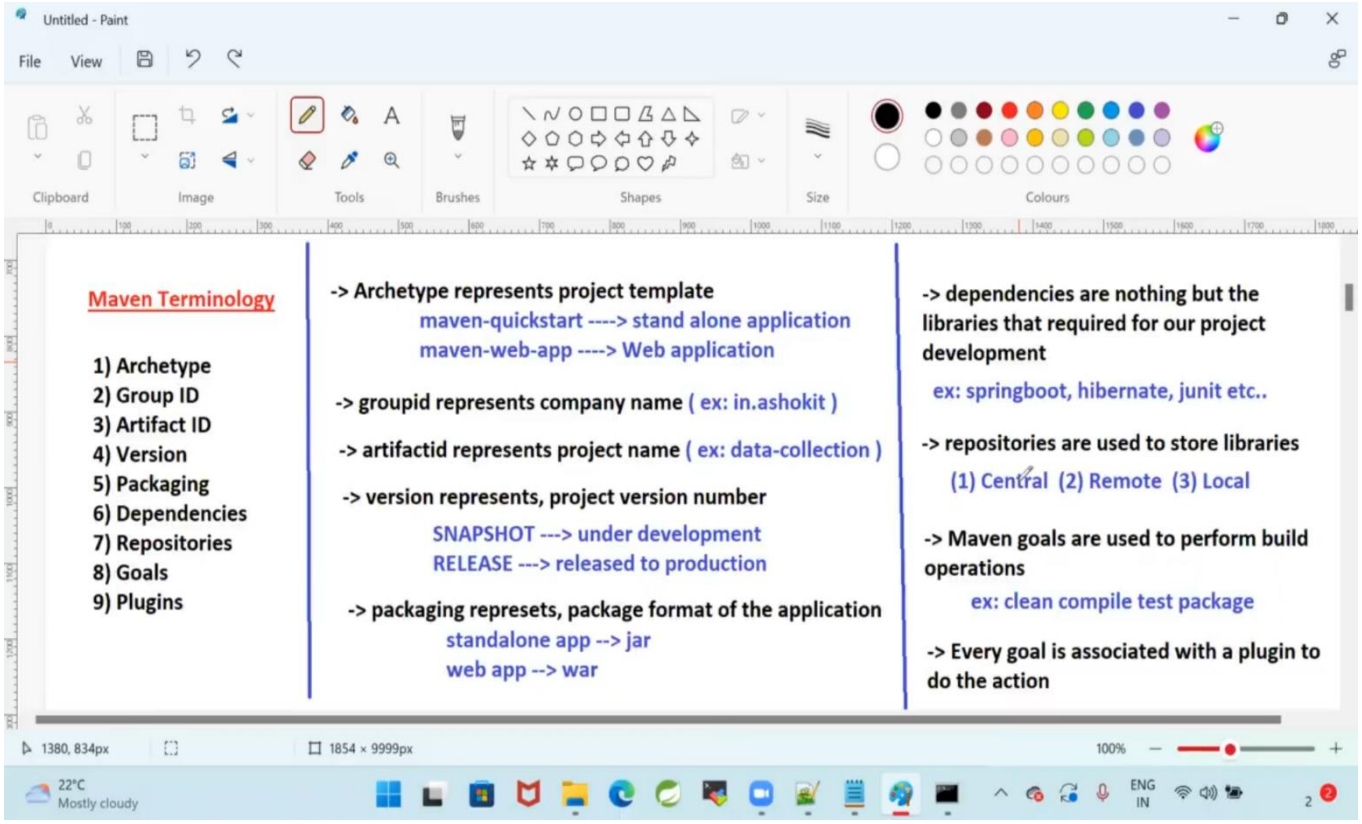
What we have learnt in Maven

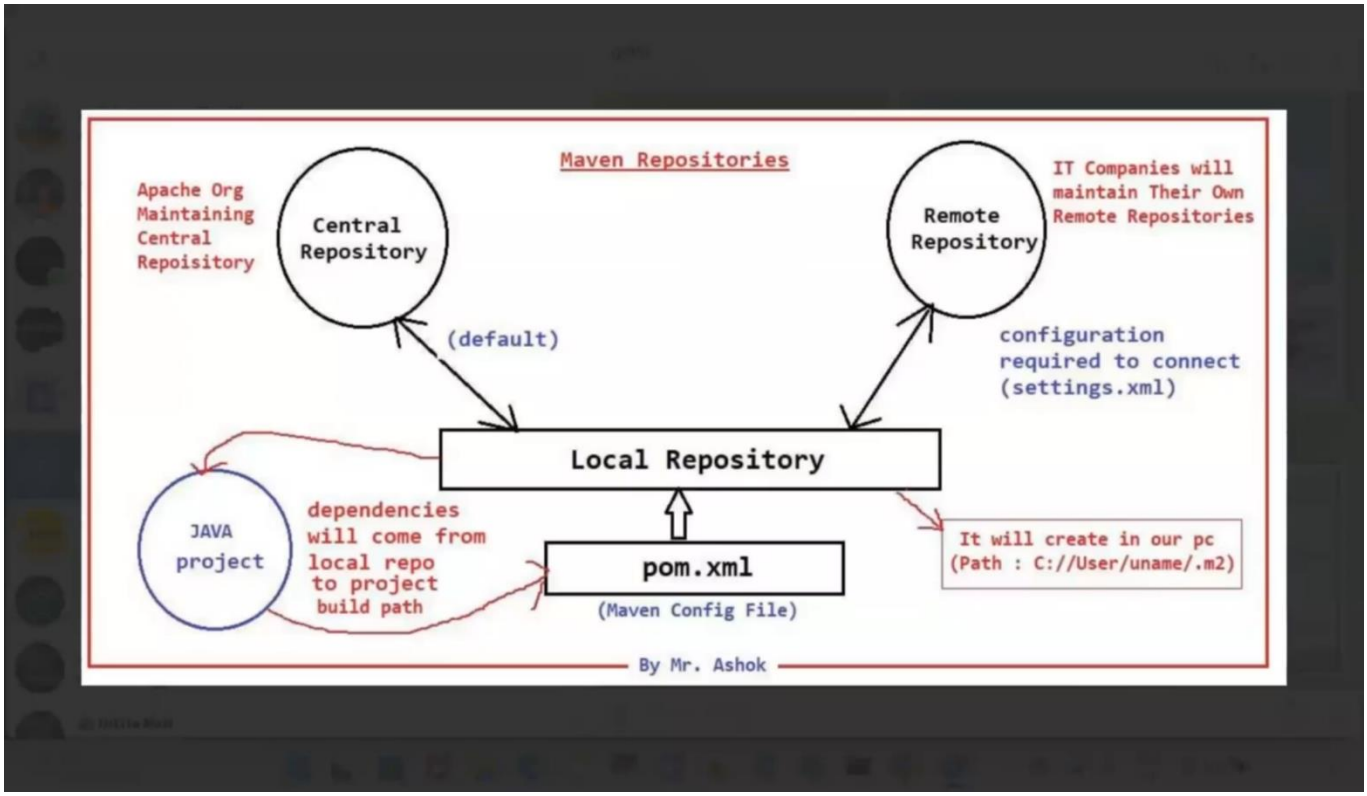
==========

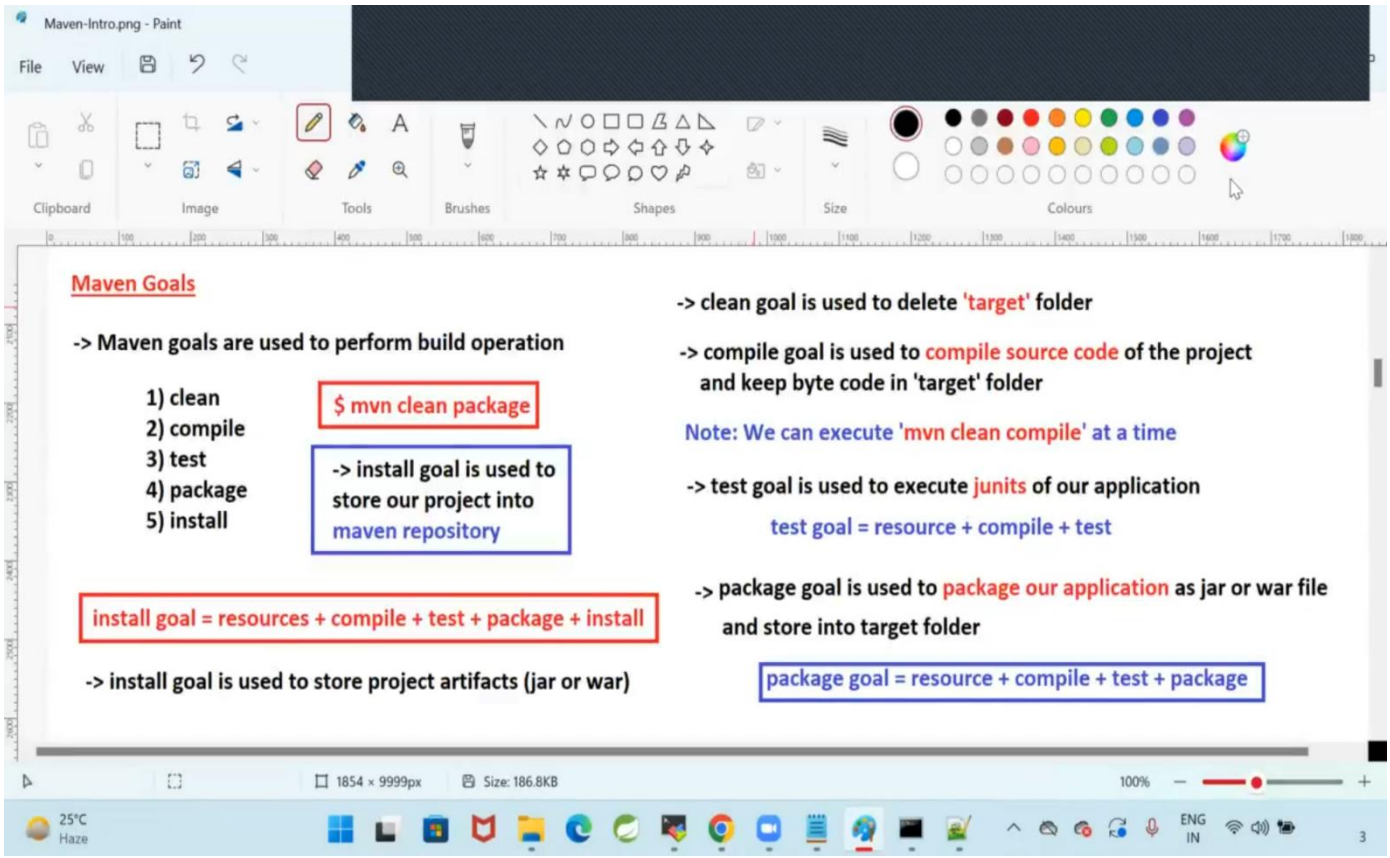
1) What is Build Tool
2) Why we need Build Tool
3) What is Maven
4) Maven Setup in Windows OS (JAVA_HOME, JAVA PATH, MAVEN_HOME, MAVEN PATH)
5) Maven Terminology
6) Standalone Application Creation using Command Prompt & IDE
7) Web Application Creation using Command Prompt & IDE
8) Maven pom.xml file
9) How to add dependencies in 'pom.xml' file (www.mvnrepository.com)

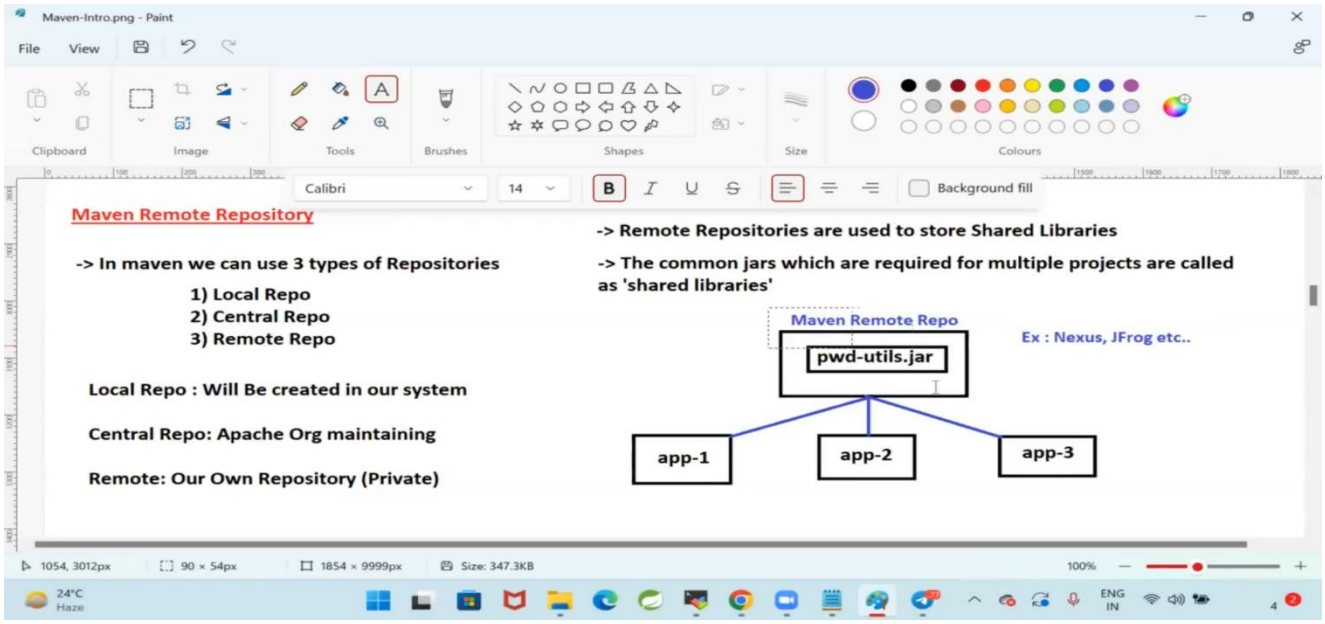
10) Maven Goals Execution in command prompt & in IDE
11) Maven Repositories (local, remote, central)
12) Working with Remote Repository (Ex: Nexus)
13) Maven Dependency Exclusion
14) Maven Multi Module Project
15) Maven Dependency Scopes

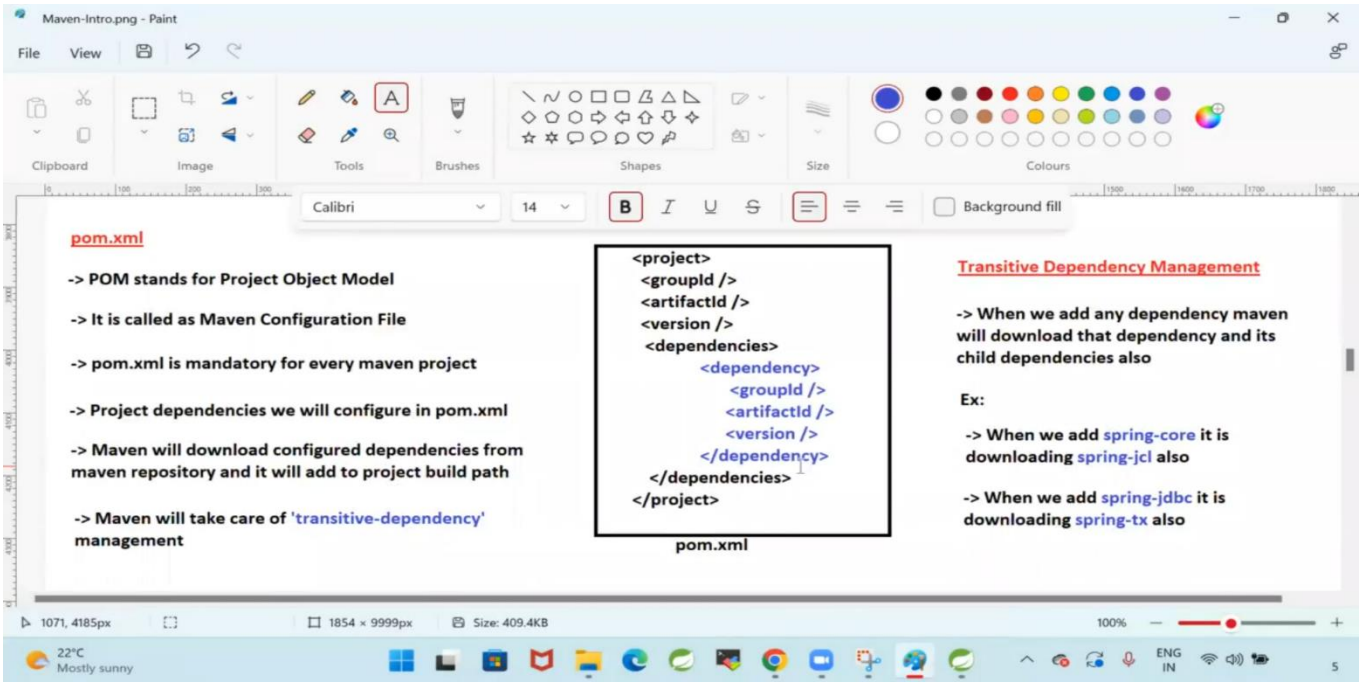


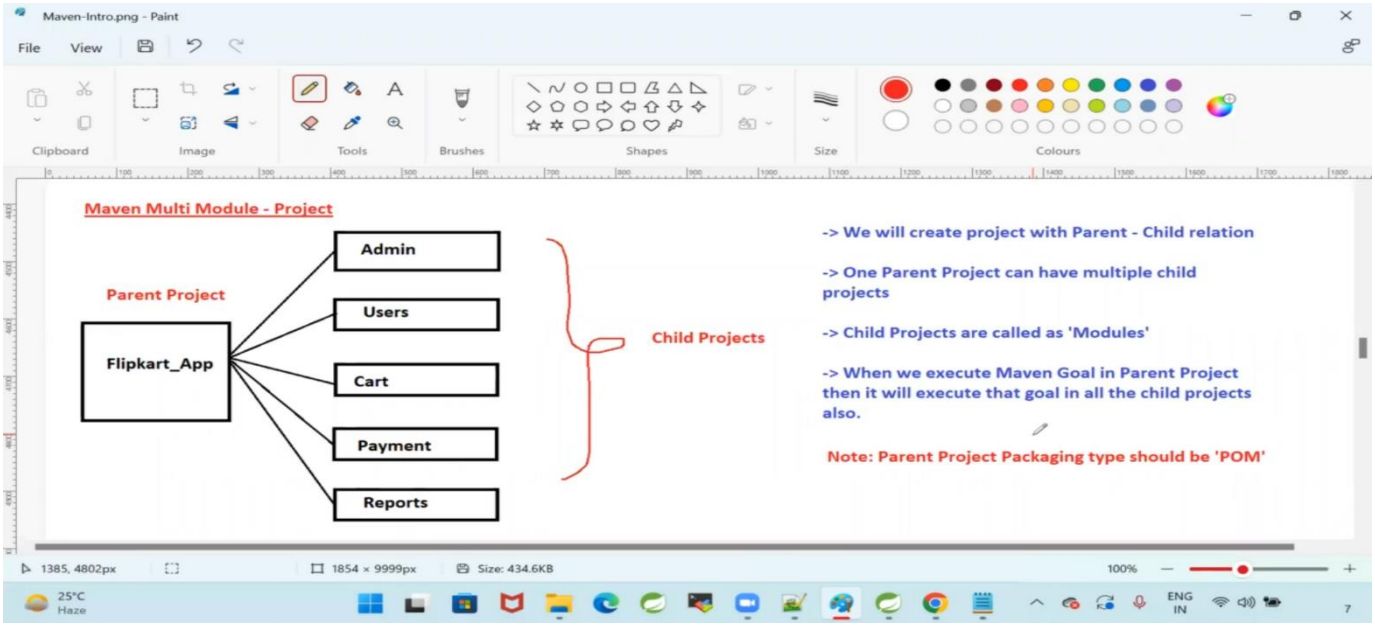



Maven Repositories
Apache Org Maintaining Central Repoisitory
Central Repository
(default)
Remote Repository
IT Companies will maintain Their Own Remote Repositories
configuration required to connect (settings.xml)
Local Repository
JAVA project
dependencies will come from local repo to project build path
pom.xml
(Maven Config File)
It will create in our pc (Path : C://User/uname/.m2)
By Mr. Ashok


Maven-Intro.png - Paint
File View
Clipboard Image Tools Brushes Shapes Size Colours
Maven Goals
-> Maven goals are used to perform build operation
1) clean
2) compile
3) test
4) package
5) install
$ mvn clean package
-> install goal is used to store our project into maven repository
install goal = resources + compile + test + package + install
-> install goal is used to store project artifacts (jar or war)
-> clean goal is used to delete 'target' folder
-> compile goal is used to compile source code of the project and keep byte code in 'target' folder
Note: We can execute 'mvn clean compile' at a time
-> test goal is used to execute junits of our application
test goal = resource + compile + test
-> package goal is used to package our application as jar or war file and store into target folder
package goal = resource + compile + test + package
1854 × 9999px
Size: 186.8KB
100%
25°C Haze
ENG IN

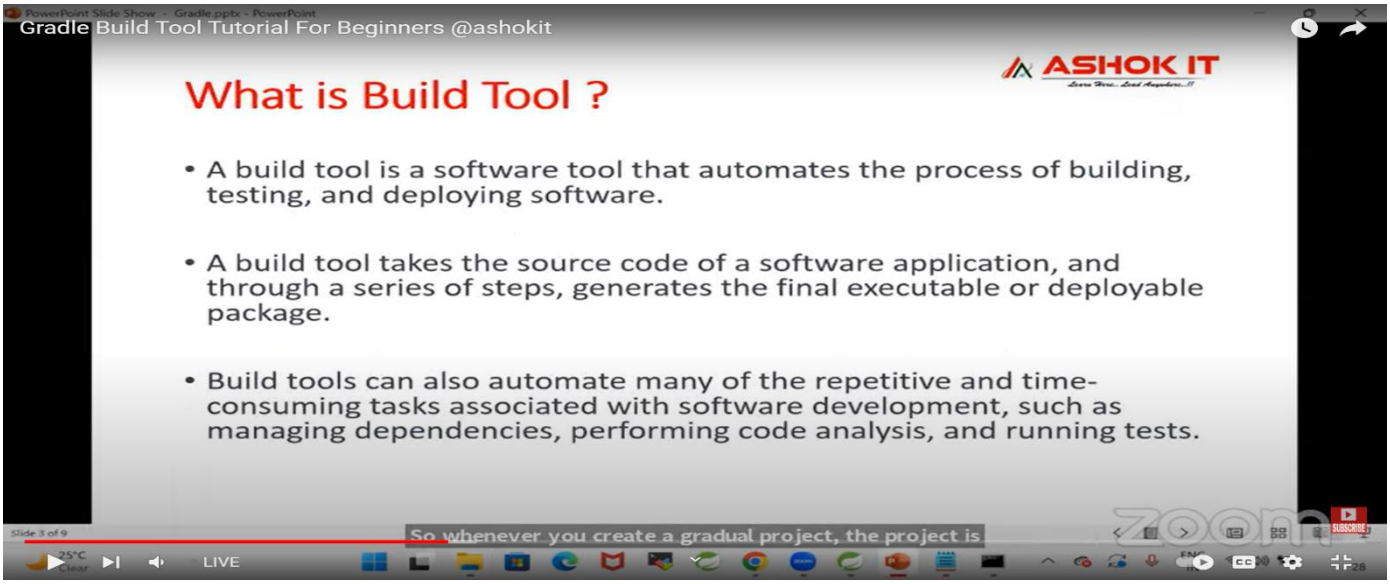

Maven-Intro.png - Paint
Maven Remote Repository
-> In maven we can use 3 types of Repositories
1) Local Repo
2) Central Repo
3) Remote Repo
Local Repo : Will Be created in our system
Central Repo: Apache Org maintaining
Remote: Our Own Repository (Private)
-> Remote Repositories are used to store Shared Libraries
-> The common jars which are required for multiple projects are called as 'shared libraries'
Maven Remote Repo
pwd-utils.jar
Ex : Nexus, JFrog etc..
app-1
app-2
app-3

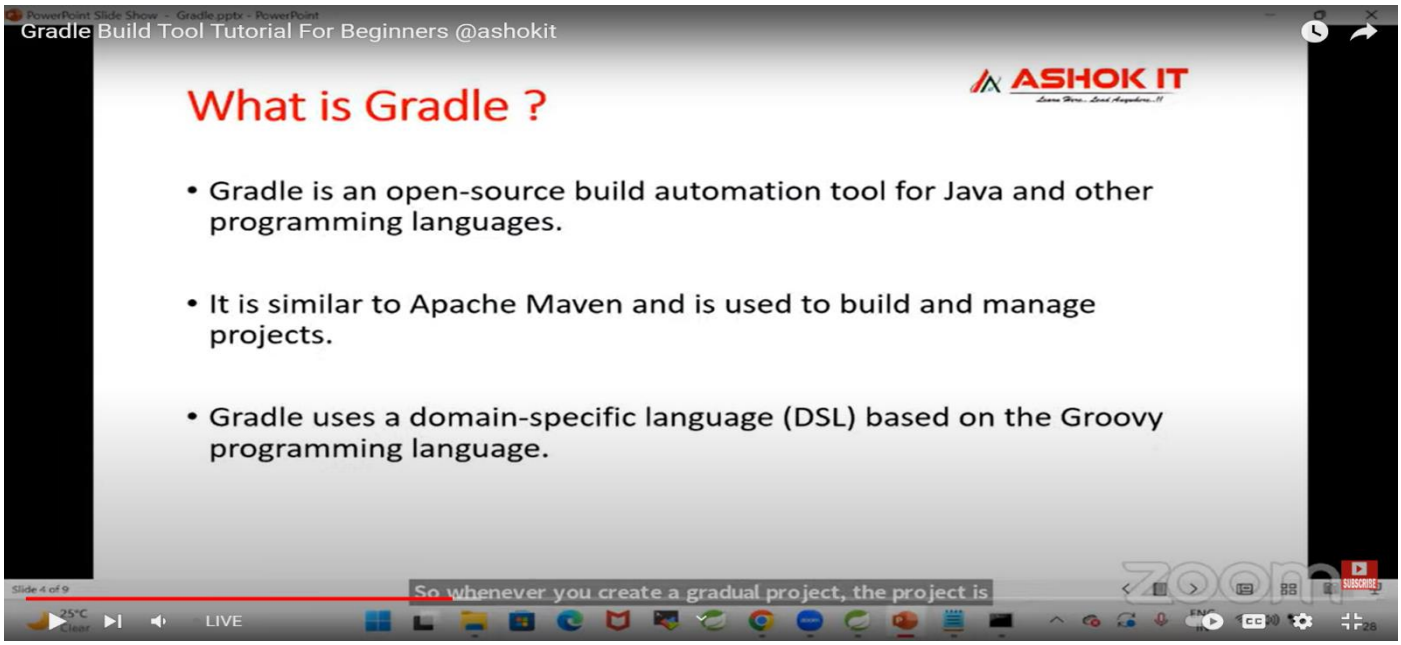

Maven-Intro.png - Paint
pom.xml
-> POM stands for Project Object Model
-> It is called as Maven Configuration File
-> pom.xml is mandatory for every maven project
-> Project dependencies we will configure in pom.xml
-> Maven will download configured dependencies from maven repository and it will add to project build path
-> Maven will take care of 'transitive-dependency' management
<project>
<groupId />
<artifactId />
<version />
<dependencies>
<dependency>
<groupId />
<artifactId />
<version />
</dependency>
</dependencies>
</project>
pom.xml
Transitive Dependency Management
-> When we add any dependency maven will download that dependency and its child dependencies also
Ex:
-> When we add spring-core it is downloading spring-jcl also
-> When we add spring-jdbc it is downloading spring-tx also

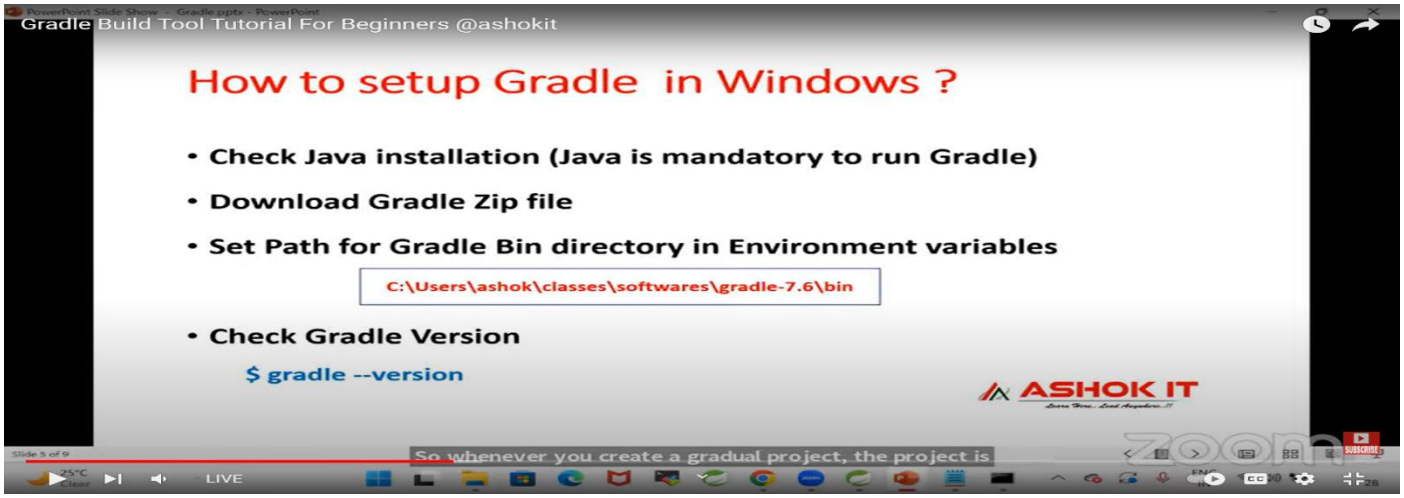

Maven-Intro.png - Paint
Maven Multi Module - Project
Parent Project
Flipkart_App
Admin
Users
Cart
Payment
Reports
Child Projects
-> We will create project with Parent - Child relation
-> One Parent Project can have multiple child projects
-> Child Projects are called as 'Modules'
-> When we execute Maven Goal in Parent Project then it will execute that goal in all the child projects also.
Note: Parent Project Packaging type should be 'POM'

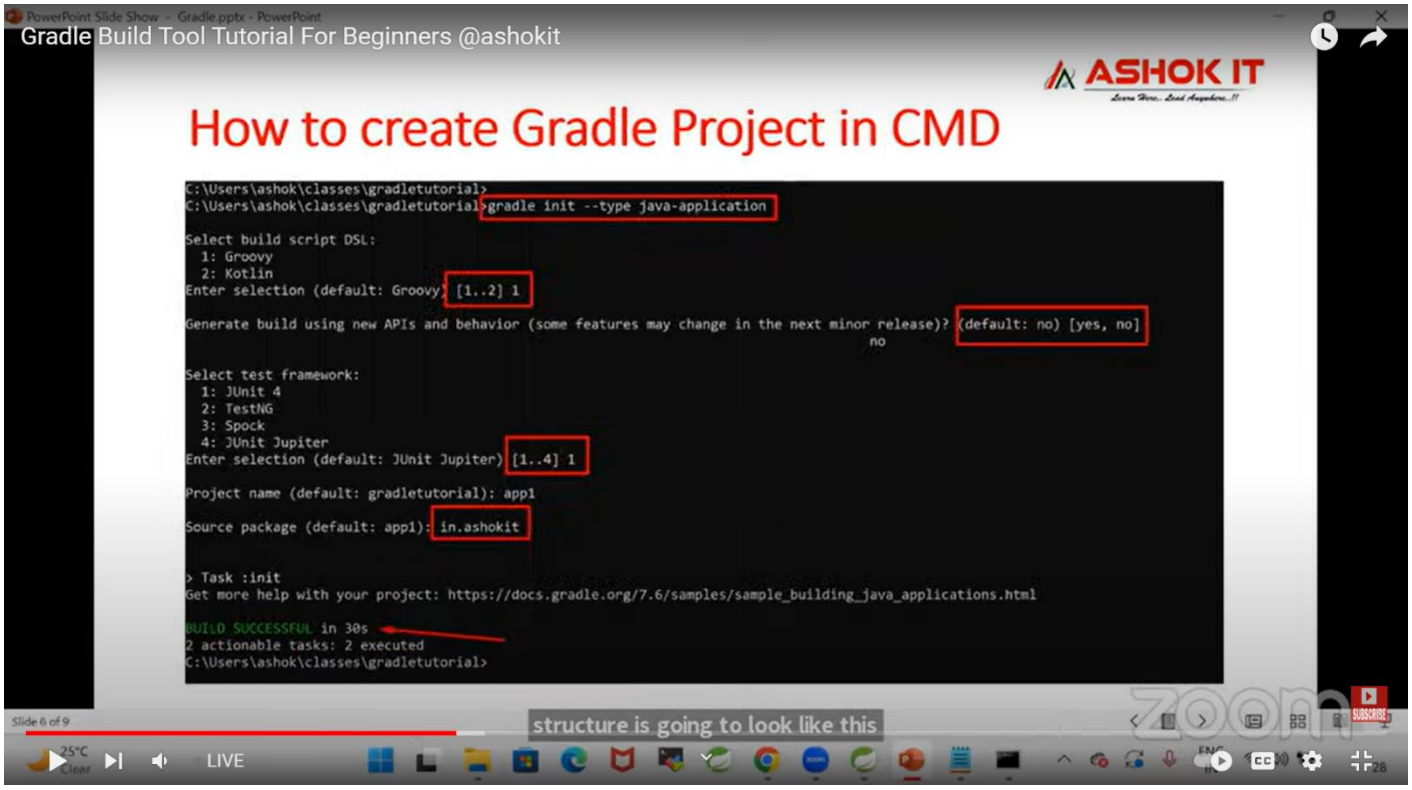
**********GRADLE**********

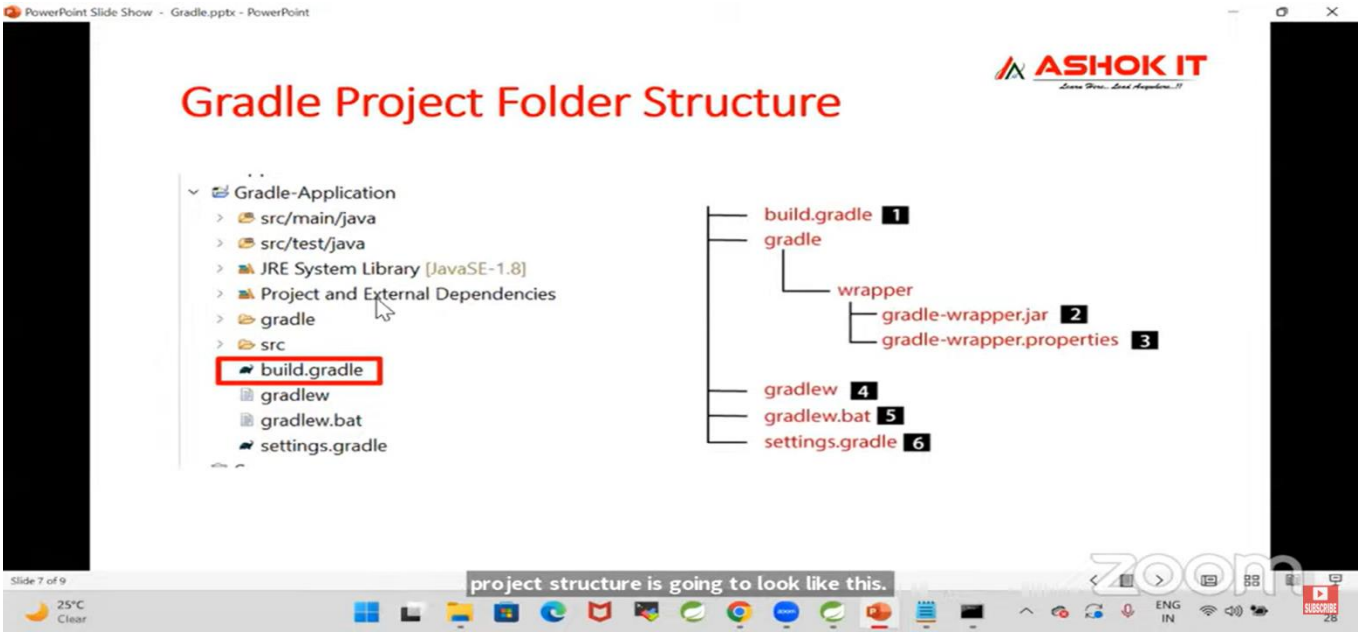

PowerPoint Slide Show - Gradle.pptx - PowerPoint
ASHOK IT
Gradle Workshop Agenda
• What is Build Tool
• Why we need Build Tools
• Maven Vs Gradle
• Gradle Setup
• Creating Gradle Project
• Gradle Project Folders
• Build Scripts & Tasks
• Adding Dependencies
• Adding Repositories
• Gradle Wrapper
• Building Java Projects
• Conclusion
how to set up a gradel software in our system, and how
Slide 2 of 9
zoom


Gradle Build Tool Tutorial For Beginners @ashokit
ASHOK IT
What is Build Tool ?
• A build tool is a software tool that automates the process of building, testing, and deploying software.
• A build tool takes the source code of a software application, and through a series of steps, generates the final executable or deployable package.
• Build tools can also automate many of the repetitive and time-consuming tasks associated with software development, such as managing dependencies, performing code analysis, and running tests.
So whenever you create a gradual project, the project is
Slide 3 of 9
LIVE
zoom

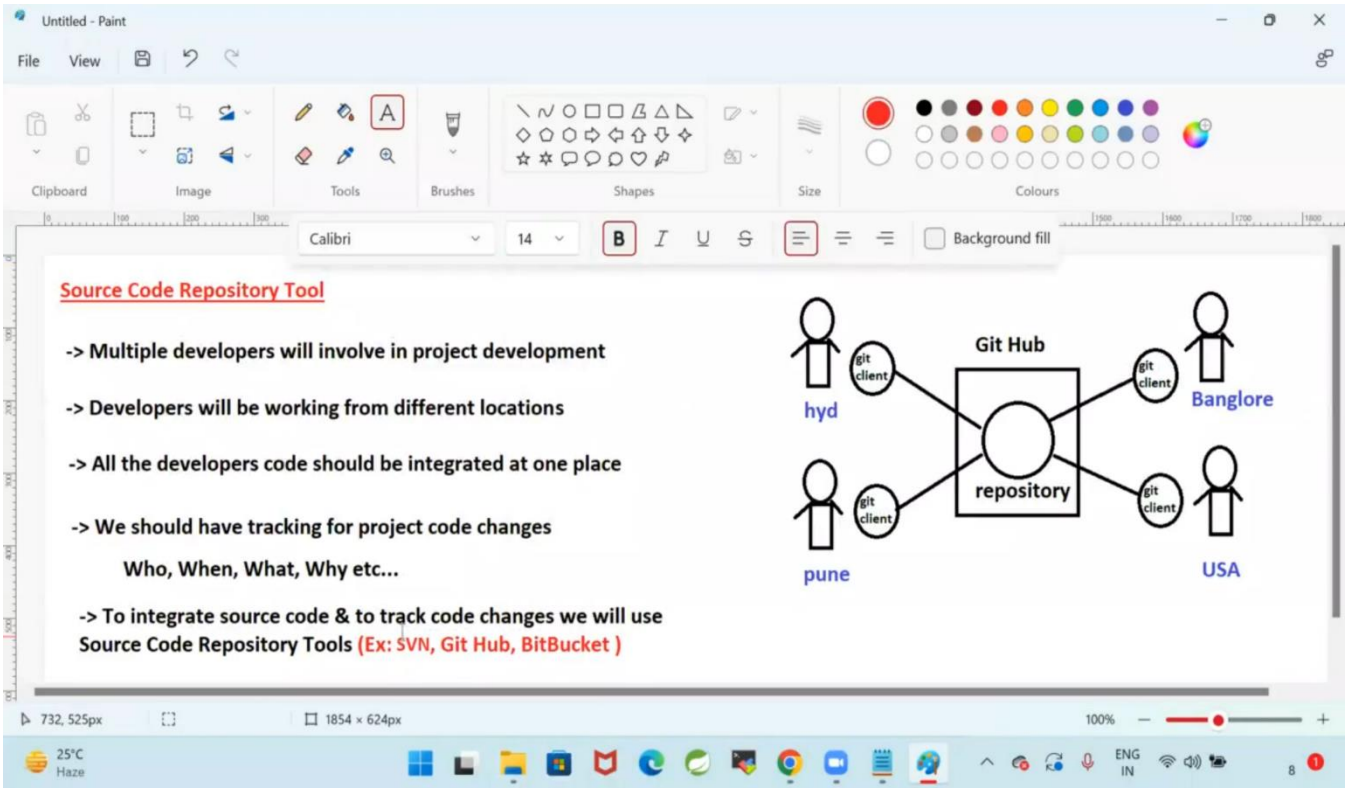

Gradle Build Tool Tutorial For Beginners @ashokit
ASHOK IT
What is Gradle ?
• Gradle is an open-source build automation tool for Java and other programming languages.
• It is similar to Apache Maven and is used to build and manage projects.
• Gradle uses a domain-specific language (DSL) based on the Groovy programming language.
So whenever you create a gradual project, the project is
Slide 4 of 9
LIVE
zoom

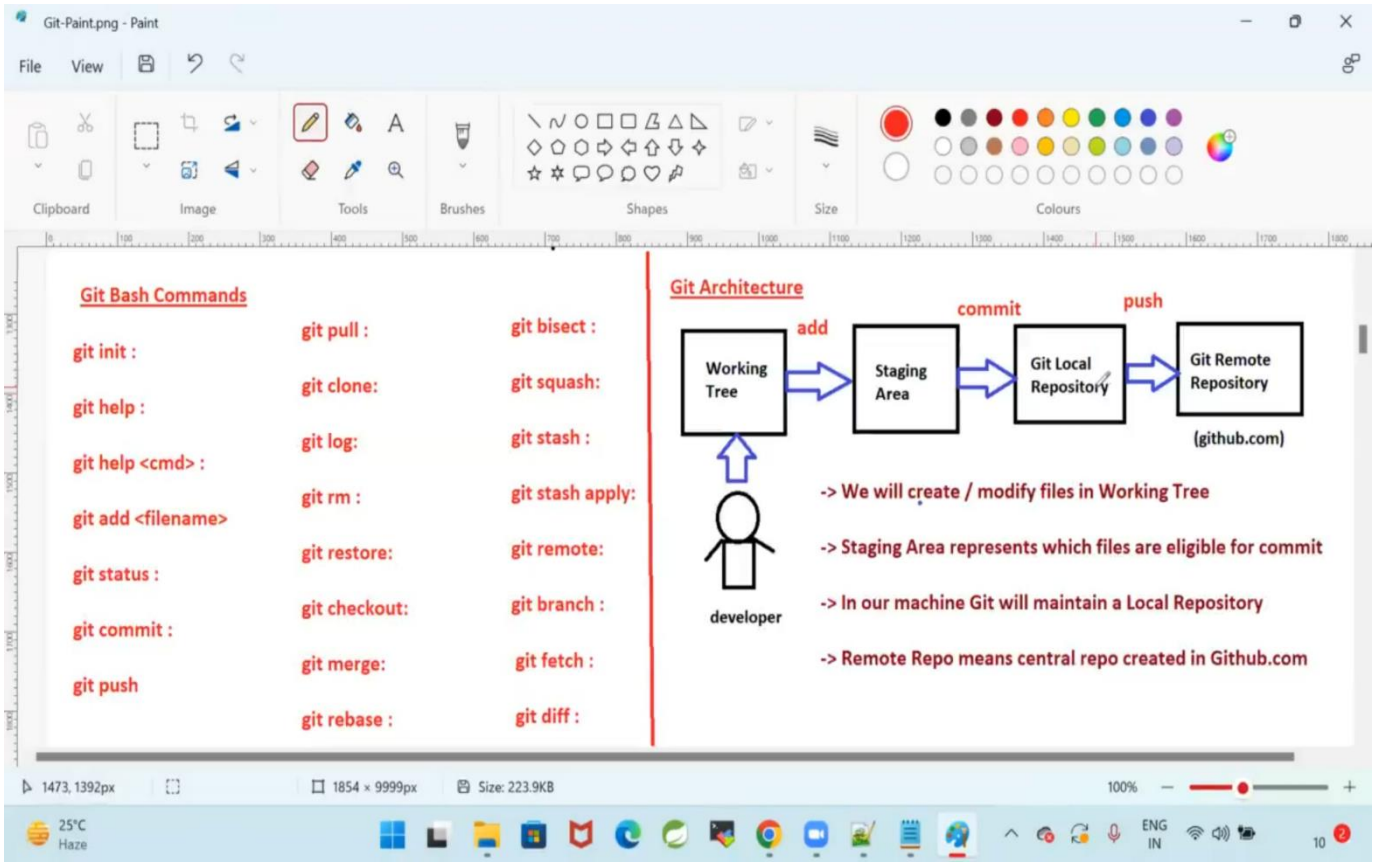

Gradle Build Tool Tutorial For Beginners @ashokit
How to setup Gradle in Windows ?
• Check Java installation (Java is mandatory to run Gradle)
• Download Gradle Zip file
• Set Path for Gradle Bin directory in Environment variables
C:\Users\ashok\classes\softwares\gradle-7.6\bin
• Check Gradle Version
$ gradle --version
ASHOK IT
So whenever you create a gradual project, the project is

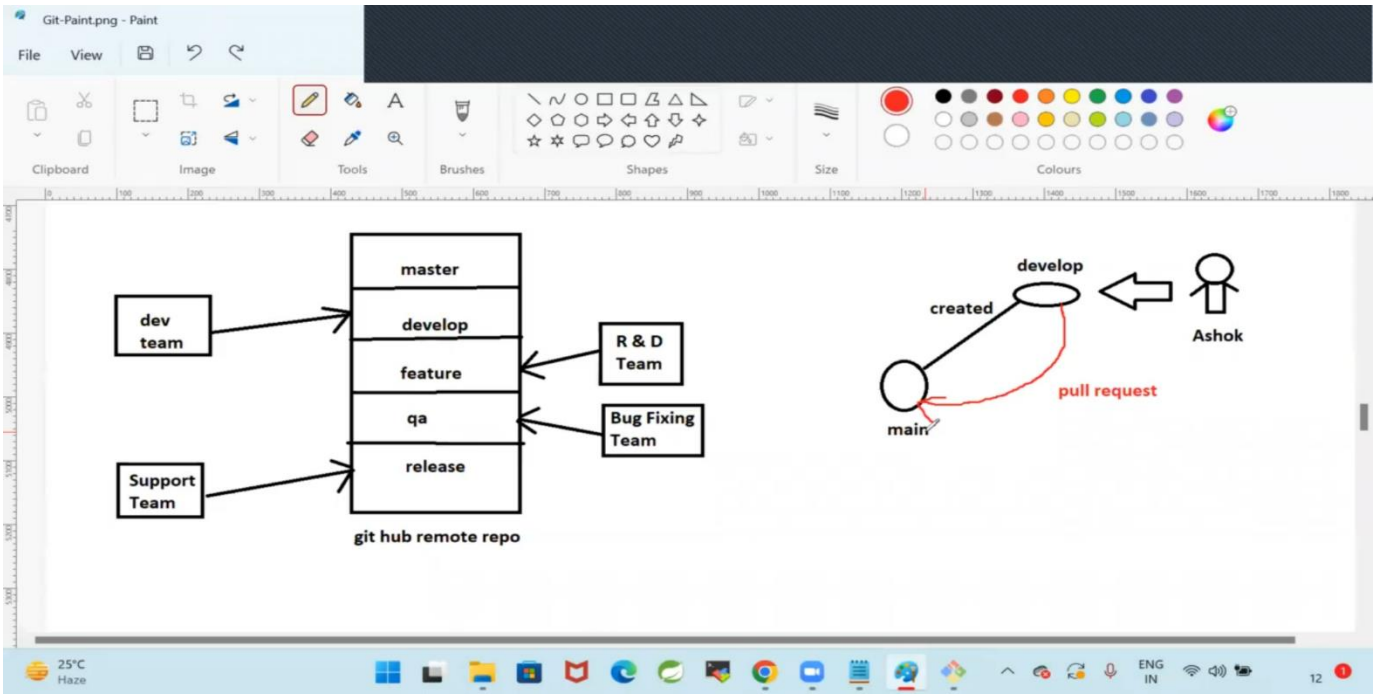

Gradle Build Tool Tutorial For Beginners @ashokit
ASHOK IT
How to create Gradle Project in CMD
C:\Users\ashok\classes\gradletutorial>
C:\Users\ashok\classes\gradletutorial>gradle init --type java-application
Select build script DSL:
1: Groovy
2: Kotlin
Enter selection (default: Groovy) [1..2] 1
Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no]
no
Select test framework:
1: JUnit 4
2: TestNG
3: Spock
4: JUnit Jupiter
Enter selection (default: JUnit Jupiter) [1..4] 1
Project name (default: gradletutorial): app1
Source package (default: app1): in.ashokit
> Task :init
Get more help with your project: https://docs.gradle.org/7.6/samples/sample_building_java_applications.html
BUILD SUCCESSFUL in 30s
2 actionable tasks: 2 executed
C:\Users\ashok\classes\gradletutorial>
structure is going to look like this


PowerPoint Slide Show - Gradle.pptx - PowerPoint
ASHOK IT
Gradle Project Folder Structure
Gradle-Application
src/main/java
src/test/java
JRE System Library [JavaSE-1.8]
Project and External Dependencies
gradle
src
build.gradle
gradlew
gradlew.bat
settings.gradle
build.gradle 1
gradle
wrapper
gradle-wrapper.jar 2
gradle-wrapper.properties 3
gradlew 4
gradlew.bat 5
settings.gradle 6
project structure is going to look like this.

********** GIT **********


Untitled - Paint
File
View
Source Code Repository Tool
-> Multiple developers will involve in project development
-> Developers will be working from different locations
-> All the developers code should be integrated at one place
-> We should have tracking for project code changes
Who, When, What, Why etc...
-> To integrate source code & to track code changes we will use
Source Code Repository Tools (Ex: SVN, Git Hub, BitBucket)
Git Hub
repository
git client
hyd
pune
Banglore
USA


Git-Paint.png - Paint
File
View
Git Bash Commands
git init :
git help :
git help <cmd> :
git add <filename>
git status :
git commit :
git push
git pull :
git clone:
git log:
git rm :
git restore:
git checkout:
git merge:
git rebase :
git bisect :
git squash:
git stash :
git stash apply:
git remote:
git branch :
git fetch :
git diff :
Git Architecture
Working Tree
add
Staging Area
commit
Git Local Repository
push
Git Remote Repository
(github.com)
developer
-> We will create / modify files in Working Tree
-> Staging Area represents which files are eligible for commit
-> In our machine Git will maintain a Local Repository
-> Remote Repo means central repo created in Github.com



==========

Source Code Repository Tools (or) Version Control Softwares

==========

-> Multiple developers will work for project development
-> Developers will be working from multiple locations
-> All developers code should be stored at one place (Code Integration Should Happen)
-> To integrate all the developers source code at one place we will use Source Code Repository Softwares

==========

Advantages with Source code repository sofwares

==========

1) All the developers can connect to repository server and can integrate the code
2) Code Integration will become easy
3) Repository server will provide monitored access

- Who
- When
- Why
- What

==========

Repository Tools

==========

SVN (outdated)
Git Hub
BitBucket

==========

Environment Setup to work with Git Hub

==========

1) Create Github account (www.github.com)
2) Download and install Git Client software (https://git-scm.com/downloads)
3) Once installation completed, right click on the mouse and verify git options display (If git options displaying our git client installation completed successfully)

==========

Working with GitHub

==========

-> Login into github account with your credentials
-> Create Repository in github
Note: Repository is used to store project source code. Every Project will have one repository

-> When we create a repository, unique URL will be generated with Repository Name (i.e Repo URL)
Ex: https://github.com/ashokitschool/hdfc_bank_app.git

-> All the developers will connect to repository using Repository URL
-> We can create 2 types of Repositories in Git Hub

1) public repository
2) private repository

-> Public Repository means everybody can access but we can choose who can modify our repository
-> Private Repository means we will choose who can access and who can modify

Repo URL : https://github.com/ashokitschool/01-devops-app.git

==========

Working with Git Bash

==========

-> Git Bash we can use as Git Client software to perform Git Operations
-> Download and install git client (https://git-scm.com/downloads)
-> Right Click on Mouse and choose "Open Git Bash Here"

git help : It will display frequently used git commands
git help <cmd-name> : It will open documentation for given command

==========

Configure Your Email and Name in Git Bash with Commands

==========

$ git config --global user.email "youremail@yourdomain.com"
$ git config --global user.name "name"

Note: email and name we will configure only for first time.

$ git init : To initialize our folder as git working tree folder
$ git clone : To clone git repository to our machine from github.com

Syntax : $ git clone <project-repo-url>

$ git status : It will display staged , un-staged and un-tracked files

Syntax : $ git status

Staged Files : The files which are added for commit
Un-Staged Files : The files which are modified but not added for commit
Un-tracked files : Newly created files

Note: To commit a file(s), we should add to staging area first

$ git add : It is used to add file(s) to staging area

Syntax : $ git add <file-name>

$ git add .

$ git add --a

$ git commit : It is used to commit staged files to git local repository

Syntax : $ git commit -m 'reason for commit'

$ git push : To push changes from git local repository to git central repository

Syntax : $ git push

$ git rm : To remove file(s) from repository

==========

Steps to push code to github central repository

==========

1) Create one public repository in git hub (take github repo url)
2) Clone github repository using 'git clone' command

$ git clone 'repo-url'

3) Navigate to repository folder
4) Create one file in repository folder

$ touch Demo.java

5) Check status of the file using 'git status' command

$ git status (It will display as untracked file)

6) Add file to staging area using 'git add' command

$ git add .

7) Commit file to git local repository

$ git commit -m 'commit-msg'

8) Push file from git local repository to git central repository using 'git push' command

$ git push

Note: If you are doing 'git push' for first time it will ask to enter your github account password.

Note: Git bash will ask our password only for first time. It will save our git credentials in Credential Manager in Windows machine.

-> Go to Credential Manager -> Windows Credentials -> Select Github -> We can modify and delete saved credentials from here

---

-> When we do git commit then it will geneate a commit-id with 40 characters length

-> From this commit-id it will display first 7 characters in git hub central repository

-> We can check commit history using 'git log' command

==========

Steps to commit Maven Project to Github Repository

==========

1) Create Maven Project
2) Create GitHub Repository

Note: After creating git repository, it will display set of commands to execute

3) Open gitbash from project folder and execute below commands
$ git init
$ git add .
$ git commit -m 'commit-msg'
$ git branch -M main
$ git remote add origin <repo-url>
$ git push -u origin master

Git Stash:-

When we are working on one task suddenly we may get some other priority task.

Usecase
++++++++
-> ager assigned task id : 101
-> I am working on that task (i am in middle of the task)
-> Manager told that stop the work for 101 and complete 102 on priority.
-> Once 102 is completed then resume your work on 101

-> When manager asked me to start 102 task, i have already done few changes for 101 (partially completed)

-> We can't push partial changes to repository because with our partial changes existing functionality may break.

-> We can't delete our changes because we have spent few hours of time to implement those changes

********* In this scenario we will go for 'git stash' option **********

=> Git stash is used to save working tree changes to temporary location and make working tree clean.

-> After priority work completed we can get stashed changes back using 'gitstash apply'

==========

Git Branches

==========

-> Branches are used to maintain seperate code bases for our project
-> In GiT repository we can create multiple branches

main
develop
qa
release
research

-> development team will integrate the code in 'develop' branch
-> bug-fixing team will integrate the code in 'qa' branch
-> R & D team will integrate the code in 'research' branch

-> In github we can create branches
-> To clone particular branch in git repo we will use below command

```
$ git clone -b <branch-name> <repo-url>
```

==========

Branch Merging

==========

=> The process of merging changes from one branch to another branch is called as Branch merging
=> We will use Pull Request for Branch Merging

==========

Steps to do branch merging

==========

1) Create feature branch from main branch
2) clone feature branch
3) create new file in feature branch then commit and push to central repo

4) Go to central repository then create pull request to merge feature branch changes to main branch

Note: Once feature branch changes are merged to main branch then we can delete feature branch (if required)

==========
What is .gitignore ?
==========

-> This .gitignore is used to configure the files or folders which we want to ignore from our commits

-> The files and folders which are not required to commit to central repository those things we can configure in .gitignore file

Ex: In maven project, 'target' folder will be available which is not required to commit to central repository. This we can configure in .gitignore file.

----------.gitignore----------
HELP.md
target/
!.mvn/wrapper/maven-wrapper.jar
!**/src/main/**/target/
!**/src/test/**/target/

STS
.apt_generated
.classpath
.factorypath
.project
.settings
.springBeans
.sts4-cache

IntelliJ IDEA
.idea
*.iws
*.iml
*.ipr

NetBeans
/nbproject/private/
/nbbuild/
/dist/
/nbdist/
/.nb-gradle/
build/
!**/src/main/**/build/
!**/src/test/**/build/

VS Code
.vscode/

==========
git merge vs git rebase
==========

=> These commands are used to merge changes from one branch to another branch
-> git merge will preserve commit history
-> git rebase will not preserve that rebase history

-> When we are working on particular sprint and we want to merge changes from one branch to another branch then we will use 'git merge' command

-> Once sprint-1 is delivered then we want to take latest code of sprint-1 to start sprint-2 development. In this scenario we don't need commit history so we will use 'git rebase' command.

==========
What is git pull command
==========

-> pull command is used to take latest changes from repository to local.
-> When we want to make some changes to code, it is always recommended to take git pull first then start your changes.

Note: When we execute 'git pull' there is a chance of getting conflicts. We need to resolve the conflict and we should push the code without conflicts.

==========

git diff : It is used to compare the files (local file & repository file)
git squash : It is used to combine multiple commits into single commit
git bisect : It is used to identify the bad commit which introduced bug.

==========

==========
Git Hub Lab Task
==========

1) Create Maven Web Application
2) Add 'Spring-Core' dependency in project pom.xml file (www.mvnrepository.com)
3) Package maven project as war file using maven goal
4) Create Git repository in github.com (public repo)
5) Push maven project into github repo using gitbash
   (target folder shouldn't be commited, add this in .gitignore file)
6) Make changes in pom.xml and push changes to github repo using git bash
7) Create 'feature' branch in git repo from main branch
8) Make changes in 'feature' branch pom.xml file
9) Switch to feature branch from git bash and push changes to central repo
10) Create pull request and merge 'feature' branch changes to 'main' branch

==========

Summary

==========

1) What is Source Code repository
2) Why we need source code repository
3) What are the source code repositories available
4) What is git hub
5) What is git
6) What is Repository
7) Public Repository vs Private Repository
8) Cloning Repository
9) Staged vs Unstaged vs Untracked File

10) Adding Files to Stating Area
11) Unstaging the files from staging
12) Discarding local changes
13) What is working tree
14) What is Local Repostiory
15) What is Central Repository
16) Commit from working tree to local repo (git commit -m 'msg')
17) push from local repo to central repo (git push)
18) Taking latest code changes
19) push vs pull and clone vs pull
20) What is conflict
21) How to resolve conflicts
22) What is branch in git hub
23) How to create branches
24) How to clone particular branch
25) how to switch to particular branch
26) How to merge branches
27) What is pull request
28) git merge vs rebase
29) what is .gitignore
30) Working with Bitbucket

git init
git help
git config
git clone
git status
git add .
git add <file-name>
git restore
git commit
git push
git pull
git log
git rm
git branch
git checkout
git fetch
git merge
git rebase

********** JIRA **********

==========

What is Software Development Life Cycle ?

==========

-> The SDLC refers to a methodology
-> SDLC clearly defines the process of creating high-quality software project
-> SDLC methodology focuses on below phases of software development

1) Requirements Analysis
2) Planning
3) Software Design or Architectural Design
4) Development
5) Testing
6) Deployment
7) Support

-> We have several methodologies in SDLC

1) Waterfall Model
2) V-Model
3) Spiral Model
4) Agile Model

==========

Waterfall Model

==========

-> Waterfall Model introduced in 1970
-> The waterfall Model was the first model that introduced as Process Model.
-> It is a linear or sequential approach to develop software projects
-> It is also called as step By step approach to develop the projects
-> In this model we will complete one phase then we will move on to next phase

Note: Going back to previous phase is not allowed in waterfall model.

==========

When to use Waterfall

==========

-> When requirements are fixed
-> When Project is small
-> When Project budget is fixed

==========

Dis-Advantages of Waterfall

==========

-> Risk is high
-> Requirements can't be changed in middle
-> We can't go to previous phase
-> Testing will happen at the end
-> Client involvement is very very less

##########

Agile Methodology

##########

-> Agile is one of the SDLC methodology
-> It is a way to manage the project by breaking into several phases
-> Agile is also called as an iterative approach to develop and deliver the project
-> We can achieve continuous development and continuous delivery using Agile Model
-> It involves constant colloboration with stake holders (clients)
-> Continuous improvement at every stage
-> Client Feedback is most critical part in Agile Methodology
-> In Agile, Planning, Implementing, Testing and Delivering is a cyclic process

==========

Agile Terminology

==========

Backlog Grooming : A Team meeting to identify pending works in the project.

Stories : Tasks (work)

Backlog Stories: Pending Stories in the project.

Note: In backlog grooming meeting, team members will create backlog stories

Story Points : Duration to complete the story (3 points - 1 day, 5 points - 2 days,8 Points - 3 days)

Sprint Planning : A Team meeting to identify priority stories to work in the Sprint.

Sprint : Set of stories to complete (Sprint Goal) & Sprint Duration - 2 weeks (10 working days)

Scrum : It is a process (Daily team meeting to discuss work status for 15 - 20 minutes)

Retrospective : Review meeting after few sprint(s) completed (What went well, What went wrong, achievements, improvements, lessons learnt, new ideas)

Note: We have monthly one retrospective meeting.

Product Owner : Responsible for Deliverables

Scrum Master : Who will manage Agile teams

Tech Lead : Seniour Team Members (Who will help team members technically)

Team Members : Developers & Testers

Agile Team Size : 7 - 10 members (Industry Standards)

=======
JIRA
=======

-> Jira is called as project management software
-> In companies, project work will be managed by using JIRA s/w
-> JIRA software is also used for Bug Reporting
-> JIRA software developed by Atlasian company (BitBucket also from same company)
-> JIRA is a commercial software (we need to purchase license to use for commerical purpose)

Note: We can use JIRA trail version for practise. In IT companies we will use paid version of JIRA.

==========
JIRA Assignment
==========

1) Login into JIRA
2) Create Project in JIRA
3) Create Backlog Stories in JIRA
4) Drag the backlog stories to Sprint
5) Start the Sprint
6) Assign the story to your name and change story status to In-Progress
7) Add story comments and change status to Done

Note: In few projects, scrum master will assign stories to team members to work.

-> In few projects no body will assign the story to us, we have to assign story to ourself and we have complete it.

-> After story is completed we have to change story status to DONE.
Note: Every day we have to give our story status to scrum master in scrum call.

-> If we are not able to complete any story with in given time then we need to explain the reason in the scrum call.

Note: If we don't have any stories to work then we need to discuss with Scrum Master regarding work.

1) Who will give story points in story ?

Ans) The person who creates a story same person will give story points

2) What is the difference between Team Lead & Tech Lead...

Ans) Team Lead is responsible to assign works for the team members and collect work status...
Tech Lead is respoinsible to provide solutions for the team members technically

Waterfall Model --------> Team Lead
Agile Model ----------> Tech Lead

3) What is the duration of Sprint ?

Ans) It depends upon project management team
Note: Mostly projects will have 2 weeks sprint (10 working days)

4) When testing will happen in sprint ?

Ans) In Agile methodology Development and Testing is a parallel process

Note: In every sprint, dev team stories and testing team stories also will be available. Both teams will work paralelly

5) Who will assign daily tasks to you?

Ans) Self assignment process is there in our project. We need to check pending stories in sprint and we need to assign to ourself and we need to start working on it.

6) If you are unable to complete a story as per estimations then what you will do?

Ans) I will explain to my scrum master why i am unable to complete the story with in given time and i will explain what are the challenges i am facing to complete it. I will justify the reason for not completing.

7) What is Tech Debt in Agile methodology?

Ans) If we are facing techical challenges to complete a story which requires more analysis to complete is called as Tech Debt.

8) If there are no pending stories in Sprint then what you will do?

Ans) I will inform to scrum master regarding this and i will follow my scrum master instructions. In free time i will explore on new technologies / apis / frameworks which will help to improve our productivity in the project.

Maven
Git Hub
Bit Bucket
Nexus Repo
Jenkins CI CD
Logging
Log Monitoring
Project Lombok
Debuggnig
Agile Methodology
JIRA
Docker -----> Running
Angular ----> Running

********** SWAGGER **********

========
Swagger
========

-> Swagger is a third party api
-> Swagger is used to generate documentation for our REST APIs

Q) What is REST API documentation ?

Ans) REST API documentation will provide all the information about API like below

a) API Base URL
b) Endpoints of API
c) Request Data Format
d) Response Data Format

-> API documentation will play major role in distributed applications development.

Note: If we want to access any API then we need documentation for that. Based on the documentation we can understand api and its structure.

==========
Use-Case
==========

-> IRCTC project developed by one company to book train tickets
-> MakeMyTrip applications developed by another company to book tickets.
-> MakeMyTrip wants to communicate with IRCTC to book train tickets.
-> To communicate with IRCTC, MakeMyTrip project developers should understand IRCTC API details.

a) What is URL ?
b) What are the Endpoints ?
c) Request Data Structure
d) Response Data Structure

-> If MakeyMyTrip developers knows above details then only they can write the logic to access IRCTC api.

Note: IRCTC team should provide API documentation to MakeMyTrip team.

-> Swagger-UI is used to test our REST APIs (We can use this as alternative to POSTMAN)

==========

How to setup Swagger in Spring Boot Application ?

==========

1) Add Swagger & Swagger-UI dependencies in pom.xml file

```
<dependency>
	<groupId>io.springfox</groupId>
	<artifactId>springfox-swagger2</artifactId>
	<version>2.9.2</version>
</dependency>

<dependency>
	<groupId>io.springfox</groupId>
	<artifactId>springfox-swagger-ui</artifactId>
	<version>2.9.2</version>
</dependency>
```

########## 2) Create Swagger Config class ###########

```
package in.ashokit.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
```

```
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

@Configuration
@EnableSwagger2
public class SwaggerConfig {

	@Bean
	public Docket apiDoc() {

		return new Docket(DocumentationType.SWAGGER_2)
				.select()
				.apis(RequestHandlerSelectors.basePackage("in.ashokit.rest"))
				.paths(PathSelectors.any())
				.build();
	}
}
```

####### 3) Configure below property in application.properties file ########

```
spring.mvc.pathmatch.matching-strategy = ANT_PATH_MATCHER
```

4) Run the spring boot application and access below URL

URL : http://localhost:8080/swagger-ui.html

==========

********** Sonar Qube **********

==========
Sonar Qube
==========

-> Sonar Qube is Continuous Code Quality Checking Tool
-> We can do Code Review using Sonar Qube tool

==========
What is Code Coverage & Code Review
==========

Code Coverage : How many lines of source code is tested by unit test cases
Note: Industry standard Code Coverage is 80 %
Code Review : Checking Coding Conventions / Standards

-> Sonar Qube is an open source, software quality management tool
-> It will continuosly analyze and measures quality of the source code
-> It will generate code review report in html format / pdf format
-> It is a web based tool and it supports 29 Programming Languages

-> It will support multi OS platform
-> It will support multiple databases (MySQL, Oracle, SQL Server, PostGres SQL...)
-> It supports multiple browsers
-> Sonar Qube will identify below category of issues in project source code

1) Duplicate Code
2) Coding Standards
3) Unit Tests
4) Code Coverage
4) Complex Code
5) Commented Code
6) Potential Bugs

=> Initially Sonar Qube was developed only for Java Projects
=> Today Sonar Qube is supporting for 29 Languages

==========
Environment Setup
==========

-> Java is the pre-requisite software
7.6 --> Java 1.8v
7.8 - 8.x --> Java 11v

Note: We can check this compatability in official sonar website

==========
Hardware Requirements
==========

Minimum RAM : 2 GB
t2.medium ---> 4 GB RAM

==========
SonarQube Server Setup in Linux VM
==========

-> Create EC2 instance with 4 GB RAM (t2.medium) (Amazon Linux AMI)
-> Connect with EC2 instance using MobaXterm
-> check space (free -h)

$ sudo su
$ cd /opt
$ sudo yum install java-1.8.0-openjdk
$ java -version
$ wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-7.8.zip
$ unzip sonarqube-7.8.zip

********** Note: SonarQube server will not run with root user **********

-> Create new user in ec2 instance
$ useradd sonar
$ visudo

-> Configure sonar user without pwd in suderos file

sonar ALL=(ALL) NOPASSWD: ALL

# Change ownership for sonar folder/directory
$ chown -R sonar:sonar /opt/sonarqube-7.8/
$ chmod -R 775 /opt/sonarqube-7.8
$ su - sonar

-> Goto bin directory then goto linux directory and run sonar server

$ cd /opt/sonarqube-7.8/bin/linux-x86-64
$ sh sonar.sh start
-> Check sonar server status
$ sh sonar.sh status

Note: Sonar Server runs on 9000 port number by default
Note: We can change default port of sonar server (conf/sonar.properties)
Ex: sonar.web.port=6000

-> Enable port number in EC2 VM - Security Group
-> Access Sonar Server in Browser
URL : http://EC2-VM-IP:9000/

-> Default Credentials of Sonar User is admin & admin
-> After login, we can go to Security and we can enable Force Authentication.

Note: Once your work got completed then stop your EC2 instance because we have t2.medium so bill be generated.

$ sh sonar.sh status

Note: If sonar not started, then go to log file and see

$ sudo rum -rf /opt/sonar-folder/temp/
$ cd ../bin/
$ sh sonar.sh start
$ sh sonar.sh status

-> Access sonar server in browser and login into that

==========

Integrate Sonar server with Java Maven App

==========

-> Configure Sonar Properties under <properties/> tag in "pom.xml"

```
<properties>
        <sonar.host.url>http://13.233.54.51:9000/</sonar.host.url>
        <sonar.login>admin</sonar.login>
        <sonar.password>admin</sonar.password>
</properties>
```

-> Go to project pom.xml file location and execute below goal

$ mvn sonar:sonar

-> After build success, goto sonar dashboard and verify the results

Note: Instead of username and pwd we can configure sonar token in pom.xml

==========

Working with Sonar Token

==========

-> Goto Sonar Server Dashboard -> Login -> Click on profile -> My Account -> Security -> Generate Token

-> Copy the token and configure that token in pom.xml file like below

```
<sonar.host.url>http://13.233.106.91:9000/</sonar.host.url>
<sonar.login>a14f5b5f781b968af4e7befbdd74cfbd6e96023a</sonar.login>
```

-> Then build the project using "mvn sonar:sonar" goal

==========

Quality Profile

==========

-> For each programming language sonar qube provided one quality profile with set of rules.
-> Quality Profile means set of rules to perform code review
-> We can create our own quality profile based on project requirement
-> Create One Quality Profile

- Name : SBI_Project
- Language: Java
- Parent : None

Note: We can make our quality profile as default one then it will be applicable for all the projects which gets reviewed under this sonar server.

Note: If we have any common ruleset for all projects then we can create one quality profile and we can use that as parent quality profile for other projects.

-> We can configure custom quality profile to a specific project using below steps

- click on project name
- Go to administration
- Click on quality profile
- Select profile required

==========

Quality Gate

==========

-> Quality Gate represents set of metrics to identify project quality is Passed or Failed
-> Every Project Quality Gate should be passed
-> In Sonar We have default Quality Gate
-> If required, we can create our own Quality Gate also

##########

Lessons Learnt in Code Review

##########

1) String which we want to compare should present at left side

```
if (userAcc.getAccStatus().equals("LOCKED"))  // bad-practise
if ("LOCKED".equals(userAcc.getAccStatus()))  // good practise
```

2) Replace StringBuffer with StringBuilder to improvate performance

StringBuffer -> is synchronized (only one thread can access at a time)
StringBuilder -> is not-syncronized (multiple threads can access at a time)

3) Either log or re-throw the exception

```
//bad practise
catch (Exception e) {
        e.printStackTrace( );
}

//good practise
catch (Exception e) {
        logger.error("Exception :: "+e.getMessage(), e);
}
```

4) Instead of Math.random( ) use java.util.Random.nextInt ( ) method to generate random number

6) Don't use "password" or "pwd" in our code directley. It will be considered as vulnerable. Use "pazzword" or "pzzwd" for variable names.

7) Declare variables before constructor

8) Remove un-necessary curly braces from lambda when we have single line

9) follow camel case for variables names declaration (Ex: userAccStatus)

10) Declare private constructor for class if it is not getting instantiated anywhere.

11) Use Constants class to declare String literals

12) Don't wirte too many lines of code in single method

13) Cognitive Complexity (Code is difficult to understand)

14) Cyclomatic Complexity (Code is difficult to unit test)

==========

Conclusion

==========

-> If project quality gate is failed then we should not accept that code for deployment.

-> If project is having Sonar issues then development team is responsible to fix those issues

-> DevOps Enginners we will perform Code Review and we will send Code Review report to Development team

********** Jenkins **********

==========

Jenkins Workshop Agenda

==========

What is Build & Deployment Process
Application Environments In Real-Time
Challenges in Manual Build & Deployments
What is CI CD & Why we need ?
Jenkins Introduction
Jenkins Setup
Jenkins Job Creation
Job Scheduling
Conclusion

==========

Build & Deployment Process

==========

Take latest source code from Git Hub
Compile project source code
Execute Unit Test cases
Perform Code Review using SonarQube
Package the application (jar / war)
Upload Build Artifact in Nexus
Deploy the Application in Server

==========

Application Environments:

==========

Environment : A platform which is used to run our application
DEV env : Developers will use to perform code integration testing
QA env : Testers will use to perform functional testing
UAT env : User acceptance testing (client side team will perform application testing)
Pilot env : It is also called as Pre-Prod environment (performance testing)

Prod env : Live environment. End users will access application running in PROD environment.

www.gmail.com ----> Production env url of the project
www.facebook.com ---> Production env url of the project

==========
Challenges in Manaul Build Process
==========

Every day we need to deploy latest code
Deploy code in multiple environments
Takes lot of time
Repeated Work
Error Prone

==========
Project Teams
==========

Devlopment Team : Responsible for project development (Coding)
Testing Team : Responsible for project functionality testing (verification & validation)
Operations Team : Responsible for Build & Deployment process

Dev + Ops ====> DevOps
Development + Operations ==========> DevOps

DevOps is a process which is used to colloborate development team work & operations team work.
Using DevOps process we can simply application Build & Deployment process

==========
Jenkins
==========

-> Jenkins is a free software
-> Jenkins developed using java langauge
-> Jenkins is used to automate build & deployment process
-> We can implement CI CD using jenkins

==========
What is CI CD ?
==========

CI : Continuous Integration
CD : Continuous Delivery / Continuous Deployment

-> CI CD is one of the trending approach in software development life cycle
-> CI CD is used to simplify and automate project deployment & delivery process

Continous Integration : When code changes happend it should be ready to test
Continuous Delivery : Keep it ready in repository for release
Continuous Deployment : Release / deploy the project to Production

Note: For Production deployment we need to take Client Approval.

==========
Part-1 : Summary
==========

1) What is Build & Deployment Process
2) Application Environments
3) Why we need several environments for our application
4) Challenges in Manual Build & Deployment process
5) What is Jenkins
6) What is CI CD

==========
Part-2
==========

==========
Manual Deployment
==========

Git Hub Repo URL : https://github.com/ashokitschool/maven-web-app.git

1) Downloaded code from git hub
2) Executed Maven Goal (clean package) ==> war file created
3) Uploaded war file into tomcat server (deployment)
4) Access application from tomcat dashboard

==========
Infrastructure Setup
==========

-> Create Linux VM using EC2 in AWS cloud and install tomcat server
-> Create Linux VM using EC2 in AWS cloud and install Jenkins server
-> Clone Git Hub Repository

==========
Build & Deployment Process
==========

Download project from git hub
package the project using maven
Maven will create war file

Deploy war file into tomcat (post build action)

Note: Above build & deployment process can be automated using Jenkins

Access application using URL in browser

==========

Jenkins Job Creation Process

==========

1) Login into Jenkins
2) Configure Maven in Global Tool Configuration (Jenkins will download maven)

-> Manage Jenkins
-> Global tools Configurations
-> Add Maven

3) Install 'Deploy To Container' Plugin (To deploy war to tomcat server)

-> Manage Jenkins
-> Manage Plugins
-> Go to available tab
-> search for 'Deploy To Container' plugin
-> Click on install without re-start

Note: Git s/w will be available by default in Jenkins Gloabl Tools Configuration

4) Create Free Style Project
5) Enter Git Repo URL
6) Build Trigger (configure Maven which is added in Global Tools and Provide Maven Goals as clean package)

7) Add Tomcat Server in Post Build Action For deployment
8) Save the Job configuration
9) Run the Job
10) See Tomcat Dashboard (Application should display) and access the application

==========

Poll SCM

==========

-> Click on Job name
-> Click On Configure
-> Configure Poll SCM with cron expression as (* * * * *)
-> Every minute it will check for code changes, if code changes available then jenkins job will run

==========

How to deploy web application ?

==========

-> To deploy a web application we need a server
-> We are using Apache Tomcat as webserver to run web applications
-> Web Application will be packaged as war file
-> WAR file we will keep in tomcat server webapps folder.
-> webapps folder is called as Deployment folder

==========

How to deploy Spring Boot Application ?

==========

-> Spring Boot application will execute from main ( ) method
-> Spring Boot Application can be deployed as jar file
-> We no need to configure external server to run / deploy spring boot application
-> Spring Boot having embedded server to run Spring Boot Application

==========

Running Spring Boot Application In AWS Cloud

==========

1) Create Linux Virtual Machine (Amazon Linux)
2) Connect Linux Virtual Machine using MobaXterm
3) Install Java software

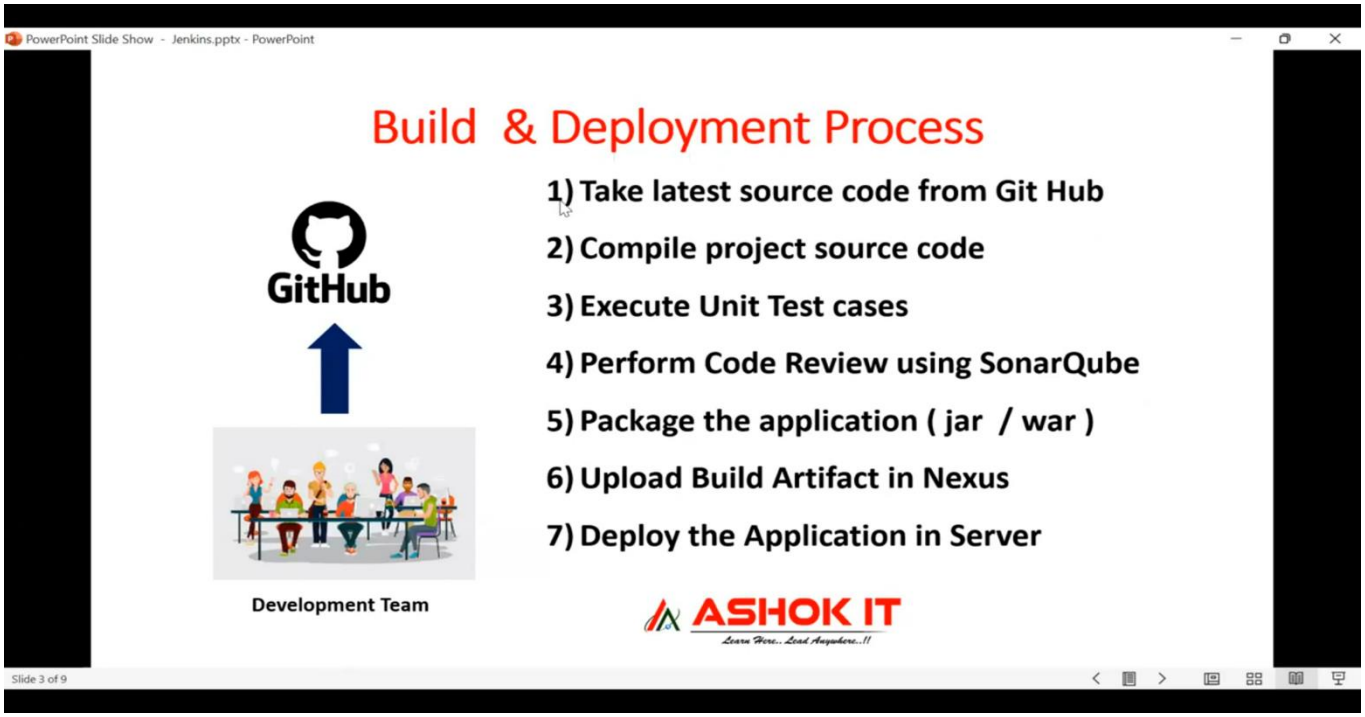
```
$ sudo yum install java
```

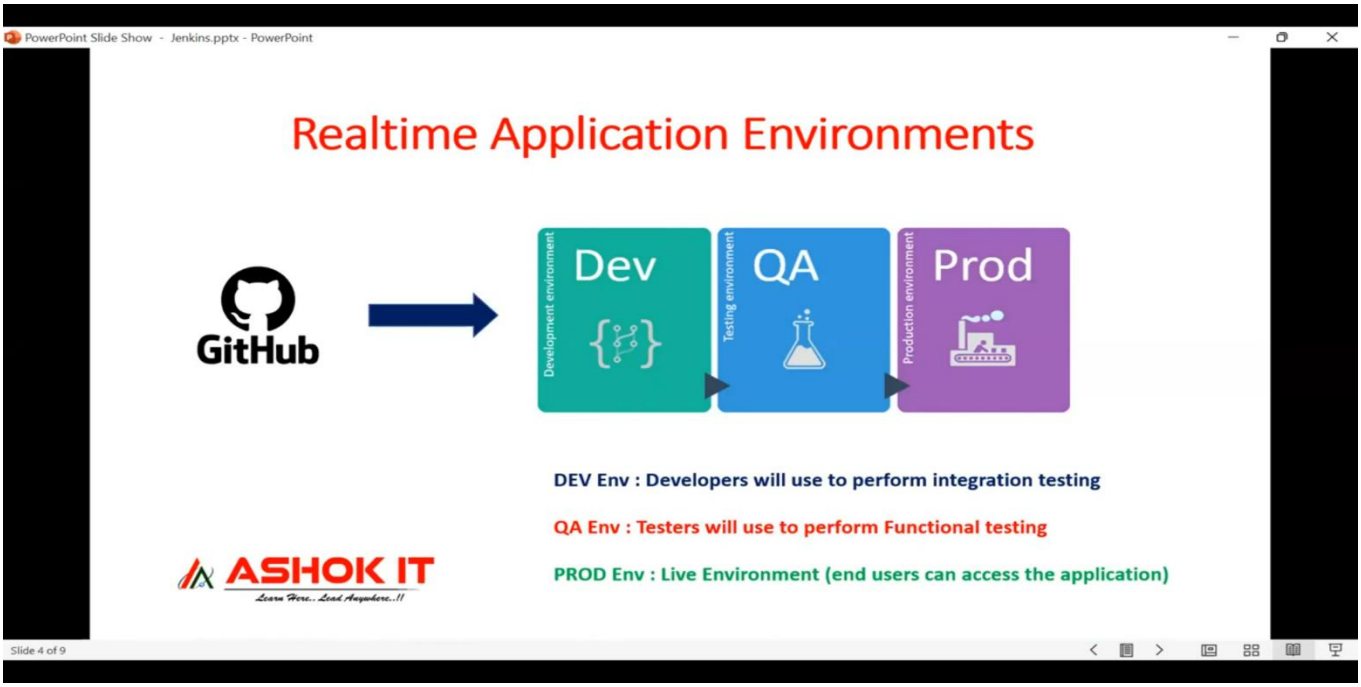
4) Upload Spring Boot Jar file into Linux VM
5) Run Spring Boot Jar file using below command

$ java -jar <jar-file-name>

6) Enable 8080 port number in security group of our Linux Virtual Machine
7) Access our application in browser

URL : http://public-ip:8080/





*Untitled - Notepad

File Edit View

Environment : A platform which is used to run our application

==========
Application Environments:
==========

DEV env : Developers will use to perform code integration testing

QA env : Testers will use to perform functional testing

UAT env : User acceptance testing (client side team will perform application testing)

Pilot env : It is also called as Pre-Prod environment (performance testing)

Prod env : Live environment. End users will access application running in PROD environment.

Ln 7, Col 69 110% Windows (CRLF) UTF-8

18°C Mostly sunny ENG IN 20

PowerPoint Slide Show - Jenkins.pptx - PowerPoint

Challenges in Manual Process

- Every day we need to deploy latest code
- Deploy code in multiple environments
- Takes lot of time
- Repeated Work
- Error Prone

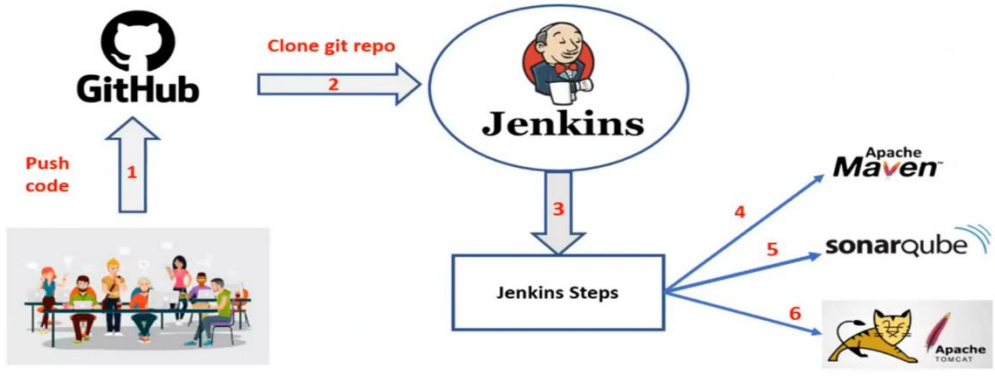
ASHOK IT
Learn Here.. Lead Anywhere..!!

Slide 5 of 9

20°C Polluted air ENG IN 20

PowerPoint Slide Show - Jenkins.pptx - PowerPoint

Build & Deployment - Automation



Slide 6 of 9

20°C Mostly sunny ENG IN 20

PowerPoint Slide Show - Jenkins.pptx - PowerPoint

What is Jenkins ?

- Jenkins is free & open source software
- Jenkins developed using Java language
- Jenkins is used to automate Build & Deployment process
- Using Jenkins we can implement CI CD

ASHOK IT
Learn Here.. Lead Anywhere..!!

Slide 7 of 9

PowerPoint Slide Show - Jenkins.pptx - PowerPoint

What is CI CD ?

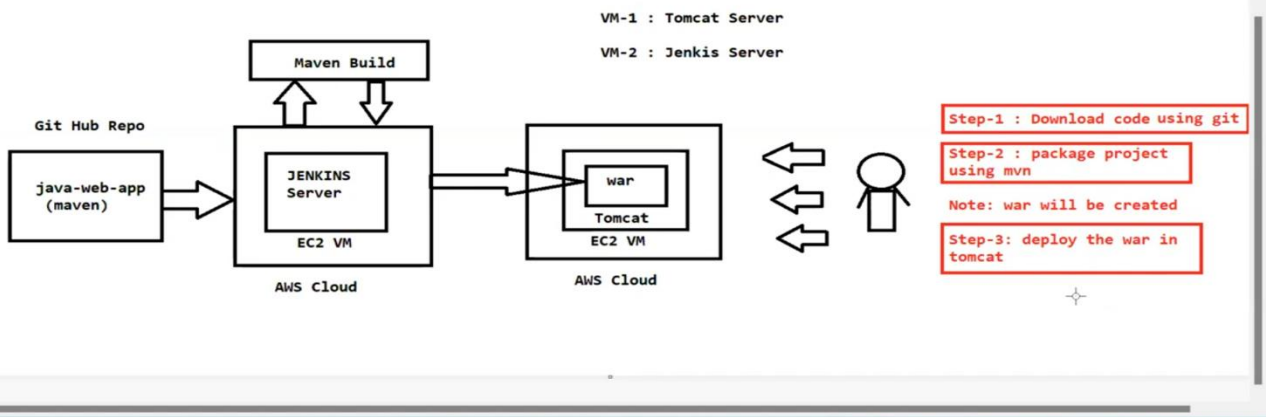
ASHOK IT
Learn Here.. Lead Anywhere..!!

Slide 8 of 9

Untitled - Paint

File View

Clipboard Image Tools Brushes Shapes Size Colours



********** Logging **********

=========
Logging
==========

-> It is the process of storing application execution details into a file / console / database.
-> Logging is very important for every project for easy maintenence by understanding Runtime behaviour of our application..
-> Logging will help us in understanding application execution details and logging will help us in identifying exceptions occured in the application.
-> Using log message we can find root cause of the problem in our application / code.
-> To implement logging in our application we have several logging frameworks, they are below

1) Log4J
2) Log4J2
3) LogBack
4) Logstash
5) SLF4J

-> Once logging completed, we can perform Log Monitoring to read log messages
-> Log Monitoring means reading log messages of log file.
-> To perform log monitoring we have several tools in the market.

1) Putty / MobaXterm
2) WinScp
3) Kibana
4) Splunk (paid s/w)

==========

Logging Architecture

==========

1) Logger
2) Layout
3) Appender

-> Logger is a class which is providing methods to generate log messages

```
Logger logger  =  LoggerFactory.getLogger(UserService.class);
                    logger.debug ("this is debug msg");
                    logger.info("this is info msg");
                    logger.warn("this is warn msg");
                    logger.error("this is error msg");
```

-> Layout represents pattern of the log message (what info should be included in log message)

-> Appender is used to store log messages in destination (console / file / database)

==========

Log Levels

==========

-> In logging we have several log levels to represent priority of log message

1) TRACE
2) DEBUG
3) INFO
4) WARN
5) ERROR
6) FATAL

TRACE > DEBUG > INFO > WARN > ERROR > FATAL

Note: In Spring Boot application, the default LOG LEVEL is INFO.

Note: When we set LOG level, from that level all the next levels logs also will be printed.

Ex:

-> If we set level as INFO, except TRACE & DEBUG every level will be printed.
-> If we set level as WARN, then TRACE, DEBUG and INFO will not be printed.
-> If we set level as ERROR, then ERROR and FATAL will be printed.

==========

Logging in Spring Boot Application

==========

-> In Spring Boot, by default LOG framework dependency will be available (no need to add seperatley)
-> In Spring Boot, the default LOG level is INFO (if required we can change in application.properties file)
-> In Spring Boot, the default Appender is ConsoleAppender (if required, we can change)

==========

Project Setup

==========

-> Create Spring Boot Application using IDE (Eclipse / STS / Intelli J)
-> Develop the classes with Logging like below

```
package in.ashokit.dao;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class ReportDao {

private static final Logger logger = LoggerFactory.getLogger(ReportDao.class);
```

```
	public String getName(Integer id) {
		String name = null;
		logger.info("getName() - method start");
		if (id == 101) {
			name = "John";
		} else if (id == 102) {
			name = "Smith";
		} else {
			name = "Invalid Id";
		}
		logger.info("getName() - method end");
		return name;
	}
}
```

```
package in.ashokit.service;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class MessageService {

private static final Logger logger = LoggerFactory.getLogger(MessageService.class);

	public String getWelcomeMsg() {
		logger.debug("getWelcomeMsg() - method start");
		String msg = "Welcome To Ashok IT..!!";
		logger.debug("getWelcomeMsg() - method end");
		return msg;
	}

	public String getGreetMsg() {
		logger.debug("getGreetMsg() - method start");
		String msg = "Good Morning..!!";
		logger.debug("getGreetMsg() - method end");
		return msg;
	}

}
```

-> Call above classes methods in Spring Boot start class like below.
Note: Spring Boot app execution will start from main ( ) method (instead of we are creating the objects we can use Autowiring concept).

```
@SpringBootApplication
public class LoggingAppApplication {
	public static void main(String[] args) {

		SpringApplication.run(LoggingAppApplication.class, args);
		// calling service class methods
		MessageService service = new MessageService();
		service.getWelcomeMsg();
		service.getGreetMsg();

		// calling dao class method
		ReportDao dao = new ReportDao();
		dao.getName(101);

	}
}
```

-> When we execute the application we can see log messages on the console because the default appender is ConsoleAppender.

-> We can set Log LEVEL and Log File name in application.properties file like below

```
logging.level.root = DEBUG
logging.file.name= MyApp.log
```

-> Right Click on Project and Run as Spring Boot App.

Note: Refresh the project then we can see MyApp.log file with log messages.

==========

Logging with Rolling

==========

-> If we use single log file to store application log messages then it will become very difficult to monitor the logs because every day lot of customers will access application and lot of logs will be generated then Log file size will increase like anything.

Note: If we see application log file after one month, TBs of data will be available then Monitoring will become very difficult.

-> To overcome the above problem we will use Rolling Concept with Logging.
-> We can implement Rolling concept in 2 ways

1) Time Based Rolling
2) Size Based Rolling

-> When we configure Time Based rolling, everyday new log file will be created with timestamp.

-> When we configure Size Based rolling, once log file reached given size then new log file will be created.

Note: If we use only FileAppeder always it will write logs to single log file. We need to use RollingFileAppender to generate logs in multiple log files based on rolling policy

-> create logback.xml file with below content in project resources folder (src / main / resources)

```
<configuration>
   <appender name="RollingFile" class="ch.qos.logback.core.rolling.RollingFileAppender">
                <file>MyApp.log</file>
                <encoder>
        <pattern>%d [%thread] %-5level %-50logger{40} - %msg%n</pattern>
                </encoder>

<rollingPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">
```

```
<fileNamePattern>MyApp-%d{yyyy-MM-dd}.%i.log</fileNamePattern>
                        <maxFileSize>1MB</maxFileSize>
                        <maxHistory>30</maxHistory>
                        <totalSizeCap>10MB</totalSizeCap>
                        <cleanHistoryOnStart>true</cleanHistoryOnStart>
                </rollingPolicy>
        </appender>
        <root level="INFO">
                <appender-ref ref="RollingFile" />
        </root>
</configuration>
```

==========

Log Monitoring

==========

-> Log Monitoring means connecting to log server and getting logs of application
-> We have several tools to monitor logs of the application

1) Putty
2) WinScp
3) Splunk

Note: We need to have log server details

********** redis cache **********

1) What is Cache ?
2) Why we need cache ?
3) Where to use Cache in our project ?

=> Cache is a temporary memory which is used to store data in key-value pair.
=> Cache is used to reduce no.of DB calls from the application.
=> We will use Cache Memory to store static data to use in application.
=> Redis Cache we can use as global cache.
=> Global Cache means multiple application can connect to that Cache to perform operations.

Note: In CO module we have 180+ apis to generate notices. All the apis needs DHS office details to display in notice footer.

=> To improve performance of our apis we can use Cache memory here.

==========

Working with Redis Cache

==========

1) Download and Install Redis cache s/w in our machine
2) Store the data into Redis Cache in key-value format
   Ex: DHS_ADDRESS=H.No#streetname#city#phno#email#website
3) Add Spring-boot-redis-starter dependency in the project

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

4) Configure JedisConnectionFactory bean with Redis Server details (URL, uname, pwd, port etc)

5) Create RedisTemplate object and inject JedisConnectionFactory object into RedisTemplate

```
@Configuration
public class RedisConfig {

	@Bean
	public JedisConnectionFactory jendisConnectionFactory() {
		JedisConnectionFactory jcf = new JedisConnectionFactory();
		// set redis server properties
		return jcf;
	}

	@Bean
	public RedisTemplate<String, String> redisTemplate(){

      RedisTemplate<String, String> redisTemplate = new RedisTemplate<>();
		redisTemplate.setConnectionFactory(jendisConnectionFactory());
		return redisTemplate;
	}
}
```

6) Inject RedisTemplate into our service impl class and get HashOperations object from RedisTemplate to perform operations with Redis Server.

```
HashOperations<String, Object, Object> hashOps = redisTemplate.opsForHash();

String addr = (String) hashOps.get("DHS", "DHS_OFC_ADDRESS");
                        // process the address text and set to pdf as footer text
```

********** Spring Security **********

==========
How to secure REST APIs using Spring Boot
==========

-> Security is very important for every web application
-> To protect our application & application data we need to implement security logic
-> Spring Security concept we can use to secure our web applications / REST APIs
-> To secure our spring boot application we need to add below starter in pom.xml file

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Note: When we add this dependency in pom.xml file then by default our application will be secured with basic authentication. It will generate random password to access our application.

Note: Generated Random Password will be printed on console.

-> We need to use below credentials to access our application

Username : user
Password : <copy the pwd from console>

-> When we access our application url in browser then it will display "Login Form" to authenticate our request.

-> To access secured REST API from postman, we need to set Auth values in POSTMAN to send the request

==========

How to override Spring Security Random Password

==========

-> To override random credentials we can configre security credentials in application.properties file or application.yml file like

```
spring.security.user.name=ashokit
spring.security.user.password=ashokit@123
```

-> After configuring credentials like above, we need to give above credentials to access our application / api.

==========

How to secure specific URL Patterns

==========

-> When we add 'security-starter' in pom.xml then it will apply security filter for all the HTTP methods of our application.

-> But in reality we need to secure only few methods not all methods

For Example

/ login-page --> security not required
/ transfer ---> security required
/ balance ---> security required
/about-us ---> security not required

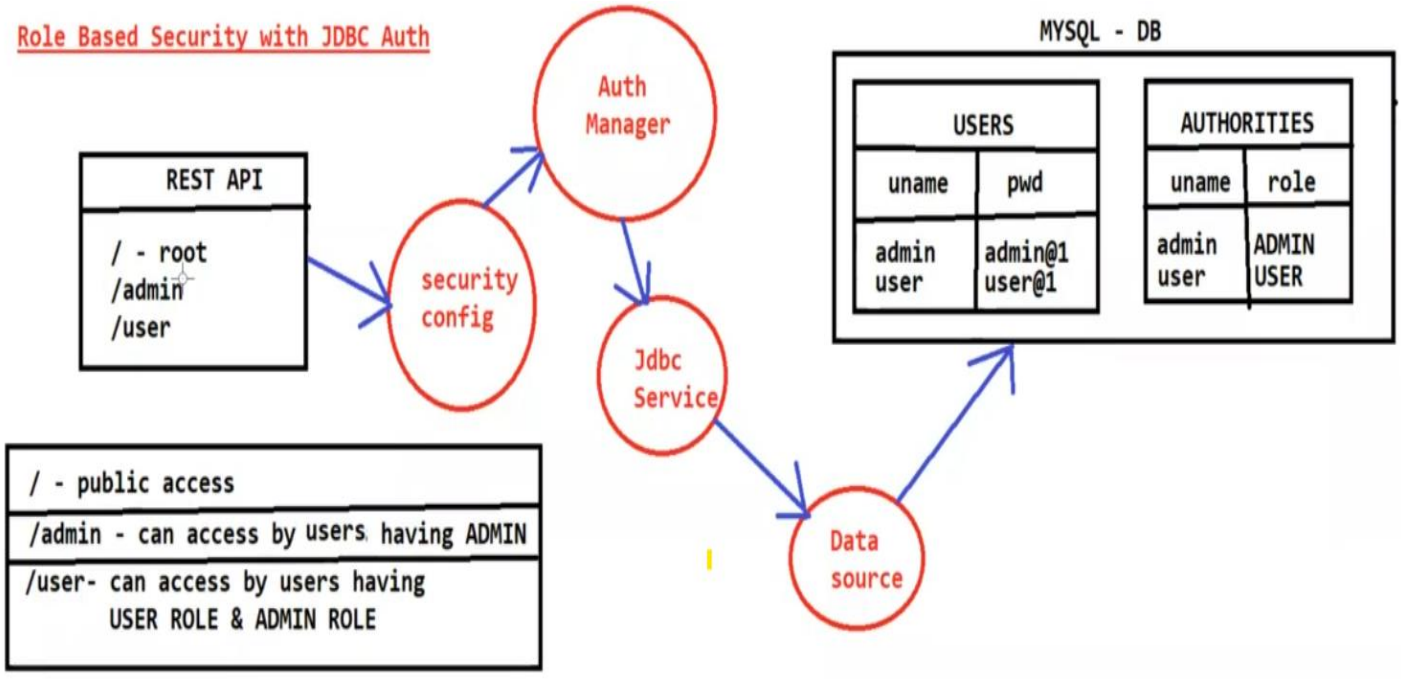
-> In order to achieve above requirement we need to Customize Security Configuration in our project like below

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

	@Bean
	public SecurityFilterChain securityFilter(HttpSecurity http) throws Exception{

		http.authorizeHttpRequests((request) -> request
			.antMatchers("/","/login","/about", "/swagger-ui.html")
			.permitAll()
			.anyRequest()
			.authenticated()
		).formLogin();

		return http.build();
	}
}
```

==========

Spring Boot Security with JDBC Authentication

==========



Step-1) Setup Database tables with required data

-- users table structure

```
CREATE TABLE `users` (
  `username` VARCHAR(50) NOT NULL,
  `password` VARCHAR(120) NOT NULL,
  `enabled` TINYINT(1) NOT NULL,
  PRIMARY KEY (`username`)
);
```

-- authorities table structure

```
CREATE TABLE `authorities` (
  `username` VARCHAR(50) NOT NULL,
  `authority` VARCHAR(50) NOT NULL,
  KEY `username` (`username`),
  CONSTRAINT `authorities_ibfk_1` FOREIGN KEY (`username`)
  REFERENCES `users` (`username`)
);
```

====== Online Encrypt : https://bcrypt-generator.com/ ==========

-- insert records into table

```
insert into users values ('admin',
'$2a$12$0l.1TapVc7dR9mRwoCuWCO3GP4ekxrmfvtYVxx8VhXRbOznIrwNfu',  1);
insert into users values ('user',
'$2a$12$mZlgVUBTMMfZKQNhxvq4PO1u7syQ40RkVUzCDSSbwJmEr09KcJybW',  1);

insert into authorities values ('admin', 'ROLE_ADMIN');
insert into authorities values ('admin', 'ROLE_USER');
insert into authorities values ('user', 'ROLE_USER');
```

Step-2) Create Boot application with below dependencies

a) web-starter
b) security-starter

c) data-jdbc
d) mysql-connector
e) lombok
f) devtools

Step-3) Configure Data source properties in application.properties file

```
#MySQL database connection strings
spring.datasource.url=jdbc:mysql://localhost:3306/jrtp
spring.datasource.username=root
spring.datasource.password=root
```

Step-4) Create Rest Controller with Required methods

```
@RestController
public class UserRestController {
	@GetMapping(value = "/admin")
	public String admin() {
		return "<h3>Welcome Admin :)</h3>";
	}

	@GetMapping(value = "/user")
	public String user() {
		return "<h3>Hello User :)</h3>";
	}

	@GetMapping(value = "/")
	public String welcome() {
		return "<h3>Welcome :)</h3>";
	}

}
```

Step-5) Create Security Configuration class like below with Jdbc Authentication Manager

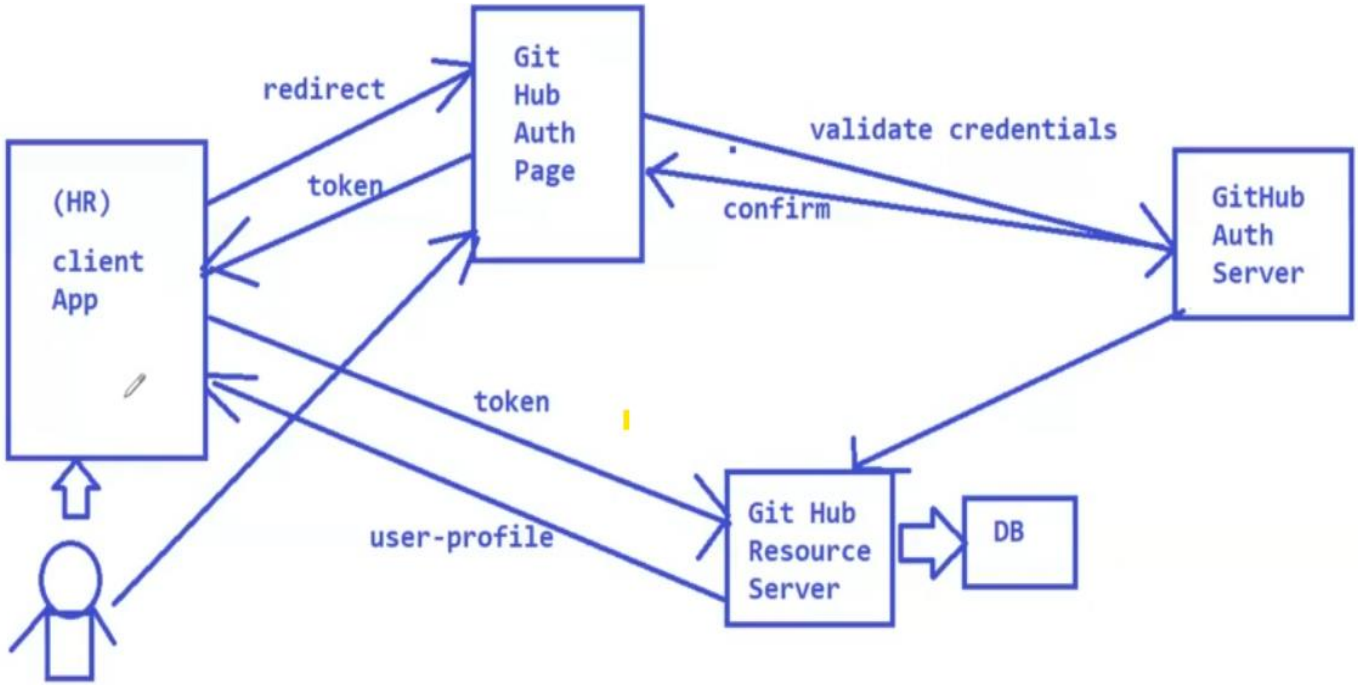
```
package in.ashokit;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
public class SecurityConfiguration {
	private static final String ADMIN = "ADMIN";
	private static final String USER = "USER";

	@Autowired
	private DataSource dataSource;


	@Autowired
	public void authManager(AuthenticationManagerBuilder auth) throws Exception {

	   auth.jdbcAuthentication()
		.dataSource(dataSource)
		.passwordEncoder(new BCryptPasswordEncoder())
		.usersByUsernameQuery("select username,password,enabled from users
		  where username=?")
		.authoritiesByUsernameQuery("select username,authority from authorities
		  where username=?");
	}

	@Bean
	public SecurityFilterChain securityConfig(HttpSecurity http) throws Exception {
```

```
                http.authorizeHttpRequests( (req) -> req
                                .antMatchers("/admin").hasRole(ADMIN)
                                .antMatchers("/user").hasAnyRole(ADMIN,USER)
                                .antMatchers("/").permitAll()
                                .anyRequest().authenticated()
                ).formLogin();

                return http.build();
        }

}
```

==========

OAuth 2.0

==========



1) Create Spring Boot application with below dependencies

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
```

```
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

2) Create OAuth app in Github.com

(Login --> Setting --> Developer Settings --> OAuth Apps --> Create App --> Copy Client ID & Secret)

3) Configure GitHub OAuth App client id & client secret in application.yml file like below

```
spring:
  security:
    oauth2:
      client:
        registration:
          github:
            clientId: <id>
            clientSecret: <secret>
```

4) Create Rest Controller with method

```
@RestController
public class WelcomeRestController {

	@GetMapping("/")
	public String welcome() {
		return "Welcome to Ashok IT";
	}
}
```

5) Run the application and test it.

********** Spring Boot security with JWT authentication **********

-> What is Security ?
-> What is Authentication ?
-> What is Authorization ?
-> Spring Security with In-Memory Authentication
-> Spring Security with JDBC Authentication

-> By using above In-Memory and JDBC Authentication we can secure web applications.

How to secure Distributed application

-> When one application is using services of other applications then implementation of security with webservices concepts becomes more important.

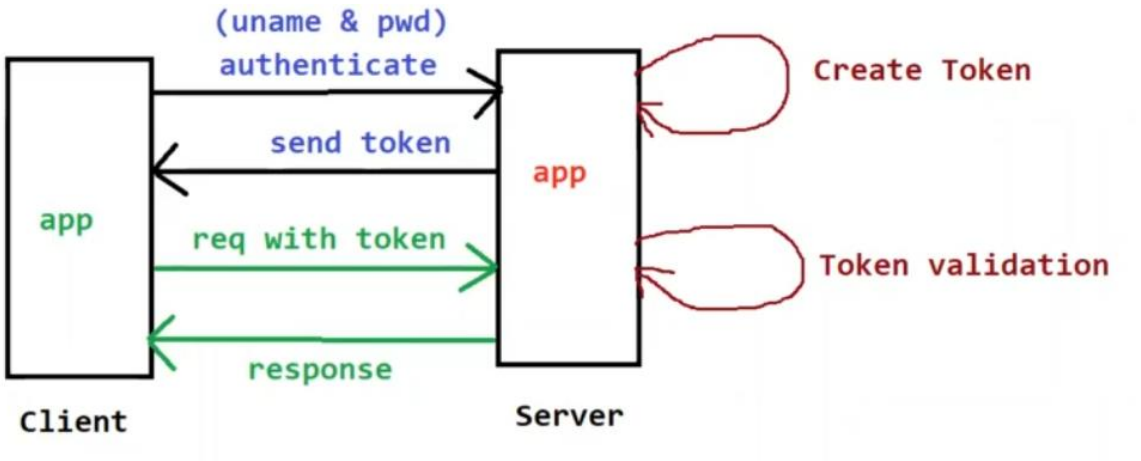
-> In this scenario we will secure our application by using a token.

What is JWT Authentication

-> JWT is standard mechanism to implement Token based security

-> JWT stands Json Web Tokens

-> JWT is not only for java, we can use this technique in other languages also to secure our applications.



-> Token is a data which will be in the encoded format
-> We will use secret key to generate token
-> JWT token will have 3 parts

i) header
ii) payload
iii) signature

Note: JWT Token each part will be seperated by dot (.)

Header : It contains JWT specific information

Payload : IT contains claims (Client ID, Client Name, Issuer Name, Audience Name, Date of Issue, Expirty Date etc..)

Signature : Base64 encoded form of header & payload, aditionally signed with secret key.

Sample Token
---
eyJhbGciOiJIUzUxMiJ9.eyJqdGkiOiJ0azk5MzEiLCJzdWIiOiJteXRva2VuIiwiaXNzIjoiQXNob2sgSVQiLCJhdWQiOiJBQkNfSVQiLCJpYXQiOjE2MzA5ODQ2NTQsImV4cCI6MTYzMDk4ODI1NH0.8WN1DMPJ7dingc4pAFmPDQyk2SnfAJ-OutGHQ5gcy0qd1h1lc3rrTApc7tvI0l-aCYRB5CcxWbBHRUDlC9i8Zg

---

Steps to generate token using JWT

---

1) Create Spring Boot Application with below dependencies

==========

a) web-starter
b) security-starter
c) lombok
d) devtools

```
	<dependencies>
		<dependency>
			<groupId>io.jsonwebtoken</groupId>
			<artifactId>jjwt</artifactId>
			<version>0.9.1</version>
		</dependency>
		<dependency>
			<groupId>javax.xml.bind</groupId>
			<artifactId>jaxb-api</artifactId>
			<version>2.3.0</version>
		</dependency>
	</dependencies>
```

==========

2) Create One Rest Controller like below

==========

```
@RestController
public class WelcomeRestController {

	@GetMapping("/welcome")
	public String welcomeMsg() {
		return "Welcome to Ashok IT";
	}
}
```

==========

3) Create WebSecurityConfigurer class like below

==========

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfigurer extends WebSecurityConfigurerAdapter {

	@Autowired
	private MyUserDetailsService userDetailsService;

	@Override
	protected void configure(AuthenticationManagerBuilder auth) throws Exception {
		auth.userDetailsService(userDetailsService);
	}
```

```
	@Bean
	public PasswordEncoder passwordEncoder() {
		return NoOpPasswordEncoder.getInstance();
	}
}
```

==========

4) Create MyUserDetailsService to load user data

==========

```
@Service
public class MyUserDetailsService implements UserDetailsService {

	@Override
	public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
		return new User("admin", "admin@123", Collections.emptyList());
	}
}
```

==========

5) Run the application and test it

==========

** With above steps our REST API is ready with Spring Security, Now lets add JWT .

==========

6) Add below two dependencies in pom.xml file

==========

```
		<dependency>
			<groupId>io.jsonwebtoken</groupId>
			<artifactId>jjwt</artifactId>
			<version>0.9.1</version>
		</dependency>
		<dependency>
			<groupId>javax.xml.bind</groupId>
			<artifactId>jaxb-api</artifactId>
		</dependency>
```

==========

7) Create JWT Util class like below to generate, validate the token

==========

```
package in.ashokit.security;

import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;
import org.springframework.security.core.userdetails.UserDetails;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;

public class JwtUtil {

	private String SECRET_KEY = "secret";

	  public String extractUsername(String token) {
	     return extractClaim(token, Claims::getSubject);
	  }

	  public Date extractExpiration(String token) {
	     return extractClaim(token, Claims::getExpiration);
	  }

	  public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
	     final Claims claims = extractAllClaims(token);
	     return claimsResolver.apply(claims);
	  }
	  private Claims extractAllClaims(String token) {
	     return Jwts.parser().setSigningKey(SECRET_KEY).parseClaimsJws(token).getBody();
	  }

	  private Boolean isTokenExpired(String token) {
	     return extractExpiration(token).before(new Date());
	  }

	  public String generateToken(UserDetails userDetails) {
	     Map<String, Object> claims = new HashMap<>();
	     return createToken(claims, userDetails.getUsername());
	  }

	  private String createToken(Map<String, Object> claims, String subject) {

	     return Jwts.builder()
	                   .setClaims(claims)
	                   .setSubject(subject)
```

```
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 60 * 10))
                .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
                .compact();
    }
    public Boolean validateToken(String token, UserDetails userDetails) {
       final String username = extractUsername(token);
       return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
    }

}
```

==========

8) Create AuthenticateRequest class to capture username & password for authentication purpose

==========

```
@Data
public class AuthenticateRequest {

    private String username;
    private String password;

}
```

==========

9) Create AuthenticationResource like below

==========

```
package in.ashokit;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import in.ashokit.security.AuthenticateRequest;
import in.ashokit.security.JwtUtil;
import in.ashokit.security.MyUserDetailsService;

@RestController
public class AuthenticationResource {

    @Autowired
    private AuthenticationManager authManager;
```

```
	@Autowired
	private MyUserDetailsService userDetailsService;

	@Autowired
	private JwtUtil jwtUtil;

	@PostMapping("/authenticate")
	public String authenticate(@RequestBody AuthenticateRequest request) throws Exception {

		try {
			authManager.authenticate(
					new
UsernamePasswordAuthenticationToken(request.getUsername(), request.getPassword()));

		} catch (Exception e) {
			throw new Exception("Invalid Username & Password");
		}

		UserDetails user = userDetailsService.loadUserByUsername(request.getUsername());

		String token = jwtUtil.generateToken(user);

		return token;

	}
}
```

==========

10) Configure AuthManager bean in SecurityConfigurer class along with permissions

==========

```
	@Override
	@Bean
	public AuthenticationManager authenticationManagerBean() throws Exception {
		return super.authenticationManagerBean();
	}

	@Override
	protected void configure(HttpSecurity http) throws Exception {
		http.csrf().disable()
					.authorizeRequests()
					.antMatchers("/authenticate")
					.permitAll()
					.anyRequest()
					.authenticated();
	}
}
```

==========

11) Create Request Filter to intercept each request

==========

```
package in.ashokit.security;

import java.io.IOException;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

@Component
public class JwtRequestFilter extends OncePerRequestFilter {

	@Autowired
	private MyUserDetailsService userDetailsService;

	@Autowired
	private JwtUtil jwtUtil;

	@Override
	protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
FilterChain filterChain)
				throws ServletException, IOException {

		String authorizationHeader = request.getHeader("Authorization");

		String username = null;
		String jwt = null;

		if (authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {
			jwt = authorizationHeader.substring(7);
			username = jwtUtil.extractUsername(jwt);
		}

		if (username != null && SecurityContextHolder.getContext().getAuthentication() ==
null) {
		UserDetails userDetails = this.userDetailsService.loadUserByUsername(username);

			if (jwtUtil.validateToken(jwt, userDetails)) {
```

```
UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken =
  new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());

usernamePasswordAuthenticationToken
.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);
                        }
                }
                filterChain.doFilter(request, response);
        }

}
```

==========

12) Add RequestFilter to security context in WebsecurityConfigurer class

==========

```
        @Override
        protected void configure(HttpSecurity http) throws Exception {
                http.csrf().disable()
                                        .authorizeRequests()
                                        .antMatchers("/authenticate")
                                        .permitAll()
                                        .anyRequest()
                                        .authenticated()
                                        .and()

        .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

        http.addFilterBefore(jwtReqFilter, UsernamePasswordAuthenticationFilter.class);
        }
```

==========

13) Run the application

14) Send request to /authenticate with username and password and in request body. If credentials are valid then it will give token

15) Send request to actual resource i.e /welcome using token. If token is valid it will give response

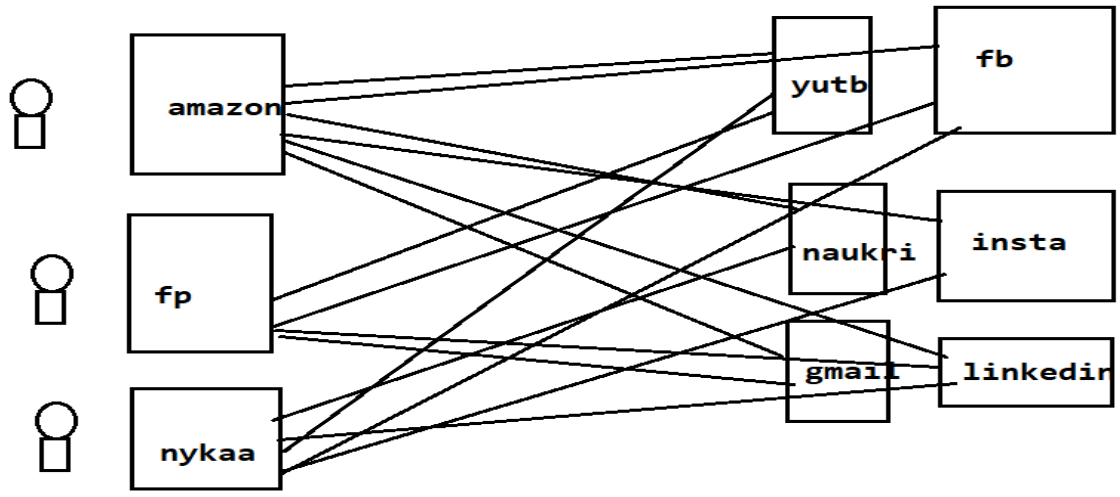
==========
==========

********** Kafka **********

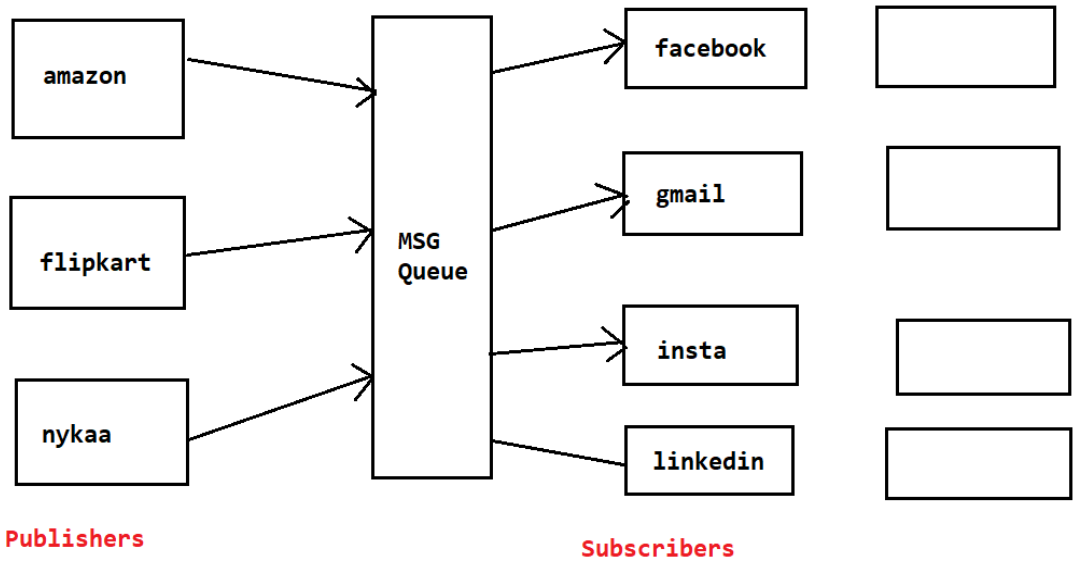
- Apache Kafka is a distributed streaming platform.
- Apache Kafka is used to process real time data feeds with high throughput and low latency.

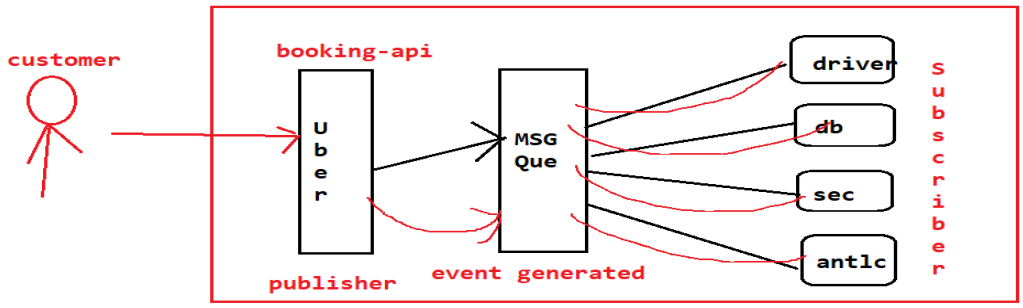
Ex : flights data, sensors data, stocks data, news data etc....
Kafka works based on Publisher and Subscriber model.

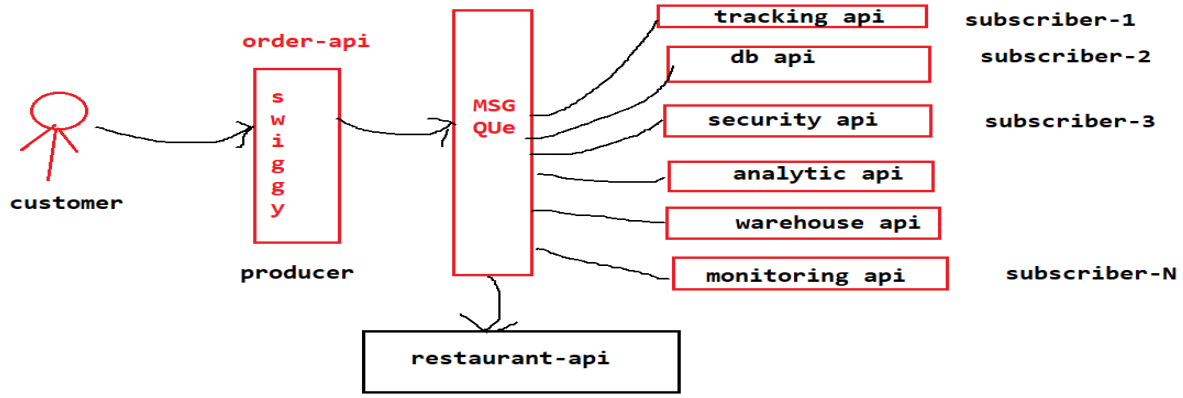
Without Kafka communication:-

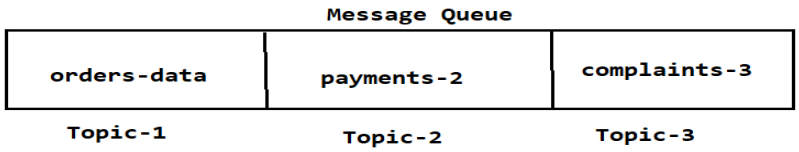


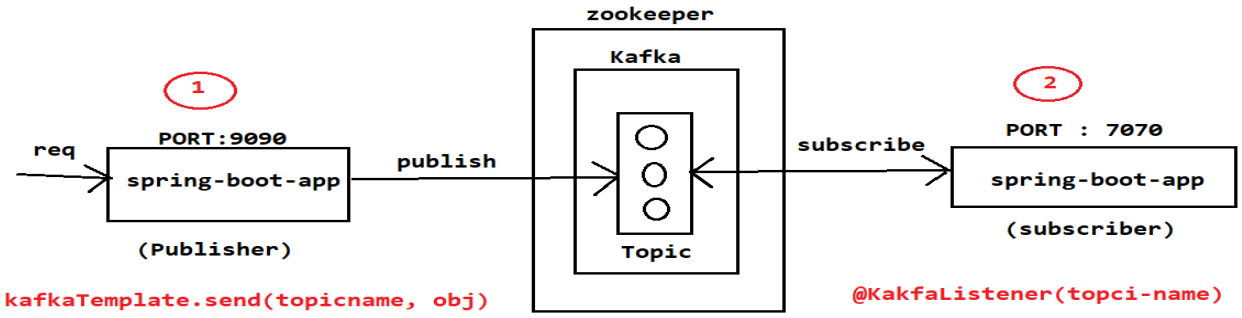
With Kafka communication Client-driven Architecture:-

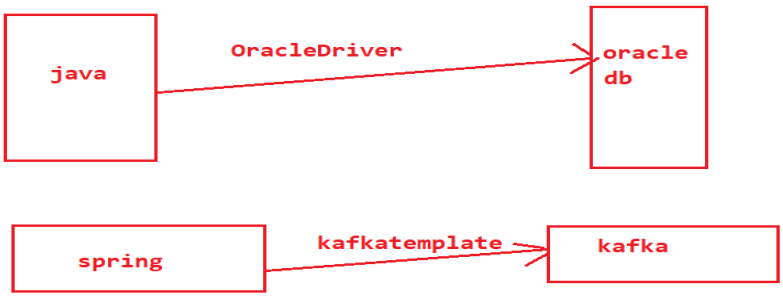



customer
booking-api
Uber
MSG Que
driver
db
sec
antlc
subscriber
publisher
event generated


order-api
swiggy
customer
producer
MSG QUe
tracking api
db api
security api
analytic api
warehouse api
monitoring api
subscriber-1
subscriber-2
subscriber-3
subscriber-N
restaurant-api


Message Queue
orders-data
payments-2
complaints-3
Topic-1
Topic-2
Topic-3


zookeeper
Kafka
Topic
1
PORT:9090
req
spring-boot-app
(Publisher)
publish
subscribe
2
PORT : 7070
spring-boot-app
(subscriber)
kafkaTemplate.send(topicname, obj)
@KakfaListener(topci-name)


java
OracleDriver
oracle db
spring
kafkatemplate
kafka

Kafka Terminology

Zookeeper
Kafka Server
Kafka Topic
Message
Publisher
Subscriber

Kafka APIs

Connector API
Publisher API
Subscriber API
Streams API

Spring Boot + Apache Kafka Application

==========

Step-1 : Download Zookeeper from below URL

URL : http://mirrors.estointernet.in/apache/zookeeper/stable/

Step-2 : Download Apache Kafka from below URL

URL : http://mirrors.estointernet.in/apache/kafka/

Step-3 : Set Path to ZOOKEEPER in Environment variables upto bin folder

Step-4 : Start Zookeeper server using below command from Kafka folder

Command : zookeeper-server-start.bat zookeeper.properties

Note: Above command will available in kafka/bin/windows folder

Note: zookeeper.properties file will be available in kafka/config folder. You can copy zookeeper.properties and server.properties files from kafka/config folder to kafka/bin/windows folder.

Step-5: Start Kafka Server using below command from Kakfa folder

Command : kafka-server-start.bat server.properties

Note: server.properties file will be available in config folder (Copied to windows folder)

Step-6 : Create Kakfa Topic using below command from kafka/bin/windows folder

Command : kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic demo-sbms-topic

Step-7 : View created Topics using below command

Command : kafka-topics.bat --list --zookeeper localhost:2181

Step-8 : Create Spring Boot Project in IDE

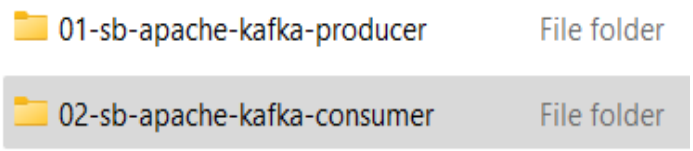
==========

Step-9: Add below kafka related dependencies in pom.xml

```
<dependency>
	<groupId>org.apache.kafka</groupId>
	<artifactId>kafka-streams</artifactId>
</dependency>
<dependency>
	<groupId>org.springframework.kafka</groupId>
	<artifactId>spring-kafka</artifactId>
</dependency>
<dependency>
	<groupId>com.fasterxml.jackson.core</groupId>
	<artifactId>jackson-databind</artifactId>
</dependency>
```

Step-9: Create RestController, KafaProducer and KafaConsumer classes to publish and subsribe message

Step-10: Test application using PostMan.



**********publisher spring boot project **********

Kafka Producer Configuration:-

```
package com.ashok.config;

import java.util.HashMap;
import java.util.Map;

import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.serialization.StringSerializer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.core.DefaultKafkaProducerFactory;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.kafka.core.ProducerFactory;
import org.springframework.kafka.support.serializer.JsonSerializer;

import com.ashok.model.Customer;
import com.ashok.util.KafkaConstants;

@Configuration
public class KafkaProduceConfig {

	/**
	 * This method is used to Kafka Producer Config details
	 */

	@Bean
	public ProducerFactory<String, Customer> producerFactory() {
		Map<String, Object> configProps = new HashMap<String, Object>();
configProps.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, KafkaConstants.HOST);
configProps.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
configProps.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, JsonSerializer.class);
		return new DefaultKafkaProducerFactory(configProps);
	}

	/**
	 * This method is used to create KafkaTemplate bean obj
	 */
	@Bean(name = "kafkaTemplate")
	public KafkaTemplate<String, Customer> kafkaTemplate() {
		return new KafkaTemplate<>(producerFactory());
	}

}
```

Controller class:-

```
package com.ashok.controller;

import java.util.ArrayList;
import java.util.List;

import javax.ws.rs.core.MediaType;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import com.ashok.model.Customer;
import com.ashok.service.CustomerService;

@RestController
public class CustomerRestController {

	@Autowired
	private CustomerService customerService;

	@PostMapping(value = "/addCustomer",
			consumes = {
					MediaType.APPLICATION_JSON,
					MediaType.APPLICATION_XML
			}
	)
	public String addCustomer(@RequestBody List<Customer> customers) {
		return customerService.add(customers);
	}
}
```

Service class:-

```
package com.ashok.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Service;
```

```
import com.ashok.model.Customer;
import com.ashok.util.KafkaConstants;


@Service("customerService")
public class CustomerService {

	@Autowired
	private KafkaTemplate<String, Customer> kafkaTemplate;

	public String add(List<Customer> customers) {

		if (!customers.isEmpty()) {
			for (Customer c : customers) {
				kafkaTemplate.send(KafkaConstants.TOPIC, c);
	System.out.println("**********Msg published to Kafka topic**********");
			}
		}
		return "Customer Record Added To Kafka Queue Successfully";
	}
}
```

Constant Utils.class:-

```
package com.ashok.util;

/**
 * This class is used to declare constants of this application
*/
public class KafkaConstants {

	public static final String TOPIC = "customer";
	public static final String GROUP_ID = "group_customer";
	public static final String HOST = "localhost:9092";

}
```

Customer Model class:-

```
package com.ashok.model;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlRootElement;
```

```
/**
 * This class serving as model to hold data
*/

@XmlRootElement(name = "customer")
@XmlAccessorType(XmlAccessType.FIELD)
public class Customer {

	private Integer customerId;
	private String customerName;
	private String customerEmail;

	public Customer() {
		// TODO Auto-generated constructor stub
	}

	/**
	 * @param customerId
	 * @param customerName
	 * @param customerEmail
	 */
	public Customer(Integer customerId, String customerName, String customerEmail) {
		super();
		this.customerId = customerId;
		this.customerName = customerName;
		this.customerEmail = customerEmail;
	}

	public Integer getCustomerId() {
		return customerId;
	}

	public void setCustomerId(Integer customerId) {
		this.customerId = customerId;
	}

	public String getCustomerName() {
		return customerName;
	}

	public void setCustomerName(String customerName) {
		this.customerName = customerName;
	}

	public String getCustomerEmail() {
		return customerEmail;
```

```
	}

	public void setCustomerEmail(String customerEmail) {
		this.customerEmail = customerEmail;
	}

	@Override
	public String toString() {
		return "Customer [customerId=" + customerId + ", customerName=" +
			customerName + ", customerEmail="+ customerEmail + "]";
	}

}
```

Application.properties:-

```
	Server.port=9090
```

**********consumer spring boot project **********

Kafka consumer Configuration:-

```
package com.ashok.config;

import java.util.HashMap;
import java.util.Map;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.annotation.EnableKafka;
import org.springframework.kafka.config.ConcurrentKafkaListenerContainerFactory;
import org.springframework.kafka.core.ConsumerFactory;
import org.springframework.kafka.core.DefaultKafkaConsumerFactory;
import org.springframework.kafka.support.serializer.JsonDeserializer;

import com.ashok.model.Customer;
import com.ashok.util.KafkaConstants;


@Configuration
@EnableKafka
public class KafkaListenerConfig {
```

```
	/**
	 * This method is used to Kafka Consumer Config details
	 */
	@Bean
	public ConsumerFactory<String, Customer> consumerFactory() {
	Map<String, Object> props = new HashMap();
	props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, KafkaConstants.HOST);
	props.put(ConsumerConfig.GROUP_ID_CONFIG, KafkaConstants.GROUP_ID);
	props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
	props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, JsonDeserializer.class);

return new DefaultKafkaConsumerFactory<>(props, new StringDeserializer(), new JsonDeserializer<>(Customer.class));
	}


	@Bean
	public ConcurrentKafkaListenerContainerFactory<String, Customer> kafkaListenerContainerFactory()
{
ConcurrentKafkaListenerContainerFactory<String, Customer> factory = new
                                        ConcurrentKafkaListenerContainerFactory<String, Customer>();
		factory.setConsumerFactory(consumerFactory());
		return factory;
	}

}
```

