

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
- Create a pipeline script for automating build processes in Jenkins
- Set up a Jenkins pipeline job from the Git version control system to enable automated CI for building, testing, and deploying software

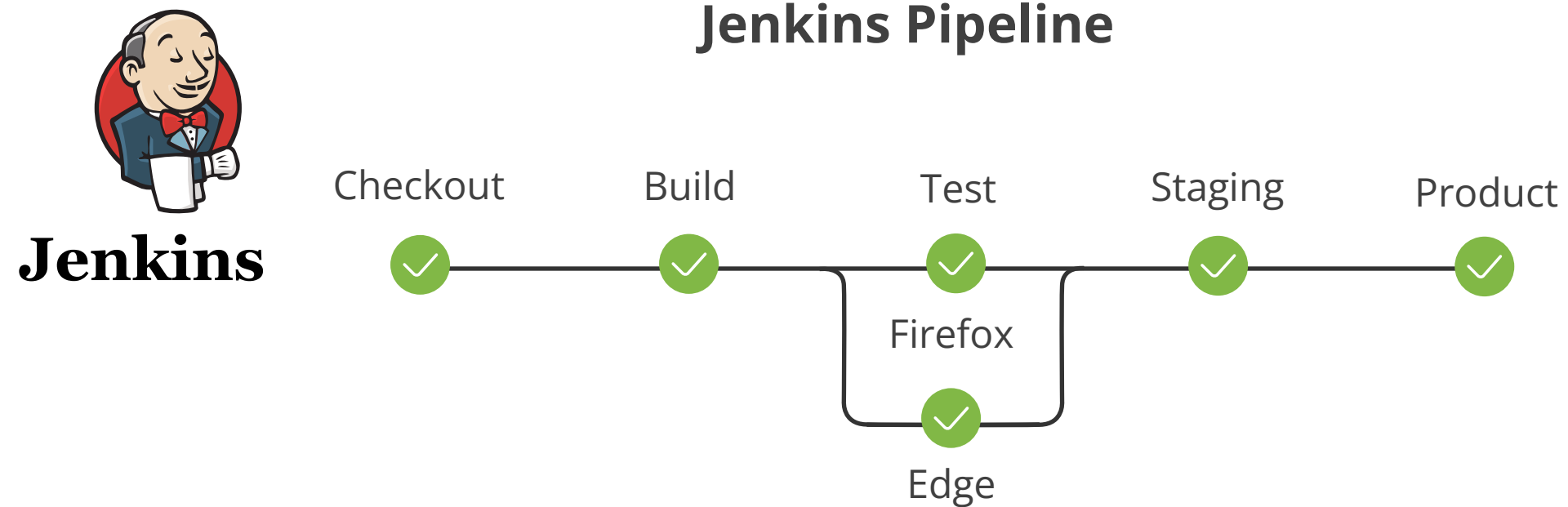




Getting Started with Jenkins Pipeline

Introduction to Jenkins Pipeline

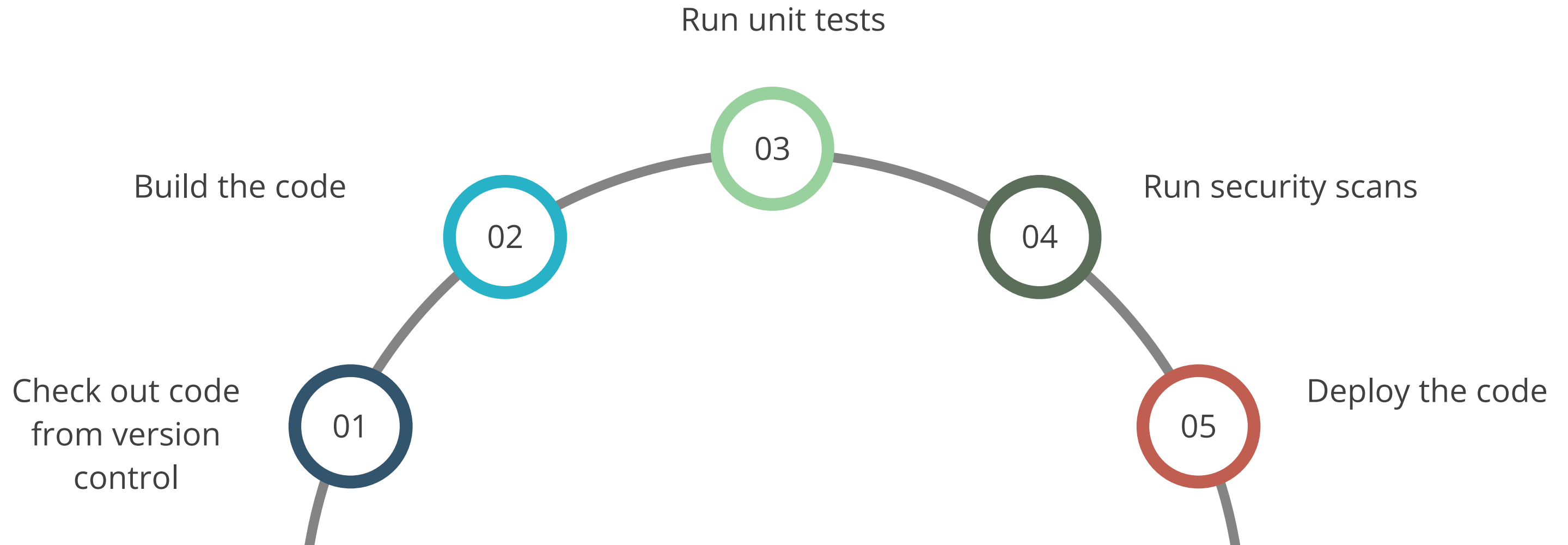
It is a suite of plugins that allows to define and automate continuous delivery workflows as code within a Jenkinsfile.



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:



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

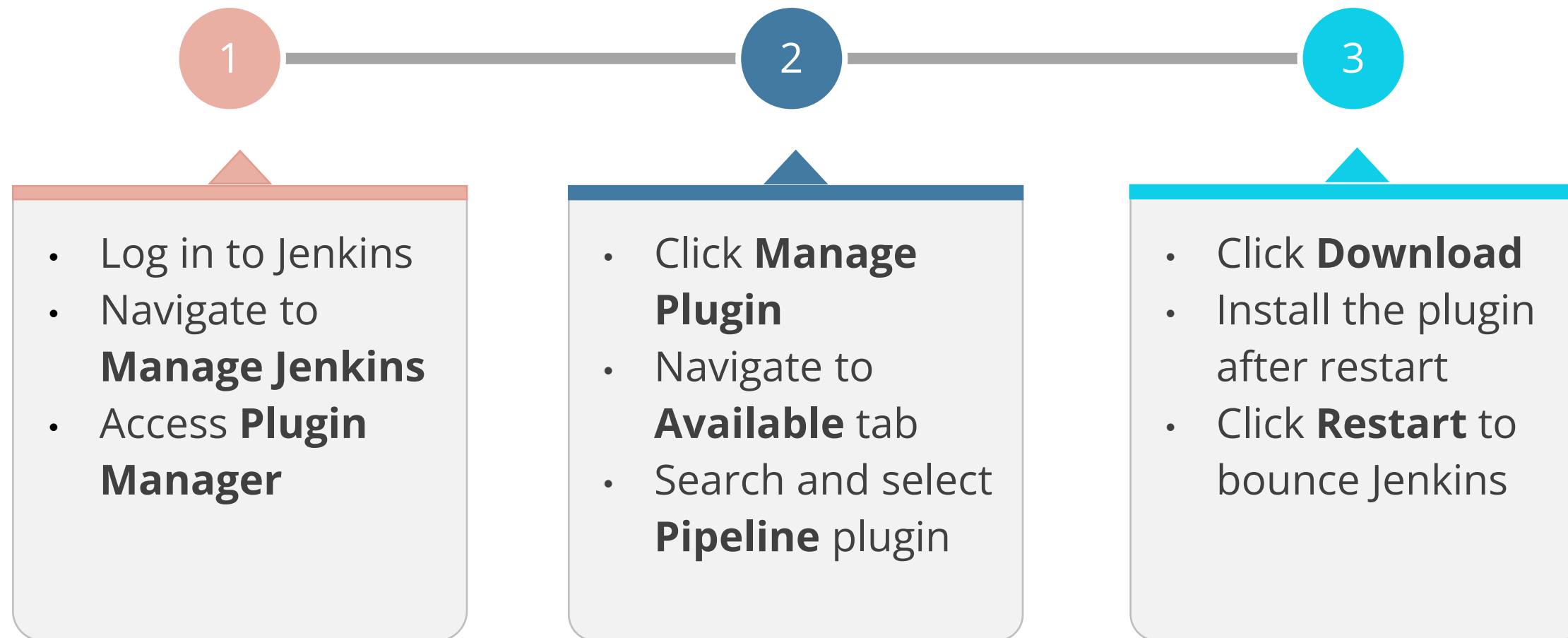
Track changes and rollback effortlessly by storing your pipeline definition in a Jenkins file

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:



Jenkins Pipeline Job: Setup

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

Q pipeline

Updates

Available

Installed

Advanced

| Install ↑ | Name | Version | Released |
|--------------------------|---|---------|------------------|
| <input type="checkbox"/> | <div><div>Pipeline: Declarative</div><div>Miscellaneouspipeline</div><div>An opinionated, declarative Pipeline.</div></div> | 1.8.5 | 1 mo 1 day ago |
| <input type="checkbox"/> | <div><div>Pipeline</div><div>Command Line InterfaceMiscellaneousAgent Launchers and ControllersBuild Triggers</div><div>A suite of plugins that lets you orchestrate automation, simple or complex. See Pipeline as Code with Jenkins for more details.</div></div> | 2.6 | 2 yr 9 mo ago |
| <input type="checkbox"/> | <div><div>Docker Pipeline</div><div>DeploymentDevOpsdockerpipeline</div><div>Build and use Docker containers from pipelines.</div></div> | 1.26 | 4 mo 12 days ago |
| <input type="checkbox"/> | <div><div>Pipeline: Declarative Agent API</div><div>Miscellaneouspipeline</div><div>Replaced by Pipeline: Declarative Extension Points API plugin.</div><div><div>This plugin is deprecated.</div><div>In general, this means that it is either obsolete, no longer being developed, or may no longer work.</div><div>Learn more.</div></div></div> | 1.1.1 | 4 yr 3 mo ago |

Install without restart

Download now and install after restart

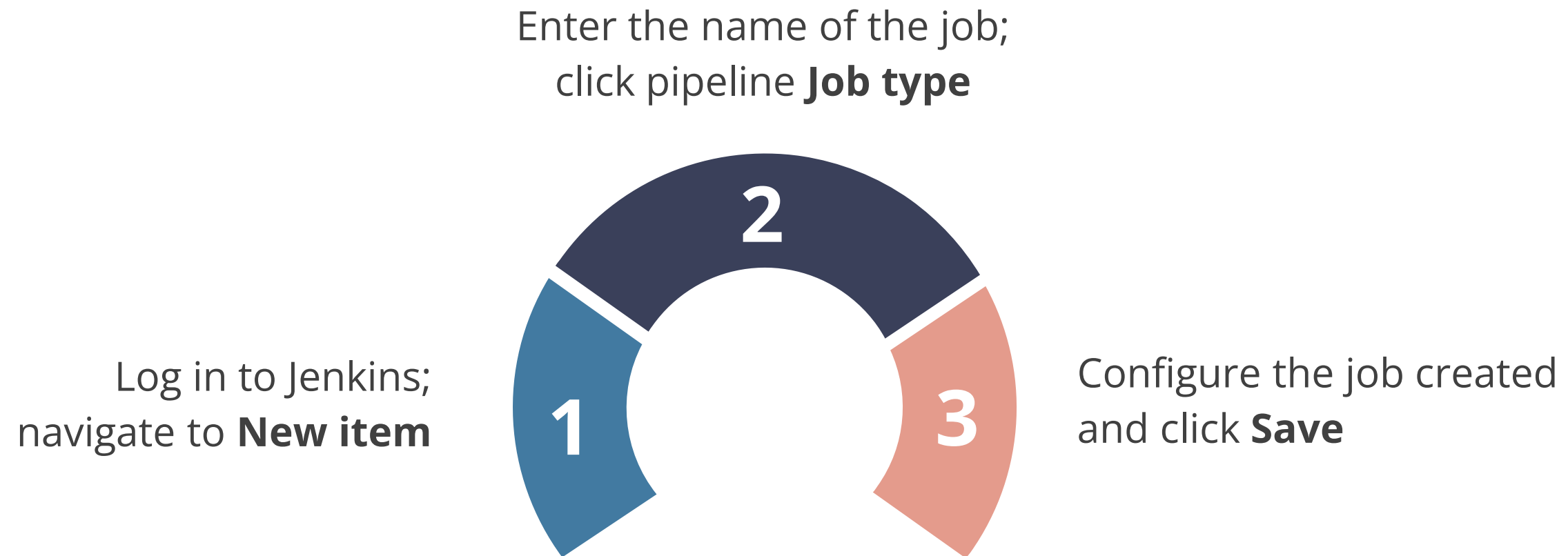
Update information obtained: 16 hr ago

Check now

Jenkins Pipeline Job: Build

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

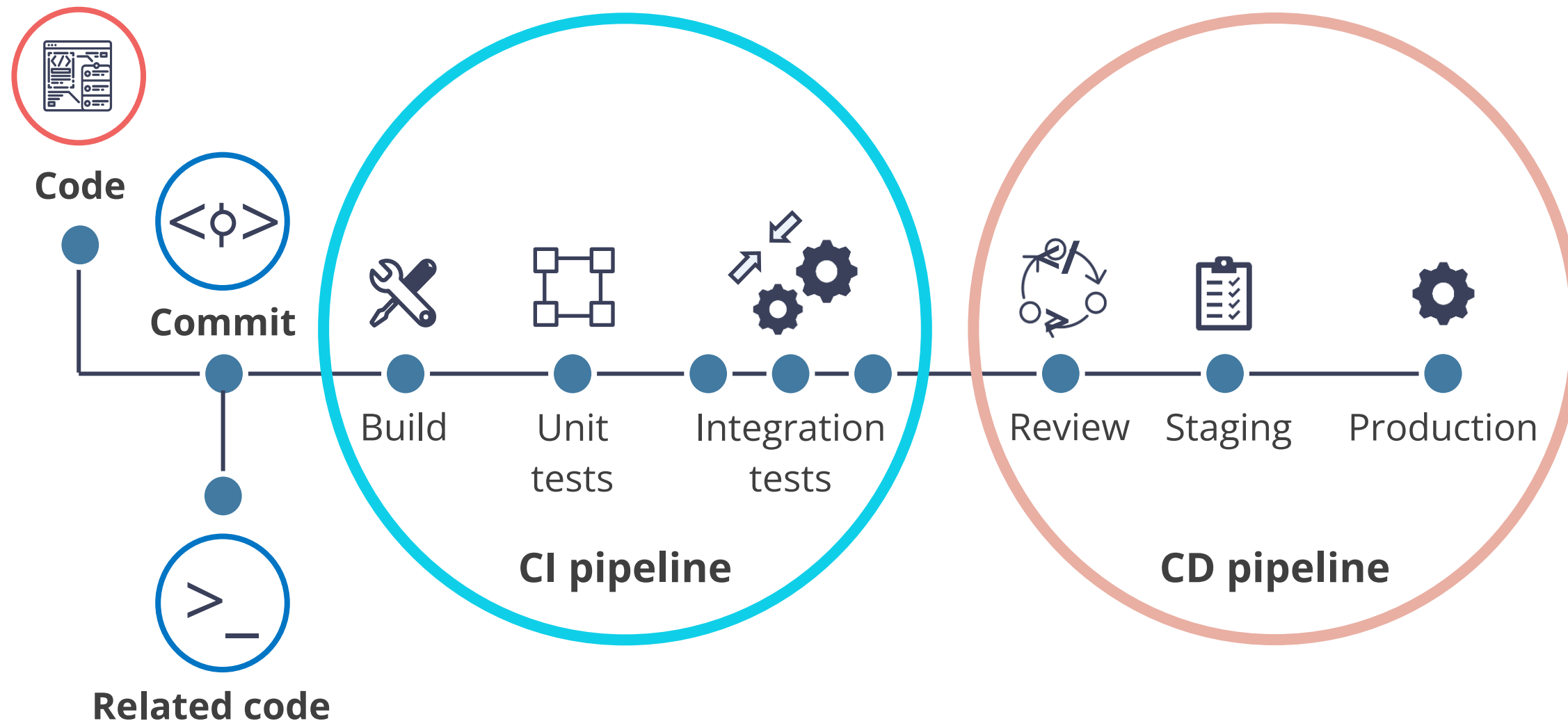
Below are the steps to create a pipeline job:



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:



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.

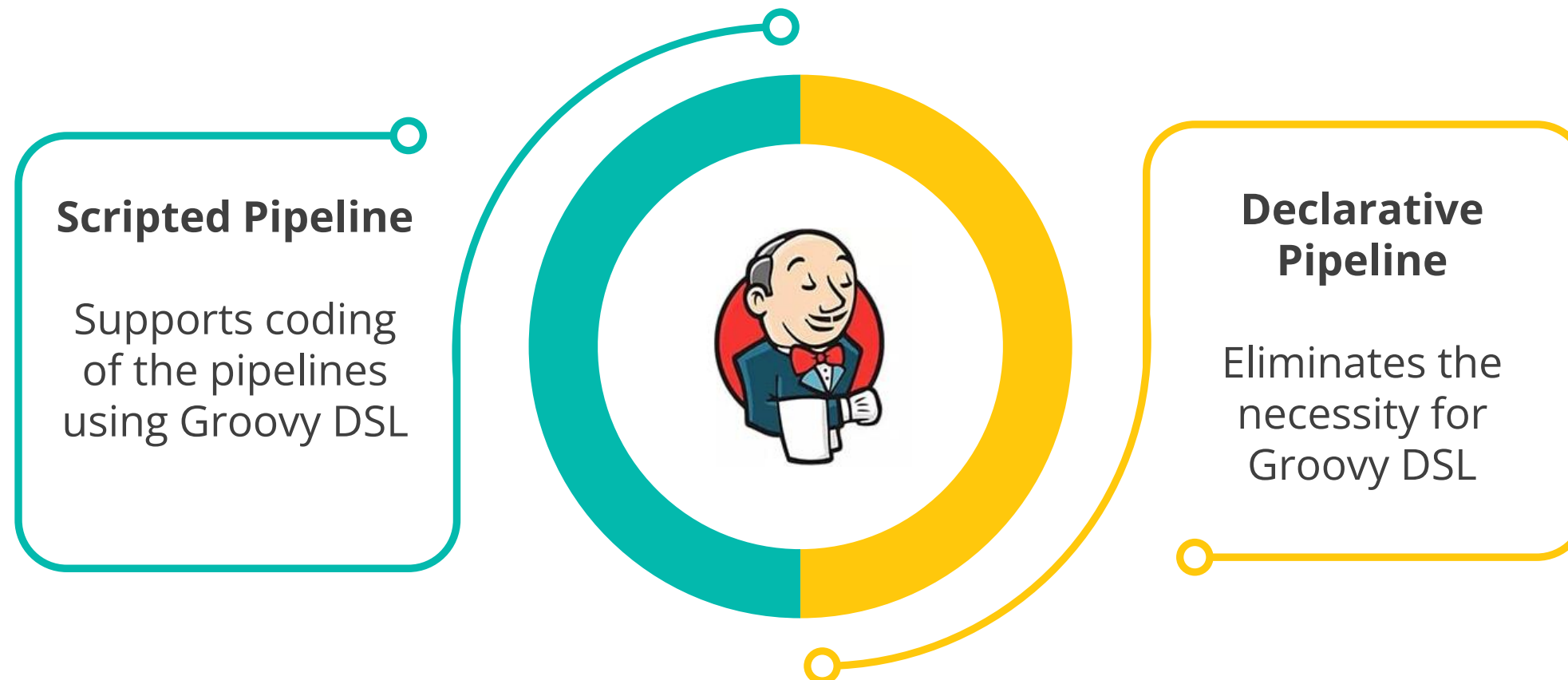
Jenkins Interfaces and Tools

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 |

Jenkins Pipeline: Syntax Types

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

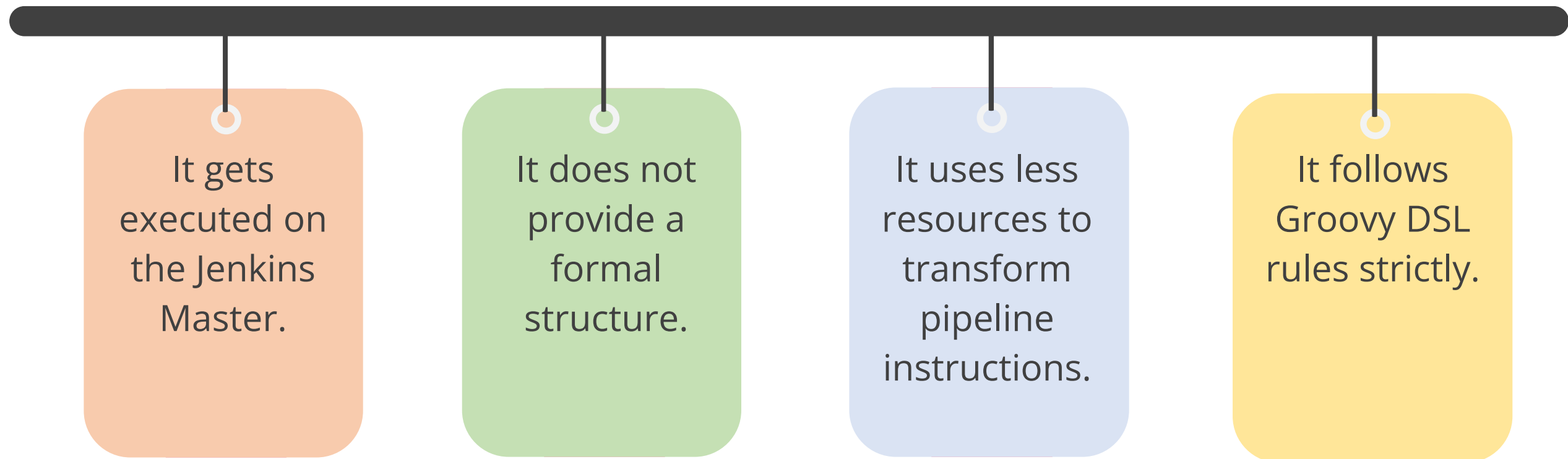


Note

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:



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:

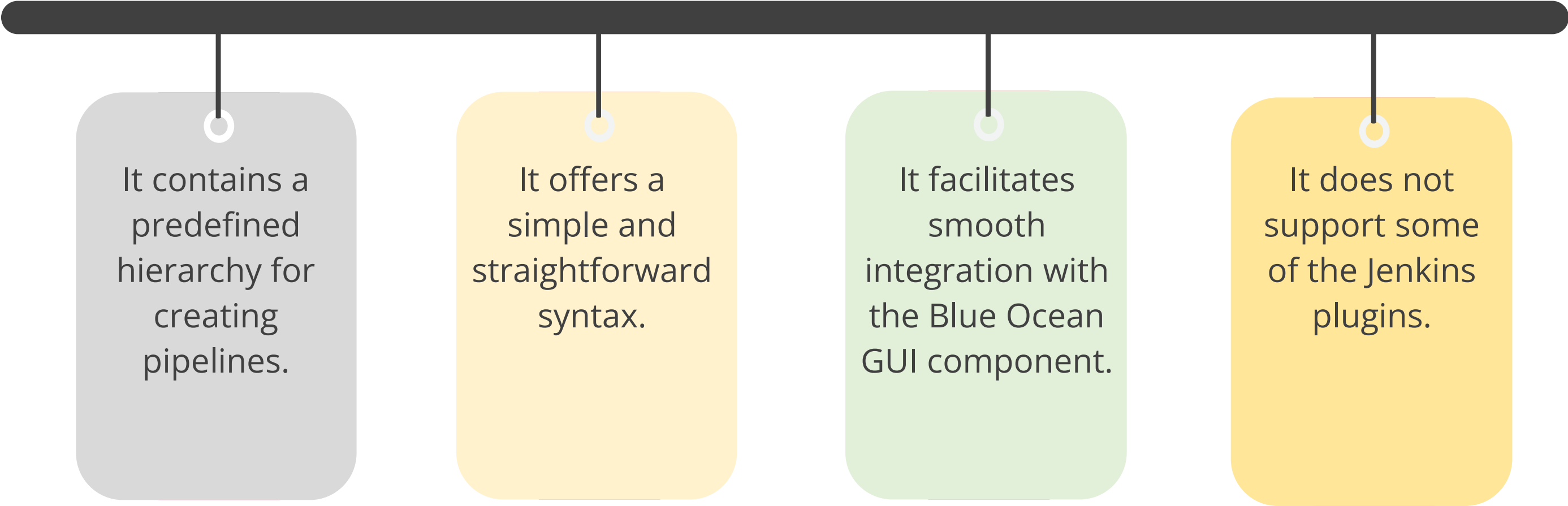
```
node('master') {  
    /* insert Scripted Pipeline  
    here */  
}
```

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 |

Assisted Practice



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:

Assisted Practice: Guidelines



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:

Pipeline

Definition

Pipeline script

Script

```
1 pipeline {
2   agent any
3
4   tools {
5     // Install the Maven version configured as "M3" and add it to the path.
6     maven "M3"
7   }
8
9   stages {
10    stage('Build') {
11      steps {
12        // Get some code from a GitHub repository
13        git 'https://github.com/jglick/simple-maven-project-with-tests.git'
14
15        // Run Maven on a Unix agent.
16        sh "mvn -Dmaven.test.failure.ignore=true clean package"
17      }
18    }
19  }
20 }
```

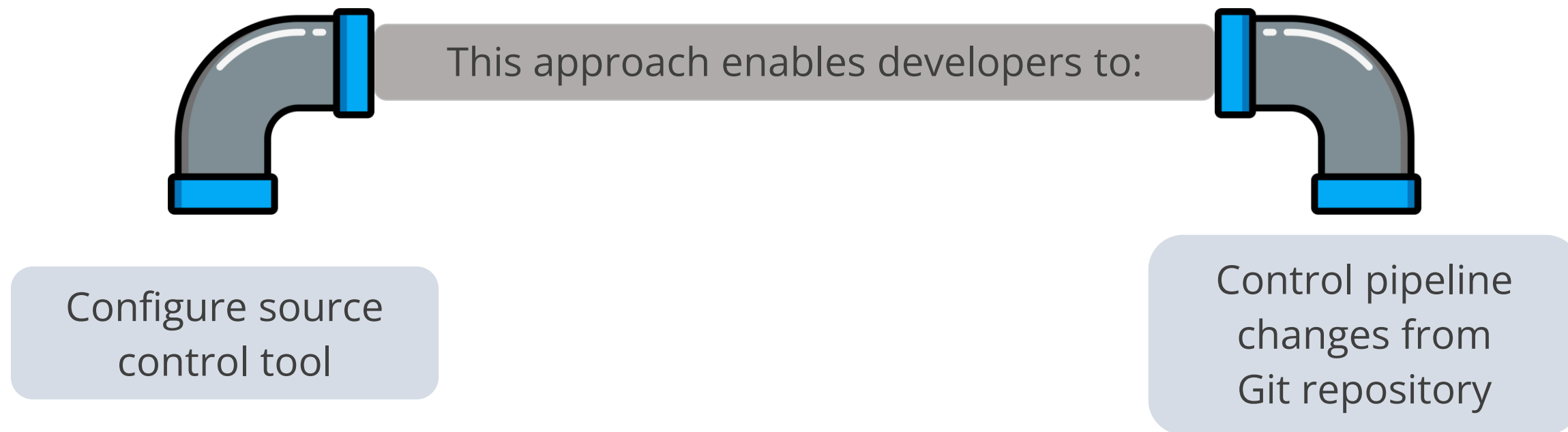
try sample Pipeline...

☒ Use Groovy Sandbox

Pipeline Syntax

External Script from Git Repository

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



Assisted Practice



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:

Assisted Practice: Guidelines

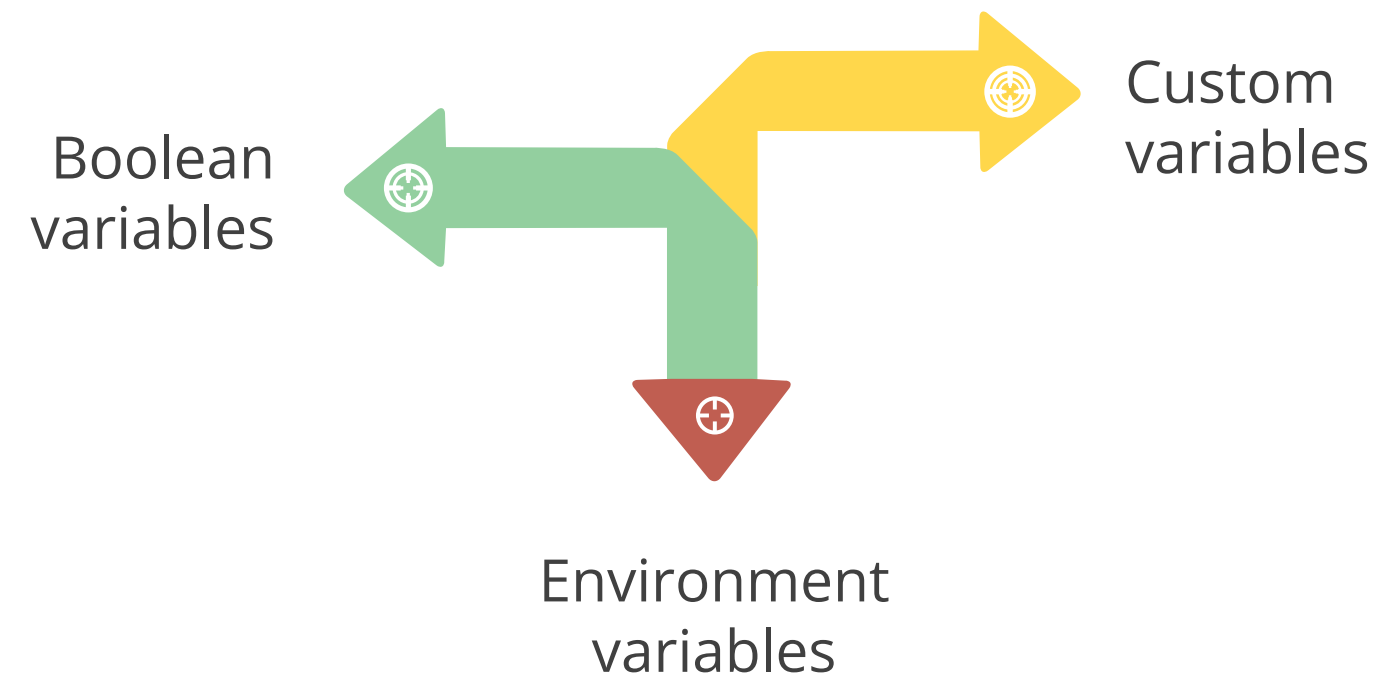


Steps to be followed:

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.



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

Quick Check



You need to set up a Jenkins pipeline for a project stored in a Git repository. The pipeline should be easily maintainable and use variables for different stages. What is the best approach?

- A. Write the pipeline script in the Jenkins UI
- B. Use a Jenkinsfile stored in the Git repository
- C. Use a separate script from another repository
- D. Configure the pipeline directly in Jenkins with inline scripting

Key Takeaways

- A Jenkins pipeline includes all the tools needed to orchestrate testing, merging, packaging, shipping, and deploying code.
- Jenkins supports the creation of pipeline and multi-branch build jobs.
- Apache Groovy seamlessly integrates with the 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.



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