

#Devops Assembly Line Assignment

Creating a Jenkins Pipeline Which Integrates the Development, Testing and Deployment Environment ultimately resulting end to end automation.

#Assuming that git, docker and Jenkins are installed in the respective systems and Jenkins has been assigned all permissions in sudoers file in Redhat 8.

Also, assuming the dev user knows all basic git commands. If not you can [click here](#) to go through all git commands.

Step 1 → Start Jenkins from your operational environment (#here I am using RHEL version 8 for the same which is running via virtualization (i.e. VMware) on Windows 10 as host OS)

Step 2 → Ask your developer to launch his environment (#here I am using Windows 10 as development environment)

Step 3 → Here I am using my GitHub repository as remote server to upload local repo files using git commit and push. Create a new repository without checking on initializing readme.md and remotely add it to your local Git repo.

Step 4 → Now in local development environment, I have created a new branch i.e. testing branch along with the master. (master is default)

➔ The command used to create branch is (git branch testing) and to switch (git checkout testing)

➔ Another way is to create and switch at a time by using (git checkout -b testing)

```
MINGW64:/c/Users/Anuddeeph Nalla/Desktop/Devops/dev_test_deploy
Anuddeeph Nalla@DESKTOP-00CUC4C MINGW64 ~
$ cd Desktop
Anuddeeph Nalla@DESKTOP-00CUC4C MINGW64 ~/Desktop
$ cd Devops
Anuddeeph Nalla@DESKTOP-00CUC4C MINGW64 ~/Desktop/Devops
$ cd dev_test_deploy/
Anuddeeph Nalla@DESKTOP-00CUC4C MINGW64 ~/Desktop/Devops/dev_test_deploy (master)
$ git branch -a
* master
  testing
  remotes/origin/master
  remotes/origin/testing
Anuddeeph Nalla@DESKTOP-00CUC4C MINGW64 ~/Desktop/Devops/dev_test_deploy (master)
$
```

Step 5 → I have a file linux.html in master branch which is also pushed/uploaded to testing branch. (using git push -u origin testing # you have to do this for the first time only after that git commit and push is enough)

```
MINGW64:/c/Users/Anuddeeph Nalla/Desktop/Devops/dev_test_deploy
Anuddeeph Nalla@DESKTOP-00CUC4C MINGW64 ~
$ cd Desktop
Anuddeeph Nalla@DESKTOP-00CUC4C MINGW64 ~/Desktop
$ cd Devops
Anuddeeph Nalla@DESKTOP-00CUC4C MINGW64 ~/Desktop/Devops
$ cd dev_test_deploy/
Anuddeeph Nalla@DESKTOP-00CUC4C MINGW64 ~/Desktop/Devops/dev_test_deploy (master)
$ git branch -a
* master
  testing
  remotes/origin/master
  remotes/origin/testing
Anuddeeph Nalla@DESKTOP-00CUC4C MINGW64 ~/Desktop/Devops/dev_test_deploy (master)
$ ls
linux.html
Anuddeeph Nalla@DESKTOP-00CUC4C MINGW64 ~/Desktop/Devops/dev_test_deploy (master)
$ ls -la
./ ../ .git/ linux.html
Anuddeeph Nalla@DESKTOP-00CUC4C MINGW64 ~/Desktop/Devops/dev_test_deploy (master)
$
```

Step 7 → Now after modifying something in my testing branch, I decided to give this to my Quality Assurance Team i.e. operations' team to check the code on their environment for testing and approval. (Using RHEL_8 here)

Step 8 → Open Jenkins dashboard and create a new job named as Testing Job for the Testing_of_code. In that put Follow these steps, Here I am using a docker container with httpd image to generate a testing environment running on port

8086. The container is attached to the directory test-ws, test-ws was a directory created in RHEL-8.

The screenshot shows the 'Source Code Management' configuration page in Jenkins. The 'Git' option is selected under 'Source Code Management'. The 'Repository URL' is set to 'https://github.com/Anuddeeph/Dev_test_deploy.git'. The 'Credentials' dropdown is set to '- none -'. The 'Branches to build' section has a 'Branch Specifier (blank for \'any\')' set to '*/testing'. The 'Repository browser' is set to '(Auto)'. The 'Additional Behaviours' section has an 'Add' button. Below this, the 'Build Triggers' section is partially visible, showing options like 'Trigger builds remotely (e.g., from scripts)', 'Build after other projects are built', 'Build periodically', 'GitHub hook trigger for GITScm polling' (which is checked), and 'Poll SCM'.

