DevOps Foundations: Version Control and CI/CD with Jenkins

Jenkins Pipeline

Learning Objectives

By the end of this lesson, you will be able to:

- Utilize Jenkins pipelines to improve software development and deployment workflows, achieving faster, automated builds and streamlined releases
- Identify the similarities and differences between scripted and declarative Jenkins pipelines for optimizing CI/CD processes
- Outline authentication options in Jenkins for secure access control and user management
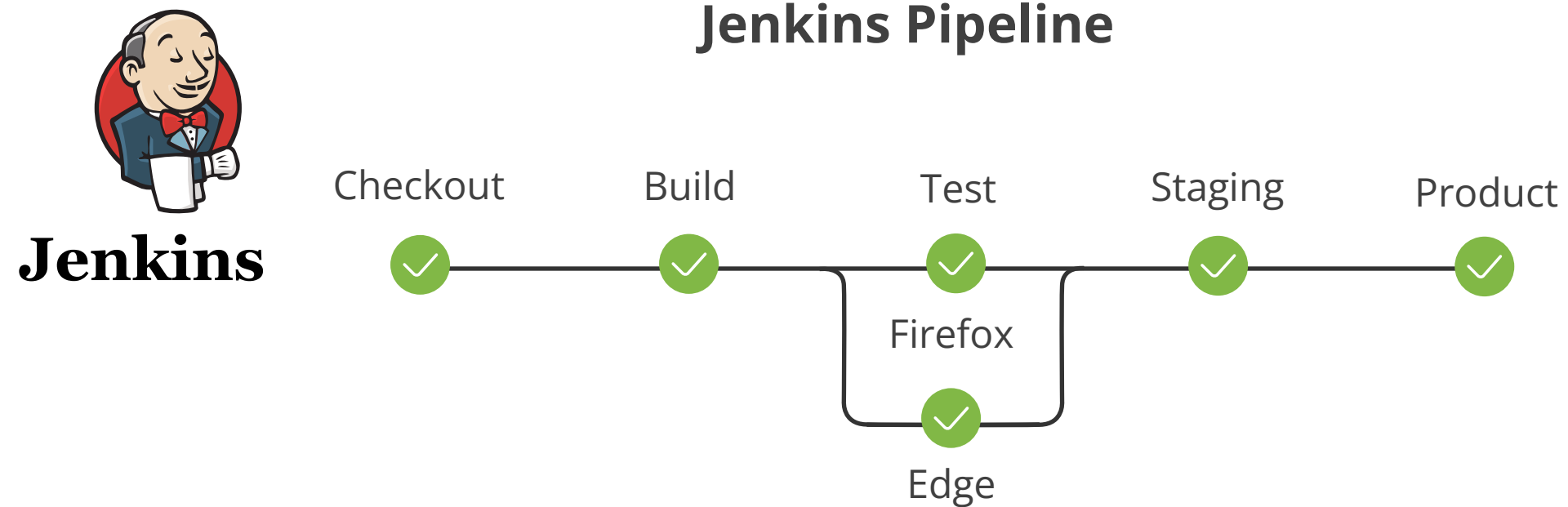
Getting Started with Jenkins Pipeline
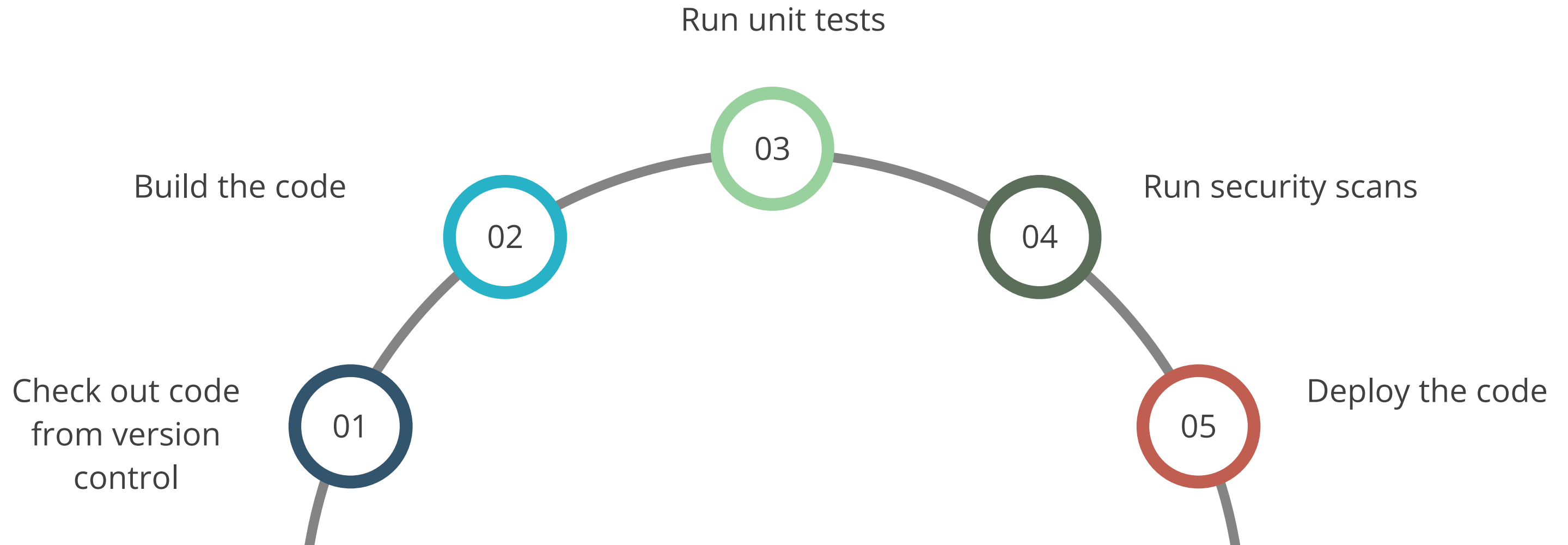
Introduction to Jenkins Pipeline

It is a suite of plugins that allows to define and automate continuous delivery workflows as code within a Jenkinsfile.

Jenkins Pipeline

Jenkins

Checkout — Build — Test — Staging — Product

Firefox

Edge

A Jenkins pipeline includes all the tools needed to orchestrate testing, merging, packaging, shipping, and deploying code.

Jenkins Pipeline: Stages

Below are the key stages of the Jenkins pipeline facilitating seamless CI/CD automation:

Run unit tests

03

Build the code

Run security scans

02

04

Check out code
from version
control

Deploy the code

01

05

Jenkins Pipeline: Stages

Check out code from version control	Retrieve the latest code from the version control system
Build the code	Compile the code to create executable files or artifacts
Run unit tests	Execute isolated tests to verify individual components or units of code
Run security scans	Conduct thorough assessments to identify and address security vulnerabilities in the codebase
Deploy the code	Release the built artifacts to the designated environment for use by end-users

Benefits of Using Jenkins Pipeline

Jenkins pipeline streamlines the software development and deployment to achieve faster builds, automated testing, and seamless delivery by providing the following advantages:

Automation — Automate the entire delivery process, reducing manual work and making it more efficient

Visibility — Provide clear visibility into the progress of the builds and deployment

Repeatability — Set up the delivery process with pipelines to ensure consistency and repeatability across deployments
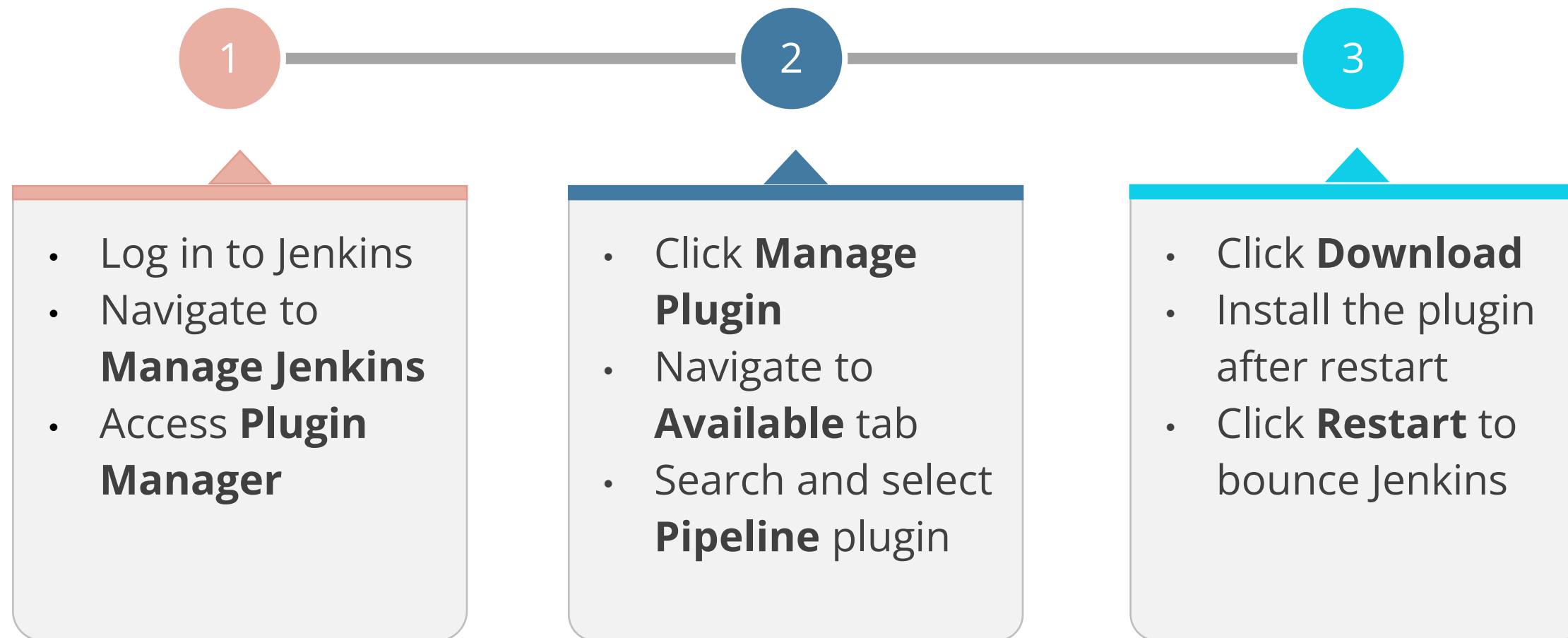
Version control — Track changes and rollback effortlessly by storing your pipeline definition in a Jenkins file

Freestyle Project vs. Jenkins Pipeline

Feature	Freestyle project	Jenkins pipeline
Configuration	Setup via a GUI with point-and-click options	Setup by coding in Jenkins pipeline DSL
Flexibility	Less flexible; limited to pre-defined steps	Highly flexible; custom steps and logic possible
Reusability	Limited reusability; needs to be recreated for reuse	Highly reusable; code can be shared across jobs
Complexity	Ideal for simple builds	Ideal for complex workflows and integrations
Learning curve	Easier to learn	Requires scripting knowledge

Jenkins Pipeline Job: Setup

Jenkins supports configuration of a pipeline job to create a CI/CD automation. To configure the pipeline jobs, developers must first download the pipeline plugin using the following steps:

1

- Log in to Jenkins
- Navigate to **Manage Jenkins**
- Access **Plugin Manager**

2

- Click **Manage Plugin**
- Navigate to **Available** tab
- Search and select **Pipeline** plugin

3

- Click **Download**
- Install the plugin after restart
- Click **Restart** to bounce Jenkins

Jenkins Pipeline Job: Setup

Below screenshot of the Jenkins dashboard lists all the available plugins:

Jenkins Pipeline Job: Build

Jenkins supports the creation of a pipeline build job after the plugins are successfully downloaded and installed.

Below are the steps to create a pipeline job:

Enter the name of the job;
click pipeline **Job type**

2

Log in to Jenkins;
navigate to **New item**

1

3

Configure the job created
and click **Save**
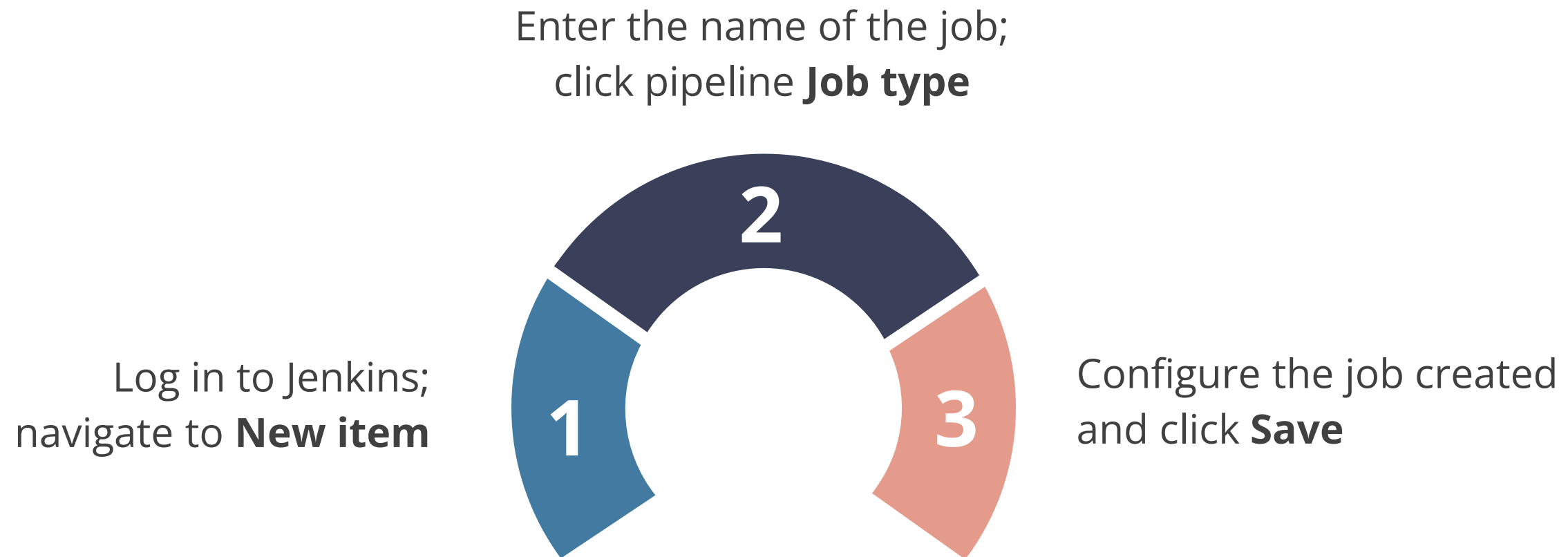
Automated Jenkins CI/CD Pipeline

The entire process from code to production is referred to as a CI/CD pipeline, which stands for continuous integration and continuous delivery.

A fully automated Jenkins CI/CD pipeline is shown below:

Code

Commit

Related code

Build

Unit tests

Integration tests

CI pipeline

Review

Staging

Production
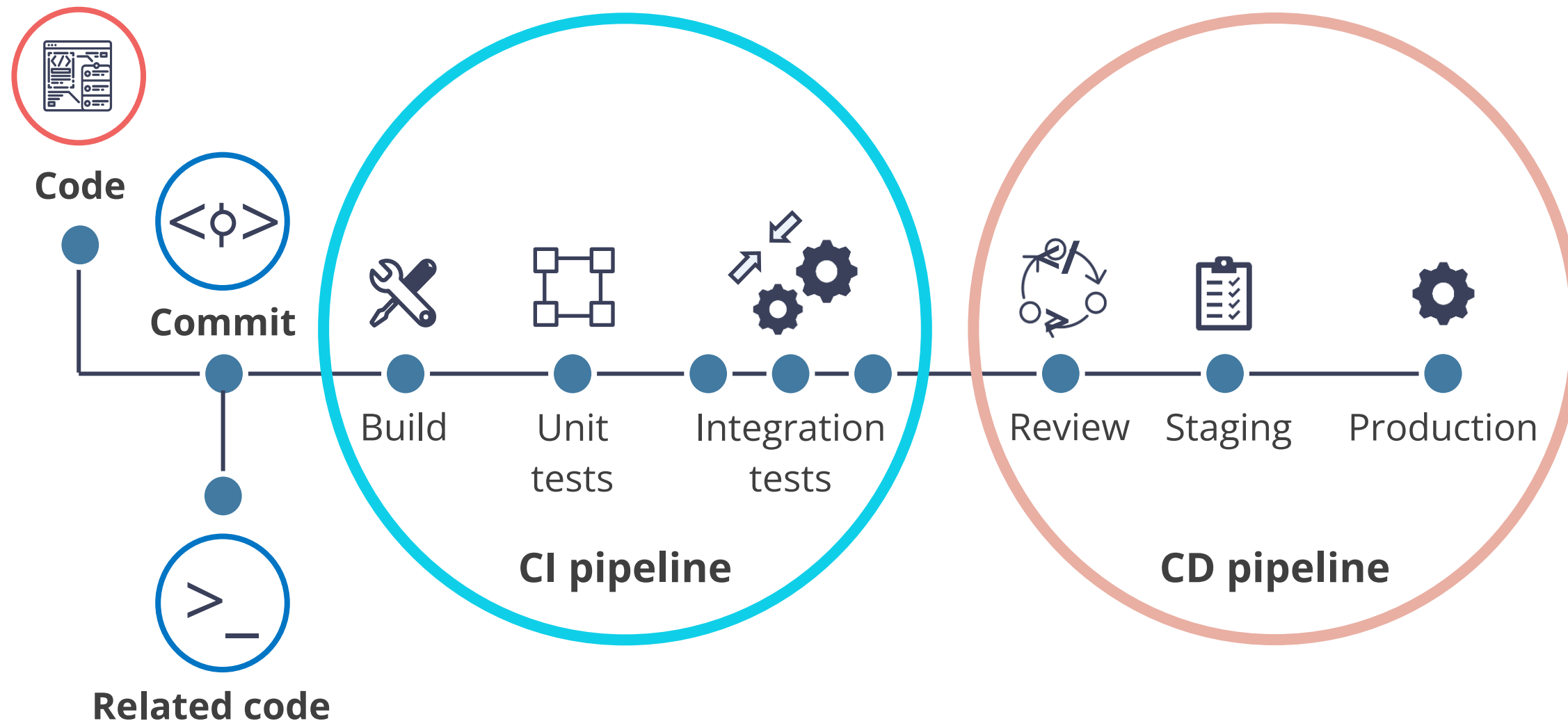
CD pipeline

CI/CD Pipeline: Phases

Below are the phases of CI/CD pipeline:

Code	Initiate the pipeline with the coding phase
Commit	Submit code to the version control system
Build	Compile code into a usable form
Unit tests	Test individual code units after building

CI/CD Pipeline: Phases

Below are the phases of CI/CD pipeline:

Integration tests	Validate overall functionality with integration testing
Review	Examine code for issues that automated tests may miss
Staging	Initiate with code review, approve, and deploy to staging for pre-production testing
Production	Deploy code to the live environment after successful staging

Quick Check

As an IT professional setting up continuous integration and continuous delivery (CI/CD), which is the most significant benefit of using Jenkins pipeline jobs over Freestyle jobs?

A. Support for static analysis and security checks

B. Integration with legacy tools and systems

C. More flexible and reusable job

D. Support for cloud-based deployments

Groovy Concepts: Scripting in CI/CD Pipeline

Apache Groovy

It is a domain-specific language (DSL), providing powerful scripting capabilities and an inline editor for streamlined pipeline creation and management.

DSL is a programming language dedicated to a particular domain, problem, or a solution technique.

Jenkins Interfaces and Tools

Below are the tools for Jenkins implementation that provide a concise definition for Jenkins interface :

Jenkins Classic UI	Offers Jenkins original web interface for job setup, security, and build history
Jenkins Blue Ocean Editor	Provides a modern Jenkins interface that simplifies and visualizes continuous delivery pipelines
Jenkins Console Interface (CLI)	Enables interaction with Jenkins jobs, nodes, and settings through a command-line tool
Jenkins Pipeline Script Generator	Simplifies generating Jenkinsfile syntax for steps, commands, and parameters within Jenkins

Blue Ocean Editor

It is a visual tool within Jenkins that simplifies creating and understanding CI/CD pipelines.
Below are the images of blue ocean editor interface:
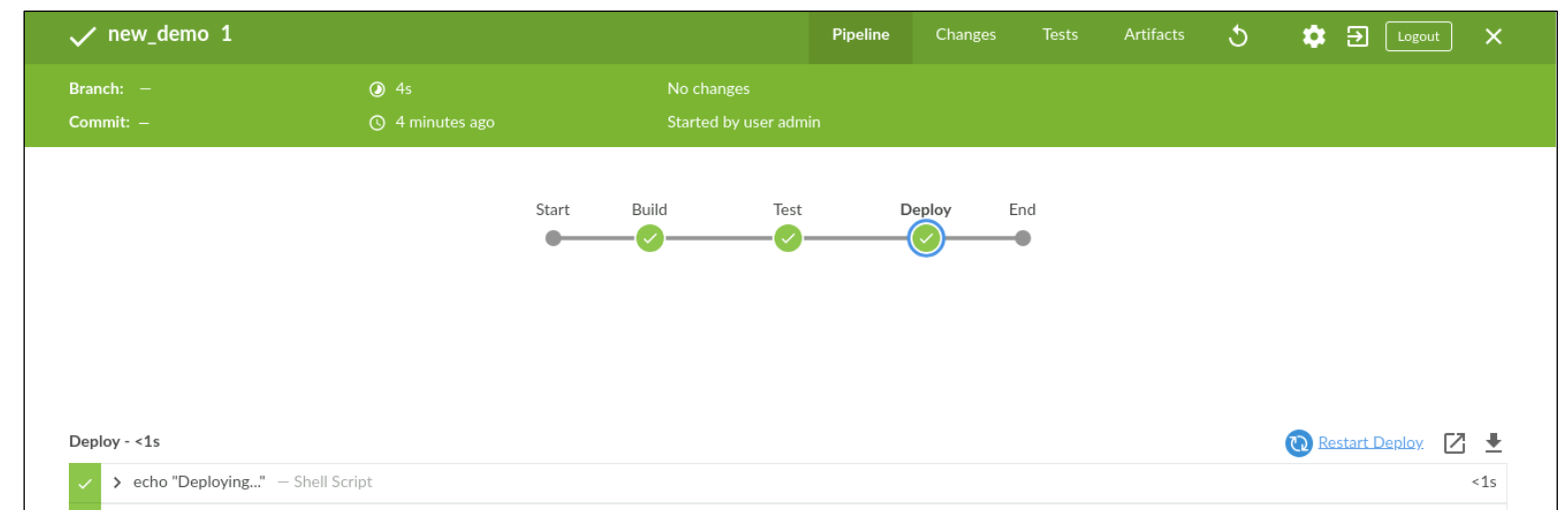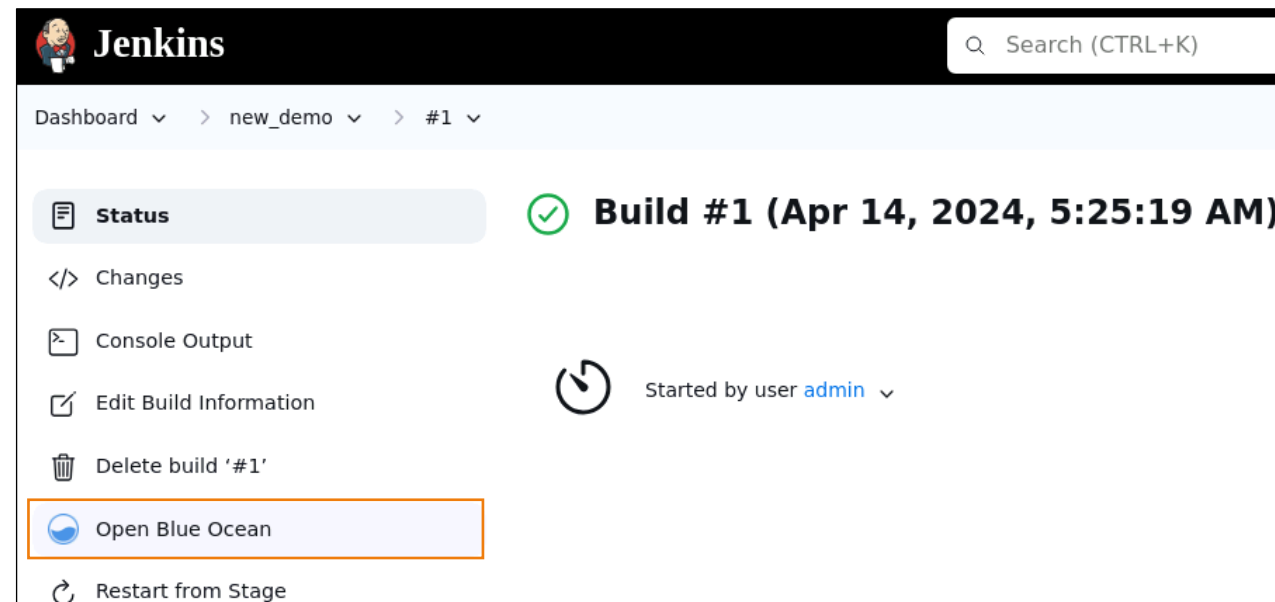
The Blue Ocean editor for Groovy scripting offers the easiest method for creating a Jenkins pipeline, facilitating a smooth transition for developers to this innovative approach.
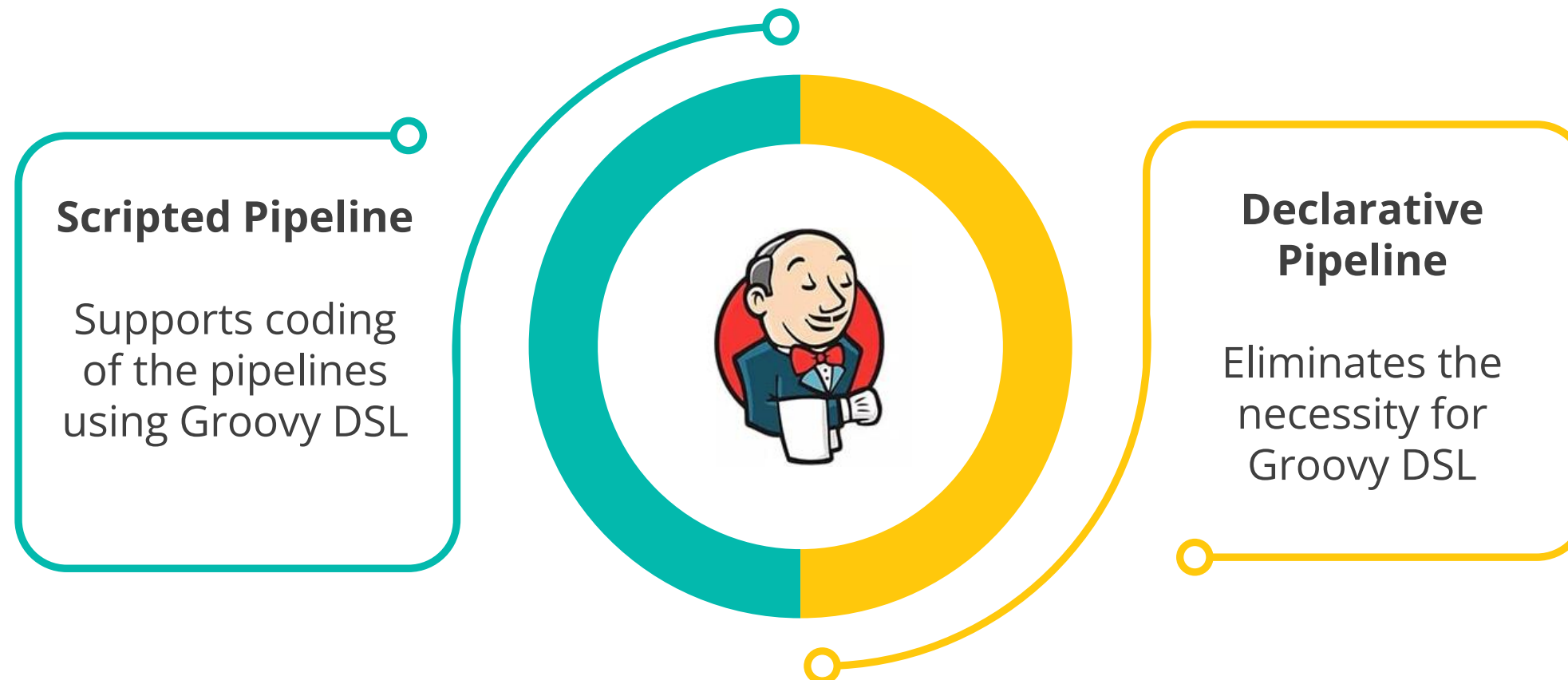
Blue Ocean Editor: Features

Following are the features and functionalities of the Blue Ocean editor in Jenkins, particularly when used for scripting with Groovy:

Provides an interface to add stages, steps, and commands necessary for the build process

Saves the pipeline contents to **Jenkinsfile** in the repository once the modifications are completed

Gets launched when the **New Pipeline** button is clicked, within the Jenkins interface

Jenkins Pipeline: Syntax Types

The Jenkins pipeline execution supports two types of syntax, namely Scripted Pipeline and Declarative Pipeline.
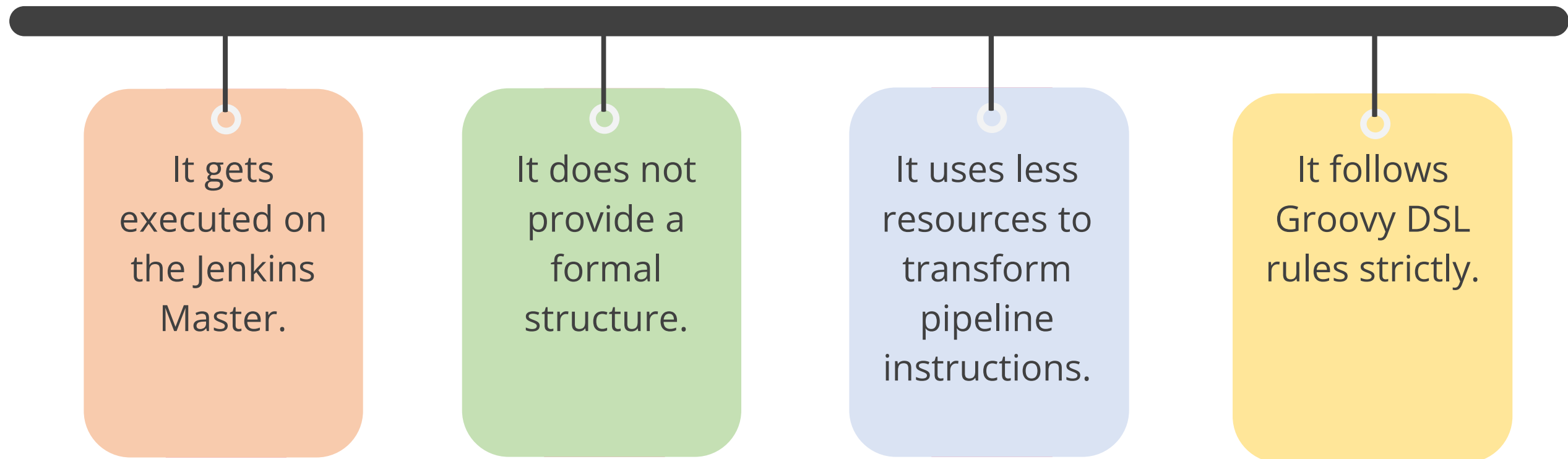
Scripted Pipeline

Supports coding of the pipelines using Groovy DSL

Declarative Pipeline

Eliminates the necessity for Groovy DSL

Note

Each type has its own syntax and features designed to cater to different needs and preferences in pipeline configuration.

Scripted Jenkins Pipeline

It is a traditional way of developing Jenkins pipelines. It supports configuration of large and complex pipelines. Following are some of its key features:

It gets executed on the Jenkins Master.

It does not provide a formal structure.

It uses less resources to transform pipeline instructions.

It follows Groovy DSL rules strictly.

To create a scripted pipeline, developers must start with the node elements within the node block.

Scripted Jenkins Pipeline: Syntax

All valid scripted pipelines must be enclosed within a node block, for example:
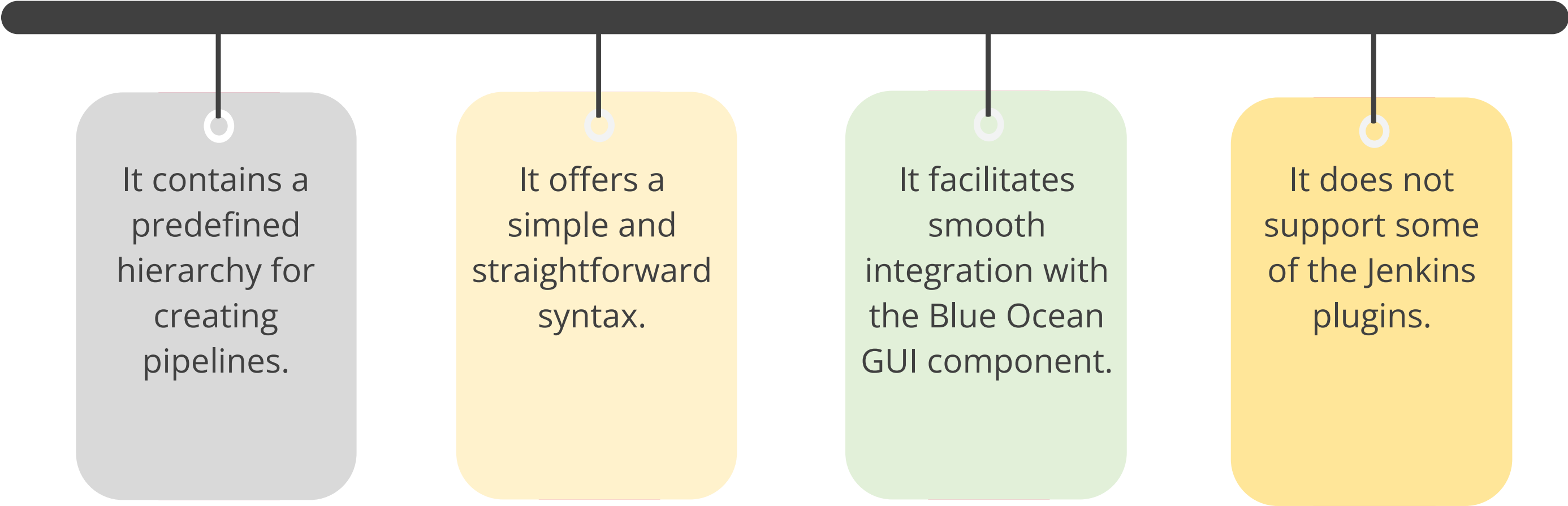
Syntax:

```
node('master') {
  /* insert Scripted Pipeline
here */
}
```

Example:

```
node('master') {
    stage('checkout code') {
        checkout scm
    }
    stage('build') {
        sh "${MVNHOME}/bin/mvn clean install"
    }
    stage('Test Cases Execution') {
        sh "${MVNHOME}/bin/mvn clean test"
    }
    stage('Production Deployment') {
        sh 'cp target/*.war /opt/tomcat8/webapps'
    }
}
```

Declarative Jenkins Pipeline

This is relatively easy to develop and manage. Developers have complete control over all the aspects of pipeline execution. Following are some of its key features:

It contains a predefined hierarchy for creating pipelines.

It offers a simple and straightforward syntax.

It facilitates smooth integration with the Blue Ocean GUI component.

It does not support some of the Jenkins plugins.

Declarative Jenkins Pipeline

All valid declarative pipelines must be enclosed within a pipeline block, for example:

Syntax:

```
pipeline {
  /* insert Declarative Pipeline
here */
}
```

Example:

```
pipeline {
    agent any
    stages {
        stage("Build") {
            steps {
                sh "${MVNHOME}/bin/mvn
clean install"
            }
        }
        stage("Test") {
            steps {
                sh "mvn clean test"
            }
        }
    }
}
```

Scripted vs. Declarative Syntax

Feature	Scripted syntax	Declarative syntax
Approach	Is a traditional approach to develop pipelines	Is a newer approach to develop pipelines
Syntax	Imperative scripting	Declarative syntax
Integration	Does not support integration with Blue Ocean GUI	Supports integration with Blue Ocean GUI
Error handling	Manual error handling required	Automatic error handling provided
Structure	Less structured; relies heavily on Groovy scripting	Highly structured; follows a predefined syntax

Creating a pipeline script   **Duration: 15 Min.**

Problem statement:

You have been assigned a task to create a pipeline script for automating build processes in Jenkins.

Outcome:

By the end of this demo, you will be able to create and execute a pipeline script in Jenkins, automating the build processes for your software projects and streamlining your development workflow.

Note: Refer to the demo document for detailed steps:

Steps to be followed:

1. Log in to Jenkins CI tool and create a pipeline script

Quick Check

As a developer, you are tasked with creating a visually appealing and intuitive Jenkins pipeline. Which tool can you leverage to accomplish this goal?

A. Jenkins Classic UI

B. Jenkins Blue Ocean Editor

C. Jenkins Console Interface (CLI)

D. Jenkins Pipeline Script Generator

Jenkinsfile Pipeline

Jenkinsfile Basics

A **Jenkinsfile** is a text file that contains the definition of a Jenkins pipeline and is checked into source control. It allows the users to script their pipeline's entire workflow.

Key components of a Jenkinsfile:

Pipeline block: Defines all the stages in the pipeline and wraps the entire pipeline process

Agent directive: Specifies where the entire pipeline or specific stages will run

Stages section: Contains one or more stage directives, which are used to conceptually segregate the steps of the entire pipeline (for example: build, test, and deploy)

Configuring Pipeline Code in Jenkins

There are two main ways to configure pipeline code in a Jenkins pipeline job:

Direct Jenkinsfile scripting:	• The Jenkinsfile contains the entire pipeline definition directly within the script. • This approach is suitable for simpler pipelines.

External script from Git repository:	• The Jenkinsfile specifies the location of a Git repository containing the pipeline code. • This allows for modularity and easier management of complex pipelines.

Direct Jenkinsfile Scripting

This process refers to directly writing the pipeline script within a Jenkinsfile stored in the source control. Shown below is a screenshot of a sample pipeline script:

External Script from Git Repository

Developers can fetch pipeline scripts from Git, enabling direct management of code changes by passing Jenkins intervention.

This approach enables developers to:

Configure source control tool

Control pipeline changes from Git repository
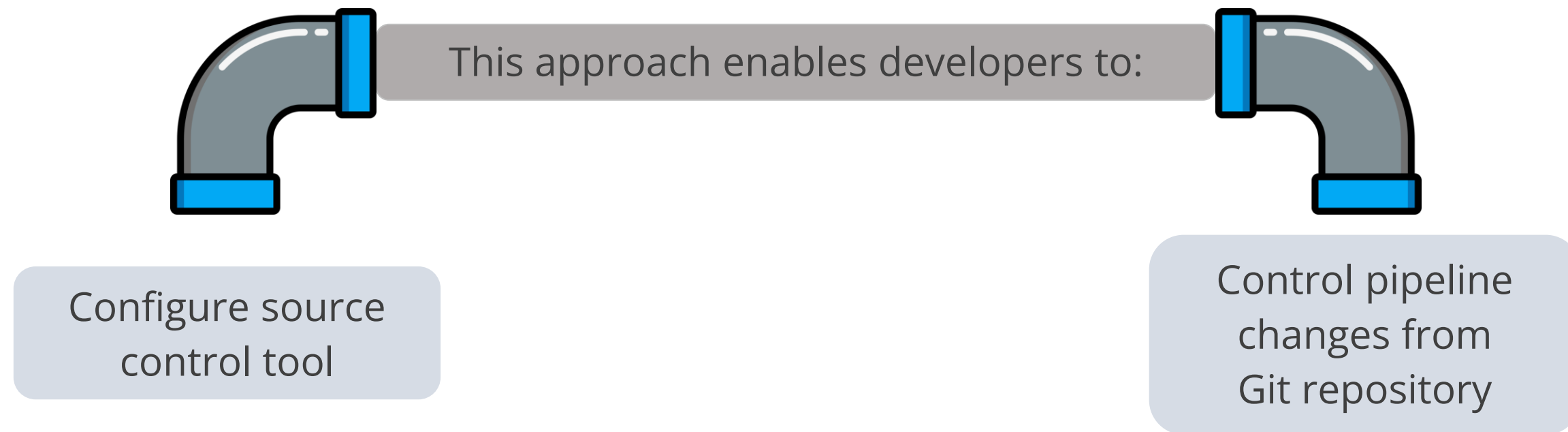
Setting up Jenkins pipeline job from Git   **Duration: 15 Min.**

Problem statement:

You have been assigned a task to set up a Jenkins pipeline job from Git version control system to enable automated CI for building, testing, and potentially deploying software upon code changes.

Outcome:

By the end of this demo, you will be able to configure a Jenkins pipeline job connected to a Git version control system, enabling automated Continuous Integration for building, testing, and potentially deploying software upon code changes.

Note: Refer to the demo document for detailed steps:

Steps to be followed:

1. Create a Git repository
2. Push the pipeline script into the Git repository
3. Create a pipeline script-based freestyle job

Jenkins Pipeline Variables

By using pipeline variables, manual coding can be avoided, and sensitive information can be moved around the entire pipeline script.

Boolean variables

Custom variables

Environment variables
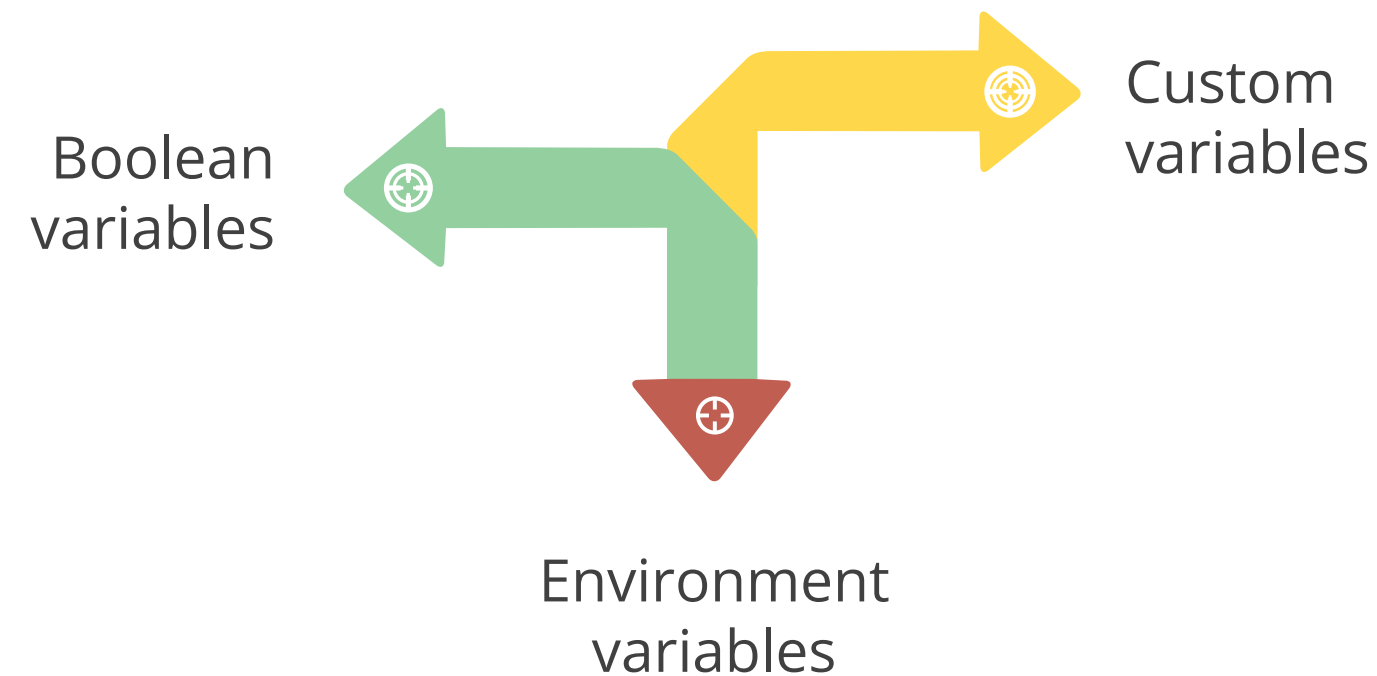
Variables in Pipeline scripts are defined using the keyword "def".
Example: def variable = "value"

Jenkins Pipeline Variables

Boolean variables
- Hold true or false values
- Example: **myBoolean** set to true

Environment variables
- Set externally, influencing pipeline behavior based on runtime conditions
- Example: **environment** set to production or development

Custom variables
- User-defined for storing diverse data types
- Example: **customVariable** may hold an application's API endpoint URL

Jenkins Credentials Plugin

The Jenkins credentials plugin provides a default internal credentials for storing sensitive information. Some key pointers describing this plugin are:
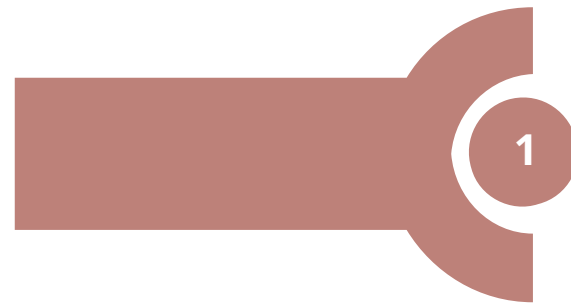
1 All the passwords and configurations used within Jenkins are primarily stored in Jenkins credentials.
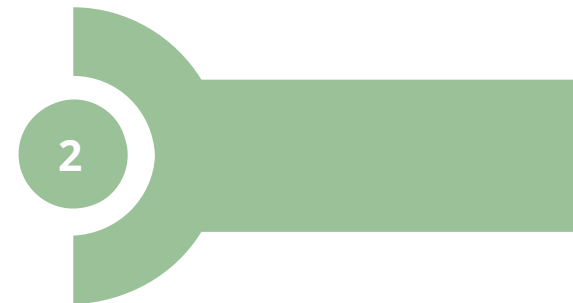
2 Both freestyle and pipeline-based projects have this component.

3 There are some vulnerabilities or hacks that make this mechanism unsafe.

Jenkins Credentials Plugin

Jenkins Credentials plugin supports two types of credentials:

Global scope

- Default scope

- Can be accessed by all jobs

- Primarily used for storing VCS credentials, AWS credentials, Secret token, and Secret text

System scope

- Only used internally by Jenkins system

- Not available to any other Jenkins job

- Restricts the usage of credentials and provides a high degree of confidentiality

Multibranch Jenkins Pipeline Setup

It is an extension of the Jenkins pipeline that automatically creates a pipeline for each branch in your repository. It is beneficial for projects with multiple branches being worked on concurrently, as it:

Helps developers to configure multiple branch configurations for a single project
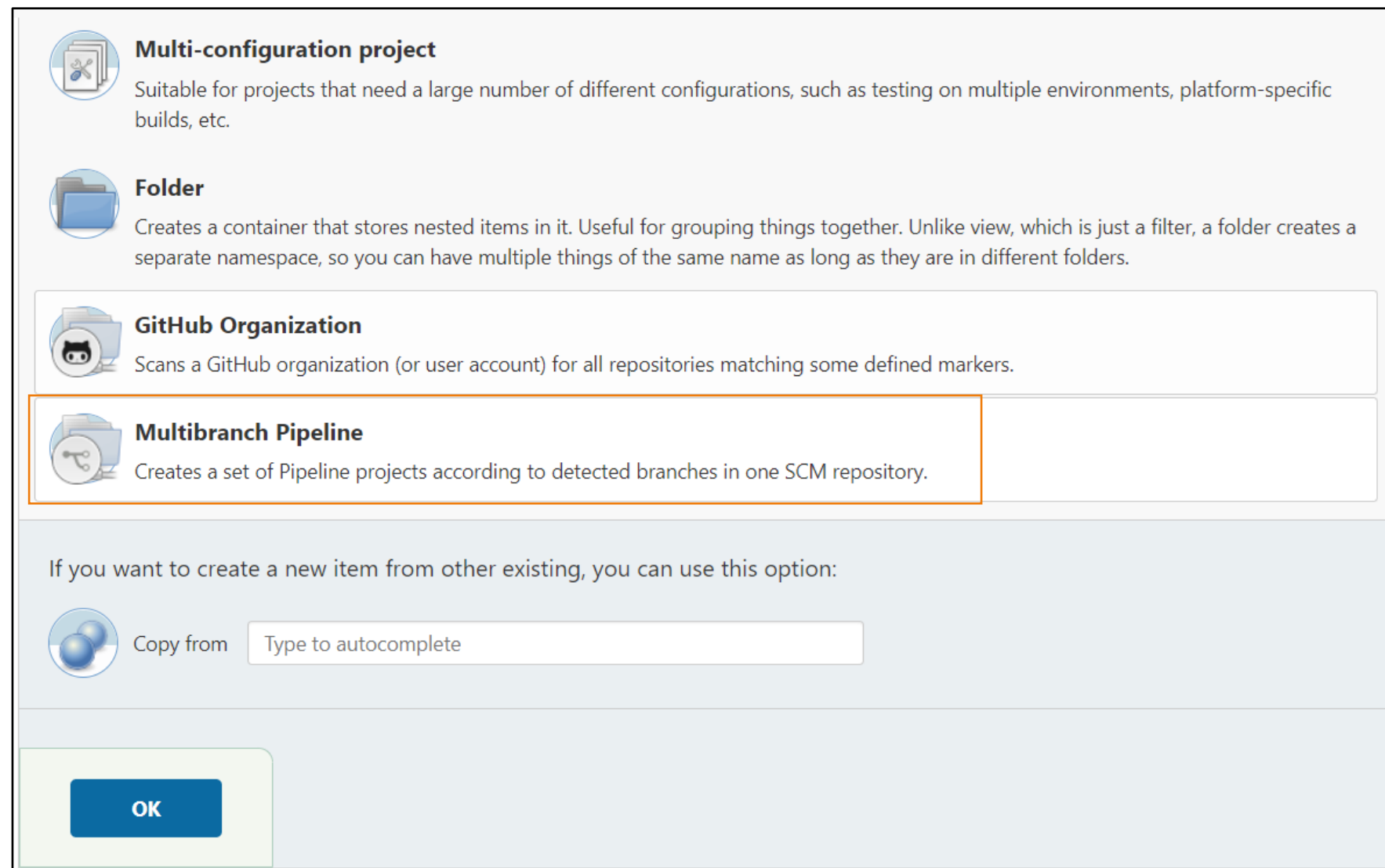
Scans for new branches

Allows developers to manage environment specific configurations inside Jenkins pipeline scripts

Multibranch Jenkins Pipeline Setup

When **New Item** is selected on the home page, a list of different project types will appear. Select **Multibranch Pipeline** as shown below:

Multibranch Jenkins Pipeline Setup

As shown in the below screenshot, add a branch source (for example, Git) and enter the location of the repository:
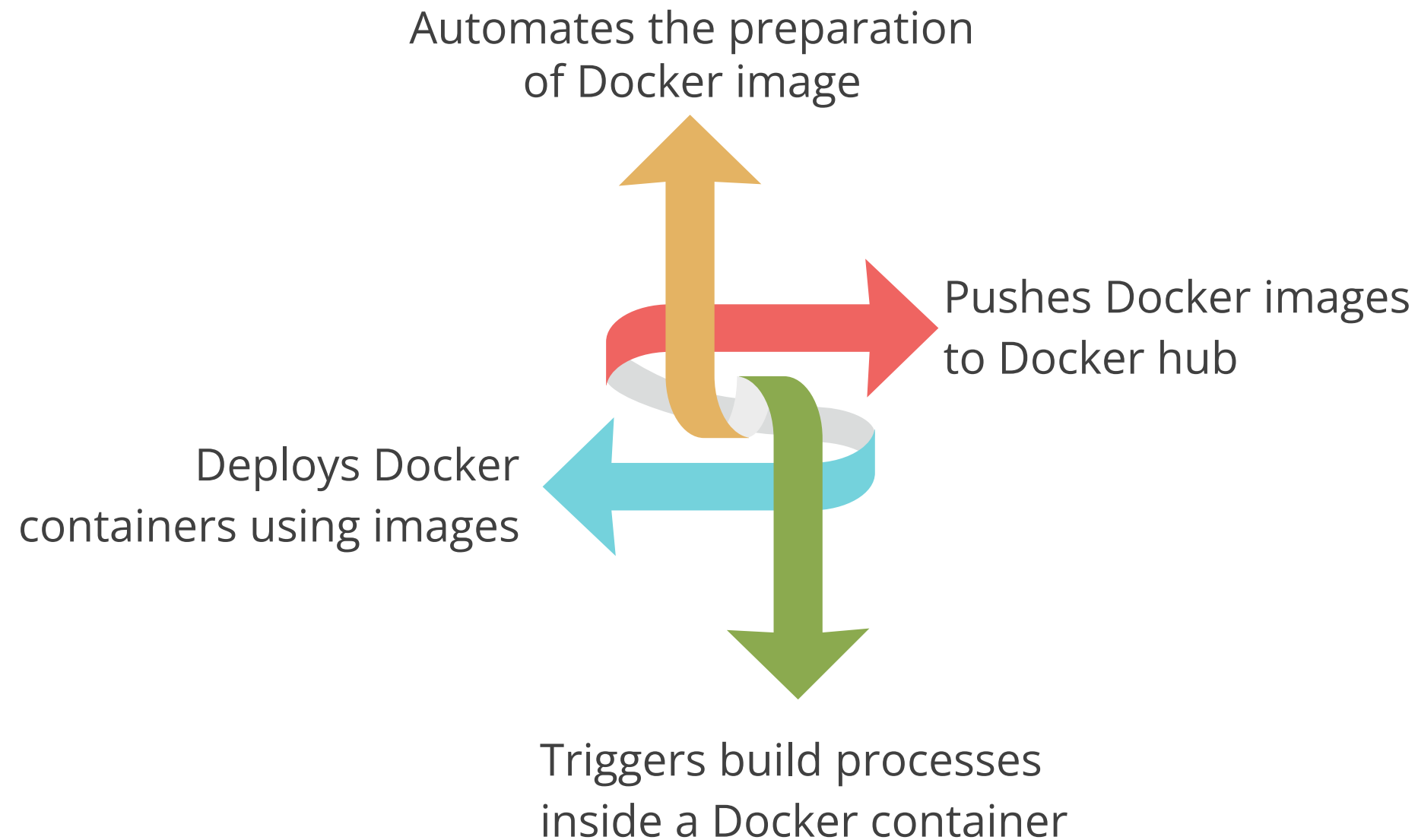
Docker Plugin for Jenkins Pipeline

This plugin supports the Pipeline setup to use Docker images as an execution environment.

Uses of Docker plugin:

Automates the preparation
of Docker image

Pushes Docker images
to Docker hub

Deploys Docker
containers using images

Triggers build processes
inside a Docker container

Docker Plugin for Jenkins Pipeline

The Docker plugin can be downloaded from the Available tab in the Jenkins dashboard.

Dockerized Jenkins Pipeline

Docker containers are utilized for building, testing, and deployment in a seamless CI/CD workflow, ensuring consistent environments and enhanced portability.

Dockerized Jenkins Pipeline

```
Pipeline {
    agent {
        docker { image 'node:14-alpine' }
    }
    stages {
        stage('Test') {
            steps {
                sh 'node --version'
            }
        }
    }
}
```

Configuring multibranch Jenkins pipeline job   **Duration: 15 Min.**

Problem statement:

You have been assigned a task to configure a Multibranch Jenkins pipeline job to facilitate managing multiple projects within a single job setup.

Outcome:

By the end of this demo, you will be able to configure a Multibranch Jenkins pipeline job, enabling efficient management of multiple projects within a single setup, streamlining your CI/CD workflow.

Note: Refer to the demo document for detailed steps:

Assisted Practice: Guidelines

Steps to be followed:

1. Create a Git repository
2. Push the code file into the Git repository
3. Create a Multibranch Pipeline

Quick Check

What is the primary file used to define Jenkins pipeline configurations?

A. Jenkinsfile

B. Dockerfile

C. config.xml

D. pipeline.yml

Jenkins Administration

Global Security Configuration

Jenkins supports security features that enable developers to make Jenkins more secure. This can be achieved by using **Global Security Configuration**.

Global Security Configuration

| Includes both authorization and authentication features to restrict access to limited users only | Adds roles and permissions for individual users to restrict Jenkins access | Assigns custom permissions to different users in Jenkins application |

Jenkins Authentication

Jenkins authentication enables developers to restrict usage of Jenkins to authenticated users only.

Jenkins authentication locks access to Jenkins UI so that only an authenticated user can log into Jenkins to access build jobs.

Individual users cannot gain unwanted access to critical Jenkins jobs like production deployment jobs and code scan jobs.

Jenkins Authentication

The **Security Realm** authentication enables developers to select an authentication source that will be used by Jenkins to authenticate users. This can be done as shown below:

Jenkins Authorization

Authorization indicates what an individual can access in Jenkins CI once authenticated.

Shown below are the various authorization options available in Jenkins:

Assign specific roles to individual users without giving similar admin access to every user

Authorization

- ○ Anyone can do anything
- ○ Legacy mode
- ● Logged-in users can do anything
  - ☐ **Allow anonymous read access**

- ○ Matrix-based security
- ○ Project-based Matrix Authorization Strategy

Jenkins Roles and Permissions

Jenkins uses the **Role-based Authorization Strategy** plugin to implement roles and permissions configurations for Jenkins authorization.

Role-based Authorization Strategy plugin

Creates specific roles which can be used to control user permissions	Creates roles such as admin, developer, read only, or anonymous user	Creates and manages project based roles

Jenkins Roles and Permissions

Below is the screenshot of the **Available** tab showing the **Role-based Authorization Strategy** plugin that can be installed by admins.

Jenkins Stop and Start Maintenance

Jenkins on Linux OS gets deployed as a **Unix service** that can be easily bounced using the Linux service command as:

service Jenkins status

service Jenkins restart
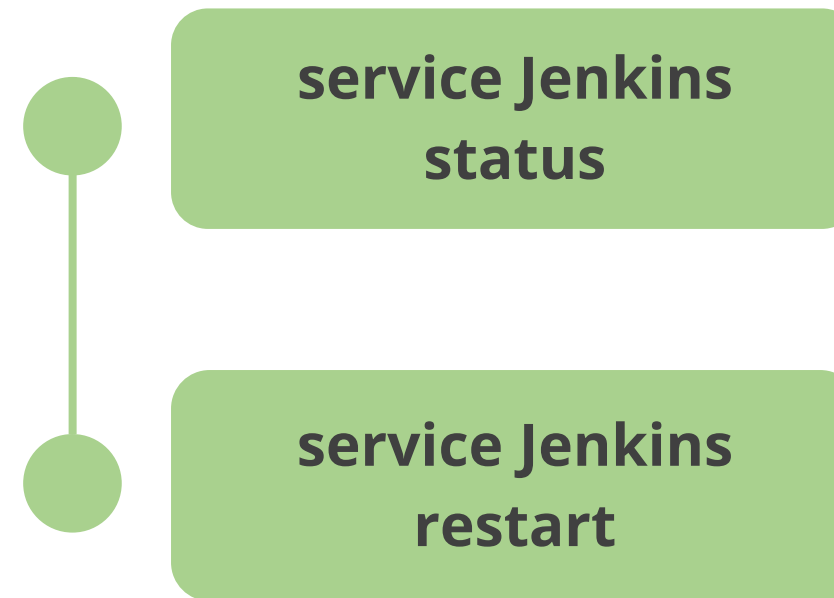
- If Jenkins is installed on windows platform, then restart Jenkins from windows service
- To access service in windows, run a program called **services**

Jenkins Backup and Restore Plugin Integration

It is important to keep a backup of the data and configurations so that admins can take a backup of Jenkins configurations before any upgrade or server patching.

Jenkins Backup and Restore Plugin Integration

Below are the plugins and their features:

Backup

- Helps admins restore Jenkins home directory
- Customizes backups
- Is a manual trigger
- Does not support scheduling

Backup and interrupt Job

- Takes a running job backup
- Interrupts a scheduled job
- Schedules the job back once Jenkins is up and running again

Google cloud backup

- Supports backup to Google cloud
- Supports auto restoration

Jenkins Backup and Restore Plugin Integration

Periodic backup

- Is an extension of the backup plugin
- Provides an extensive feature of scheduled backup
- Provides more features as compared to backup plugin

Thin backup

- Is the most reliable backup and restoration plugin
- Allows both manual and automated backups

Jenkins Build Metrics

Jenkins provides support to check build metrics and trends with the help of plugins like **build-metrics** plugin.

To install the **build-metrics** plugin, go to **Manage Jenkins** and click **Plugin Management**. Then, install the plugin listed in the **Available** tab.

Jenkins Logging

Logging is important for any application from the perspective of troubleshooting. Jenkins supports logging like any Java application, to understand the reason behind any failure or error.

Jenkins log files are available at:

/var/log/jenkins/jenkins.log in Linux machines

%JENKINS_HOME%/jenkins.out and **%JENKINS_HOME%/jenkins.err** in Windows machines

Quick Check

As an IT professional tasked with maintaining Jenkins, which feature should you prioritize to manage user access securely and ensure appropriate permissions?

A. Jenkins Authorization

B. Jenkins Stop or Start Management

C. Jenkins Builds Monitoring

D. Jenkins Disk usage Monitoring

Key Takeaways

◉ A Jenkins pipeline includes all the tools needed to orchestrate testing, merging, packaging, shipping, and deploying code.

◉ Jenkins supports creation of pipeline and multi-branch build jobs.

◉ Apache Groovy seamlessly integrates with Jenkins pipeline, providing powerful scripting capabilities and an inline.

◉ Jenkinsfile helps to implement the pipeline as a code functionality in Jenkins.

◉ Developers can fetch Pipeline scripts from Git, enabling direct management of code changes by passing Jenkins intervention.

◉ Jenkins Authentication enables developers to restrict usage of Jenkins to authenticated users only.

Implementing CI/CD Pipeline for Deploying Application to Tomcat Apache

Duration: 30 Min.

Project agenda: To implement a continuous integration and continuous deployment (CI/CD) pipeline in Jenkins for deploying a Java application to Tomcat Apache

Description: You work as a DevOps Engineer in an IT firm. Your company is undertaking a project to modernize its application deployment processes, aiming to streamline the deployment of new software updates. As part of this initiative, you are tasked with setting up a continuous integration and continuous deployment (CI/CD) pipeline to automate the deployment of applications to the Tomcat Apache server hosted on an Ubuntu VM.

Implementing CI/CD Pipeline for Deploying Application to Tomcat Apache

Duration: 30 Min.

Perform the following:

1. Install Tomcat Apache 9 on Ubuntu VM
2. Log in to Jenkins CI tool and install the Deploy to Container plugin
3. Configure the deployment stage in Jenkins pipeline

Expected deliverables: A step-by-step guide for setting up and implementing a CI/CD pipeline for deploying applications to Tomcat Apache on an Ubuntu VM, including integration with Jenkins for automated builds and deployments.

Thank You