BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Agenda

Continuous Integration

- Using Continuous Integration Software:: Jenkins
- Artifact Management

Continuous Integration System

Jenkins

- The Jenkins project was started in 2004 (originally called Hudson) by Kohsuke Kawaguch
- Open Source
- Offers more than 1400 plugins

Jenkins

Prepare your environment

- Need Version control system
- Java
- Install Jenkins
- Jenkins default port 8080

Jenkins

Post installation

- Unlocking Jenkins
  - When you first access a new Jenkins instance, you are asked to unlock it using an automatically generated password

- Linux -> Jenkins console log output
- Windows -> $JENKINS_HOME/secrets/initialAdminPassword

Jenkins

Customization

- Plugins
- Integration with Git (Manage Jenkins -> Mange Plugins)
- Integrating Maven (Manage Jenkins -> Configure System)

- https://plugins.jenkins.io/
- Removing plugin
  - Uninstall option form Jenkins UI
  - Removing the corresponding .hpi file from the JENKINS_HOME/plugins directory on the master

Jenkins

Pipeline

- "Jenkins Pipeline is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins"
- The definition of a Jenkins Pipeline is written into a text file (called a Jenkinsfile)

```
pipeline {
    agent any

    stages {
        stage('Build') {
        steps {
                sh 'make'
                archiveArtifacts artifacts: '**/target/*.jar'
            }
        }
    }
        stage('Test') {
        steps {
                /* `make check` returns non-zero on test failures, * using `true` to   allow the
Pipeline to continue nonetheless */
                sh 'make check || true'
                junit '**/target/*.xml' }
            }
        }
        stage('Deploy') {
        when {
                expression {
                        currentBuild.result == null || currentBuild.result == 'SUCCESS'
                        }
            }
        steps {
            sh 'make publish'
            }
        }
}
```

In Figure:

1. Agent: It indicates that Jenkins should allocate an executor and workspace for this part of the Pipeline
2. Stage: It describes a stage of this Pipeline
3. Steps: It describes the steps to be run in this stage
4. SH: sh executes the given shell command (linux)
5. junit: It is a Pipeline step provided by the plugin

Jenkins

Why Jenkins Pipeline

Code:

Pipelines are implemented in code and typically checked into source control, giving teams the ability to edit, review

Durable:

Pipelines can survive both planned and unplanned restarts of the Jenkins master

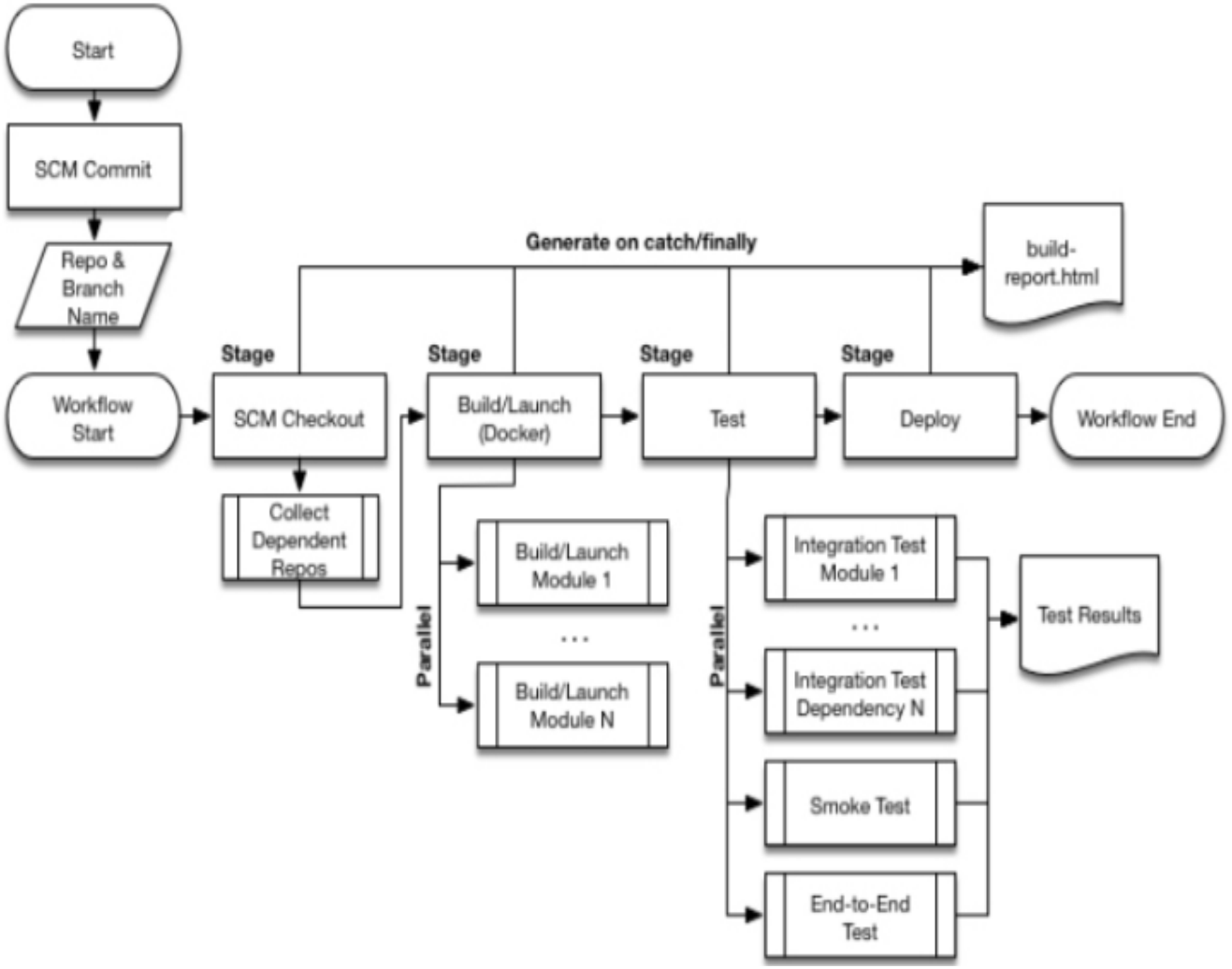
Pausable:

Pipelines can optionally stop and wait for human input or approval before continuing the Pipeline run

Versatile:

perform work in parallel

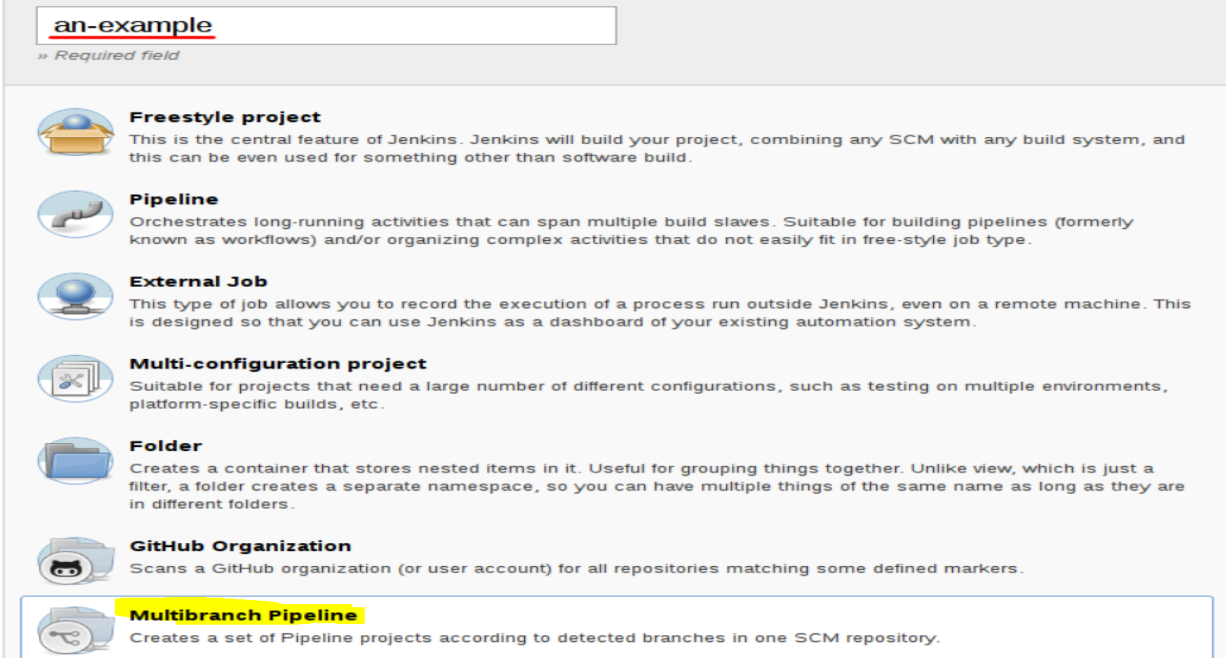
Jenkins Pipeline



Jenkins

Multibranch

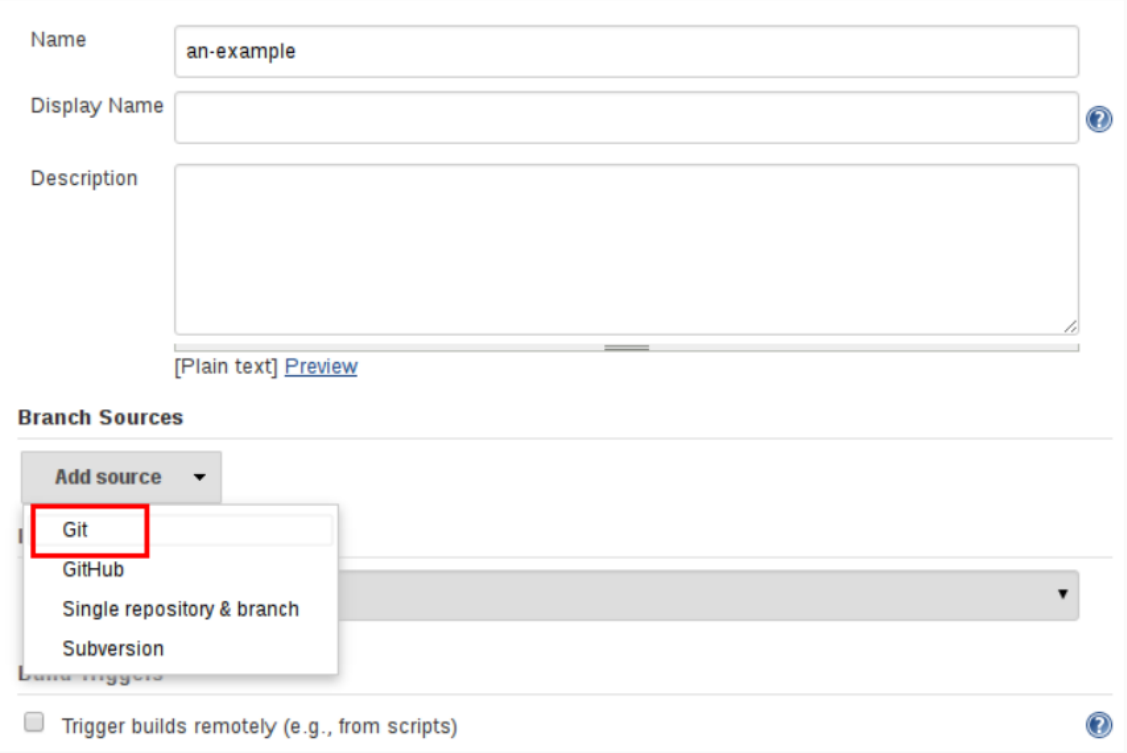
- The Multibranch Pipeline project type enables you to implement different Jenkinsfiles for different branches of the same project
- In a Multibranch Pipeline project, Jenkins automatically discovers, manages and executes Pipelines for branches which contain a Jenkinsfile in source control
- This eliminates the need for manual Pipeline creation and management
- Steps to To create a Multibranch Pipeline:
  - Step1: Click New Item on Jenkins home page
  - Step2 : Enter a name for your Pipeline, select Multibranch Pipeline and click OK

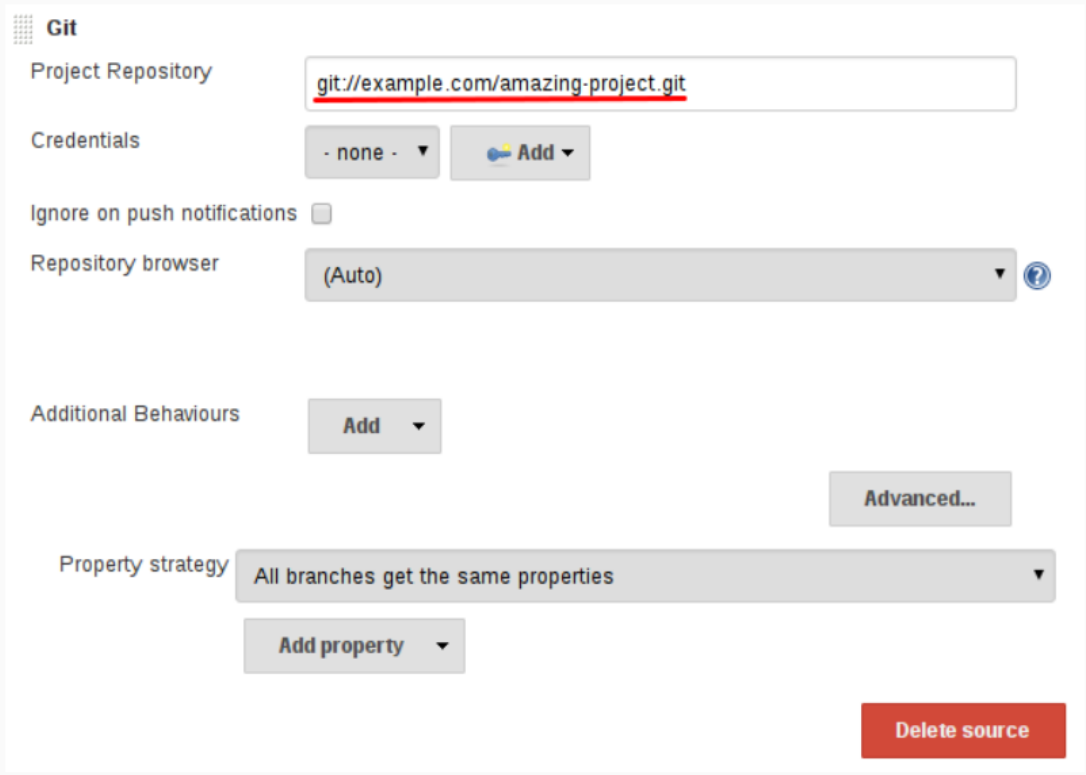


Jenkins

Multibranch

- Step3 : Add a Branch Source (for example, Git) and enter the location of the repository
- Step4: Save the Multibranch Pipeline project