Controller class:-

```
package com.ashok.controller;

import org.springframework.web.bind.annotation.RestController;

@RestController
public class CustomerRestController {

}
```

Service class:-

```
package com.ashok.service;

import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Service;

import com.ashok.model.Customer;
import com.ashok.util.KafkaConstants;
```

```
@Service("customerService")
public class CustomerService {


@KafkaListener(topics = KafkaConstants.TOPIC, groupId = KafkaConstants.GROUP_ID)
public Customer listener(Customer c) {
		System.out.println("***Msg recieved from Kafka Topic ::" + c);
		return c;
	}

}
```

Model class:-

```
package com.ashok.model;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "customer")
@XmlAccessorType(XmlAccessType.FIELD)
public class Customer {

	private Integer customerId;
	private String customerName;
	private String customerEmail;

	public Customer() {
		// TODO Auto-generated constructor stub
	}

	public Customer(Integer customerId, String customerName, String customerEmail) {
		super();
		this.customerId = customerId;
		this.customerName = customerName;
		this.customerEmail = customerEmail;
	}

	public Integer getCustomerId() {
		return customerId;
	}

	public void setCustomerId(Integer customerId) {
		this.customerId = customerId;
	}

	public String getCustomerName() {
```

```
		return customerName;
	}

	public void setCustomerName(String customerName) {
		this.customerName = customerName;
	}

	public String getCustomerEmail() {
		return customerEmail;
	}

	public void setCustomerEmail(String customerEmail) {
		this.customerEmail = customerEmail;
	}

	@Override
	public String toString() {
		return "Customer [customerId=" + customerId + ", customerName=" +
			customerName + ", customerEmail="+ customerEmail + "]";
	}

}
```

Utils.class:-

```
package com.ashok.util;

public class KafkaConstants {

	public static final String TOPIC = "customer";
	public static final String GROUP_ID = "group_customer";
	public static final String HOST = "localhost:9092";

}
```

Application.properties:-

```
Server.port=7070
```

Pom.xml:-

```
<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
```

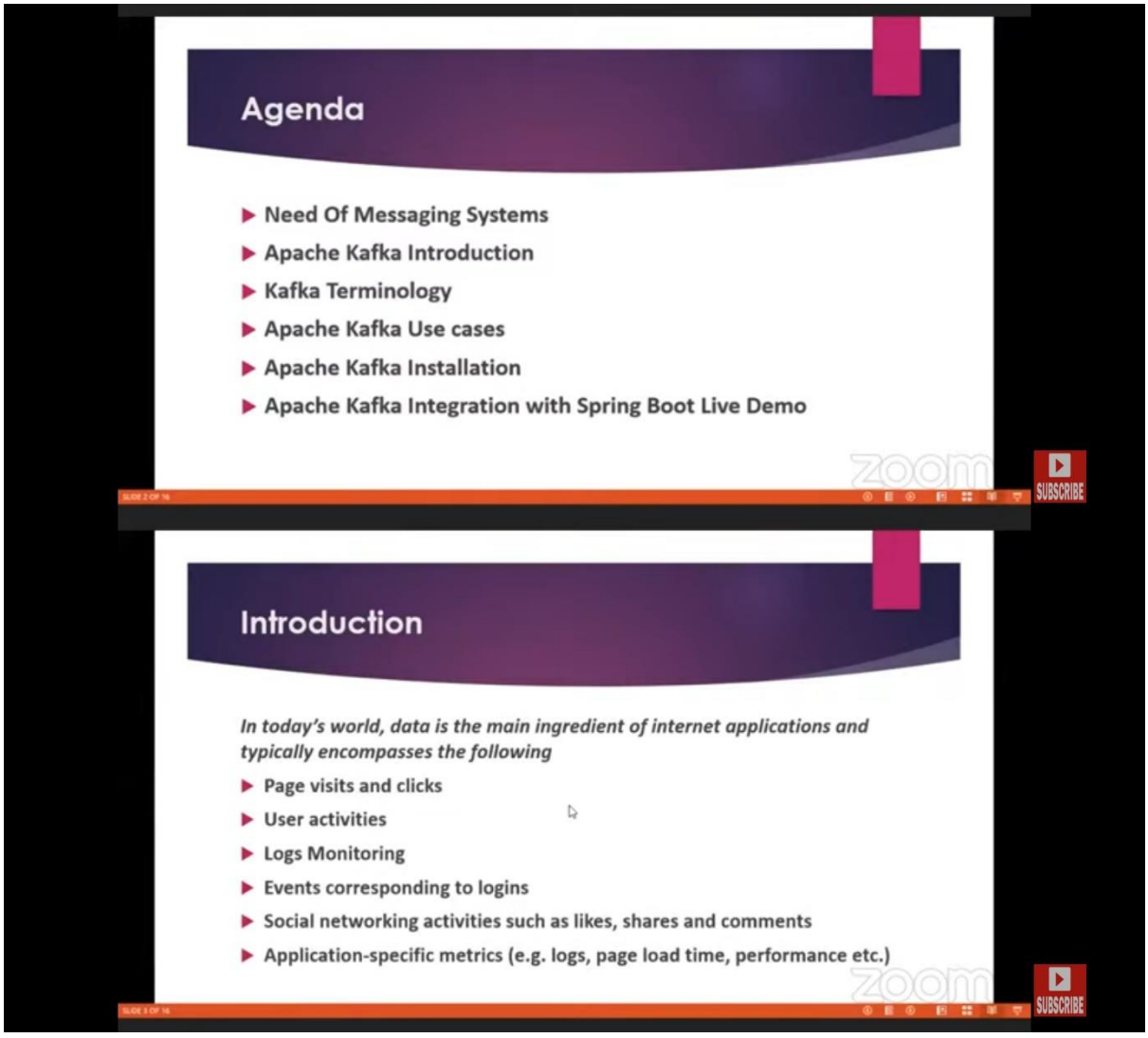
```
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-streams</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.kafka</groupId>
            <artifactId>spring-kafka</artifactId>
        </dependency>

        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.kafka</groupId>
            <artifactId>spring-kafka-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

Sample Data

```
{
"customerId":101,
"customerName":"Ashok",
"customerEmail":"ashok@gmail.com"
}
```

```
[

{
    "customerId":101,
    "customerName":"Ashok",
```

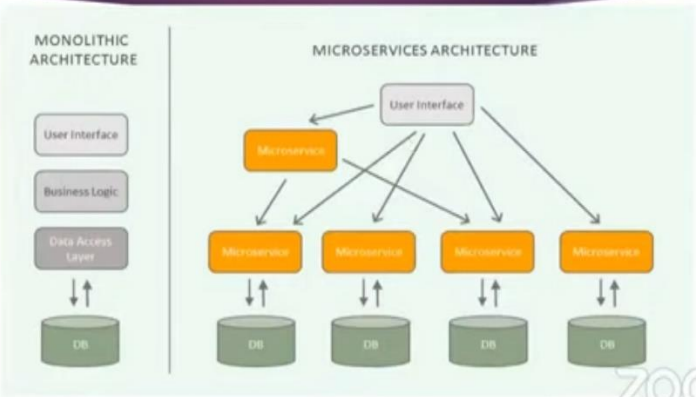
```
        "customerEmail":"ashok@gmail.com"
},

{
        "customerId":102,
        "customerName":"Raj",
        "customerEmail":"raj@gmail.com"
},
{
        "customerId":102,
        "customerName":"John",
        "customerEmail":"john@gmail.com"
}

]
```



What is Event Stream ?

- Event streaming is the practice of capturing data in real-time from event sources
- Sources can be databases, sensors, mobile devices, cloud services, and software applications
- Storing these event streams durably for later retrieval; manipulating, processing, and reacting to the event streams in real-time as well as retrospectively; and routing the event streams to different destination technologies as needed
- Using Event Streams we can keep right information at right place at right time

Microservices Architecture

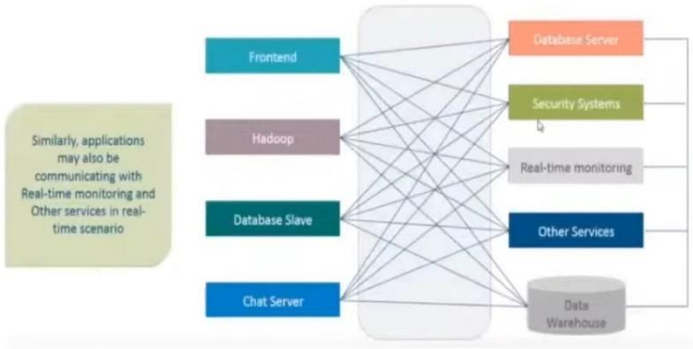


Uses of Data

Applications data can be used to run analytics in real time serving various purposes, some of which are

- Delivering advertisements
- Tracking abnormal user behaviors
- Displaying search based on relevance
- Showing recommendations based on previous activities

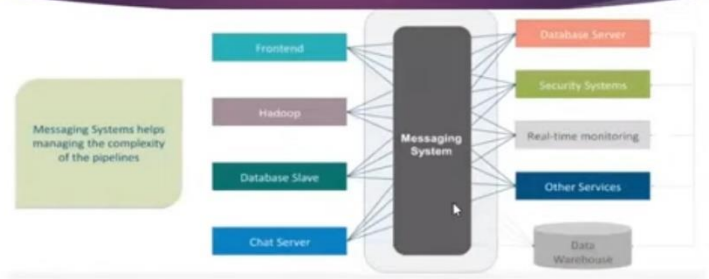
Complex Data Pipelines



Why Messaging Systems ?

- Problem: Collecting all the data is not easy as data is generated from various sources in different formats
- Solution: One of the ways to solve this problem is to use a messaging system. Messaging systems provide a seamless integration between distributed applications with the help of messages.

Messaging Systems



What is Apache Kafka ?

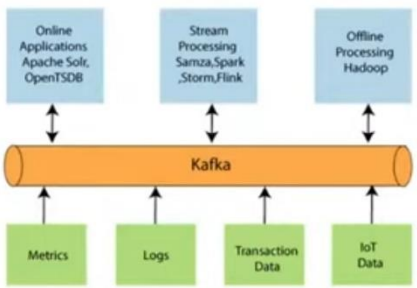
- Apache Kafka is a distributed streaming platform
- Apache Kafka is used to process real time data feeds with high throughput and low latency

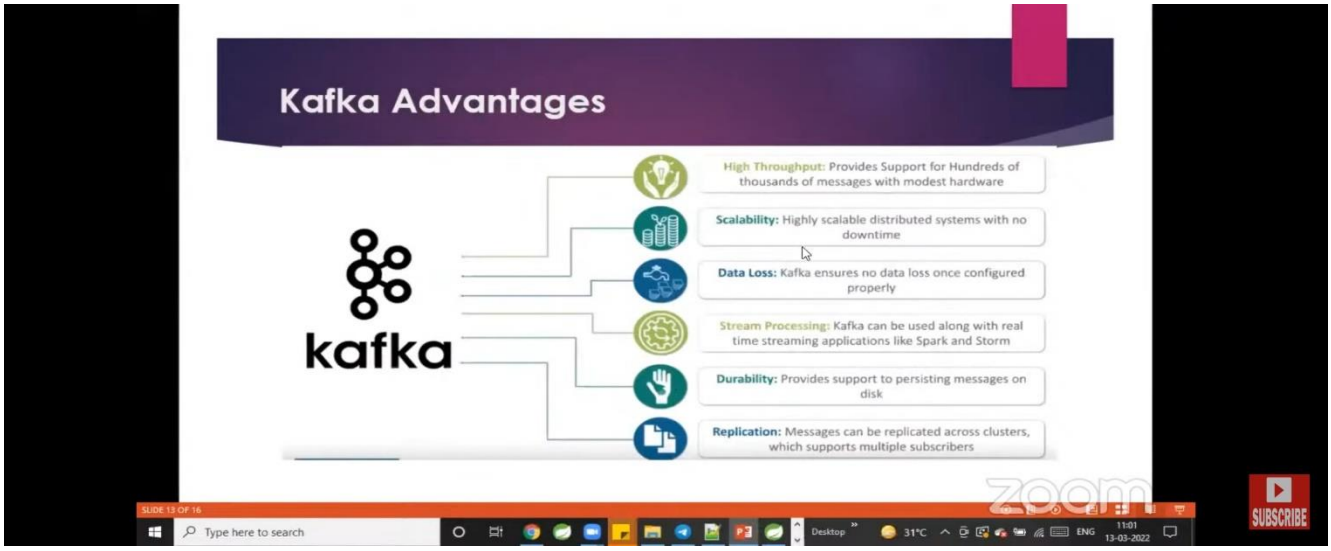
  Ex : flights data, sensors data, stocks data, news data, user activities etc....
- Kafka works based on Publisher and Subscriber model
- Kafka was originally developed at LinkedIn in 2011

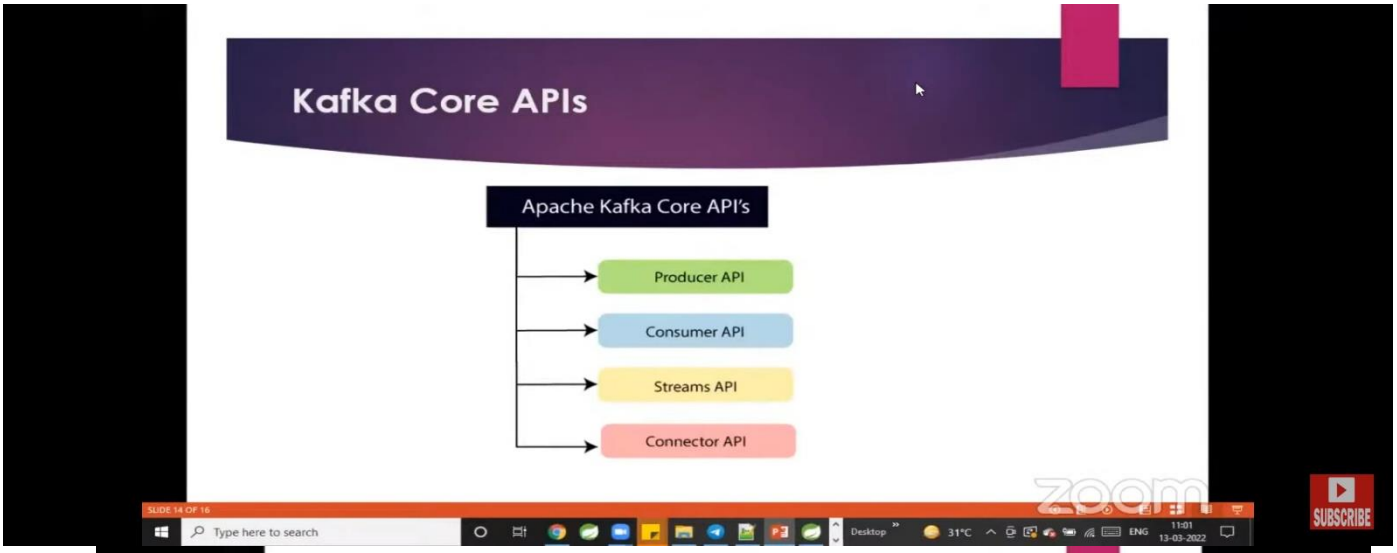
How Kafka Works?

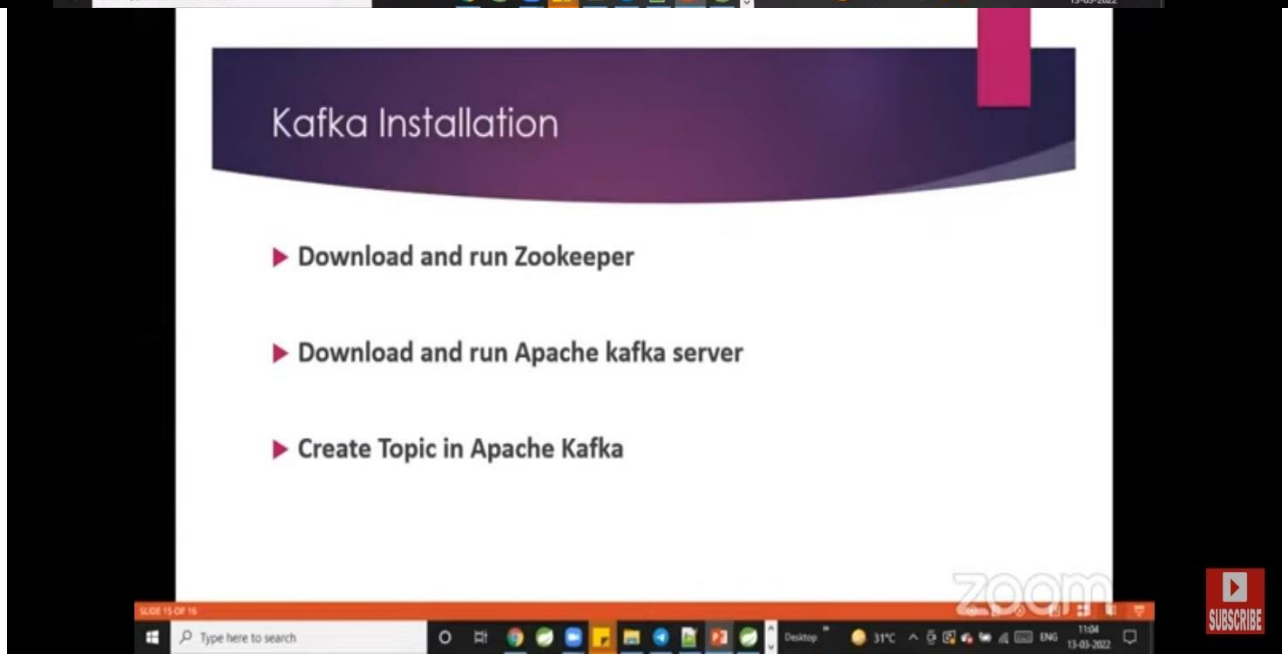
- Apache Kafka is a distributed streaming platform
- At its core, it allows systems that generate data (called Producers) to persist their data in real-time in an Apache Kafka Topic
- Any topic can then be read by any number of systems who need that data in real-time (called Consumers)
- Kafka is a Pub/Sub system. Behind the scenes, Kafka is distributed, scales well, replicates data across brokers (servers), can survive broker downtime, and much more

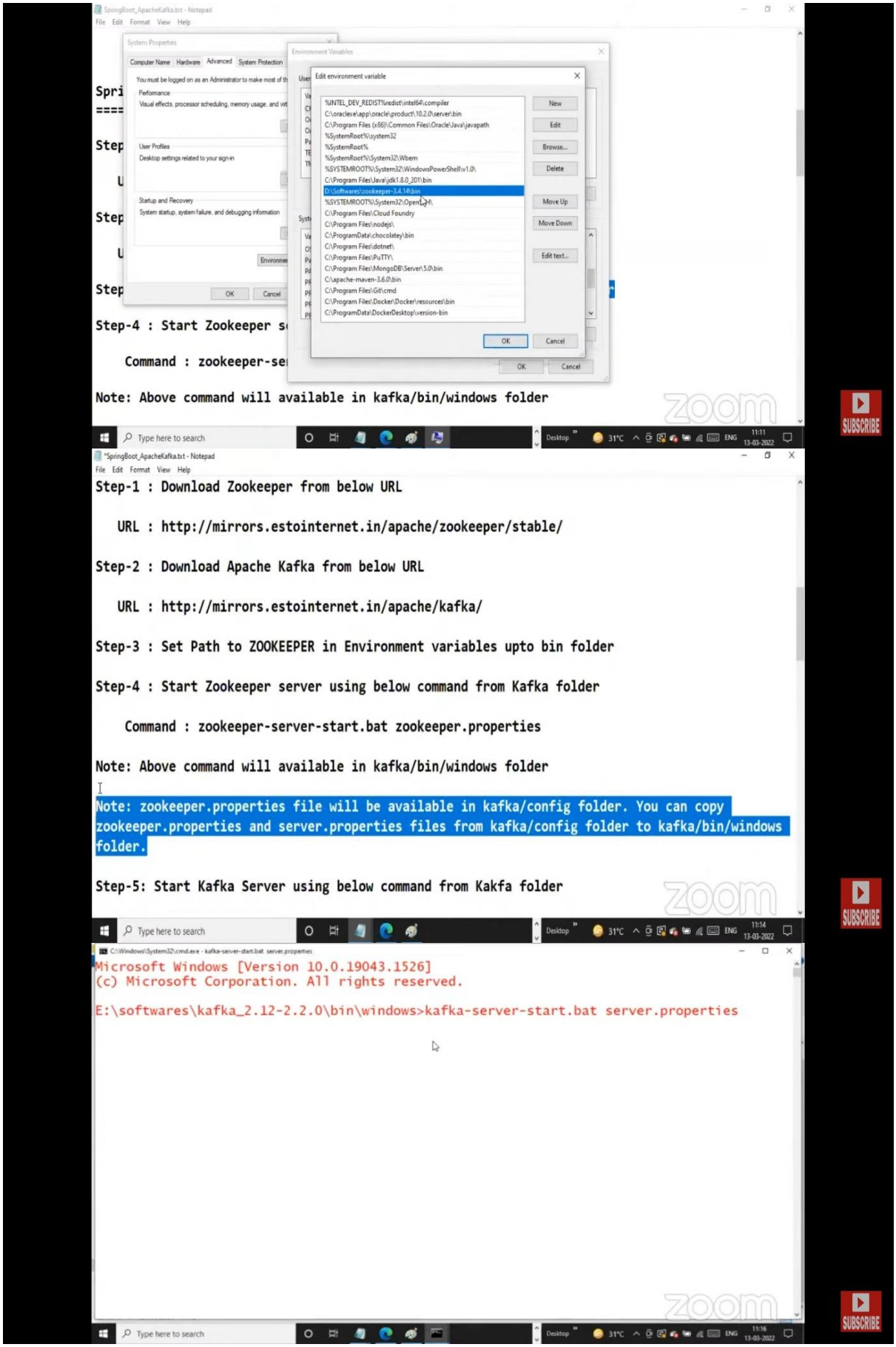
How Kafka Works

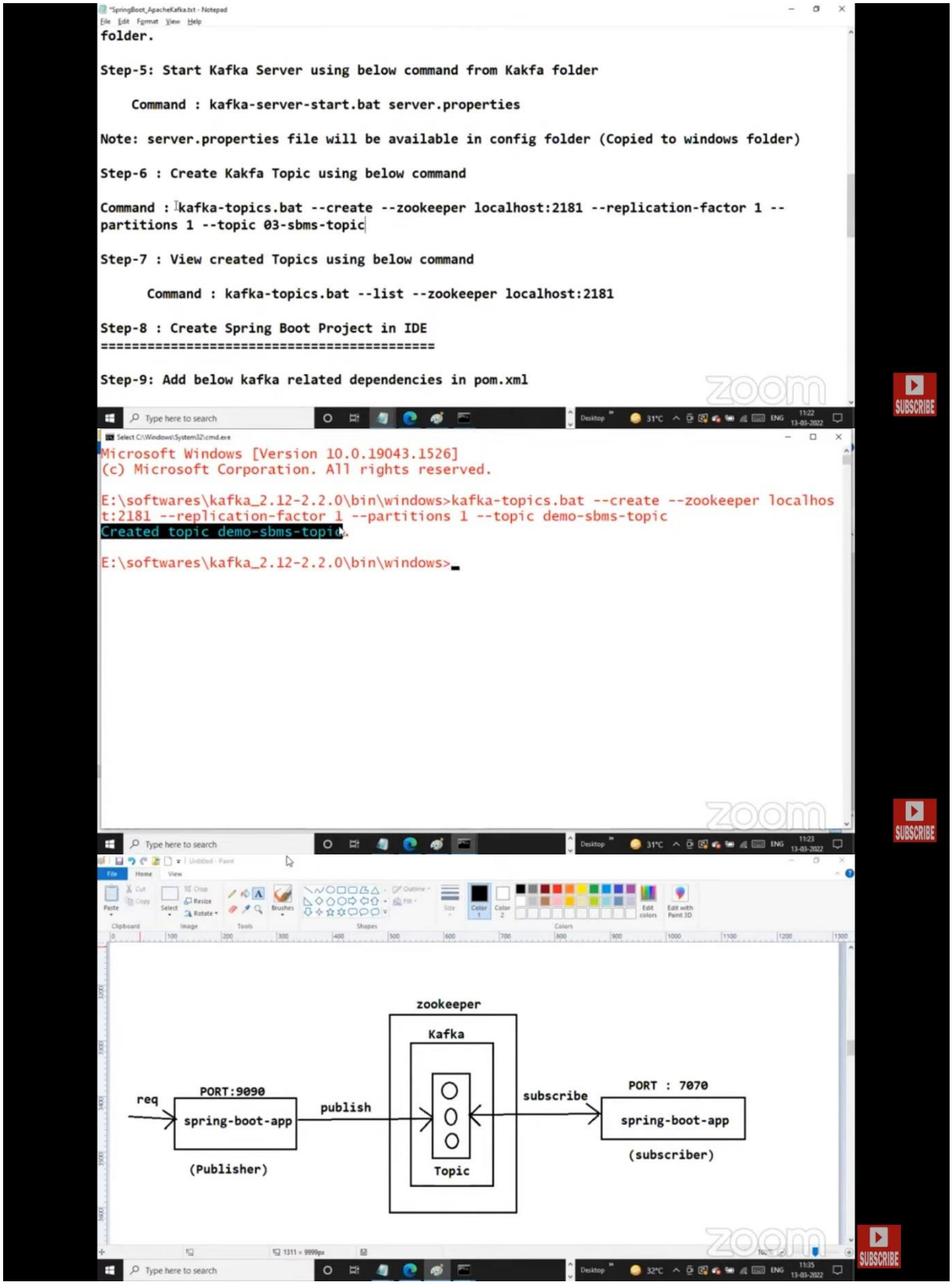


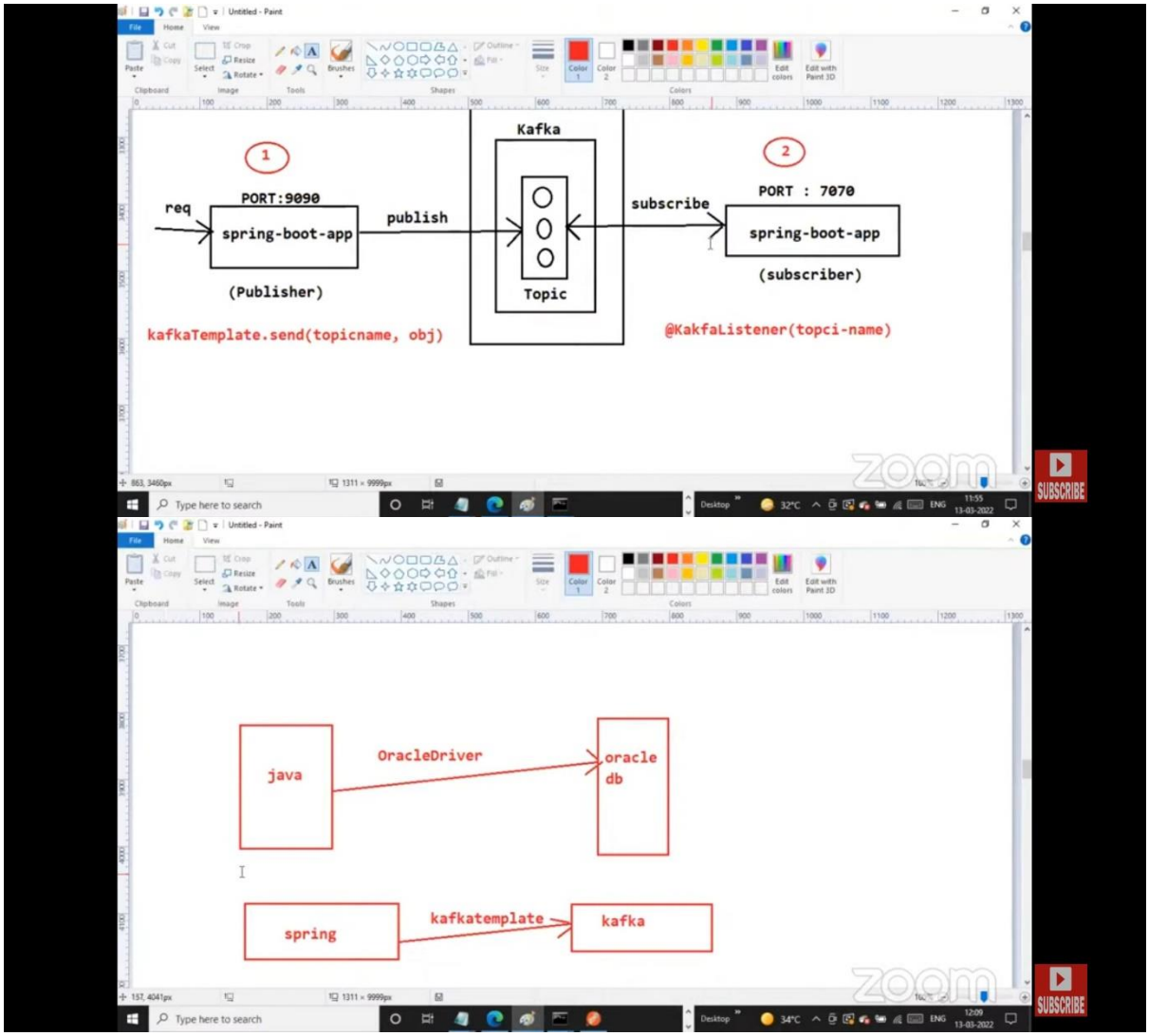

Kafka Advantages
kafka
High Throughput: Provides Support for Hundreds of thousands of messages with modest hardware
Scalability: Highly scalable distributed systems with no downtime
Data Loss: Kafka ensures no data loss once configured properly
Stream Processing: Kafka can be used along with real time streaming applications like Spark and Storm
Durability: Provides support to persisting messages on disk
Replication: Messages can be replicated across clusters, which supports multiple subscribers
SUBSCRIBE


Kafka Core APIs
Apache Kafka Core API's
Producer API
Consumer API
Streams API
Connector API
SUBSCRIBE


Kafka Installation
Download and run Zookeeper
Download and run Apache kafka server
Create Topic in Apache Kafka
SUBSCRIBE


Edit environment variable
Step-4 : Start Zookeeper s
Command : zookeeper-se
Note: Above command will available in kafka/bin/windows folder
Step-1 : Download Zookeeper from below URL
URL : http://mirrors.estointernet.in/apache/zookeeper/stable/
Step-2 : Download Apache Kafka from below URL
URL : http://mirrors.estointernet.in/apache/kafka/
Step-3 : Set Path to ZOOKEEPER in Environment variables upto bin folder
Step-4 : Start Zookeeper server using below command from Kafka folder
Command : zookeeper-server-start.bat zookeeper.properties
Note: Above command will available in kafka/bin/windows folder
Note: zookeeper.properties file will be available in kafka/config folder. You can copy zookeeper.properties and server.properties files from kafka/config folder to kafka/bin/windows folder.
Step-5: Start Kafka Server using below command from Kakfa folder
Microsoft Windows [Version 10.0.19043.1526]
(c) Microsoft Corporation. All rights reserved.
E:\softwares\kafka_2.12-2.2.0\bin\windows>kafka-server-start.bat server.properties


folder.
Step-5: Start Kafka Server using below command from Kakfa folder
Command : kafka-server-start.bat server.properties
Note: server.properties file will be available in config folder (Copied to windows folder)
Step-6 : Create Kakfa Topic using below command
Command : kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic 03-sbms-topic
Step-7 : View created Topics using below command
Command : kafka-topics.bat --list --zookeeper localhost:2181
Step-8 : Create Spring Boot Project in IDE
Step-9: Add below kafka related dependencies in pom.xml
Microsoft Windows [Version 10.0.19043.1526]
(c) Microsoft Corporation. All rights reserved.
E:\softwares\kafka_2.12-2.2.0\bin\windows>kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic demo-sbms-topic
Created topic demo-sbms-topic.
E:\softwares\kafka_2.12-2.2.0\bin\windows>
zookeeper
Kafka
req
PORT:9090
spring-boot-app
(Publisher)
publish
Topic
subscribe
PORT : 7070
spring-boot-app
(subscriber)


Kafka
1
req
PORT:9090
spring-boot-app
(Publisher)
publish
Topic
subscribe
2
PORT : 7070
spring-boot-app
(subscriber)
kafkaTemplate.send(topicname, obj)
@KakfaListener(topci-name)
java
OracleDriver
oracle
db
spring
kafkatemplate
kafka

Assocation Mapping
==========

==========
Association Mapping / Relationships in DB tables
==========

=> We can divide DB side relations into 4 types
Note: To establish relationships between table we will use Foregin Key.

1) One To One (Ex: One Person will have one Passport)
2) One To Many (Ex : One Employee will have Multiple Addresses)
3) Many To One (Ex: Multiple Books belongs to one Author)
4) Many To Many (Ex: Multiple Users having Multiple Roles)

=> When DB tables are having Relation then we need to represent that relation in our Entity classes also.

=> The process of representing DB tables relation in Entity classes is called as Association Mapping.

Cascade Type : Default Type is NONE : It represents operations on parent record should reflect on child record or not.

Ex: When we delete parent record then we want to delete all Child Records of that Parent.

Fetch Type : Default Type is LAZY : It represents weather to load child records along with Parent or not.

Ex: When we retrieve Parent record then i want to retrieve all child records of that Parent.

==========One To One Relation==========

