Exploring Jenkins

Introduction to Jenkins

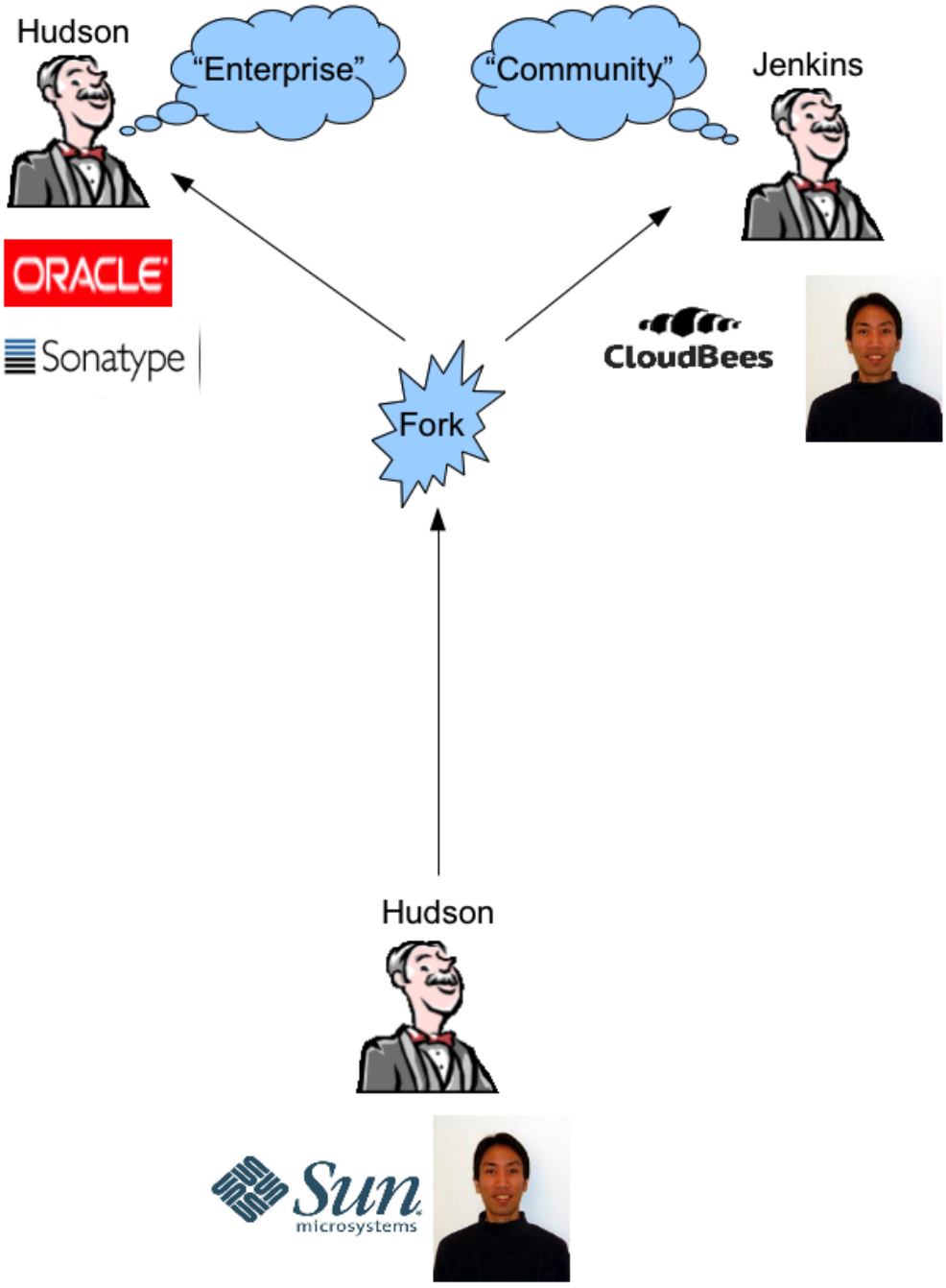
What is Jenkins?

- **Jenkins** is an open-source automation server used to automate the building, testing, and deployment of software applications. It is one of the most popular tools for continuous integration and continuous delivery (CI/CD) processes.

- It helps developers and teams to implement automation in a consistent and efficient manner, reducing manual intervention and increasing the speed of software delivery.
- Jenkins is highly extensible, with a vast library of plugins that allow it to integrate with various development, testing, and deployment tools and platforms.

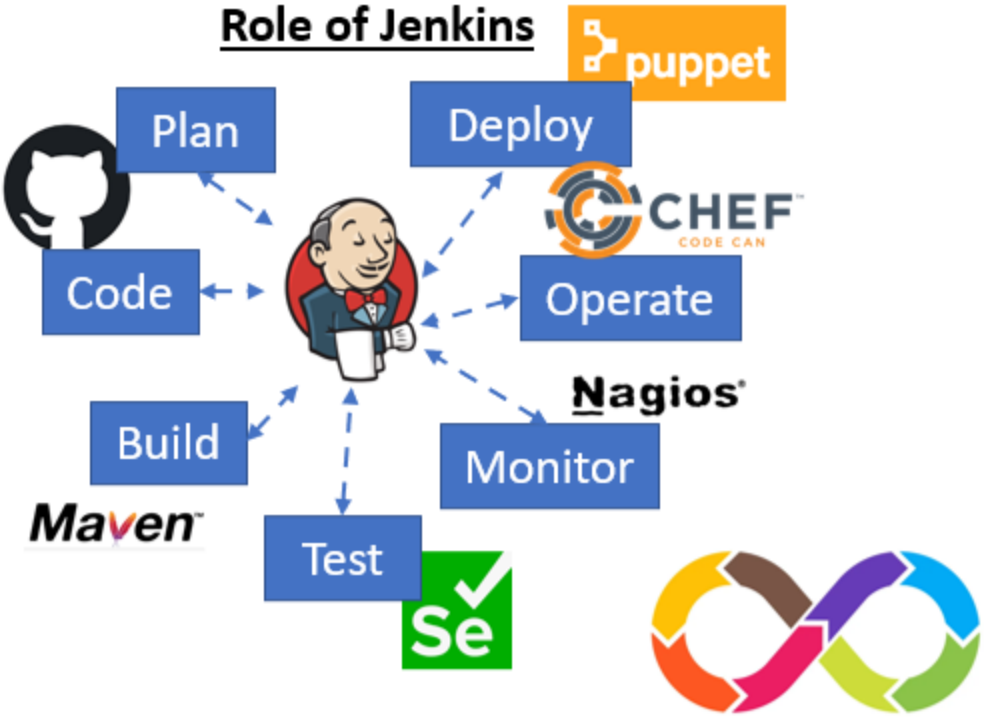
History and Evolution of Jenkins

- **Origins** : Jenkins was originally developed as an extension of the Hudson project in 2004 at Sun Microsystems. The goal was to create a robust tool for continuous integration that streamlined software development processes.
- **Fork from Hudson** : In 2011, due to conflicts between the open-source community and Oracle (which acquired Sun Microsystems), the Jenkins project was forked from Hudson. This led to the independent Jenkins community focusing on innovation and expansion of the platform.


Hudson
“Enterprise”
“Community”
Jenkins
ORACLE
Sonatype
CloudBees
Fork
Hudson
Sun
microsystems

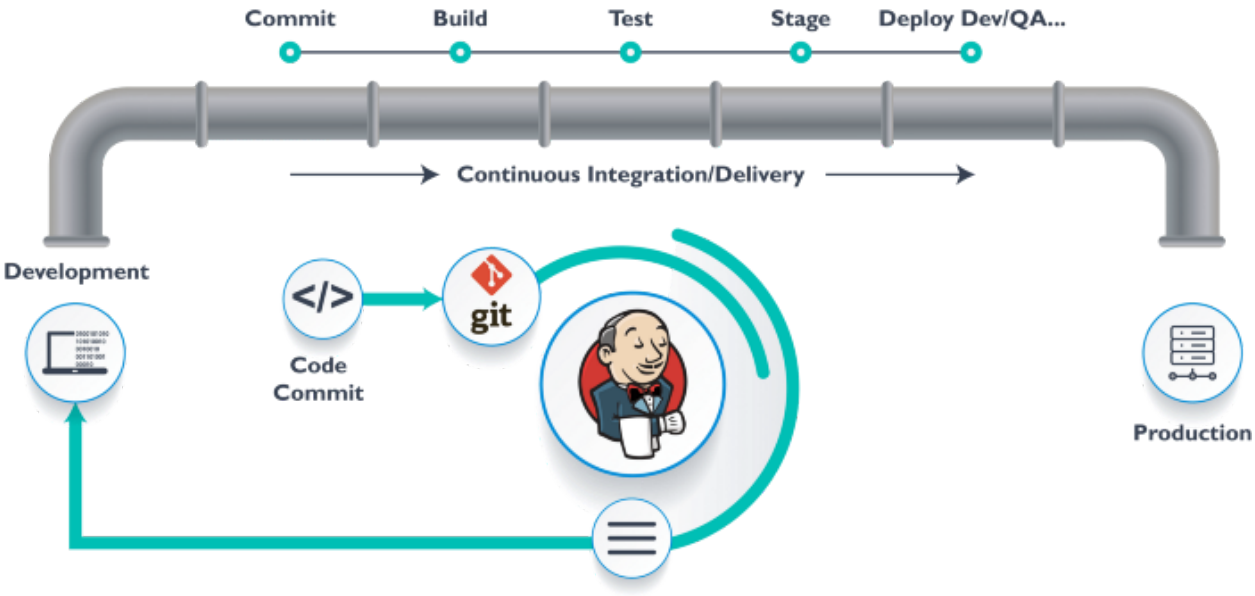
- **Growth and Popularity** : Over the years, Jenkins has seen widespread adoption in the development community because of its flexibility, ease of setup, and a strong ecosystem of plugins. It has become the standard choice for CI/CD workflows, supported by a large and active open-source community.
- **Jenkins 2.0** : Released in 2016, Jenkins 2.0 introduced the concept of "Pipeline as Code," allowing users to define their continuous delivery pipelines through a Jenkinsfile written in a domain-specific language. This release marked a significant enhancement in usability and functionality.

Importance of Jenkins in CI/CD Pipelines



- **Automation** : Jenkins automates various parts of the software development lifecycle, from code integration to testing and deployment, reducing manual errors and increasing efficiency.

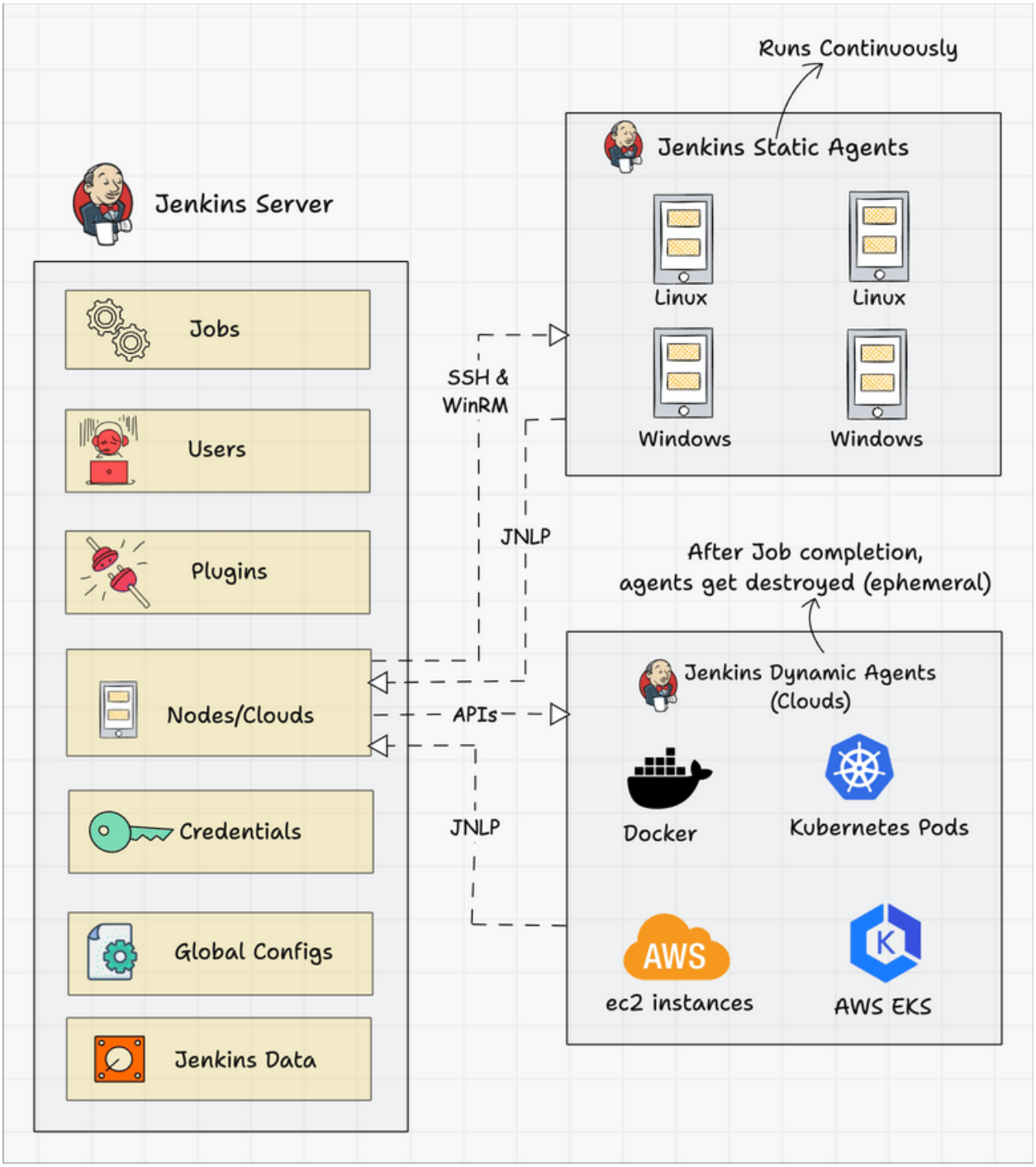
- **Continuous Integration (CI)** : Jenkins can retrieve the latest code changes from a version control system, automatically build the application, and run tests. This ensures that the codebase remains in a deployable state and accelerates the development feedback loop.
- **Continuous Delivery (CD)** : With Jenkins, teams can automate the deployment process, making it repeatable and reliable. Jenkins pipelines can incorporate various stages, such as build, test, and deploy, to ensure applications are delivered continuously and consistently.
- **Flexibility and Extensibility** : The plugin ecosystem allows Jenkins to support a wide range of technologies and platforms, making it adaptable to different project requirements and technology stacks.
- **Community and Support** : Jenkins benefits from a strong community and extensive documentation, making it an accessible choice for teams of all sizes looking to implement CI/CD practices.



Jenkins serves as a central hub in modern DevOps practices, enabling teams to achieve faster and more reliable software delivery. By adopting Jenkins, organizations can enhance collaboration and improve their overall development workflow.

Core Concepts of Jenkins

Jenkins Architecture Overview: Master-Slave Architecture



- **Jenkins Master**
  - The Jenkins master is the central server responsible for orchestrating the entire Jenkins system's workflow, acting as the core orchestrator for managing jobs, scheduling builds, and delegating tasks to the slave nodes.
  - It holds the configuration of all jobs and is responsible for maintaining the user interface and communication with external systems, such as source control repositories and notification systems.
- **Jenkins Slave (Agent)**
  - Jenkins slaves, also known as agents, are machines configured to execute build jobs.
  - They can run on numerous platforms, enabling horizontal scaling, which allows for efficient distribution of tasks across multiple nodes.
  - Agents can be connected to the master using SSH, JNLP, or web sockets.
- **Overall Workflow**
  - The Jenkins master schedules and distributes build tasks to the Jenkins slaves based on their availability.
  - This distributed build architecture helps handle large-scale build processes by distributing them across multiple environments, increasing efficiency and reliability.

Key Components

- **Jobs**
  - Jobs are the fundamental unit of work in Jenkins. They represent a task or a sequence of tasks that Jenkins will perform, such as building a project, running tests, or deploying an application.
- **Pipelines**
  - Jenkins Pipelines are a suite of plugins that support implementing and integrating continuous delivery pipelines into Jenkins.
  - A pipeline is defined using a Jenkinsfile, which employs either a Declarative or Scripted syntax to describe the process of building, testing,

and deploying applications.

    - Pipelines offer improved version control, visibility, and resilience over traditional Freestyle jobs, supporting more complex workflows and easier maintenance of repetitive tasks.

- **Nodes and Executors**
    - A node in Jenkins refers to a machine on which Jenkins can execute jobs. This could be the master or any configured slave.
    - Executors are the number of concurrent builds that Jenkins can execute on a node. Each executor on a node can run one job at a time, allowing multiple jobs to be executed in parallel if multiple executors are configured.

Understanding Jenkins Plugins and How They Extend Capabilities

- **Plugins in Jenkins**
    - Jenkins is highly extensible, and its functionality can be expanded through plugins. There are over 1800 plugins available, covering a wide range of needs from source control management to build tools, UI enhancements, and more.
    - Plugins allow integration with external services and tools, customizing Jenkins to fit specific project needs and enhancing its core capabilities.
- **Types of Plugins**
    - **Source Control Plugins** : Allow integration with various version control systems (e.g., Git, SVN).
    - **Build Tools Plugins** : Facilitate tasks such as compiling code or managing dependencies (e.g., Maven, Gradle).
    - **Notification Plugins** : Enable communication with teams via email, Slack, or other messaging services.
    - **UI Plugins** : Enhance the Jenkins interface with additional views or dashboards.

- **Plugin Management**
  - Plugins can be installed and managed through the Jenkins Plugin Manager within the Jenkins UI.
  - Before installing or updating plugins, it is crucial to check compatibility and dependencies to avoid conflicts that could affect the Jenkins environment.

By understanding these core concepts, participants can appreciate how Jenkins structures and processes CI/CD tasks, offering a robust platform for automating various elements of software development and delivery.

Installation and Config

Installation Methods

1. **Local Setup**
   - Download Jenkins from the official website, selecting the appropriate package for your operating system (Windows, macOS, Linux).
   - Ensure a compatible Java version (Java 11 or later) is installed.
2. **Container-Based Setup**
   - Use Docker for a quick and isolated Jenkins environment, useful for easy deployment and scalability.
   - Benefits include simplified dependency management and consistent setup across various development environments.

```
git clone https://github.com/cloudpoet-in/jenkins-tekton-trai

cd jenkins/docker

docker compose up -d
```

Basic Configuration and Initial Setup Tasks

- **Unlock Jenkins** : Access Jenkins via a browser and use the initial admin password provided by the installation process.
- **Plugin Installation** : Choose to install suggested plugins to get started quickly with essential functionalities.
- **Create Admin User** : Set up an admin account for managing Jenkins.
- **Configure Instance** : Set up the Jenkins URL, email notifications, and basic security settings.

Overview of Jenkins UI

- **Dashboard** : Offers an overview of job statuses and system information, allowing for job creation and management.
- **Monitoring** : Provides access to build histories, logs, and basic trends directly in the UI.
- **Configuration** :
  - **Manage Jenkins** : Central hub for global settings, tool configurations, and security.
  - **Job Configuration** : Set build steps, triggers, and environment variables for each job.
  - **Plugin Management** : Install, update, or remove plugins to extend Jenkins' functionality.

This streamlined explanation covers the essential steps and features to help participants get started with Jenkins quickly and effectively.

Important Configuration Parameters in Jenkins

System Configuration

- **Jenkins URL** :

    - **Purpose** : Ensures that notifications and webhooks reference the correct server address.
    - **Configuration** : Set the Jenkins URL under **Manage Jenkins > Configure System** to establish the correct base URL for your Jenkins instance.
- **Environment Variables** :
    - **Purpose** : Define global environment variables for use across all Jenkins jobs.
    - **Configuration** : In **Manage Jenkins > Configure System** , add environment variables applicable to your builds and pipelines.

Security Settings

- **Access Control** :
    - **Purpose** : Manage who can access Jenkins and what they can do.
    - **Configuration** : Use security realms like Jenkins' own user database or integrate with LDAP for user authentication. Define authorization strategies (matrix-based, role-based) to control permissions.
- **Credentials Management** :
    - **Purpose** : Securely store and manage credentials (e.g., SSH keys, API tokens) needed for builds and integrations.
    - **Configuration** : Use the **Manage Jenkins > Manage Credentials** section to securely configure and manage credentials.
- **CSRF Protection** :
    - **Purpose** : Protect against Cross-Site Request Forgery attacks.
    - **Configuration** : Ensure CSRF protection is enabled under **Manage Jenkins > Configure Global Security** .

Build and Job Configuration

- **Build Executors** :
    - **Purpose** : Define the number of concurrent builds that can run.

    - **Configuration** : Adjust the number of executors on your Jenkins master and slave nodes under **Manage Jenkins > Nodes** .
- **Workspace and Build Directory Management** :
    - **Purpose** : Manage disk usage efficiently to prevent space-related issues.
    - **Configuration** : Decide on retention policies for build artifacts and workspaces. You can configure these in each job's settings or globally.

Node Configuration

- **Node Management** :
    - **Purpose** : Configure Jenkins nodes (agents) to distribute build workloads.
    - **Configuration** : Add, remove, and configure nodes via **Manage Jenkins > Manage Nodes and Clouds** . Define labels and the number of executors per node for efficient job distribution.

Plugin Management

- **Plugin Installation and Updates** :
    - **Purpose** : Extend Jenkins functionality and keep up with the latest features and security updates.
    - **Configuration** : Regularly review and update plugins via the **Manage Jenkins > Manage Plugins** interface. Be cautious with compatibility and dependencies when installing plugins.

Backup and Restore

- **Purpose** : Ensure you can recover Jenkins configurations and jobs in case of failures.
- **Configuration** : Implement regular backups of Jenkins home directory and associated configurations. Use job DSL or configuration as code plugins to manage and version control Jenkins configurations.

Logging and Monitoring

- **Logging Level** :
  - **Purpose** : Gather insights into the Jenkins operations and troubleshoot issues.
  - **Configuration** : Adjust the logging level for various components via **Manage Jenkins > System Log** . Consider integrating with external monitoring tools for enhanced visibility.

These parameters and settings form the backbone of a secure, efficient, and manageable Jenkins instance. Properly configuring these elements ensures that Jenkins can effectively handle the demands of your CI/CD workflow, leading to more reliable and consistent software delivery processes.

Creating Your First Jenkins Job

Different Types of Jenkins Jobs

1. **Freestyle Projects**
   - The simplest type of Jenkins job.
   - Useful for straightforward build and deployment tasks with limited complexity.
   - Offers a flexible setup for basic scripting, email notifications, and source control integration.
2. **Pipeline**
   - Allows defining build steps in Jenkinsfile using either Declarative or Scripted syntax.
   - Supports complex workflows involving multiple stages.
   - Provides better version control and visibility of the CI/CD process.
3. **Multi-branch Pipeline**
   - Designed to handle multiple branches of a project in a single workflow.
   - Automatically creates pipelines for individual branches in a repository.
   - Useful for projects with active feature branch development.

Step-by-Step Creation of a Simple Freestyle Project

1. **Create a New Job**
    - Access the Jenkins dashboard and select **“New Item.”**
    - Enter a project name and choose **“Freestyle project”** as the job type.
    - Click **“OK”** to create the job.
2. **Configure Build Settings**
    - Configure the job description and any necessary build triggers (e.g., polling SCM, time-based triggers).
    - Add build steps, such as executing a shell command or calling a build tool (e.g., Maven).
3. **Save the Configuration**
    - Click **“Save”** to commit the job configuration. This will return you to the job dashboard.

Configuring Source Control Integration

1. **Source Control Management (SCM)**
    - Under the job configuration, find the **“Source Code Management”** section.
    - Choose a version control system, such as **Git** .
2. **Configure Repository Details**
    - Enter the repository URL from platforms like GitHub or GitLab.
    - If necessary, configure credentials for private repositories.
3. **Specify Branches**
    - Optionally, specify which branch to build from by entering branch names (e.g., `main`, `develop`).
4. **Save and Build**
    - Once SCM settings are configured, save the job.

- Trigger a build manually from the job dashboard to test the setup.

Creating Freestyle Project

Creating Pipeline project