Abschlussprüfung Sommer 2023

Fachinformatiker für Systemintegration

Dokumentation zur betrieblichen Projektarbeit

Kontinuierliche Integration -und Verteilung

Einrichtung eines Systems zur kontinuierlichen Integrations- und Verteilung (CI/CD-Pipeline)

Abgabedatum: Kassel, den 25.04.2023

Prüfungsbewerber:

Selahattin Karakaya

Memelstr. 2 A

34537 Bad Wildungen

Ausbildungsbetrieb:

NEXUS / IPS

Unternehmensstraße 1

34131 Kassel

nexus | ips
integrated process solutions

1 Inhaltsverzeichnis

Abbildungsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API ..... *Application Programming Interface*

CD..... *Continuous Delivery*

CI ..... *Continuous Integration*

GUI..... *Grafical User Interface*

JRE ..... *Java Runtime Environment*

JVM..... *Java Virtual Machine*

LAN..... *Local Area Network*

NAT..... *network address translation*

ssh ..... *Secure Socket Shell*

yml ..... yet another markup language, *Yet Another Markup Language*

1 Einleitung

1.1 Projektumfeld

Die Nexus / IPS GmbH ist eine Tochterfirma der Nexus/ag und ein herstellerunabhängiger Anbieter für die Dokumentation der Medizinprodukte-Aufbereitung und Prozesskommunikation in Deutschland mit seinen Softwarelösungen. Nexus / IPS beschäftigt 33 Mitarbeiter mit im Standort Kassel. Seit 30 Jahren ermöglicht Nexus /IPS eine einfache und transparente Dokumentation aller Prozessschritte im Medizinprodukte-Kreislauf. Die Nexus / IPS GmbH ist ein Unternehmen der Nexus -Gruppe und entstand 2020 aus der IBH Datentechnik GmbH und der Nexus IPS GmbH.

Weil die Hauptbeschäftigung der Nexus/IPS auf Softwareentwicklung basiert, spielt die Methodik in der Softwareentwicklung eine wichtige Rolle. Das kontinuierliche Integration -und Verteilung System (Continuous Integration und Continuous Delivery System) (CI/CD Pipeline) Projekt wurde aus einem internen Auftrag von am Softwareentwicklungsprozess beteiligten Vorgesetzten initiiert.

Kontinuierliche Integration ist eine Praxis, bei der alle Codeänderungen in den Master-Branch eines häufig gemeinsam genutzten Quellcode-Repositorien integriert werden, und danach wo die nachfolgende Kompilieren-, Testen-, Bauen- und Verteilungsverfahren automatisch durchgeführt wird. Kontinuierliche Verteilung ist eine Softwareentwicklungspraxis, die in Verbindung mit CI arbeitet, um den Anwendungsfreigabeprozess und auch für die Infrastrukturbereitstellung zu automatisieren. Sobald der Code als Teil des CI-Prozesses getestet und erstellt wurde, übernimmt CD in den letzten Phasen, um sicherzustellen, dass er mit allem ausgestattet ist, was er benötigt, um ihn jederzeit in jeder Umgebung bereitzustellen.

1.2 Projektziel

Durch CI/CD werden viele oder alle der menschlichen Eingriffe bei der Softwareentwicklung automatisiert, die traditionell erforderlich sind, um neuen Code von einem Commit in die Produktion zu bringen. In diesem Zusammenhang sind Bauen-, Testen und Delivery/Deployment-Phasen sowie die Infrastruktur Bereitstellung zu berücksichtigen.

In der Firma werden die Codeänderungen mit Jenkins schon kompiliert, getestet und gebaut. Mit diesem Projekt ist es gezielt, die hergestellten Artefakte, die im .jar Form ist, in einem Docker Image umzuwandeln, dieses Image an einem privaten Registry zu senden und die Applikation in einer Ausführungsumgebung auszuführen. Und sind alle diese Prozesse automatisch durchzuführen.

1.3 Projektbegründung

Die im .jar Form hergestellte Artefakt wird vom Produktmanager manuell gezogen und es wird überprüft, wie diese Anwendung funktioniert. Diese Prozesse werden manuell abgewickelt. Andererseits scheint es möglich, diese manuell gehandhabten Prozesse mit Hilfe von Container-Technologie und CI/CD-Pipeline zu automatisieren. Aus diesen Gründen wurde beschlossen, das Projekt zu starten.

Dadurch wird auch in Gesamt erhebliche Zeit freigesetzt, die für die Produktentwicklung aufgewendet werden könnte, da durch Einsatz der CI/CD Pipeline Zeit eingespart werden.

1.4 Projektschnittstellen

Bei jedem Projektschritt wurden Hilfe und Rat von den Mitarbeitern am Arbeitsplatz für die Bedürfnisse und Systemanforderungen und -kapazitäten erhalten. Das Ergebnis des Projekts wird dem Auftragsgeber, Auftragsverantwortlicher und Ausbildungsverantwortlicher in der Firma präsentiert.

1.5 Projektabgrenzung

Im Rahmen des Projekts wurde versucht, die Anforderungen des CI/CD-Konzepts zu fokussieren. Konfigurationen virtueller Maschinen wie SSD- oder HDD-Festplattenauswahl, RAM-Kapazität werden im Projekt nicht berücksichtigt, da sie für das Projekt nicht kritisch sind. Die Netzwerkkonfigurationen sind nicht Teil dieses Projekts, da wird das vorhandene LAN des Unternehmens verwendet. Nach der Installation des Betriebssystems Ubuntu auf den Servern werden statische IP-Adressen verwendet. Außerdem werden die Konfigurationen des im LAN verwendeten Switches gelten.

2 Projektplanung

2.1 Projektphasen

Für die Vorbereitung des Projekts sind 35 Stunden vorgesehen. Das Projekt wurde in vier Hauptphasen abgewickelt und die grobe Zeitplanung ist unten dargestellt.

Projektphase	Geplante Zeit
Planung	5 h
Durchführung	18 h
Validierung	3 h
Projekt Abschluss	9 h
Gesamt	**35 h**

Tabelle 1: Grobe Zeitplanung

Eine detaillierte Zeitplanung befindet sich im Anhang A1.

2.2 Personalplanung

Tabelle 2: Personalplanung

Name	Tätigkeit	Zeitaufwand
Selahattin Karakaya	Projektumsetzung	35 h
Michael Sieber	Projektdefinition, Projektunterstützung	7 h
André Hoßbach	Ansprechpartner, Projektübergabe	3 h
Gesamt		**45 h**

2.3 Sachmittelplanung

Für die Installation der im Rahmen des Projekts zu verwendenden virtuellen Maschinen wird ein Notebook mit SSD-Festplatte verwendet. Dieses Notebook muss mit dem LAN-Netzwerk des Unternehmens verbunden sein.

Bei der Auswahl der einzusetzenden Software wurde, um möglichst unnötige Kosten zu vermeiden, vom bereits gekauft oder Freeware Software ausgewählt, sofern sie für die Projektziele ausreicht. Liste der ausgewählten Software befindet sich im Anhang-2.

2.4 Kostenplanung

Die Kosten der Hardware und Software sind nicht zu berücksichtigen, die Ausgewählte Hard- und Software wurde zuvor vom Unternehmen erworben oder sind Freeware.

Im Rahmen des Projektes wurde Räumlichkeit mit den Arbeitskosten des Personals berechnet. Während der Praktikumszeitraum des Prüfungsbewerbers gibt es kein Kosten für die Firma. Für ein Software Produkt Manager 30,93 €, für ein Senior Software Developern 25,83 € zu berücksichtigen. Für die Mitarbeiter wird 35 € für die Nutzung der Räumlichkeiten, Strom und Arbeitsmaterialien berücksichtigt.

Ausgaben	Zeit	Kosten pro Stunde	Kosten
Ausbildungsverantwortlicher	3 h	30,93 €	*92,79 €*
Auftragsverantwortlicher	7 h	*25,83 €*	*180,81 €*
Räumlichkeit	45 h	*35,00 €*	*1575,00 €*
		Gesamt	***1848,6 €***

Tabelle 3: Kostenaufstellung

2.5 Analyse

2.5.1 Ist-Analyse

Vor dem Projekt wurde das Jenkins-Tool im Unternehmen verwendet. Der vom Entwickler vorbereitete Quellcode wird in das GitHub-Repository gepusht. Nach diesem Push-Verfahren wird der Quellcode von Jenkins gezogen und die Kompilieren-, Testen- und Bauen-Verfahren werden automatisch von Jenkins ausgeführt. Das resultierende Artefakt in .jar-Form wird manuell entnommen.

Bei all diesen Prozessen ist im Vergleich zur CI/CD-Pipeline ein längerer Zeit- und Arbeitsaufwand erforderlich. Es kostet viel Zeit und Geld. Eine Swim-Lane-Diagramm zur Ist-Analyze befindet sich im Anhang-3.

2.5.2 Soll-Konzept

Es ist wünschenswert durch Nutzung der Docker-Containers, den gesamten Prozess im Rahmen einer CI/CD-Pipeline zu automatisieren, die mit einem Jenkinsfile definiert wird. Darüber hinaus wird es auch gezielt, die erhaltenen Docker-Images in einem separaten privaten Repository aufzubewahren. Darüber hinaus ist es wünschenswert, die Anwendungen bereitzustellen, indem ein Kubernetes-Cluster auf einem anderen Server eingerichtet wird.

Mit diesem Projekt werden folgende Prozesse im Rahmen einer CI/CD Pipeline automatisiert durchgeführt. Das in .jar-Form-Artefakt wird mit einem Dockerfile in ein Docker-Image umgewandelt. Dieses Docker-Image wird an den privaten Repository-Server gesendet, der auf einem separaten Computer installiert und als Agent-Knoten für den Automatisierungsserver definiert ist, und dort das Docker-Image gespeichert. Diese Anwendung wird auf einem Kubernetes-Server mit einem Knoten bereitgestellt, der auf einem anderen Computer installiert und als Agent-Knoten auf dem Automatisierungsserver definiert ist.

Da für das im Projekt zu verwendende Jenkins Tool unterschiedliche Konfigurationen erforderlich sind, wird Jenkins getrennt von dem im Unternehmen neu installiert. All diese Prozesse laufen automatisch ab. Eine Swim-Lane-Diagramm zum Soll-Konzept befindet sich im Anhang-4.

2.6 Wirtschaftlichkeitsanalyse

Durch den Einsatz der mit dem Projekt zu beschaffenden CI/CD Pipeline werden Zeit und damit Personalkosten eingespart. Durch die Automatisierung der manuellen Arbeit entfällt der Zeitaufwand für diese Arbeit durch das Personal. Daher werden auch finanzielle Einsparungen erzielt.

Artefakte in .jar-Form werden vom Produktmanager manuell erntgenommen und in einem Ausführungsumgebung ausgeführt und geprüft. Dieser Vorgang dauert mindestens 30 Minuten pro Arbeitstag. Es sollte akzeptiert werden, dass dieser Prozess alle 220 Arbeitstage in einem Jahr durchgeführt wird. Das Personal, das diese Tätigkeit ausführt, ist Produktmanager und es wird akzeptiert, dass es einen Lohn von 30,93 Euro für jede geleistete Arbeitsstunde erhält. Basierend auf diesen Daten werden in einem Jahr 3402,3 Euro eingespart, was bedeutet, dass sich die Kosten des Projekts nach etwa sieben Monaten amortisiert haben und danach eingespart wird.