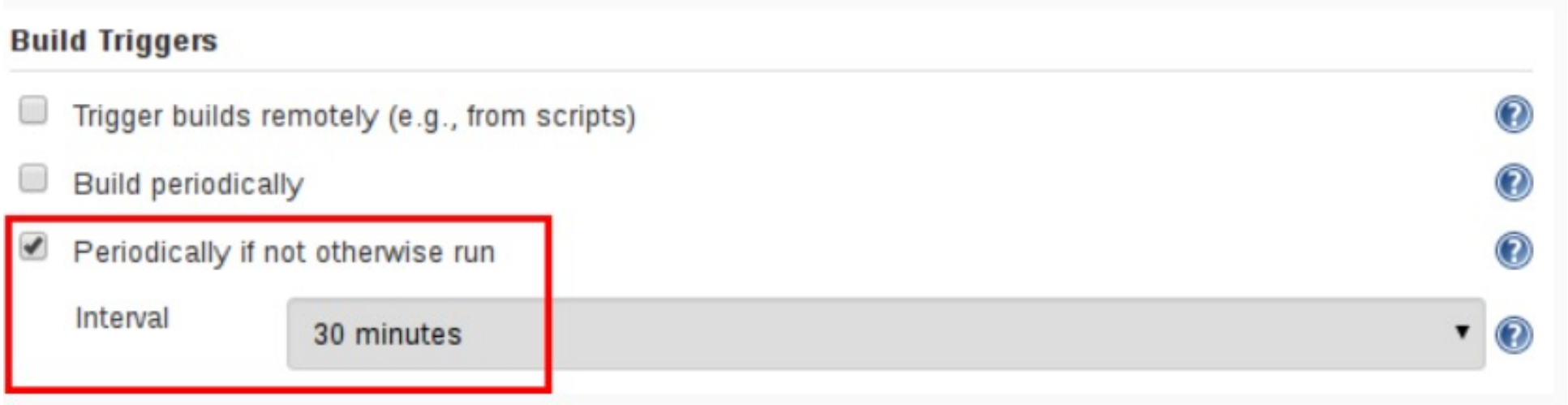




Jenkins

Multibranch

- Step5: Build Trigger Interval can be set
- Additional Environment Variables
  - BRANCH_NAME : Name of the branch for which this Pipeline is executing, for example master
  - CHANGE_ID : An identifier corresponding to some kind of change request, such as a pull request number



Artifact Management

What is Artifact

- An artifact is the output of any step in the software development process
- Artifacts can be a number of things like JARs , WARs, Libraries, Assets and application
- Generally, an artifact repository can serve as:
  - A central point for management of binaries and dependencies;
  - A configurable proxy between organization and public repositories; and
  - An integrated depot for build promotions of internally developed software

Artifact Management

Artifact Management Tools

- An artifact repository should be:
  - secure;
  - trusted;
  - stable;
  - accessible; and
  - Versioned
- Artifact Management Tools:
  - JFrog
  - Nexus
  - Maven
  - HelixCore

Artifact Management

Why

- Having an artifact repository allows you to treat your dependencies statically
- Artifact repositories allow you to store artifacts the way you use them
- Some repository systems only store a single version of a package at a time
- Ideally, your local development environment has the same access to your internal artifact repository as the other build and deploy mechanisms in your environment
- This helps minimize the “it works on my laptop” syndrome because the same packages and dependencies used in production are now used in the local development environment
- If you don’t have access to the internet within your environment; artifact management helps to have your own universe
- Relying on the internet for your dependencies means that somebody else ultimately owns the availability and consistency of your builds, something that many organizations hope to avoid
- An artifact repository organizes and manages binary files and their metadata in a central place

Artifact Management

Benefits

- The Binary Repository Manager: Artifact repositories allow teams to easily find the correct version of a given artifact
- This means developers can push to, and pull from, the artifact repository as part of the DevOps workflow
- Single Source of Truth: Artifact repositories improve the way teams work together by ensuring everyone is using the correct files
- It helps CI / CD to be more reliable

Q&A

Thank You!

In our next session: