
◆ **What is Jenkins?**

- **Jenkins is a CI/CD tool** that automates software development workflows.
- It allows developers to **automate build, test, and deployment** processes.
- Jenkins integrates with **Git, Docker, Kubernetes, AWS, Maven, Gradle**, and many other tools.
- It is **written in Java** and has **a web-based interface** for easy configuration.
- Jenkins supports **plugins**, making it highly customizable.

✅ **Use Cases of Jenkins:**
✓ Automating software builds
✓ Running tests and generating reports
✓ Deploying applications automatically
✓ Monitoring and integrating code changes

◆ **Installing Jenkins on Local or Cloud Environment**

1. Install Jenkins on Windows (Local Environment)

Step 1: Download Jenkins

1. Go to **Jenkins official website**:
   🔗 https://www.jenkins.io/download/
2. Download the **Windows Installer** (jenkins.msi).

Step 2: Install Java (if not installed)

Jenkins requires Java. Install **Java 17 or Java 21** before proceeding:

- Download Java from: https://adoptium.net/
- Install it and set the **JAVA_HOME** environment variable.

Step 3: Install Jenkins

1. Run jenkins.msi and follow the installation steps.
2. Select the **default Windows service** installation.
3. Jenkins will install and start the service automatically.

Step 4: Open Jenkins Web Interface

1. Open a browser and go to:

http://localhost:8080

2. Enter the **Administrator Password** (found in C:\Program Files\Jenkins\secrets\initialAdminPassword).
3. Follow the setup wizard to complete the installation.

✅ **Jenkins is now installed on Windows!**

****

Continuous Integration with Jenkins 🚀

◆ **What is Continuous Integration (CI)?**

Continuous Integration (CI) is a **software development practice** where developers frequently **merge their code** into a shared repository. Each integration triggers an **automated build and test** process, ensuring early detection of errors.

✅ **Benefits of CI with Jenkins:**

✓Detects bugs early in development.
✓Automates build and testing processes.
✓Reduces integration issues.
✓Improves software quality and reliability.

◆ **Setting Up a CI Pipeline in Jenkins**

1 Install Required Plugins

Before setting up the pipeline, ensure Jenkins has the necessary plugins installed:

1. Go to **Jenkins Dashboard** → **Manage Jenkins** → **Manage Plugins**
2. Install the following:
    - **Maven Integration Plugin**
    - **Git Plugin**
    - **Pipeline Plugin**

---

2 Install Maven on Jenkins Server

◆ **On Windows:**

1. Download Apache Maven from https://maven.apache.org/download.cgi.
2. Extract and set the MAVEN_HOME environment variable.
3. Add the bin folder to the system PATH.

