MADURAI KAMARAJ UNIVERSITY COLLEGE

ALAGARKOVIL ROAD, MADURAI – 624710

SOFTWARE DEVELOPMENT TECHNOLOGIES LAB

RECORD NOTE BOOK

NAME : _____

BRANCH : _____ **SEMESTER** _____

UNIVERSITY REGISTER NO: _____

MADURAI KAMARAJ UNIVERSITY COLLEGE

ALAGARKOVIL ROAD, MADURAI – 624710

CERTIFICATE

NAME OF LABORATORY : _____

UNIVERSITY REGISTER NO : _____

BRANCH: _____SEMESTER: _____

Certified that this is the bonafide record of work done by _____during the Ist Semester in the year 2023-2024.

Staff in Charge **Head of the Department**

Date:

Submitted for the Practical Examination held on _____ at Madurai Kamaraj University College, Madurai.

EXTERNAL EXAMINER **EXTERNAL EXAMINER**

TABLE OF CONTENT

1. DEPLOY VERSION CONTROL SYSTEM / SOURCE CODE MANAGEMENT, INSTALL GIT AND CREATE A GITHUB

Setting Up Git and GitHub

Introduction: In this lab, you will learn how to set up a Version Control System (VCS) using Git and create a GitHub account. Version control is essential for managing and tracking changes in your source code or any other files, allowing multiple collaborators to work on projects efficiently.

Prerequisites:

- A computer with an internet connection
- Basic familiarity with the command line (Terminal or Command Prompt)
- A valid email address

Lab Steps:

1. Install Git:

- Visit the official Git website: https://git-scm.com/downloads
- Download the appropriate Git installer for your operating system (Windows, macOS, or Linux).
- Follow the installation instructions provided on the website to install Git on your machine.

Verify the installation by opening a terminal or command prompt and running the following command:

git –version

You should see the Git version number, confirming that Git is installed.

2. Configure Git:

- Open a terminal or command prompt.
- Set your name and email address to be associated with your Git commits:

git config --global user.name "Your Name"

git config --global user.email "your.email@example.com"

- Replace "Your Name" and "your.email@example.com" with your actual name and email.

3. Create a GitHub Account:

- Open your web browser and go to https://github.com.
- Click on the "Sign up" button to create a new GitHub account.
- Follow the registration steps, providing your username, email, and password.
- Complete the registration process by verifying your email.

5. Create Your First Repository:

- Login to your GitHub account.
- Click on the "+" sign in the top right corner and select "New repository."
- Follow the instructions to create a new repository, providing a name and optional description.

- You can choose to initialize the repository with a README, which is useful for documenting your project.
- Click the "Create repository" button.

6. Clone Your Repository:

- Go to your newly created repository on GitHub.
- Click the "Code" button, and copy the repository URL provided (either HTTPS or SSH).
- In your terminal or command prompt, navigate to the directory where you want to store your local copy of the repository.
- Clone the repository by running:

```
git clone git@github.com:username/repository.git
```

Replace **<repository_url>** with the URL you copied from GitHub

Result:

The above program has been successfully executed.

2. PERFORM VARIOUS GIT OPERATIONS ON LOCAL AND REMOTE REPOSITORIES USING GIT CHEAT-SHEET

Git Configuration:

- Set your name and email (global configuration):

```
git config --global user.name "Your Name"
git config --global user.email "youremail@example.com"
```

Local Repository Operations:

1. **Initialize a Git repository**:

```
git init
```

2. **Clone a remote repository**:

```
git clone <repository_url>
```

3. **Add changes to the staging area**:

```
git add <file(s)>
```

4. **Commit changes with a message**:

```
git commit -m "Your commit message"
```

5. **View the status of your repository**:

```
git status
```

6. **View the commit history**:

```
git log
```

7. **Create a new branch**:

```
git branch <branch_name>
```

8. **Switch to a different branch**:

```
git checkout <branch_name>
```

9. **Merge changes from one branch into another**:

```
git merge <branch_name>
```

10. **Delete a branch**:

```
git branch -d <branch_name>
```

11. **Revert to a previous commit**:

```
git reset <commit_id>
```

12. **Discard changes in the working directory**:

```
git checkout -- <file(s)>
```

Remote Repository Operations:

1. **Add a remote repository**:

```
git remote add <remote_name> <repository_url>
```

2. **List remote repositories**:

```
git remote -v
```

3. **Push changes to a remote repository**:

```
git push <remote_name> <branch_name>
```

4. **Pull changes from a remote repository**:

```
git pull <remote_name> <branch_name>
```

5. **Fetch changes from a remote repository (without merging)**:

```
git fetch <remote_name>
```

6. **Remove a remote repository**:

```
git remote remove <remote_name>
```

7. **Create a new branch in a remote repository**:

```
git push <remote_name> <branch_name>
```

8. **Delete a branch in a remote repository**:

```
git push <remote_name> --delete <branch_name>
```

Tagging:

1. **Create an annotated tag**:

```
git tag -a <tag_name> -m "Tag message"
```

2. **Push tags to a remote repository**:

```
git push --tags
```

3. **List tags**:

```
git tag
```

4. **Checkout a specific tag**:

```
git checkout <tag_name>
```

These are some common Git operations. Remember to replace placeholders like **<repository_url>**, **<branch_name>**, **<remote_name>**, and **<tag_name>** with your actual values.

Result:

The above program has been successfully executed.

3. INSTALL AND CONFIGURE JENKINS WITH MAVEN/ANT/GRADLE TO SETUP A BUILD JOB.

Prerequisites:

1. Ensure you have Java Development Kit (JDK) installed on your system.
2. Have administrative privileges on the machine where you are installing Jenkins.

Installing Jenkins:

1.Linux:

Update your package manager:

sudo apt update

Install Jenkins:

sudo apt install jenkins

2.macOS:

Install Homebrew if not already installed:

/bin/bash -c "$(curl -fsSL

https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

Install Jenkins using Homebrew:

brew install jenkins

3.Windows:

Download the Jenkins installer for Windows from the official Jenkins website: Jenkins Windows Installer

Run the installer and follow the on-screen instructions.

Starting Jenkins:

- **Linux/macOS**:
  - Start Jenkins service:

    sudo systemctl start jenkins
- **Windows**:
  - Start Jenkins service from the Services application.
- Access Jenkins in your web browser by navigating to **http://localhost:8080**.
- Retrieve the Jenkins unlock key from the Jenkins installation directory (usually **/var/lib/jenkins/secrets/initialAdminPassword** on Linux) and follow the installation wizard to complete the setup.

Configuring Jenkins for Maven:

1. **Install Maven on your system** if you haven't already: Download Maven.
2. **Configure Maven in Jenkins**:
   - Go to Jenkins dashboard.

- Click on "Manage Jenkins" -> "Global Tool Configuration."
- Find the "Maven" section and click "Add Maven."
- Provide a name (e.g., "Maven") and specify the Maven installation directory.

Configuring Jenkins for Ant:

1. **Install Ant on your system** if you haven't already: Download Ant.
2. **Configure Ant in Jenkins**:
    - Go to Jenkins dashboard.
    - Click on "Manage Jenkins" -> "Global Tool Configuration."
    - Find the "Ant" section and click "Add Ant."
    - Provide a name (e.g., "Ant") and specify the Ant installation directory.

Configuring Jenkins for Gradle:

1. **Install Gradle on your system** if you haven't already: Download Gradle.
2. **Configure Gradle in Jenkins**:
    - Go to Jenkins dashboard.
    - Click on "Manage Jenkins" -> "Global Tool Configuration."
    - Find the "Gradle" section and click "Add Gradle."
    - Provide a name (e.g., "Gradle") and specify the Gradle installation directory.

Creating a Build Job:

1. Click on "New Item" on the Jenkins dashboard.
2. Enter a name for your project and select "Freestyle project."
3. In the configuration page, go to the "Source Code Management" section, choose your version control system (e.g., Git), and provide the repository URL.
4. In the "Build" section, choose your build tool (Maven, Ant, or Gradle) and configure build goals or targets accordingly.
5. Save your job configuration.

