@theshubhamgour

Jenkins

CI/CD – Continuous integration & continuous delivery

Introduction

What is Jenkins

- Jenkins is an open source continuous integration (CI) and continuous delivery (CD) tool written in Java.
- It's an automation server used to build and deliver software projects
- A major benefit of using jenkins is that it has a lot of plugins available

Why Jenkins?

- Jenkins is the most popular tool to do Continuous Integration and Continuous Delivery of your software.
- It's free and open source.
- Jenkins is still being actively developed .
- It has a strong community with thousands of plugins you can use.

CI/CD

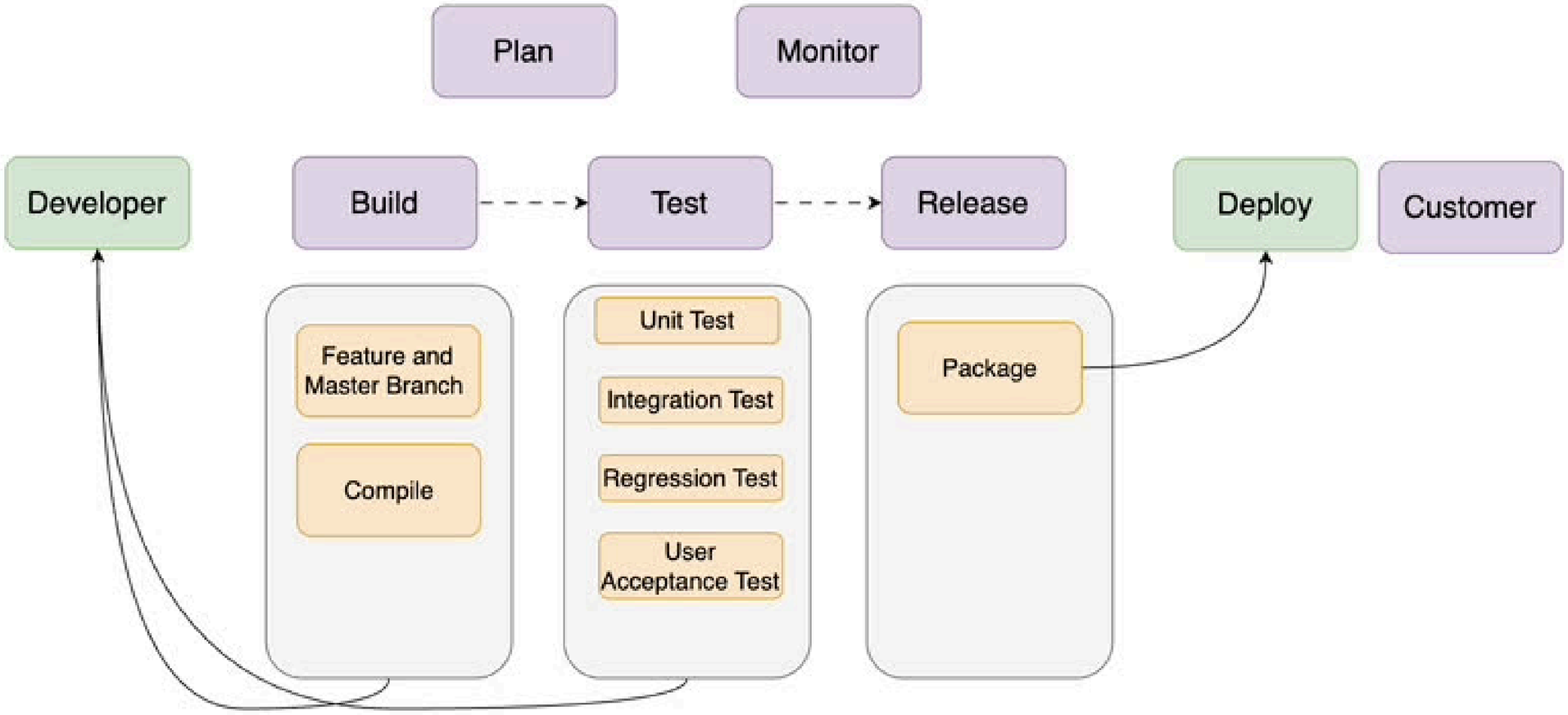
What is CI / CD

- Continuous integration (CI) is the practice, in software engineering, of merging all developer working copies to a shared mainline several times a day.
- Continuous delivery (CD) is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time.
- In Practice: verify and publish work by triggering automated builds & tests · For every commit or at least once a day

What is CI / CD

- All developers should push their changes to a version control, which should then be built and tested – which can be done by Jenkins
- Jenkins doesn't merge code nor it resolves code conflicts, that's still for the developer to do (using for instance git and merge tools)

CI / CD with SDLC



Installation

Explore Jenkins

@theshubhamgour

Jenkins

New Item

Build History

Build Queue

No builds in the queue.

Build Executor Status 0/2

Add description

Welcome to Jenkins!

This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.

Start building your software project

Create a job

Set up a distributed build

Set up an agent

Configure a cloud

Learn more about distributed builds

REST API Jenkins 2.516.2

@theshubhamgour

Jenkins / New Item

Enter an item name

» This field cannot be empty, please enter a valid name

Field where you must provide a unique name for your Jenkins job.

Select an item type

Freestyle project
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

Classic, general-purpose job.

Pipeline

Defines the entire build process as code (Jenkinsfile).

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Used when you need to test the same project on multiple environments/configurations (e.g., different OS, Java versions, browsers).

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

Creates a separate namespace to avoid name conflicts.

Multibranch Pipeline

Automatically creates pipelines for each branch in a repository.

Creates a set of Pipeline projects according to detected branches in one SCM repository.

Organization Folder

Automatically creates subfolders and multibranch pipelines for repositories.

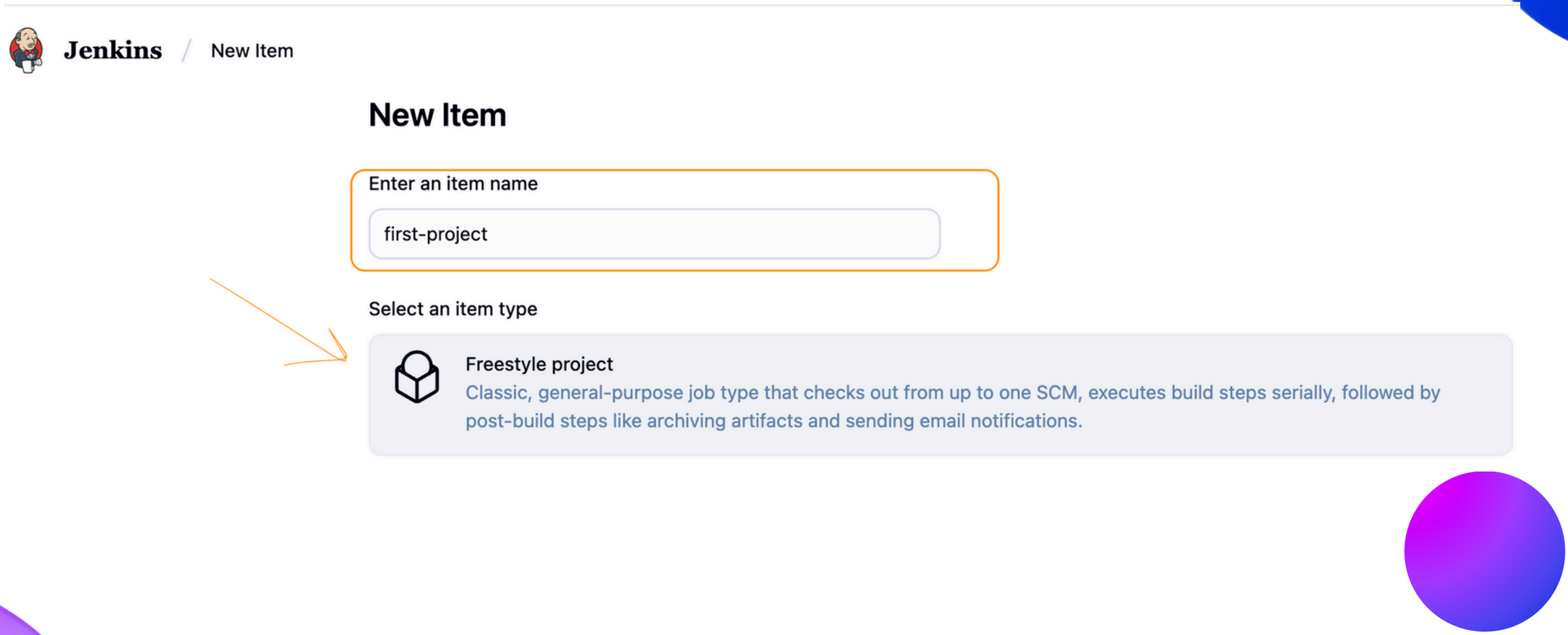
Creates a set of multibranch project subfolders by scanning for repositories.

OK

Demo

Creating first job

@theshubhamgour



General

Enabled

Description

Description for this Jenkins Freestyle job

Plain text Preview

- [ ] Discard old builds ? — This will discard the old builds
- [ ] GitHub project — When you want to make use of a github repo
- [ ] This project is parameterized ? — When you wish to take input from user during execution
- [ ] Throttle builds ? — Limiting number of build execution based on time
- [ ] Execute concurrent builds if necessary ? — Allowing to execute multiple jobs parallely

Advanced

Source Code Management

Connect and manage your code repository to automatically pull the latest code for your builds.

- None — Don’t link any repository.
- Git ? — Connect to a Git repo for pulling code.

Triggers

Set up automated actions that start your build based on specific events, like code changes or scheduled times.

- Trigger builds remotely (e.g., from scripts) ? — Start build via API/script.
- Build after other projects are built ? — Run after dependent jobs finish.
- Build periodically ? — Schedule builds with cron.
- GitHub hook trigger for GITScm polling ? — Build on GitHub push event.
- Poll SCM ? — Check repo at intervals for changes.

Environment

Configure settings and variables that define the context in which your build runs, like credentials, paths, and global parameters.

- [ ] Delete workspace before build starts — Clean files before build.
- [ ] Use secret text(s) or file(s) ? — Inject credentials securely.
- [ ] Add timestamps to the Console Output — Show time for each log line.
- [ ] Inspect build log for published build scans — Check logs for scan results.
- [ ] Terminate a build if it's stuck — Auto-stop hanging builds.
- [ ] With Ant ? — Run Ant builds using build.xml.

Build Steps

Automate your build process with ordered tasks like code compilation, testing, and deployment.

Add build step — Define tasks to execute during the build.

Post-build Actions

Define what happens after a build completes, like sending notifications, archiving artifacts, or triggering other jobs.

Add post-build action — Define tasks to execute after the build.

Save Apply

@theshubhamgour

Jenkins freestyle job (demonstration)

What are Jenkins Plugins?

- Plugins in Jenkins are add-ons that extend its core functionality.
- Think of Jenkins as a lightweight automation engine.
- Out of the box, it only has limited features.
- With plugins, you can add support for things like Git, Docker, notifications, different build tools, and even UI improvements.

- It keeps the core system small and stable.
- You only install what you actually need.
- This makes Jenkins flexible enough to fit any CI/CD use case – from small projects to enterprise pipelines.

⚠️ A few important things to remember when working with plugins

- Keep plugins updated – new versions fix bugs and security issues.
- Avoid unnecessary plugins – too many plugins can slow Jenkins and cause conflicts.
- Be careful of outdated plugins – old plugins might have vulnerabilities that attackers can exploit.

What is pipeline??

What is Jenkins Pipeline?

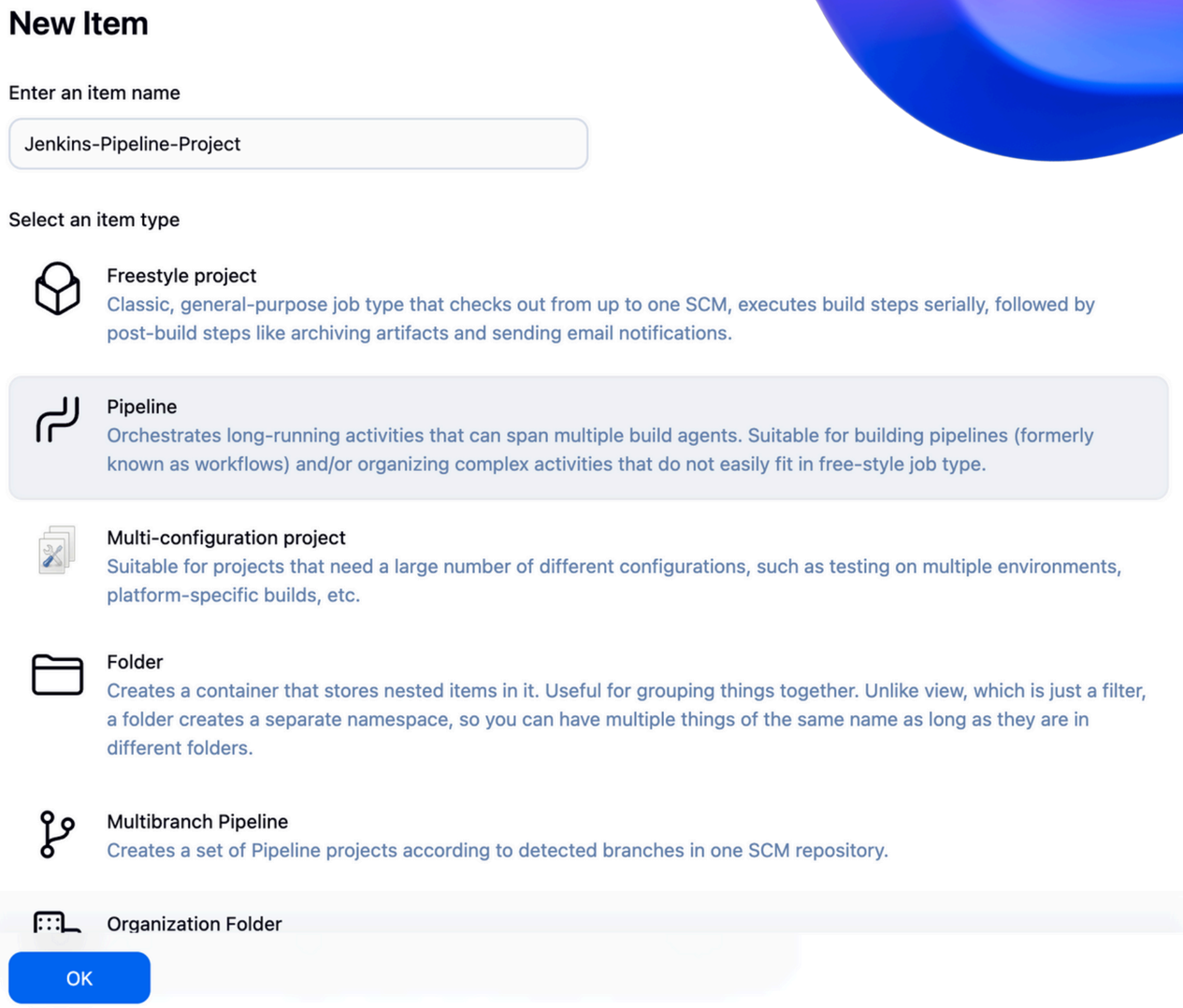
- A pipeline is a set of automated processes in Jenkins to build, test, and deploy applications.
- Helps in implementing CI/CD (Continuous Integration & Continuous Delivery).
- Written as Pipeline as Code using a Jenkinsfile.

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                echo 'Building...'
            }
        }
        stage('Test') {
            steps {
                echo 'Testing...'
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploying...'
            }
        }
    }
}
```

@theshubhamgour

Why Use Pipeline?

- Automates software delivery process.
- Provides consistency & repeatability.
- Easy to version control (Jenkinsfile stored in Git).
- Supports complex workflows.



Types of Pipelines in Jenkins

- Declarative Pipeline – Simple & structured (recommended).
- Scripted Pipeline – More flexibility, written in Groovy.

```
node {
    stage('Build') {
        echo "Building..."
    }
    stage('Test') {
        echo "Testing..."
    }
    stage('Deploy') {
        echo "Deploying..."
    }
}
```

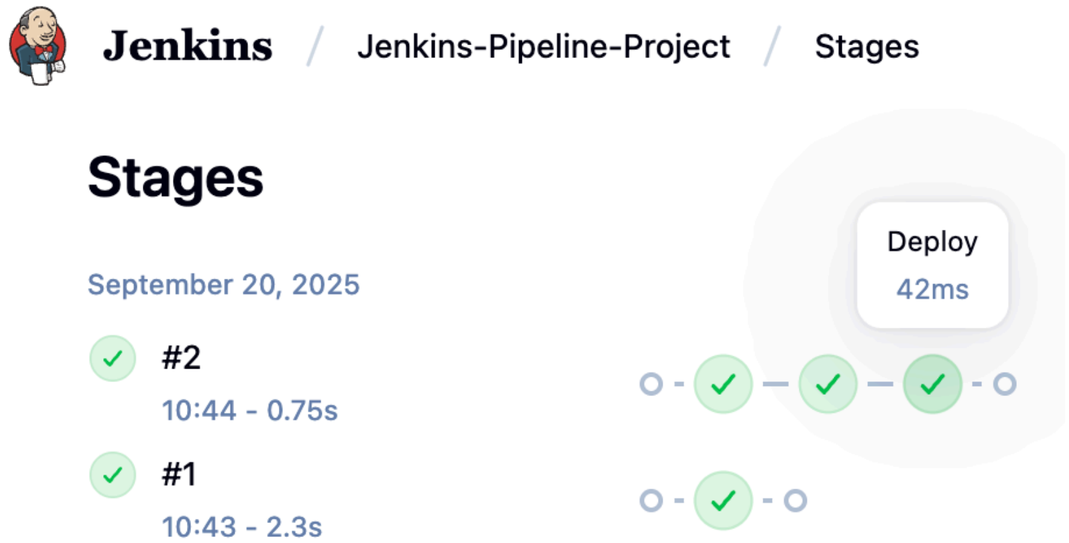
Scripted Pipeline

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                echo 'Building...'
            }
        }
        stage('Test') {
            steps {
                echo 'Testing...'
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploying...'
            }
        }
    }
}
```

Declarative Pipeline

Jenkinsfile Example (Declarative)

- pipeline {} defines pipeline.
- stages {} contain multiple steps.
- Runs build → test → deploy automatically.




```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                echo 'Building...'
            }
        }
        stage('Test') {
            steps {
                echo 'Testing...'
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploying...'
            }
        }
    }
}
```

Stages in a Pipeline

- Build – Compile code, install dependencies.
- Test – Run unit/integration tests.
- Deploy – Push code to servers/containers/cloud.

Definition

Pipeline script

Script ?

```
3    stages {
4        stage('Build') {
5            steps {
6                echo 'Building...'
7            }
8        }
9        stage('Test') {
10           steps {
11               echo 'Testing...'
12           }
13       }
14       stage('Deploy') {
15           steps {
16               echo 'Deploying...'
17           }
```

Use Groovy Sandbox ?

Save Apply

Pipeline Benefits

- ✅ Code as pipeline (Jenkinsfile)
- ✅ Reusable & shareable
- ✅ Integrates with Git, Docker, Kubernetes, etc.
- ✅ Error handling & notifications

Real-world CI/CD Flow

- 1. Developer commits code to GitHub.
- 2. Jenkins pipeline triggers.
- 3. Build & test stages run.
- 4. Deploy to staging/production.

Conclusion

- Jenkins Pipeline = Automation backbone of CI/CD.
- Provides flexibility, scalability, and better collaboration.
- Future: Integrates with AI/ML, Kubernetes, Cloud DevOps.

@theshubhamgour

Declarative and Scripted Pipeline

Declarative Pipeline

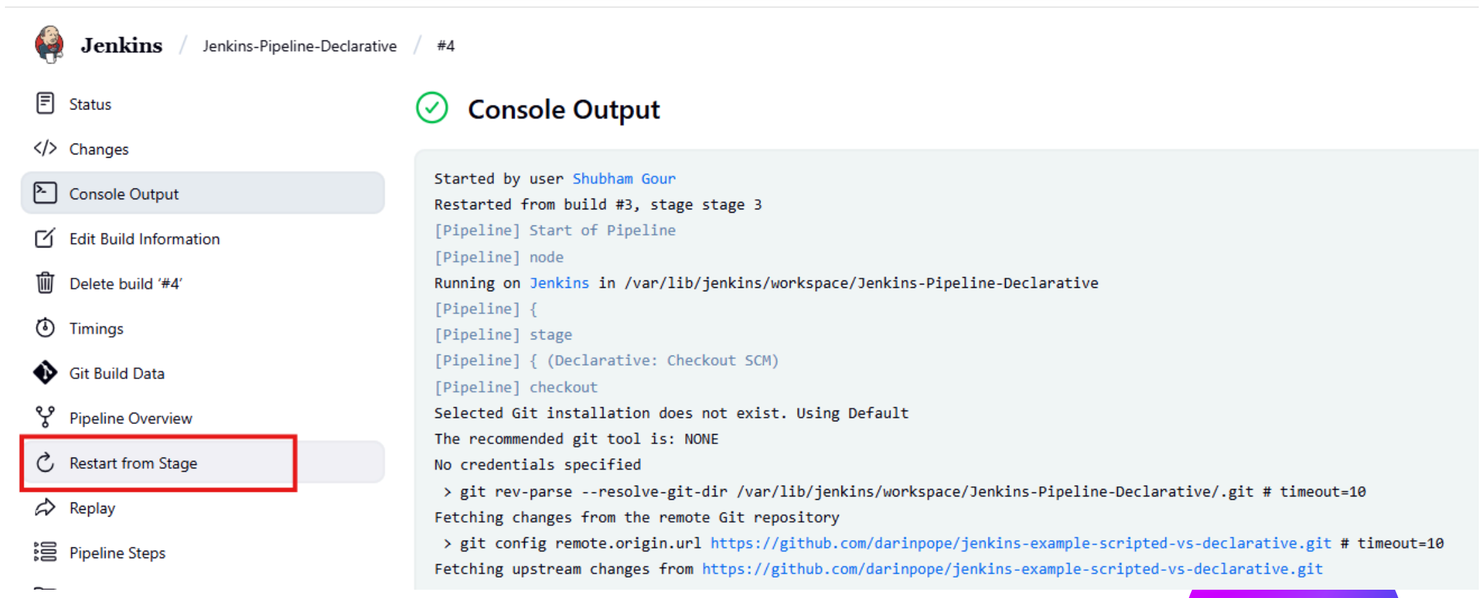
- Easy, structured, beginner-friendly.
- Like filling a form with fixed sections.
- Easy to read, less chance of mistakes.

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                echo 'Building...'
            }
        }
        stage('Test') {
            steps {
                echo 'Testing...'
            }
        }
    }
}
```

@theshubhamgour

Declarative Pipeline

Benefit of using declarative pipeline is we can start from any stage we want



@theshubhamgour

Declarative Pipeline

```
Started by user Shubham Gour
Obtained 02-restart/Jenkinsfile-declarative from git https://github.com/darinpope/jenkins-example-scripted-vs-declarative.git
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/Jenkins-Pipeline-Declarative
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
No credentials specified
 > git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/Jenkins-Pipeline-Declarative/.git # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/darinpope/jenkins-example-scripted-vs-declarative.git # timeout=10
Fetching upstream changes from https://github.com/darinpope/jenkins-example-scripted-vs-declarative.git
 > git --version # timeout=10
 > git --version # 'git version 2.43.0'
 > git fetch --tags --force --progress -- https://github.com/darinpope/jenkins-example-scripted-vs-declarative.git +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision 79dab97a2f8bc027318334a8d14f0da046146d31 (refs/remotes/origin/main)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 79dab97a2f8bc027318334a8d14f0da046146d31 # timeout=10
Commit message: "rename"
 > git rev-list --no-walk 79dab97a2f8bc027318334a8d14f0da046146d31 # timeout=10
[Pipeline] }
[Pipeline] // stage
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Hello)
[Pipeline] sh
+ echo Hello World
Hello World
```

```
+ echo Hello World
Hello World
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (stage 2)
[Pipeline] sh
+ echo Hello World 2
Hello World 2
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (stage 3)
[Pipeline] sh
+ echo Hello World 3
Hello World 3
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Scripted Pipeline

- Like writing your own story – you decide everything.
- Written in Groovy language.

```
node {
    stage('Build') {
        echo 'Building...'
    }
    stage('Test') {
        echo 'Testing...'
    }
}
```

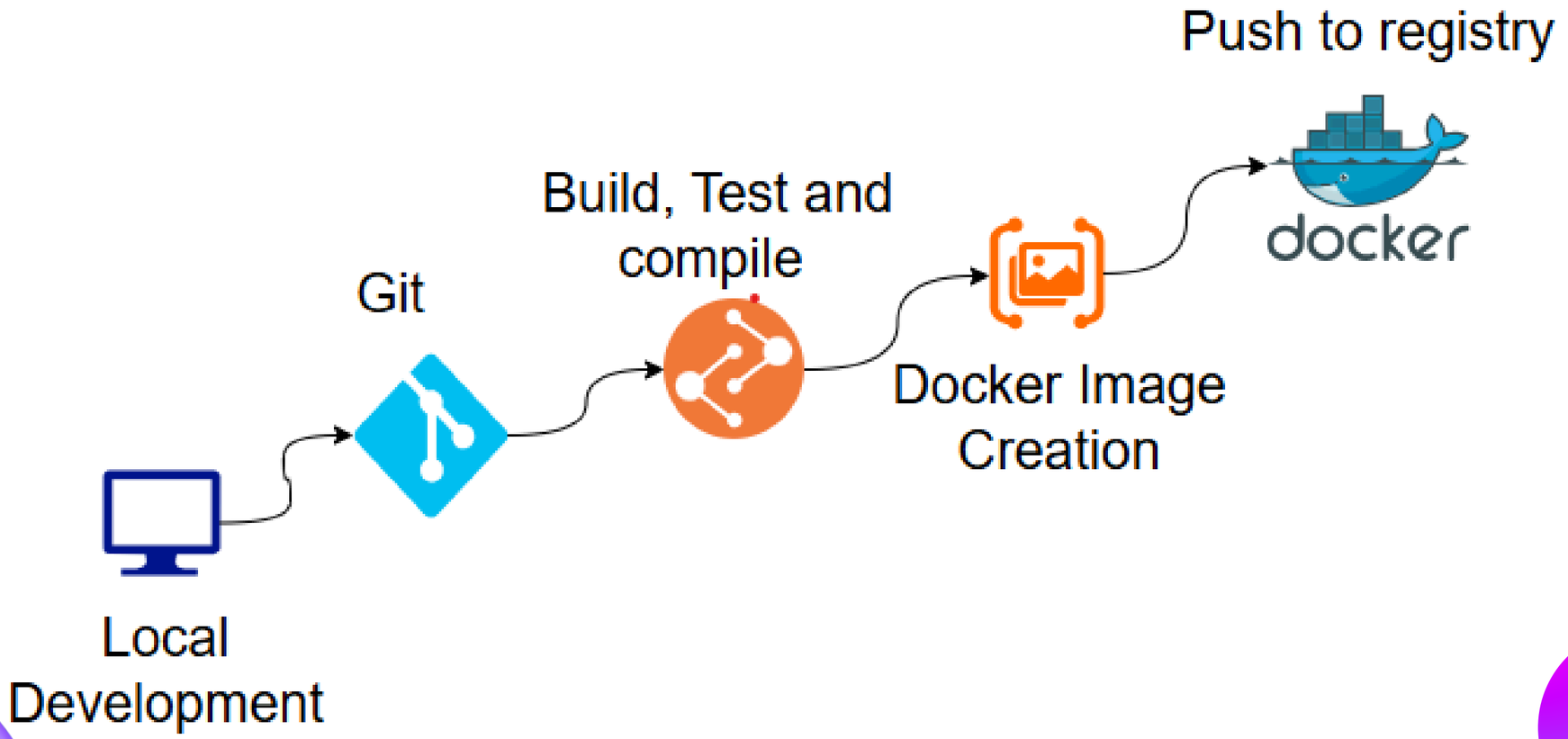
@theshubhamgour

Scripted Pipeline

```
Started by user Shubham Gour
Obtained 02-restart/Jenkinsfile-scripted from git https://github.com/darinpope/jenkins-example-scripted-vs-declarative
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/Jenkins-Pipeline-Scripted
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Hello)
[Pipeline] sh
+ echo Hello World
Hello World
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (stage 2)
[Pipeline] sh
+ echo Hello World 2
Hello World 2
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (stage 3)
[Pipeline] sh
+ echo Hello World 3
Hello World 3
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
```

Demo

Build and deploy the first app push to registry


Push to registry
docker
Build, Test and
compile
Git
Docker Image
Creation
Local
Development

Agents

What is a Jenkins Agent?

- A Jenkins Agent is a worker machine where the jobs (build, test, deploy) actually run.
- Jenkins itself has 2 parts:
- Master/Controller – Brain 🧠 (manages jobs).
- Agent – Hands 🖐 (executes jobs).

Why Agents?

- The controller cannot do everything alone.

- Agents help by:
  - ✅ Distributing workload
  - ✅ Running jobs on different OS (Linux, Windows, Mac)
  - ✅ Running jobs in parallel
  - ✅ Scaling for large projects

Types of Agents

- Built-in Agent – Runs on the same machine as Jenkins.
- Static Agent – A fixed machine connected to Jenkins.
- Dynamic Agent – Created on demand (e.g., Docker, Kubernetes, Cloud)

Types of Agents

- Jenkins Agent = Worker machine for running jobs.
- Helps in scaling, flexibility, and distributing workload.
- Can be static, dynamic, or built-in.

Agents (Demonstration)

@theshubhamgour

Jenkins Credentials Management

Securely Handling Secrets in Pipelines

What is Credentials Management?

- A secure way to store secrets (like passwords, API keys, tokens, SSH keys) in Jenkins.
- Instead of writing them directly in the pipeline, store them in Jenkins.
- Jenkins will provide them to the pipeline safely and only when needed.

Why Do We Need It?

- ⚠️ Hardcoding passwords in pipelines = security risk.
- ✅ Credentials Manager protects sensitive info.
- ✅ Centralized storage of all secrets.
- ✅ Easier to update/change without editing pipeline code.

Types of Credentials in Jenkins

- Username with Password
- Secret Text (API keys, tokens)
- SSH Keys
- Certificates
- Secret Files

Best Practices

- ✅ Always store secrets in Credentials Manager.
- ✅ Never hardcode passwords or tokens in pipelines.
- ✅ Use descriptive IDs for credentials (e.g., aws-prod-key).
- ✅ Restrict access to credentials (RBAC).
- ✅ Rotate credentials regularly.

Summary

- Credentials Management = Safe handling of secrets.
- Supports multiple credential types.
- Integrated with pipelines using withCredentials.
- Essential for secure DevOps pipelines.

@theshubhamgour

Jenkins Environment Variables

What are Environment Variables?

- Special variables used in Jenkins to store and reuse values.
- Examples: paths, credentials, project names, Docker image tags.
- Helps avoid hardcoding values in the pipeline.
- Makes pipeline secure, reusable, and easy to manage.

Why Use Environment Variables?

- Store sensitive information (username, password).
- Avoid repeating values everywhere.
- Change value in one place → applies everywhere.
- Works across multiple stages of the pipeline.

Jenkins Built-in Environment Variables

- BUILD_ID → Unique build number.
- JOB_NAME → Name of the Jenkins job.
- WORKSPACE → Directory where code is checked out.
- BUILD_URL → URL of the running build.

Summary

- Environment variables make pipelines flexible & secure.
- Use them for credentials, tags, project names, and paths.
- Combine with Credentials Manager for production-ready pipelines.

@theshubhamgour

RBAC in Jenkins

Role-Based Access Control

What is RBAC in Jenkins (Role-Based Access Control)?

- RBAC (Role-Based Access Control) is a security model that defines user permissions based on roles rather than individuals.

⚙️ Why Use RBAC in Jenkins?

- Maintain security and prevent unauthorized access.
- Control who can view, build, or configure jobs.
- Simplify management by grouping users with similar responsibilities.

🔐 Best Practices

- Apply least privilege principle.
- Regularly review roles and access.
- Use LDAP or Active Directory for user management.
- Backup Jenkins configuration.

@theshubhamgour

Webhook in Jenkins

github-webhook

What is Webhook?

A webhook is an automatic HTTP request sent from one system to another when an event occurs.

In this case:

- Source: GitHub
- Destination: Jenkins
- Used to trigger builds or deployments automatically.

Example :

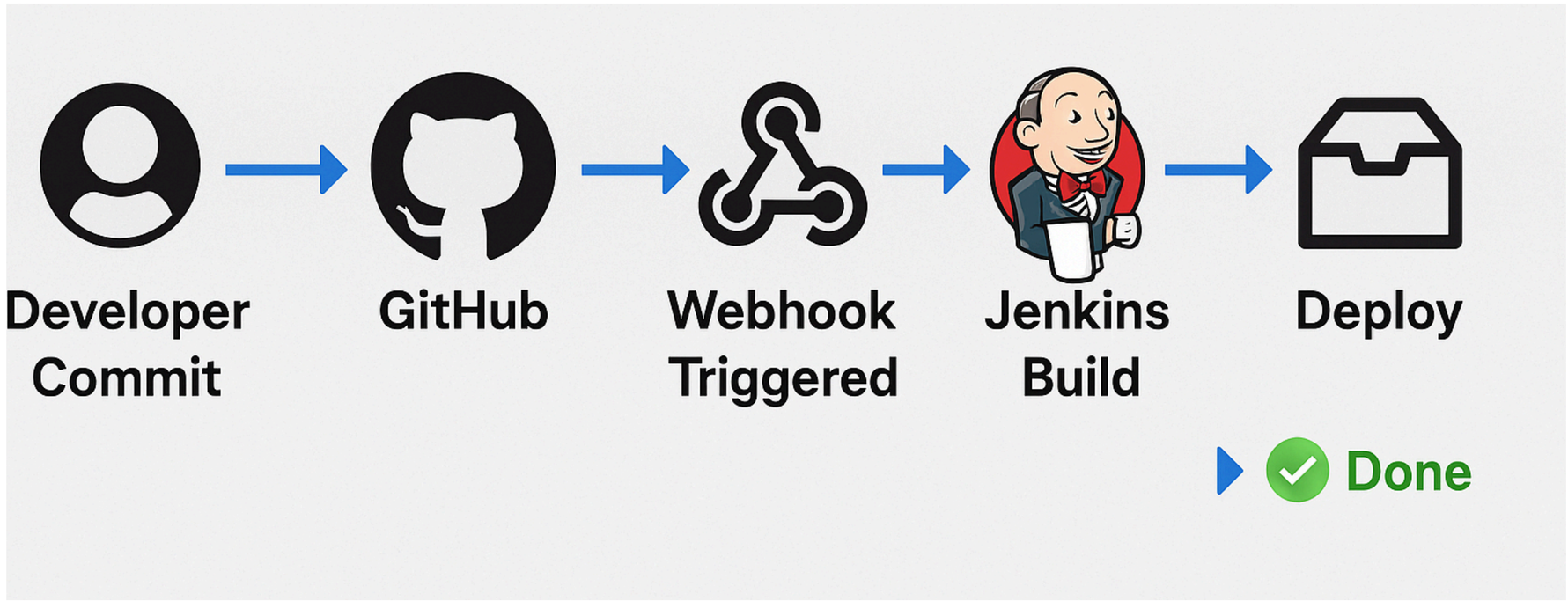
When a developer pushes code to GitHub, the webhook notifies Jenkins — and Jenkins starts a new build.

Why Use Webhooks in Jenkins?

- Enable Continuous Integration (CI) – automatic builds on every push.
- Speed up feedback loop for developers.
- Reduce manual effort – no need to trigger jobs manually.
- Real-time synchronization between GitHub and Jenkins.

Example :

When a developer pushes code to GitHub, the webhook notifies Jenkins – and Jenkins starts a new build.


Developer Commit
GitHub
Webhook Triggered
Jenkins Build
Deploy
Done

@theshubhamgour

Shared libraries

in jenkins

What Are Jenkins Shared Libraries?

- Reusable Groovy code for pipelines
- Helps avoid repetitive code
- Makes pipelines cleaner & modular
- Perfect for teams working on multiple projects
- Stored in a Git repo (GitHub, GitLab, Bitbucket)

Why Use Shared Libraries?

- Reduce duplication in Jenkinsfiles
- Centralized logic → easy updates
- Faster development
- Standardized functions for all teams
- One change affects all pipelines

Structure of a Shared Library

```swift
(root)
 ├── vars/
 │     └── myFunction.groovy
 ├── src/
 │     └── com/company/ (optional)
 ├── resources/ (optional)
 └── README.md
```

What Goes in vars/ ?

- Global pipeline functions
- Each .groovy file = one function

```
def call() {
    // your code
}
```

Best Practices

- Keep functions small
- Add proper logs
- Use src/ for advanced code
- Version your library
- Add README for every function

@theshubhamgour

Multibranch Pipeline

in jenkins

What is a Multibranch Pipeline?

- A Jenkins job that scans your Git repo
- Automatically creates jobs for each branch
- Runs Jenkinsfile from that branch
- No manual job creation required

Why Use Multibranch Pipelines?

- CI/CD for every branch
- Auto-detect new branches
- Auto-remove deleted branches
- Developers work independently
- Perfect for GitFlow & feature branching

Advantages

- No manual pipeline duplication
- No hardcoded branch names
- Works well with PRs
- Faster dev cycles
- More automation, fewer mistakes

Thank You

@theshubhamgour

https://theshubhamgour.hashnode.dev