3 Configure Maven in Jenkins

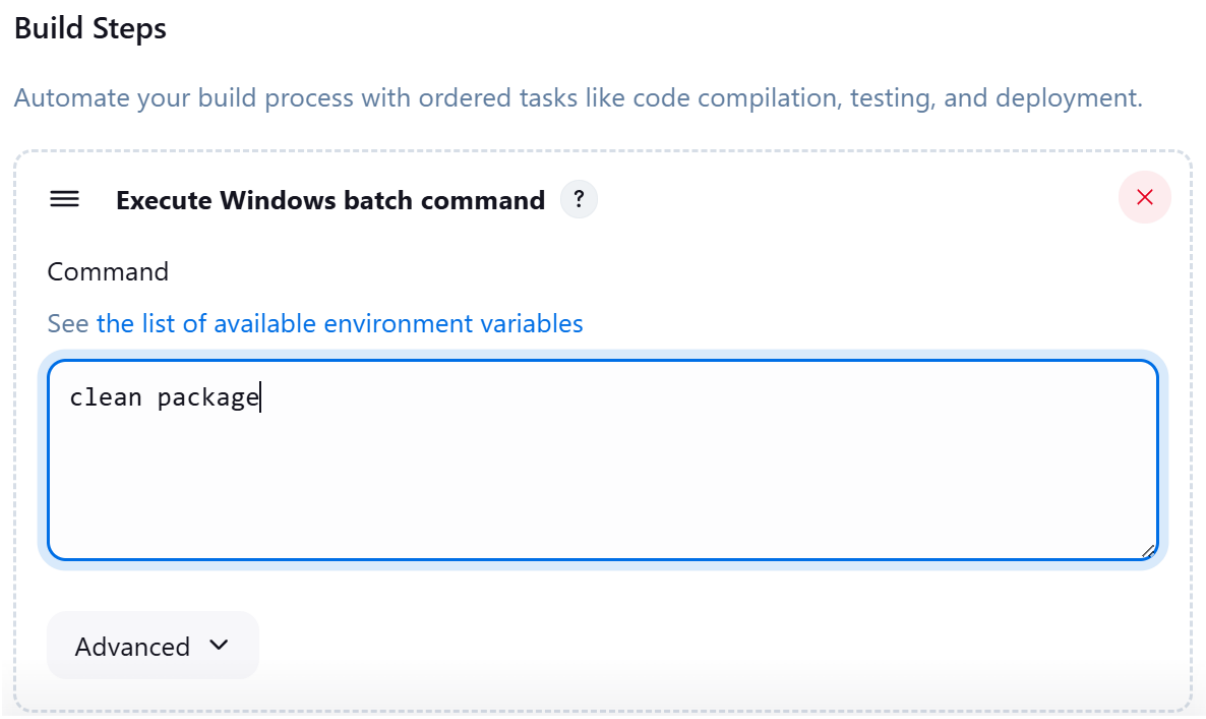
1. Go to **Jenkins Dashboard** → **Manage Jenkins** → **Global Tool Configuration**.
2. Under **Maven**, click **Add Maven**:
   - **Name:** Maven-3.8.6 (or any version installed)
   - **Install Automatically:** Unchecked (if installed manually)
   - **MAVEN_HOME:** Provide the installation path

---

4 Create a New Jenkins Job (Freestyle or Pipeline)

◆ **Option 1: Using Freestyle Project**

1. **Go to Jenkins Dashboard** → Click **New Item** → Select **Freestyle Project**.
2. **Source Code Management**:
   - Select **Git**.
   - Enter repository URL (e.g., https://github.com/user/maven-project.git).
3. **Build Triggers**:
   - Select **Poll SCM** (H/5 * * * * for every 5 minutes).
4. **Build Steps**:
   - Click **Add Build Step** → Select **Invoke top-level Maven targets**.
   - Enter **Goals**: clean package



   -
5. Click **Save** and **Build Now** to test the pipeline.

---

5 Option 2: Using a Jenkins Pipeline (Recommended)

A **Jenkins Pipeline** automates builds using a script.

Step 1: Create a Jenkinsfile

Inside your **Maven project repository**, create a Jenkinsfile:

```
pipeline {
  agent any

  tools {
    maven 'Maven-3.8.6' // Use the Maven tool configured in Jenkins
  }

  stages {
    stage('Checkout') {
      steps {
        git 'https://github.com/user/maven-project.git'
      }
    }

    stage('Build') {
      steps {
        bat 'mvn clean package'
      }
    }

    stage('Test') {
      steps {
        bat  'mvn test'
      }
    }
```

```
        stage('Deploy') {
            steps {
                echo 'Deploying application...'
            }
        }
    }

    post {
        success {
            echo 'Build and Test Passed! 🎉'
        }
        failure {
            echo 'Build Failed! 🚨'
        }
    }
}
```

Step 2: Create a New Pipeline Job in Jenkins

1. Go to **Jenkins Dashboard** → Click **New Item** → Select **Pipeline**.
2. **Pipeline Definition**:
    - Select **Pipeline Script from SCM**.
    - Choose **Git** and provide the **repository URL**.
    - Set the **Script Path** to Jenkinsfile.
3. Click **Save** and **Build Now** to start the pipeline.

◆ **Running Automated Builds and Tests**

Once Jenkins is set up:
✅ **Each code commit will trigger a build**.
✅ **Maven will compile and package the project**.
✅ **Unit tests will run automatically**.
✅ **Build status (Success/Failure) will be reported**.

****

Configuration Management with Ansible on Windows 🖥️ □

◆ What is Ansible?

Ansible is an **open-source IT automation tool** used for **configuration management, application deployment, and task automation**. Unlike other tools, Ansible is **agentless**, meaning it does not require installing software on target machines.

✅ Key Features of Ansible:
✓ Simple YAML-based automation
✓ Agentless architecture (only requires SSH or WinRM)
✓ Idempotency (ensures tasks don't run if already completed)
✓ Scalability (manages multiple machines easily)

📌 For this tutorial, we will:

- **Install Ansible on Windows**
- **Understand Inventory, Playbooks, and Modules**
- **Automate server configuration with Playbooks**

◆ Step 1: Installing Ansible on Windows

Ansible runs **natively on Linux** but can be installed on **Windows using WSL (Windows Subsystem for Linux)** or via a **Control Node (Linux VM/Cloud Instance)**.

Option 1: Install Ansible via WSL (Windows Subsystem for Linux) 🏆 (Recommended)

1 **Enable WSL on Windows**

- Open **PowerShell as Administrator** and run:

```
wsl --install
```

- Restart your computer.

2 **Install Ubuntu from Microsoft Store**

- Open **Microsoft Store** → Search for **Ubuntu 22.04 LTS** → Click **Install**
- Launch Ubuntu and create a new user.

3 **Update the system and install Ansible**

- Inside Ubuntu, run:

```
sudo apt update
sudo apt install -y ansible
```

- Verify the installation:

```
ansible --version
```

Option 2: Install Ansible on a Linux VM (Alternative)

If you **don't want to use WSL**, install Ansible on a **Linux VM** (e.g., Ubuntu on VirtualBox or AWS). Follow the same installation steps as above.

◆ **Step 2: Understanding Ansible Basics**

1 Ansible Inventory (Managing Hosts)

An **inventory file** contains the list of target machines (Windows Servers) that Ansible will manage.

📌 **Create an inventory file** (hosts.ini):

```
[windows]

192.168.1.100


[windows:vars]

ansible_user=Administrator

ansible_password=YourPasswordHere

ansible_connection=winrm

ansible_winrm_transport=basic
```

- **192.168.1.100** → Replace with your **Windows machine's IP**
- **winrm** → Windows Remote Management (**Ensure it's enabled**)

✅ **Enable WinRM on Windows (Target Machine)**
Run these commands in **PowerShell (Admin)**:

```
winrm quickconfig

winrm set winrm/config/service '@{AllowUnencrypted="true"}'

winrm set winrm/config/service/auth '@{Basic="true"}'
```

This allows Ansible to connect to Windows.

---

2 Ansible Modules (Commands & Tasks)

Modules are small **automation scripts** that Ansible runs on target machines.

📌 **Example: Run a command on Windows Machine**

```
ansible windows -i hosts.ini -m win_ping
```

💡 **Expected Output:**

```
192.168.1.100 | SUCCESS => {
```

```
    "changed": false,
    "ping": "pong"
}
```

✅ If you see this, Ansible can communicate with Windows!

◆ Step 3: Writing and Running Ansible Playbooks

A **playbook** is a **YAML file** that contains a list of tasks to automate configurations.

📌 Example 1: Creating a Basic Playbook to Install Software

Create a file **install_choco.yml**

```
- name: Install Chocolatey on Windows
  hosts: windows
  tasks:
    - name: Ensure Chocolatey is installed
      win_chocolatey:
        name: chocolatey
        state: present

    - name: Install Notepad++
      win_chocolatey:
        name: notepadplusplus
        state: present
```

💡 **What this playbook does?**

1. Ensures **Chocolatey (Windows Package Manager)** is installed.
2. Installs **Notepad++** using Chocolatey.

Run the Playbook:

```
ansible-playbook -i hosts.ini install_choco.yml
```

✅ **Expected Output:**
✓Chocolatey Installed
✓Notepad++ Installed

📌 Example 2: Configuring Firewall and Creating Users

Create a new file **setup_windows.yml**

```
- name: Configure Windows Server
  hosts: windows
  tasks:
    - name: Open Firewall Port 80
      win_firewall_rule:
        name: "Allow HTTP"
        localport: 80
        action: allow
        direction: in
        protocol: TCP
        state: present

    - name: Create a new user
      win_user:
        name: DevUser
        password: SecurePass123
        groups: Administrators
        state: present
```

