

Jenkins ist ein eigenständiger Open-Source-Automatisierungsserver, der zur Automatisierung aller Arten von Aufgaben im Zusammenhang mit dem Erstellen, Testen und Bereitstellen oder Bereitstellen von Software verwendet werden kann. Es hilft, den kontinuierlichen Integrationsprozess zu erreichen.

Jenkins ist kostenlos und vollständig in Java geschrieben. Es ist eine häufig verwendete weltweite Anwendung. Jenkins kann über native Systempakete oder Docker installiert oder sogar eigenständig auf jedem Computer ausgeführt werden, auf dem eine Java Runtime Environment (JRE) installiert ist.In unserem Fall haben wir OpenJDK-11 installiert.

Kontinuierliche Integration ist eine Praxis, bei der Entwickler mehrmals am Tag Code in ein gemeinsam genutztes Repository integrieren müssen und Jeder Check-in führt zu einem automatisierten Build.

Kontinuierliche Delivery ist die Fähigkeit, automatisierte Bereitstellungen durchzuführen

Es ist ein serverbasiertes System, das auf einem Webserver wie Apache Tomcat läuft. Mit Jenkins können Unternehmen den Softwareentwicklungsprozess durch Automatisierung beschleunigen. Jenkins kombiniert alle Arten von Lebenszyklus-Entwicklungsprozessen, einschließlich Erstellen, Dokumentieren, Testen, Verpacken, Bereitstellen, Bereitstellen, statischer Analyse und vielem mehr.

Kontinuierliche Integration

Kontinuierliche Integration (CI) ist die Praxis, die mehrmalige tägliche Integration von Codeänderungen aller Entwickler in die gemeinsame Hauptlinie zu automatisieren. Die Software wird nach einem Code-Commit sofort durch Kontinuierliche Integration gebaut und getestet. Wenn der Unit-Test bestanden wird, wird der Build für die Bereitstellung getestet. Der Code wird in die Produktion verschoben, wenn die Bereitstellung erfolgreich ist. All dies ist ein kontinuierlicher Prozess und daher der Name Kontinuierliche Integration.

Vor Jenkins haben alle Entwickler, sobald die ihnen zugewiesenen Programmieraufgaben abgeschlossen waren, ihren Code gleichzeitig übergeben. Der Build wird später getestet und bereitgestellt. Da der Code auf einmal erstellt wurde, mussten einige Entwickler warten, bis andere Entwickler mit dem Codieren fertig waren, um ihren Build zu testen. Das Isolieren, Erkennen und Beheben von Fehlern bei mehreren Commits ist nicht einfach. Wenn alle Fehler behoben und getestet wurden, wird der Code bereitgestellt. Der Entwicklungszyklus ist also träge.

Mit Jenkins wird die Software nach einem Code-Commit sofort erstellt und getestet. Da der Code erstellt wird, nachdem jeder einzelne Entwickler festgeschrieben hat, ist es einfach zu erkennen, wessen Code das Fehlschlagen des Builds verursacht hat. Nach jedem erfolgreichen Build und Test wird der Code bereitgestellt. Es beschleunigt also den Softwareentwicklungsprozess.

Durch die Verwendung von Plugins erreicht Jenkins Kontinuierliche Integration. Plugins ermöglichen die Integration verschiedener DevOps-Stufen. Wenn Sie eine bestimmte Anwendung integrieren möchten, müssen die Plugins für diese Anwendung installiert werden.

Jenkins-Terminologie

|  |  |
| --- | --- |
| Name | Description |
| Agent | Ein Agent ist normalerweise eine Maschine oder ein Container, der sich mit einem Jenkins-Master verbindet und Aufgaben ausführt, wenn er vom Master angewiesen wird. |
| Build | Ergebnis einer einzelnen Ausführung eines Projekts |
| Core | Die primäre Jenkins-Anwendung (jenkins.war), die die grundlegende Webbenutzeroberfläche, Konfiguration und Grundlage bereitstellt, auf der Plugins erstellt werden können. |
| Job/Project | Jenkins scheint diese Begriffe synonym zu verwenden. Sie beziehen sich alle auf ausführbare Tasks, die von Jenkins gesteuert/überwacht werden. |
| Master | Der zentrale, koordinierende Prozess, der die Konfiguration speichert, Plugins lädt und die verschiedenen Benutzeroberflächen für Jenkins rendert. |
| Node | Eine Maschine, die Teil der Jenkins-Umgebung ist und Pipelines oder Projekte ausführen kann. Sowohl der Master als auch die Agenten werden als Knoten betrachtet. |
| Pipeline | Eine Pipeline automatisiert den Prozess der Softwarebereitstellung. Es erstellt Code, führt Tests durch und hilft Ihnen, eine neue Version der Software sicher bereitzustellen. |
| Plugin | Eine Erweiterung der Jenkins-Funktionalität, die separat von Jenkins Core bereitgestellt wird. |
| Publisher | Teil eines Builds nach Abschluss aller konfigurierten Schritte, der Berichte veröffentlicht, Benachrichtigungen sendet usw. |
| Step | Eine einzige Aufgabe; Grundsätzlich sagen Schritte Jenkins, was innerhalb einer Pipeline oder eines Projekts zu tun ist. |

Jenkins installieren

Voraussetzungen

Für die Installation benötigen wir:

Eine Maschine mit 256 MB RAM, obwohl mehr als 512 MB empfohlen werden, und 10 GB Speicherplatz (für Jenkins und unser Docker-Image)

* Java 8 oder 11 (entweder ein JRE oder ein Java Development Kit (JDK) ist in Ordnung)
* Docker

1. Wir laden Jenkins herunter und führen wir es aus

For Linux:

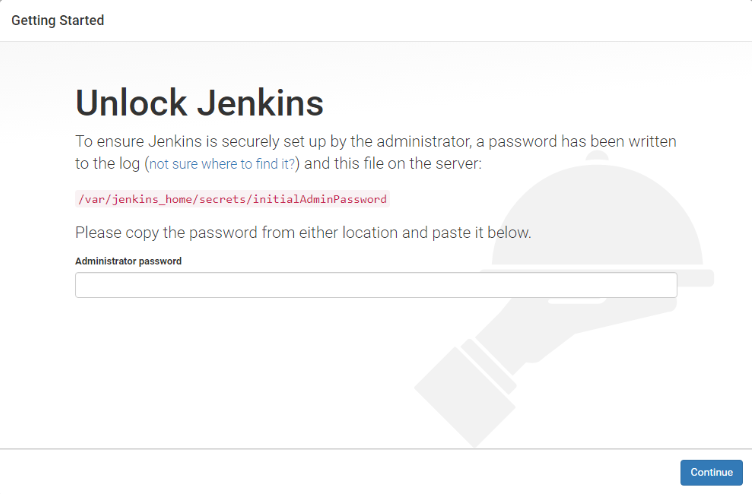
*wget* [*http://mirrors.jenkins.io/war-stable/latest/jenkins.war*](http://mirrors.jenkins.io/war-stable/latest/jenkins.war)

1. Wir öffnen ein Terminal im Download-Verzeichnis.
2. Wir führen diesen Befehl aus.

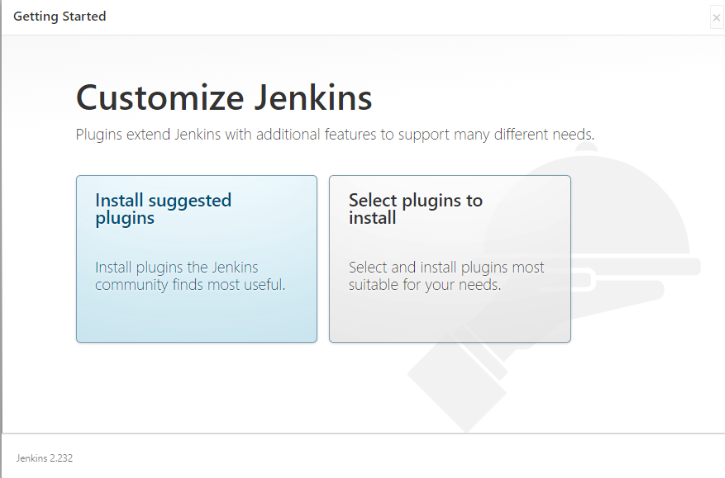
*java -jar jenkins.war --httpPort=8080*

Dieser Befehl lädt ein Jenkins herunter und führt es auf unserem Computer aus. Viele Meldungen werden auf dem Bildschirm angezeigt. Nach der Initialisierung von Jenkins gibt uns die Konsole ein Administratorkennwort, um die Ersteinrichtung für Jenkins zu starten.

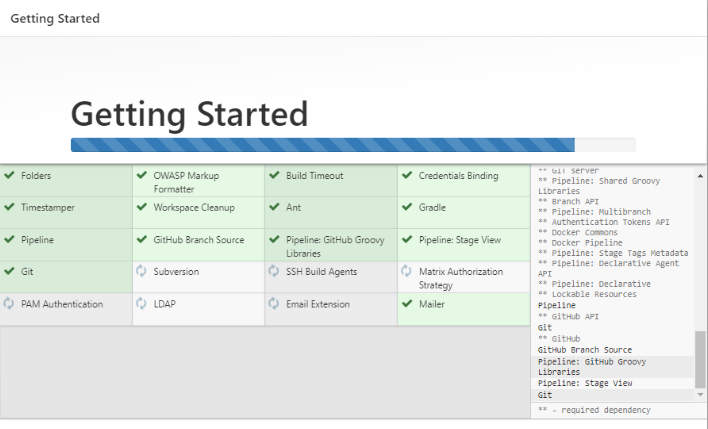
1. Wir navigieren zu http://localhost:8080. Im Cloud http://(Ihre ec2-Instance IPv4 Public IP):8080). Wir sehen den Bildschirm unten. Wir geben das Admin-Passwort auf dem Bildschirm ein und drücken wir auf Weiter.



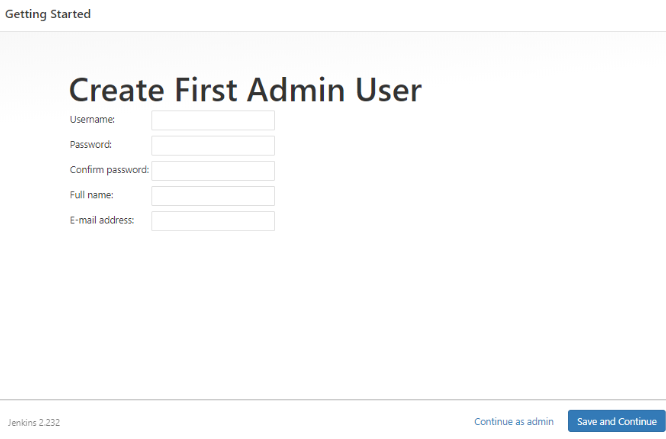
Wir klicken im nächsten Bildschirm auf „Vorgeschlagene Plugins installieren“.



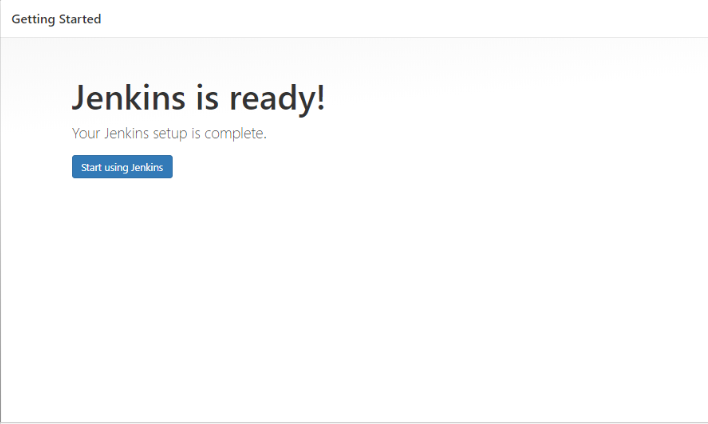
Wir warten auf die Installationen.



Wir erstellen einen Benutzer.



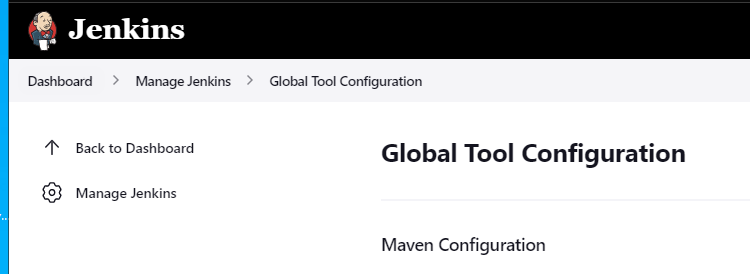
Wir schließen die Installation ab, klicken auf „Jenkins verwenden“

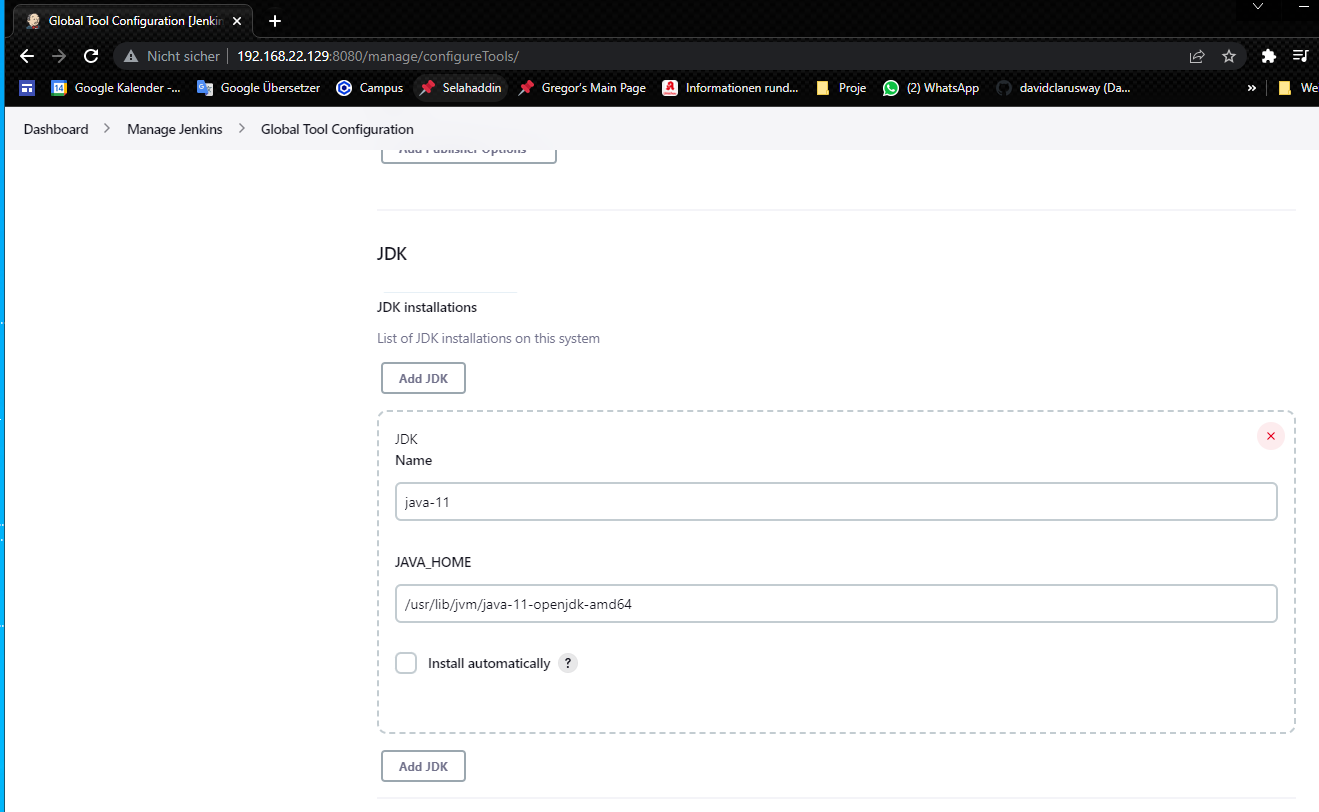


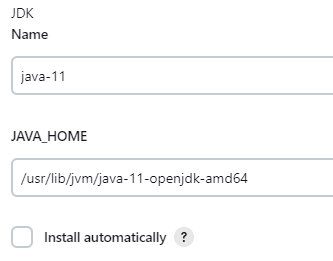
Wenn die Installation abgeschlossen ist, können wir damit beginnen, Jenkins zum Laufen zu bringen!

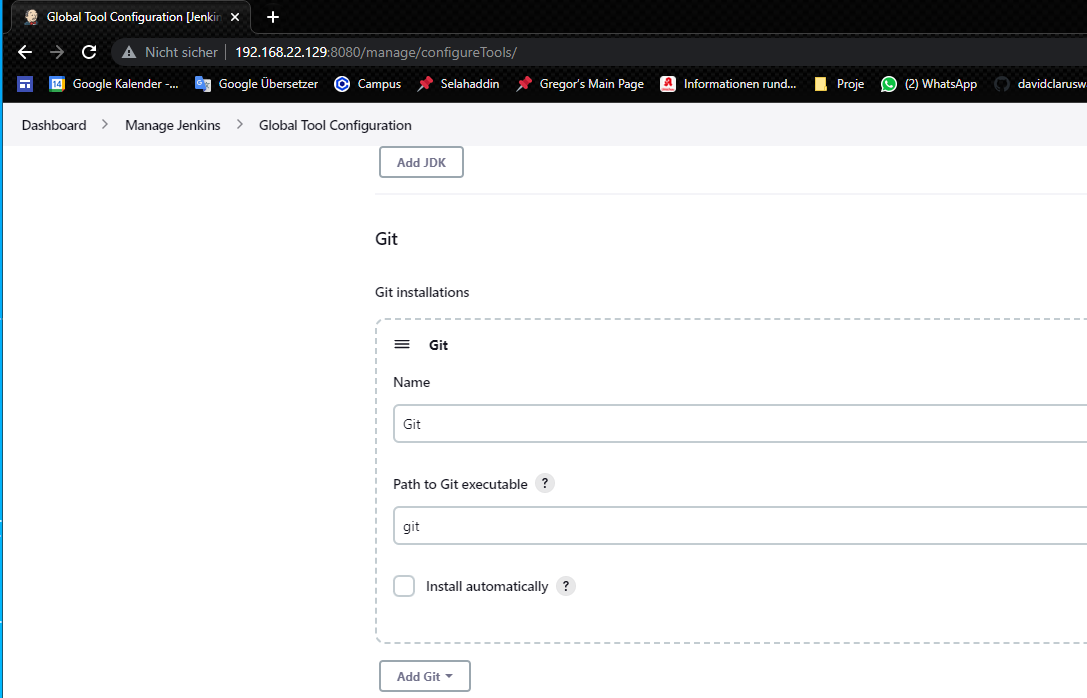
Jenkins Global Tool Configurations

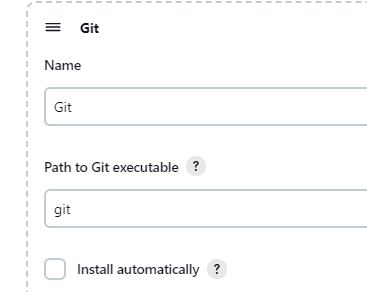
Das Home-Verzeichnis der Programme, die automatisch vom Jenkins-Server zu verwenden , muss angegeben werden.

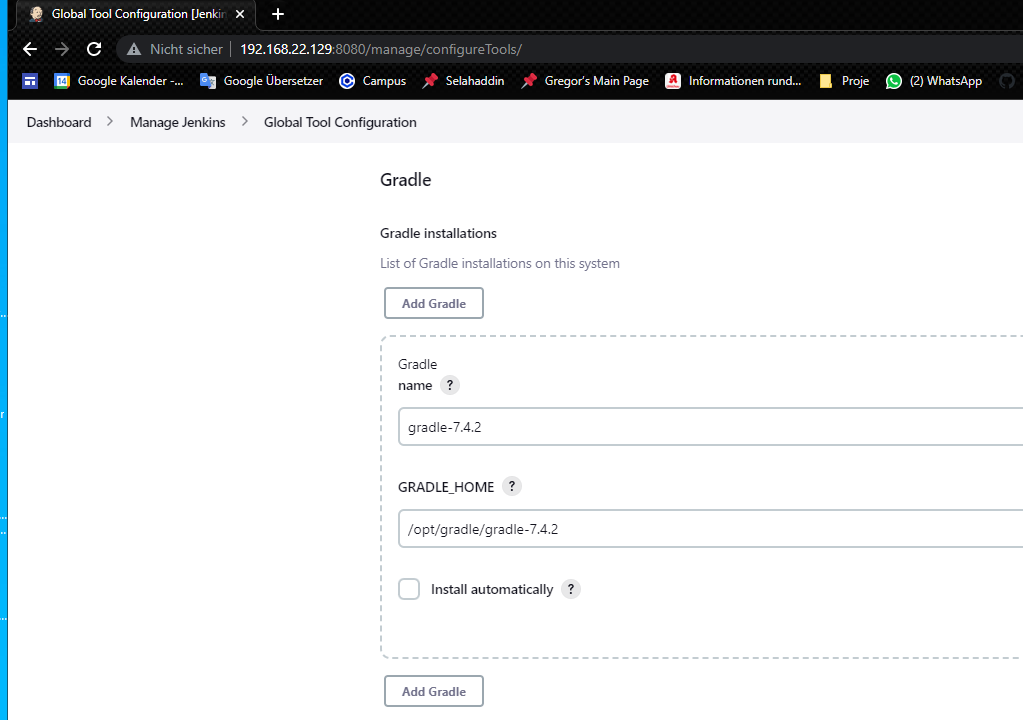


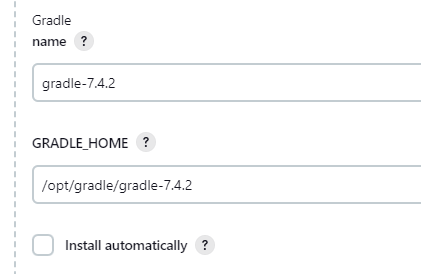


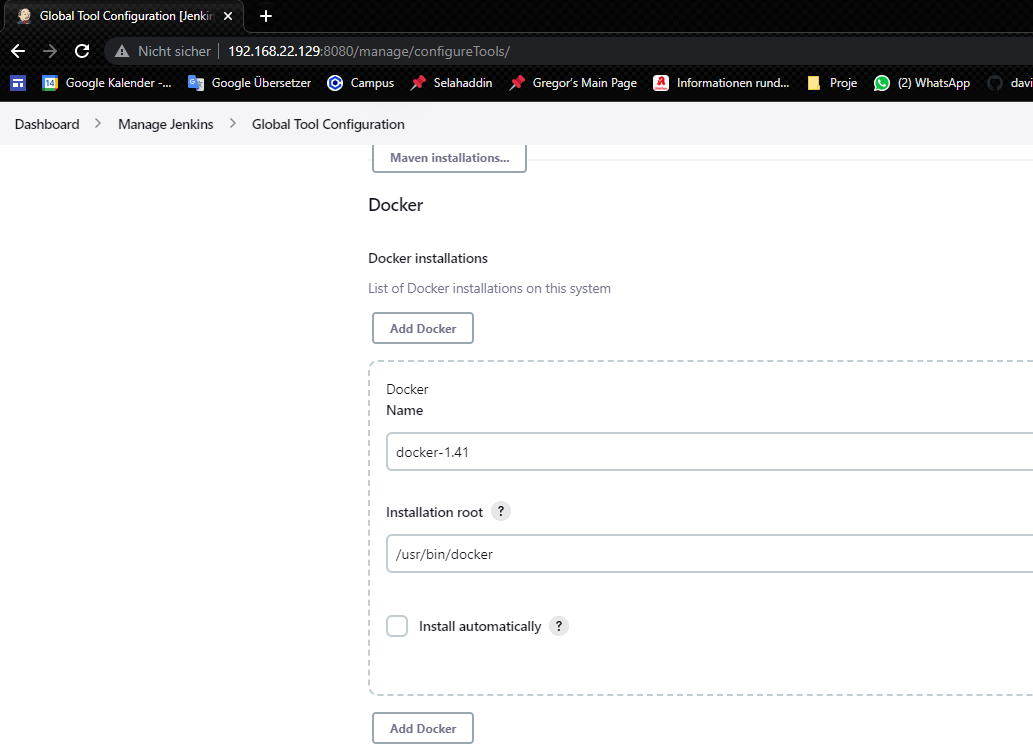


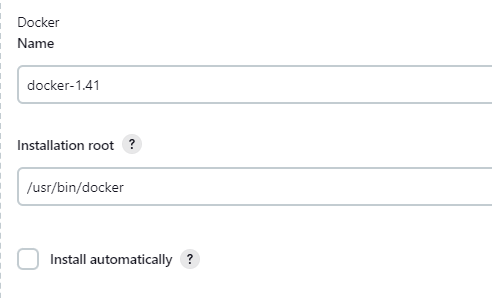












### Installieren Jenkins mit Docker(Optional)

Zuerst erstellen wir einen Container in unserer Maschine. Docker muss auf unseren Maschinen installiert sein und ausgeführt werden. Wir führen diesen Befehl aus:

*docker run -p 8080:8080 -p 50000:50000 -v ~/$(USER):/var/jenkins\_home jenkins*

Hier nutzen wir also ein Volume. Dadurch wird der Arbeitsbereich in /var/jenkins\_home gespeichert. Darin befinden sich alle Jenkins-Daten – einschließlich Plugins und Konfiguration. Wir werden dies wahrscheinlich zu einem dauerhaften Volume machen wollen (empfohlen).

Quelle: <https://hub.docker.com/_/jenkins>

<https://www.jenkins.io/doc/book/installing/docker/>

Zusammenfassend: Dieser Befehl lädt ein Jenkins herunter und führt es auf unserem Computer aus. Viele Meldungen werden auf dem Bildschirm angezeigt. Nach der Initialisierung von Jenkins gibt uns die Konsole ein Administratorkennwort, um die Ersteinrichtung für Jenkins zu starten.

Wir können auf unsere Jenkins unter http://localhost:8080/ zugreifen (für die Cloud-Instanz IPv4 Public IP:8080). Wir sehen den Bildschirm unten. Wir geben das Admin-Passwort auf dem Bildschirm ein und drücken wir auf Weiter.

So greifen wir auf das Admin-Passwort zu: Wenn wir Jenkins innerhalb von Docker als getrennten Container ausführen, können wir die Befehle docker logs < containerId > verwenden, um die Jenkins-Protokolle anzuzeigen.Oder führen Sie die folgenden Befehle aus:

*docker exec -it < containerId > bash*

*cat /var/jenkins\_home/secrets/initialAdminPassword*

Hinzufügen von Agenten/Slave-knoten zum Jenkins-Server

Um Prozesse im Rahmen der CI/CD Pipeline auf anderen Nodes mit Jenkins Server auszuführen, sollten diese Nodes natürlich als Agenten gegenüber Jenkins Server definiert werden.

Dazu müssen wir eine SSH\_Verbindung zwischen Jenkins\_Server and Agent Node setzen und die „agent“ file vom Jenkins-Server auf den Slave-Knoten kopieren.

a. Schritte zum Konfigurieren der SSH-Verbindung zwischen Master- und Slave-Knoten

- Wir gehen zum Jenkins Master-Server.

- Wir wechseln zum Benutzer "jenkins". ( jenkins funtoniert auf Agentkonoten als jenkins-user. )

„ sudo su - jenkins -s /bin/bash „

- Wir generieren mit keygen einen öffentlichen und einen privaten Schlüssel.

„ ssh-keygen „

- Wir drücken bei jeder Frage die Eingabetaste, um mit den Standardoptionen fortzufahren

- Wir überprüfen den Ordner „.ssh“ und sehen Sie öffentliche (id\_rsa.pub) und private Schlüssel (id\_rsa)

„cd .ssh ; ls „

-Wir müssen den öffentlichen Schlüssel auf den Slave-Knoten kopieren.

„cat id\_rsa.pub „

-Wir wählen alle Codes in id\_rsa.pub aus und kopieren wir sie. Wir wechseln zum Ordner /root/.ssh auf der Slave-Knoteninstanz.

„sudo su

cd /root/.ssh „

- Wir öffnen die Datei „authorized\_keys“ mit einem Editor und fügen wir den Code ein, den wir aus dem öffentlichen Schlüssel (id\_rsa.pub) kopiert haben. Wir speichern die Datei „authorized\_keys“. Wir holen sich die Slave-Knoten-IP.

„ ifconfig „

- IP-Nummer kopieren und wir gehen zum Jenkins-Master-Server und testen wir die SSH-Verbindung.

„ ssh root@<slave-node-ip-number> „

b. die Agentendatei vom Jenkins-Server auf den Slave-Knoten zu kopieren

- Wir wechseln zum Ordner „/root“ auf der Slave-Knoteninstanz. Wir erstellen einen Ordner unter „/root“ und nennen Sie ihn „bin“. Agent-Datei vom Jenkins-Master-Server abrufen.

„ mkdir bin

cd bin

wget http://<jenkins\_master\_ip>:8080/jnlpJars/slave.jar „

- Wir gehen zum Jenkins-Dashboard und klicken wir im linken Menü auf „Manage Jenkins“.

- Wir wählen „Manage Nodes and Clouds“

- Wir klicken im linken Menü auf „New Node“.

- Wir geben „SlaveNode-1“ in das Feld „Node name“ ein und wählen wir „Permanent Agent“ aus.

- Klicken auf die Schaltfläche "OK".

- Wir geben „Dies ist ein Linux-Slave-Knoten für Jenkins“ in das Beschreibungsfeld ein.

- "Number of Executors" ist die maximale Anzahl gleichzeitiger Builds, die Jenkins auf diesem Knoten ausführen kann. Wir geben in dieses Feld „1“ ein. Eigentlich ist die geeignete Anzahl zu diesem Bereich ist die Anzahl von CPU-Kerne.

- Ein Agent muss ein Verzeichnis haben, das Jenkins gewidmet ist. Wir geben den Pfad zu diesem Verzeichnis auf dem Agenten an. Wir geben `/usr/jenkins` in das Feld "Remote root directory" ein.

- Geben "Linux" in das Feld "Labels" ein.

- Wählen im Dropdown-Menü im Feld „Launch method“ die Option „Launch agent via execution of command on the master“ aus.

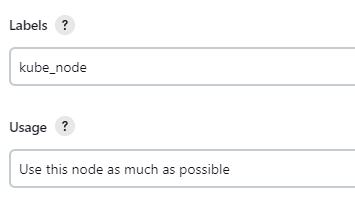
- Enter `ssh -i /var/lib/jenkins/.ssh/ id\_rsa root@<slave\_ip> java -jar /root/bin/slave.jar` in das Feld " Launch command " ein.

- Wählen im Dropdown-Menü im Feld "Availability" die Option " Keep this agent online as much as possible ".

- Klicken auf „Save“.

- Überprüfen die Konsolenprotokolle, falls der Agent-Knoten nicht gestartet werden kann. Wenn es ein Genehmigungsproblem gibt, gehen zu „Manage Jenkins“, wählen „`In-process Script Approval“ und „approve“ Sie das Skript.

- Gehen zum Jenkins-Dashboard. Überprüfen die Master- und Slave-Knoten im linken Menü.



Aufbau eines Jobs

Ein Jenkins-Projekt ist ein wiederholbarer Build-Job, der Schritte und Post-Build-Aktionen enthält. Die Arten von Aktionen, die wir in einem Build-Schritt oder einer Post-Build-Aktion ausführen können, sind ziemlich begrenzt. Innerhalb eines Jenkins-Freestyle-Projekts sind viele Standard-Plugins verfügbar, die Ihnen helfen, dieses Problem zu lösen. Wir ermöglichen Ihnen die Konfiguration von Build-Triggern und bieten projektbasierte Sicherheit für Ihr Jenkins-Projekt.

Ein neues Jenkins-Element kann eine von sechs Auftragsarten sowie ein Ordner zum Organisieren von Elementen sein:

Freestyle-Projekt

Pipeline

externer Arbeitsplatz

Projekt mit mehreren Konfigurationen

Ordner

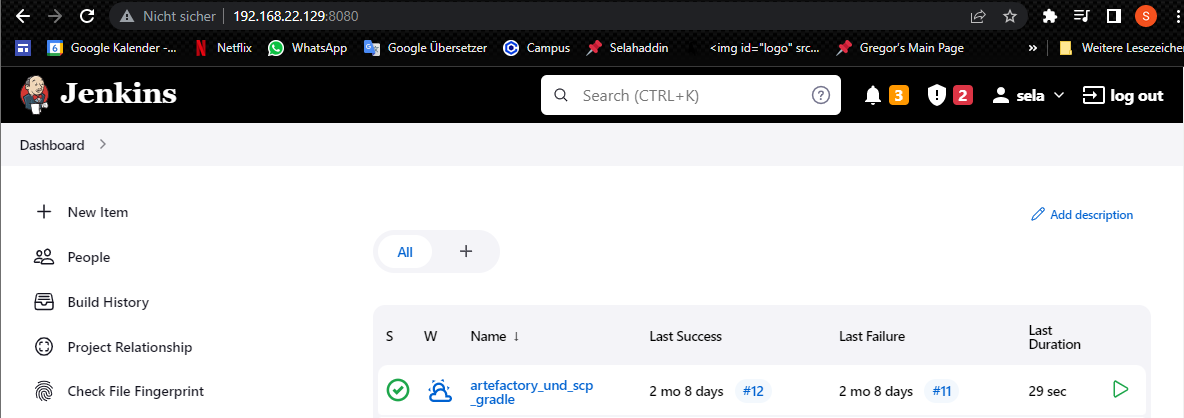
Github-Organisation

Multibranch-Pipeline

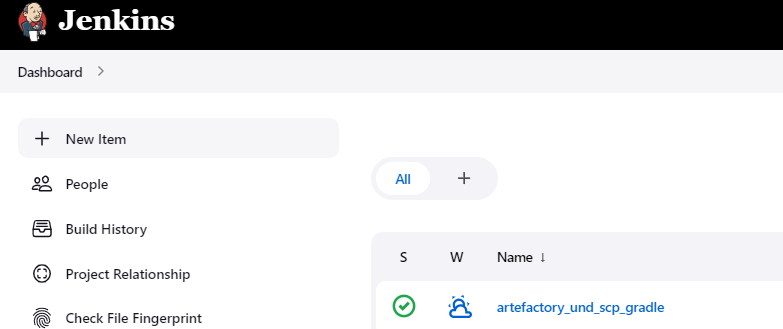
In Unserem Fall werden häufig Freestyle-Projekt oder Pipeline

Der Freestyle-Build-Job ist eine sehr flexible und einfach zu verwendende Option. Wir können es für jede Art von Projekt verwenden; Es ist einfach einzurichten und viele seiner Optionen erscheinen in anderen Build-Jobs.

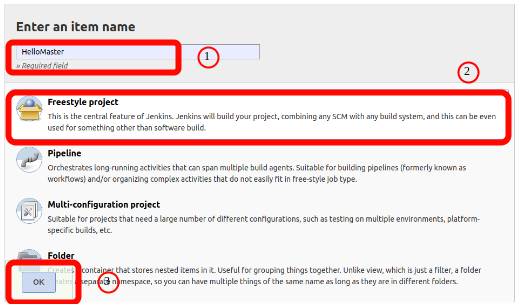
1 - Um einen Jenkins-Freestyle-Job zu erstellen, öffnen wir Ihren Jenkins-Installationspfad. Es wird auf http://localhost:8080 oder http://cloud-ip-address:8080 gehostet. Wenn wir auf einem anderen Port installiert haben, sollen wir diese Portnummer verwenden. Dann melden wir uns mit Benutzername und Passwort bei unserem Jenkins-Dashboard an. In Unserem Fall die local private Netzwerkaadrese von unseres Jenkins\_server ist: 192.168.22.129



2. Lass uns öffnen wir unser Jenkins-Dashboard und klicken wir auf New Item, um ein neues Element zu erstellen.



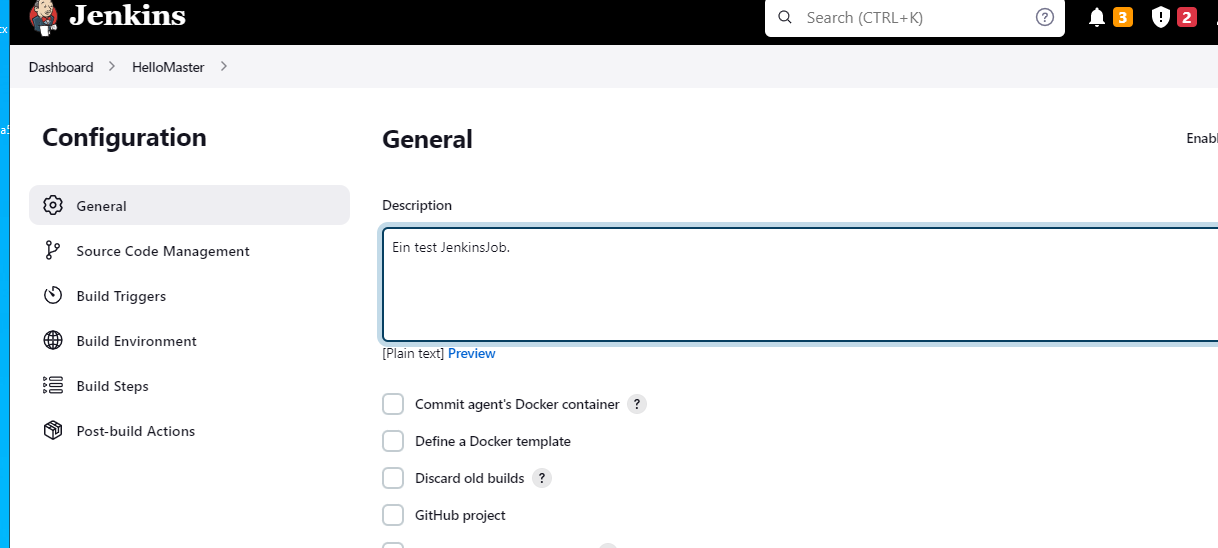
3. Wir geben den Itemnamen ein, wählen wir dann ein Freestyle-Projekt und klicken wir auf OK.



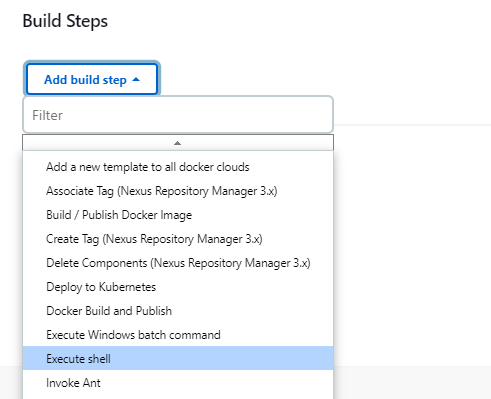
Beim Erstellen des ersten Projekts sehen wir sechs Registerkarten. Diese sind:

1. Source Code Management Tab : optionales SCM (Source Code Management), wie CVS (Concurrent Versions System) oder Subversion, wo sich Ihr Quellcode befindet.
2. Build Triggers : Optionale Trigger zur Steuerung, wann Jenkins Builds ausführt.
3. Build Environment: Eine Art Build-Skript, das den Build durchführt (Ameise, Maven, Shell-Skript, Batch-Datei usw.), wo die eigentliche Arbeit stattfindet
4. Build : Optionale Schritte zum Sammeln von Informationen aus dem Build, z. B. Archivieren der Artefakte und/oder Aufzeichnen von Javadoc- und Testergebnissen.
5. Post-build Actions : Optionale Schritte, um andere Personen/Systeme mit dem Build-Ergebnis zu benachrichtigen, z. B. Senden von E-Mails, IMs, Aktualisieren des Issue-Trackers usw
6. General :

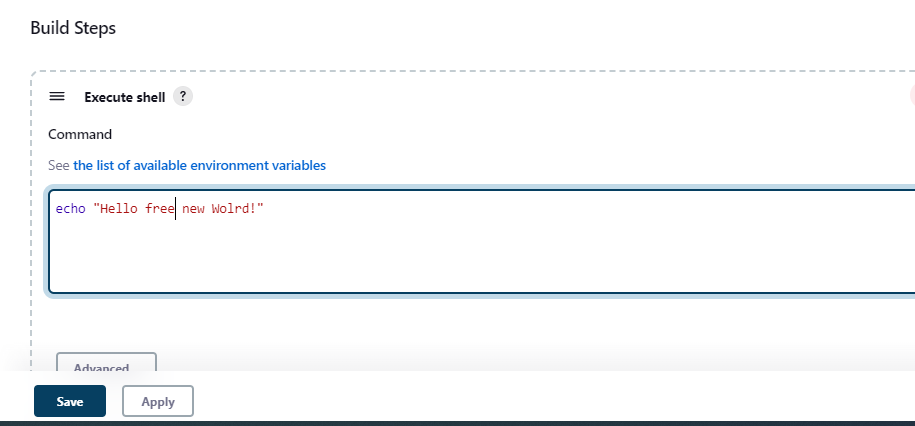
4.Wir geben die Beschreibung Ihres Jenkins-Projekts an. Wir gehen dann zu Bauen.



1. Wir wählen den entsprechenden Build-Schritt aus "Build-Schritt hinzufügen". Es hängt von Ihrem Betriebssystem ab. Wenn wir ein Windows-Benutzer sind, können wir Windows-Stapelbefehl ausführen auswählen, oder wenn wir ein Linux- oder MacOS-Betriebssystem verwenden, können wir Shell ausführen auswählen.



1. Wenn wir den Erstellungsschritt auswählen, wird der Befehlsraum angezeigt. Notieren wir einfach echo "Hello World", um den Shell-Befehl/Windows-Batch-Befehl auszuführen. Dann klicken wir dann auf Anwenden und Speichern. Danach sehen wir das Innere des Projekts. Klicken wir dann auf Jetzt erstellen und warten wir einige Sekunden. Das Ergebnis dieser Build-Aktion wird im Build-Verlauf angezeigt



1. Wie oben erwähnt, sehen wir die Ergebnisse des Builds. Wenn wir auf die Build-Nummer klicken, können wir die Build-Seite erreichen. Wenn wir die blaue Zahl sehen, hat der Aufbau gut funktioniert. Auf der Build-Seite können wir die Konsolenergebnisse von "Console Output" sehen. Wir können die Ergebnisse Ihres Builds sehen. Wie wir unser Ergebnis sehen können, wurde unser Job mit "Hello World" ausgedruckt und mit "SUCCESS" beendet.



## Pipeline (Pipeline-Projekt auf Slave-Knoten)

Jenkins Pipeline (oder einfach „Pipeline“) ist eine Suite von Plugins, die die Implementierung und Integration von Continuous-Delivery-Pipelines in Jenkins unterstützt.

Eine Continuous-Delivery-Pipeline ist ein automatisierter Ausdruck Ihres Prozesses, um Software von der Versionskontrolle bis zu Ihren Benutzern und Kunden zu bringen. Jenkins Pipeline bietet einen erweiterbaren Satz von Tools zum Modellieren einfacher bis komplexer Bereitstellungspipelines „als Code“. Die Definition einer Jenkins-Pipeline wird normalerweise in eine Textdatei (als Jenkinsfile bezeichnet) geschrieben, die wiederum in das Quellcodeverwaltungs-Repository eines Projekts eingecheckt wird.

Jenkins-Pipelines können mithilfe einer Textdatei namens JenkinsFile definiert werden. Wir können die Pipeline mithilfe von JenkinsFile als Code implementieren, und dies kann mithilfe einer domänenspezifischen Sprache (DSL) definiert werden. Mit JenkinsFile können wir die Schritte schreiben, die zum Ausführen einer Jenkins-Pipeline erforderlich sind.

Es gibt zwei Arten von Syntax, die zum Definieren Ihrer JenkinsFile verwendet werden.

1. Deklarativ: Die deklarative Pipeline-Syntax bietet eine einfache Möglichkeit, Pipelines zu erstellen. Es enthält eine vordefinierte Hierarchie zum Erstellen von Jenkins-Pipelines.

Pipeline {

/\*Hier geben wir Deklarative Formulierungen ein\*/

}

2. Geskriptet: Geskriptete Jenkins-Pipeline läuft auf dem Jenkins-Master mit Hilfe eines leichtgewichtigen Executors. Es verwendet sehr wenige Ressourcen, um die Pipeline in atomare Befehle zu übersetzen.

Node {

stage ( `Build´) {

// some Codes

}

stage ( `Test´) {

// some Codes

}

}

Pipeline-Konzepte

Code: Pipelines werden im Code implementiert und in der Regel in die Quellcodeverwaltung eingecheckt, sodass Teams ihre Bereitstellungspipeline bearbeiten, überprüfen und iterieren können.

Langlebig: Pipelines können sowohl geplante als auch ungeplante Neustarts des Jenkins-Masters überstehen.

Pausierbar: Pipelines können optional anhalten und auf menschliche Eingaben oder Genehmigungen warten, bevor sie die Pipeline-Ausführung fortsetzen.

Vielseitig: Pipelines unterstützen komplexe reale CD-Anforderungen, einschließlich der Fähigkeit, Arbeiten parallel zu forken/beizutreten, zu loopen und auszuführen.

Erweiterbar: Das Pipeline-Plugin unterstützt benutzerdefinierte Erweiterungen seiner DSL (domänenspezifische Sprache) und mehrere Optionen für die Integration mit anderen Plugins.

Pipeline: Eine Pipeline ist ein benutzerdefiniertes Modell einer CD-Pipeline. Der Code einer Pipeline definiert Ihren gesamten Build-Prozess, der normalerweise Phasen zum Erstellen einer Anwendung, zum Testen und dann zum Bereitstellen umfasst.

Knoten: Ein Knoten ist eine Maschine, die Teil der Jenkins-Umgebung ist und eine Pipeline ausführen kann.

Stage: Ein Stageblock definiert eine konzeptionell unterschiedliche Teilmenge von Tasks, die über die gesamte Pipeline durchgeführt werden (z. B. die Phasen „Build“, „Test“ und „Deploy“), die von vielen Plugins verwendet werden, um den Status/Fortschritt der Jenkins-Pipeline zu visualisieren oder darzustellen.

Step: Eine einzelne Task. Grundsätzlich teilt ein Step Jenkins mit, was zu einem bestimmten Zeitpunkt (oder „Step“ im Prozess) zu tun ist. Um beispielsweise den Shell-Befehl auszuführen, verwenden Sie den sh-Step: sh 'make'.

### Einzel Step-Pipeline

### 1. Nachdem wir uns bei Jenkins angemeldet haben, klicken wir auf die Menüoption „New Item“. Geben wir den Namen der Jenkins-Pipeline ein, klicken wir auf "Pipeline". Drücken wir dann die „OK“-Taste.

### 2. Wir gelangen direkt zur Seite "Konfiguration" des Projekts, wählen wir den Abschnitt "Pipeline" aus und fügen wir den folgenden Code ein.

### Jenkinsfile (Deklerative Pipeline)

### …

### *pipeline {*

### *agent none*

### *stages {*

### *stage('Build Gradle') {*

### *steps {*

### *echo "sample shell code oder DSL -code"*

### *}*

### *}*

### *}*

### *}…*

### Innerhalb der Pipeline kann es „stages“ geben (Wir haben eine). Innerhalb der „stages“ können sich mehrere „stage elements“ befinden.

### Innerhalb jeder „stage“ müssen „steps“ vorhanden sein. Die Schritte selbst sind Jenkins-Befehle.

### „echo“ Befehl wird einfach etwas auf der Konsole ausgeben. Es kann nützlich sein, um Werte anzuzeigen, während die Pipeline Fortschritte macht.

### 

### 

### Running Multiple Steps

### Pipelines bestehen aus mehreren Schritten, mit denen wir Anwendungen erstellen, testen und bereitstellen können. Mit Jenkins Pipeline können wir auf einfache Weise mehrere Schritte zusammenstellen, mit denen wir jede Art von Automatisierungsprozess modellieren können.

### Wenn ein Schritt erfolgreich ist, geht es zum nächsten Schritt über. Wenn ein Schritt nicht korrekt ausgeführt wird, schlägt die Pipeline fehl.

### Wenn alle Schritte in der Pipeline erfolgreich abgeschlossen wurden, gilt die Pipeline als erfolgreich ausgeführt

### … /\*Ein Sample Multi Steps Pipeline\*/

### *pipeline {*

### *agent none*

### *tools {*

### *gradle 'gradle-7.4.2'*

### *}*

### *stages {*

### *stage('Build Gradle') {*

### *agent {label 'jenkins'}*

### *steps {*

### *checkout([$class: 'GitSCM', branches: [[name: '\*/main']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/Selahattinasn/gradle\_exercise\_java11']]])*

### *sh 'gradle clean'*

### *sh 'gradle build'*

### *sh 'whoami'*

### *}*

### *}*

### *stage('Build Docker Image') {*

### *agent {label 'jenkins'}*

### *steps {*

### *script{*

### *sh 'docker build -t sennurmiray/snapshotintegration . '*

### *sh 'docker tag sennurmiray/snapshotintegration:latest docker.registry:5000/selo/sennurmiray/snapshotintegration:v1 '*

### *}*

### *}*

### *}*

### *stage('Push Docker Image') {*

### *agent {label 'jenkins'}*

### *steps {*

### *script{*

### *withCredentials([string(credentialsId: 'dockerHub-pwd', variable: 'docker\_login')]) {*

### *sh 'docker login -u sennurmiray -p ${docker\_login}'*

### *}*

### *sh 'docker push sennurmiray/snapshotintegration '*

### *sh 'docker push docker.registry:5000/selo/sennurmiray/snapshotintegration:v1 '*

### *}*

### *}*

### *}*

### *stage('Send Docker Image') {*

### *agent {label 'jenkins'}*

### *steps {*

### *sshPublisher(publishers: [sshPublisherDesc(configName: 'jenkins-node', transfers: [sshTransfer(cleanRemote: false, excludes: '', execCommand: 'pwd', execTimeout: 120000, flatten: false, makeEmptyDirs: false, noDefaultExcludes: false, patternSeparator: '[, ]+', remoteDirectory: './', remoteDirectorySDF: false, removePrefix: 'build/libs/', sourceFiles: 'build/libs/\*SNAPSHOT.jar')], usePromotionTimestamp: false, useWorkspaceInPromotion: false, verbose: false)])*

### *}*

### *}*

### 

### *stage('deploy kube') {*

### *agent {label 'jenkins'}*

### *steps {*

### *sh 'uname'*

### *sh 'ip r'*

### *}*

### *}*

### *stage('deploy KubeNode') {*

### *agent {label 'kubeNode'}*

### *steps {*

### *sh 'uname'*

### *sh 'ip r'*

### *sh 'runuser -l selahattin -c whoami'*

### *sh 'runuser -l selahattin -c "uptime"'*

### *sh 'runuser -l selahattin -c "kubectl cluster-info"'*

### *sh 'runuser -l selahattin -c "cd /home/selahattin"'*

### *sh 'runuser -l selahattin -c "pwd"'*

### *sh 'runuser -l selahattin -c "ls"'*

### *sh 'runuser -l selahattin -c "ls"'*

### *sh 'runuser -l selahattin -c "kubectl apply -f deploymentfile.yaml"'*

### *sh 'sleep 17'*

### *sh 'runuser -l selahattin -c "kubectl get pods"'*

### *sh 'runuser -l selahattin -c "curl 192.168.49.2:30456/greeting"'*

### *}*

### *}*

### *}*

### *}* ###

### 

### 