3 Durchführungsphase (Realisierung)

Im Rahmen des Projekts werden drei separate Server verwendet: Dies ist sowohl für die Isolierung als auch für die Sicherheit und Leistung eine angemessene Vorgehensweise. Die zu verwendenden Server sind Jenkins-Automation-Server, Docker-Registry-Server als privates Repository und ein Kubernetes/Minikube-Server, auf dem wir Artefakte bereitstellen können.

Im Rahmen der Systemeinrichtung werden im ersten Schritt drei virtuelle Maschinen als Server unter demselben Subnetz eingerichtet. Danach wurden die notwendigen Programme auf den Servern installiert und den Servern konfiguriert, damit diese den zugeteilten Zweck erfüllen können. Wir müssen auch einige Deklarationsfile wie Dockerfile, Docker-Compose-File und Kubernetes Manifest-File und Jenkinsfile schreiben. Danach werden alle Server entsprechend der anzuwendenden Pipeline konfiguriert. Schließlich wird die Pipeline ausgeführt.

3.1 Einrichtung von Virtuelle Maschinen

Hardware Voraussetzungen für Jenkins wurde aus offiziellen Internetseite vom Jenkins ermittelt (Anon., 2023). Wir werden für Jenkins-Server eine Virtuelle Maschine mit 4 GB RAM, 2 Kern CPU, 60 GB SATA, Net_Adapter Type: NAT verwenden.

Auf der Docker-Registry-Website konnte keine Empfehlung für die erforderlichen Hardware-Voraussetzungen gefunden werden (Docker-Registry, 2013-2023). Für den Server von Docker-Registry eine Virtuelle Maschine mit 2 GB RAM + 10 GB Swap Area) , 2 Kern CPU, 60 GB SATA, Net_Adapter Type: NAT verwenden. Weil wir die drei Server auf einem einzigen Laptop betrieben wollen, SWAP-Space wurde definiert, um verwendet zu werden, wenn mehr RAM-Kapazität benötigt wird.

Hardware Voraussetzungen für Minikube wurde aus offiziellen Internetseite vom Kubernetes ermittelt (The Kubernetes Authors, 2023). Wir werden für den Deployment-Server eine Virtuelle Maschine mit 4 GB RAM, 2 Kern CPU, 60 GB SATA, Net_Adapter Type: NAT verwenden.

Als Betriebssystem wird aufgrund der Eigenschaften wie Stabilität, Benutzerfreundlichkeit und weit verbreitenden Community-Unterstützung Ubuntu 20.04.LTS ausgewählt.

Es ist erforderlich, dass alle zu verwendete Virtuelle Maschinen sicher und isoliert miteinander kommunizieren können. Als beste Praktik werden sich die Maschinen in denselben LAN befinden. Wie im Abschnitt „Projektabgrenzung“ erklärt, werden nach der Installation des

Betriebssystems „Ubuntu“ auf den Servern, werden statische IP-Adressen zugewiesen. Außerdem werden die Konfigurationen des im LAN verwendeten Switches gelten.

3.2 Einrichtung von ssh Verbindungen zwischen virtuellen Maschinen

Die Hauptmerkmale der CI/CD Pipeline ist die Automatisierung. Auch bei der Kommunikation zwischen Jenkins-Server und anderer virtuellen Maschinen, die als Agent-Node von Jenkins betrieben werden, muss die Authentifizierung automatisch erfolgen. Mit Openssh wird sichere, verschlüsselte Verbindung zwischen vertrauenswürdigen Rechnern über ein Netzwerk eingerichtet.

Die Anweisungen einschließlich der Reihenfolge all dieser Software Installation Vorgänge, der erforderlichen Befehle und Erläuterungen dazu finden Sie in Anhang A5.

3.3 Einrichtung des Jenkins-Servers

Jenkins wurde im Rahmen des Projekts für automatisierte Prozesse als geeignet befunden. Jenkins ist ein eigenständiger Open-Source-Automatisierung-Server, der zur Automatisierung aller Arten von Aufgaben zum Kompilieren der Quellcode in Binary, Testen und Verpacken im .jar oder .war Form genutzt werden kann. Das ausführbare Package kann vom Jenkins danach in Delivery- oder Deployment-Umgebung gesendet werden. Es hilft, den kontinuierlichen Integrationsprozess zu realisieren Jenkins ist kostenlos und vollständig in Java geschrieben. Es ist eine Weltweit sehr häufig verwendete 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. Installation von benötigter Software

Für die Jenkins-Serverinstallation müssen wir Git, Docker, Docker-Compose, Open-JDK.11, Gradle auf dem Rechnern installieren. Da Push- und Pull-Verfahren mit GitHub Repo durchgeführt werden, installieren wir die Applikation „git“. Außerdem muss Gradle installiert werden, um den von GitHub abgerufenen Quellcode zu testen und ihn als Artefakt im .jar-Format zu packen. Docker muss installiert sein, um das gepackte Binär-Image zu erhalten. Da Jenkins eine Java-basierte Anwendung ist, benötigen wir auch eine Java-Laufzeit-Umgebung. Bei Bedarf wird vorzugsweise das Java Development Kit verwendet, um das mögliche Testen zu erleichtern. OpenJDK-11, das sowohl Open Source als auch einfach zu installieren ist, wird bevorzugt.

Die Anweisungen einschließlich der Reihenfolge all dieser Softwareinstallationsvorgänge, der erforderlichen Befehle und Erläuterungen dazu finden Sie in Anhang A6.

3.3.1 Schreiben von Dockerfile

Ein Dockerfile wird benutzt, um eine Container-Image haben zu können. Als Base-Image wird in Docker-Hub bereits befundene „adoptopenjdk/openjdk11“ verwendet.

Als „Working-directory“ muss innerhalb des Docker-Images ein Ordner mit einem frei wählbaren Namen angelegt werden. In diesem Projekt wurde ein Ordner mit dem Namen „temporary“ erstellt.

Das von Gradle erstellte Artifact, das im .jahr-Format ist , wird in diesen Working-directory kopiert.

Um die Anwendung in Form eines Docker-Containers anzuwenden, muss ein spezieller Befehl für die Anwendung eingegeben werden. In unserem Projekt ist es „CMD ["java","-jar", "./containertest-0.0.1-SNAPSHOT.jar"] “ (Bernd Öggl, 2020).

Die Dockerfile befindet sich im Anhang A7.

3.3.2 Installieren von Jenkins auf dem Rechner

Als Automatisierung-Server wurde Jenkins installiert. Zuerst wird der Repository-Schlüssel zum System hinzugefügt. Nachdem der Schlüssel hinzugefügt wurde, kehrt das System mit OK zurück. Als nächstes wird die Debian-Paket-Repository-Adresse zu den Quellen hinzugefügt. Serverliste hinzugefügt. Die Liste der erforderlichen Befehle befindet sich in Anhang A8.

Standardmäßig wird Jenkins auf Port 8080 unter Verwendung von Serverdomänennamen oder IP-Adressen besucht: http://<your_server_ip_or_domain>:8080. In diesem Projekt lautet die Jenkins-Server-Adresse 192.168.22.129:8080.

Als nächstes wird der Bildschirm „Jenkins Unlock“ aufgerufen, der den Speicherort des Anfangskennworts anzeigt: Das Administratorkennwort wird abgerufen und zur Authentifizierung des Servers verwendet. Danach wird der erste "Admin-Benutzer" definiert und dann werden die Basis-Plugins installiert.

Als weitere Möglichkeit kann der Jenkins-Server als Docker-Container installiert werden.

```
docker run --name myjenkins -p 8080:8080 -p 50000:50000 -v
    ~/$(USER):/var/jenkins_home jenkins
```

3.4 Einrichtung des Docker-Registry-Servers

3.4.1 Installation von benötigter Software

Docker Registry ist kostenlos, einfach zu installieren und verfügt über die Funktionalität zum Speichern von Docker-Image-Dateien. Aufgrund dieser Funktionen wird Docker-Registry als privates Repository verwendet, um genau zu steuern, wo Docker-Images gespeichert werden, und um die Verteilung von Docker-Images in Ihrer internen Entwicklungsumgebung sicherzustellen.

Die Docker-Registry-Anwendung wird als Multi-Container Applikation ausgeführt. Dazu müssen docker und docker-compose auf dem Server installiert werden.

Die Installation von Programmen ist genau die gleiche wie auf einem Jenkins-Server. Daher werden auch hier die gleichen Befehle verwendet.

3.4.2 Schreiben von Docker-Compose File

Um Multi-Container-Anwendungen auszuführen, wird eine Compose-Datei geschrieben, die vom Docker-Compose-Programm ausgeführt wird. Dann werden zwei separate Container-Images zusammen für die Docker-Registrierungsanwendung und auch eine GUI verwendet.

In diesem Projekt erreichen wir die als Docker-Container laufende Anwendung über Port 8080. Für die Anwendung wird ein Volum zum Speichern von Docker-Images definiert. Der Pfad zu diesen Volumen ist als „/var/lib/registry“ geplant.

Die Docker-Compose-Datei befindet sich im Anhang A9.

Nachdem wir die docker-compose-Datei irgendwo auf dem Server gespeichert und diese Manifest-Datei mit dem Befehl „docker-compose -f <compose-file-name> up“ ausgeführt haben, starten wir die docker-compose-Anwendung. In diesem Projekt gehen wir nach dem Erstellen eines Ordners namens „docker-compose“ im Home-Verzeichnis in diesen Ordner und starten die Anwendung doker-registry mit dem Befehl „doker-compose -f docker-compose“. In diesem Projekt ist die Adresse von Docker-Registry-Server 192.168.22.128:8080 .

3.5 Einrichtung des Delivery/Deployment Servers

Kubernetes bietet automatisierte Funktionen zum Verwalten und Orchestrieren mehrerer Container. Die von Kubernetes bereitgestellten Funktionen tragen dazu bei, den Zeit- und Arbeitsaufwand für die Bereitstellung von Anwendungen zu reduzieren. Aus diesen Gründen wird auf unserem Delivery/Deployment-Server ein Kubernetes-Cluster eingerichtet.

Ein Kubernetes-Cluster mit einem einzelnen Knoten reicht aus, um containerisierte Anwendungen in der Bereitstellungsumgebung im Projekt zu sehen. Minikube ist eine leichtgewichtige Kubernetes-Anwendung, die eine virtuelle Maschine auf Ihrer lokalen Maschine erstellt und ein einfaches Cluster mit einem einzigen Knoten bereitstellt. Und Minikube ist einfach zu installieren, verbraucht weniger Ressourcen, ist Open Source und kostenlos. Aus diesen Gründen wird Minikube zum Bereitstellen von Kubernetes-Clustern verwendet. Die IP-Adresse des Lieferungs-/Verteilungsservers lautet 192.168.22.134.

3.5.1 Installation von benötigter Software

Es werden auf dem Delivery/Deployment-Server git, docker und docker-compose, openjdk11 und schließlich benicube und kubectl installiert. Docker- und Java-Applikationen sind erforderlich, um diese Image auszuführen. Docker ist in diesem Zusammenhang ebenfalls erforderlich, da die Minikube-Anwendung auch von einem Container-Image ausgeführt wird.

Kubectl ist das Terminalprogramm, das zum Verwalten von Minikube erforderlich ist.

Die Installation von Docker und openjdk11 ist dieselbe wie beim Jenkins-Server. Daher werden auch hier die gleichen Befehle verwendet.

Die Reihenfolge der Minikube- und kubectl-Softwareinstallationsprozesse, die erforderlichen Befehle und die Anleitung mit ihren Erläuterungen finden Sie in Anhang A10.

3.5.2 Schreiben von Deployment und Service Files for Kubernetes Cluster

Ein Objekttyp namens Deployment wird verwendet, um eine Anwendung in einer Kubernetes-Umgebung auszuführen und nach Bedarf zu skalieren, zurückzusetzen und bereitzustellen. Deployment ist ein API-Objekt in Kubernetes. Ein Service-Objekt wird auch verwendet, um die Anwendung im Internet zu veröffentlichen und zu übertragen. Für Kubernetes-Objekte wird eine deklarative Methode verwendet, um dem Benutzer die Bereitstellung und Verwaltung des Objekts zu erleichtern. Eine Datei im .yaml-Format wird erstellt und das Verteilungsobjekt und das Dienstobjekt werden in dieser Datei definiert. Die Konfigurationsfile für Deployment und Service-Objects befinden sich im Anhang A11.

3.6 Pipeline

3.6.1 Global Tool Konfigurationen

Programme, die automatisch von Jenkins in der Pipeline verwendet werden sollen, müssen auf Jenkins im Abschnitt globale Toolkonfigurationen angegeben werden.

In diesem Zusammenhang sollte der Pfad zu Git, OpenJDK11, Gradle, Docker Executable eingegeben werden..

Die Anweisung dazu befindet sich im Anhang A12.

3.6.2 Docker-Registry-Server und Deployment-Server als Agent-Knoten von Jenkins-Server Konfigurieren

Um Prozesse auf einem Jenkins-Server und anderen Knoten als Teil der CI/CD-Pipeline auszuführen, müssen diese Knoten natürlich als Agenten für Jenkins-Server definiert werden. Dazu müssen wir eine ssh-Verbindung zwischen Jenkins-Server und Agent-Knoten herstellen und die „Agent“-Datei vom Jenkins-Server auf den Agent-Knoten kopieren.

Die Anweisung dazu befindet sich im Anhang A13.

3.6.3 Schreiben von Jenkinsfile

Wir klicken auf die Menüoption „Neues Element“, um sich bei Jenkins anzumelden. Wir geben den Namen der Jenkins-Pipeline ein und klicken auf Pipeline. Danach drücken wir die „OK“-Taste. Wir gelangen direkt zur Seite "Konfiguration" des Projekts, wählen den Abschnitt "Pipeline" aus und fügen wir den Code ein, die im Anhang A14 befinder Jenkinsfile steht.

Innerhalb der Pipeline kann es „stages“ eingegeben werden. Innerhalb der „stages“ kann es mehrere „stage elements“ geben. Innerhalb jeder „stage“ müssen „steps“ vorhanden sein. Es kann hilfreich sein, Werte im Verlauf der Pipeline anzuzeigen. Pipelines bestehen aus mehreren Schritten, mit denen wir Anwendungen erstellen, testen und bereitstellen können. Jenkins Pipeline macht es einfach, mehrere Schritte zu kombinieren, die verwendet werden können, um jeden Automatisierungsprozess zu modellieren. Wenn ein Schritt erfolgreich ist, wird der nächste Schritt gestartet. Pipeline schlägt fehl, wenn ein Schritt nicht korrekt ausgeführt wird. Wenn alle Schritte in der Pipeline erfolgreich abgeschlossen sind, erhalten Sie die Meldung, dass die Pipeline erfolgreich ausgeführt wird..

Die Jenkinsfile befindet sich im Anhang A14.

4 Validierungsphase

Eine Pipeline wird ausgeführt, um das Projekt zu validieren. Ein Quellcodebeispiel, das die Programmiersprache Java beschreibt, wurde dem GitHub-Repository entnommen.

Alle Stufen der Pipeline werden in Jenkinsfile deklariert und Jenkinsfile wird in Pipeline eingegeben.

Zum Ausführen der Pipeline wird die Konsole in der GUI von Broser verwendet. Sobald wird auf dem Knopf „Buildnow“ gedruckt, startet die Pipeline zu laufen.

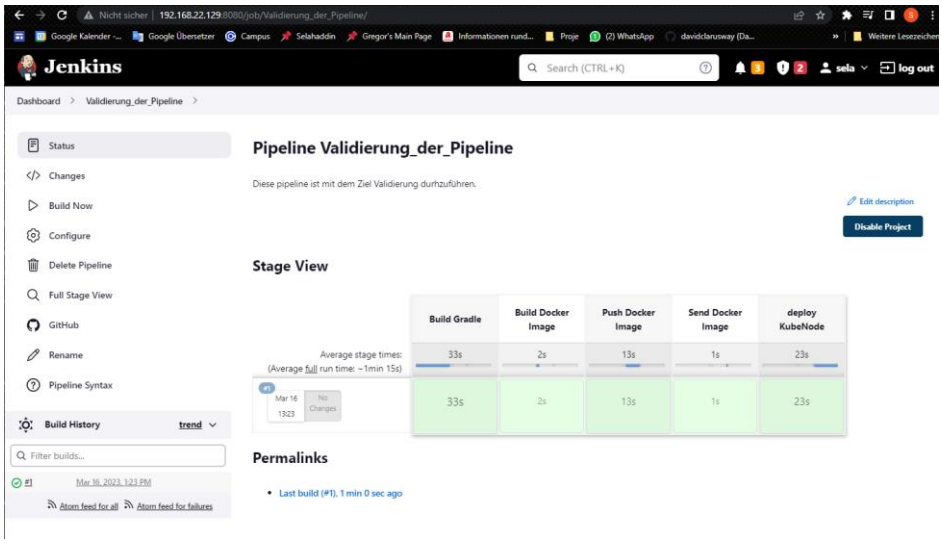

Abbildung 1: Pipeline Start

Hier sehen wir die Stages.

Zunächst wird der Quellcode von GitHub übernommen.

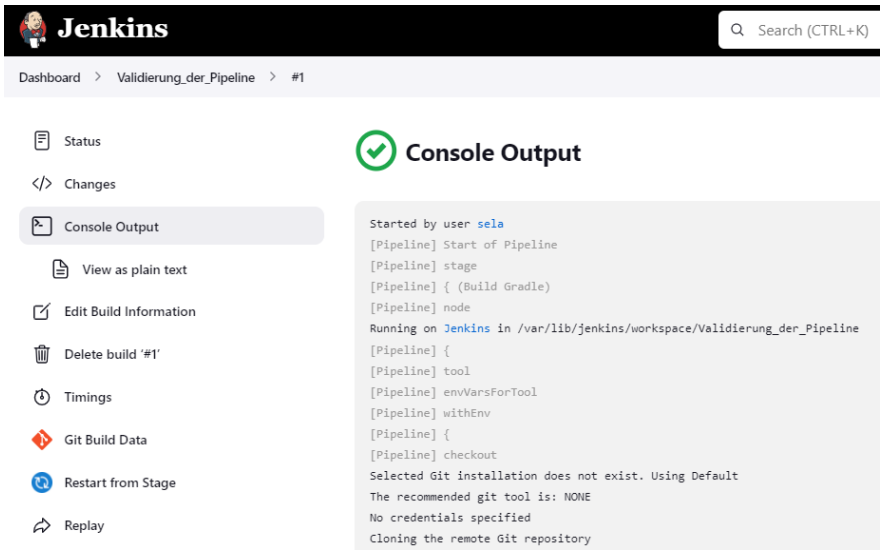

Abbildung 2: Cloning QuellCode

Dann werden Unit-Tests und Kompilierungsprozesse mit Gradle durchgeführt.

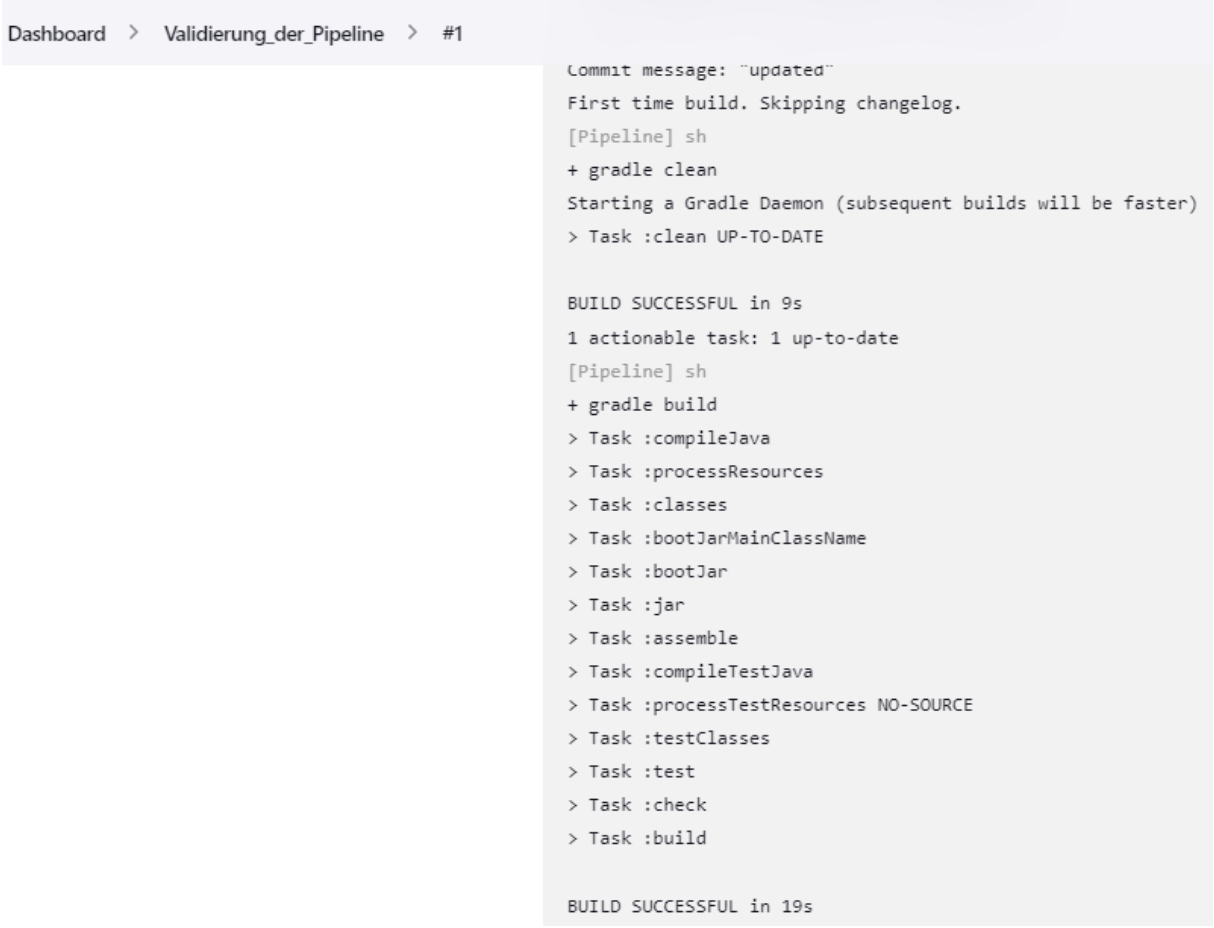

Abbildung 3: Test -und Bauen Verfahren

Dann werden Docker-Image-Generierung und Tagging-Operationen durchgeführt..

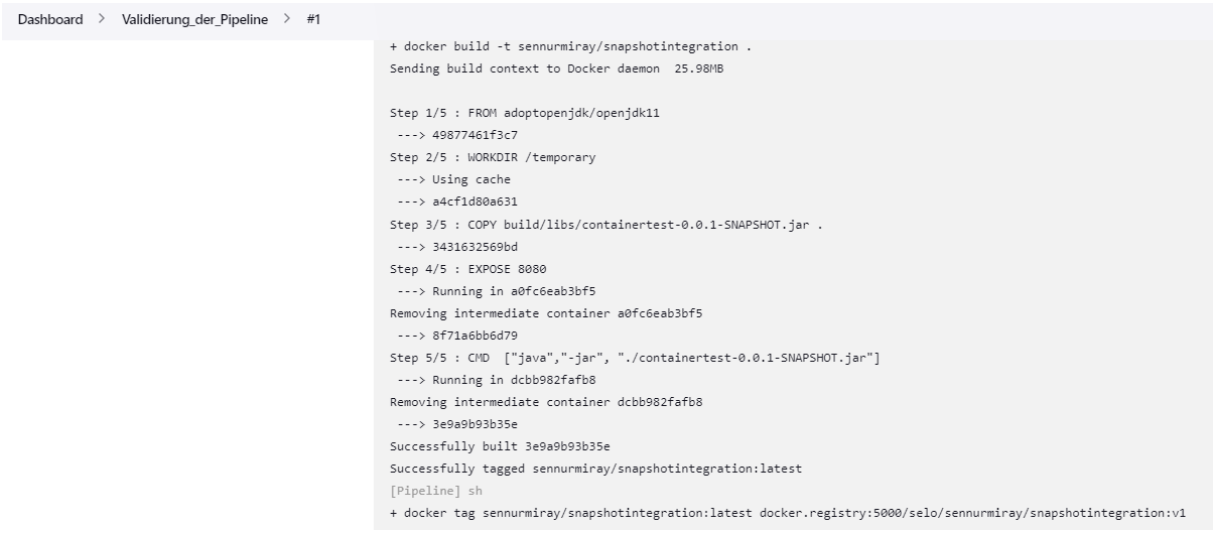

Abbildung 4: Docker-Image Verfahren

Endlich wird die Applikation als Deployment-Objekt im Delivery-Server durchgeführt.

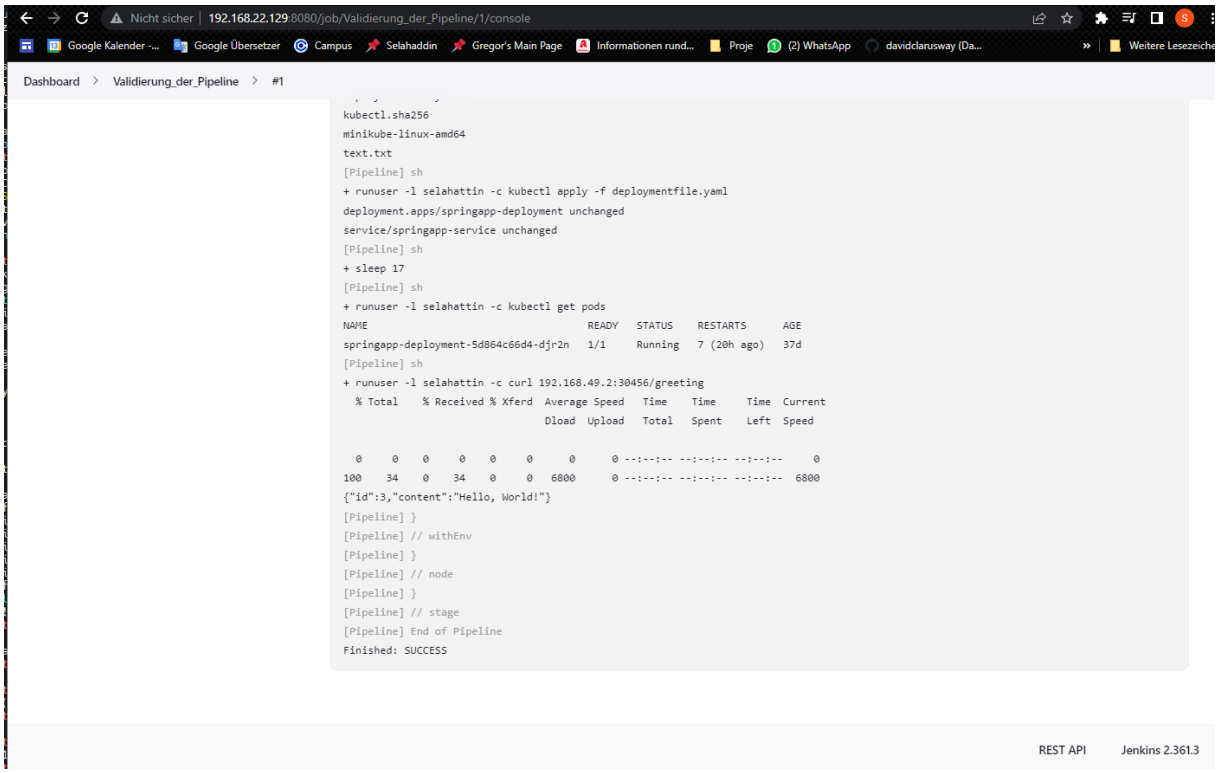

Abbildung 5: Deployment

Auf die Anwendung kann mit einem Browser auf dem Hauptserver zugegriffen werden.

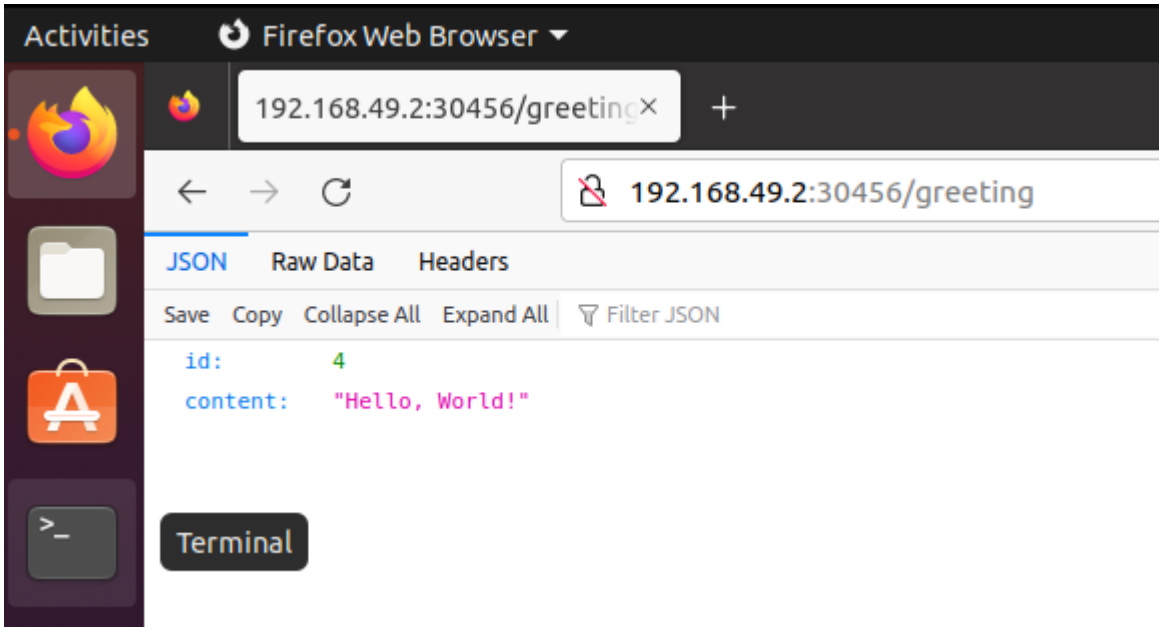

Abbildung 6: Browser Output

5 Fazit

5.1 Soll-/Ist-Vergleich

Die Projektziele wurden erreicht. Wie in der Validierungsphase geprüft werden, können von GitHub pullen, Kompilieren, Testen, Bauen, Containerisieren und Bereitstellung Verfahren automatisch durchgeführt werden. Damit wird viel Zeit verdient, und werden menschliche Eingriffe verringert.

Wie in der Tabelle 1 zu erkennen ist, konnte Zeitplan bis auf wenige Ausnahmen eingehalten werden.

Phase	Geplant	Tatsächlich	Differenz
Planung	5 h	5 h	
Durchführung	18 h	19 h	+1 h
Validierung	3 h	2 h	-1 h
Projekt Abschluss	9 h	9 h	
Gesamt	**35 h**	**35 h**	

Tabelle 4: Soll-/Ist-Vergleich

Wie geplant konnte in 35 Stunden abgeschlossen werden. Bei der Durchführungsphase mussten eine Stunde mehr und für Validierungsphase konnte eine Stunde weniger benutzt werden.

5.2 Lessons-Learned

Im Rahmen dieses Projekts hat Herr Karakaya sowohl das Server-Setup als auch die Anwendung von Technologien wie Docker und Kubernetes praktisch ausgeübt. Im Zuge dessen erlebte Herr Karakaya theoretisch und praktisch die ganz entscheidenden Vorteile, die das deklarative Verfahren und die Containertechnologie bieten, sowohl in Bezug auf den Hardwareverbrauch als auch auf die Zeitersparnis.

5.3 Ausblick

Das von Herr Karakaya etablierte System kann sowohl in der Testumgebung als auch in der Stage- und Produktivumgebung eingesetzt und mit anderen Technologien wie Vagrant, Ansible, Terraform, GitLab und Helm weiterentwickelt werden.

Literaturverzeichnis

Anon., 2023. *Jenkins.* [Online] Available at: https://www.jenkins.io/doc/book/installing/linux/#debianubuntu [Zugriff am 16 03 2023].

Bernd Öggl, M. K., 2020. *Docker.* 2. Hrsg. Bonn: Rheinwerk.

Docker Inc., 2013-2023. *Docker-Container.* [Online] Available at: https://docs.docker.com/engine/install/ubuntu/ [Zugriff am 16 März 2023].

Docker-Registry, 2013-2023. *docker.* [Online] Available at: https://docs.docker.com/registry/configuration/ [Zugriff am 16 März 2023].

Kofler, M., 2017. *Linux, Das umfassende Handbuch.* 15. Hrsg. Bonn: Rheinwerk.

Schulz, M., 2021. *Continuous Integration mit Jenkins.* Bonn: Rheinwerk.

The Kubernetes Authors, 2023. *Minikube.* [Online] Available at: https://minikube.sigs.k8s.io/docs/start// [Zugriff am 16 März 2023].

Eidesstattliche Erklärung

Ich, Selahattin Karakaya, versichere hiermit, dass ich meine Dokumentation zur betrieblichen Projektarbeit mit dem Thema

> *Kontinuierliche Integration -und Verteilung – Einrichtung eines Systems zur kontinuierlichen Integrations- und Verteilung (CI/CD-Pipeline)*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Abgabeort, den 11.04.2023

_____

SELAHATTIN KARAKAYA

Anhang

A1 Detaillierte Zeitplanung

Planungsphase			**5 h**
1. Ist-Zustands		2 h	
1.1. Fachgespräch	1 h		
1.2. Prozessanalyse	1 h		
2. Soll-Konzept		3 h	
2.1. Fachgespräch	1 h		
2.2. Auswahl von bevorzugten Werkzeugen und Applikationen	2 h		
Entwurfsphase			**19 h**
1. Einrichtung von Virtuelle Maschinen		1 h	
2. Einrichtung von ssh Verbindungen zwischen Maschinen		1 h	
3. Einrichtung des Jenkins-Servers		4 h	
3.1. Installieren von benötigten Software	2 h		
3.2. Schreiben von Dockerfile	1 h		
3.3. Installieren von Jenkins auf dem Server	1 h		
4. Einrichtung von Docker-registry Server		3 h	
4.1. Installieren von benötigten Software	2 h		
4.2. Schreiben von docker-compose file	2 h		
5. Einrichtung von Delivery/Deployment Server		4 h	
5.1. Installieren von benötigten Software	2 h		
5.2. Schreiben von Konfigurationfile für Deployment und Service Objekt	2 h		
6. Vorbereitung der Pipeline		5 h	
61. Global Tool Konfiguration	1 h		
6.2. Agent-Knoten Verbinden	1 h		
6.3. Schreiben von Jenkinsfile	3 h		
Validierungsphase			**2 h**
Projekt Abschluss			**9 h**
1. Soll-Ist Vergleich		1 h	
2. Erstellen der Projektdokumentation		8 h	
Gesamt			**35 h**

Tabelle 5: Detaillierte Zeitplanung

A2 Sachmittelplanung

Tabelle 6: Sachmittel-Planung

Software/Werkzeug	Funktion
Git	Versionskontrollsystem
Docker	Containerisierung-Tool
Docker-Registry	Multi Container Orchestrierungswerkzeug
Jenkins	Automatisierung-Server
Gradle	Build-Management-Automatisierung-Tool
Minikube	Kubernetes-Cluster Management Tool
MobaXterm	Remote-Desktop-Server
VMWare Workstation	Hosted Hypervisor

A3 Ist-Zustand

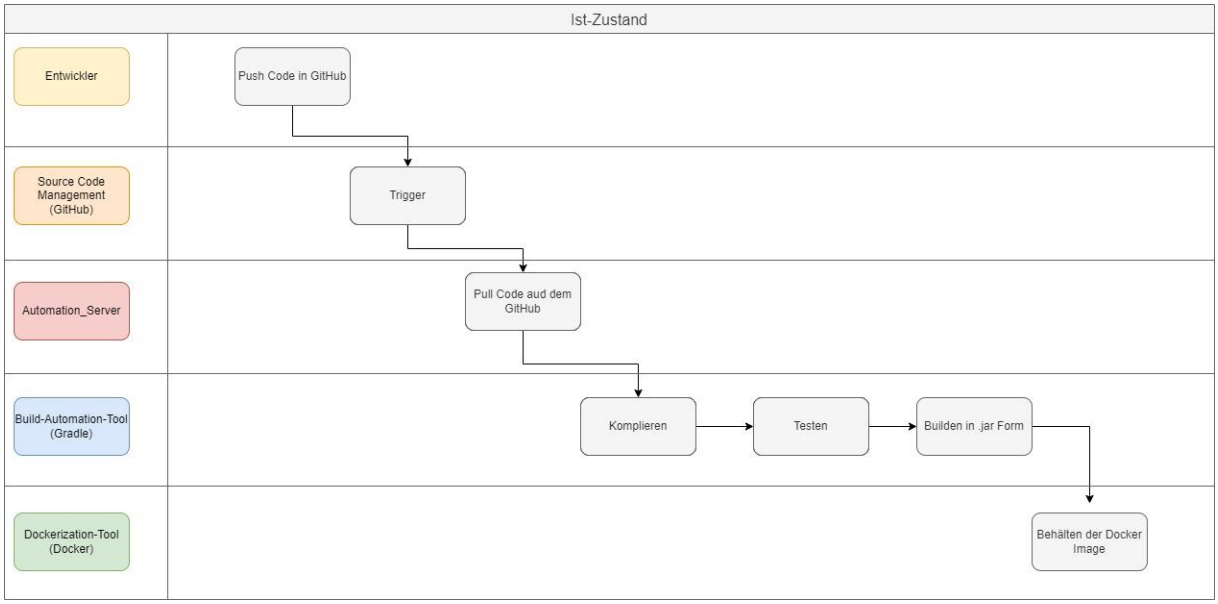

Abbildung 7 Ist-Zustand

A4 Soll-Konzept

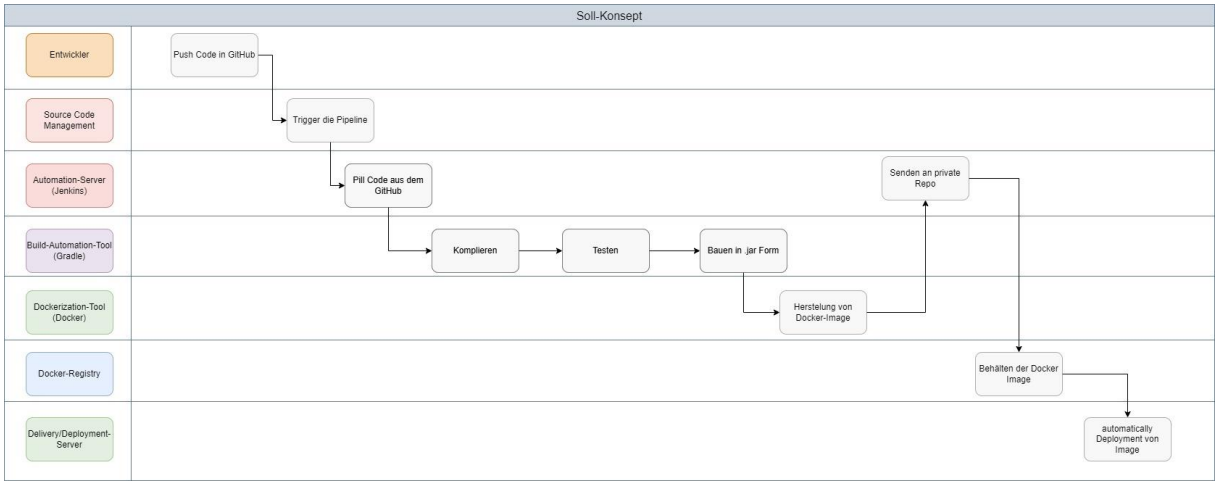

Abbildung 8 Soll-Konzept

A5 Einrichtung von ssh-Verbindungen zwischen Servern

Da wir das Betriebssystem Ubuntu-Debian verwenden, wird bei der Installation der Programme „apt“ bzw. „apt-get“ als Paketmanager verwendet. Linux-Komponenten und Repo sollen aktualisiert werden.

```
sudo apt update
sudo apt upgrade
```

OpenSSH ist eine Suite von Programmen für sichere Netzwerke. Es basiert auf dem Secure-Shell_protocol. Hier wird OpenSSH installiert.

```
sudo apt install openssh-server
```

ssh-service wird aktiviert.

```
sudo systemctl enable ssh
```

Oder:

```
sudo systemctl enable ssh --now
```

ssh-service wird gestartet.

```
sudo systemctl start ssh
sudo systemctl status ssh
```

Standardmäßig werden die Netzwerksicherheitseinstellungen vom ufw-Dienst verwaltet. Daher aktivieren wir den vorinstallierten ufw-Dienst.

```
sudo ufw allow ssh
sudo ufw enable
sudo ufw status
```

Eine ssh-Verbindung wird aufgebaut. Hier erfolgt die Authentifizierung mit den öffentlichen und privaten Schlüsseln, die wir auf dem Jenkins-Server erstellen werden. Ein öffentlicher/privater SSH-Schlüssel wird generiert.

```
ssh-keygen
```

Die ssh public key wird (id_rsa.pub) an den remote Servern gesendet. Hier sind die Remote-Maschinen des Docker-Registrierungsservers und des Verteilungsservers. Danach gehen wir in den .ssh-Ordner unter dem Home Directory.

```
cd ~/.ssh
ls
```

Die private Key wurde an remote Servern gesendet.

```
ssh-copy-id remoteUser@remoteHost
```

```
ssh username@Your-server-name-IP (ssh selahattin@192.168.22.134; ssh
    selo@jenkins
```

A6 Installation von Git, Docker, Docker-Compose, OpenJDK-11, Gradle

Zunächst wird die „apt“ Paket Verwaltungstools verwendet, um Ihren lokalen Paket Index zu aktualisieren.

```
sudo apt update -y
sudo install git -y
```

Einige notwendige Pakete werden für Docker installiert.

```
sudo apt install apt-transport-https ca-certificates curl software-
    properties-common
```

Es wird den offiziellen GPG-Schlüssel von Docker hinzufügen.

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-
    key add -
```

Es wird das stabile Repository eingerichtet.

```
sudo add-apt-repository "deb [arch=amd64]
    https://download.docker.com/linux/ubuntu focal stable"
```

Es wird ein Update durchgeführt, damit die neuste Version von Docker installiert werden kann.

```
apt-cache policy docker-ce
```

Die neueste Version von Docker CE wird installiert.

```
sudo apt install docker-ce
```

Wir überprüfen den Status des Docker-Dienstes.

```
sudo systemctl status docker
```

Der Benutzer wird der Docker-Gruppe hinzugefügt.

```
sudo usermod -aG docker ${USER}
```

Jenkins-Benutzer zur Docker-Gruppe hinzugefügt.

```
su - ${USER}
sudo usermod -aG docker jenkins
```

Testen der Docker-Installation.

```
docker run --rm hello-world
```

Melden Sie sich bei Bedarf beim Docker-Hub an.

```
docker login -u docker-registry-username
```

Die docker-compose binary wird heruntergeladen.

```
sudo curl -L
    "https://github.com/docker/compose/releases/download/1.29.2/dock
    er-compose-$(uname -s) -$(uname -m)" -o /usr/local/bin/docker-
    compose
```

Der entsprechende Einstellungsbefehl ist auf die ausführbare Docker-Compose-Datei festgelegt.

```
sudo chmod +x /usr/local/bin/docker-compose
```

Docker-Compose-Versionskontrolle.

```
docker-compose --version
```

Bereiten eines Ordners für docker-compose-File.

```
mkdir ~/compose-demo
```

Wir gehen zum Ordner.

```
cd ~/compose-demo
```

Wir können nano-editor verwenden, um docker-compose zu schreiben.

```
nano ./docker-registry-compose.yaml
```

Es wird geprüft, ob Java bereits installiert ist:

```
java -version
```

Wenn Java derzeit nicht installiert ist,

```
sudo apt install openjdk-11-jdk
java -version
javac -version
sudo update-alternatives --config java
```

Mit diesen Befehle kann kann Java von überall aus ausgeführt werden.

```
export JAVA_HOME=$"/usr/lib/jvm/java-11-openjdk-amd64"
export PATH=$PATH: $JAVA_HOME/bin
echo $JAVA_HOME
```

Gradle wird installiert.

```
wget https://services.gradle.org/distributions/gradle-$ {VERSION}-
    bin.zip -P /tmp
sudo unzip -d /opt/gradle /tmp/gradle-${VERSION}-bin.zip
sudo ln -s /opt/gradle/gradle-${VERSION} /opt/gradle/latest
sudo nano /etc/profile.d/gradle.sh
export GRADLE_HOME=/opt/gradle/latest
export PATH=${GRADLE_HOME}/bin:${PATH}
sudo chmod +x /etc/profile.d/gradle.sh
source /etc/profile.d/gradle.sh
gradle -v
```

A7 Dockerfile

```
FROM adoptopenjdk/openjdk11

WORKDIR /temporary

COPY build/libs/containertest-0.0.1-SNAPSHOT.jar.

EXPOSE 8080

CMD ["java","-jar", "./containertest-0.0.1-SNAPSHOT.jar"]
```

A8 Installation von Jenkins

```
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key |
    sudo apt-key add -


sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >
    /etc/apt/sources.list.d/jenkins.list'
sudo apt update
sudo apt install jenkins
```

Jenkins starten

```
sudo systemctl status jenkins
sudo systemctl start jenkins
```

Einstellungen für die Vernetzung des Jenkins-Servers in der Firewall.

```
sudo ufw allow 8080
sudo ufw allow OpenSSH
sudo ufw enable
sudo
ufw status
```

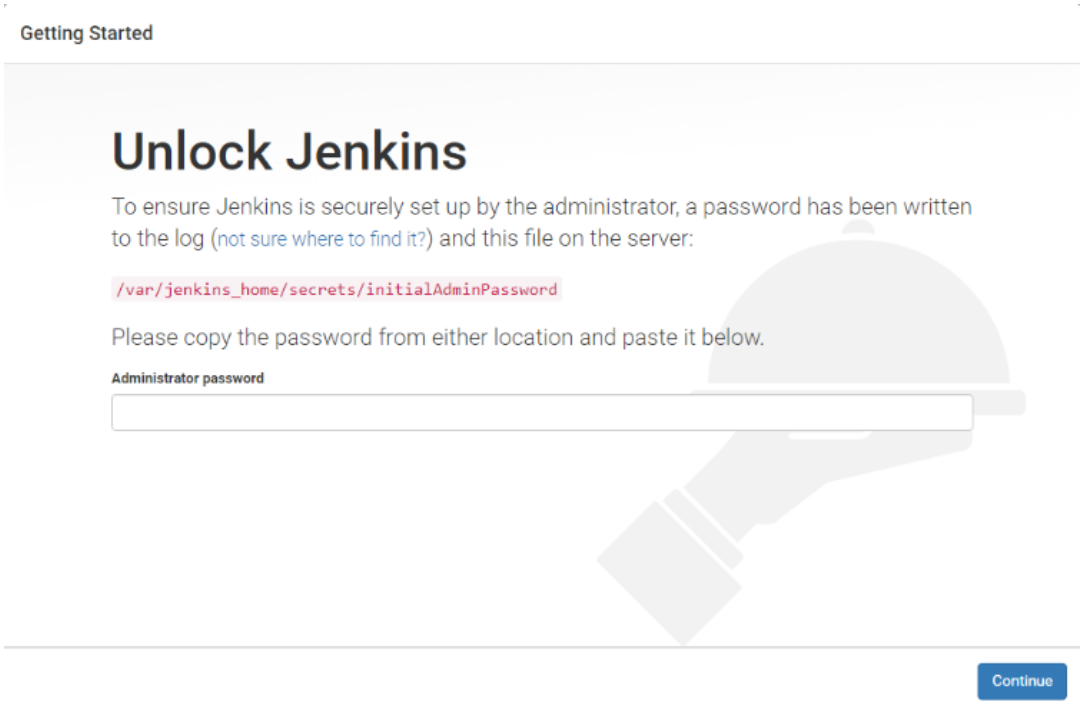

Abbildung 9: Unlock_Jenkins

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

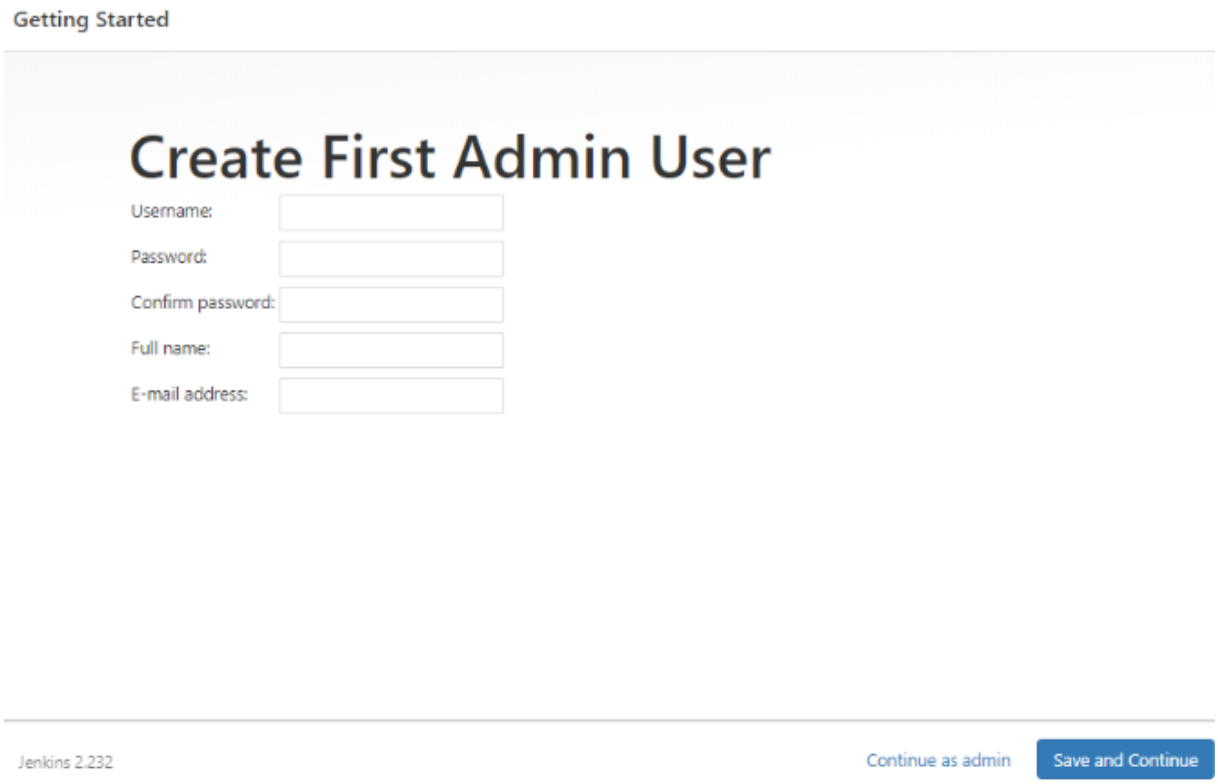

Abbildung 10: First Admin User

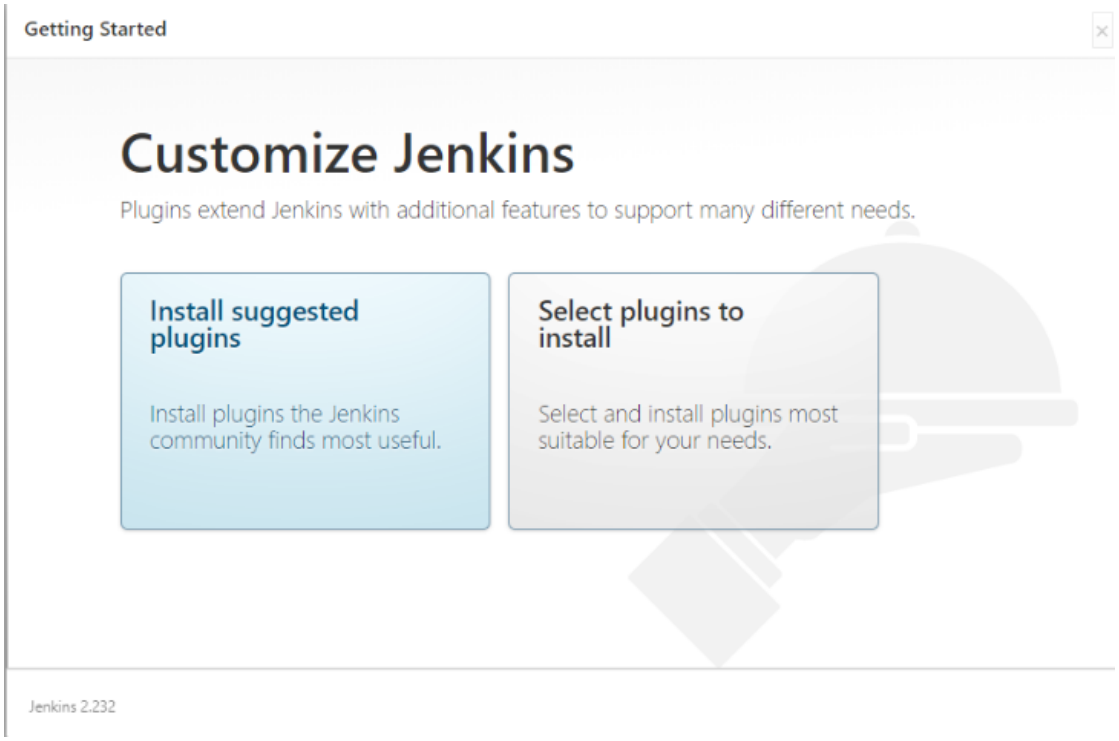

Abbildung 11: Costomize Jenkins#

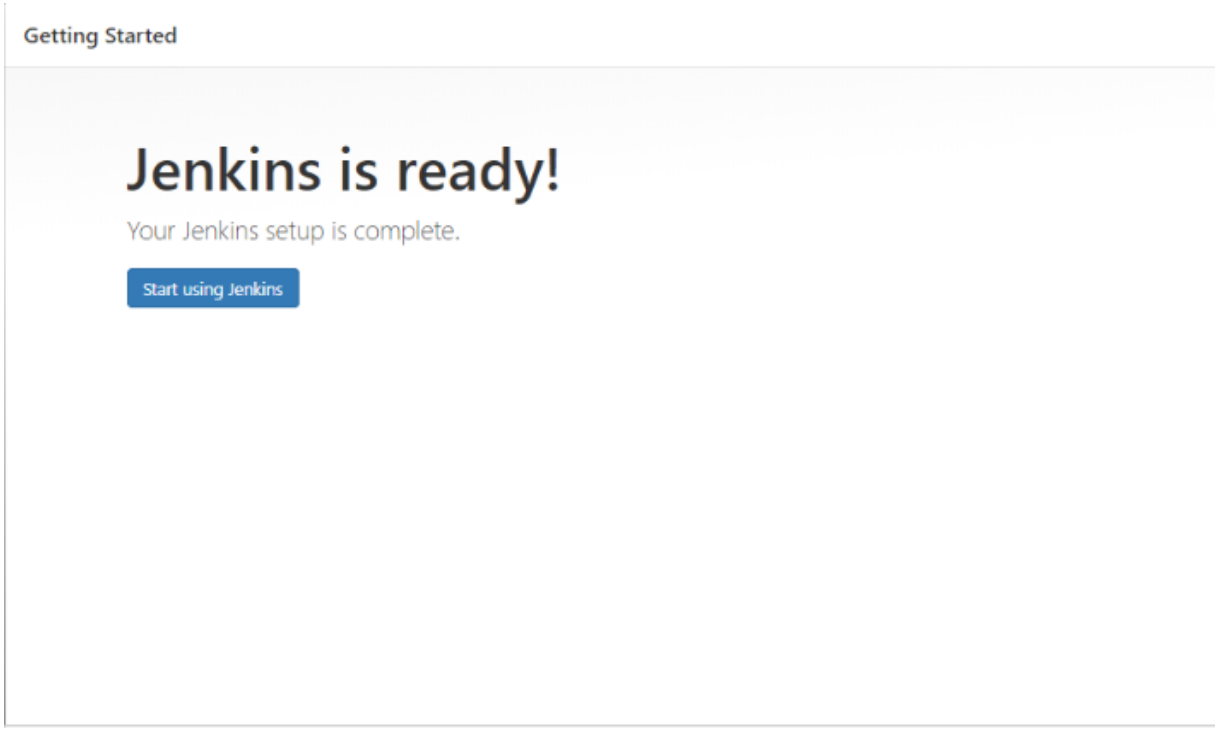

Abbildung 12: Jenkins Ready

Auf Wunsch kann der Jenkins-Server als Docker-Container installiert werden..

```
docker run --name myjenkins -p 8080:8080 -p 50000:50000 -v
    ~/$(USER):/var/jenkins_home jenkins
```

A9 Docker-Compose File

Im Home-Verzeichnis wird ein neues Verzeichnis definiert.

```
mkdir ~/docker-compose
cd ~/docker-compose
```

Die Datei „docker-compose“ wird mit dem Nano-Texteditor geschrieben.

```
nano ./docker-registry-compose.yaml
```

Hier ist die docker-compose-Datei.

```
version: '3'

services:
    docker-registry:
        container_name: docker-registry
        image: registry:2
        ports:
            - 5000:5000
        restart: always
        volumes:
            - ./volume:/var/lib/registry
    docker-registry-ui:
        container_name: docker-registry-ui
        image: konradkleine/docker-registry-frontend:v2
        ports:
            - 8080:80
        environment:
            ENV_DOCKER_REGISTRY_HOST: docker-registry
            ENV_DOCKER_REGISTRY_PORT: 5000
```

’’’’

Wir müssen diesen Befehl ausführen, um Docker-Registry zu starten.

```
docker-compose -f docker-compose.yaml up
```

A10 Installation von Minikube und kubectl

Da Minikube nicht im Ubuntu-Standard-Repository verfügbar ist, muss die Minikube-Binärdatei von der offiziellen Website heruntergeladen werden..

```
wget https://storage.googleapis.com/minikube/releases/latest/minikube-
    linux-amd64
```

Dann wird die Ausführungsberechtigung mit dem folgenden Befehl angegeben.

```
chmod +x minikube-linux-amd64
```

Danach wird die heruntergeladene Binärdatei in den Systempfad kopiert:

```
sudo mv minikube-linux-amd64 /usr/local/bin/minikube
```

Die Minikube-Version kann mit dem folgenden Befehl überprüft werden.

```
minikube version
```

Die neueste Version von kubectl wird mit dem Befehl heruntergeladen.

```
curl -LO https://dl.k8s.io/release/$(curl -L -s
    https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl
```

Die kubectl-Prüfsummendatei wird heruntergeladen.

```
curl -LO https://dl.k8s.io/$(curl -L -s
    https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sh
    a256
```

Mit diesen Befehlen kann validiert werden.

```
echo "$(cat kubectl.sha256)  kubectl" | sha256sum --check
```

Wenn ist es valid, kann kubectl installiert werden.

```
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

Überprüfung von kubectl-Konfigurationen

```
kubectl cluster-info
```

Endlich kann minikube gestartet werden.

```
minikube start
```

A11 deploymentfile.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: springapp-deployment
  labels:
    app: springapp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: springapp
  template:
    metadata:
      labels:
        app: springapp
    spec:
      containers:
      - name: springapp-container
        image: sennurmiray/snapshotintegration
        ports:
        - containerPort: 8080


---

apiVersion: v1
kind: Service
metadata:
  name: springapp-service
spec:
  selector:
    app: springapp
  type: NodePort
  ports:
    - nodePort: 30456
      targetPort: 8080
      port: 9898
```

A12 Global Tool Konfigurationen

Wir öffnen http://192.168.22.129:8080. Danach gehen wir zu Dashboard/Manage Jenkins/Global Tool Configurations..

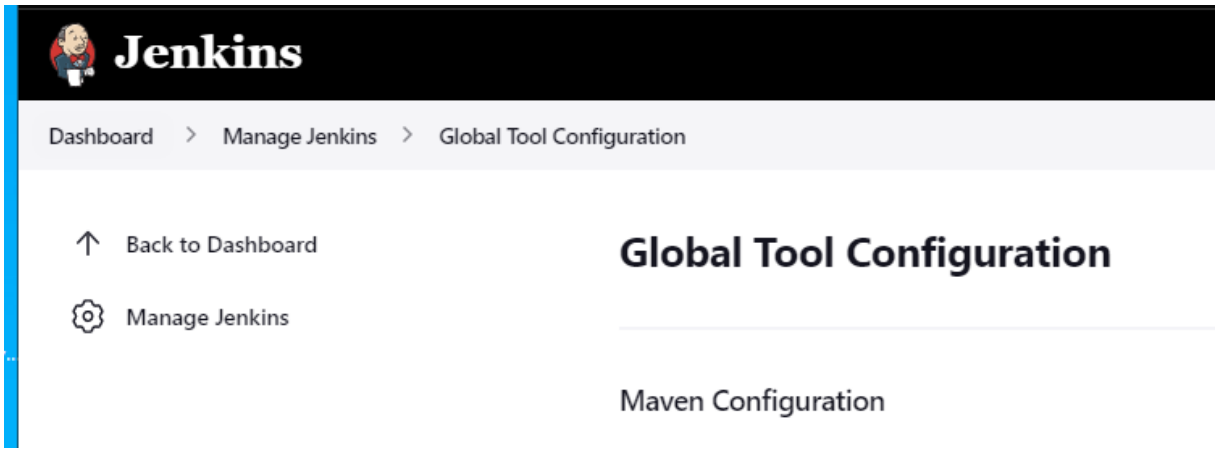

Abbildung 13: Global Tool Configuration

Die Path für git-Binärdatei muss eingegeben werden.

gehen wir auf Git-Feld, und wir geben ein:

Name: Git

Path_to_Git_executable:git

Wenn das Home/Binary-Verzeichnis unbekannt ist, kann es mit dem „which“-Befehl festgelegt werden.

```
which git
```

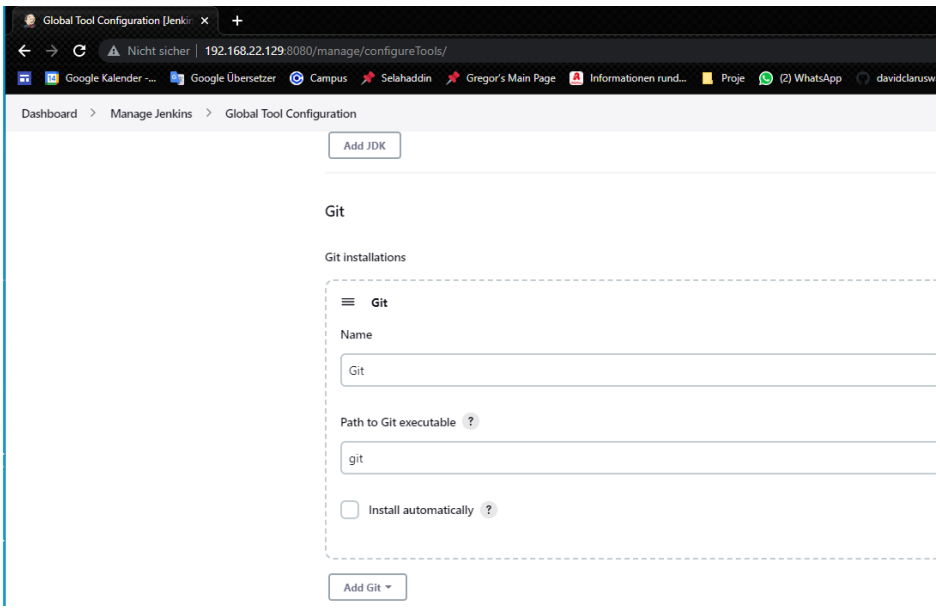

Abbildung 14: Konfigurieren von Git

Die Binary Directory von „docker“ muss spezifiziert werden.

gehen wir auf Docker-Feld, und wir geben ein :
Name: docker-1.41
Installation root: /usr/bin/docker

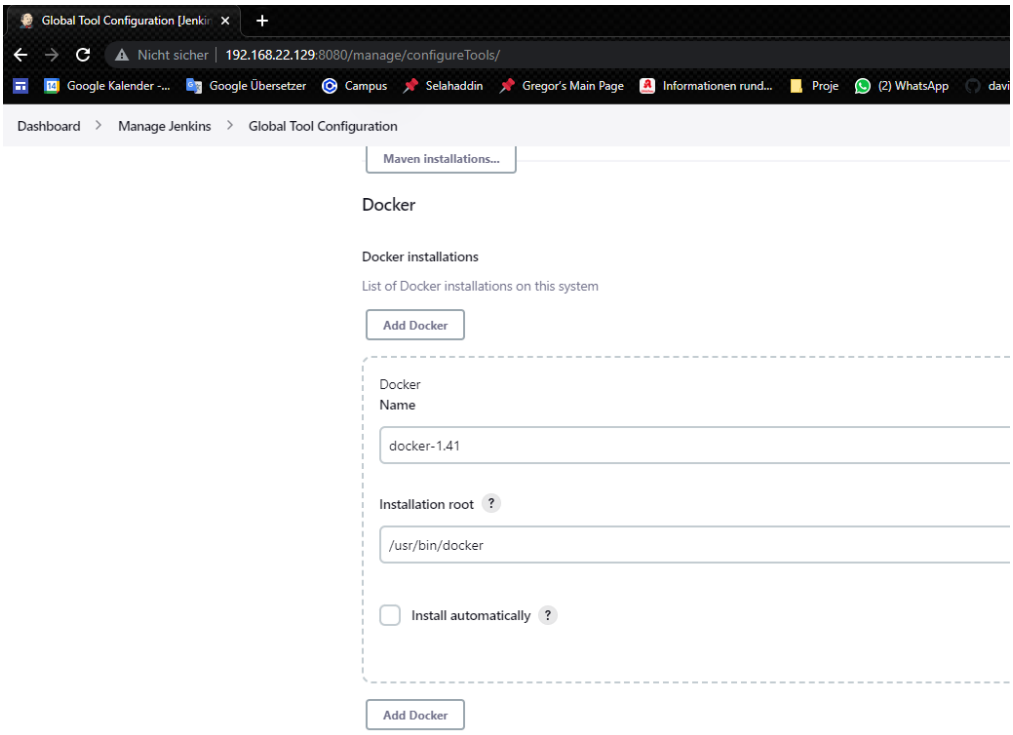

Abbildung 15: Konfigurieren von Docker

Die Home Directory von „OpenJDK-11“ muss spezifiziert werden.
gehen wir auf Java-Feld, und wir geben ein :
Name: java-11
JAVA_HOME: /usr/lib/jvm/java-11-openjdk-amd64

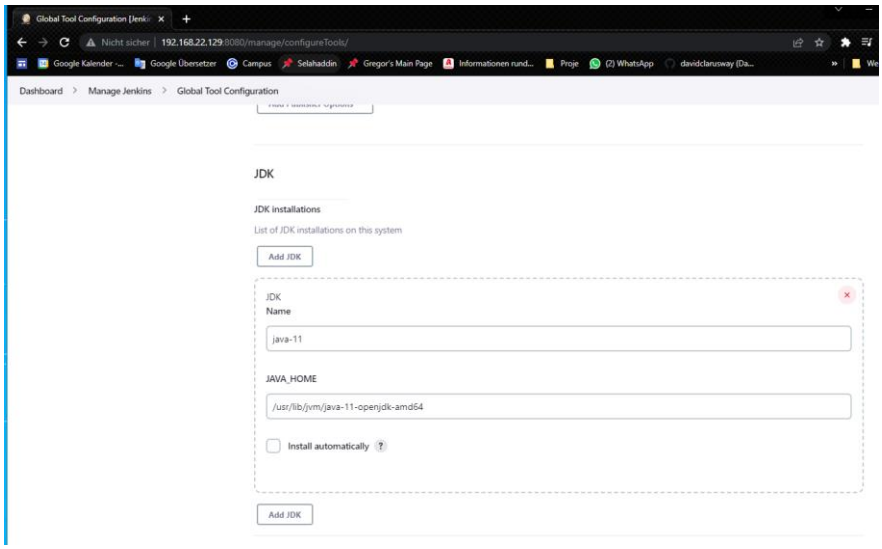

Abbildung 16: Konfigurieren von OpenJDK-11

Die Binary Directory von „Gradle“ muss spezifiziert werden.
gehen wir auf Gradle-Feld, und wir geben ein :
Name: gradle-7.4.2.
GRADLE_HOME: /opt/gradle/gradle-7.4.2

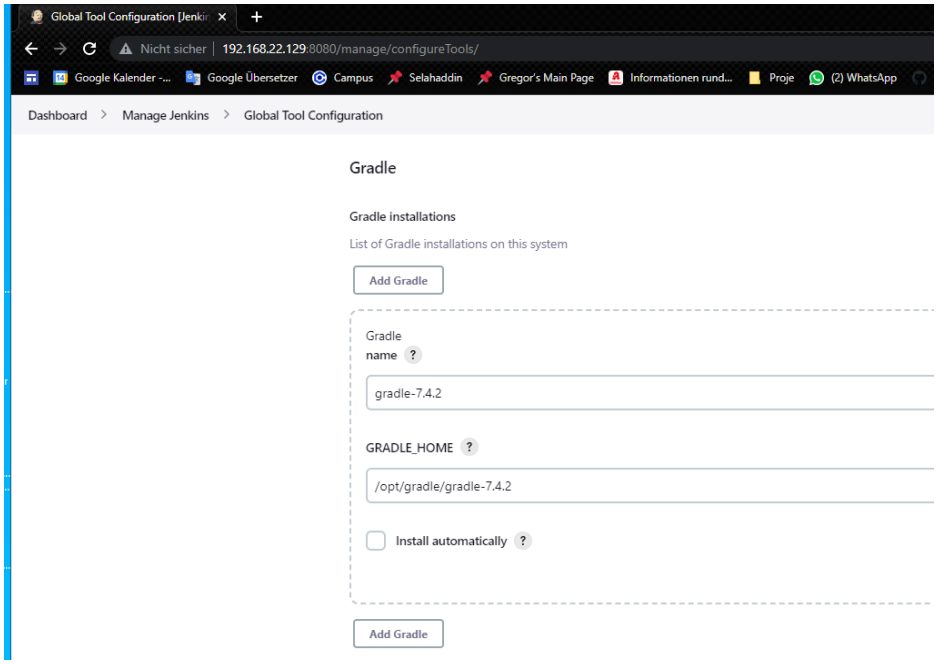

Abbildung 17: Konfigurieren von Gradle

A13 Konfigurieren von Docker-Registry Server und Delivery/Deployment Server als Agent Knoten von Jenkins

Schritte zum Konfigurieren der ssh-Verbindung zwischen Master -und Slave-Knoten:

- gehen zum Jenkins Master-Server.
- wechseln zum Benutzer "jenkins". (Jenkins funktioniert auf Agent-Konoten als Jenkins-User.)

```
sudo su - jenkins -s /bin/bash
```

- generieren mit „keygen“ einen öffentlichen und einen privaten Schlüssel.

```
ssh-keygen
```

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

- wählen alle Codes in id_rsa.pub aus und kopieren wir sie. Wir wechseln zum Ordner /root/.ssh auf der Slave-Knoteninstanz.
- Das folgende Befehlen sowohl Docker-Registry Server als auch Delivery-Server anzugeben.

```
sudo su
cd /root/.ssh
```

- 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. Syntax ist ssh root@<slave-node-ip-number>

```
ssh root@192.168.22.128
ssh root@192.168.22.134
```

Schritte zum Kopieren der Agent-Datei auf dem Agent-Knoten:

- 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.
- gehen zum Jenkins-Dashboard und klicken wir im linken Menü auf „Manage Jenkins“.
- wählen „Manage Nodes and Clouds“
- klicken im linken Menü auf „New Node“.

- geben „SlaveNode-1“ in das Feld „Node name“ ein und wählen wir „Permanent Agent“ aus.
- Klicken auf die Schaltfläche "OK"
- 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.
- Gib `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.

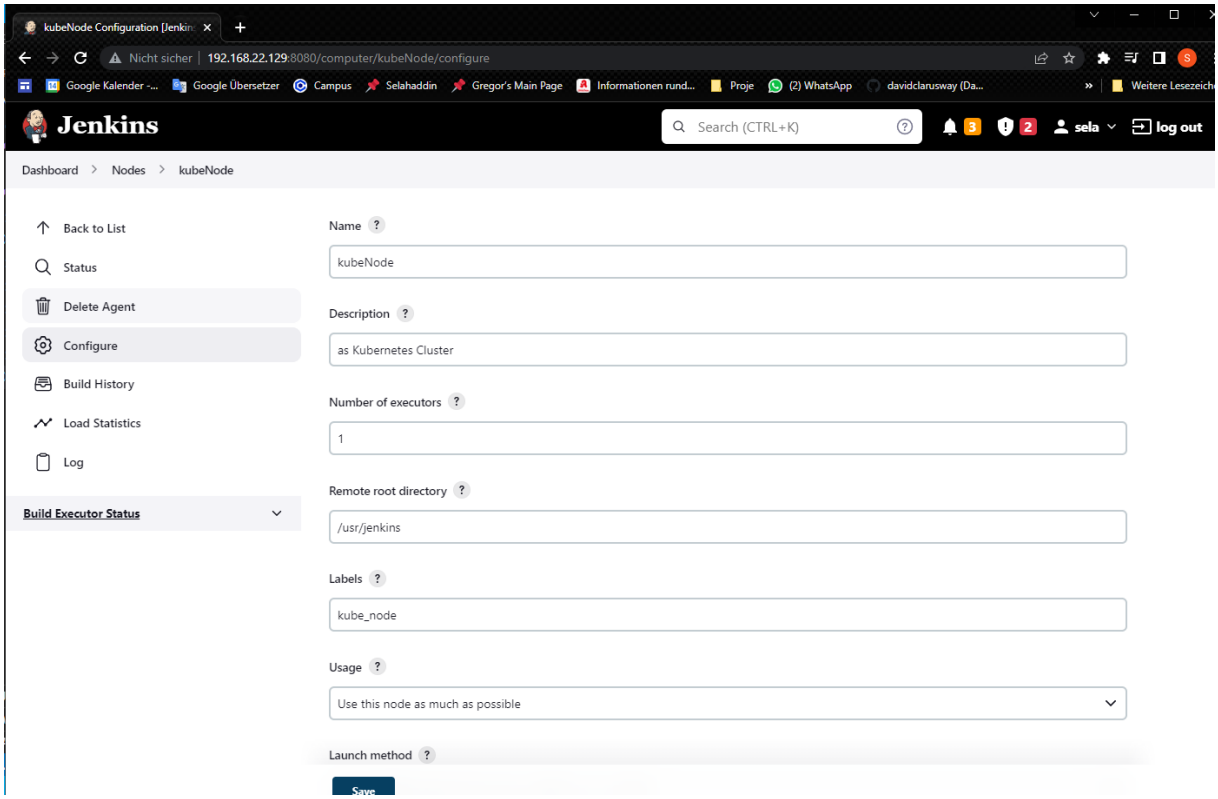

Abbildung 18: Agent-Node Konfigurieren

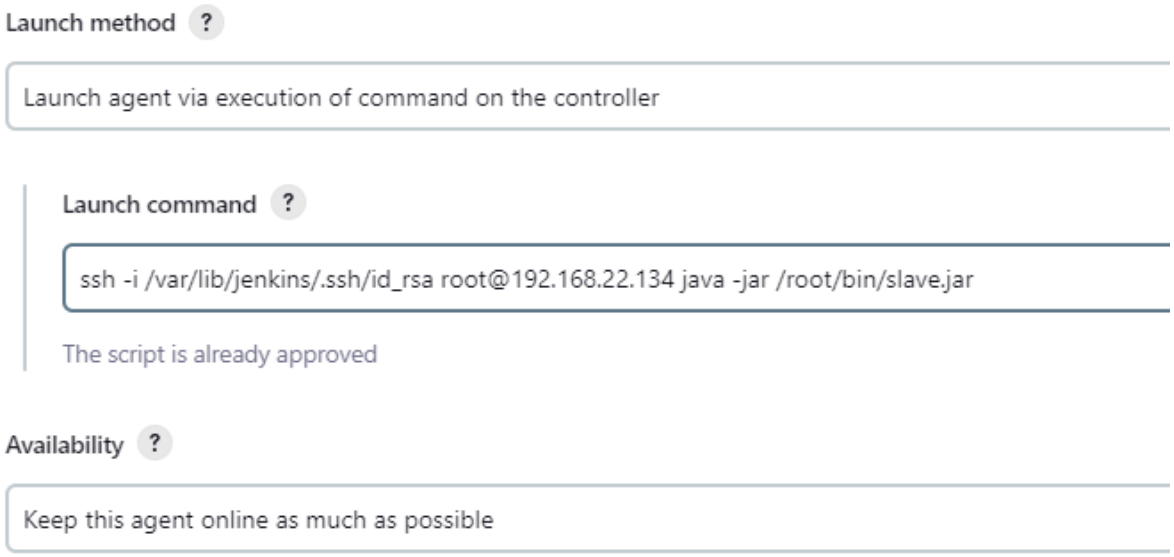

Abbildung 19: Launch Kommando

A14 Jenkinsfile

```
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"'
                }
            }

    }
}
```