5B)

 **Creating a Maven Project in Eclipse**:

* Navigate to File -> New -> Other -> Maven Project.
* Select an archetype (template), such as "quick start" version 1.4, for a basic Java project.
* Provide a Group ID (e.g., com.app.demo) and Artifact ID (project title).

 **Project Structure**:

* Generated folders include SRC/main/java (main code), SRC/test/java (test files), and a pom.xml file for managing dependencies and project settings.
* pom.xml is crucial for defining project configurations.

 **Building the Project Manually**:

* Run Maven lifecycle steps:
  + **Maven Clean**: Deletes previous builds.
  + **Maven Install**: Compiles code and converts it to bytecode.
  + **Maven Test**: Runs tests using JUnit.
  + **Maven Build**: Generates a .jar file.
* Customize the build goals (e.g., clean install test or package).

 **Testing the Project**:

* After building, run the project as a Java application to verify output (e.g., "Hello World" displayed in the console).

 **Pushing the Project to GitHub**:

* Create a GitHub repository and use the Maven .gitignore template to exclude unnecessary files.
* Clone the repository locally via git bash.
* Copy the project files to the cloned directory and commit them using Git commands (git add, git commit, git push).

 **Verifying the Upload**:

* Refresh the GitHub repository to confirm all files are uploaded correctly.

 **Next Steps**:

* Upcoming session will cover creating a Maven web project and another method to push projects to GitHub.

5C)

 **Introduction to Maven Web Project**

* The session focuses on creating a Maven web project in Eclipse IDE and pushing it to GitHub.
* Explains Maven's manual build process to understand its working.

 **Setting Up the Project**

* Navigate to **File > New > Other > Maven Project** in Eclipse.
* Select **Maven archetype-webapp** (version 1.4) for a pre-configured web project template.
* Name the project and check its structure (e.g., index.jsp, pom.xml).

 **Editing the Project Files**

* Customize index.jsp to display content (e.g., "Hello, welcome to SEAB 2024").
* Modify pom.xml to add dependencies manually by searching the Maven repository (e.g., javax.servlet API for server interaction).

 **Adding Dependencies**

* Dependencies allow the project to interact with a web server.
* Example: Add javax.servlet API by copying its Maven dependency snippet.

 **Building the Project**

* Use Maven lifecycle commands in Eclipse:
  + Maven Clean: Removes previous builds.
  + Maven Install: Compiles the code.
  + Maven Test: Tests the code using pre-installed JUnit dependencies.
  + Maven Build: Executes all steps and generates a WAR file for deployment.

 **Deploying the Project**

* Deploy the WAR file to a server (e.g., Tomcat 9).
* Run the project on the server and verify the output on a web browser (e.g., "Hello, welcome to SEAB 2024").

 **Pushing to GitHub**

* Create a GitHub repository with a .gitignore file (specific to Maven).
* Use Git Bash in Eclipse to initialize the repository, add files, commit changes, and push to GitHub using commands (git init, git add, git commit, git push).

 **Version Control**

* Ensure connection between the project and GitHub using git remote add origin.
* Push files to the repository branch (e.g., master) and verify on GitHub.

 **Next Steps**

* The next session will cover integrating Jenkins for automated builds and deployments.

6A)

 **Introduction to Experiment**:

* The session focuses on building a CI/CD pipeline for Maven Java projects using Jenkins.
* Previous experiments included creating Maven Java and web projects in Eclipse and pushing them to GitHub.

 **Jenkins Overview**:

* Jenkins projects automate tasks like building, testing, and deploying code.
* Different project types in Jenkins include Freestyle, Maven, Pipeline, Multi-Configuration, Multibranch, and Organization Folder projects.

 **Freestyle Project for Build**:

* Steps include:
  + Creating a project and naming it for identification (e.g., "Sample Maven Project Build").
  + Configuring the source code repository (GitHub link) and specifying the branch (e.g., main).
  + Adding build steps using Maven goals: clean (to remove previous builds) and install.

 **Archiving and Triggering Next Steps**:

* Post-build actions include archiving artifacts (saving all files and logs) and triggering another project for testing.

 **Freestyle Project for Test**:

* Steps include:
  + Creating a second project (e.g., "Sample Maven Project Test").
  + Configuring it to use artifacts from the build project only if it is successful.
  + Adding build steps using Maven goals: test.
  + Archiving test results as artifacts.

 **Pipeline Setup**:

* Pipelines provide a graphical representation of linked projects (e.g., build and test).
* Created a pipeline named "Sample Maven Project Pipeline" to integrate build and test.
* Pipeline execution: Blue indicates not started, yellow indicates execution, and green indicates success.

 **Execution and Monitoring**:

* Running the pipeline triggers both build and test projects in sequence.
* Logs and artifacts are viewable for both build and test steps.

 **Key Takeaways**:

* CI/CD pipelines in Jenkins allow automation of build and test processes.
* Successful pipeline execution combines multiple projects with a single click and provides clear visual feedback.

