# Introduction

Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.

By integrating regularly, you can detect errors quickly, and locate them more easily.

Continuous Integration is a practice, not a tool.  Implementing a successful CI practice within an organization requires discipline, particularly when your development environment contains a significant amount of complexity.

Block Diagram

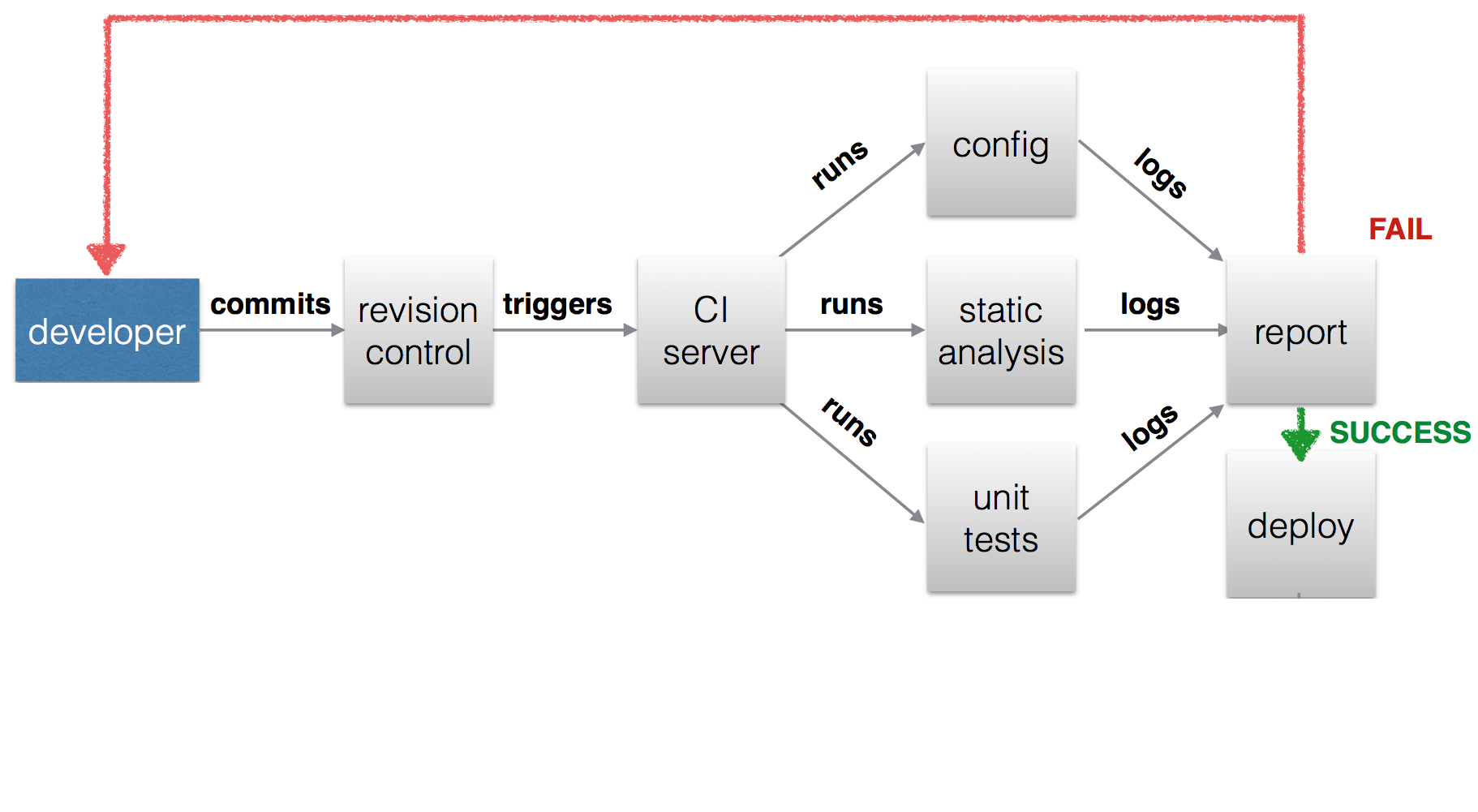


Fig: CI generic block diagram

**Continuous Integration brings multiple benefits to your organization:**

* Say goodbye to long and tense integrations
* Increase visibility which enables greater communication
* Catch issues fast and nip them in the bud
* Spend less time debugging and more time adding features
* Proceed in the confidence you’re building on a solid foundation
* Stop waiting to find out if your code’s going to work
* Reduce integration problems allowing you to deliver software more rapidly

# BEST PRACTICES OF ContinUous Integration

Following are the guidelines for developers:

1. **Maintain a code repository :**

There should be single repo for entire project so that it will be easy for source code management.

All developer’s should clone that repo into their local systems and should create feature branch and start coding on it and once it is reviewed merge it into master branch**.**

1. **Version all Test Scripts and configuration files**

Keeping test scripts and configuration file will help Development team to audit and trace code changes, as well as help to revert to previous version any time for code comparison.

1. **Everyone commits to the baseline every day:**

Every developer should commit his whole day work at the end of the day into repository.

1. **Automate the build :**

It is best practice to use build tool to automate build process.  Build tool automates the creation of executable applications from source code. As per programming language, there are multiple build tools available for automation. Such as maven, ant, gradle, grunt, gulp etc.

1. **Enable Triggering of Builds from the Command Line**Teams should streamline the build process so that it uses common, standard tools.  For example in the Java it would be Maven, ANT. To enable triggering of builds from the command line, teams should also standardize on a build script/tool and incorporate unit testing into the build process.
2. **Every commit should be build**

For build automation it is advisable to use tools like Jenkins or Bamboo. These tools are used for integration and to automate the build process using build tools. For every code commit on repo, Jenkins build should be triggered.

1. **Fast Builds**

Ideally any new build should not take more than 10 minutes as people won’t commit frequently. Break large

builds into multiple jobs and execute them parallel.

1. **Test using replica of production**

It is good practice to keep testing environment same as production to avoid dependencies issues like Operating System, patches, libraries etc.

1. **Integrate Frequently**

Integrate code changes frequently so that Integration doesn’t become a hard process. By doing so Team can have CI server building that mainline.

1. **Use of static code analysis in build process**

It is always advisable to follow certain sets of coding standards, which results in less code duplication, less coding standard violation, less coding complexity .It comes under code review. Previously it was done manually but in automated CI process (using Jenkins/bamboo), there are multiple tools by means of which we can carry out static code analysis in automated CI process (using Jenkins/bamboo) by means of which we can display it’s reports on Jenkins/bamboo dashboard.

It is advisable to use static code analysis tools such as sonarqube, Istanbul etc.

1. **Every change should be tested :**

Every changes in codebase should be tested .It is advisable to use different frameworks for testing.

For e.g. Phpunit, Junit, Selenium framework etc.

1. **Use of code coverage tools in build process.**

Your test cases should be such that it should cover entire code. In current CI build process these things should be automated .This can be done by using code coverage tools like Emma, clover, coberture etc. It is possible to render reports generated by code coverage tools on Jenkins/bamboo for more readability and understanding in case of build failures.

1. **Generate various reports through CI servers to have good visibility and readability.**

Reports generated by using build tools are in csv or xml format, which are not readable. It is advisable to use CI servers to display (render) reports on CI server’s dashboard in tabular or statistical format.

1. **Securing CI servers**

There should be levels of authentications provided in CI servers for group of users. For some CI servers this facility is in build but for some CI servers we need to explicitly provide it by adding some plugins related to it.

1. **Integrating CI server with defect management tool and test management tool.**

In case of Defect management tool, we recommend JIRA. If we integrate CI server with Jira, We are able to view for what issue which build is run, what kind of results it produced. Most of the CI servers currently have end to end traceability with Jira. For Test Management tool we can prefer Zephyr, Testlink.

Integration between these tools can help for continuous improvement during development, which then help to reduce development lifecycle to few extents.

1. **Integrating User Authentication Management system. (SSO)**

In CI process, we use many applications and for every application, authentication is required. So it is better to use some Authentication Management system which provides facility of single sign on.

1. **Tagging sprint/releases/important builds through CI servers.**

There should be some provision for tagging of sprints, releases, builds in CI build, which will in turn be helpful while doing rollback of the builds from the environments.

1. **End to end build process:**

Every developer should have knowledge of entire build process dedicated for entire sprint**.**

1. **Violation limits for stable build**

There are certain violations like checkstyle, pmd etc. . We can set certain values for these violations.

If your build is not fulfilling limits which were set in Jenkins/Bamboo, then it should result in build failure.

1. **Create stable and fast builds**

The build should be stable .there should not be any long running builds or any build queues. These things affect the overall performance of the build

1. **Easy access of artifacts to all developers**

Any successful build will result in creation of artifacts, so keep track of these artifacts it is advisable to integrate Jenkins with binary repository systems like Artifactory/Nexus. This helps to manage and govern binary artifacts and its associated metadata. It also helps to enable end to end traceability by making it associated with builds and version control system. All developers should have access to view the build results of Jenkins/bamboo.

1. **Significant of Environment variable.**

Most of the CI servers use their own sets of environment variables. These variables can be injected during running of build or build pipeline to make certain changes. One of the uses of environment variable is tagging the build. It is also possible to perform Rollback process by use of Environment variables.

1. **Automate Deployment**  
   Automated deployment helps to reduce manual affords. Many continuous integration servers already have plugins for automating deployment to various web application servers like TomCat, JBoss, and WebSphere.
2. **Build Types**

Apply different Build types & Mechanisms as per provision – Some of the suggested build types are Private Build, Integration Build & Release Build.

1. **Run CI for every change**

CI should run whether it is a code change, database change or infrastructure change.

**Continuous Delivery/Continuous Deployment**

# Table of Contents

[Table of Contents 2](#_Toc424132769)

[1. Continuous Delivery: 3](#_Toc424132770)

[2. Continuous Deployment: 4](#_Toc424132771)

[3. Continuous Delivery Vs Continuous Deployment 5](#_Toc424132772)

[4. Conclusion: 6](#_Toc424132773)

# 

# Introduction

* **Continuous software Development (CD)** is a term that covers several aspects of an iterative application development process based on making each change when it is ready, rather than wrapping many changes into large batches.
* Continuous software development is an umbrella term that describes several aspects of iterative software application development, including **Continuous Integration, Continuous Delivery, Continuous Testing and Continuous Deployment**.
* Continuous integration increases your opportunities for feedback. It can be used to reduce the time between when a defect is introduced and when it is fixed, thus improving overall software quality.
* Continuous delivery builds on continuous integration and as with continuous integration; each code commit is automatically tested at the time it is added.

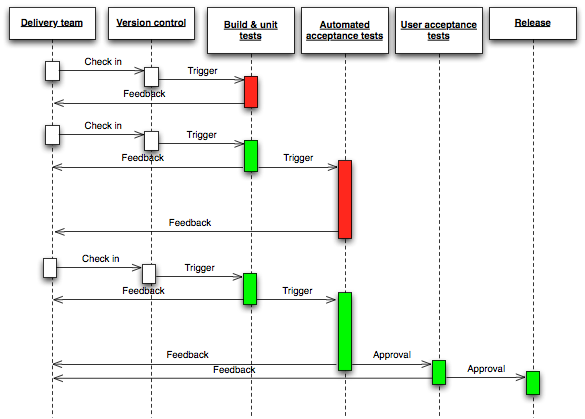
Continuous deployment adds more automation to the software development process. After passing all the automated delivery tests, each code commit is deployed into production as soon as it is available.

# Continuous Delivery:

Continuous Delivery is a software engineering approach in which teams keep producing valuable software in short cycles and ensure that the software can be reliably released at any time. It is used in software development to automate and improve the process of software delivery. Techniques such as automated testing and continuous integration(CI) allow software to be developed to a high standard and easily packaged and deployed to test environments, resulting in the ability to rapidly, reliably and repeatedly push out enhancements and bug fixes to customers at low risk and with minimal manual overhead. CD builds on CI by adding the regular deployments to production as part of the process; however CD is not a requirement of CI. The technique (CD) was one of the assumptions of extreme programming but at an enterprise level has developed into a discipline of its own, with job descriptions for roles such as "buildmaster" calling for CD skills as mandatory.

Here are the major benefits of Continuous Delivery:

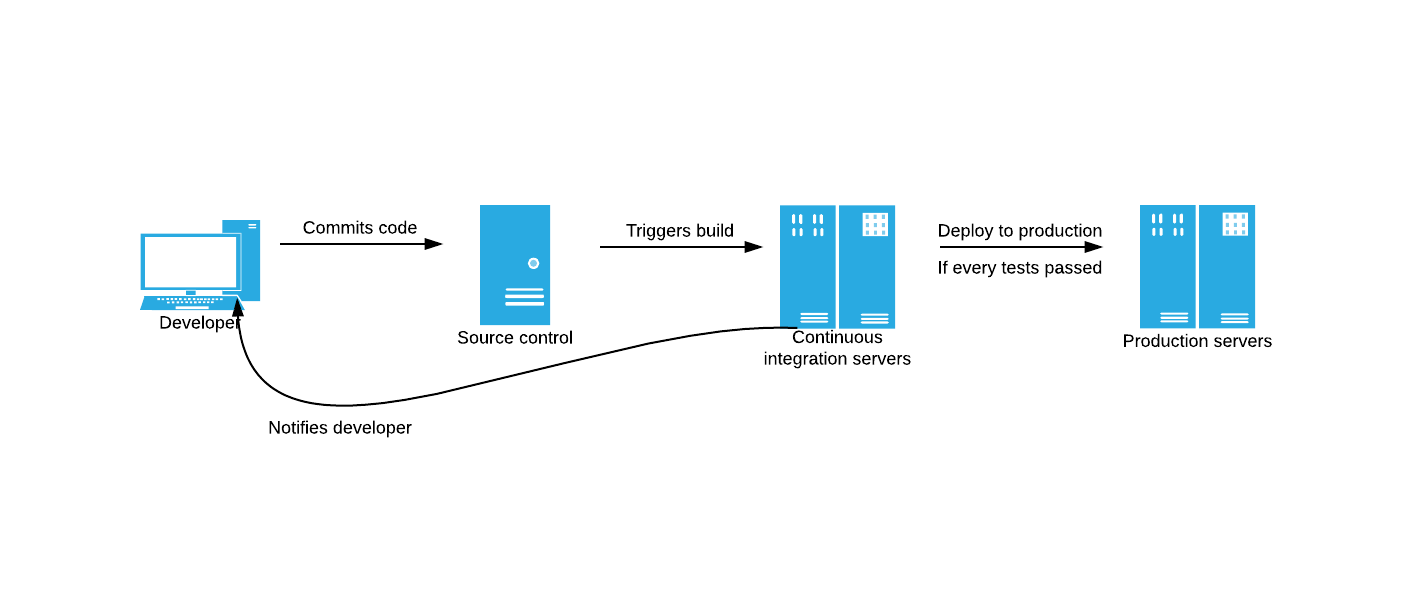
* Working software is deployed to any environment with just the push of a button.
* Code is delivered to a user base for continual review and inspection.
* Small batches of work can be continually filtered up to the next step, which will help identify any issues early on and keep the developer working to fix any problems while the issues are top-of-mind.



# Continuous Deployment:

Continuous deployment is the next step of continuous delivery: Every change that passes the automated tests is deployed to production automatically. Continuous deployment should be the goal of most companies that are not constrained by regulatory or other requirements.

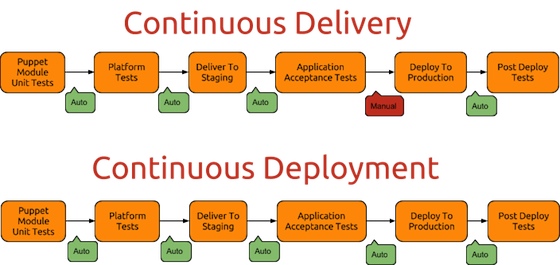
There are business cases in which IT must wait for a feature to go live, making continuous deployment impractical. While application feature toggles solve many of those cases, they don’t work in every case. The point is to decide whether continuous deployment is right for your company based on business needs — not on IT limitations.



Continuous Deployment can offer the following benefits:

* It creates smaller, faster deployments that are less risky.
* It fosters the positive image that the DevOps team and company are constantly improving their product.
* It creates a more digestible product to the user compared to a big release in which productivity may be temporarily affected by a new learning curve.

# Continuous Delivery Vs Continuous Deployment



# USer Guidelines FOr continuous dEVELOPMENT

1. **Manage Source Code, Using a Version Control Product –** To provide a complete audit trail and history of the code. Using build server you can monitor the code to determine any changes.
2. **Preferably use a Central Binary Repository - Use a** central artifact binary repository to manage and govern the binary artifacts and associated metadata.  It also helps to enable end to end traceability by making it easier to associate binaries, builds and versions of source code.
3. **Version All Test Scripts and Configuration Files** -Test groups and configuration files are similar to source code in that they change frequently and need to be audited and traced. That way, teams can revert back anytime to previous versions of the code for comparison.
4. **Script an efficient Delivery Eco-System and Release plan.** A Maturity Model for Configuration and Release Management.
5. Should take backups of database/code.
6. Everything is scripted - no manually connecting to the database to add in that extra column.
7. You have tested the deployment on a staging area already (on a copy of the live data).
8. Should have a rollback plan .Rolling Back Deployments to last successful Build.
9. The deployment is repeatable - it should contain all files needed to setup the site as per that release, not just the files that changed.
10. You release the exact same build to each environment; you don't build a new version of the code when going from staging to production.
11. Release from the build server, not from an individual developers machine, that way it should be the same no matter who does the release.
12. The smaller the change you are releasing live the less that can go wrong, releasing should happen often and it should be easy.

# COnfiguration Guidelines FOr continuous DEVELOPMENT

1. Configuration management (CM) **-** Teams should streamline the build process so that it uses common, standard tools such as Build Tool, Versioning Tool.   
   Teams should also standardize on a build script/tool and incorporate unit testing into the build process.

1. **Continuous Integration (CI) -** Developers should integrate code changes frequently, and should use the continuous integration server building that main line.
2. **Implementing an efficient Continuous Testing Plan -** Testing should be done in a production-like environment. Build needs to have the same version of operating system, patches, libraries, and so forth.  
   When the binary goes into production, there won’t be dependency issues.

1. **Deployment Pipeline Practices -**
2. Build your binaries at least once.
3. Deploy the same to every Environment.
4. Smoke test your Deployments.
5. Each change should propagate through the pipeline instantly.
6. If any part of the pipeline fails, then stop the line or Rollback.
7. **Test Non-Functional Requirements -**
8. Capacity Testing via User Interface.
9. Record Interactions against a Service or Public API.
10. Use Capacity Test stubs to Develop Tests.

# 7. Principle for Contituous delivery

* **Build your binaries only once**: Compiling code should only happen once, eliminating the risk of introducing difference due to environments, third party libraries, different compilation contexts, and configuration differences.
* **Deploy the same way to all environments**: Use the same automated release mechanism for each environment, making sure the deployment process itself is not a source of potential issues. While you deploy many times to lower environments (integration, QA, etc.) and fewer times to higher environments (pre-production and production), you can’t afford failing deployment to production because of the least tested deployment process.
* **Smoke-Test your deployments**: A non-exhaustive software test (essentially testing “everything” – services, database, messaging bus, external services, etc.) that doesn’t bother with finer details but ascertains that the most crucial functions of a program work, will give you the confidence that your application actually runs and passes basic diagnostics.
* **Deploy into a copy of production**: Create a production-like or pre-production environment, identical to production, to validate changes before pushing them to production. This will eliminate mismatches and last minute surprises. A copy of production should be as close to production as possible with regards to infrastructure, operating system, databases, patches, network topology, firewalls, and configuration.
* **Instant propagation**: The first stage should be triggered upon every check-in, and each stage should trigger the next one immediately upon successful completion. If you build code hourly, acceptance tests nightly, and load tests over the weekend, you will prevent the achievement of an efficient process and a reliable feedback loop.
* **Stop the line**: When a stage in the pipeline fails, you should automatically stop the process. Fix whatever broke, and start again from scratch before doing anything else.

**Nexus/Artifactory**

**(Binary Repository Manager)**

**Guidelines and Best Practices**

# Table of Contents

[Table of Contents 2](#_Toc424649092)

[1. Introduction 3](#_Toc424649093)

[2. benefits of using repository management tool 3](#_Toc424649094)

[3. best practices of using repository management tool 3](#_Toc424649095)

[4. best practices of using Nexus 4](#_Toc424649096)

[5. nexus and alm 4](#_Toc424649097)

[6. Best practices of using artifactory 5](#_Toc424649098)

[7. artifactory and alm 8](#_Toc424649099)

[8. list of available repository management tools 8](#_Toc424649100)

# 

# Introduction

Nexus/ Artifactory are repository management tools. A repository manager is a dedicated server application designed to manage repositories of binary components. The usage of a repository manager is considered an essential best practice for any significant usage of Maven.

# benefits of using repository management tool

Following are the benefits of using a repository management tool:

* Significantly reduced number of downloads off remote repositories, saving time and bandwidth resulting in increased build performance
* Improved build stability due to reduced reliance on external repositories
* Increased performance for interaction with remote SNAPSHOT repositories
* Creates a central storage and access to artifacts and meta data about them exposing build outputs to consumer such as other projects and developers, but also QA or operations teams or even customers
* Provides an effective platform for exchanging binary artifacts within your organization and beyond, without the need for building artifact from source

# best practices of using repository management tool

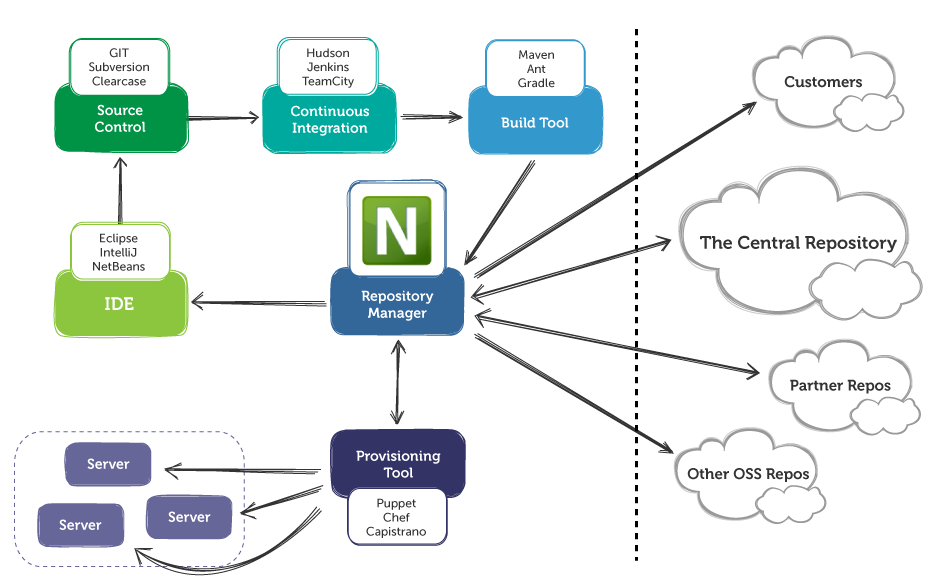
Following are the best practices of using a repository management tool:

* **Use repository management tool as proxy**: Use them as dedicated proxy server for public Maven repositories
* **Use these tools for storing and versioning of binary artifacts**: Provide repositories as a deployment destination for your Maven project outputs, using repository management tool
* **Use build promotion strategies**: Build promotion strategies basically involve moving artifacts from one repository to another depending on their stage in the build promotion process. These strategies should be implemented for promotion of artifacts from one environment to another environment and then release them once all the test criteria are passed.
* **Integration with Active Directory for Users/Roles and Privileges should be done**
* **Use security models to enhance security:** Both Artifactory and Nexus have flexible security models, build around users, user groups, and repository targets. These should be used to limit the rights of a particular user or group to a particular part of a repository
* **Mandate Usage of Checksums and PGP Signatures in Repositories**: If you are looking for an extra level of security, you should definitely start signing releases with PGP signatures
* If you happen to use ANT projects as well then you can  
  Upload/Download artifacts from nexus using **Ivy for ANT** based projects.
* **Use different repositories properly**: Repository management tools provide good support to different kinds of repositories such as Releases and Snapshots. Releases repository contains artifacts which are developed completely and which are not going to be modified. Whereas Snapshot repository contains artifacts whose development is in progress. These repository policies should be used properly while developing and deploying an artifact.

# best practices of using Nexus

* **Encourage Developers and Build & Release engineers to subscribe RSS feeds:** All developers in an organization can subscribe to the RSS feeds for New Deployed Artifacts as a way to ensure that everyone is aware when a new release has been pushed to Nexus.
* **Usage of Scheduled Task:** We can maintain and monitor health of Nexus using any of scheduled tasks, which can be set for recurrences and it also has alerting mechanisms.

# nexus and alm



# Best practices of using artifactory

* Artifactory integrates with CI server more efficiently than that of Nexus. Jenkins, hudson have good plugins to integrate with Artifactory. Following table shows possible options available for integrating Artifactory with CI server.



**Some of the best practices:-**

* Resolve project dependencies and plugins from Artifactory even though you are using some build tool.
* Store the generated artifact of the project to Artifactory.
* For deploying an artifact to the target server, artifact should be fetched from Artifactory.
* Always backup Artifactory data weekly or daily.
* For Authentication or authorization purpose of Artifactory, integrate Artifactory with LDAP or Crowd if you’re already using it.
* Artifactory should be integrated with CI server, such that you can resolve artifacts from and deploy artifacts to Artifactory using CI server.
* You should be able to tag the releases from the CI server.
* Tagged release should be displayed from the Artifactory. Moreover, you should be able to rollback a particular artifact if it does not look good.
* If you are using several technologies, (e.g. Nuget, Maven, NPM, PyPi etc..) define a unique repository for each of them. By doing that you are making sure that all of the build requests are directed to the right place rather than going to a repository that may not even have the necessary packages.
* Define a virtual repository for each of your team / builds. By only including repositories that each team/build needs in its virtual repository, you save time because requests only get sent to relevant repositories. This allows you to optimize the resolution time for your builds.Please also note that when working with different technologies, each packaging technology expects to get metadata in its corresponding format, therefore, you should define a separate virtual repository for each packaging format.
* Use Artifactory’s[Include / Exclude content feature](http://www.jfrog.com/confluence/display/RTF4/Common+Settings#CommonSettings-IncludesandExcludesPattern) – Define an exclude pattern on remote repositories to prevent Artifactory for searching through them for packages that you know cannot exist on them.

For example, if you store all of your internal proprietary packages (e.g. \*\*/mycompany/myproject.\*) in a local repository, there is no point searching for them on a remote repository. So define an exclude pattern on the remote repository that excludes it when you are resolving your local proprietary packages. This avoids redundant network traffic and ultimately saves a lot of time. Conversely, if you know that a package only exists on a specific remote repository, define an include pattern on that repository. That way, Artifactory will only try to resolve that specific package from the repository and not waste time and network resources trying to resolve other packages from that repository.

* Working with Maven – There is no need to have 5 Maven remote repositories. just point your builds at JCenter and all of your java requests will receive a quick response. JCenter is managed and maintained by JFrog and works perfectly for Artifactory users.

**Artifactory Features :**

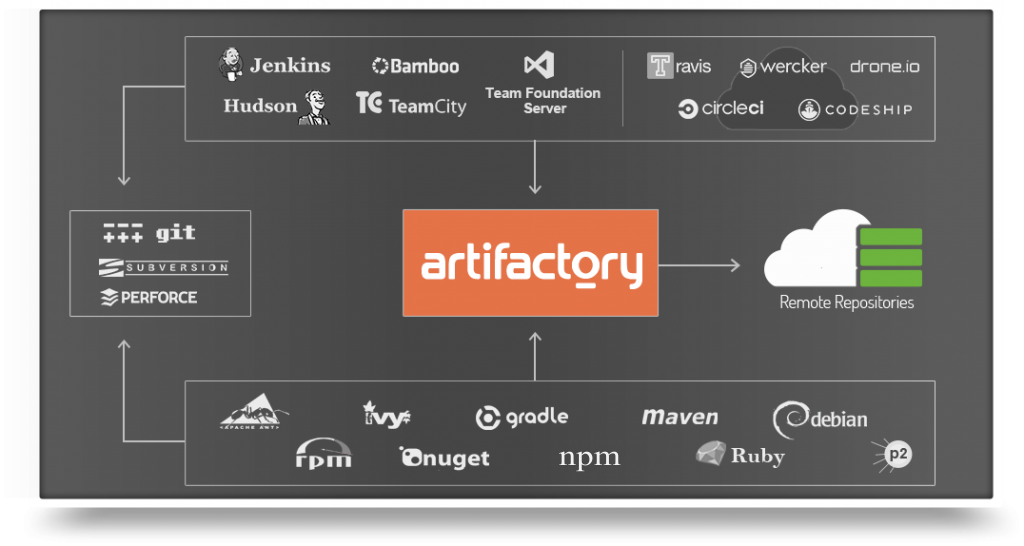
* **Checksum-based content transfer**. Only files with unmatched checksum are passed over the wire. It saves time and money.
* **Support for both push and pull modes.**To provide a firewall-friendly connection, use pull. To enforce the artifacts on another server, use push.
* **Replication of metadata, not only artifacts**. It includes all kinds of metadata (e.g. maven, user-created properties, etc.), and it also uses the checksum logic.
* **Streaming transfer for superb performance.**
* Supports p2 eclipse plugin repository, nugget, gradle, ivy, npm, yum and many more
* Supports Crowd SSO, LDAP integration
* Supports integration with defect management tool like JIRA .
* Watchers, Good UI to manage Administrative part and permissions
* Upload artifacts from UI
* Automatically generate client configurations
* Supports build promotion, after build promotions
* Manages in-house Docker containers.
* Store build data, which helps in traceability.

**Pre-Defined Local Repositories:-**

Artifactory comes with a set of pre-defined local repositories, which reflect best practices in binary repository management as follows:

|  |  |
| --- | --- |
| libs-release-local | Your code releases |
| libs-snapshot-local | Your code snapshots |
| ext-release-local | Manually deployed 3rd party libs (releases) |
| ext-snapshot-local | Manually deployed 3rd party libs (shapshots) |
| plugins-release-local | Your and 3rd party plugins (releases) |
| plugins-snapshot-local | Your and 3rd party plugins (snapshots) |

# artifactory and alm



# list of available repository management tools

* [Apache Archiva](http://archiva.apache.org/) (open source)
* [JFrog Artifactory Open Source](http://www.jfrog.com/open-source) (open source)
* [JFrog Artifactory Pro](http://www.jfrog.com/artifactory/) (commercial)
* [Sonatype Nexus OSS](http://www.sonatype.org/nexus/go/) (open source)
* [Sonatype Nexus Pro](http://links.sonatype.com/products/nexus/pro/home) (commercial)

##### 



|  |  |  |
| --- | --- | --- |
| Document name | Jenkins & Bamboo comparison | System Recruitment Specification |
| Version no. | 1.0 | 1.2 |
| Release date | 1-Dec-2020 | 1st Oct ‘09 |
| Document ID | Jenkins & Bamboo comparison document |  |
| Classification | ALM |  |

This document of Cybage Software Pvt. Ltd. is for restricted circulation. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means – recording, photocopying, electronic and mechanical, without prior written permission of Cybage Software Pvt. Ltd.

|  |
| --- |
| **J**enkins & Bamboo comparison |

Table of Contents

[1. Introduction 3](#_Toc57717843)

[2. Jenkins Vs Bamboo 3](#_Toc57717844)

[ Scalability: 7](#_Toc57717845)

[ Upgrade steps: 8](#_Toc57717846)

[ Job market and adoption rate: 8](#_Toc57717847)

[ Price and support: 8](#_Toc57717848)

[ Summary 9](#_Toc57717849)

##### Document History

| Ver. No | Release Date | Created By / Modified By and Date | Reviewed By and Date | Approved By and Date | Remarks and Changes Made |
| --- | --- | --- | --- | --- | --- |
| 1.0 | 1-Dec-2020 | Jayesh Pawar  01-Dec-2020 | Pradeep Borse  01-Dec-2020 | Devesh Yadav  01-Dec-2020 | Jenkins & Bamboo comparisons |
|  |  |  |  |  |  |

# Introduction

This document contains comparison between CI tools i.e. Jenkins & Bamboo.

# Jenkins Vs Bamboo

|  |  |  |
| --- | --- | --- |
| **Features** |  |  |
| User Friendliness | It is very user friendly and easy to  understand. | Bamboo is complex to use for  beginners. |
| Application Integration | It can be easily integrated with Atlassian tools and other tools by using available Plugins | It can be integrated with Atlassian tools and other tools via its Application Link  Feature |
| No of Plugin Available | More than 5000+ and most of the  plugins are free. | Between 200-500 and many  of the plugins are paid. |
| Code coverage | There are many plugins that  generates code coverage reports i.e. pmd,checkstyle,phpdocx etc. | No Such plugin available. |
| SonarQube Integration | Supported | Not supported by latest  Bamboo versions. |
| Build Failure on Error  Thresholds | Possible | Not possible |
| Maintainability | Jenkins stores its configuration and build result as XML. The XML approach makes Jenkins very flexible and portable. You can easily migrate Jenkins jobs across  multiple instances just by copying the configuration. | Bamboo is storing them in RDBMS. We can’t migrate the Plans to any other instances. |
| **Scalability:** To scale out CI servers, more remote slaves are needed. The more agents or slaves you have, the more horizontally scaled you  are. | You can have unlimited amount of slaves in Jenkins OSS. Jenkins enterprise is priced per slaves as well. | Bamboo license are priced per remote agents. |
| **Availability:** It is important to have your CI server with High Availability as we can't afford down time if our CI  server is being used by multiple global teams. | This is a first class support by Cloudbees in Jenkins enterprise. | Bamboo doesn't support high availability/fail over out of the box. |
| **Matrix project** | Jenkins supports Matrix project | Bamboo doesn’t support this. |

|  |  |  |
| --- | --- | --- |
| **configuration:** When a software grows, a software will need to be tested in a matrix of platform e.g. multiple databases, webservers,  jdk, etc. | configuration. |  |
| **Upgrade steps** | Easily Upgradable. | It is too heavy compared to  Jenkins |
| Price and support | Jenkins is open source; you are much more likely to have a workaround from the community until it's fixed. Jenkins Cloudbees is licensed. | You are stuck in Bamboo with Atlassian support. One of the biggest example is the build template ticket in Bamboo that raised 2 years ago with 99 votes now. In Jenkins, the plugins are available for build template supported by the community. |

**Jenkins Vs Bamboo (Description in Detail)**

* Jenkins has a Build Pipeline plugin that provides a full blown view on how the jobs are connected; this is all missing in Bamboo.
* Jenkins has large variety of plugins. Bamboo has not that much of plugins.
* Jenkins can be easily integrated with all the Atlassian tools with help of Plugin Available.

The below are the integration features that we think are important in Bamboo and see how it's being supported in Jenkins.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Product | Integration feature | Jenkins support | | | | | | |
|  |  | Yes. Can be done by poll out of the box. | | | | | | |
| Bitbucket | Bitbucket tells Bamboo when to build | Push | trigger | requires | | plugin | config: | |
|  |  | webhook & stashbot | | | | | | |
| Bitbucket | Bitbucket tells Bamboo when to update plan branches to match changes in repository branches | Yes. Use job-dsl-plugin to hit Bitbucket REST API. Workflow-plugin claims to support this as well. | | | | | | |
| Bitbucket | Bitbucket commits are displayed in the relevant Bamboo builds | Yes. Supported out of the box by Git Plugin. | | | | | | |
| Bitbucket | Bamboo notifies Bitbucket automatically about build results | Yes. Supported hitting Bitbucket API. | | | via bitbucketNotifier. | | | It’s |
| JIRA | Deployment Panel | No. We can use JIRA REST API to add a comment at the very least. | | | | | | |
| JIRA | Viewing build results in JIRA | Yes. Supported by a Bitbucket plugin or a Jenkins plugin. | | | | | | |

* Jenkins stores its configuration and build result as XML, Bamboo is storing them in RDBMS. The XML approach makes Jenkins very flexible and portable. You can easily migrate Jenkins jobs across multiple instances just by copying the configuration. You can't do this in Bamboo. Storing the configuration in XML has enabled Jenkins to be configured if they are not available in the UI such as resetting build number that Bamboo can't do.

## Scalability:

Winner: Jenkins

To scale out CI servers, more remote slaves are needed. The more agents or slaves you have, the more horizontally scaled you are. Bamboo license are priced per remote agents. You can have unlimited amount of slaves in Jenkins OSS. Jenkins enterprise is priced per slaves as well.

The steps to configure the agents in Bamboo are quite hefty. There's no automatic install of the tools hence you will need to invest in other configuration management tools e.g. ansible, chef or puppet.

Jenkins wins here with the ease of spinning up a new remote agent/slave. You can literally use any empty machine as a slave without any setup. The slaves are configured via Jenkins UI and all of the tools can be installed automatically. This is an out of the box feature from Jenkins. It also goes to the extent of utilizing unused computing power with Swarm Plugin. This plugin enables automatic slaves discovery nearby Jenkins master.

Shall you be moving to SaaS model for the entire company, a well separation of concerns in between jobs is very important. Bamboo does a very good job here by having the concept of projects and plans. One project has multiple plans, one plan has multiple jobs (and stages). With this concept, you will be able to support multiple projects well in Bamboo. Sleek!

Jenkins does the separation of jobs with folders or views, which is not as integrated as Bamboo's concept of project. A common alternative would be for the individual teams to have their own Jenkins. We are definitely moving away from SaaS model if we do this, that's why CloudBees has Jenkins Operations Center to manage multiple master nodes.

It is important to have your CI server with High Availability as we can't afford down time if our CI server is being used by multiple global teams. Bamboo doesn't support high availability/fail over out of the box, but you can achieve it traditionally. This is a first class support by Cloudbees in Jenkins enterprise.

When software grows, software will need to be tested in a matrix of platform e.g. multiple databases, webservers, jdk, etc. Jenkins supports Matrix project configuration. Configure one job, and the job will be executed repeatedly in all of the platform matrices configured. This is not supported in Bamboo.

## Upgrade steps:

Winner: Jenkins

Bamboo has a clear upgrade documentation though too hefty as compared to Jenkins. You need to do a backup for all of the home directories, babysit the update process and monitor the log files. Bamboo may also require a reindex after the upgrade.

What's not easier than dropping a war to tomcat? Jenkins is clearly a winner. Jenkins doesn't change your XMLs during the upgrade as the communities are crazy about backward compatibility. Not happy with the later version? Drop in the previous war. When you are happy with the new version, there is a click of a button away from updating all of your configuration to the latest model.

## Job market and adoption rate:

Winner: Jenkins Company Adoption

Jenkins list here :

https://wiki.jenkins-ci.org/pages/viewpage.action?pageId=58001258

* 1. Bamboo list here : https:/[/www.atl](http://www.atlassian.com/customers/?page=1&sortParam=date_created%20desc)a[ssian.com/customers/?page=1&sortParam=date\_created%20desc](http://www.atlassian.com/customers/?page=1&sortParam=date_created%20desc)

Last year's Infoq poll:

* + - Jenkins adoption rate 85%
    - Bamboo adoption rate 68%
    - Heatmap shows that users are steadily using Jenkins, while there are some heat spots of people moving away from Bamboo.
    - More details in the link: <http://www.infoq.com/research/ci-server>

## Price and support:

Winner: Jenkins

Bamboo is commercial software. Jenkins comes with two flavors, free and enterprise. The commercial support of Jenkins Enterprise comes from Cloudbees. The price can't be compared here as Cloudbees is not displaying the price in their site.

Both Jenkins and Bamboo have numerous bugs and you will definitely run into problems. Because Jenkins is open sourced, you are much more likely to have a workaround from the community until it's fixed. You are stuck in Bamboo with Atlassian support. One of the biggest example is the build template ticket in Bamboo that raised 2 years ago with 99 votes now. In Jenkins, the plugins are supported by the community.

## Summary

Bamboo is easy to use, good looking and has good Atlassian integration. Jenkins is rough around the edges, but far more flexible than Bamboo. When Jenkins is running standalone, it's obvious that it's far more superior to Bamboo. Bamboo shines when it's placed with other Atlassian applications with its good UX.

The decision of the adoption should fall down on the stakeholder’s priorities.

* Is integration more important?
* Is maintainability more important?
* Is scalability more important?

**References**:

* <https://www.atlassian.com/software/bamboo/comparison/bamboo-vs-jenkins>
* <https://valiantys.com/en/blog/atlassian-administration/jenkins-vs-bamboo/#:~:text=Jenkins%20is%20an%20open%2Dsource,more%20concerned%20with%20other%20features>.
* <https://www.edureka.co/blog/jenkins-vs-bamboo>