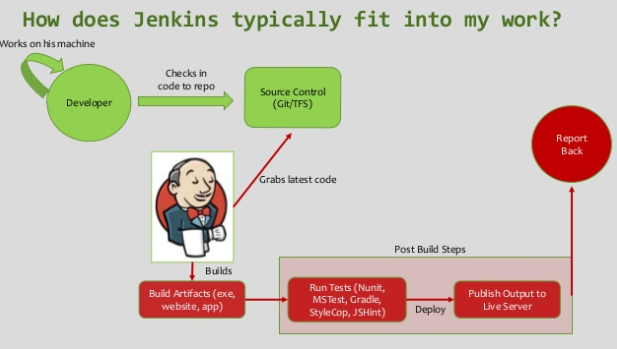
Jenkins is a Java application (platform independent)

used for continuous integration and continuous delivery ( CI/CD) tool.



### Continuous integration

Developers practicing continuous integration merge their changes back to the main branch as often as possible. The developer's changes are validated by creating a build and running automated tests against the build. By doing so, you avoid the integration hell that usually happens when people wait for release day to merge their changes into the release branch.

Continuous integration puts a great emphasis on testing automation to check that the application is not broken whenever new commits are integrated into the main branch.

### Continuous delivery

Continuous delivery is an extension of continuous integration to make sure that you can release new changes to your customers quickly in a sustainable way. This means that on top of having automated your testing, you also have automated your release process and you can deploy your application at any point of time by clicking on a button.

In theory, with continuous delivery, you can decide to release daily, weekly, fortnightly, or whatever suits your business requirements. However, if you truly want to get the benefits of continuous delivery, you should deploy to production as early as possible to make sure that you release small batches that are easy to troubleshoot in case of a problem.

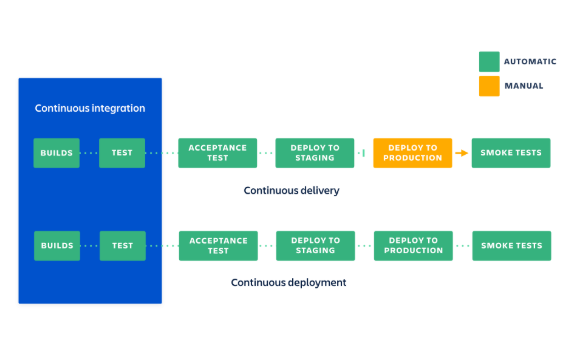
### Continuous deployment

Continuous deployment goes one step further than continuous delivery. With this practice, every change that passes all stages of your production pipeline is released to your customers. There's no human intervention, and only a failed test will prevent a new change to be deployed to production.

Continuous deployment is an excellent way to accelerate the feedback loop with your customers and take pressure off the team as there isn't a Release Day anymore. Developers can focus on building software, and they see their work go live minutes after they've finished working on it.

### How the practices relate to each other

To put it simply continuous integration is part of both continuous delivery and continuous deployment. And continuous deployment is like continuous delivery, except that releases happen automatically.



Jenkins comes with default servers jetty and winstone server.

Download Jenkins

1. Jenkins.io
2. Place the war file into any location on your system
3. Go to command prompt and to the folder
4. java -jar jenkins.war (winstone is the server)/ java -jar jenkins.war --httpPort=9191
5. access on browser
6. enter admin password
7. select plugins to install
8. All/None/Suggested: select suggested and install
9. Username: / continue as admin

\*C:\Users\10663655\.jenkins

\*jenkins/hudson.model.UpdateCenter.xml and change url to use http instead of https

Build periodically: means we can assign some expression and based on that expression our job will be executed at interval.

` \* \* \* \* \* : make Jenkins run after every minute.

H/15 \* \* \* \* : every 15 minutes

Poll SCM: We have our source code management system and in case we have configured any SCM like Git repository. As soon as a build takes place in the repository our job should get executed.

Build:

Execute windows batch command: javac test.java

Post build action: leave blank

Available: Github integration plugin

Why should we deploy Jenkins on tomcat?

Using tomcat, we can start all applications of tomcat server.

Required:

Tomcat 5 or above

Java 7 or above

Copy the Jenkins.war file inside tomcat/webapps folder

Go to tomcat bin directory

Make all files executable: chmod +x \*.sh

Start tomcat: ./startup.sh

Verify tomcat started: <http://localhost:8080>

Run Jenkins: <http://localhost:8080/jenkins>

Examples:

Start build daily at 08:30 in the morning, Monday - Friday: 30 08 \* \* 1-5

Weekday daily build twice a day, at lunchtime 12:00 and midnight 00:00, Sunday to Thursday: 00 0,12 \* \* 0-4

Start build daily in the late afternoon between 4:00 p.m. - 4:59 p.m. or 16:00 -16:59 depending on the projects hash: H 16 \* \* 1-5

Start build at midnight: @midnight or start build at midnight, every Saturday: 59 23 \* \* 6

Every first of every month between 2:00 a.m. - 02:30 a.m.: H(0,30) 02 01 \* \*

@hourly, @daily, @weekly, @monthly, @midnight

@hourly --> Build every hour at the beginning of the hour --> 0 \* \* \* \*

@daily, @midnight --> Build every day at midnight --> 0 0 \* \* \*

@weekly --> Build every week at midnight on Sunday morning --> 0 0 \* \* 0

@monthly --> Build every month at midnight of the first day of the month --> 0 0 1 \* \*

To schedule a cron job every 5 minutes, you need to define the cron settings like this:

\*/5 \* \* \* \*

Build every hour:

H \* \* \* \*

Build every 20 minutes:

H/20 \* \* \* \*

Build every 20 minutes 2am to 11pm:

H/20 5-23 \* \* \*

Build every 20 minutes, work time/days (8am-6pm, MON-FRI) only:

H/20 8-18 \* \* 1-5

Build every hour MON-WED and FRI only:

H \* \* \* 1-3,5

Build every hour, weekends in April and December:

H \* \* 4,12 \*

Build at 8.30am on July 4:

30 8 4 7 \*

https://www.lenar.io/jenkins-schedule-build-periodically/

### Build Triggers

**What is a trigger?**

A trigger lets us execute a job on an event occurrence. This event is called a trigger. To see the list of build triggers, we need to login to Jenkins and click on any item already created and click on configure.

* **Trigger build remotely**: The job is usually triggered by accessing a specified URL. This is convenient for scripts. With the URL, one needs to mention the authorization token as well.
* **Build after other projects are built**: As it reads, we need to mention the list of other projects, once those projects are built then the present job is executed.
* **Build periodically**: The build is triggered based on the mentioned time. A cron has to be mentioned here.
* **Github hook trigger for GITSCM polling**: If Jenkins receives push GitHub hook from a repository associated with git, then the build process gets executed.
* **Poll SCM**: Configure Jenkins to poll the SCM for any pushes or commits and then trigger the jobs.

Poll SCM" polls the SCM periodically for checking if any changes/ new commits were made and shall build the project if any new commits were pushed since the last build, whereas the "build" shall build the project periodically irrespective to whether or not any changes were made.

# **Auto build a job in jenkins if there is any change in code on Github repository**

Follow these steps.

1. Open Jenkins dashboard. Click on manage jenkins

2. Click on Configure system and under github configuration click advanced tab.

3. Check**'**Specify another hook url**'** for GitHub configuration.

4. Now you will get a url in the textbox. Copy this url as it is required in the next steps.

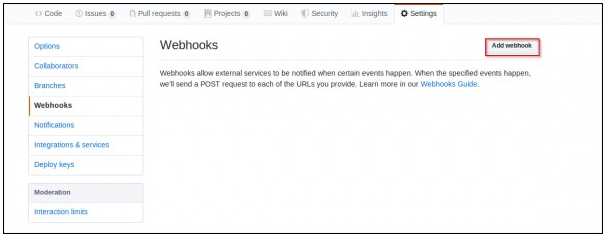
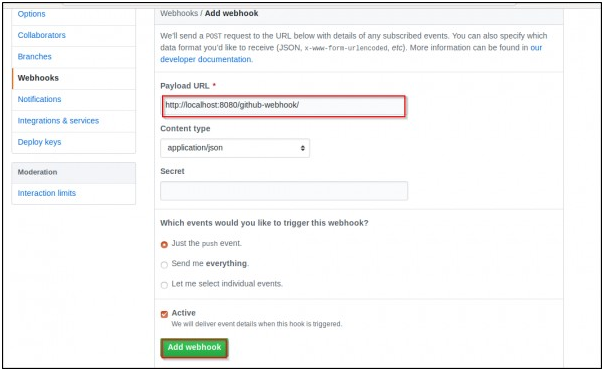
5. Now open your github repository. Go to settings -> webhooks -> add webhooks.

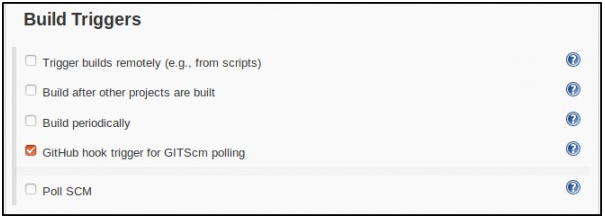
6. Now paste the url from step 4 in the payload url section. Next click on just push the event Now you should be able to see the added webhook in the list of webhooks.

7. Now go to jenkins dashboard. Go to your project configuration. In the build triggers section select github hook trigger for git scm polling. Save the changes.

OR

To automatically run build on code commit, follow these steps:

1. **Configuring Jenkins:**   
   Jenkins listens for POST requests at a Hook URL. We need to give this URL to the repository on GitHub. Then, whenever code is pushed to that repository, GitHub will send a POST request to the Hook URL and Jenkins will run the build.
   * To get the Hook URL of Jenkins, Open the Jenkins Dashboard.
   * Go to: **Manage Jenkins** > **Configure System**
   * Under GitHub Plugin Configuration, Click on ‘**Advanced**' tab.   
     
   * Check**'Specify another hook url'** for GitHub configuration.   
       
     
   * A textbox will appear with a hook URL. This is the Hook URL at which Jenkins will listen for POST requests.**Copy this URL** and go to the next step.
2. **Configuring GitHub Repository:**   
   We now have to provide the Hook URL we got from Jenkins in the previous step.
   * Open your repository on GitHub.
   * Click ‘**Settings**’ on the navigation bar on the right-hand side of the screen.
   * Click ‘**Webhooks** ’ on the navigation bar on the left-hand side of the screen.
   * Click ‘**Add webhook**’ to add the webhook.
   * 
   * **Paste the URL** you copied in the previous step as the ‘**Payload URL**’.
   * You can select the events for which you want the Jenkins build to be triggered. We will select ‘**Just the push event**’ because we want to run the build when we push our code to the repository.
   * Click ‘**Add webhook**’.   
     
   * You should now see the webhook you just added in the list of Webhooks for that repository.
3. **Configuring Jenkins Project :**We now have Jenkins configured to run builds automatically when code is pushed to central repositories. However, Jenkins doesn’t run all builds for all projects. To specify which project builds need to run, we have to modify the project configuration.
   * In Jenkins, go to the**project configuration** of the project for which you want to run an automated build.
   * In the ‘Build Triggers’ section, select **'Github hook trigger for GITScm Polling'.**

  
​

* + **Save** your project.

Jenkins will now run the build whenever you push/commit your code to the GitHub repository.

---------------- Deploy web app to tomcat ---------------------------------------

1. tested out web app successfully.
2. configured tomcat server
3. pushed the project on git hub
4. jenkin configuration:

a. git,maven and deploytocontainer is installed

b. install maven integration plugin

c. create new job as maven project

d. add git repository path

e. add pom.xml as “root-path/pom.xml”

f. add goal : clean install

5. post-build action:

Select deploy war/ear to container

Add tomcat user credential

Enter tomcat url

Apply and save.

Build now.

web app on git hub

|

build trigger: build periodically/ poll scm

|

build action : maven build package install

|

post-build action: deploy the war to container (tomcat)