International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 05 | May 2020 www.irjet.net p-ISSN: 2395-0072

A Central Provisioning Service for Sharable, Stable and Scalable Build Agents in Jenkins

Tarun Chaitanya Varma.K[1], Mohan Kumar.K[2], Meghana Amirineni[3], Rakesh Reddy.N[4],

[1,2,3,4] *Computer Science & Engineering, Lendi Institute of Engineering and Technology, Vizianagaram*

---***---

Abstract - *In the software development practice, project code is integrated and build under various operating System environments. For this build, the developer needs different OS environments. In continuous integration, the code is integrated and build in different target platforms like Windows, Linux by using some services. We provide one of those services to build the code. Our service is Jeeves. It will provide the multiple OS required by the user as agents. Jeeves uses the cloud service to provide multiple OS such as AWS and Digital Ocean. We use this service in the Continuous Integration tool which is Jenkins. In Jenkins, we Initialize Jeeves service as a Jenkins plugin. Jeeves has a user interface that is used to display the agents which are available to build. When the user wanted to build the code he requests the agent to Jeeves then Jeeves looks for an available agent. If the agent is available it will lease the agent and after build is done the user will return the agent to the Jeeves and it will place it in the agent pool. The reason for distributed agents is for scale, or to provide different tools, or build on different target platforms like Windows, Mac, and Linux. This service is developed by java, nodejs, and operated on Windows or Linux operating systems. By using this we perform Static agents sharing among different Jenkins server fleet.*

***Key Words*: Continuous Integration, Automated Build, Distributed Agents, Scalability needs.**

1. INTRODUCTION

With growing competition in the continuous integration (CI), the limitations in Jenkins come in the way of teams.CI is a critical service nowadays. People are running bigger workloads, needing more plugins, and high availability. Services like instant messaging platforms need to be online all the time. Jenkins is unable to stay up with this expectation and an outsized instance requires tons of overhead to stay it running. It is common for somebody to restart Jenkins a day which delays processes. Errors need to be contained to a specific area without impacting the whole service.

Continuous Integration also referred to as CI, is a crucial part of modern software development. In fact, it is a real game-changer when Continuous Integration is introduced into an organization since it radically alters the way teams think about the whole development process. It has excellent potential to reinforce the event process from continuous build integration to continuous deployment. CI is a development process in which developers commit their work frequently followed by build and testing before it's acceptance. Jenkins is an Open Source CI tool built in java which supports building and testing of projects virtually. The basic task of Jenkins is to execute some predefined steps known as jobs based on a trigger. In short, Jenkins monitors your version system for changes. Whenever a change is detected, it automatically compiles and builds the application. If something goes wrong, it immediately notifies the developers so that they can fix the issue immediately. Traditionally, Jenkins master and agents run on a dedicated server and are available only on a company intranet. In this setup, we have a fixed number of agents, and therefore it is not scalable. Jenkins talks to the GitLab server which is additionally hosted internally and available on an equivalent company intranet.

1.1. JENKINS

Jenkins may be a tool to use either as a server for continuous integration or an endless delivery hub that comes with many additional plugins to tweak continuous delivery workflow. Besides Jenkins, there're many proprietary continuous integration tools. If you'd wish to see these options also check our corresponding article where we compare the main CI tools in today's market. Git is a Version Control System with a repository for source code management that enables working online and offline. Servers. Another popular tool during this category is Ansible that automates configuration management, cloud provisioning, and application deployment.

Jenkins is a monolithic application based on a combination of a master and a slaves. The Jenkins master monitors sources, triggers jobs when predefined conditions are met, publishes logs, and artifacts. It doesn't run actual tasks but makes sure that they're executed. The Jenkins agent/slaves, on the opposite hand, do the particular work. When the master triggers a job execution, the actual work is performed by an agent. Everything fails sooner or later so we should be ready for failure recovery. The Jenkins master must run all the time so as to stay Jenkins up and running. Hence, the Jenkins master may be a single point of failure. Amazon web services are the biggest cloud service provider currently in the cloud market.

In AWS, you can deploy Jenkins on Amazon Elastic Compute Cloud (Amazon EC2). The Jenkins environment will be placed inside Amazon Virtual Private Cloud (Amazon VPC). It will also use Amazon Elastic Block Store (EBS) volume for storage purposes. This intercontinental development may have multiple teams working separately on different components or they may collaborate together to work on overlapped components.

1.2 ADVANTAGE OF GOING TO THE CLOUD

Advantage of going to the cloud

- On-demand Jenkins slaves
- Reliable managed environment
- No manual management of slaves
- Highly (if not infinitely) scalable
- AWS provides a variety of instance types, use instance types to define various slaves labels and Use the slave best suited to the job without over or under provisioning computing resources.

2. LITERATURE REVIEW

A few works have been discussed in history on CI and Jenkins. Nikitha Seth proposed as [1] Jenkins as master-slave architecture and the implementation of Jenkins for software patch integration and release to the client. Wettinger proposed as [2] they present a collaborative and holistic approach to capturing DevOps knowledge in a knowledgebase. Besides the ability to implement a crawling framework to automatically discover and capture DevOps knowledge. Charanjot Singh discussed [3] be comparing different continuous integration and delivery tools taking into consideration different parameters like performance monitoring post-deployment, pipeline integration, cloud compatibility, and server monitoring.

3. OUR APPROACH

We can provide service to two different server fleet to share build agents. The service is built by using the plugins by connect the Jenkins server and the build agents. When Jenkins server needs an agent to run a build on a remote agent it ask service to allocate an agent. The service will look at the available inventory and find suitable agent it allocate agent to Jenkins master Jenkins master uses agent for workload. The reason for distributed agents is for scale, or to provide different tools, or build on different target platforms like Windows, Mac and Linux. The result is orchestrating build agents needs of CI servers to allow sharing agents among same CI servers. By using this we perform Static agents sharing among different Jenkins server fleet. A Jenkins Slave Agent is required to be run at the slave node. Once the slaves are configured properly. We create Jobs to be executed on slaves. Slaves can be grouped as per the Operating System (OS) of slave and OS-specific jobs can be assigned to these groups easily.

We have created two group labels for the slave one for windows and other for Linux. Jenkins allows creating Maven, Freestyle or External job, etc. We created the Freestyle Project for our implementation. Jenkins gives options to add an SCM system like Git, Gerrit Repo, etc. The builds/Jobs can be parameterized based on requirements. It provides remote agents to servers for building projects. This is one of the important actions to be performed in the DevOps style of approach. It provides different platforms for projects to be built upon depending on the requirements. This solves one of the most common problems in the industry: “This code worked on my system, but not working here”, which is a huge breakthrough as the developers can develop the code for different kinds of platforms for much less effort than it usually is. This reduces a lot of time/latency in delivery. In this implementation, we have used a parameterized build step. Jenkins allows adding shell scripts/windows batch commands. We added our build steps into this script.

4. WORKING

Initially, Jenkins server to be installed in the master system. In Jenkins, we will initialize the service as a plugin and this service will be installed in the Jenkins server in the master system. The service is configured with many agents or nodes which are in the cloud platforms like AWS, Digital Ocean. The Jenkins will have many child servers like jenkins1, jenkins2, etc. The developer wants to build the project and asks the service to give the required agent. Then the service will look for any available agents in the server fleet. If an agent is available it will assign to the developer else it will look for available local agents. It may be the Windows, Linux, mac agent the service will provide it. Once the build is done the developer will release the agent to service and the service will place in its agent pool. The process will continue like this.

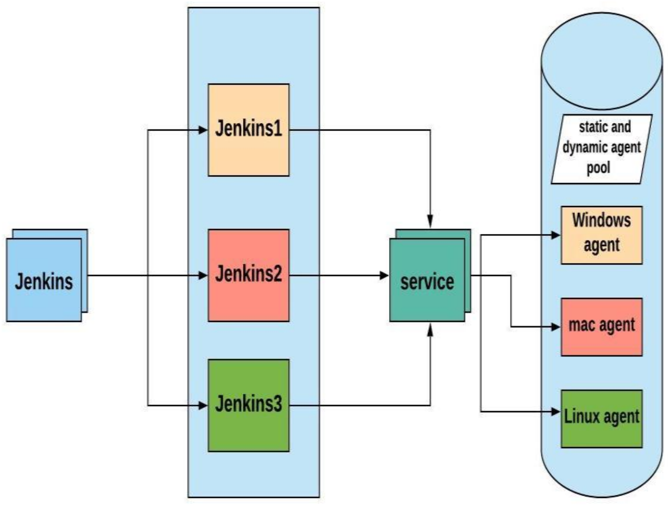


Fig 4.1. System architecture of Jenkins and the service to agent pool.

The above figure is the block diagram of the Jenkins with the service provided through the different build agents. The master server will be having many servers in it. The different servers will access the different build agents in the same time. So, the build process will be easy and the more stable.

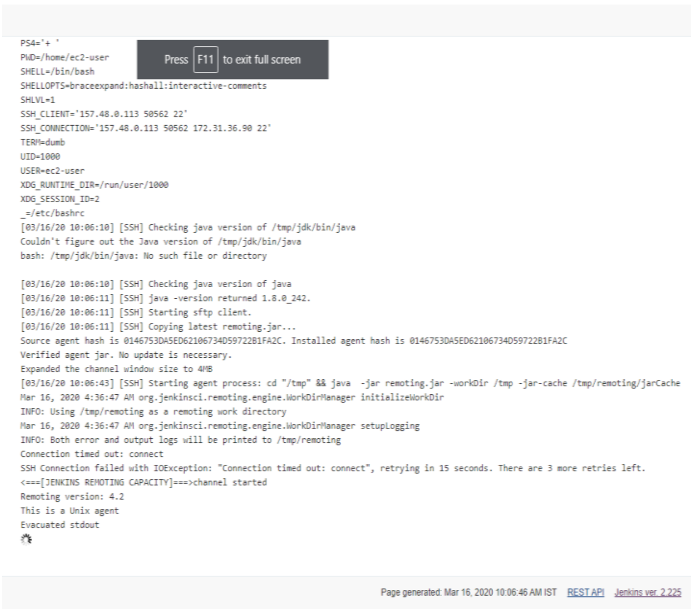
4.1 METHODOLOGY

Input: remote agent credentials (username, password, dns address)
Output: build output in remote agent.

```
Step 1 - Initialize the Jenkins server fleet
Step 2 - Activate Jeeves in Jenkins
Step 3 - Initialize the build
Step 4 - If remote agent credentials valid goto step 5
            Else goto step 10
Step 5 - If DNS address exists goto step 6
            Else goto step 10
Step 6 - if status == online goto step 7
            Else goto step 8
Step 7 - Run our build on the agent goto step 11
Step 8 - If any other agents meets the
            Requirements goto step 6
            Else goto step 9
Step 9 - Wait until (status == online) then goto step 7
Step 10 - Print “credentials invalid”
Step 11 - End
```

5. RESULTS

The below pictures are the build results obtained when the project is build in executed from git source code management.

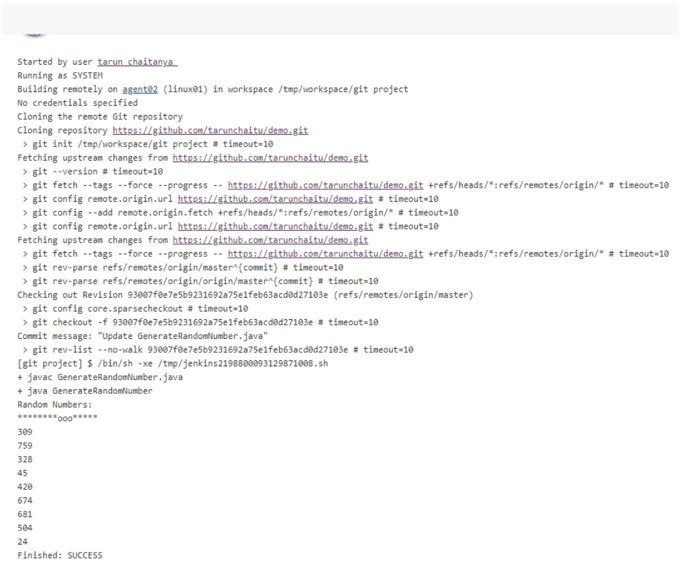


```
PS4='+ '
PWD=/home/ec2-user
SHELL=/bin/bash
SHELLOPTS=braceexpand:hashall:interactive-comments
SHLVL=1
SSH_CLIENT='157.48.0.113 50562 22'
SSH_CONNECTION='157.48.0.113 50562 172.31.36.90 22'
TERM=dumb
UID=1000
USER=ec2-user
XDG_RUNTIME_DIR=/run/user/1000
XDG_SESSION_ID=2
_=/etc/bashrc
[03/16/20 10:06:10] [SSH] Checking java version of /tmp/jdk/bin/java
Couldn't figure out the Java version of /tmp/jdk/bin/java
bash: /tmp/jdk/bin/java: No such file or directory

[03/16/20 10:06:10] [SSH] Checking java version of java
[03/16/20 10:06:11] [SSH] java -version returned 1.8.0_242.
[03/16/20 10:06:11] [SSH] Starting sftp client.
[03/16/20 10:06:11] [SSH] Copying latest remoting.jar...
Source agent hash is 0146753DA5ED62106734D59722B1FA2C. Installed agent hash is 0146753DA5ED62106734D59722B1FA2C
Verified agent jar. No update is necessary.
Expanded the channel window size to 4MB
[03/16/20 10:06:43] [SSH] Starting agent process: cd "/tmp" && java  -jar remoting.jar -workDir /tmp -jar-cache /tmp/remoting/jarCache
Mar 16, 2020 4:36:47 AM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFO: Using /tmp/remoting as a remoting work directory
Mar 16, 2020 4:36:47 AM org.jenkinsci.remoting.engine.WorkDirManager setupLogging
INFO: Both error and output logs will be printed to /tmp/remoting
Connection timed out: connect
SSH Connection failed with IOException: "Connection timed out: connect", retrying in 15 seconds. There are 3 more retries left.
<===[JENKINS REMOTING CAPACITY]===>channel started
Remoting version: 4.2
This is a Unix agent
Evacuated stdout
```

Press F11 to exit full screen
Page generated: Mar 16, 2020 10:06:46 AM IST REST API Jenkins ver. 2.225

Fig 5.1 Connecting to build agent.



```
Started by user tarun chaitanya
Running as SYSTEM
Building remotely on agent02 (linux01) in workspace /tmp/workspace/git project
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/tarunchaitu/demo.git
 > git init /tmp/workspace/git project # timeout=10
Fetching upstream changes from https://github.com/tarunchaitu/demo.git
 > git --version # timeout=10
 > git fetch --tags --force --progress -- https://github.com/tarunchaitu/demo.git +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git config remote.origin.url https://github.com/tarunchaitu/demo.git # timeout=10
 > git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git config remote.origin.url https://github.com/tarunchaitu/demo.git # timeout=10
Fetching upstream changes from https://github.com/tarunchaitu/demo.git
 > git fetch --tags --force --progress -- https://github.com/tarunchaitu/demo.git +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
 > git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 93007f0e7e5b9231692a75e1feb63acd0d27103e (refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 93007f0e7e5b9231692a75e1feb63acd0d27103e # timeout=10
Commit message: "Update GenerateRandomNumber.java"
 > git rev-list --no-walk 93007f0e7e5b9231692a75e1feb63acd0d27103e # timeout=10
[git project] $ /bin/sh -xe /tmp/jenkins2198800093129871008.sh
+ javac GenerateRandomNumber.java
+ java GenerateRandomNumber
Random Numbers:
********ooo*****
309
759
328
45
420
674
681
504
24
Finished: SUCCESS
```

Fig 5.2 Build output

6. CONCLUSION

The main aim of this project is to provide the remote build agents for projects o run on multiple environments. We run our algorithm on the distributed environment so that time consumption is less and also our algorithm is easy to implement. By applying our algorithm we will be able to understand the process of continuous integration in DevOps.

We have identified that there are no open-source for this type of service. Ability to grasp advanced programming techniques to solve contemporary issues. We formulated the The threshold called the support upon analysis of the data. We designed the steps to be of two parts. So, it is essential to hosting Jenkins on a reliable platform. Running and managing it on your own can become a very hectic process, especially when you start scaling and you have several builds to take care of. The cloud-based integration in Jenkins will play a crucial role in future DevOps. DevOps leverage the benefits of cloud computing to possess a higher degree of automation and good quality delivery of software.

REFERENCES

[1] Nikita Seth, Rishi Khare "ACI (Automated Continuous Integration) using Jenkins: Key for Successful Embedded Software Development",978-1-4673-8253-3/15/$31.00 c 2015 IEEE.

[2] Johannes Wettinger, Vasilios Andrikopoulos, Frank Leymann "Automated Capturing and Systematic Usage of DevOps Knowledge for Cloud Applications", 978-1-4799-8218-9/15 $31.00 © 2015 IEEE, DOI 10.1109/IC2E.2015.23.

[3] Charanjot Singh, Nikita Seth Gaba, Manjot Kaur, Bhavleen Kaur 'Comparison of Different CI/CD Tools Integrated with Cloud Platform' 978-1-5386-5933-5/19/$31.00c 2019 IEEE

[4] Online Resource: https://jenkins-ci.org/.

[5] Git, https://git-scm.com/

[6] Online Resource:https://wiki.jenkins-ci.org/display/JENKINS/Home/.

[7] S.A.I.B.S. Arachchi, Indika Perera"Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management",978-1-5386-4417-1/18/$31.00 ©2018 IEEE

[8] Amazon. Amazon EC2 Instance Types - Amazon WebServices (AWS). URL: https : / / aws . amazon. com /ec2/instance-types/.