BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Review CS#5

Automating Build Process

- Manage Dependencies
- Automate the process of assembling software components with build tools
- Use of Build Tools
  - Maven
  - Gradle
- Unit testing
- Automates Test Suite - Selenium
- Continuous Code Inspection
- Code Inspection Tools
  - Sonarqube

Agenda

Continuous Integration

- Continuous Integration
- Prerequisites for Continuous Integration Version Control
- Continuous Integration Practices
- Using Continuous Integration Software :: Jenkins
- Artifact Management

Test Automation

Selenium

- Preform an sort of interaction
- Selenium helps to automate web browser interaction
- Scripts perform the interactions

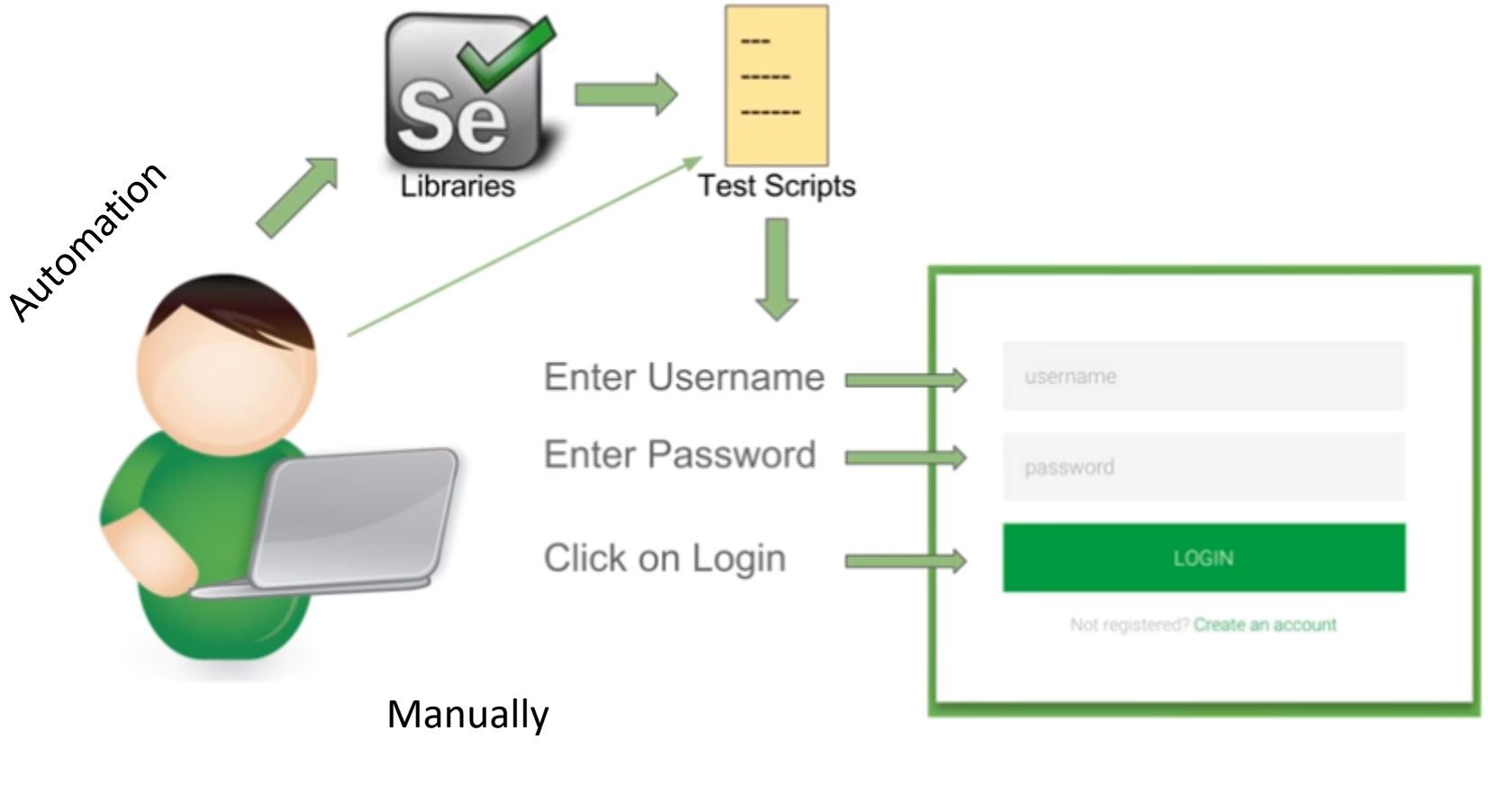
Selenium

Benefits

- Frequent regression testing
- Rapid feedback to developers
- Virtually unlimited iterations of test case execution
- Support for Agile and extreme development methodologies
- Disciplined documentation of test cases
- Customized defect reporting
- Finding defects missed by manual testing
- Reduced Business Expenses
- Reusability of Automated Tests
- Faster Time-to-Market

Selenium

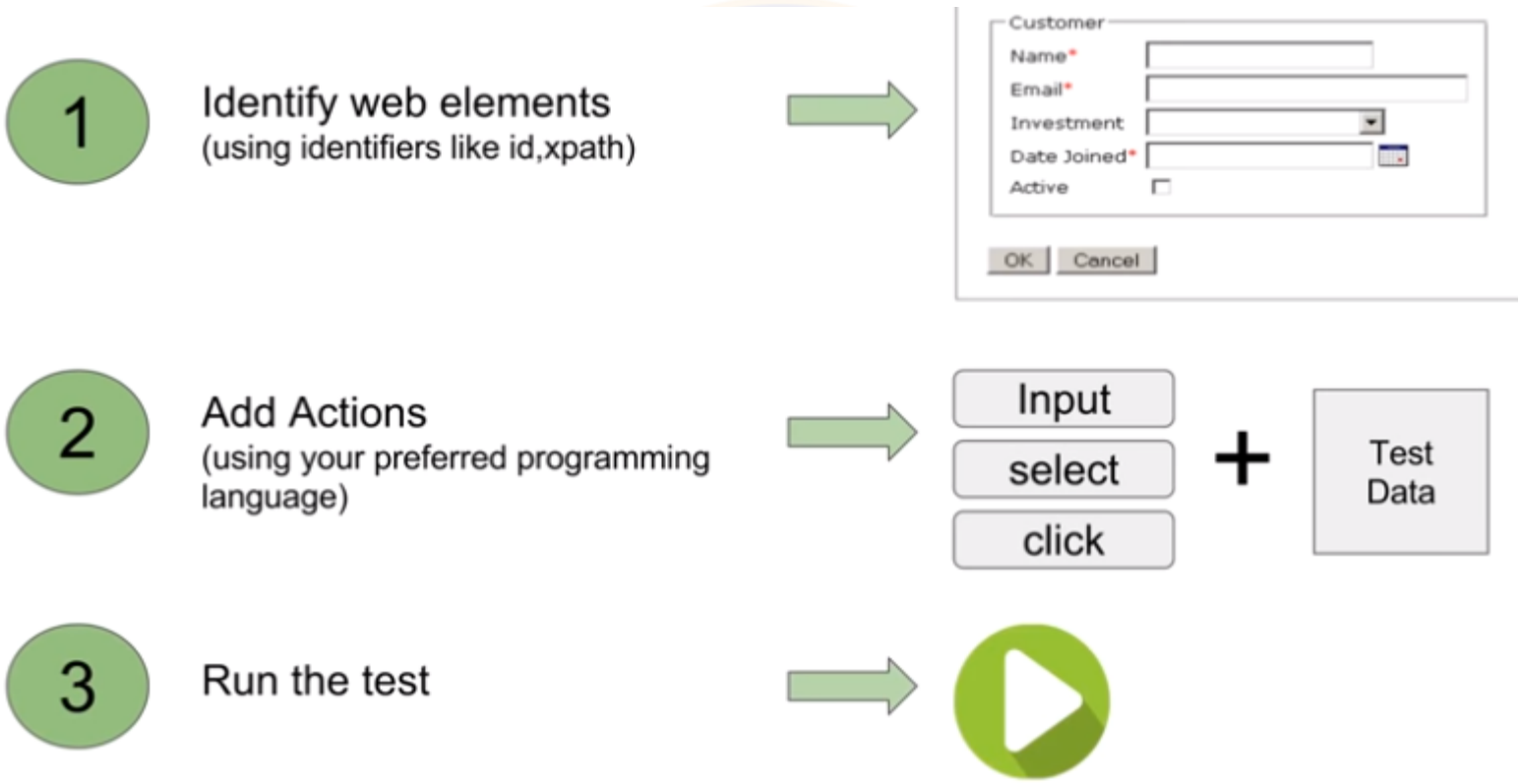
Lets say you want to test one login page



Selenium

Example

- At a high level you will be doing three things with Selenium



Selenium

Components

- Selenium IDE
  - A Record and playback plugin for Firefox add-on
  - Prototype testing
- Selenium RC (Remote Control)
  - Also known as selenium 1
  - Used to execute scripts (written in any language) using Javascript
  - Now Selenium 1 is deprecated and is not actively supported
- WebDriver
  - Most actively used component today
  - An API used to interact directly with the web browser
  - Is a successor to Selenium 1 / Selenium RC
  - Selenium RC and WebDriver are merged to form Selenium 2
- Selenium Grid
  - A tool to run tests in parallel across different machines and different browser simultaneously
  - Used to minimize the execution time

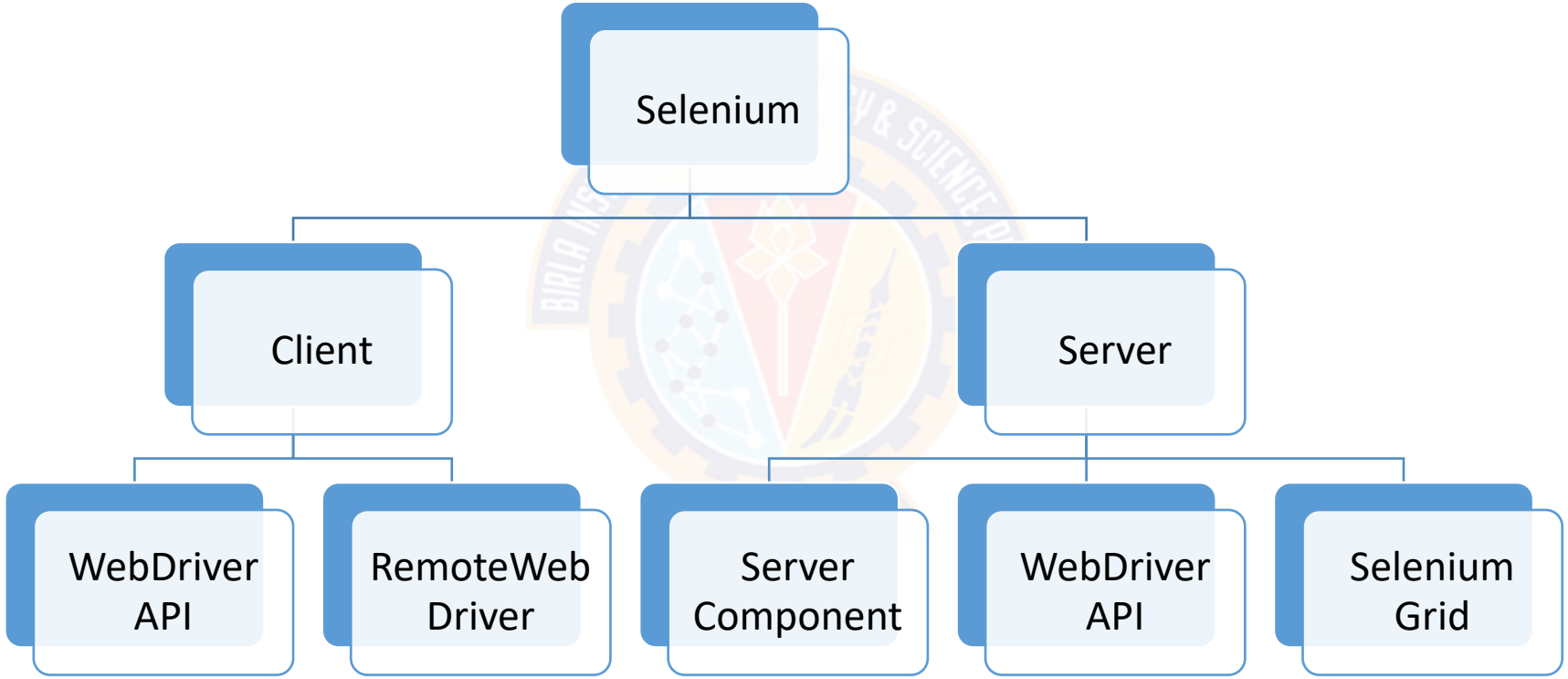
Selenium

Supports

- Browsers
- OS
- Language

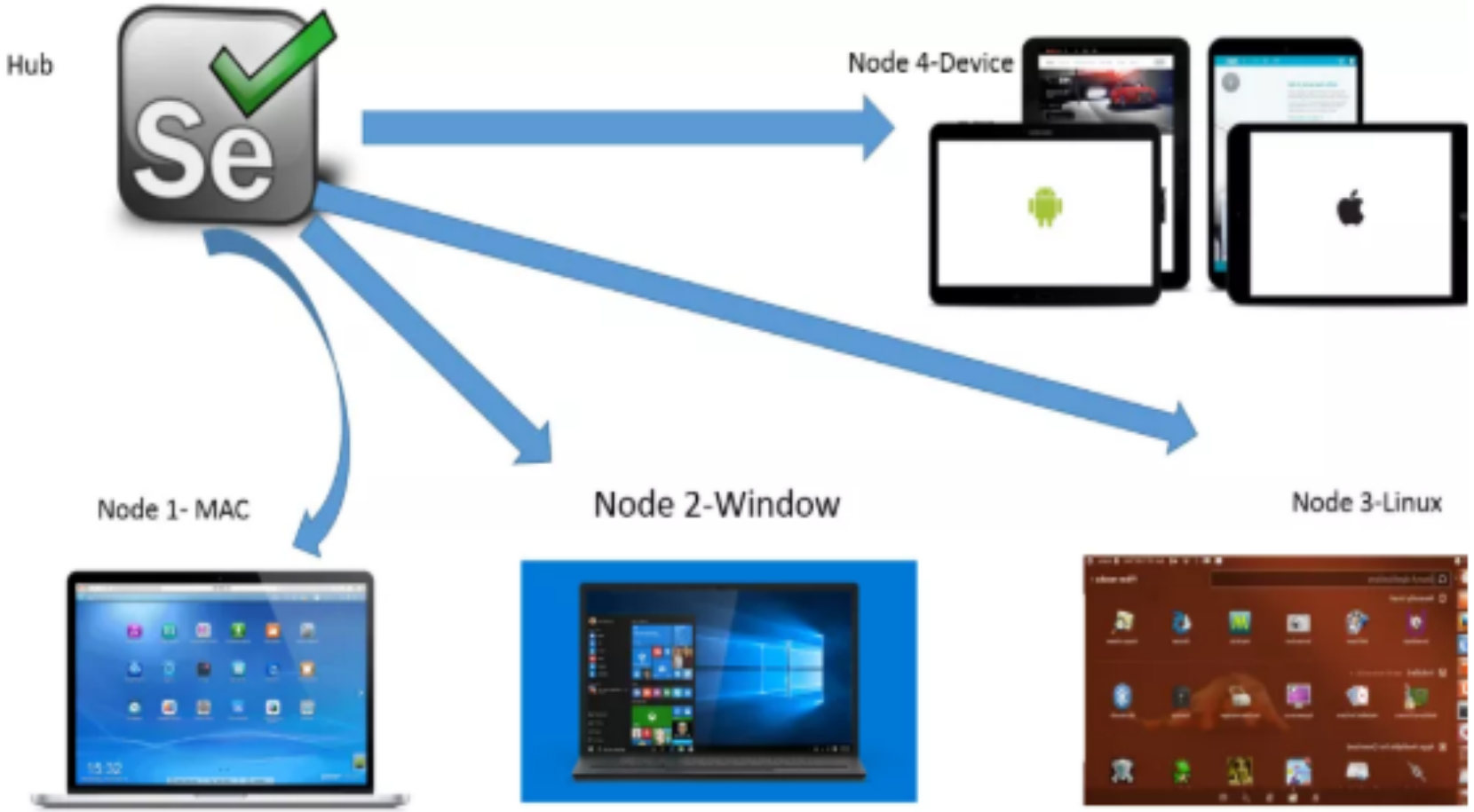
Selenium

Architecture



Selenium

Selenium Grid

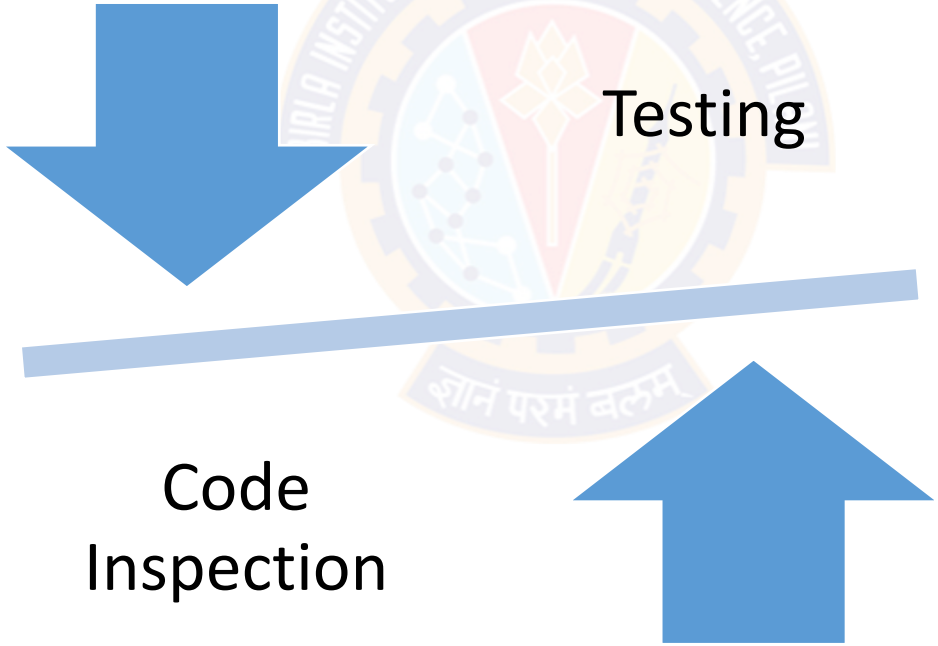


Continuous Code Inspection

Continuous code inspection = Constantly scanning code

- Identify if any defects
- It is a process of code review
- Its been proved 90% of defects can be addressed using code inspections tools

- **Find defects at the code level**
- **Minimizing defects during code inspections, -> testing efforts more efficient**



- **Verifies functionality** and **improves software quality**
- **Expensive** if you have to go through it over and over again

Note: Even with automated testing, it takes time to verify functionality; by resolving defects at the code level, you'll be able to test functionality faster

Continuous Code Inspection

Code Inspection Measures

- Code inspections must be well-defined as per requirements:
  - Functional requirements : User Needs : Cosmetic
  - Structural requirements : System Needs : Re-engineering
- Run Time Defects:
  - Identify run time errors before program run
  - Examples: Initialization (using the value of unset data), Arithmetic Operations (operations on signed data resulting in overflow) & Array and pointers (array out of bounds, dereferencing NULL pointers), etc.,
- Preventative Practices:
  - This help you avoid error-prone or confusing code
  - Example: Declarations (function default arguments, access protection), Code Structure (analysis of switch statements) & Safe Typing (warnings on type casting, assignments, operations), etc.,
- Style:
  - In-house coding standards are often just style, layout, or naming rules and guidelines
  - Instead using a proven coding standard is better for improving quality

Continuous Code Inspection

Improve Your Code Inspection Process:

- Involve Stakeholders
  - Developer, Management & Customer
- Collaborate
  - Collaboration — both in coding and in code inspections
- Recognize Exceptions
  - Sometimes there are exceptions to the rule
  - In an ideal world, code is 100% compliant to every rule in a coding standard
  - The reality is different
- Document Traceability
  - Traceability is important for audits
  - Capture the history of software quality

- **What to Look For in Code Inspection Tools**
  - Automated inspection
  - Collaboration system

Continuous Code Inspection Tool

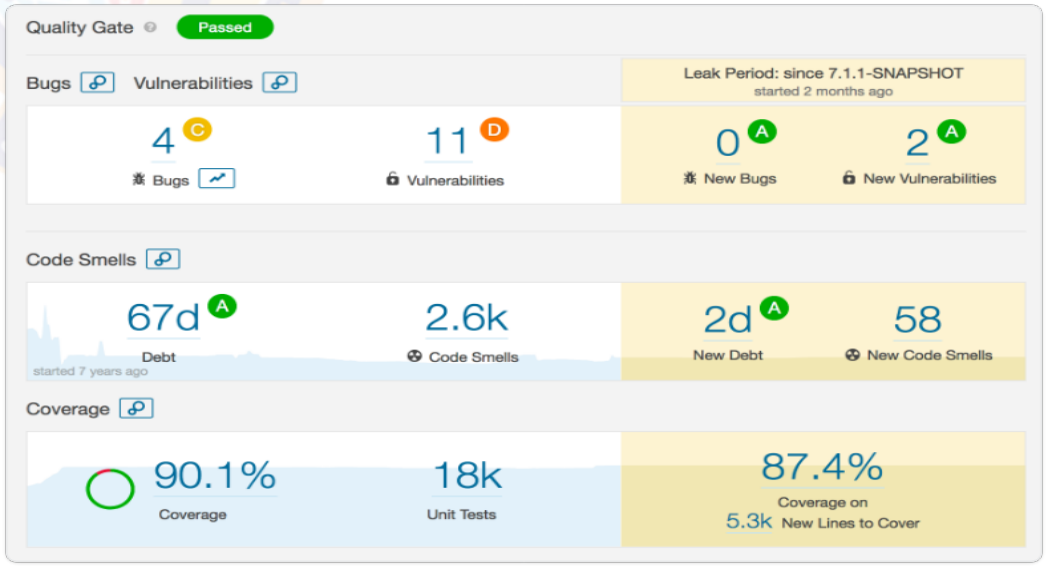
SonarQube

SonarQube

capability to show health of an application

Highlight issues newly introduced

Quality Gate, you can fix the leak and therefore improve code quality systematically

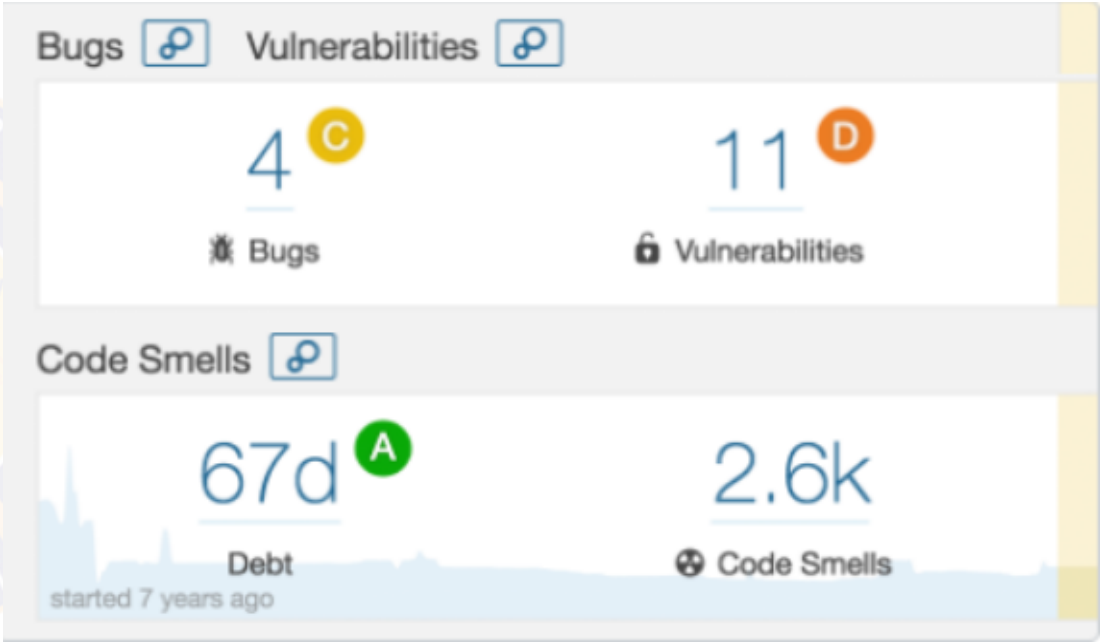


sonarqube

SonarQube

Overall health

- Bug:
  - An issue that represents something wrong in the code
  - If this has not broken yet, it will, and probably at the worst possible moment
- Code Smell:
  - A maintainability-related issue in the code
  - Examples: Dead Code, Duplicate code, Comments, Long method, Long parameter list, Long class etc.,
- Vulnerability:
  - A security-related issue which represents a backdoor for attackers



sonarqube

SonarQube

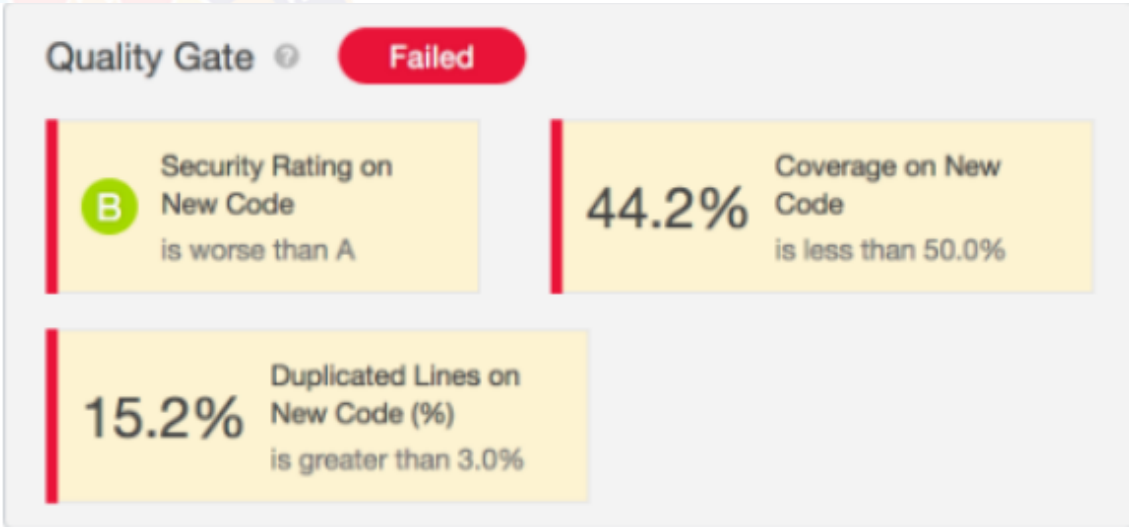
Enforce Quality Gate

- To fully enforce a code quality practice across all teams, need a Quality Gate
- A set of requirements that tells whether or not a new version of a project can go into production
- SonarQube's default Quality Gate checks what happened on the Leak period and fails if your new code got worse in this period

A quality gate is the best way to enforce a quality policy in your organization

Define a set of Boolean conditions based on measure thresholds against which projects are measured

It supports multiple quality gate definitions



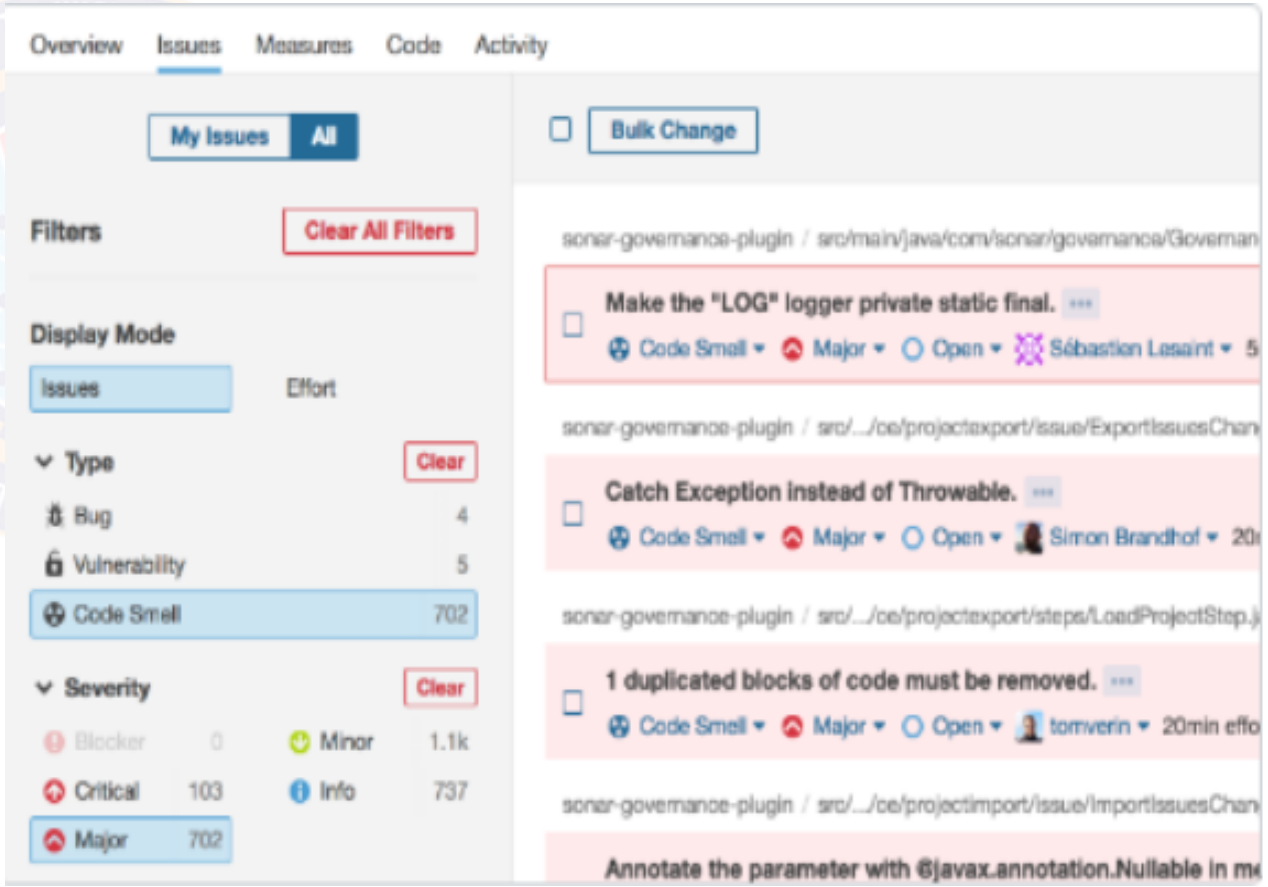
Example: Failed Project

sonarqube

SonarQube

Dig into issues

- The “Issues” page of your project gives you full power to analyze in detail
- What the main issues are?
- Where they are located?
- When they were added to your code base?
- And who originally introduced them?



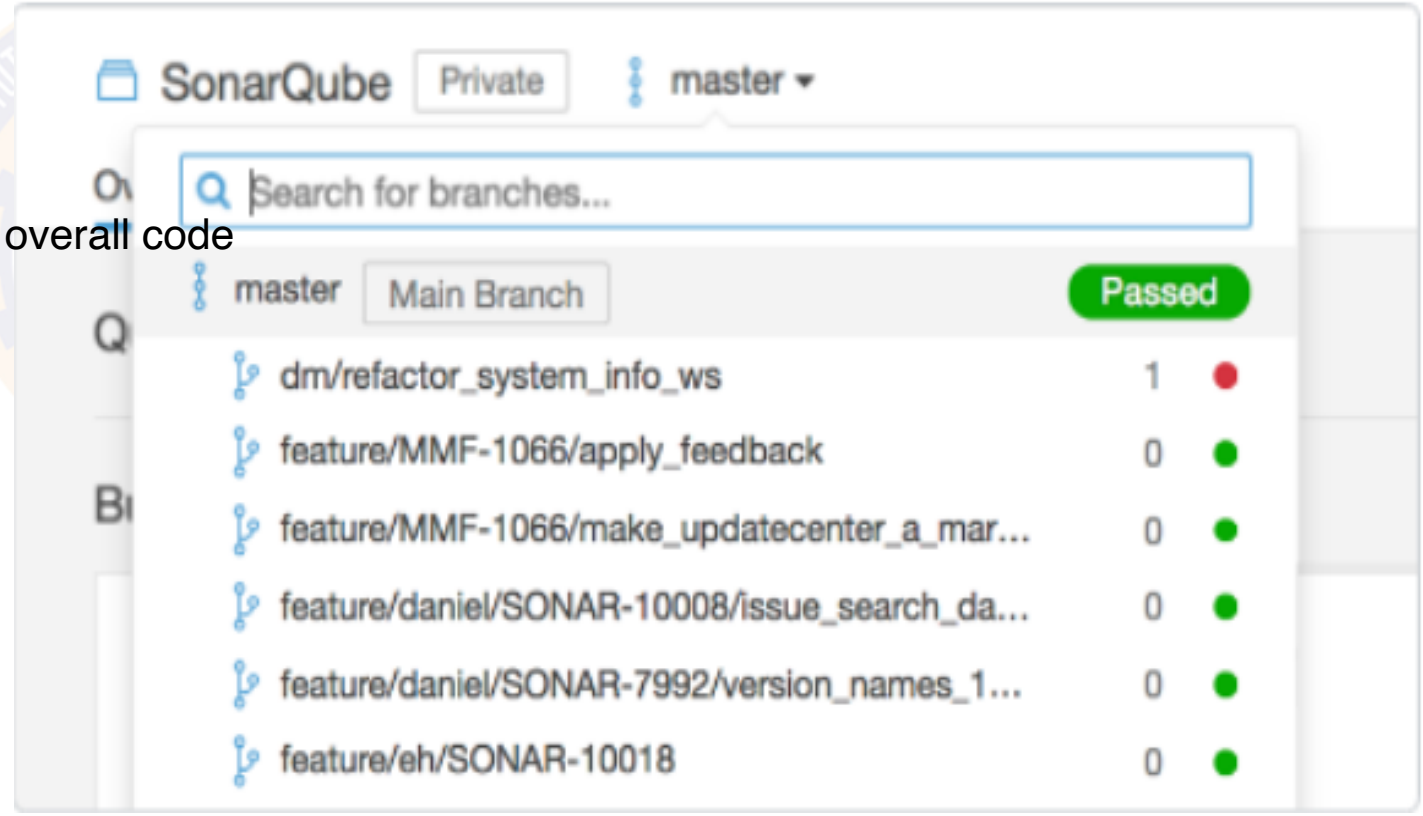
sonarqube

SonarQube

Analyzing Source Code

- Analyze pull requests
  - Focuses on new code – The Pull Request quality gate only uses your project's quality gate conditions that apply to "on New Code" metrics.

- Branch Analysis
  - Each branch has a quality gate that:
  - Applies on conditions on New Code and overall code
  - Assigns a status (Passed or Failed)



sonarqube

SonarQube

Integration for DevOps

Makefile

MSBuild

Continuous Integration

Continuous integration (CI)

- Process of integrating new code written by developers with a mainline or “master” branch frequently throughout the day

“Nobody is interested in trying to run the whole application until it is finished”

- Application wont be in a working state
- Team during development spends significant proportion time in an unusable state
- Lengthy integration phases at the end of development

Continuous Integration

Continuous integration requires

- Every time somebody commits any change, the entire application is built and a comprehensive set of automated tests is run against it
- If the build or test process fails, the development team stops whatever they are doing and fixes the problem immediately
- The goal of continuous integration is that the software is in a working state all the time

In simple words we can say "Finish First Before Start"

Implementing Continuous Integration

Pre-requisites

1. Version Control 

2. An Automated Build 

3. Agreement of the Team 

Continuous Integration Pre-requisite

Agreement of the Team

- This is more about People & Culture
- Continuous integration is a practice, not a tool
- It requires a degree of commitment and discipline from your development team or people involved
- As said “Fix first before Proceed”: Need everyone to check in small incremental changes frequently to mainline and agree that the highest priority task on the project is to fix any change that breaks the application
- If people don’t adopt the discipline necessary for it to work, attempts at continuous integration will not lead to the improvement in quality that you hope for

Continuous Integration

CI Tools

- Open Source :
  - Jenkins
  - Cruise Control
  - GitLab CI
  - GitHub Actions
- Commercial:
  - ThoughtWorks Studios
  - TeamCity by JetBrains
  - Bamboo by Atlassians
  - BuildForge by IBM

How it was before Continuous Integration

Just a glance !!!

- Nightly Build ☹

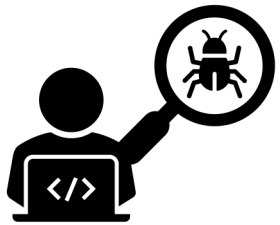
Note: this strategy will not be a good Idea when you have a geographically dispersed team working on a common codebase from different time zones

Continuous Integration

Pre-requisite & Best Practices

Prerequisite

Check In Regularly -> frequent check-ins



Create a Comprehensive Automated Test Suite

- Unit Test
- Component Test
- Acceptance Test

Will provide extremely high level of confidence that any introduced change has not broken existing functionality

Keep the Build and Test Process Short

Standard Recommendation:

10 min is Good

5 min is Better

90 Sec is IDEAL

Managing Your Development Workspace

local Development Workspace must be replica of Production

Continuous Integration

Pre-requisite & Best Practices

Best Practices

Don’t Check In on a Broken Build

What if we do Check In on Broken Build:

If any new check-in or build trigger during broken state will take much longer time to fix

Frequent broken build will encourage team not to care much about working condition

Commit locally than direct to Production

Wait for Commit Tests to Pass before Moving On

At time of Check-in, you should monitor the build progress

Never go home with broken build

Always Be Prepared to Revert to the Previous Revision

The previous revision was good because; you don’t check in on a broken build

Continuous Integration

Scenario 1

Friday at5.30 PM; your build is fail.

- You will be leaving late, and try to fix it
- You can revert changes
- You can leave the build broken

What Option you will opt here?

Leaving Broken Build:

- On Monday your memory will no longer be fresh
- It will take you significantly longer to understand the problem and fix it
- Everyone at work will yell at you
- If you are late on Monday; be ready to answer n number of calls
- Not the least your name will be mud

✗ Stay late to fix the build after working hours

Check in early enough so you have helping hands around
If it is late to Check In then Save your Check in for Next Day
Make rule of not checking in less than an hour before the end of work

Best Solution if you are alone then:
Revert it from your Source Control

Continuous Integration

Scenario 2

If you try to revert every time then; how can you make progress?

Time Boxing

- Establish a team rule
- When the build breaks on check-in, try to fix it for ten minutes or any approximate time your environment can bare
- However it should not be so long
- If, after ten minutes, you aren't finished with the solution, revert to the previous version from your version control system

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

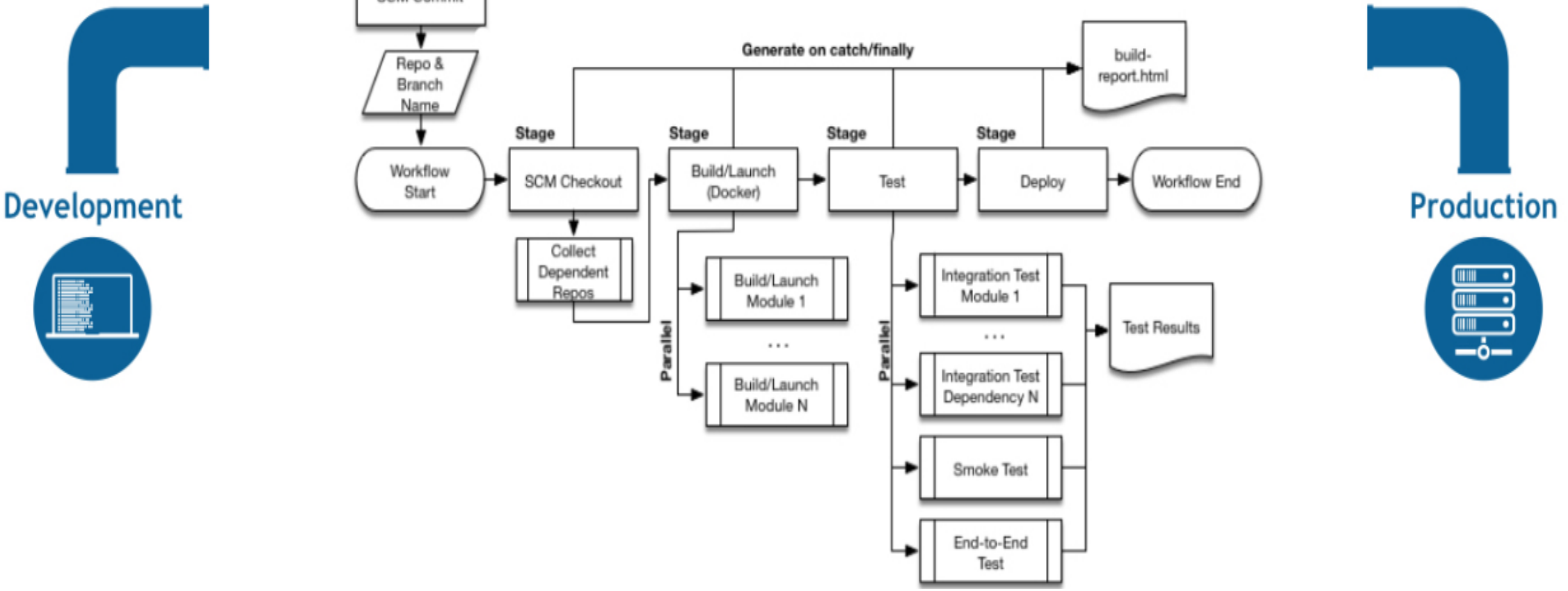
Pausable:

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

Versatile:

perform work in parallel

Jenkins Pipeline



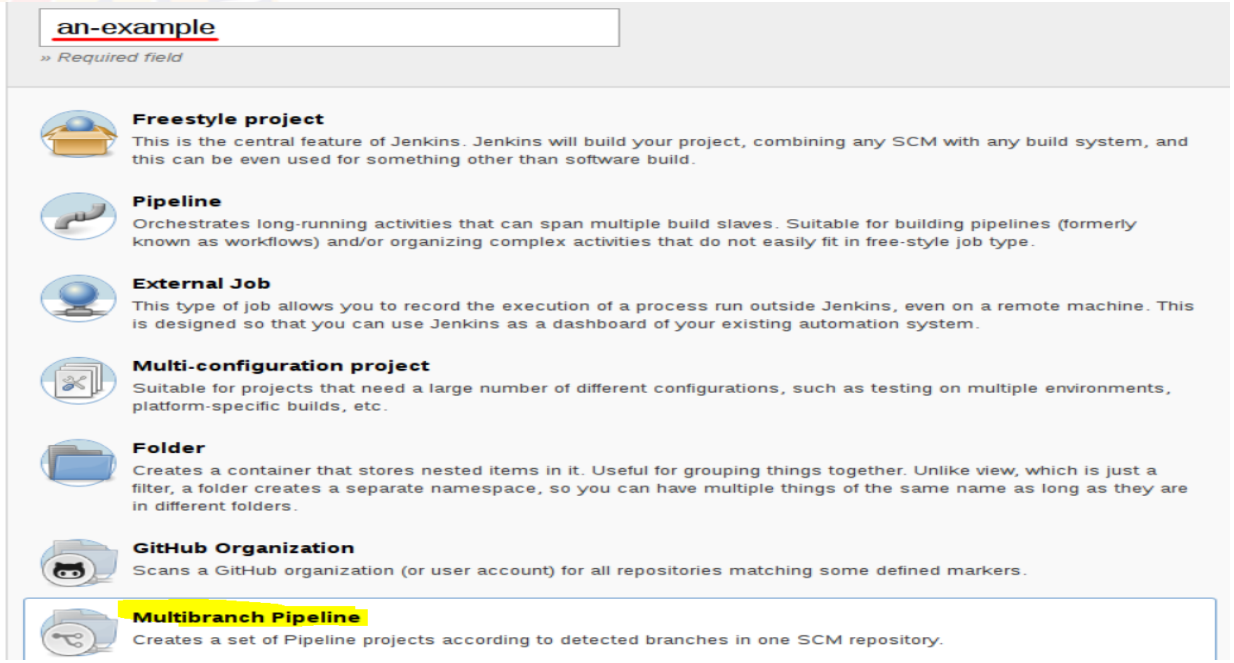
Jenkins

Lets Demo

Jenkins

Multibranch

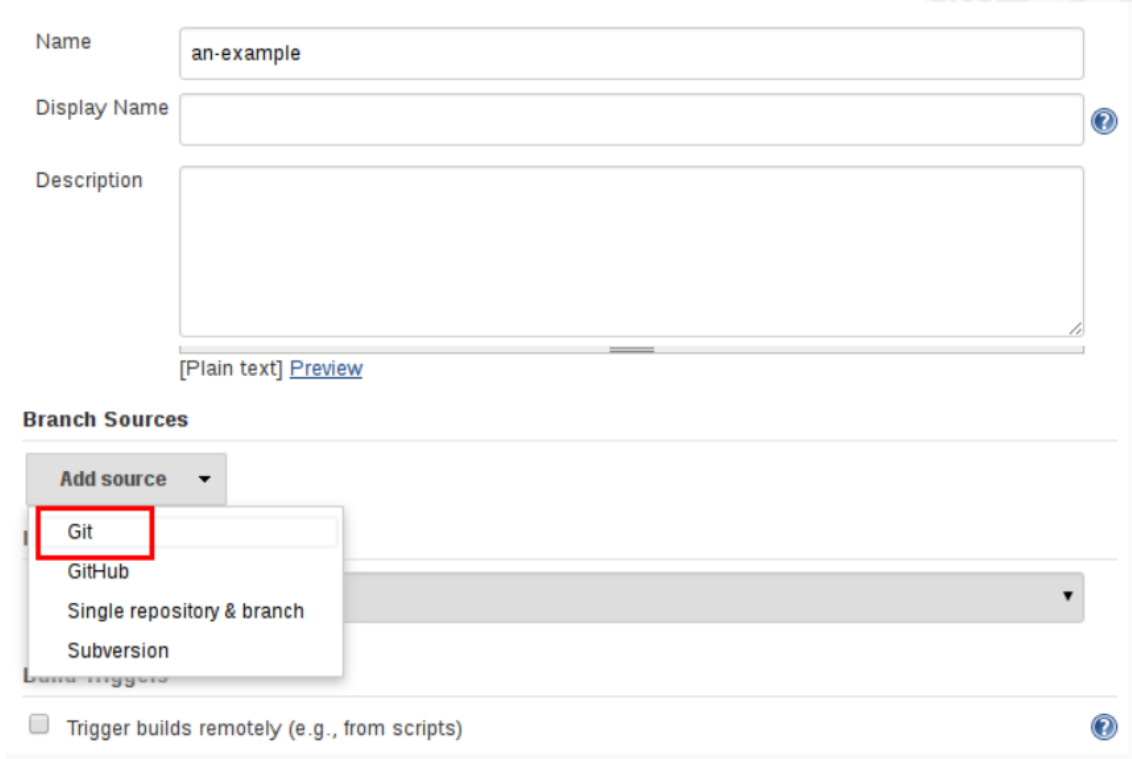
- The Multibranch Pipeline project type enables you to implement different Jenkinsfiles for different branches of the same project
- In a Multibranch Pipeline project, Jenkins automatically discovers, manages and executes Pipelines for branches which contain a Jenkinsfile in source control
- This eliminates 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

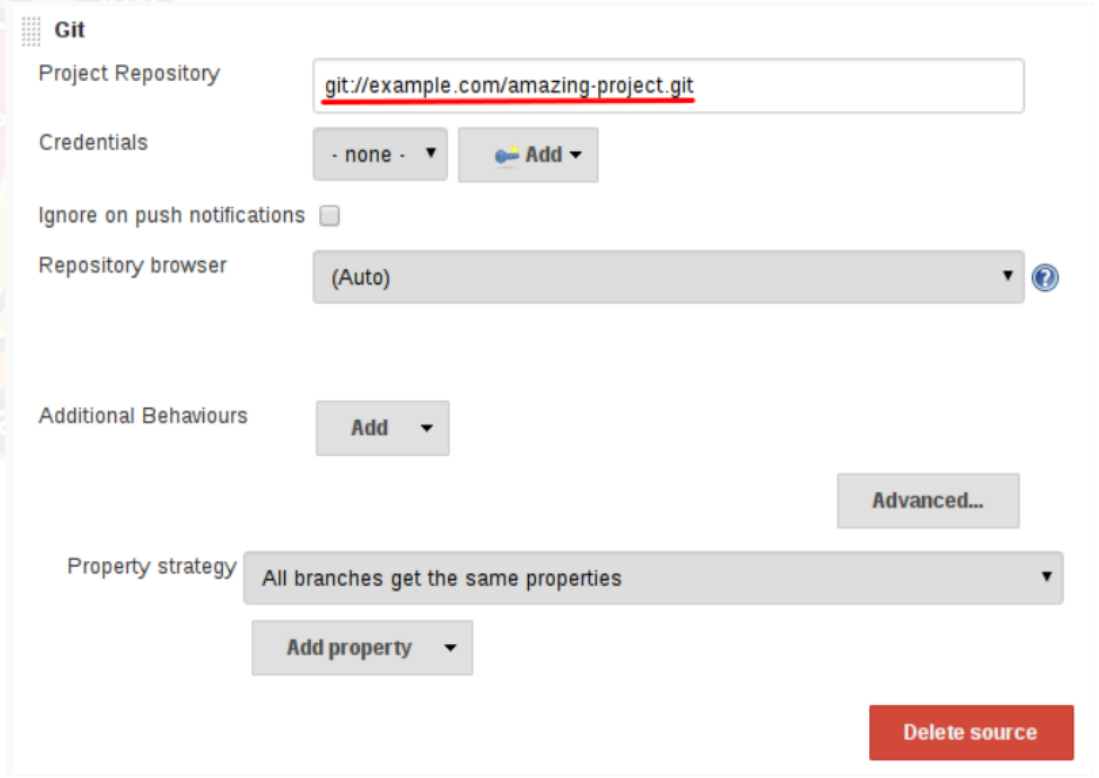


Jenkins

Multibranch

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

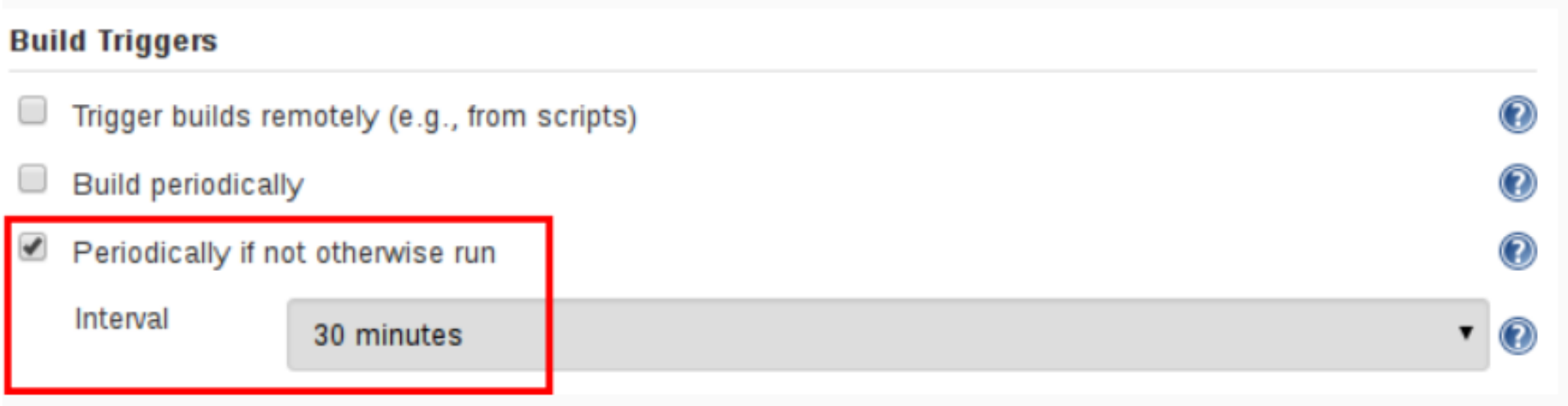




Jenkins

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: