

TABLE OF CONTENT

EX NO	DATE	NAME OF THE EXPERIMENT	SIGNATURE
1		Create Maven Build Pipeline in Azure.	
2		Run regression tests using Maven Build pipeline in Azure.	
3		Install Jenkins in Cloud	
4		Create CI pipeline using Jenkins	
5		Create a CD pipeline in Jenkins and deploy in Cloud	
6		Create an Ansible Playbook for a simple web application infrastructure.	
7		Build a simple application using Gradle.	
8		Install Ansible and configure ansible roles and to write playbooks.	

Ex. No: 1 Date :	Create Maven Build Pipeline in Azure
-----------------------------------	---

AIM:

The aim is to create a Maven Build pipeline in Azure DevOps. This pipeline will automate the process of building and packaging a Java project using Maven.

ALGORITHMS:

1. Sign in to Azure DevOps:

- Open your Azure DevOps account.
- Navigate to your project.

2. Create a new Build Pipeline:

- Go to Pipelines > Pipelines in the Azure DevOps portal.
- Click on "New pipeline" to create a new pipeline.

3. Select a Repository:

- Choose the repository where your Maven project is hosted (e.g., GitHub, Azure Repos Git).

4. Select a Template:

- Choose the Maven template as a starting point.

5. Configure Maven Build:

- Update the YAML file to specify the Maven goals and options.
- Ensure that necessary Maven tasks are configured, such as 'MavenAuthenticate', 'MavenPackage', etc.

6. Trigger Settings:

- Configure the triggers to automatically start the build pipeline on code commits or pull requests.

7. Save and Run:

- Save the pipeline configuration.
- Trigger the pipeline manually or wait for the configured triggers to initiate the build.

PROGRAM:

Yaml

Example Azure DevOps YAML configuration for Maven Build Pipeline trigger:

```
-      m
  aster
  pool:
    vmImage: 'ubuntu-latest'

  steps:
  - task:
    MavenAuthenticate@0
    inputs:
      mavenServiceConnection:
        'YourMavenServiceConnection' mavenPomFile:
        'path/to/your/pom.xml'
        options: '-Xmx3072m'

  - task: Maven@3
    inputs:
      mavenPomFile:
        'path/to/your/pom.xml' goals: 'clean
        package'
        options: '-Xmx3072m'
```

OUTPUT:

The output of the pipeline will be displayed in the Azure DevOps pipeline execution logs. It will include information about each step of the build process, such as Maven dependencies resolution, compilation, testing, packaging, and any other configured goals.

RESULT:

Upon successful execution of the pipeline, a packaged artifact (e.g., JAR file) generated by Maven. This artifact can be deployed to other environments for further testing or production use.

Ex. No: 2	Run regression tests using Maven Build pipeline in Azure
Date :	

AIM:

The aim is to extend the Maven Build pipeline created earlier to include the execution of regression tests. This will ensure that the automated tests are integrated into the continuous integration process.

ALGORITHM:

1. Update Maven Pipeline YAML:

- Open the YAML file of your existing Maven Build pipeline.
- Add a new Maven task to execute regression tests after the packaging step.

2. Configure Maven for Testing:

- Ensure that your Maven project's `pom.xml` file includes the necessary dependencies and configurations for running regression tests.
- Set up the appropriate test frameworks (e.g., JUnit, TestNG) and plugins.

3. Define Test Execution Goals:

- Specify the Maven goals for test execution in the pipeline YAML file (e.g., `clean test`).

4. Configure Test Reports:

- Integrate configurations in the pipeline to generate and publish test reports.
- Utilize plugins like the Surefire Plugin to manage test reports.

5. Save and Trigger:

- Save the updated pipeline configuration.
- Trigger the pipeline manually or wait for the configured triggers to initiate the build and test process.

PROGRAM:

Yaml

Updated Azure DevOps YAML configuration for Maven Build Pipeline with Regression Tests

trigger:

- master

pool:

vmImage: 'ubuntu-latest'

steps:

- task:

MavenAuthenticate@0

inputs:

mavenServiceConnection:

'YourMavenServiceConnection' mavenPomFile:

'path/to/your/pom.xml'

options: '-Xmx3072m'

- task: Maven@3

inputs:

mavenPomFile:

'path/to/your/pom.xml' goals: 'clean

package test'

options: '-Xmx3072m'

OUTPUT:

The output of the pipeline will now include the results of the regression tests. Test reports, logs, and any failures will be displayed in the Azure DevOps pipeline execution logs.

RESULT:

Upon successful execution of the pipeline, it will be able to review the test results in Azure DevOps. If any regression tests fail, the pipeline will indicate the issues, allowing for quick identification and resolution.

Ex. No: 3	Install Jenkins in Cloud
Date:	

AIM:

The aim is to install Jenkins on a cloud platform, providing a scalable and centralized solution for continuous integration and continuous delivery (CI/CD) processes.

ALGORITHM:

1. **Sign in to Azure Portal:**
 - Log in to the Azure portal with your credentials.
2. **Create a Virtual Machine (VM):**
 - Navigate to "Virtual machines" in the Azure portal.
 - Click on "Add" to create a new VM.
 - Follow the wizard to configure the VM, including selecting an image (e.g., Ubuntu), setting up authentication, and configuring networking.
3. **Open Ports for Jenkins:**
 - In the Azure portal, go to the "Networking" section of your VM.
 - Open ports 8080 (Jenkins web interface) and 50000 (Jenkins agent communication) for inbound traffic.
4. **Connect to VM:**
 - Connect to your VM using SSH.
 - Update package repositories and install Java.
5. **Install Jenkins:**
 - Add the Jenkins repository and key.
 - Install Jenkins using the package manager.
 - Start the Jenkins service.
6. **Access Jenkins Web Interface:**
 - Open a web browser and go to `http://your-vm-ip-address:8080`.
 - Follow the on-screen instructions to complete the Jenkins setup, including retrieving the initial admin password.
7. **Install Plugins:**
 - Install recommended plugins or customize the installation based on your requirements.
8. **Create Admin User:**
 - Create an admin user for Jenkins.
9. **Configure Jenkins URL:**
 - Set the Jenkins URL to match your public IP or domain.
10. **Complete Setup:**
 - Complete the Jenkins setup wizard.



OUTPUT:

Access Jenkins through the web browser, and you should see the Jenkins dashboard. Now we can start creating jobs, pipelines, and integrate Jenkins with your version control system.

RESULT:

Jenkins is successfully installed on the virtual machine in the cloud. You can use Jenkins to automate build, test, and deployment processes, improving the efficiency of your development workflow.

Ex. No: 4 Date :	Create CI pipeline using Jenkins
-----------------------------------	---

AIM:

The aim of this lab is to create a Continuous Integration pipeline in Jenkins, which automates the build and test process whenever changes are pushed to the version control system.

ALGORITHM:

1. Install Required Plugins:

- Open Jenkins in your web browser.
- Navigate to "Manage Jenkins" > "Manage Plugins."
- Install necessary plugins like Git, Pipeline, and any other plugins required for your project.

2. Create a New Pipeline Job:

- Click on "New Item" on the Jenkins dashboard.
- Choose "Pipeline" as the project type and provide a name for your job.

3. Configure Pipeline:

- In the pipeline configuration:
 - Define your pipeline script using Jenkins Pipeline DSL (Scripted or Declarative).
 - Specify the source code management system (e.g., Git) and provide the repository URL.
 - Configure the build triggers (e.g., GitHub webhook, periodic build).

4. Save and Run:

- Save the pipeline configuration.
- Manually trigger the pipeline for the first run or wait for automatic triggering based on the configured webhook or periodic build.

OUTPUT:

Viewing the console output of the pipeline to monitor the build and test process. The console output will show the progress of each stage and any errors or failures encountered during the pipeline execution.

RESULT:

Upon successful execution, Jenkins pipeline will have automated the build and test process of project. Future changes pushed to the version control system will trigger Jenkins to automatically build and test your project, providing continuous integration and early detection of issues.

Ex. No: 5	Create a CD pipeline in Jenkins and deploy in Cloud
Date :	

AIM:

The aim of this lab is to enhance the existing Jenkins pipeline to include deployment steps, allowing for continuous delivery of your application to a cloud environment.

ALGORITHM:

1. Install Required Plugins:

- Ensure that Jenkins has plugins installed for the cloud provider you are using (e.g., AWS, Azure).
- Install any additional plugins required for deployment steps.

2. Enhance Pipeline Script:

- Update your existing Jenkins pipeline script to include deployment stages.
- Add stages for deploying your application to the cloud.

3. Configure Cloud Credentials:

- In Jenkins, navigate to "Manage Jenkins" > "Manage Credentials."
- Add credentials for accessing your cloud provider's services (e.g., AWS Access Key, Secret Key).

4. Configure Cloud-Specific Settings:

- Update the pipeline script to include any cloud-specific settings or configurations.
- For example, if deploying to AWS, configure the AWS Elastic Beanstalk environment or S3 bucket.

5. Save and Run:

- Save the updated pipeline configuration.
- Trigger the pipeline manually or wait for automatic triggering based on the configured webhook or periodic build.

OUTPUT:

Viewing the console output of the extended pipeline to monitor the deployment process. The console output will show the progress of each stage, including the deployment steps.

RESULT:

Upon successful execution, your Jenkins pipeline will automate the deployment of your application to the cloud. Changes pushed to the version control system will trigger Jenkins to build, test, and deploy application, providing continuous delivery to cloud environment.

Ex. No: 6	Create an Ansible playbook for a simple web application infrastructure
Date :	

AIM:

The aim of this lab is to use Ansible to automate the setup of a simple web application infrastructure.

ALGORITHM:

1. Create Ansible Playbook:

- Create a new file named `web_app_infrastructure.yml` for your Ansible playbook.

2. Define Playbook Structure:

- Define the structure of your Ansible playbook, including hosts, tasks, and roles.

Yaml

```

---
- name: Setup Web Application Infrastructure
  hosts: web_servers
  become: true

  tasks:
    - name: Install
      Nginx apt:
        name: nginx
        state: present

    - name: Configure
      Nginx template:
src: nginx.conf.j2
        dest:
          /etc/nginx/nginx.conf
        notify: Reload Nginx

    - name: Install
      MySQL apt:
        name: mysql-server
        state: present

```

```
- name: Secure MySQL Installation
  mysql_secure_installation:
    login_user: root
    login_password: "{{ mysql_root_password }}" new_password: "{{
      mysql_root_password }}" validate_password: no

handlers:
  - name: Reload Nginx
    service:
      name: nginx
      state: reloaded
```

3. Create Nginx Configuration Template:

- Create a Jinja2 template for Nginx configuration. Save it as `nginx.conf.j2`.

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;

events {
worker_connections 768;
}

http {
sendfile on; tcp_nopush
on; tcp_nodelay on;
keepalive_timeout 65;
types_hash_max_size 2048;

include /etc/nginx/mime.types; default_type
application/octet-stream;
```

```

server {
listen 80;

    server_name {{ ansible_fqdn }};
    root /var/www/html;

location / {
index index.html;
    }

error_page 500 502 503 504 /50x.html; location =
    /50x.html {
root /usr/share/nginx/html;
    }
}
}

```

4. Create Inventory File:

- Create an Ansible inventory file (e.g., `inventory.ini`) with your server details.

```

[web_servers]
web_server ansible_host=your_web_server_ip
ansible_user=your_ssh_user

[database_servers]
db_server ansible_host=your_db_server_ip
ansible_user=your_ssh_user

```

5. Run the Playbook:

- Run the Ansible playbook using the command:

```

ansible-playbook -i inventory.ini web_app_infrastructure.yml --extra-
vars "mysql_root_password=your_mysql_root_password"

```


OUTPUT:

The console output as Ansible runs the playbook. It will display the progress of each task and report any errors encountered.

RESULT:

Upon successful execution of the Ansible playbook, web servers will have Nginx installed and configured, and database server will have MySQL installed and secured. The web application infrastructure is now set up according to the defined specifications.

Ex. No: 7

Build a simple application using Gradle

Date :

AIM:

The aim of this lab is to create a simple Java application and use Gradle to build and manage dependencies.


ALGORITHM:

1. Create a New Java Project:

- Create a new directory for your project and navigate into it.

```
bash

mkdir SimpleJavaApp
cd SimpleJavaApp
```


 Copy code

2. Create Project Structure:

- Create the basic project structure with a source directory.

```
bash

mkdir -p src/main/java
```


 Copy code

3. Create Java Class:

- Create a simple Java class. Save it as `SimpleApp.java` in `src/main/java`.

```
java

public class SimpleApp {
    public static void main(String[] args) {
        System.out.println("Hello, Gradle!");
    }
}
```

 Copy code

4. Create Gradle Build Script:

- Create a file named `build.gradle` in the project root.

```
groovy                                                                    Copy code

plugins {
    id 'java'
}

repositories {
    mavenCentral()
}

dependencies {
    // Define your dependencies here
}

application {
    mainClassName = 'SimpleApp'
}
```

You can customize the `dependencies` block based on the libraries or frameworks you want to use.

5. Build the Project:

- Open a terminal in the project directory.

```
bash                                                                    Copy code

gradle build
```

Gradle will download dependencies, compile the code, and build the project. You should see a `build` directory created with the compiled classes and JAR file.

6. Run the Application:

- Run the application using the following command:

```
bash                                                                    Copy code

java -jar build/libs/SimpleJavaApp.jar
```

You should see the "Hello, Gradle!" message printed to the console.

OUTPUT:

The output in the terminal as Gradle downloads dependencies, compiles the code, and builds the JAR file.

RESULT:

Upon successful execution, a simple Java application built with Gradle. This project can be extended by adding more classes, dependencies, and configurations to meet the requirements of application.

Ex. No: 8	Install Ansible and configure ansible roles and to write
Date :	playbooks

AIM:

The aim of this lab is to install Ansible, configure Ansible roles, and write playbooks for managing configurations on target machines.


ALGORITHM:

1. Install Ansible:

- On your control machine (the machine from which you will run Ansible), install Ansible. The installation steps vary based on your operating system.


For example, on Ubuntu:

```
bash
sudo apt update
sudo apt install ansible
```

 Copy code

On CentOS:

```
bash
sudo yum install ansible
```


 Copy code

Configure Ansible Roles:

1. Create Ansible Roles Directory:

- Create a directory structure for your Ansible roles.

```
bash
mkdir -p ~/ansible-roles
```

 Copy code

2. Create Roles:

- Inside the `~/ansible-roles` directory, create subdirectories for each role.

```
bash
cd ~/ansible-roles
mkdir common apache mysql
```

3. Organize Role Structure:

- Inside each role directory, create directories for tasks, handlers, defaults, and files.

```
bash
cd common
mkdir tasks handlers defaults files
```

4. Write Tasks and Handlers:

- Write tasks in the `tasks/main.yml` file for each role and corresponding handlers in the `handlers/main.yml` file.

Example (common/tasks/main.yml):

```
yaml
---
- name: Update package cache
  apt:
    update_cache: yes
  become: true
```

Example (common/handlers/main.yml):


```
yaml
---
- name: Restart services
  service:
    name: "{{ item }}"
    state: restarted
  loop:
    - apache2
    - mysql
```

Write Playbooks:

1. **Create Ansible Playbooks Directory:**

- Create a directory for your Ansible playbooks.

```
bash
mkdir ~/ansible-playbooks
```


 Copy code

2. **Write Playbooks:**

- Inside the `~/ansible-playbooks` directory, create YAML files for your playbooks.


Example (~/ansible-playbooks/setup-web-server.yml):

```
yml
---
- hosts: web_servers
  become: true
  roles:
    - common
    - apache
```

 Copy code

Example (~/ansible-playbooks/setup-db-server.yml):


```
yml
---
- hosts: db_servers
  become: true
  roles:
    - common
    - mysql
```

 Copy code

3. **Run Playbooks:**

- Run the playbooks using the `ansible-playbook` command.

```
bash
ansible-playbook -i inventory.ini ~/ansible-playbooks/setup-web-server.yml
ansible-playbook -i inventory.ini ~/ansible-playbooks/setup-db-server.yml
```

 Copy code

OUTPUT:

The output in the terminal as Ansible executes tasks defined in the playbooks and roles.

RESULT:

Upon successful execution of the playbooks, target machines will be configured based on the roles and tasks defined. Ansible will manage configurations, ensuring consistency across infrastructure.