

LAB ASSIGNMENT NO-06

Aim: To build Pipeline using Jenkins.

Theory:

Jenkins Pipeline:

Jenkins Pipeline is a powerful automation tool for building, testing, and deploying software. It uses a domain-specific language called Groovy to define workflows as code. Here's a brief overview of Jenkins Pipeline in 7 lines:

Jenkins Pipeline is a suite of plugins that enables you to define, manage, and automate your CI/CD processes within Jenkins.

It allows you to define your build, test, and deployment steps as code using the "Jenkinsfile," which can be stored in your version control system.

Jenkins Pipelines can be written in two ways: Declarative Pipeline (simplified, structured syntax) and Scripted Pipeline (more flexible, but complex).

Pipelines support parallel execution, enabling efficient use of resources and faster build and test cycles. You can use various built-in and custom plugins to integrate with tools and services like Git, Docker, AWS, and more. Pipeline stages, steps, and post-build actions can be defined to create a structured and customizable workflow. Jenkins Pipeline offers visibility into the entire CI/CD process, making it easier to troubleshoot, monitor, and scale your automated workflows.

Continuous delivery of jenkins pipelines and how it works:

Continuous Delivery (CD) in Jenkins Pipelines is a practice that focuses on automating and streamlining the process of deploying software to production or other target environments in a reliable and consistent manner. Jenkins Pipelines provide a flexible and powerful way to implement CD by defining the entire deployment process as code. Here's how it works in brief:

Code Development: Developers write and commit code changes to a version control system (e.g., Git).

Version Control Integration: Jenkins is configured to monitor the version control system for changes. When changes are detected, Jenkins triggers a pipeline job.

Pipeline Definition: The CD process is defined as a Jenkins Pipeline script, typically written in Groovy, and stored in a "Jenkinsfile" in the project's code repository. The Jenkinsfile specifies all the stages and steps required for CD, including building, testing, and deploying the application.

Build and Test: The pipeline starts by building the code, running automated tests, and performing quality checks. If any of these stages fail, the pipeline stops, and

notifications are sent to the team.

Artifact Creation: Successful builds create deployable artifacts, such as compiled binaries, Docker images, or packages, which are stored in an artifact repository or other designated location.

Deployment Stages: The pipeline defines one or more deployment stages, which may include deploying to development, staging, and production environments. Each stage can have pre-deployment and post-deployment actions, including infrastructure provisioning, database migrations, or configuration updates.

Manual Approvals: For certain stages (e.g., production), manual approvals may be required to ensure that changes are validated by stakeholders before proceeding.

Rollbacks: The pipeline should also include rollback mechanisms to revert to a previous version in case of deployment failures or issues in production.

Monitoring and Testing in Production: After deployment, the application should be monitored in the production environment to detect issues. Automated tests and health checks can be part of the pipeline to validate the deployment's success.

Continuous Delivery with Jenkins Pipelines promotes automation, collaboration, and consistency throughout the software development lifecycle, resulting in faster and more reliable software releases.

Declaration and scripted pipeline syntax in Jenkins:

1. Declarative pipeline-

```
pipeline {
  agent any

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

2. Scripted Pipeline:

```
node {  
    stage('Build') {  
        echo 'Building...'  
        // Custom build logic here  
    }  
  
    stage('Test') {  
        echo 'Testing...'  
        // Custom test logic here  
    }  
  
    stage('Deploy') {  
        echo 'Deploying...'  
        // Custom deployment logic here  
    }  
}
```

Benefits of Jenkins pipeline:

Jenkins Pipeline offers several benefits that make it a popular choice for managing and automating Continuous Integration and Continuous Delivery (CI/CD) processes. Here are some key benefits of Jenkins Pipeline:

Pipeline as Code: Jenkins Pipeline allows you to define your CI/CD workflows as code (in a "Jenkinsfile"), which can be versioned and stored in your source code repository. This promotes the principles of Infrastructure as Code (IaC) and enables easy collaboration and code reviews.

Reproducibility: Pipelines provide a consistent and reproducible way to build, test, and deploy software. Since the pipeline is defined as code, it ensures that every build and deployment follows the same steps, reducing the risk of configuration drift and inconsistencies.

Flexibility: Jenkins Pipeline supports both Declarative and Scripted syntax, providing flexibility for simple and complex workflows. You can choose the level of abstraction that best suits your needs.

Parallel Execution: Pipelines can execute multiple stages and steps in parallel, which can significantly reduce build and test times, making your CI/CD process more efficient.

Integration with Tools: Jenkins has a vast ecosystem of plugins, and Jenkins Pipeline can easily integrate with various tools, including source code repositories (e.g., Git), build tools, testing frameworks, deployment platforms,

and notification services. This flexibility allows you to create end-to-end automation pipelines.

Output:

The screenshot shows the Jenkins Dashboard at localhost:8080. The left sidebar contains links to 'view item', 'People', 'Build History', 'Manage Jenkins', and 'My Views'. The main area displays the 'Build Queue' and 'Build Executor Status'.

Build Queue

No builds in the queue.

Build Executor Status

S	W	Name	Last Success	Last Failure	Last Duration
✓	☁	java-sanika	6 days 23 hr #2	6 days 23 hr #1	1.8 sec
✓	☁	java-shreya-t11	7 days 0 hr #2	7 days 0 hr #1	1.7 sec
✓	☁	java-shreya-t11-09	7 days 0 hr #2	7 days 0 hr #1	1 sec
✓	☀	java-t11-09	6 days 23 hr #1	N/A	11 sec
✓	☀	myfirstpipeline	28 min #1	N/A	6.3 sec
✗	☁	pipeline-shreya-t11	N/A	10 min #1	29 ms
✓	☁	Shreya-09-t11	7 days 0 hr #2	7 days 0 hr #1	0.72 sec

Icon: S M L | Icon legend | Atom feed for all | Atom feed for failures | Atom feed for just latest builds

The screenshot shows the Jenkins Pipeline configuration page for 'myfirstpipeline'. The left sidebar contains links to 'Status', 'Changes', 'Build Now', 'Configure', 'Delete Pipeline', 'Full Stage View', 'Rename', and 'Pipeline Syntax'. The main area displays the 'Pipeline myfirstpipeline' configuration.

Pipeline myfirstpipeline

first pipeline

[Plain text] Preview

Save

Disable Project

Stage View

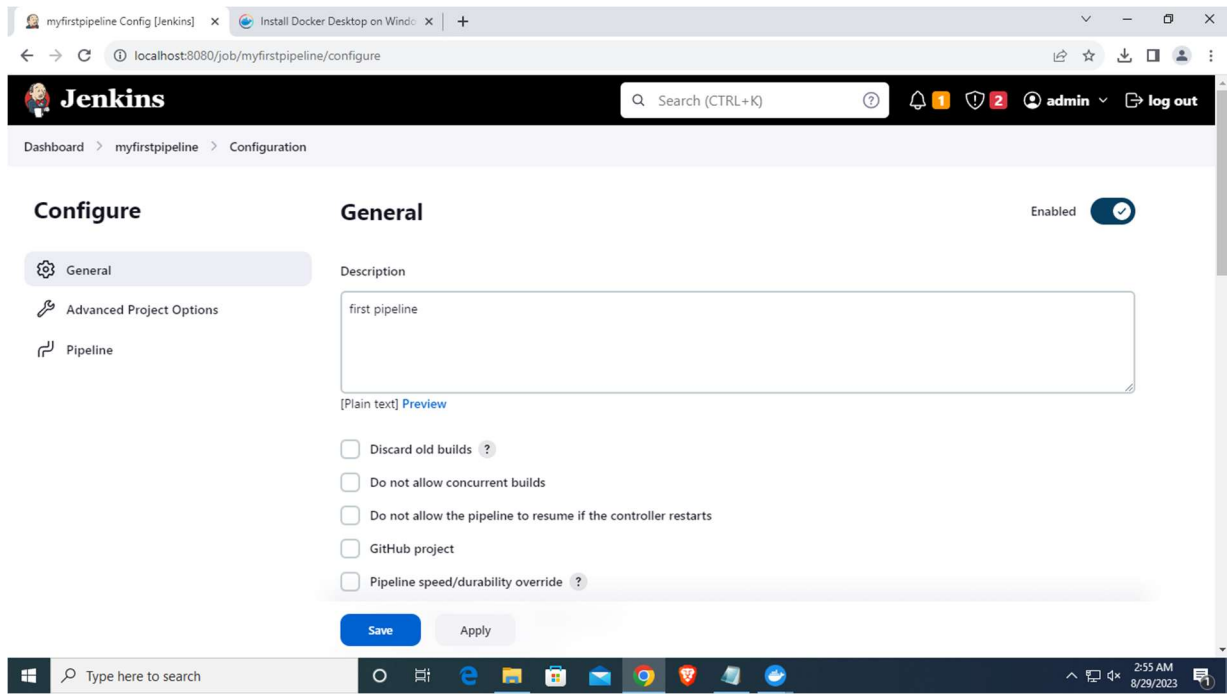
Build History trend

Filter builds...

Aug 29, 2023, 2:26 AM

Average stage times: 540ms (Average full run time: ~6s)

Hello



This screenshot shows the Jenkins configuration page for a job named 'myfirstpipeline'. The 'General' tab is selected in the left sidebar. The main area is titled 'General' and shows the job is 'Enabled'. The 'Description' field contains the text 'first pipeline'. Below this, there are several unchecked checkboxes: 'Discard old builds', 'Do not allow concurrent builds', 'Do not allow the pipeline to resume if the controller restarts', 'GitHub project', and 'Pipeline speed/durability override'. At the bottom of the configuration area are 'Save' and 'Apply' buttons. The browser's address bar shows 'localhost:8080/job/myfirstpipeline/configure'. The Windows taskbar at the bottom shows the time as 2:55 AM on 8/29/2023.

myfirstpipeline Config [Jenkins] x Install Docker Desktop on Wind... x +

localhost:8080/job/myfirstpipeline/configure

Jenkins Search (CTRL+K) admin log out

Dashboard > myfirstpipeline > Configuration

Configure

General

Advanced Project Options

Pipeline

General Enabled

Description

first pipeline

[Plain text] Preview

☐ Discard old builds ?

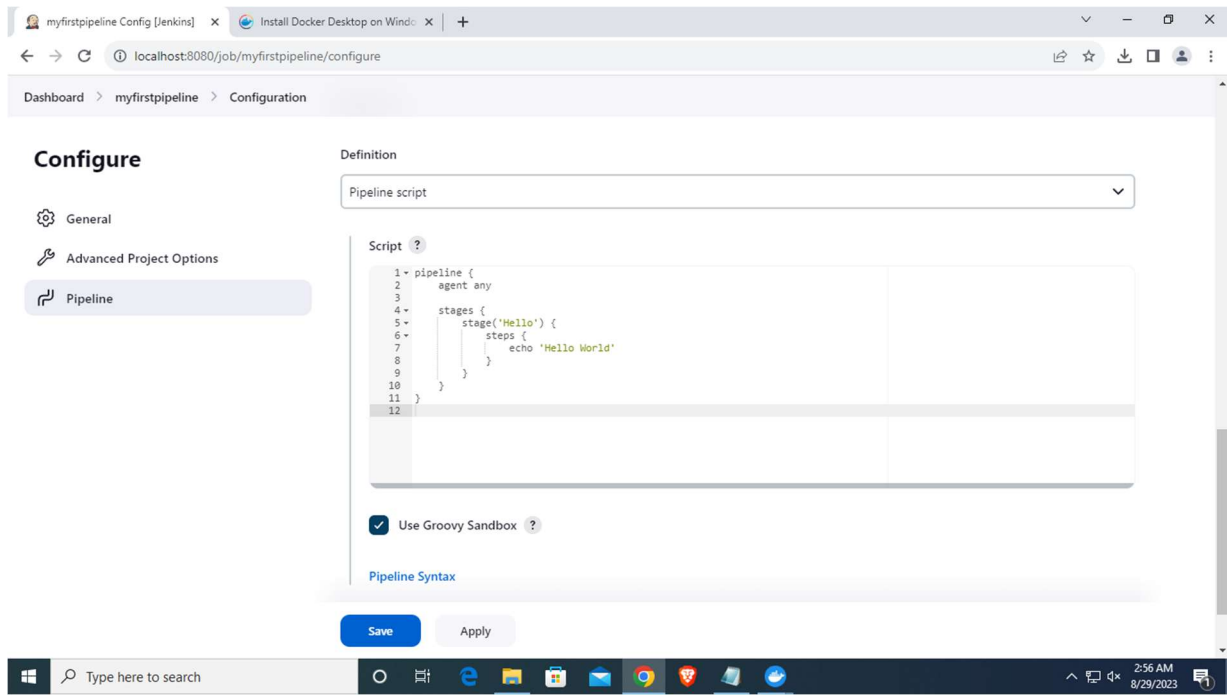
☐ Do not allow concurrent builds

☐ Do not allow the pipeline to resume if the controller restarts

☐ GitHub project

☐ Pipeline speed/durability override ?

Save Apply



This screenshot shows the same Jenkins configuration page, but with the 'Pipeline' tab selected in the left sidebar. The 'Definition' section shows 'Pipeline script' selected from a dropdown menu. Below this, the 'Script' field contains a Groovy pipeline script. The script defines a pipeline with an agent 'any', a stage named 'Hello' with a step that echoes 'Hello World'. The 'Use Groovy Sandbox' checkbox is checked. At the bottom are 'Save' and 'Apply' buttons. The browser's address bar and Windows taskbar are the same as in the first screenshot.

myfirstpipeline Config [Jenkins] x Install Docker Desktop on Wind... x +

localhost:8080/job/myfirstpipeline/configure

Jenkins Search (CTRL+K) admin log out

Dashboard > myfirstpipeline > Configuration

Configure

General

Advanced Project Options

Pipeline

Definition

Pipeline script

Script ?

```
1 pipeline {
2   agent any
3
4   stages {
5     stage('Hello') {
6       steps {
7         echo 'Hello World'
8       }
9     }
10  }
11 }
12
```

☒ Use Groovy Sandbox ?

Pipeline Syntax

Save Apply

The screenshot shows the Jenkins web interface in a browser window. The address bar indicates the URL is `localhost:8080/job/myfirstpipeline/2/console`. The Jenkins logo and search bar are at the top. The left sidebar contains navigation links: Status, Changes, Console Output (selected), View as plain text, Edit Build Information, Delete build '#2', Restart from Stage, Replay, Pipeline Steps, Workspaces, and Previous Build. The main area is titled 'Console Output' with a green checkmark icon. The output text shows a successful pipeline execution:

```
Started by user admin
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in C:\ProgramData\Jenkins\.jenkins\workspace\myfirstpipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Hello)
[Pipeline] echo
Hello World
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

The Windows taskbar at the bottom shows the time as 2:56 AM on 8/29/2023.

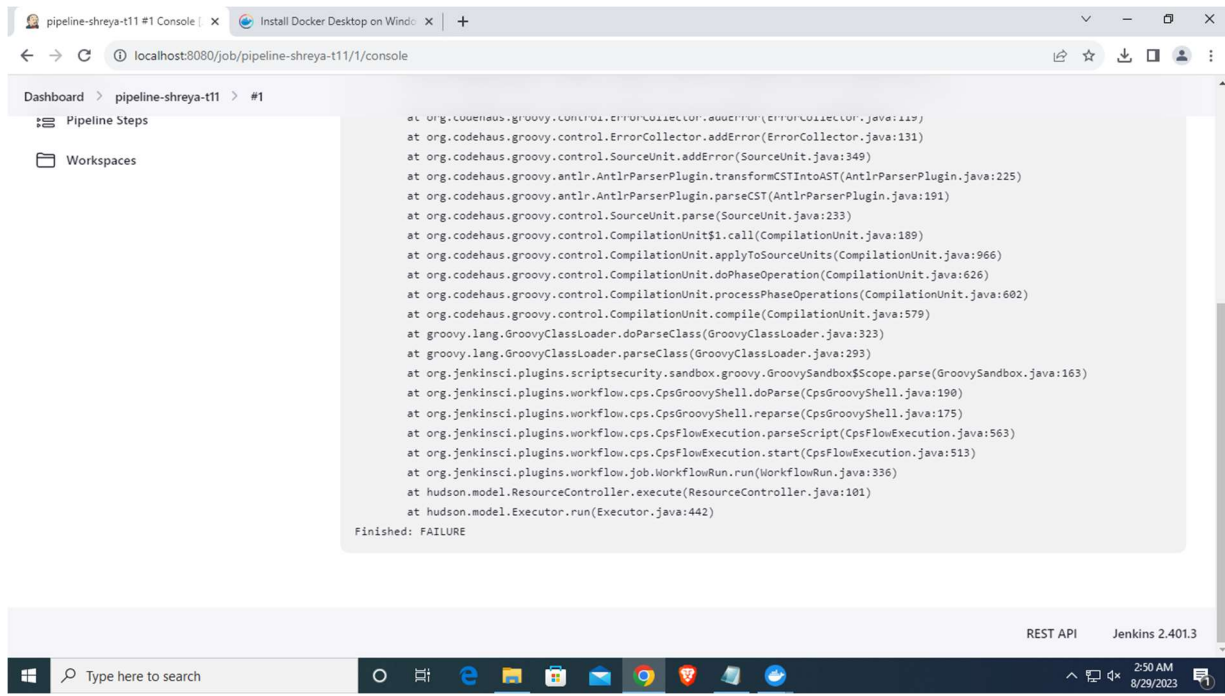
The screenshot shows the Jenkins web interface for a failed pipeline run. The address bar indicates the URL is `localhost:8080/job/pipeline-shreya-t11/1/console`. The left sidebar is similar to the first screenshot, with 'Console Output' selected. The main area is titled 'Console Output' with a red 'X' icon. The output text shows a startup failure:

```
Started by user admin
org.codehaus.groovy.control.MultipleCompilationErrorsException: startup failed:
WorkflowScript: 8: unexpected token: } @ line 8, column 13.
    }
    ^

1 error

at org.codehaus.groovy.control.ErrorCollector.failIfErrors(ErrorCollector.java:309)
at org.codehaus.groovy.control.ErrorCollector.addFatalError(ErrorCollector.java:149)
at org.codehaus.groovy.control.ErrorCollector.addError(ErrorCollector.java:119)
at org.codehaus.groovy.control.ErrorCollector.addError(ErrorCollector.java:131)
at org.codehaus.groovy.control.SourceUnit.addError(SourceUnit.java:349)
at org.codehaus.groovy.antlr.AntlrParserPlugin.transformCSTIntoAST(AntlrParserPlugin.java:225)
at org.codehaus.groovy.antlr.AntlrParserPlugin.parseCST(AntlrParserPlugin.java:191)
at org.codehaus.groovy.control.SourceUnit.parse(SourceUnit.java:233)
at org.codehaus.groovy.control.CompilationUnit$1.call(CompilationUnit.java:189)
at org.codehaus.groovy.control.CompilationUnit.applyToSourceUnits(CompilationUnit.java:966)
at org.codehaus.groovy.control.CompilationUnit.doPhaseOperation(CompilationUnit.java:626)
```

The Windows taskbar at the bottom shows the time as 2:50 AM on 8/29/2023.



The screenshot shows a web browser window displaying the Jenkins console output for a pipeline named 'pipeline-shreya-t11 #1'. The console output is a stack trace indicating a failure in the Jenkins workflow. The stack trace starts with 'at org.codehaus.groovy.control.ErrorCollector.addError' and ends with 'Finished: FAILURE'. The browser window also shows the Jenkins dashboard on the left and the system taskbar at the bottom.

```
at org.codehaus.groovy.control.ErrorCollector.addError(ErrorCollector.java:119)
at org.codehaus.groovy.control.ErrorCollector.addError(ErrorCollector.java:131)
at org.codehaus.groovy.control.SourceUnit.addError(SourceUnit.java:349)
at org.codehaus.groovy.antlr.AntlrParserPlugin.transformCSTIntoAST(AntlrParserPlugin.java:225)
at org.codehaus.groovy.antlr.AntlrParserPlugin.parseCST(AntlrParserPlugin.java:191)
at org.codehaus.groovy.control.SourceUnit.parse(SourceUnit.java:233)
at org.codehaus.groovy.control.CompilationUnit$1.call(CompilationUnit.java:189)
at org.codehaus.groovy.control.CompilationUnit.applyToSourceUnits(CompilationUnit.java:966)
at org.codehaus.groovy.control.CompilationUnit.doPhaseOperation(CompilationUnit.java:626)
at org.codehaus.groovy.control.CompilationUnit.processPhaseOperations(CompilationUnit.java:602)
at org.codehaus.groovy.control.CompilationUnit.compile(CompilationUnit.java:579)
at groovy.lang.GroovyClassLoader.doParseClass(GroovyClassLoader.java:323)
at groovy.lang.GroovyClassLoader.parseClass(GroovyClassLoader.java:293)
at org.jenkinsci.plugins.scriptsecurity.sandbox.groovy.GroovySandbox$Scope.parse(GroovySandbox.java:163)
at org.jenkinsci.plugins.workflow.cps.CpsGroovyShell.doParse(CpsGroovyShell.java:190)
at org.jenkinsci.plugins.workflow.cps.CpsGroovyShell.reparse(CpsGroovyShell.java:175)
at org.jenkinsci.plugins.workflow.cps.CpsFlowExecution.parseScript(CpsFlowExecution.java:563)
at org.jenkinsci.plugins.workflow.cps.CpsFlowExecution.start(CpsFlowExecution.java:513)
at org.jenkinsci.plugins.workflow.job.WorkflowRun.run(WorkflowRun.java:336)
at hudson.model.ResourceController.execute(ResourceController.java:101)
at hudson.model.Executor.run(Executor.java:442)
Finished: FAILURE
```

Conclusion: In this experiment We had executed project on Jenkins pipeline.

Lab Outcome: LO1-To understand the fundamentals of DevOps engineering and be fully proficient with DevOps terminologies, concepts, benefits, and deployment options to meet your business requirements.

LO3-To understand the importance of Jenkins to Build and deploy Software Applications on server environment.