Run the Playbook:

```
ansible-playbook -i hosts.ini setup_windows.yml
```

✅ **Expected Output:**
✓ **Firewall Rule Added (Port 80 open for HTTP)**
✓ **User "DevUser" created with Admin access**

📌 **Example 3: Deploying IIS Web Server**

Create a file **deploy_iis.yml**

```
- name: Install IIS Web Server on Windows
  hosts: windows
  tasks:
    - name: Install IIS
```

```
    win_feature:
      name: Web-Server
      state: present

  - name: Start IIS Service
    win_service:
      name: W3SVC
      start_mode: auto
      state: started

  - name: Create a homepage
    win_copy:
      content: "<h1>Welcome to Ansible Managed IIS Server</h1>"
      dest: C:\inetpub\wwwroot\index.html
```

Run the Playbook:

```
ansible-playbook -i hosts.ini deploy_iis.yml
```

✅ **Expected Output:**
✓ **IIS Installed**
✓ **IIS Service Started**
✓ **Custom Homepage Created (index.html)**

💡 **Verify IIS is Running:**
Open a browser and go to http://192.168.1.100 🎉

◆ **Step 4: Best Practices for Ansible on Windows**

1 **Use Groups in Inventory**:

```
[webservers]
192.168.1.101
192.168.1.102

[dbservers]
192.168.1.103
```

- This helps manage **multiple servers** easily.

2 **Use Variables in Playbooks**:

```
vars:
  username: DevUser
  password: SecurePass123
```

3 **Enable Logging** for Troubleshooting:

```
export ANSIBLE_LOG_PATH=ansible.log
ansible-playbook -i hosts.ini setup_windows.yml -vvv
```

◆ **Summary**

🚀 **You have successfully automated Windows configuration with Ansible!**
✅ Installed Ansible on Windows (WSL/Linux VM)
✅ Created Inventory for Windows Servers
✅ Used Modules to run commands
✅ Wrote Playbooks to Install Software, Configure Firewall, Create Users, and Deploy IIS

****

🔥 **Practical Exercise: Setting Up a Jenkins CI Pipeline for a Maven Project & Deploying with Ansible (Windows Only)** 🚀

💡 **Objective**

- **Set up Jenkins** on Windows for CI/CD.
- **Create a Maven-based Java project** in Jenkins.
- **Configure a Jenkins Pipeline** for Continuous Integration.
- **Use Ansible** to deploy the built artifacts from Jenkins.

◆ **Step 1: Install & Configure Jenkins on Windows**

📌 **1. Install Jenkins**

1**Download Jenkins**

- Go to **Jenkins Official Site**
- Download **Windows MSI Installer (Jenkins LTS)**

2 **Install Jenkins**

- Run the .msi file and follow the setup wizard.

- During installation, select **Install as a Windows Service**
- Note down the **admin password** from:

C:\Program Files\Jenkins\secrets\initialAdminPassword

3 **Start Jenkins & Unlock**

- Open **http://localhost:8080** in your browser.
- Enter the **admin password** from the above step.
- Install **suggested plugins**.

✅ **Jenkins is installed and ready!** 🎉

◆ Step 2: Set Up a Jenkins CI Pipeline for a Maven Project

📌 1. Install Java & Maven in Jenkins

1**Download & Install Java 17 or 21**

- Get Java from: [Oracle JDK]
- Set **JAVA_HOME** in System Properties > Environment Variables
- Verify installation:

```
java -version
```

2 **Install Maven**

- Download **Maven** from: [Maven Apache Site]
- Extract it & set **MAVEN_HOME** in Environment Variables
- Verify installation:

```
mvn -version
```

📌 2. Create a New Jenkins Job

1Open **Jenkins Dashboard** → Click **New Item**
2 Select **Freestyle Project** → Name it **Maven-CI-Pipeline**
3 Under **Source Code Management**, select **Git**

- Enter **Repository URL** (e.g., GitHub)
  4 Under **Build Environment**, check **Delete Workspace Before Build**

📌 3. Configure Build Steps

1 Click **Add Build Step** → Select **Invoke top-level Maven targets**
2 **Enter Goals**:

```
clean package
```

3 Under **Post-Build Actions**, add **"Publish JUnit test report"**

- Report Path:

target/surefire-reports/*.xml

4 Click **Save & Build Now** ✅

💡 **Jenkins will now pull code, compile using Maven, run tests, and generate a target/*.jar artifact.**

◆ **Step 3: Deploy Artifacts Using Ansible**

📌 **1. Configure Ansible for Windows**

1 Install **Ansible** via WSL (see previous guide)
2 Enable **WinRM** on the Windows target machine:

```
winrm quickconfig

winrm set winrm/config/service '@{AllowUnencrypted="true"}'

winrm set winrm/config/service/auth '@{Basic="true"}'
```

📌 **2. Define Ansible Inventory**

Create a file hosts.ini to define the target Windows machine:

```
 [windows]

192.168.1.100


[windows:vars]

ansible_user=Administrator

ansible_password=YourPasswordHere

ansible_connection=winrm

ansible_winrm_transport=basic
```

📌 **3. Create Ansible Playbook for Deployment**

Create deploy.yml to **copy & deploy the artifact**:

```
- name: Deploy Maven Artifact to Windows Server

  hosts: windows

  tasks:

    - name: Copy JAR file from Jenkins

      win_copy:

        src: C:\ProgramData\Jenkins\.jenkins\workspace\Maven-CI-Pipeline\target\myapp.jar

        dest: C:\Deployments\myapp.jar
```

```
  - name: Ensure Java App is Running
    win_shell: java -jar C:\Deployments\myapp.jar
```

📌 **4. Run the Playbook**

```
ansible-playbook -i hosts.ini deploy.yml
```

✅ **Expected Output:**
✔ JAR file is copied to **Windows Server**
✔ Application is started

◆ **Step 4: Automate Deployment in Jenkins**

📌 **1. Add Post-Build Deployment Step**

- Go to **Jenkins Job** → Click **Configure**
- Under **Post-Build Actions** → Add **"Execute Shell"**
- Enter the command:

```
ansible-playbook -i C:\ansible\hosts.ini C:\ansible\deploy.yml
```

- Click **Save & Build Now** ✅

💡 **Jenkins will now automatically deploy the artifact using Ansible after a successful build!** 🎉

◆ **Summary**

✅ Installed **Jenkins & Maven** on Windows
✅ Created **a CI pipeline** for a Maven project
✅ Used **Ansible to deploy the JAR artifact**
✅ Automated **deployment in Jenkins**

****

🚀 **Introduction to Azure DevOps: Overview & Setup Guide**

💡 **What is Azure DevOps?**

Azure DevOps is a cloud-based DevOps platform by Microsoft that provides tools for software development, collaboration, automation, and deployment. It includes:

◆ **Key Azure DevOps Services**

Service	Description
Azure Repos	Source control (Git)
Azure Pipelines	CI/CD automation
Azure Boards	Agile project tracking
Azure Artifacts	Package management
Azure Test Plans	Test automation

◆ Step 1: Setting Up an Azure DevOps Account

📌 1. Create a Free Azure DevOps Account

1Go to [Azure DevOps]
2 Click **"Start free"** (Requires a Microsoft account)
3 Sign in with your **Microsoft Account**
4 Click **"Continue"** to access the Azure DevOps portal

✅ **Azure DevOps account is now created!** 🎉

◆ Step 2: Create a New Azure DevOps Project

📌 1. Create a Project

1 Click **"New Project"**
2 Enter:

- **Project Name** (e.g., MyFirstProject)
- **Description** (Optional)
- **Visibility**:
  - *Public*: Open for everyone
  - *Private*: Restricted to team members
    3Click **Create**

📌 2. Configure Version Control

Git (Recommended) → Distributed Version Control

- **TFVC** → Centralized Version Control

✅ **Project is now set up and ready!** 🎉

🚀 Creating Build Pipelines in Azure DevOps for Maven/Gradle Projects

💡 Objective

- **Set up a Build Pipeline** in Azure DevOps for a **Maven/Gradle** project.
- **Integrate Code Repository** (GitHub/Azure Repos).
- **Run Unit Tests** and **Generate Reports** automatically.

◆ Step 1: Prerequisites

✅ **Azure DevOps Account & Project** (If not created, follow this guide)
✅ **GitHub/Azure Repos Repository** with a Maven/Gradle project
✅ **Azure DevOps Agent** (for self-hosted runners, optional)

◆ Step 2: Create an Azure Build Pipeline for Maven/Gradle

📌 1. Navigate to Pipelines in Azure DevOps

1 Open **Azure DevOps Portal** → Select your project
2 Click on **Pipelines** → **New Pipeline**

📌 2. Select Your Source Repository

- Choose **GitHub** or **Azure Repos**
- Authenticate and select your **Maven/Gradle project repository**

📌 3. Choose a Pipeline Configuration

- Select **"Starter pipeline"**
- Replace azure-pipelines.yml content with the relevant configuration:

◆ Step 3: Define the Build Pipeline (YAML)

📌 1. Maven Build Pipeline

```
trigger:
- main  # Triggers the pipeline on changes to the main branch

pool:
  vmImage: 'ubuntu-latest'  # Use a Microsoft-hosted agent
```

```
steps:
- task: Maven@3
  inputs:
    mavenPomFile: 'pom.xml'  # Path to Maven's pom.xml
    goals: 'clean package'
    javaHomeOption: 'JDKVersion'
    jdkVersionOption: '1.17'
    publishJUnitResults: true
    testResultsFiles: '**/target/surefire-reports/*.xml'
```

📌 **2. Gradle Build Pipeline**

```
trigger:
- main

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: Gradle@2
  inputs:
    gradleWrapperFile: 'gradlew'
    tasks: 'clean build'
    publishJUnitResults: true
    testResultsFiles: '**/build/test-results/test/*.xml'
```

◆ **Step 4: Save & Run the Pipeline**

1Click **Save & Run** → This triggers the build.
2 Monitor logs under **Pipelines → Runs**.
3 If successful, artifacts (JAR/WAR) will be generated in **target/** (Maven) or **build/libs/** (Gradle).

◆ **Step 5: Running Unit Tests & Generating Reports**

✅ Azure DevOps will automatically **run unit tests** using surefire-reports/*.xml (Maven) or test-results/*.xml (Gradle).

✅ View test results:

- Go to **Pipelines > Your Pipeline > Runs**
- Click **Tests** to see **passed/failed** test cases.

✅ Generate reports using **JUnit Report Publisher** (already configured in YAML).

🚀 **Creating Release Pipelines in Azure DevOps**

📌 **Goal**

✅ Deploy a **Maven/Gradle Application** to **Azure App Services**
✅ Manage **Secrets & Configurations** using **Azure Key Vault**
✅ Implement **Continuous Deployment (CD)** with **Azure Pipelines**

🔹 **Step 1: Prerequisites**

🔸 **Azure DevOps Project** (Created in previous steps)
🔸 **Build Pipeline** (Already set up for Maven/Gradle)
🔸 **Azure App Services** (Web App running on Azure)
🔸 **Azure Key Vault** (For managing secrets securely)

🔹 **Step 2: Create an Azure Release Pipeline**

📌 **1. Navigate to Releases in Azure DevOps**

1Open **Azure DevOps Portal** → Select your project
2 Go to **Pipelines → Releases**
3 Click **"New Release Pipeline"**
4 Select **"Empty Job"**

📌 **2. Add a Build Artifact**

1 Click **Add an Artifact**
2 Select **"Build"** → Choose the build pipeline from the dropdown
3 Click **"Add"**

🔹 **Step 3: Deploy to Azure App Services**

📌 **1. Add a Deployment Stage**

1 Click **Stage 1 → Rename to "Deploy to Azure"**
2 Click **"Add Task"** → Search for **Azure App Service Deploy**
3 Configure the task:

- Select **Azure Subscription**
- Select **App Service Name**
- Choose **Package or Folder** → $(System.DefaultWorkingDirectory)/_your-build-pipeline/drop
  4Click **Save**

◆ **Step 4: Manage Secrets with Azure Key Vault**

📌 **1. Link Azure Key Vault to the Pipeline**

1 Go to **Pipelines → Library**
2 Click **"Add" → "Azure Key Vault"**
3 Select your **Azure Subscription & Key Vault**
4 Click **"Add"** to link it

✅ **Secrets (DB passwords, API keys) are now securely managed!**

◆ **Step 5: Enable Continuous Deployment (CD)**

1 Open **Releases → Edit the Release Pipeline**
2 Click on **Artifacts → Enable Continuous Deployment**
3 Click **Save & Trigger Release**

🚀 **Now, every successful build triggers an automatic deployment!**

****

🚀 **Practical Exercise: Build and Deploy a Complete DevOps Pipeline**

📌 **Objective**

In this exercise, we will:
✅ **Build a CI/CD Pipeline** using Azure DevOps
✅ **Integrate Maven/Gradle, Jenkins, Ansible & Azure Pipelines**
✅ **Deploy an Application to Azure App Services**
✅ **Use Azure Key Vault for Secrets Management**

◆ **Step 1: Setup the Environment**

✅ **Tools Required**

🔹 **Azure DevOps Account**
🔹 **Jenkins Installed on a Windows Machine**

🔹 **Ansible Installed on Windows (via WSL/Windows Subsystem for Linux)**
🔹 **Maven/Gradle Installed on Windows**
🔹 **Docker Installed (Optional for Containerized Deployments)**

◆ Step 2: Build & Test the Application in Jenkins

📌 1. Configure Jenkins for CI

1Open **Jenkins Dashboard** → Click **New Item**
2 Select **Maven/Gradle Project** → Click **OK**
3 Configure **Source Code Management (GitHub/Azure Repos)**
4 Add a **Build Step** → Choose **Invoke top-level Maven targets**
5 Set **Goals:** clean package
6 Click **Post-Build Actions** → Add **Publish JUnit test result report**
7 Save & Run **Build**

✅ **Output:** A packaged JAR/WAR file in target/

◆ Step 3: Deploy Artifacts Using Ansible

📌 1. Create an Ansible Playbook for Deployment

📌 **File:** deploy.yml

yaml

CopyEdit

```
---

- name: Deploy Application

  hosts: webserver

  tasks:

    - name: Copy JAR to Server

      win_copy:

        src: C:\Jenkins\workspace\myapp\target\myapp.jar

        dest: C:\deploy\myapp.jar


    - name: Restart Application

      win_service:

        name: MyAppService

        state: restarted
```

📌 **2. Run the Playbook on Windows**

ansible-playbook -i inventory deploy.yml

✅ **Output:** Application is copied and restarted

◆ **Step 4: Set Up a Release Pipeline in Azure DevOps**

📌 **1. Configure Release Pipeline**

1 Navigate to **Pipelines → Releases**
2 Create a **New Release Pipeline** → Select **Azure App Service**
3 Link the **Jenkins Artifact** as a Source
4 Add a Deployment Task: **Azure App Service Deploy**
5 Select **Subscription & App Service**
6 Click **Save & Deploy**

✅ **Output:** Application is deployed to Azure!

◆ **Step 5: Manage Secrets with Azure Key Vault**

📌 **1. Store Secrets in Azure Key Vault**

```
az keyvault create --name myVault --resource-group myResourceGroup

az keyvault secret set --vault-name myVault --name "DB_PASSWORD" --value "mysecurepassword"
```

📌 **2. Fetch Secrets in Azure Pipelines**

```
steps:

- task: AzureKeyVault@2

  inputs:

    azureSubscription: 'MyServiceConnection'

    KeyVaultName: 'myVault'

    SecretsFilter: '*'
```

✅ **Output:** Securely fetches passwords during deployment

◆ **Discussion on Best Practices**

✅ **Use Infrastructure as Code (IaC)** (Terraform, Ansible)
✅ **Automate Testing** (Unit, Integration, Security Tests)
✅ **Implement Security Controls** (Azure Key Vault, IAM Policies)

✅ **Use Blue-Green Deployments** for Zero-Downtime Releases
✅ **Monitor Pipelines with Azure Monitor & Logs**