**What is DevOps?**

It is a software engineering culture and practice that aims at unifying software development (Dev) and software operation (Ops). A combination of practices and tools to increase an organization’s ability to deliver applications and services faster than a traditional software development process.

Aims:

1. Shorter development cycles
2. [Increased deployment frequency](https://en.wikipedia.org/wiki/Continuous_delivery)
3. More dependable releases, in close alignment with business objectives.

**What are Continuous Integration and Continuous Delivery?**

Continuous integration focuses on integrating work from individual developers into a main repository multiple times a day to catch integration bugs early and accelerate collaborative development. Continuous delivery is concerned with reducing friction in the deployment or release process, automating the steps required to deploy a build so that code can be released safely at any time. Continuous deployment takes this one step further by automatically deploying each time a code change is made.

**Continuous integration is a practice that encourages developers to integrate their code into a main branch of a shared repository early and often. Instead of building out features in isolation and integrating them at the end of a development cycle, code is integrated with the shared repository by each developer multiple times throughout the day**.

In practice, continuous integration relies on robust test suites and an automated system to run those tests. When a developer merges code into the main repository, automated processes kick off a build of the new code. Afterwards, test suites are run against the new build to check whether any integration problems were introduced. If either the build or the test phase fails, the team is alerted so that they can work to fix the build.

The end goal of continuous integration is to make integration a simple, repeatable process that is part of the everyday development workflow in order to reduce integration costs and respond to defects early.

**Continuous delivery** is an extension of continuous integration. It focuses on automating the software delivery process so that teams can easily and confidently deploy their code to production at any time i.e. ensuring that the codebase is always in a deployable state. Continuous delivery leans heavily on deployment pipelines to automate the testing and deployment processes. A **deployment pipeline** is an automated system that runs increasingly rigorous test suites against a build as a series of sequential stages.

**Difference between Continuous Delivery and Continuous Deployment?**

**Continuous deployment** is an extension of continuous delivery that automatically deploys each build that passes the full test cycle. Instead of waiting for a human gatekeeper to decide what and when to deploy to production, a continuous deployment system deploys everything that has successfully traversed the deployment pipeline.

**DevOps Assessment Sheet. Maturity Drivers.**

Details about the technology used by the application (Java, etc) and different details like app team size and future scope.

A rating for the different maturity drivers like Source Code Management (GIT, etc), CI, CD, CT (continuous testing), etc. This rating, from 0 to 5, is dependent on a number of factors, all predefined.

**What are Jenkins, Maven, Artifactory and SonarQube?**

**Jenkins** is an open source automation server used for continuous integration and continuous delivery, written in JAVA.

**Maven** is a build automation tool used primarily for Java projects. Maven helps manage builds, documentation, reporting, dependencies, software configuration management (SCM), releases and distribution.

**Artifactory** is a binary repository manager whose purpose is to aggregate various repositories in one location. This allows for storage of libraries for any CI/CD and DevOps tools in Artifactory including NuGet packages for Microsoft .Net applications, Node.js and Common.js packages for Node applications, as well as Jar files for Java applications plus any other type of libraries.

**SonarQube** is a platform for continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs, code smells and security vulnerabilities. SonarQube offers reports on duplicated code, coding standards, unit tests, code coverage, code complexity, comments, bugs, and security vulnerabilities.

**What are smoke testing and unit testing?**

**Smoke tests** are a special kind of initial checks designed to ensure very basic functionality as well as some basic implementation and environmental assumptions. Smoke tests are generally run at the very start of each testing cycle as a sanity check before running a more complete test suite. Smoke tests are not very extensive, but should be extremely quick. If a change fails a smoke test, its an early signal that core assertions were broken and that you should not devote any more time to testing until the problem is resolved.

**Unit tests** are responsible for testing individual elements of code in an isolated and highly targeted way. The functionality of individual functions and classes are tested on their own. Any external dependencies are replaced with stub or mock implementations to focus the test completely on the code in question.

Unit tests are essential to test the correctness of individual code components for internal consistency and correctness before they are placed in more complex contexts. The limited extent of the tests and the removal of dependencies makes it easier to hunt down the cause of any defects.

kreddy92

Nayagara@123