DEVOPS

What is Jenkins

Phone/Whatsapp: +1 (515) 309-7846 (USA)

Email: info@zarantech.com

Website: www.zarantech.com

Contents

What is Jenkins? 3
What is Continuous Integration? 3
A Sample Use Case in Jenkins 4
What is Jenkins? How is Jenkins used for Continuous Integration? 5
How does Jenkins work? 6
Jenkins Master–Slave Installation on AWS 6
Configuring the Slaves in Jenkins 7
Installing Jenkins Plugins 9
Creating Jenkins Builds 11
Creating Scheduled Builds 11
GIT Webhook 12
Jenkins Pipelines 14
Conclusion 18

DevOps Tools

What is Jenkins?

Prior to delving into the main topic of "What is Jenkins?," it's essential to commence this tutorial with an introduction to continuous integration

What is Continuous Integration?

Continuous integration (CI) happens to be one of the most vital parts of DevOps. It is primarily used to integrate various stages of DevOps together. In other words, it is a coding practice that essentially enables the development team to make and implement small changes in the code and version control methods quite frequently.

Continuous integration is usually done in the form where all developers push the code onto a shared repository mostly multiple times a day. It is pretty fit for a project that should be coded and developed on different platforms with multiple tools. Currently, it has become important to have such a mechanism in place to integrate and validate the changes made to the code in a parallel way.

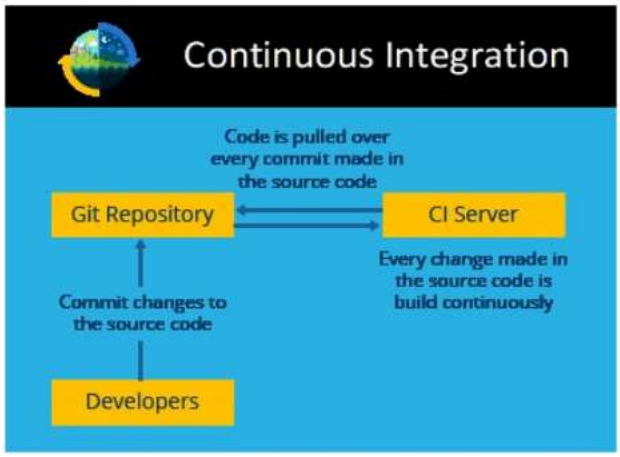
- Reduction of integration complexities: Collaborative development in projects heightens the risk of errors during integration, especially in complex code. Continuous integration (CI) facilitates regular integration, alleviating these challenges.
- Enhanced code quality: Reduced risks allow for a shift of resources and time towards creating more functional and robust code.
- Effective version control: CI triggers notifications for any commits that disrupt the build, preventing the distribution of flawed code.
- Streamlined testing: Preservation of different code versions and builds simplifies the work of quality assurance professionals in identifying, locating, and tracing bugs efficiently.
- Efficient deployment process: The automated deployment process conserves time and resources.
- Increased confidence: The absence of potential failures fosters peace of mind among developers, resulting in heightened productivity and superior product quality.

A Sample Use Case in Jenkins

Consider an existing system that is meant to create a healthcare system that pulls the code from the shared repository at a certain time every day and builds it. This is something like CI except for the fact that it only builds once a day. This essentially leads to finding bugs only at one point of time in a day.

An alternate way to this system is to make it CI-compliant, i.e., to enable the system to push the code onto the shared repository every time a change is made to it and build it.

This will make sure that all the developers, who work on the system and are making changes to the latest code, set, find, and resolve bugs as soon as they appear in the system. This also ensures an up-to-date software rollout especially with critical systems such as healthcare.



What is Jenkins? How is Jenkins used for Continuous Integration?

Jenkins, a Java-based automation tool, boasts built-in plugins tailored for continuous integration tasks. It optimizes the continuous building and testing of projects, simplifying the incorporation of evolving code changes.

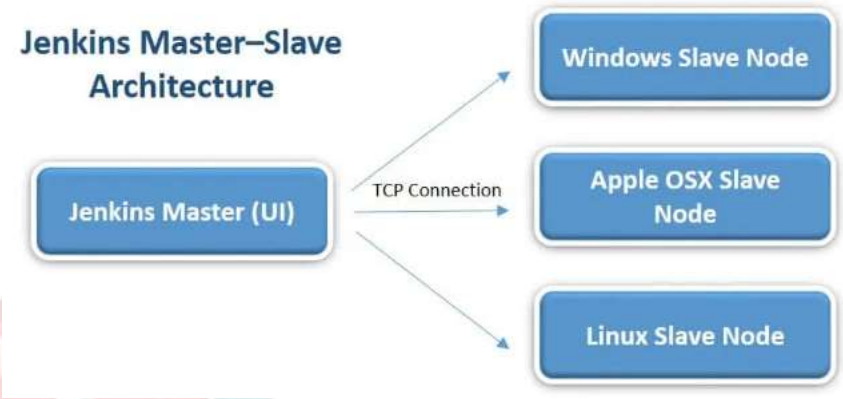
Jenkins facilitates swift software delivery by seamlessly interfacing with a wide array of deployment and testing technologies. Additionally, it expedites development by automating tasks. Functioning as a server-based application, Jenkins necessitates a web server like Tomcat.

The tool gained prominence due to its adeptness in monitoring recurring tasks. In a project development scenario, Jenkins continuously scrutinizes and assesses the code, thereby enabling early detection of potential errors or failures in the development phase.

How does Jenkins work?

Standalone Jenkins instances can be demanding on disk and CPU resources. To mitigate this, scaling can be achieved by introducing slave nodes, which effectively assist in sharing the workload of the master node. A slave node functions as an executor on behalf of the master. While the master handles basic operations and serves the user interface, the slaves perform the actual work.

In the illustration below, the Jenkins master manages the user interface, while the slave nodes operate on various OS types, executing tasks as directed by the master.



Jenkins Master–Slave Installation on AWS

Creating a Master

Step 1: Create a new EC2 instance with Amazon Linux AMI 2016.03

Step 2: Update the system and install Jenkins on it with the following commands:

$yum update –y$ wget -O /etc/yum.repos.d/jenkins.repo
https://pkg.jenkins.io/redhat/jenkins.repo

$ rpm –import http://pkg.jenkins-ci.org/redhat/jenkins-ci.org.key

Yum install –y Jenkins

Step 3: Update /etc/sysconfig/Jenkins to authorize Jenkins to access the environment variables used by Jenkins plugins and remember to change the default time zone to the one we live in.

Step 4: Now that we have updated it, we need to only register and start it. Do so with the following command:

$chkconfig jenkins on$service jenkins start

Step 5: Now Jenkins is enabled and is running. Proceed to the site http://SERVER IP :8080. When being redirected to the 'Getting Started' screen, retrieve the password and unlock Jenkins

Configuring the Slaves in Jenkins

We will have to configure the slaves and update them in the master configuration.

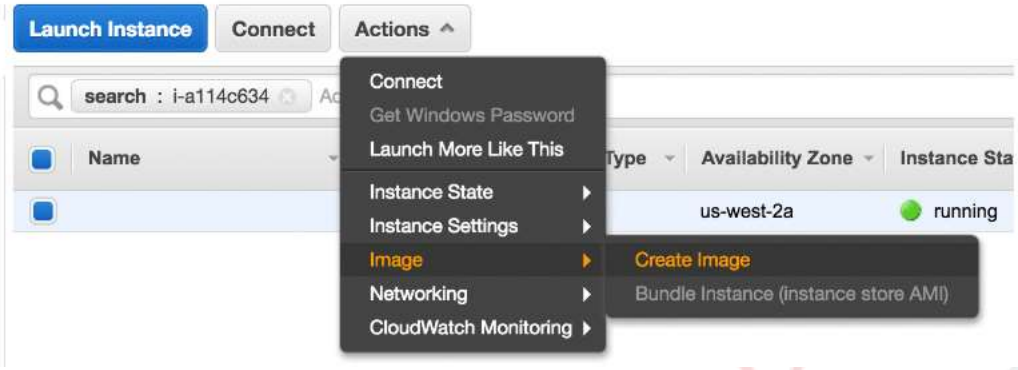
Step 1: Create a new EC2 instance just as before with either the sudo or the root access

Step 2: Add the base dependencies like Java, Git, Docker, and so on with the following commands:

$ yum install -y docker git java-1.8.0-openjdk$ curl -L https://github.com/docker/compose/releases/download/1.6.2/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose$ chmod +x /usr/local/bin/docker-compose

Step 3:

Create the AMI on AWS; on the EC2 panel, go to Launch Instance and click on the slave that we recently configured and create a new image; once created, note down the AMI ID



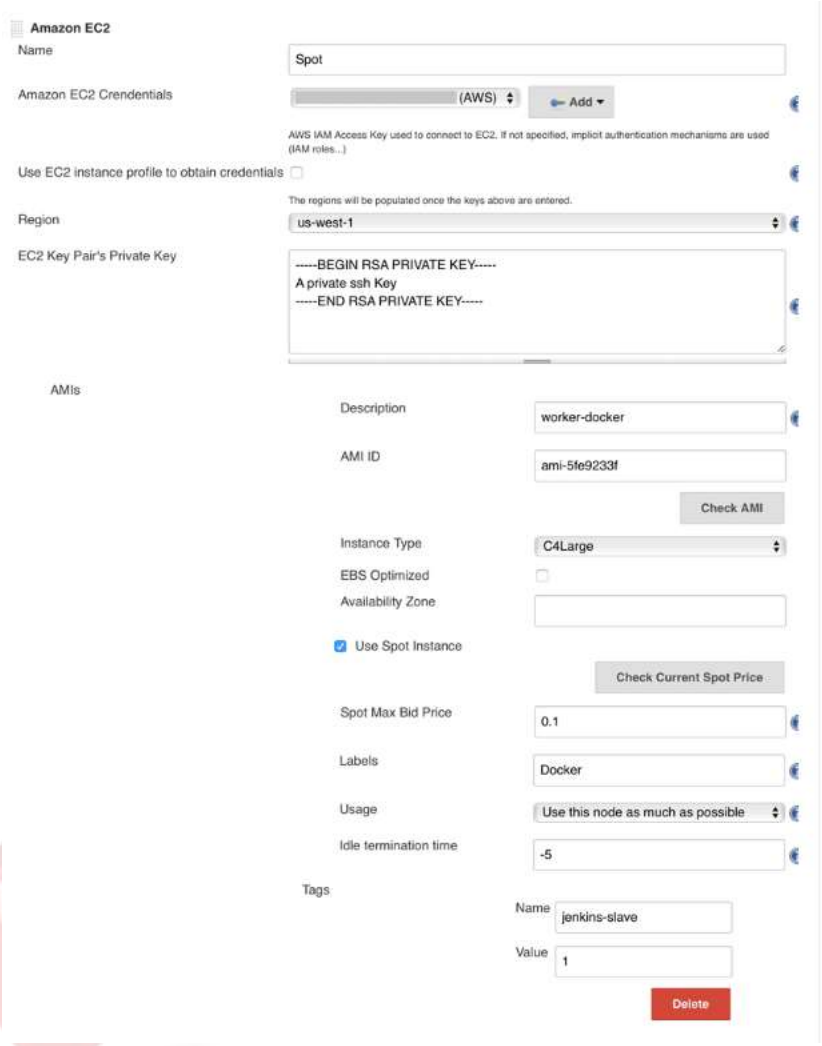
Step 4: Now, configure the master to use the AMI; to do so, follow the below steps:

- Create AWS credentials
- Once chosen, limit the scope to the system and complete the form
- For adding AWS as the cloud providers, follow:

Manage Jenkins > configure the system > add a new cloud > Amazon EC2 > complete the form as needed

Now, we have a master and a slave configured successfully!

Refer to the screenshot below to see the sample setup screen. We may practice Jenkins installation with other specifications of our choice. This is only the default and the generalized format.

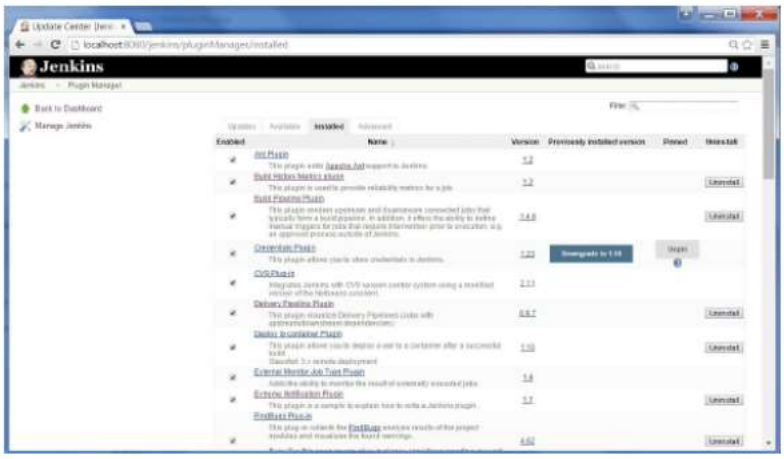


Installing Jenkins Plugins

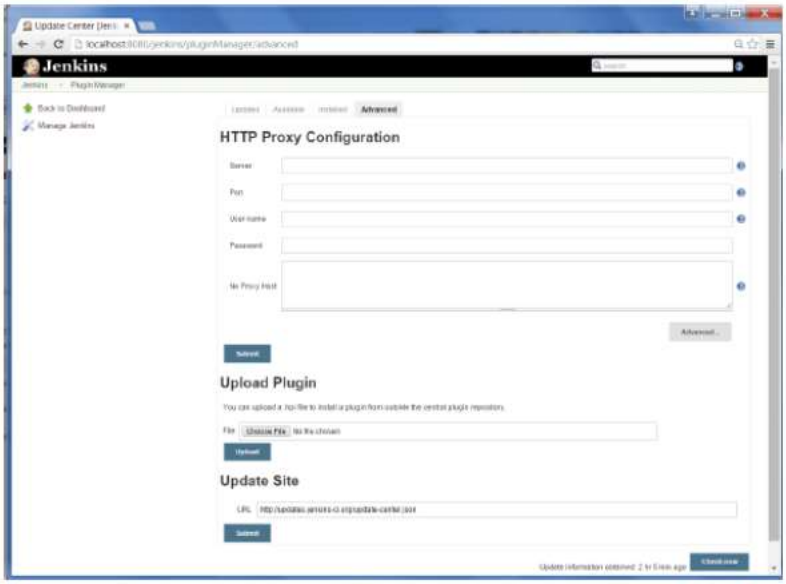
One of Jenkins' central features is its extensibility through the integration of plugins. These extensions enhance the core functionality, delivering more robust tools for project management.

To manage Jenkins plugins:

- After logging in, navigate to the 'Manage Jenkins' tab located on the left-hand side to handle installed plugins and add or remove new ones.
- Within the 'Manage Plugins' tab, plugins can be searched or all available plugins can be viewed.
- Select a plugin and click 'Install without restart' to expedite the installation process and promptly assess its functionality, avoiding the need to restart Jenkins.
- To uninstall a plugin, go to the 'Installed' tab, select the plugin for removal, and click 'Uninstall'. It is essential to restart Jenkins for the changes to take effect.



- In some cases, an older version of a plugin may be required, necessitating manual download and upload to Jenkins.
- Custom plugins developed by users should be uploaded to the site to contribute to the expansion of the community base.



Creating Jenkins Builds

A build is often called when the source code is converted into a usable and runnable form. It allows compiling the code into an executable form. The process of building is typically handled by the build tool.

Builds are usually done when we reach a critical standpoint such as the integration of a feature or so on. As Jenkins is CI-based, we have a powerful feature where we can automate the build process to happen at a particular time or event. This is called as 'scheduled builds.'

Creating Scheduled Builds

In this Jenkins tutorial, we will explore the process of scheduling builds at specific times and triggers by following these steps:

Step 1: Navigate to the 'Build Triggers' section and select the 'Build periodically' checkbox.

Step 2: Enter the scheduling parameters, including the minute (0-59), hour (0-23), day (1-31), month (1-12), and day of the week (0-7), in the provided text box. This syntax guides the scheduling process.

Here are some examples of scheduling builds:

- Starting the build every day at 8:30 AM from Monday to Friday: 30 08 * * 1-5
- Initiating the daily build in the afternoon from 4:00 PM to 4:59 PM, depending on the project's hash: H 16 * * 1-5
- Commencing the build at midnight: @midnight or 59 23 * * 6
- Triggering the build every hour: H * * * *

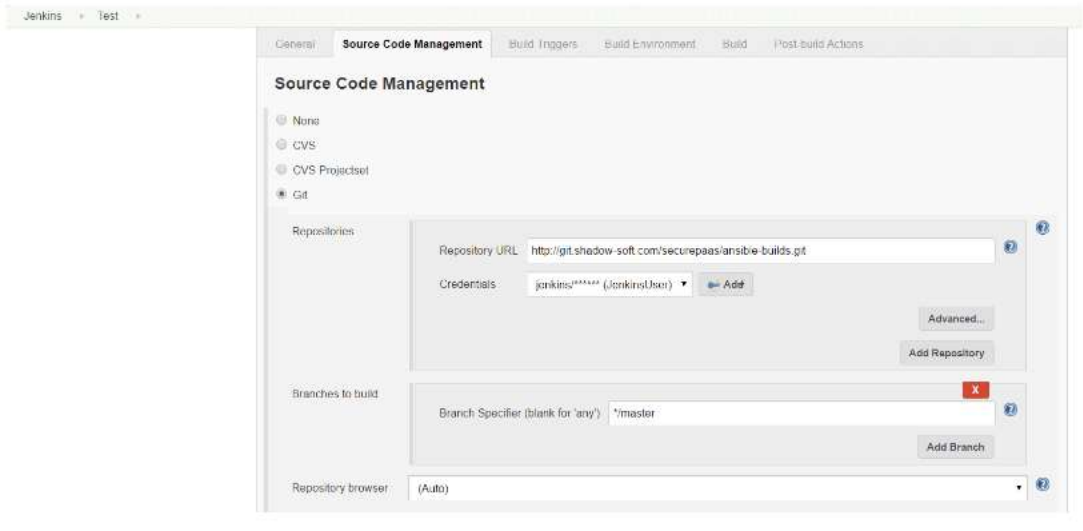
GIT Webhook

We have covered automating builds in Jenkins based on specific times or dates. Now, let's explore automating builds whenever code is pushed.

To achieve this:

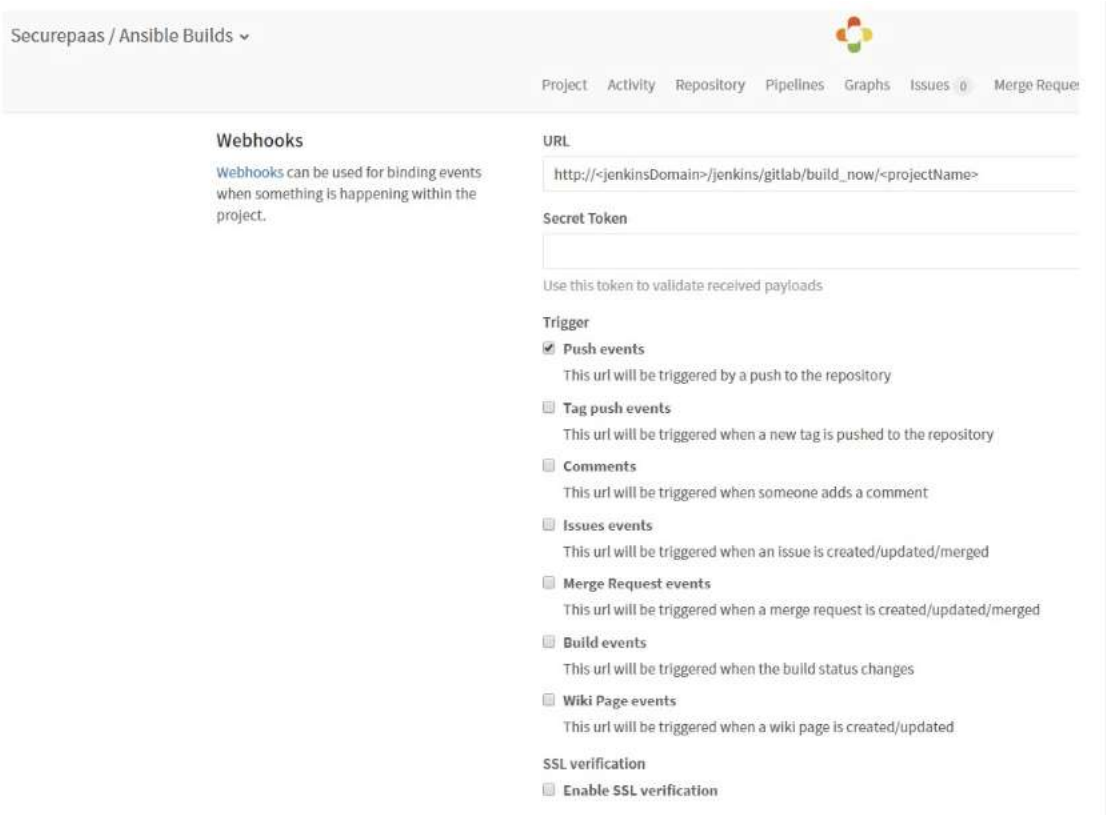
- Install the 'GitLab Hook Plugin' following the steps provided in the earlier sections.
- If a new project needs to be created, navigate to 'Source code management,' and input the Git repository's URL and necessary credentials.
- Specify the desired build steps by navigating to 'Add build setup' and adding a build step.

It's essential to note that the specific steps to add will depend on the intended actions and the environment. Refer to the setup page screenshot for further guidance.



To add a webhook for the repository in GitLab, follow these steps:

- Go to the instance, select the cog icon, and choose the webhook option.
- Input the following URL in the designated field: http:///jenkins /gitlab/build_now/
- Click on the "Add webhook" button located at the end of the page.



Jenkins Pipelines

What is CI/CD?

In its simplest form, it is a coding practice that compels the development team to make small changes and frequently perform version control checks.

The primary objective of CI, as discussed earlier, is to establish an automated method for building and testing applications. This approach is expected to lead to higher software quality due to more frequent changes and improved collaboration within the team. On the other hand, continuous delivery (CD) is an extension of CI that continues the process where CI concludes.

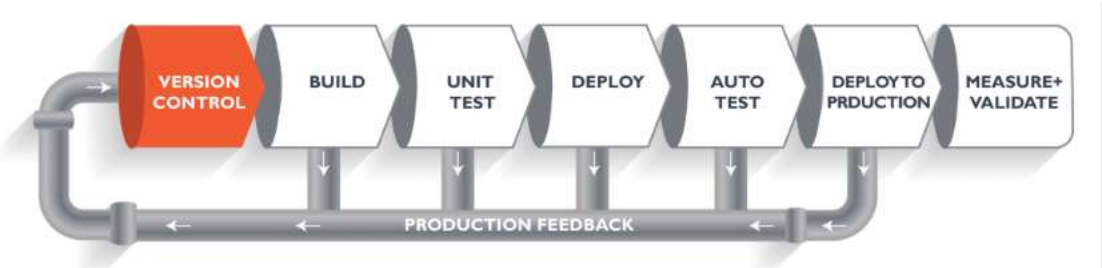
CD allows for the automation of all releases to the defined infrastructure. It essentially ensures an automated method for pushing code changes, as well as performing necessary service calls to servers, databases, and other components that may require a restart.

What is CI/CD Jenkins Pipeline?

The 'CI/CD Jenkins Pipeline' refers to the amalgamation of tasks and jobs responsible for transforming source code into a release-ready form. These tasks are organized into a pipeline structure, where the completion of one task with a successful status triggers the automated commencement of the next task in the sequence. This can also be termed as a ‘CD Pipeline,’ ‘Deployment Pipeline,’ or ‘Software Dev Pipeline.’

A supervisory application handles the management, execution, monitoring, and reporting of the various pipeline components as they progress through the sequence. The real-world implementation of a Jenkins pipeline can differ based on the specific project requirements. While the overall workflow remains consistent, variations arise from the specifics of source tracking, building, gathering metrics, and testing relevant to the project.

Creating a CI/CD Jenkins Pipeline



Jenkins serves as a robust tool for automating the entire DevOps process through integration with various interfaces and tools. Typically, the Dev team commits their code to a GIT repository.

Jenkins initiates the process by employing a frontend tool to define the job or task. Subsequently, it pulls the code and progresses to the commit phase. Next, the code undergoes the build phase wherein it is compiled. Finally, with the aid of Docker, the code is deployed to the staging area.

Moving forward, we will delve into creating the Jenkins pipeline using Jenkins and Docker.

Step 1: Access the terminal in the VM and execute the following command:

systemctl start Jenkins

systemctl enable Jenkins

systemctl start Docker

Step 2: Access Jenkins on the designated port and click on the option to create a new job.

Step 3: Choose the 'freestyle' option and input the desired item name.

Step 4: Configure the 'Source code management' by specifying the GIT repository. Then, click on Apply and save.

Step 5: Navigate to the build section and select 'execute shell.'

Step 6: Input the shell commands to generate a wat file. This process involves pulling up the code, installing the necessary package and dependencies, and compiling the application.

Step 7: Repeat Steps 3–5 to set up integration and initiate the build process in Docker through the provided shell commands.

Step 8: Repeat Steps 3–5, using a different job name and entering the relevant shell commands. This step involves checking the Docker container file and deploying it to the predefined port.

Step 9: To configure the jobs, access Job1 and proceed to 'Post build actions' and 'Build other projects' respectively.

Step 10: Specify the project name and save the configuration.

Step 11: Repeat Step 9 to configure Job2.

Step 12: To create a pipeline view, click on the plus (+) symbol, choose the build pipeline view, and enter a view name.

Step 13: Select 'Run' to initiate the CI/CD process in the system.

Step 14: Upon completion of the build, access 'localhost:8180/sample.text' to run the application.

Conclusion

As we conclude this Jenkins tutorial, let's recap the central concepts encompassing Jenkins, including its role in the CI/CD space, the setup of Jenkins, creation of masters and slaves, and the extensive array of available plugins. It's recommended to explore the installation and contribution of new plugins to the community to enrich the Jenkins ecosystem.

Delving deeper into this tutorial, we've acquired insights into building creation and scheduling based on specific requirements. Additionally, we've delved into the fundamental significance of creating CI/CD pipelines. Hence, the next step involves crafting our own jobs and scheduling them with diverse parameters to gain a comprehensive understanding of Jenkins.

Jenkins stands as a pivotal DevOps tool at our disposal. Mastering its functionalities and skills can prove to be an invaluable asset. For those seeking further knowledge and certification, Intellipaat offers top-notch trainers and e-learning courses in DevOps. Embark on your journey to become DevOps certified today!

THANK YOU

Corporate Training Course Catalog

https://lnkd.in/g38XTcqU

Salesforce Learner Community

https://www.linkedin.com/showcase/cloud-technology-learner-community

Get any Salesforce Video Training

https://zarantech.teachable.com/courses/category/cloud-computing

Phone/Whatsapp: +1 (515) 309-7846

Email: info@zarantech.com

www.zarantech.com