 **Next Steps**:

* Future sessions will cover deploying projects to specific environments using Maven web projects.

6B)

**Building a CI/CD Freestyle Pipeline using Jenkins for a Maven Web Project**

1. **Introduction and Objective**:
   * The session focuses on building a CI/CD pipeline for a Maven web project using Jenkins.
   * Three Jenkins projects are created: **Build**, **Test**, and **Deploy**.
2. **Setup and Configuration**:
   * The Maven web project is deployed to a **Tomcat 9 server**.
   * The source code is managed using **Git**, with the repository URL and branch (master) configured in Jenkins.
3. **Build Triggers**:
   * Four types of build triggers in Jenkins were discussed:
     + Trigger build remotely.
     + Build after other projects.
     + Build periodically.
     + GitHub hook trigger (not used due to local Jenkins setup, alternative: **Poll SCM**).
4. **Poll SCM**:
   * Configured to check the repository every minute using a cron schedule (\* \* \* \* \*).
   * Automatically triggers the build when changes are detected in the GitHub repository.
5. **Build Steps**:
   * Added build steps to:
     + **Clean** the project.
     + **Install** Maven dependencies.
     + Archive all artifacts (\*\*/\*\*).
6. **Testing Pipeline**:
   * The **Test** project copies artifacts from the **Build** project.
   * Configured Maven to execute the **test** goal.
   * Archived test results.
7. **Deployment Pipeline**:
   * The **Deploy** project copies artifacts from the **Test** project.
   * Deploys .war files to the **Tomcat 9 server**.
   * Credentials and Tomcat server details are configured for deployment.
8. **Pipeline Visualization**:
   * A Jenkins pipeline view is created to visualize the Build-Test-Deploy workflow.
9. **Automation Verification**:
   * Demonstrated an automated pipeline trigger when changes are made to the GitHub repository.
   * Successfully deployed changes to the Tomcat server and verified the updated output.
10. **Conclusion**:
    * Freestyle pipelines in Jenkins can be set up to automate build, test, and deployment processes.
    * Poll SCM effectively handles change detection and triggers automation.

6C)

Step 1: Click on “New Item”.

Step 2: Enter a project name.

Step 3: Select the third option, “Pipeline”, and click “OK”.

Select “Build Periodically”: This option allows Jenkins to trigger builds at specified intervals, even without any code changes.

•Example: \*\*H/15 \* \* \* \*\*\* - Triggers a build every 15 minutes.

pipeline {

agent any

tools {

maven 'MAVEN\_HOME'

}

stages {

stage('git repo & clean') {

steps {

bat """

if exist SE (

rmdir /s /q SE

) else (

echo Directory SE does not exist, skipping deletion.

)

"""

bat "git clone https://github.com/sathwikbalu/SE.git"

bat "mvn clean -f SE"

}

}

stage('install') {

steps {

bat "mvn install -f SE"

}

}

stage('test') {

steps {

bat "mvn test -f SE"

}

}

stage('package') {

steps {

bat "mvn package -f SE"

}

}

}

}

7A)

 **docker version**

* **Purpose:** Checks if Docker is successfully installed.
* **Output:** Displays the installed version of Docker.

 **docker pull <image-name>**

* **Purpose:** Downloads a specific image from Docker Hub.
* **Example:** docker pull hello-world pulls the "Hello World" image.
* **Notes:** Requires a stable internet connection.

 **docker images**

* **Purpose:** Lists all Docker images stored locally.
* **Output:** Displays details like repository, tag, image ID, creation date, and size.

 **docker run <image-name>**

* **Purpose:** Runs a Docker image as a container.
* **Example:** docker run hello-world executes the "Hello World" image.
* **Output:** Displays confirmation if the container runs successfully.

 **docker ps -a**

* **Purpose:** Lists all Docker containers, including running, stopped, and exited ones.
* **Output:** Provides container ID, image name, status, creation date, and other details.

 **docker rm <container-ID>** *(Referenced but not directly mentioned in the transcript)*

* **Purpose:** Deletes a specific container by its ID.
* **Example:** docker rm <container-ID> removes the specified container.
* **Notes:** Useful for cleanup of exited or unused containers.

**Creating the Project Folder**

* Navigate to the project folder, create a new folder for the project (e.g., for Redis image).
* Create an empty file named Dockerfile without any extensions (ensure only the "D" is capitalized).

Inside the folder, create a file named Dockerfile (no extension). 2. Add the following content:

FROM redis:latest

CMD ["redis-server"]

Docker Commands (Step-by-step):

* 1. docker build -t redisnew .
  2. docker run --name myredisnew -d redisnew
  3. docker ps
  4. docker stop myredisnew
  5. docker login
  6. docker ps -a
  7. docker commit <id> <username>/redis1
  8. docker push <username>/redis1
  9. docker rm <id>
  10. docker rmi <username>/redis1
  11. docker ps -a
  12. docker logout
  13. docker pull <username>/redis1
  14. docker run --name myredis -d <username>/redis1
  15. docker exec -it myredis redis-cli

7B)

**Modify and Push Docker Image**

1. **Pull an Image**  
   Use docker pull ubuntu to download the Ubuntu base image.
2. **Run a Container**  
   Start a container with docker run -it --name new -d ubuntu.
3. **Modify the Container**
   * Enter the container: docker exec -it <container\_id> bash.
   * Update packages: apt update.
   * Install Git: apt install git -y.
   * Verify Git installation with git --version.
4. **Save Changes as a New Image**
   * Exit the container: exit.
   * Stop the container: docker stop <container\_id>.
   * Commit changes:

docker commit <container\_id> username/repository:new\_tag

1. **Push the New Image to Docker Hub**
   * Log in: docker login.
   * Push the image:

docker push username/repository:new\_tag

1. **Test the Pushed Image**
   * Remove the local container and image:

docker rm <container\_id>

docker rmi username/repository:new\_tag

* + Pull the image back:

docker pull username/repository:new\_tag

* + Verify modifications by running the container:

docker run -it --name test\_container username/repository:new\_tag

7C) CREATE AND PUSH DOCKER FILE IMAGE

Step 1: Create the JavaScript File

// calculator.js

function add(a, b) {

return a + b; }

function subtract(a, b) {

return a - b; }

function multiply(a, b) {

return a \* b; }

function divide(a, b) {

if (b === 0) {

return "Cannot divide by zero!";

}

return a / b; }

// Print the calculated values

console.log("Addition (2 + 3):", add(2, 3));

console.log("Subtraction (5 - 2):", subtract(5, 2));

console.log("Multiplication (4 \* 3):", multiply(4, 3));

console.log("Division (10 / 2):", divide(10, 2));

Step 2: Create a Dockerfile

FROM node:16-alpine

WORKDIR /app

COPY calculator.js /app

CMD ["node", "calculator.js"]

Open a terminal in the directory containing your Dockerfile and calculator.js.

Step3 :Docker cmds:

* 1. docker build -t simple-calculator .
  2. docker run simple-calculator
  3. docker login
  4. docker tag simple-calculator your-dockerhub-username/simplecalculator
  5. docker push your-dockerhub-username/simple-calculator
  6. docker ps -a
  7. docker rm
  8. docker rmi your-dockerhub-username/simple-calculator
  9. docker pull your-dockerhub-username/simple-calculator
  10. docker run your-dockerhub-username/simple-calculator
  11. docker ps -a

7C)

RUNNING MULTIPLE CONTAINERS USING DOCKER COMPOSE

Writing a Basic docker-compose.yml File

version: "3.1"

services:

db:

image: mysql:5.7

container\_name: mysql\_container

environment:

MYSQL\_ROOT\_PASSWORD: rootpassword

MYSQL\_DATABASE: wordpress\_db

MYSQL\_USER: wordpress\_user

MYSQL\_PASSWORD: wordpress\_pass

volumes:

- db\_data:/var/lib/mysql

wordpress:

depends\_on:

- db

image: wordpress:latest

container\_name: wordpress\_container

ports:

- "8000:80"

environment:

WORDPRESS\_DB\_HOST: db:3306

WORDPRESS\_DB\_USER: wordpress\_user

WORDPRESS\_DB\_PASSWORD: wordpress\_pass

WORDPRESS\_DB\_NAME: wordpress\_db

volumes:

- ./wordpress\_data:/var/www/html

volumes:

db\_data:

Docker cmds:

* 1. docker-compose up -d
  2. Go to <http://localhost:8000>.
  3. docker-compose stop
  4. docker-compose start
  5. docker-compose down

7E)

DEPLOYING AND SCALING APPLICATIONS USING MINIKUBE

Run this command to start the Minikube cluster:

minikube start

kubectl create deployment mynginx --image=nginx

kubectl get deployments

kubectl expose deployment mynginx --type=NodePort --port=80 -- target-port=80

kubectl scale deployment mynginx --replicas=4

kubectl port-forward svc/mynginx 8081:80

Open your browser and go to: <http://localhost:8081>

minikube tunnel

minikube service mynginx –url

kubectl delete deployment mynginx

kubectl delete service mynginx

minikube stop

minikube delete

7F) DEPLOYING AND MANAGING MONITORING SYSTEMS USING NAGIOS IN DOCKER

. Command to Pull Nagios Open a terminal and type this command to download the Nagios image:

docker pull jasonrivers/nagios:latest

* 1. Command to Run Nagios

docker run --name nagiosdemo -p 8888:80 jasonrivers/nagios:latest

* + 1. Open a browser and type: localhost:8888 This will open the Nagios web interface. 2. Login Credentials:

Username: nagiosadmin

Password: nagios

docker ps

Stop the Container: Use the following command to stop the Nagios container:

docker stop nagiosdemo

docker rm nagiosdemo

docker rmi jasonrivers/nagios:latest