**Continuous Integration questions**

Now, let’s look at Continuous Integration interview questions:

**Q1. What is meant by Continuous Integration?**

Continuous Integration is a development practice in which the developers are required to commit changes to the source code in a shared repository several times a day or more frequently. Every commit made in the repository is then built. This allows the teams to detect the problems early. Apart from this, depending on the Continuous Integration tool, there are several other functions like deploying the build application on the test server, providing the concerned teams with the build and test results etc.

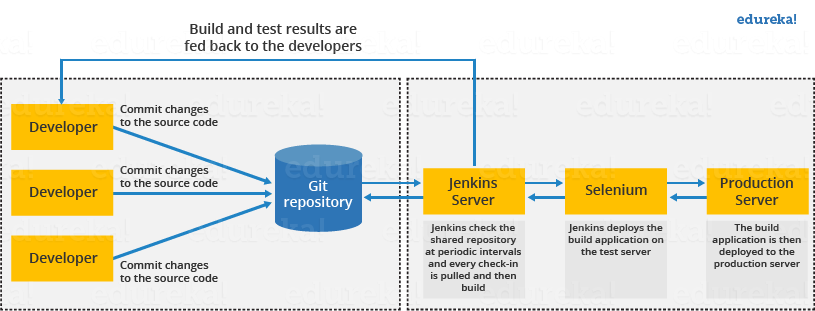
**Continuous Integration With Jenkins**

Let us imagine a scenario where the complete source code of the application was built and then deployed on test server for testing. It sounds like a perfect way to develop a software, but, this process has many flaws. I will try to explain them one by one:

* Developers have to wait till the complete software is developed for the test results.
* There is a high possibility that the test results might show multiple bugs. It was tough for developers to locate those bugs because they have to check the entire source code of the application.
* It slows the software delivery process.
* Continuous feedback pertaining to things like coding or architectural issues, build failures, test status and file release uploads was missing due to which the quality of software can go down.
* The whole process was manual which increases the risk of frequent failure.

It is evident from the above stated problems that not only the software delivery process became slow but the quality of software also went down. This leads to customer dissatisfaction. So to overcome such a chaos there was a dire need for a system to exist where developers can continuously trigger a build and test for every change made in the source code. This is what CI is all about. Jenkins is the most mature CI tool available so let us see how Continuous Integration with Jenkins overcame the above shortcomings.

I will first explain you a generic flow diagram of Continuous Integration with Jenkins so that it becomes self explanatory, how Jenkins overcomes the above shortcomings:



The above diagram is depicting the following functions:

* First, a developer commits the code to the source code repository. Meanwhile, the Jenkins server checks the repository at regular intervals for changes.
* Soon after a commit occurs, the Jenkins server detects the changes that have occurred in the source code repository. Jenkins will pull those changes and will start preparing a new build.
* If the build fails, then the concerned team will be notified.
* If built is successful, then Jenkins deploys the built in the test server.
* After testing, Jenkins generates a feedback and then notifies the developers about the build and test results.
* It will continue to check the  source code repository for changes made in the source code and the whole process keeps on repeating.

You now know how Jenkins overcomes the traditional SDLC shortcomings. The table below shows the comparison between “Before and After Jenkins”.

|  |  |
| --- | --- |
| **Before Jenkins** | **After Jenkins** |
| The entire source code was built and then tested. Locating and fixing bugs in the event of build and test failure was difficult and time consuming, which in turn slows the software delivery process. | Every commit made in the source code is built and tested. So, instead of checking the entire source code developers only need to focus on a particular commit. This leads to frequent new software releases. |
| Developers have to wait for test results | Developers know the test result of every commit made in the source code on the run. |
| The whole process is manual | You only need to commit changes to the source code and Jenkins will automate the rest of the process for you. |

**Q2. Why do you need a Continuous Integration of Dev & Testing?**

For this answer, you should focus on the need of Continuous Integration. My suggestion would be to mention the below explanation in your answer:  
Continuous Integration of Dev and Testing improves the quality of software, and reduces the time taken to deliver it, by replacing the traditional practice of testing after completing all development. It allows Dev team to easily detect and locate problems early because developers need to integrate code into a shared repository several times a day (more frequently). Each check-in is then automatically tested.

**Q3. What are the success factors for Continuous Integration?**

Here you have to mention the requirements for Continuous Integration. You could include the following points in your answer:

* Maintain a code repository
* Automate the build
* Make the build self-testing
* Everyone commits to the baseline every day
* Every commit (to baseline) should be built
* Keep the build fast
* Test in a clone of the production environment
* Make it easy to get the latest deliverables
* Everyone can see the results of the latest build
* Automate deployment

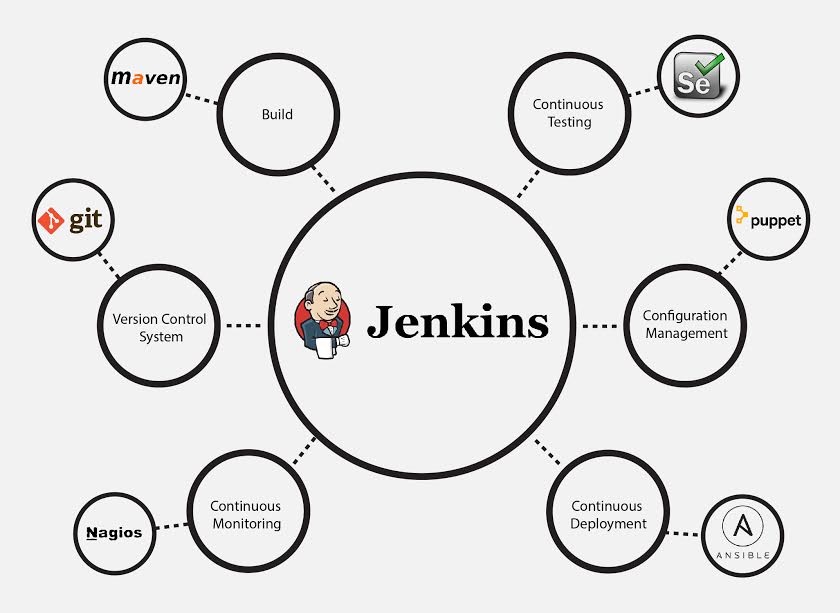
**Q4. What is Jenkins?**

Jenkins is an open source automation tool written in Java with plugins built for Continuous Integration purpose. Jenkins is used to build and test your software projects continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build. It also allows you to continuously deliver your software by integrating with a large number of testing and deployment technologies.

With Jenkins, organizations can accelerate the software development process through automation. Jenkins integrates development life-cycle processes of all kinds, including build, document, test, package, stage, deploy, static analysis and much more.

Jenkins achieves Continuous Integration with the help of plugins. Plugins allows the integration of Various DevOps stages. If you want to integrate a particular tool, you need to install the plugins for that tool. For example: Git, Maven 2 project, Amazon EC2, HTML publisher etc.

The image below depicts that Jenkins is integrating various DevOps stages:



Advantages of Jenkins include:

* It is an open source tool with great community support.
* It is easy to install.
* It has 1000+ plugins to ease your work. If a plugin does not exist, you can code it and share with the community.
* It is free of cost.
* It is built with Java and hence, it is portable to all the major platforms.

There are certain things about Jenkins that separates it from other the Continuous Integration tool. Let us take a look on those points.

**Q5. Explain how you can move or copy Jenkins from one server to another?**

I will approach this task by copying the jobs directory from the old server to the new one. There are multiple ways to do that; I have mentioned them below:   
You can:

* Move a job from one installation of Jenkins to another by simply copying the corresponding job directory.
* Make a copy of an existing job by making a clone of a job directory by a different name.
* Rename an existing job by renaming a directory. Note that if you change a job name you will need to change any other job that tries to call the renamed job.

**Q6. Explain how can create a backup and copy files in Jenkins?**

Answer to this question is really direct. To create a backup, all you need to do is to periodically back up your JENKINS\_HOME directory. This contains all of your build jobs configurations, your slave node configurations, and your build history. To create a back-up of your Jenkins setup, just copy this directory. You can also copy a job directory to clone or replicate a job or rename the directory.

**Q7. Explain how you can setup Jenkins job?**

My approach to this answer will be to first mention how to create Jenkins job. Go to Jenkins top page, select “New Job”, then choose “Build a free-style software project”.  
Then you can tell the elements of this freestyle job:

* Optional SCM, such as CVS or Subversion where your source code resides.
* Optional triggers to control when Jenkins will perform builds.
* Some sort of build script that performs the build (ant, maven, shell script, batch file, etc.) where the real work happens.
* Optional steps to collect information out of the build, such as archiving the artifacts and/or recording javadoc and test results.
* Optional steps to notify other people/systems with the build result, such as sending e-mails, IMs, updating issue tracker, etc..

**Q8. Mention some of the useful plugins in Jenkins.**

Below, I have mentioned some important Plugins:

* Maven 2 project
* Amazon EC2
* HTML publisher
* Copy artifact
* Join
* Green Balls

These Plugins, I feel are the most useful plugins. If you want to include any other Plugin that is not mentioned above, you can add them as well. But, make sure you first mention the above stated plugins and then add your own.

**Q9. How will you secure Jenkins?**

The way I secure Jenkins is mentioned below. If you have any other way of doing it, please mention it in the comments section below:

* Ensure global security is on.
* Ensure that Jenkins is integrated with my company’s user directory with appropriate plugin.
* Ensure that matrix/Project matrix is enabled to fine tune access.
* Automate the process of setting rights/privileges in Jenkins with custom version controlled script.
* Limit physical access to Jenkins data/folders.
* Periodically run security audits on same.