→ Don't forget to mention the testing branch!!

The screenshot shows the 'Build Triggers' and 'Build' configuration pages in Jenkins. In the 'Build Triggers' section, the 'GitHub hook trigger for GITScm polling' checkbox is checked. In the 'Build' section, the 'Execute shell' step is configured with the following command:

```
sudo cp -vrf * /test-ws
#testing
if sudo docker ps | grep testing_environment
then
sudo echo " Already running"
else
sudo docker run -dit -p 8086:80 -v /test-ws:/usr/local/apache2/htdocs --name testing_environment httpd
fi
```

Below the command field, there is a link to 'See the list of available environment variables'. At the bottom, there are 'Save' and 'Apply' buttons.

➔ Do not forget to tick on GitHub hook trigger for GITScm polling.

The Job firstly copies the downloaded data to test-ws from GitHub, when triggered by webhooks(explained further) and then checks if a container named testing_environment is running, if not then it launches the container of same name with httpd image on port 8086, here test-ws works as mounted volume for the container, and we also exposing the webserver using -p.

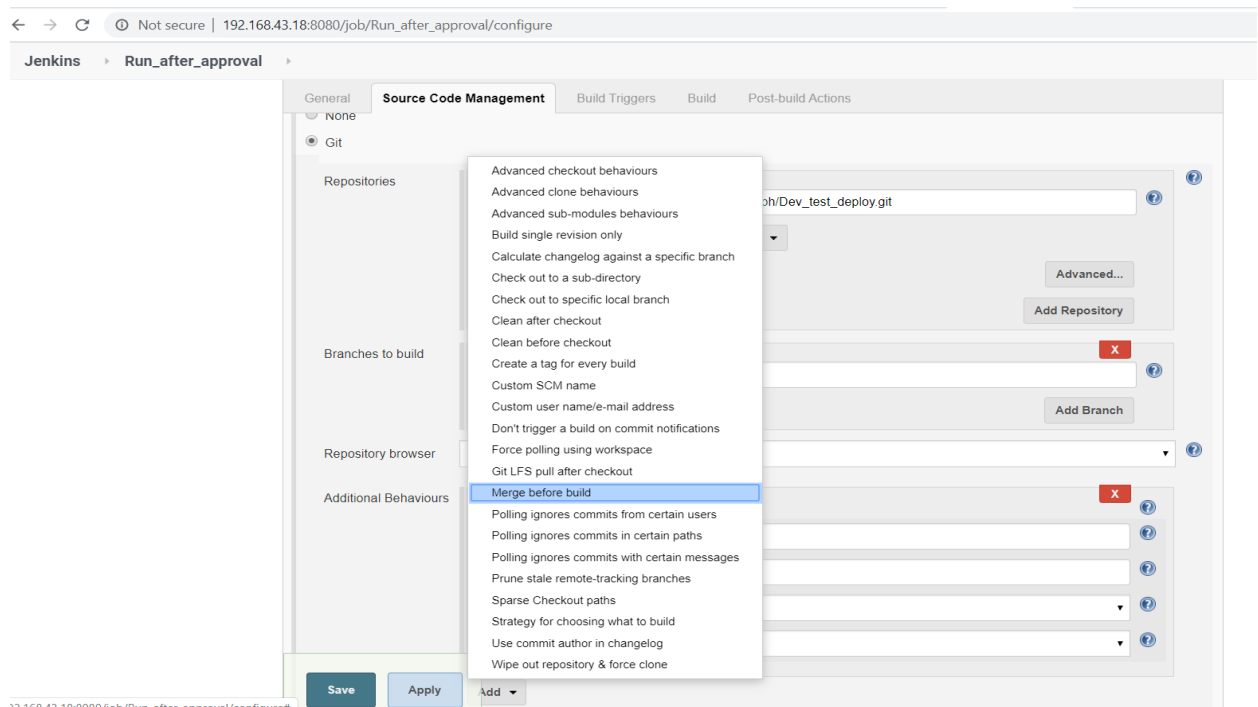
The Job is triggered on GitHub webhooks. For this give the path of your GitHub repository and go to your GitHub repository settings.

- ➔ Open GitHub, go to the settings, select webhooks.
- ➔ Add Webhook
- ➔ Switch to RHEL-8, for tunnel, and run ngrok program (i.e. #./ngrok http 8080)
- ➔ Copy the ngrok link into the webhook and add last /github-webhook/ (for example <https://45c5f4eb.ngrok.io/github/webhook/>)
- ➔ Select application/json in the Content type box.
- ➔ Select your desired options
- ➔ Checkbox active and Update webhook.
- ➔ Also go to notifications option located under webhooks and place the email id of operation's team/(here for learning yours) to get notified whenever a new update is performed in the repository.

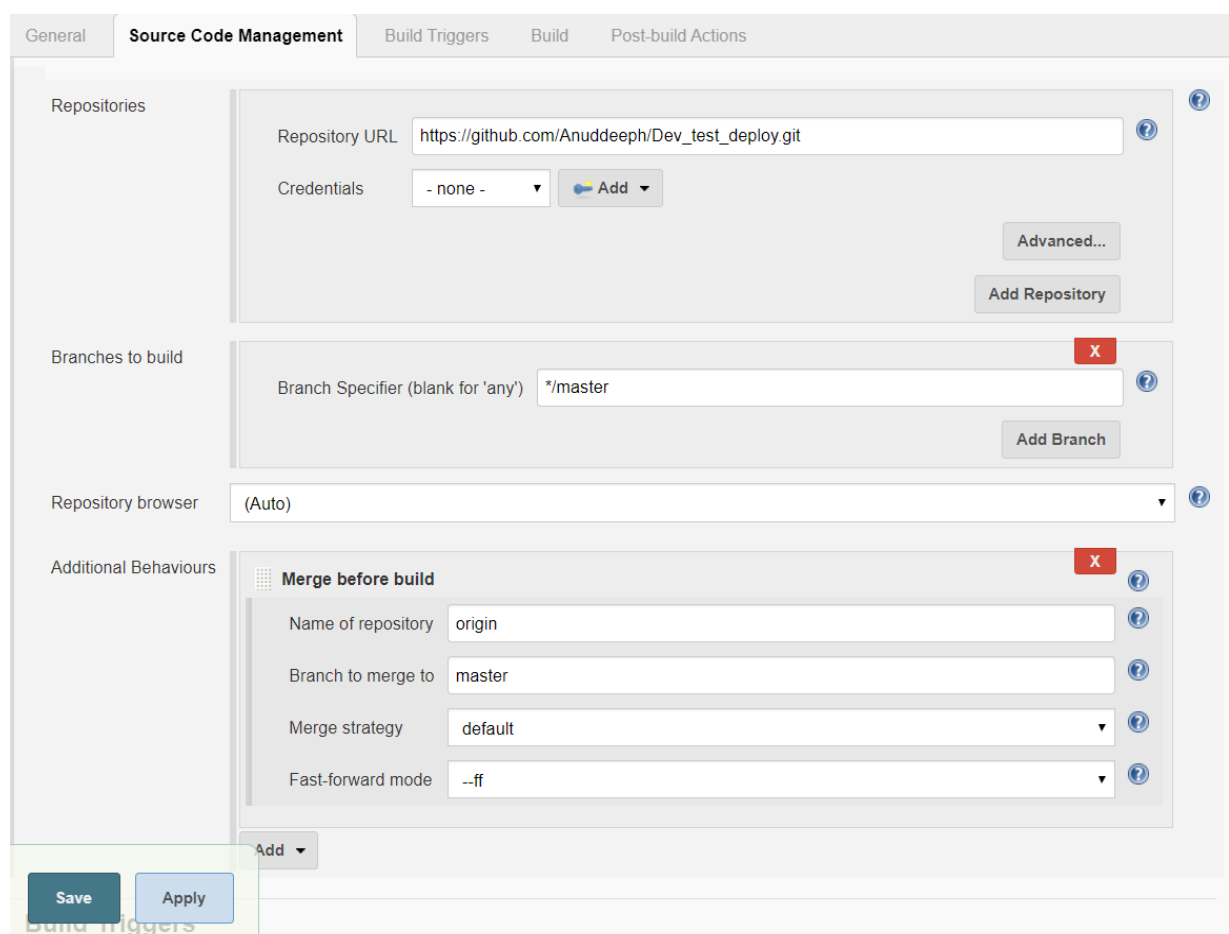
Options	Webhooks / Manage webhook
Manage access	We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in our developer documentation .
Branches	
Webhooks	
Notifications	Payload URL *
Integrations	<input type="text" value="https://45c5f4eb.ngrok.io/github-webhook/"/>
Deploy keys	Content type
Secrets	<input type="text" value="application/json"/>
Actions	Secret
	<input type="text"/>
	SSL verification
	<input type="checkbox"/> By default, we verify SSL certificates when delivering payloads.
	<input checked="" type="radio"/> Enable SSL verification <input type="radio"/> Disable (not recommended)
	Which events would you like to trigger this webhook?
	<input checked="" type="radio"/> Just the push event.
	<input type="radio"/> Send me everything.
	<input type="radio"/> Let me select individual events.
	<input checked="" type="checkbox"/> Active
	We will deliver event details when this hook is triggered.
	<input type="button" value="Update webhook"/> <input type="button" value="Delete webhook"/>

Options	Notifications
Manage access	Setup email addresses to receive notifications when push events are triggered.
Branches	
Webhooks	
Notifications	
Integrations	
Deploy keys	
Secrets	
Actions	

Step 6 → Fall back to your Jenkins dashboard and create another new job named as Run_after_approval as this is the job you will / QAT team will build/run this job when it checks the testing environment and the site/code is ready to be launched for clients.



➔ Go to the additional behaviors and select “Merge before build”.



➔ Input all the details and branch names accordingly.

#If merge conflicts occur please go manually and solve the conflict first and then run this job.

Step 7 ➔ Again in Jenkins dashboard, create a Job named as `Deployment_to_Production_Environment`. This job is to be run just after the QAT team approves and builds “Run_after_approval job.” So, to chain this job to the former one goes to Build Triggers and tick Build after other projects are build, then mention the job name you want to chain it (Here Run after approval.)

The screenshot shows the Jenkins job configuration page for a job named "tion_environment". The "Source Code Management" tab is selected. Under "Repositories", the "Repository URL" is set to "https://github.com/Anuddeeph/Dev_test_deploy.git" and "Credentials" is set to "- none -". The "Branches to build" section shows a "Branch Specifier (blank for 'any')" set to "*/master". The "Repository browser" is set to "(Auto)". The "Build Triggers" section has the checkbox "Build after other projects are built" checked, and "Projects to watch" is set to "Run_after_approval,". The "Trigger only if build is stable" radio button is selected. At the bottom, there are "Save" and "Apply" buttons.

Step 8 ➔ The following commands theory is same as the `testing_environment_job` build theory and only difference is that it is pulling the code from the master branch to `deploy-ws` directory and deploying it to the `production_environment` running on port 8087.

GeneralSource Code ManagementBuild TriggersBuildPost-build Actions

Trigger even if the build fails

☐ Build periodically

☒ GitHub hook trigger for GITScm polling

☐ Poll SCM

Build

Execute shell

Command

```
sudo cp -vrf * /deploy-ws
#production
if sudo docker ps | grep production_environment
then
sudo echo " Already running"
else
sudo docker run -dit -p 8087:80 -v /deploy-ws:/usr/local/apache2/htdocs --name production_environment ht
fi
```

See [the list of available environment variables](#)

Advanced...

Add build step

Post-build Actions

Add post-build action

Save

Apply

**FINALLY, RUN ALL THE JOBS ACCORDINGLY
GIVEN INSTRUCTIONS AND YOUR END TO
END JENKINS PIPELINE INTEGRATING DEV,
GIT, GITHUB, DOCKER CONTAINERS, OPs are
Now fully functional.**

