

Abstract

The following document demonstrates how to trigger build in Jenkins when anyone commits a change in the GitHub repository.

The document also demonstrates how to configure Jenkins to send an email to the Configured Email ID when any build fails along with the Build Log file.

Contents

Abstract	1
Prerequisites.....	3
Configuring GitHub	4
Configuring Jenkins	7
Creating a New Project	9

Prerequisites

- **Jenkins Up and Running.**
- **GitHub** account
- **GitHub Repository** with a **Maven Project**
- **Git Plugin installed** on **Jenkins**

Configuring GitHub

Step 1:

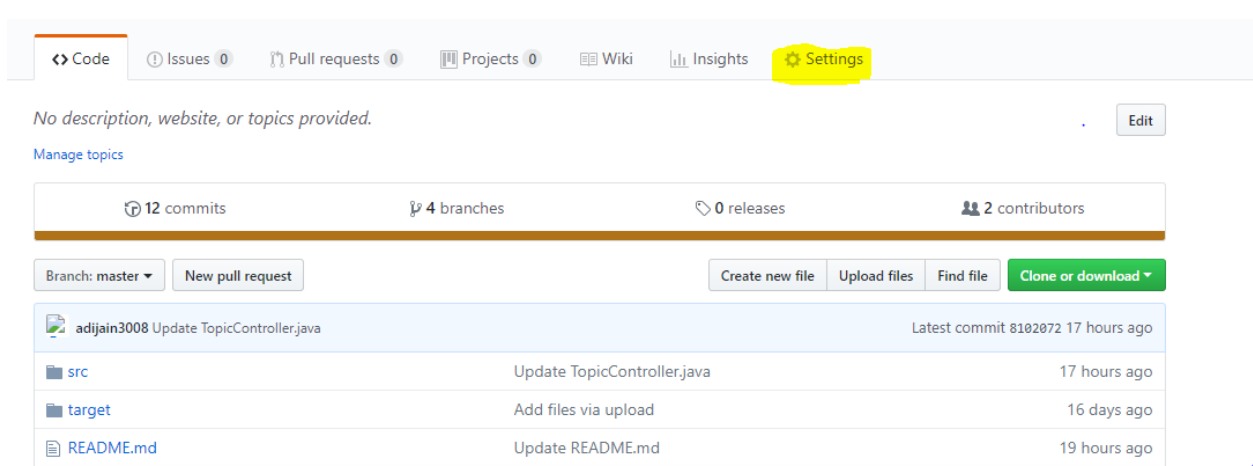
Open <https://github.com/> and login with your ID and Password

Step 2:

Open the Repository which contains the Maven Project

Step 3:

Click on the Settings Button

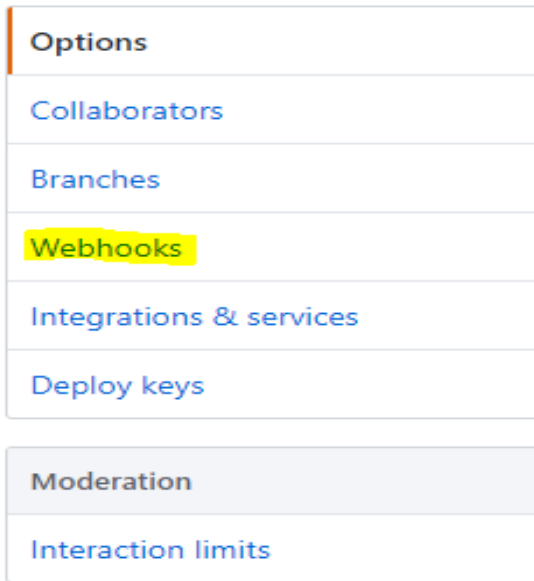


The screenshot shows a GitHub repository page for a user named 'adijain3008'. The repository is named 'Update TopicController.java'. The 'Settings' button in the top navigation bar is highlighted in yellow. Below the repository name, there is a table showing the commit history. The table has three columns: file name, commit message, and time ago. The latest commit is 'Update TopicController.java' from 17 hours ago. The repository has 12 commits, 4 branches, 0 releases, and 2 contributors.

File	Commit Message	Time Ago
src	Update TopicController.java	17 hours ago
target	Add files via upload	16 days ago
README.md	Update README.md	19 hours ago

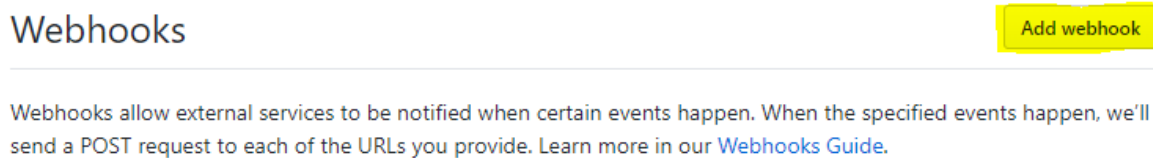
Step 4:

Click on the Webhooks Button



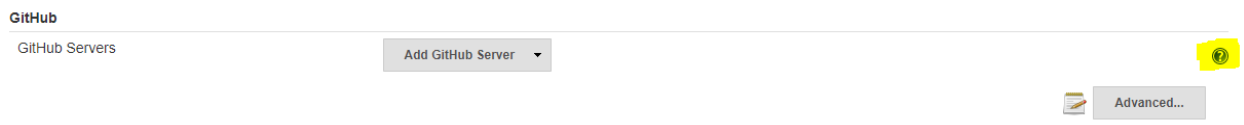
Step 5:

Click on Add webhook Button



Step 6:

Go to Jenkins → Manage Jenkins → Configure System and scroll down to GitHub section. Click on this button



Step 7:

Copy this URL

By default

This plugin doesn't do anything with the GitHub API unless you add a configuration with credentials. So if you don't want to add any configuration, you can set up hooks for this Jenkins instance manually.

In this mode, in addition to configuring projects with "Build when a change is pushed to GitHub", you need to ensure that Jenkins gets a POST to its <http://yuhur54667dns.westus.cloudapp.azure.com:8080/github-webhook/>

Step 8:

Paste it in Payload URL section and Click on Add Webhook Button

Payload URL *

http://yuhur54667dns.westus.cloudapp.azure.com:8080/github-webhook/

Content type

application/x-www-form-urlencoded

Secret

Which events would you like to trigger this webhook?

☒ Just the push event.

☐ Send me everything.

☐ Let me select individual events.

☒ **Active**

We will deliver event details when this hook is triggered.

Add webhook

Step 9:

Your Webhook has been added

Webhooks

Add webhook

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our [Webhooks Guide](#).

✓ <http://yuhur54667dns.westus.cloudapp.azure.com:8080/github-webhook/> (push)

Edit

Delete

Configuring Jenkins

Step 1:

Go to Jenkins → Manage Jenkins → Configure System → Extended Email Notification Section

Step 2:

Write SMTP Server address of the sender's Email ID in the SMTP Server Textbox (In this case Gmail). Then click on Advanced Button

Extended E-mail Notification

SMTP server	<input type="text" value="smtp.gmail.com"/>	?
Default user E-mail suffix	<input type="text"/>	?

 Advanced...

Step 3:

Select the Use SMTP Authentication and Use SSL CheckBox. Add Sender's Email ID and Password. Write SMTP Port as 465 (for Gmail)

<input checked="" type="checkbox"/> Use SMTP Authentication	
User Name	<input type="text" value="redacted"/>
Password	<input type="password" value="....."/>
Advanced Email Properties	<input type="text"/>
Use SSL	<input checked="" type="checkbox"/>
SMTP port	<input type="text" value="465"/>

Step 4:

Scroll down to E-Mail Notification Section and follow the above 2 Steps again

E-mail Notification	
SMTP server	smtp.gmail.com
Default user e-mail suffix	
<input checked="" type="checkbox"/> Use SMTP Authentication	
User Name	original@domain.com
Password
Use SSL	<input checked="" type="checkbox"/>
SMTP Port	465

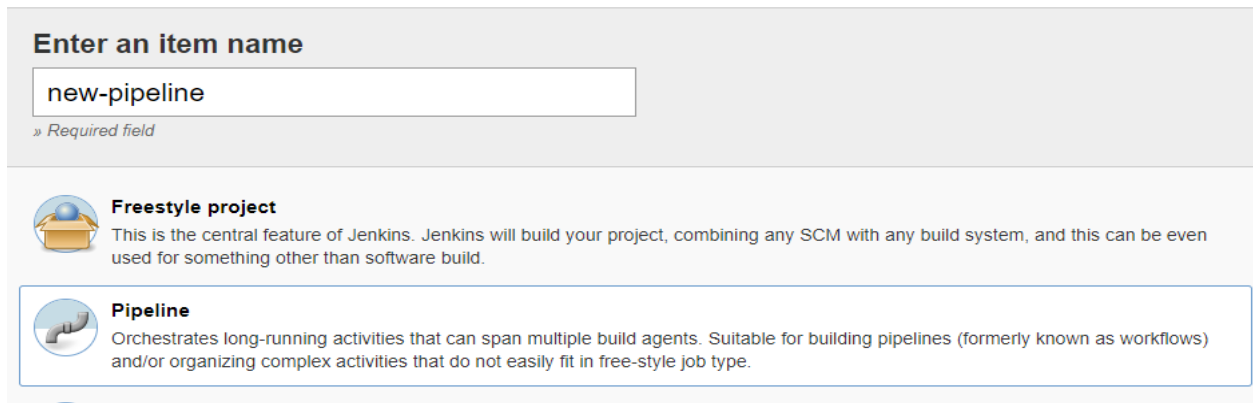
Step 5:

Click on Save Button

Creating a New Project

Step 1:

Click on New Item. Give a name for the project and Select Pipeline Project and Click OK



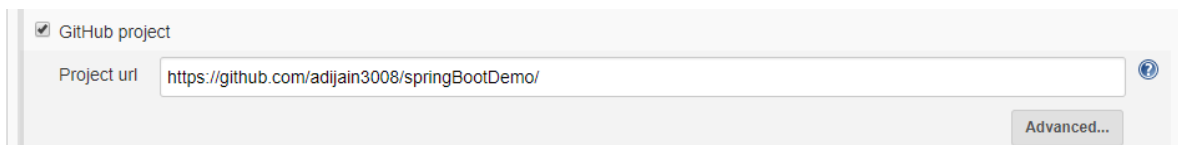
The screenshot shows the 'Enter an item name' dialog box in Jenkins. It has a text input field containing 'new-pipeline' and a 'Required field' message below it. Below the dialog box, there are two options: 'Freestyle project' and 'Pipeline'. The 'Pipeline' option is highlighted with a blue border. The 'Freestyle project' option has a description: 'This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.' The 'Pipeline' option has a description: 'Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.'

Step 2:

Click on your Project Name and then Click on Configure

Step 3:

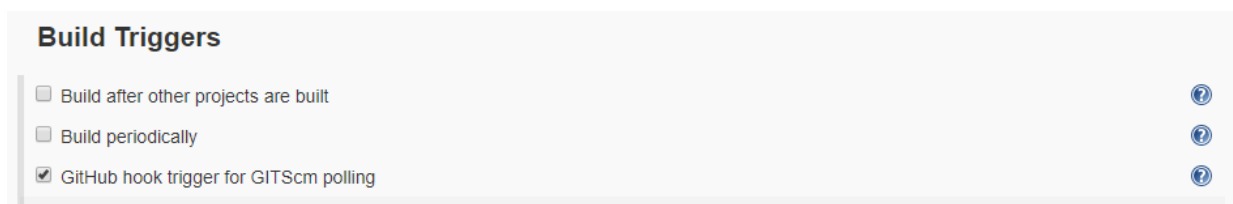
Provide the Git Repository Pages URL in GitHub Project TextBox



The screenshot shows the 'GitHub project' configuration dialog box in Jenkins. It has a checkbox labeled 'GitHub project' which is checked. Below it, there is a 'Project url' text input field containing 'https://github.com/adijain3008/springBootDemo/'. To the right of the input field is a help icon. At the bottom right, there is an 'Advanced...' button.

Step 4:

Select the GitHub hook trigger for GITScm polling CheckBox in Build Triggers Section



The screenshot shows the 'Build Triggers' section in Jenkins. It has three checkboxes: 'Build after other projects are built', 'Build periodically', and 'GitHub hook trigger for GITScm polling'. The 'GitHub hook trigger for GITScm polling' checkbox is checked. To the right of each checkbox is a help icon.

Step 5:

Enter the following Script in the Pipeline Section

```
node {  
  
    def mvn_version = 'M3'  
  
    try{  
  
        stage('Preparation') {  
  
            git 'https://github.com/adijain3008/springBootDemo.git'  
  
            withEnv( ["PATH+MAVEN=${tool mvn_version}/bin"] ) {  
  
                sh "mvn clean install"  
  
            }  
  
        }  
  
    }catch(err){  
  
        emailx attachLog: true, body: 'Build failed', subject: 'Build Failure', to: 'receiver's-email-address'  
  
        currentBuild.result = 'FAILURE'  
  
    }  
}
```



The screenshot shows the Jenkins Pipeline configuration interface. The 'Definition' dropdown is set to 'Pipeline script'. The 'Script' section contains a Groovy script for a pipeline. The script defines a stage named 'Preparation' which clones a repository and runs 'mvn clean install'. It includes a catch block for errors, sending an email and setting the build result to 'FAILURE'. The 'Use Groovy Sandbox' checkbox is checked. A 'Pipeline Syntax' link is visible at the bottom.

```
1 node {  
2     def mvn_version = 'M3'  
3  
4     try{  
5  
6         stage('Preparation') {  
7             git 'https://github.com/adijain3008/springBootDemo.git'  
8             withEnv( ["PATH+MAVEN=${tool mvn_version}/bin"] ) {  
9                 sh "mvn clean install"  
10            }  
11        }  
12    }catch(err){  
13        emailx attachLog: true, body: 'Build failed', subject: 'Build Failure', to: 'aditya.jain@aditya.com'  
14        currentBuild.result = 'FAILURE'  
15    }  
16 }
```

☒ Use Groovy Sandbox

[Pipeline Syntax](#)

Step 6:

Click on Save to Save the Changes

Step 7:

Create any error in your GitHub project and Commit the Changes

Step 8:

A build should be triggered in Jenkins as soon as the commit is saved

Step 9:

Check your supplied mail address for a mail with the error log and supplied message



Build failed