1. **Configure the Jenkins Pipeline**:
    - In the job configuration, scroll down to the "Pipeline" section.
    - Define the pipeline script to build your project. Here's a basic example for Maven:

For Ant or Gradle, replace the "Build" stage's script accordingly.

2. **Save and Build the Job**:
    - Click "Save" to save the job configuration.
    - To manually start the build, click "Build Now" from the job's dashboard. Jenkins will execute the pipeline defined in the script.
3. **View Build Results**:

- After the build is complete, you can view build results, including console output and test reports, from the job's dashboard.

4. **Setup Webhooks (Optional)**:
    - For automatic builds triggered by code changes, you can set up webhooks or use a version control system's post-commit hooks. Jenkins provides webhook support for many VCS platforms.

Now, Jenkins is set up to perform continuous integration using Maven, Ant, or Gradle for your project. You can trigger builds manually or configure webhooks to automate builds on code pushes. Make sure to customize your build job according to your project's specific requirements.

Result:

The above program has been successfully executed.

4. BUILD THE PIPELINE OF JOBS USING MAVEN / GRADLE / ANT IN JENKINS, CREATE A PIPELINE SCRIPT TO TEST AND DEPLOY AN APPLICATION OVER THE TOMCAT SERVER.

Prerequisites:

1. Jenkins installed and running.
2. Tomcat server installed and configured.
3. Version control system (e.g., Git) repository for your application.

Pipeline Script:

```
node {
   def mvnHome
   stage('Preparation') {
      git branch: 'main', url: 'https://github.com/bhaarn/gs-maven'
      mvnHome = tool 'Maven'
   }
   stage('Build') {
      withEnv(["PATH+MAVEN=${mvnHome}/bin"]) {
         dir('initial') {
            bat 'mvn -Dmaven.test.failure.ignore clean package'
            bat 'mvn compile'
         }
      }
   }
   stage('Package') {
      withEnv(["PATH+MAVEN=${mvnHome}/bin"]) {
         dir('initial') {
            bat 'mvn package'
            bat 'java -jar target/gs-maven-0.1.0.jar'
         }
      }
   }
   stage('Test') {
      withEnv(["PATH+MAVEN=${mvnHome}/bin"]) {
         dir('initial') {
            bat 'mvn test'
```

```
        }
    }
  }
}
```

This pipeline script does the following:

1. Checks out your source code from the version control system.
2. Builds the application using either Maven, Gradle, or Ant (you can choose one).
3. Deploys the built WAR file to the Tomcat server.
4. Runs tests (you can customize this part based on your testing needs).
5. Performs cleanup tasks after the pipeline is finished.

Make sure to replace **/path/to/tomcat**, **/path/to/maven**, and **/path/to/gradle** with the actual paths on your Jenkins server.

You can further customize this script to suit your specific needs and project structure. Additionally, consider setting up authentication and security configurations if your Tomcat server requires them for deployments.

Result:

The above program has been successfully executed.

5. IMPLEMENT JENKINS MASTER-SLAVE ARCHITECTURE AND SCALE YOUR JENKINS STANDALONE IMPLEMENTATION BY IMPLEMENTING SLAVE NODES.

Prerequisites:

1. Jenkins Master instance (already installed and running).
2. Access to additional machines (physical or virtual) that will act as Jenkins slave nodes.
3. Java installed on both the master and slave machines.

Setting Up Jenkins Master:

1. **Install Jenkins**:
    - Ensure Jenkins is installed and running on the master machine.
2. **Configure Jenkins Master**:
    - Install necessary plugins for your projects on the master instance.
    - Set up the necessary global configurations (e.g., Git credentials, Maven/Gradle/Ant installations) in Jenkins.

Setting Up Jenkins Slave Nodes:

1. **Install Java**:
    - Make sure Java is installed on each slave machine. Jenkins requires Java to run.
2. **Download Jenkins Agent (Slave) JAR**:
    - On each slave machine, download the Jenkins agent JAR file (**agent.jar**) from the Jenkins master. You can find the link to download the agent JAR on the Jenkins master web interface: Manage Jenkins > Manage Nodes > New Node > Permanent Agent.
3. **Start Jenkins Agent on Slave**:
    - Open a terminal on the slave machine and navigate to the directory where **agent.jar** is located.
    - Start the Jenkins agent using the following command, replacing **http://jenkins-master-url** with the URL of your Jenkins master and **secret** with the secret key obtained during the agent registration process.

      java -jar agent.jar -jnlpUrl http://jenkins-master-url/computer/slave-name/slave-agent.jnlp -secret <secret>
    - The agent will connect to the Jenkins master and become available for running jobs.
4. **Configure Jenkins Slave Node on Master**:
    - On the Jenkins master, go to Manage Jenkins > Manage Nodes > New Node.
    - Enter a node name and select "Permanent Agent."

- Enter the necessary details, such as the number of executors, remote root directory, and labels. The remote root directory should be the path where Jenkins workspace will be created on the slave machine.
- Click "Save" to add the slave node.

5. **Test the Configuration**:
    - Once the slave node is added, you can run jobs on it. When creating or configuring Jenkins jobs, specify the label you assigned to the slave node. Jenkins will schedule the jobs on the appropriate slave nodes based on the labels.

Scaling Your Jenkins Implementation:

1. **Add More Slave Nodes**:
    - You can repeat the steps to add more slave nodes to your Jenkins master, distributing the workload across multiple machines.
2. **Manage Load Balancing**:
    - If you have multiple Jenkins slave nodes, you can set up load balancing to distribute jobs evenly among the available nodes. Consider using load balancers like HAProxy or software solutions like Nginx for this purpose.
3. **Monitoring and Maintenance**:
    - Regularly monitor the performance of your Jenkins master and slave nodes. Tools like JVisualVM and Jenkins built-in monitoring can help identify performance bottlenecks.
    - Ensure that slave nodes are properly maintained, updated, and have sufficient resources to handle the workload.

By implementing a Jenkins Master-Slave Architecture and scaling your Jenkins setup with multiple slave nodes, you can significantly improve the efficiency and reliability of your CI/CD processes.

Result:

The above program has been successfully executed.

6. SETUP AND RUN SELENIUM TESTS IN JENKINS USING MAVEN

Prerequisites:

1. Selenium WebDriver tests written in your preferred programming language (Java, Python, etc.).
2. Jenkins installed and running.
3. Maven installed on the system where Jenkins is running.
4. Selenium WebDriver dependencies and configurations set up in your Maven project.

Steps:

1. **Configure Selenium Tests in Your Maven Project**:

Ensure your Maven project includes Selenium WebDriver dependencies and necessary configurations. Here's an example **pom.xml** file for a Java project using Selenium WebDriver:

```
<project>
  <groupId>com.example</groupId>
  <artifactId>selenium-tests</artifactId>
  <version>1.0-SNAPSHOT</version>

  <dependencies>
    <!-- Selenium WebDriver Dependency -->
    <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-java</artifactId>
      <version>3.141.59</version>
    </dependency>
    <!-- Other dependencies -->
  </dependencies>

  <!-- Build and configuration settings -->
</project>
```

2. **Push Your Code to Version Control**:

Make sure your Selenium test code, along with the **pom.xml** file, is pushed to a version control repository (e.g., Git, SVN).

3. **Configure Jenkins Job**:
    - In Jenkins, create a new Maven project job.
    - In the job configuration, under the "Source Code Management" section, specify the repository URL and credentials if required.

- In the "Build" section, add a build step: "Invoke top-level Maven targets".
- Specify the Maven goals, such as **clean test** or **verify**, depending on your project's setup.
- Save the Jenkins job configuration.

4. **Set Up Selenium WebDriver in Jenkins**:
    - If your Selenium tests require a web browser (Chrome, Firefox, etc.), ensure the corresponding WebDriver executable is available on the machine where Jenkins is running. You can download WebDriver executables and configure their paths in your Selenium tests.
    - Alternatively, you can use tools like WebDriverManager to automatically download and configure WebDriver executables during test execution.
5. **Run the Jenkins Job**:
    - Manually trigger the Jenkins job to run your Selenium tests.
    - Jenkins will fetch the code from the version control system, build the Maven project, and execute the Selenium tests.
6. **View Test Results**:
    - Jenkins will display the test results in the job console output.
    - You can also configure Jenkins to generate and publish test reports. For example, if you are using TestNG, surefire reports can be generated and published as artifacts.

Notes:

- Ensure that the Jenkins machine has the necessary network access to download dependencies and access the web application you are testing.
- If your tests require specific browser configurations, you might need to handle those configurations in your test code or use tools like Selenium Grid or Docker containers for browser isolation.
- Make sure to handle WebDriver instances properly, including opening and closing the browser windows, to avoid resource leaks.

By following these steps, you can set up and run Selenium tests in Jenkins using Maven, allowing you to automate your web application testing as part of your CI/CD pipeline.

Result:

The above program has been successfully executed.

7. IMPLEMENT DOCKER ARCHITECTURE AND CONTAINER LIFE CYCLE, INSTALL DOCKER AND EXECUTE DOCKER COMMANDS TO MANAGE IMAGES AND INTERACT WITH CONTAINERS.

Prerequisites:

1. A machine or virtual machine with a supported operating system (Linux, Windows, or macOS) to install Docker.
2. Administrative or root access to the machine to install Docker.
3. Basic familiarity with the command line.

Steps:

1. Install Docker:

- **Linux**:
  - Use your distribution's package manager to install Docker (e.g., **apt**, **yum**, or **dnf**). You may also use the official Docker installation script:

    curl -fsSL https://get.docker.com | sh
  - Start the Docker service:

    sudo systemctl start docker
- **Windows**:
  - Download the Docker Desktop for Windows from the Docker website and install it.
- **macOS**:
  - Download the Docker Desktop for Mac from the Docker website and install it.

2. Verify Docker Installation:

- Open a terminal or command prompt and run the following command to verify that Docker is installed correctly:

  docker --version

3. Docker Architecture:

- Docker uses a client-server architecture. The Docker client communicates with the Docker daemon (server) to build, run, and manage containers.

4. Docker Container Life Cycle:

- **Create a Docker Container**:

  docker run -d --name my-container -p 8080:80 nginx
  - This command runs an NGINX web server in a container named **my-container** and maps port 8080 on your host to port 80 in the container.
- **List Running Containers**:

  docker ps
- **Stop a Container**:

docker stop my-container

- **Start a Container**:

  docker start my-container

- **Remove a Container**:

  docker rm my-container

- **List All Containers (including stopped ones)**:

  docker ps -a

- **Pull an Image**:

  docker pull ubuntu:latest

- **Build an Image from a Dockerfile**:
  - Create a Dockerfile describing the image you want to build.
  - Build the image:

    docker build -t my-custom-image .

- **List Docker Images**:

  docker images

- **Remove an Image**:

  docker rmi my-custom-image

5. Run a Container Interactively:

- To run a container and interact with it, you can use the following command:

  docker run -it --name interactive-container ubuntu:latest /bin/bash

  - This runs an Ubuntu container in interactive mode and starts a Bash shell.

6. Cleanup:

- Periodically, it's a good idea to clean up unused containers and images to free up disk space.
  - Remove all stopped containers:

    docker container prune

  - Remove all dangling (unused) images:

    docker image prune -a

These are the basic steps to install Docker, understand its architecture, and manage containers and images. Docker provides a wide range of features and options, and you can explore more advanced use cases as your needs grow.

Result:

The above program has been successfully executed.

8. IMPLEMENT DOCKERFILE INSTRUCTIONS, BUILD AN IMAGE FOR A SAMPLE WEB APPLICATION USING DOCKERFILE.

Sample Node.js Web Application:

For this example, we'll create a simple Node.js web application that serves a "Hello, Docker!" message. You can replace this with your own application code.

1. **Create a directory for your project and navigate to it**:

   mkdir my-node-app

   cd my-node-app

2. **Create a basic Node.js application**:

   Create a file named **app.js** with the following content:

```
const http = require('http');

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello, Docker!\n');
});

const port = 8080;
server.listen(port, () => {
  console.log(`Server is running at http://localhost:${port}`);
});
```

3. **Create a package.json file**:

   Create a file named **package.json** with the following content to define the project's dependencies:

```
    {
  "name": "my-node-app",
  "version": "1.0.0",
  "description": "A simple Node.js app for Docker",
  "main": "app.js",
  "dependencies": {},
  "scripts": {
    "start": "node app.js"
  },
  "author": "Your Name"
```

```
}
```

4. **Initialize the project**:

   Run the following command to create a **node_modules** directory and initialize the project:

   npm init -y

5. **Install Express (optional)**:

If you want to use the Express.js framework, install it by running:

npm install express

Modify your **app.js** to use Express if you installed it.

Create a Dockerfile:

1. Create a file named **Dockerfile** (no file extension) in the project directory. Define the Dockerfile instructions as follows:

```
# Use an official Node.js runtime as a parent image
FROM node:14

# Set the working directory in the container
WORKDIR /usr/src/app

# Copy package.json and package-lock.json to the container
COPY package*.json ./

# Install project dependencies
RUN npm install

# Copy application code to the container
COPY . .

# Expose port 8080 to the outside world
EXPOSE 8080

# Define the command to run your application
CMD ["npm", "start"]
```

Build the Docker Image:

1. Open a terminal or command prompt and navigate to the project directory containing the Dockerfile.
2. Build the Docker image by running the following command:

   docker build -t my-node-app .

Replace **my-node-app** with your desired image name.

Run the Docker Container:

Once the image is built, you can run a container based on this image using the following command:

docker run -p 8080:8080 my-node-app

This command maps port 8080 on your host to port 8080 in the container. Access the application in a web browser by navigating to **http://localhost:8080**.

You've successfully created a Docker image for a sample Node.js web application using a Dockerfile. You can replace the sample application code with your own application for more complex use cases.

Result:

The above program has been successfully executed.

9. INSTALL AND CONFIGURE PULL BASED SOFTWARE CONFIGURATION MANAGEMENT AND PROVISIONING TOOLS USING PUPPET.

Prerequisites:

1. You will need two servers, one to act as the Puppet master and another as a Puppet agent (client).
2. Both servers should have a supported operating system (e.g., Ubuntu, CentOS, RHEL).

Installation and Configuration:

1. Install Puppet Master:

- On the server that will be your Puppet master, you'll need to install the Puppet Server. The steps may vary depending on your server's operating system.
    - **Ubuntu**:

      sudo apt update sudo apt install puppetserver
    - **CentOS/RHEL**:

      sudo rpm -Uvh https://yum.puppetlabs.com/puppetlabs-release-pc1-el-7.noarch.rpm sudo yum install puppetserver

2. Configure Puppet Master:

- Edit the Puppet Server configuration file (**/etc/puppetlabs/puppet/puppet.conf**) to specify the DNS name or IP address of your Puppet master. This file should include the following:

  [main] certname = puppetmaster.example.com

3. Start the Puppet Master Service:

- Enable and start the Puppet master service:

sudo systemctl enable puppetserver sudo systemctl start puppetserver

4. Install Puppet Agent on Clients:

- On the server(s) that will act as Puppet agents, install the Puppet agent. Use the same installation steps as for the master, but replace "puppetserver" with "puppet" for the package name.
    - **Ubuntu**:

      sudo apt update

      sudo apt install puppet
    - **CentOS/RHEL**:

      sudo rpm -Uvh https://yum.puppetlabs.com/puppetlabs-release-pc1-el-7.noarch.rpm

      sudo yum install puppet

5. Configure Puppet Agent:

- Edit the Puppet agent configuration file (**/etc/puppetlabs/puppet/puppet.conf**) on the agent to specify the Puppet master server:

  [main] server = puppetmaster.example.com

6. Start Puppet Agent Service:

- Enable and start the Puppet agent service on the client:

  sudo systemctl enable puppet

  sudo systemctl start puppet

7. Configure Puppet Manifests and Modules:

- Create Puppet manifests and modules on the Puppet master to define the desired configuration for your infrastructure. Puppet code typically resides in the **/etc/puppetlabs/code/environments/production/** directory on the master.
- Create Puppet classes, resources, and profiles to define how your servers should be configured.

8. Request Puppet Agent Run:

- On the Puppet agent, request a Puppet run to apply the configuration from the master:

  sudo puppet agent -t

- The agent will contact the master and apply the specified configuration.

9. Automation:

- You can schedule regular Puppet runs on the agents to ensure that the desired configuration is maintained. Use tools like cron jobs to automate this process.

10. Monitor and Manage:

- Monitor and manage your infrastructure by reviewing Puppet reports, making adjustments to your Puppet code, and scaling your Puppet environment as needed.

Result:

The above program has been successfully executed.

10.IMPLEMENT LAMP/MEAN STACK USING PUPPET MANIFEST

LAMP Stack with Puppet:

Here's an example of Puppet manifests to set up a LAMP stack:

1. Create Puppet Manifests:

- In your Puppet module directory (e.g., **/etc/puppetlabs/code/environments/production/modules/lamp),** create a file named **init.pp:**

```
class lamp {
  # Install Apache
  package { 'apache2':
    ensure => 'installed',
  }

  # Install PHP
  package { 'php':
    ensure => 'installed',
  }

  # Install MySQL Server
  class { 'mysql::server':
    root_password => 'your-root-password',
  }

  # Configure and start Apache
  service { 'apache2':
    ensure    => 'running',
    require   => Package['apache2'],
  }

  # Configure and start MySQL
  service { 'mysql':
    ensure    => 'running',
    require   => Class['mysql::server'],
  }
```

}

2. Install Puppet Modules:

- You need to install the necessary Puppet modules to manage Apache, PHP, and MySQL. You can install these modules from the Puppet Forge:

  puppet module install puppetlabs-apache

  puppet module install puppetlabs-mysql

3. Apply Puppet Manifest:

- Apply the Puppet manifest on the target server:

  sudo puppet apply -e "class { 'lamp': }"

This will install and configure Apache, PHP, and MySQL on your target server.

MEAN Stack with Puppet:

Setting up a MEAN stack requires configuring a web server (e.g., Nginx or Apache) and installing Node.js and MongoDB. Below is an example of Puppet manifests to set up a MEAN stack using Nginx as the web server:

1. Create Puppet Manifests:

- In your Puppet module directory (e.g., **/etc/puppetlabs/code/environments/production/modules/mean**), create a file named **init.pp**:

```
class mean {
 # Install Nginx
 package { 'nginx':
  ensure => 'installed',
 }

 # Install Node.js (you may need to adapt this based on your distribution)
 package { 'nodejs':
  ensure => 'installed',
 }

 # Install MongoDB
 class { 'mongodb::server':
  bind_ip => '127.0.0.1',
 }

 # Configure and start Nginx
 service { 'nginx':
```

```
  ensure => 'running',
 }

 # Configure and start MongoDB
 service { 'mongod':
  ensure => 'running',
 }
}
```

2. Install Puppet Modules:

- You need to install the necessary Puppet modules to manage Nginx and MongoDB. You can install these modules from the Puppet Forge:

puppet module install puppet/nginx

puppet module install puppetlabs-mongodb

3. Apply Puppet Manifest:

- Apply the Puppet manifest on the target server:

sudo puppet apply -e "class { 'mean': }"

This will install and configure Nginx, Node.js, and MongoDB on your target server.

Result:

The above program has been successfully executed.