```
@Entity
@Data
public class Person {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Integer personId;

	private String personName;

	private String personGender;

	@OneToOne(mappedBy = "person", cascade = CascadeType.ALL)
	private Passport passport;

}

@Entity
@Data
public class Passport {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Integer passportId;

	private String passportNum;

	private LocalDate issuedDate;

	private LocalDate expiryDate;

	@OneToOne
	@JoinColumn(name = "person_id")
	private Person person;

}
```

```
@SpringBootApplication
public class Application {

	public static void main(String[] args) {
ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);
		PersonRepository personRepo = context.getBean(PersonRepository.class);
	PassportRepository passportRepo = context.getBean(PassportRepository.class);

		/*Person person = new Person();
		person.setPersonName("Ashok");
		person.setPersonGender("Male");

		Passport passport = new Passport();
		passport.setPassportNum("KV7979HKI");
		passport.setIssuedDate(LocalDate.now());
		passport.setExpiryDate(LocalDate.now().plusYears(10));

		person.setPassport(passport);
		passport.setPerson(person);

		personRepo.save(person);*/

		//personRepo.findById(1);

		personRepo.deleteById(1);

	}
}
```

=> When we insert parent record (person) then child record (passport) also getting inserted

=> When we retrieve parent record then it is retriving both parent and child records

=> When we delete parent record then it is deleting child record also

==========One To Many Mapping ==========

```
@Entity
@Data
public class Employee {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Integer empId;
	private String empName;
	private Double empSalary;

	@OneToMany(mappedBy = "emp", cascade = CascadeType.ALL, fetch = FetchType.EAGER)
	private List<Address> addr;

}

@Entity
@Data
public class Address {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Integer addrId;
	private String city;
	private String state;
	private String country;

	@ManyToOne(fetch=FetchType.LAZY)
	@JoinColumn(name = "emp_id")
	private Employee emp;
}

public interface EmpRepository extends JpaRepository<Employee, Integer>{

}

public interface AddressRepository extends JpaRepository<Address, Integer>{

}
```

```
@SpringBootApplication
public class Application {

	public static void main(String[] args) {
ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);
EmpRepository empRepository = context.getBean(EmpRepository.class);
AddressRepository addrReposiotry = context.getBean(AddressRepository.class);

		Employee e = new Employee();
		e.setEmpName("Raja");
		e.setEmpSalary(4000.00);

		Address a1 = new Address();
		a1.setCity("Hyd");
		a1.setState("TG");
		a1.setCountry("India");
		a1.setEmp(e);

		Address a2 = new Address();
		a2.setCity("GNT");
		a2.setState("AP");
		a2.setCountry("India");
		a2.setEmp(e);

		// setting addresses to emp
		List<Address> addrList = Arrays.asList(a1, a2);
		e.setAddr(addrList);

		// empRepository.save(e);

		// empRepository.findById(2);

		// empRepository.deleteById(1);

		// addrReposiotry.findById(3);
	}
}
```

=> When we insert parent (emp) record then child (address) records also inserted

=> When we retrieve parent record then child records retriveal depends on FETCHTYPE

Lazy => Retrieve only parent record (It is default)
Eager => Retrieve parent record + child records

=> When we delete parent record then child records also getting deleted.

=> When we retrieve child record it is retrieveing child + parent record.

=> We can't delete child record directley when it is having relationship with parent.

Note: To delete particular child record then we need to make it as ORPHAN record and then we can delete.

Orphan Record : The record which doesn't have any relationship with parent.

========== Many To Many ==========

=> Multiple Authors will publish Multiple Books
=> Multiple Books are having Multiple Lessons
=> Multiple Users are having Mulitple Roles

==========

```
@Entity
@Data
public class User {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Integer userId;
```

```
	private String username;
	private LocalDate userDob;
	private String gender;


@ManyToMany(cascade = { CascadeType.PERSIST, CascadeType.MERGE })
@JoinTable(
                    name="user_roles",
                    joinColumns = {
                  @JoinColumn(name = "user_id")
                },
                inverseJoinColumns = {
                  @JoinColumn(name = "role_id")
                }
	)
	private List<Role> roles;

}

@Entity
@Data
public class Role {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Integer roleId;
	private String roleName;

	@ManyToMany(mappedBy = "roles")
	private List<User> users;

}

package in.ashokit.service;

import java.time.LocalDate;
import java.util.Arrays;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```
import in.ashokit.entities.Role;
import in.ashokit.entities.User;
import in.ashokit.repo.RoleRepo;
import in.ashokit.repo.UserRepo;

@Service
public class UserRoleService {

	@Autowired
	private UserRepo userRepo;
	@Autowired
	private RoleRepo roleRepo;

	public void saveUserWithRoles() {

		Role r1 = new Role();
		r1.setRoleName("Manager");

		Role r2 = new Role();
		r2.setRoleName("Admin");

		User user = new User();
		user.setUsername("Ashok");
		user.setGender("Male");
		user.setUserDob(LocalDate.now().minusYears(20));
		user.setRoles(Arrays.asList(r1, r2));

		userRepo.save(user);

	}

	public void getUser(int id) {
		Optional<User> findById = userRepo.findById(id);

	}

	public void getRole(int id) {
		Optional<Role> findById = roleRepo.findById(id);

	}}
```

J-UNIT

==========

What is Unit Testing ?

==========

=> The process of testing individual units of software is called as Unit testing
=> Unit means piece of code (we can consider one method as one unit)
=> The purpose of Unit Testing is to verify the behaviour of our code and fix if there are any bugs
=> The main aim of unit testing is to provide high quality code (bug free code)
=> Developers are responsible to perform Unit Testing
=> Unit Testing will happen in development phase only

==========

What is Junit ?

==========

-> Free & Open source software
-> Used for java projects unit testing
-> Releasedi in 1997
-> Latest Version of Junit is 5.x

Note: It is used for only Java projects and it is developed using Java.

1) Junit Jupiter : Programming Support + New Annotations
2) JUnit Vintage : Backward compatability (Junit 3 and Junit 4)
3) Junit Platform : Environment to run our Junit test cases

==========

Junit Annotations

==========

@Test
@BeforeEach
@AfterEach
@BeforeAll
@AfterAll
@ParameterizedTest
@ValueSource

==========

Junit Assertions

==========

-> Assertions are used to compare expected results with actual results
-> Assertions class provided several static methods

assertEquals( )
assertNotEquals ( )
assertNull ( )
assertNotNull ( )
assertThrows ( )

==========

Mocking

==========

=> The process of creating substitute object for real object is called as Mocking
Ex: Service layer methods is calling DAO layer method

=> When we are performing Unit testing for service then only service method code we should test (DAO code shouldn't execute).

Note: Here we will create Mock object for DAO and inject into service.

=> We have several Mock frameworks to implement Mocking.

1) EasyMock
2) PowerMock
3) Wiremock
4) Mockito

1) What is Unit Testing ?
2) Why Unit Testing ?
3) When to perform Unit Testing ?
4) What is Junit ?
5) JUnit annotations

6) JUnit Assertions
7) What is Mocking
8) Mock Frameworks
9) Creating Mock Object and Define Mock Object Behaviour

==========

How to perform Unit Testing For REST API

==========

@WebMvcTest : It is represent which class we are testing and it provides required environment for that

@MockBean : It is used to create Mock object for class / interface

MockMvc : This class provides methods to make HTTP Request

```
@WebMvcTest(value = WelcomeRestController.class)
public class WelcomeRestControllerTest {

	@MockBean
	private WelcomeService service;

	@Autowired
	private MockMvc mvc;

	@Test
	public void welcomeMsgTest() throws Exception {

	when(service.getWelcomeMsg()).thenReturn("Welcome to Ashok IT...");

MockHttpServletRequestBuilder reqBuilder = MockMvcRequestBuilders.get("/welcome");

		MvcResult mvcResult = mvc.perform(reqBuilder).andReturn();

		MockHttpServletResponse response = mvcResult.getResponse();
```

```
		int statusCode = response.getStatus();

		assertEquals(200, statusCode);
	}
}
```

==========

What is Code Coverage ?

==========

=> It is the process of identifying how many lines of code is tested as part of Unit Testing in our project.

Note: Industry Standard is 80% of coverage.

=> We can use Jacocco plugin to see code coverage Report.

==========Jacocco Plugin ==========

```
<plugin>
	<groupId>org.jacoco</groupId>
	<artifactId>jacoco-maven-plugin</artifactId>
	<version>0.8.2</version>
	<executions>
		<execution>
			<goals>
				<goal>prepare-agent</goal>
			</goals>
		</execution>
		<execution>
			<id>report</id>
			<phase>test</phase>
				<goals>
					<goal>report</goal>
				</goals>
		</execution>
	</executions>
</plugin>
```

==========

=> When we join in a project, first 3 Months they will assign below tasks for you

1) Sonar Issues Fixing

2) JUnits Implementation

3) Code Coverage Improvement

==========
JMETER
==========

-> JMETER is a free & open source software given by Apache Organization
-> JMETER is used for performance testing

-> Performance testing means the process of verifying stability & responsiveness of the application

-> How our application is responding for different work loads we can verify using JMETER

-> Using JMETER we can create virtual users to test our application performance

-> JMETER is a java based desktop application

-> Using JMETER we can test performance of any web application

Note: We can't implement performance testing manually

What is the response time of our application for 100 users ?
What is the response time of our application for 200 users ?
What is the response time of our application for 300 users ?
What is the response time of our application for 400 users ?
What is the response time of our application for 1000 users ?

-> We need to provide application performance details to client

-> If application performance is slow then we need to troubleshoot the issue and we need to fix it.

==========

JMETER Setup

==========

1) Download JMETER software

URL : https://dlcdn.apache.org//jmeter/binaries/apache-jmeter-5.5.zip

2) Extract JMETER zip file

3) Goto JMETER bin folder and run "jmeter.bat" file (it will open JMETER tool)

==========

Creating Test Plan

==========

1) Right Click on Test plan

- Add Threads
- Add Thread Group
- Enter Thread/users count

2) Right Click on Thread Group (For sampler)

- Add Sampler
  - Http Request
  - Add Server IP, Port Number, URL Pattern

3) Right Click on Thread Group (For Listerns)

- Add Listener
  - Add View Results Tree
  - Add Summary Report

4) Save the test and run the test

5) Verify the results (we can change thread group count and we can test it)

==========

JMETER Best Practise

==========

-> Create the TEST in GUI mode and run the test in CLI mode

Cmd to execute the test :::: jmeter -n -t test-plan.jmx -l test-results.jtl

-> After test execution complete we can import JTL file into JMETER summary report to see the test results.

==========

How to read data from YML file to Java Class

==========

----------application.yml----------

```
app:
  messages:
    welcomeMsg: Welcome To Ashok IT..!!
    greetMsg: Good Evening..!!
```

----------ApplicationProps.java----------

```
@Configuration
@EnableConfigurationProperties
@ConfigurationProperties(prefix="app")
public class AppProperties {

	private Map<String, String> messages = new HashMap<>();
	public Map<String, String> getMessages() {
		return messages;
	}

	public void setMessages(Map<String, String> messages) {
		this.messages = messages;
	}
}
```

----------MessageController.java----------

```
package in.ashokit.rest;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import in.ashokit.props.AppProperties;
```

```
@RestController
public class MsgController {

	@Autowired
	private AppProperties appProps;

	@GetMapping("/welcome")
	public String getWelcomeMsg() {

		Map<String, String> messages = appProps.getMessages();
		String msg = messages.get("welcomeMsg");

		return msg;
	}

	@GetMapping("/greet")
	public String getGreetMsg() {
		Map<String, String> messages = appProps.getMessages();
		String msg = messages.get("greetMsg");
		return msg;
	}
}
```
