

BACHELORTHESIS

Felix Schulze

Evaluation von Open- Source Werkzeugen für das automatisierte Deployment von Software

FAKULTÄT TECHNIK UND INFORMATIK

Department Informatik

Faculty of Computer Science and Engineering

Department Computer Science

Felix Schulze

Evaluation von Open-Source Werkzeugen für das automatisierte Deployment von Software

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Wirtschaftsinformatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Olaf Zukunft
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 23. August 2022

Felix Schulze

Thema der Arbeit

Evaluation von Open-Source Werkzeugen für das automatisierte Deployment von Software

Stichworte

Deployment, Microservices, Automatisierung

Kurzzusammenfassung

Diese Arbeit evaluiert eine große Auswahl an Open-Source Deployment-Werkzeugen auf das Finden eines optimalen Tools für einen Großteil der Unternehmen. Dafür wurden nach aufstellen von Metriken nach der Goal-Question-Metrics-Methode, eine konzeptionelle und eine experimentelle Evaluation durchgeführt.

Felix Schulze

Title of Thesis

Evaluation of open-source tools for automated software deployment

Keywords

Deployment, Microservices, Automation

Abstract

This thesis evaluates a wide range of open source deployment tools to find an optimal solution for most of the companies. For this purpose, a conceptual and experimental evaluation was carried out after establishing metrics using the Goal-Question-Metrics method.

Inhaltsverzeichnis

Abbildungsverzeichnis	vi
1 Einleitung	1
1.1 Motivation.....	1
1.2 Zielsetzung/Problemstellung.....	2
2 Grundlagen.....	3
2.1 Open-Source.....	3
2.1.1 Lizenzen	3
2.2 Microservices	4
2.3 Automatisiertes Deployment.....	7
2.3.1 Deployment	7
2.3.2 Automatisierung.....	9
2.3.3 Automatisiertes Deployment	9
2.4 Container.....	10
2.5 Evaluation	11
2.6 Metriken.....	13
2.6.1 Arten von Metriken.....	13
2.6.2 Aufstellen von Metriken.....	14
3 Metriken	15
3.1 Ziele.....	15
3.2 Fragen	17
4 Konzeptionelle Evaluation	19
4.1 Metriken.....	19
4.2 Deployment-Werkzeuge	22
4.2.1 AWS CodeDeploy	22

4.2.2	GitLab Deployment.....	23
4.2.3	Harness Drone (CI) & Harness CD.....	23
4.2.4	JFrog	23
4.2.5	Docker Compose.....	24
4.2.6	Juju.....	25
4.2.7	Rundeck.....	26
4.2.8	Spinnaker.....	27
4.2.9	AppVeyor	28
4.2.10	CircleCI.....	28
4.2.11	Jenkins.....	29
4.2.12	Buildbot.....	30
4.2.13	GoCD	30
4.2.14	TeamCity (JetBrains)	31
4.2.15	ArgoCD	32
4.2.16	Codefresh (codefresh.io).....	32
4.2.17	PHP Deployer.....	33
4.2.18	Capistrano.....	34
4.2.19	Rancher	34
5	Experimentelle Evaluation.....	35
5.1	Metriken.....	35
5.2	Softwareprojekte (basierend auf einer Microservice-Architektur).....	39
5.3	Deployment-Werkzeuge	40
5.3.1	Juju.....	40
5.3.2	ArgoCD	41
5.3.3	Rancher	42
5.3.4	Jenkins.....	43
5.3.5	GoCD	45
6	Bewertung.....	45
7	Fazit und Ausblick	46
7.1	Ausblick	47
	Literaturverzeichnis.....	49

Abbildungsverzeichnis

Abbildung 1: Microservice-Architektur vs. Monolith (Schneider, 2020).....	5
Abbildung 2: Deployment Lifecycle.....	7
Abbildung 3: Update-Prozess.....	9
Abbildung 4: Container vs. VM (Redaktion Open Telekom Cloud, 2020).....	11
Abbildung 5: Ablauf einer Evaluation (vgl. Balzer, et al., 1999).....	13
Abbildung 6: GQM-Methode (Basili, et al., 1994).....	15

1 Einleitung

1.1 Motivation

In den letzten Jahren hat sich eine neue Art der Softwarearchitektur durchgesetzt. Die sogenannte Microservice-Architektur vereinfacht zwar zum überwiegenden Teil die Softwareentwicklung, dennoch entsteht zusätzliche Komplexität durch die größere Anzahl an Services und die damit notwendige Kommunikation untereinander. Das Starten einer solchen Software ist eine widerkehrende Arbeit, die für den Entwickler eine arbeitsintensive und langweilige Tätigkeit darstellt. In Verbindung mit der erhöhten Komplexität, die durch eine Microservice-Architektur gegeben ist, steigt die Fehlerquote und der zeitliche Aufwand für den Entwickler. Diese Art von Arbeit kann durch automatisierte Prozesse dahingehend verändert werden, dass eine Steigerung der Produktivität und der Mitarbeiterzufriedenheit gelingt. Des Weiteren ist die Überwachung der verschiedenen Services besonders bei verteilten Systemen von großer Relevanz. Mehrere Systeme stellen eine erhöhte Gefahr für Ausfälle dar, da bei einem Softwareremololithen nur dieser überwacht werden muss und auf Funktionalität zu überprüfen ist. Aus mehreren Services resultiert ein erhöhter Aufwand, weil diese Arbeiten für jeden Funktionsbaustein einzeln durchgeführt werden müssen.

Die Reaktion auf dieses Problem sind Deployment-Werkzeuge, welche den Entwickler bei seinen wiederholenden Aufgaben unterstützen und zugleich eine neue Möglichkeit der Verwaltung bieten. Ziel dieser Tools ist das Schaffen einer neuen Lösung, die trotz weniger Arbeitsaufwand, veraltete Prozesse oder widerkehrende Arbeitsschritte beschleunigt. Deployment-Werkzeuge können den gesamten Prozess des Deployment-Lifecycles abdecken und so den Entwickler bei seinen täglichen Tätigkeiten unterstützen.

1.2 Zielsetzung/Problemstellung

Diese Bachelorarbeit basiert auf einem Problem vor das viele Unternehmen vor der Einführung einer Automatisierungssoftware für Deployments gestellt sind. Die Auswahl an Softwarelösungen variiert von Jahr zu Jahr, da ständige Weiter- und Neuentwicklungen die große Zahl bereits existierender Software ständig verändern. Eine Evaluation der vielen möglichen Tools ist kosten- und zeitintensiv, trotzdem soll am Ende das bestmögliche Produkt ausgewählt und produktiv im Unternehmen eingesetzt werden. Daraus resultiert, dass oft die bekanntesten Softwarelösungen eingesetzt werden. Meist werden die Möglichkeiten eines Unternehmens hier nicht ausgeschöpft, sodass unbekanntere Lösungen keine große Beachtung finden.

Im Rahmen dieser Bachelorarbeit sollen verschiedene Open-Source Deployment-Werkzeuge evaluiert werden. Es wird sich lediglich auf Open-Source Lösungen beschränkt, da bereits hier eine Vielzahl an möglichen Softwarelösungen zur Auswahl stehen und diese den meisten unternehmerischen Anforderungen genügen. Das primäre Ziel der Arbeit besteht in dem Herausfiltern eines optimalen Open-Source Deployment-Werkzeugs aus der gesamten Auswahl der auf dem Softwaremarkt existierenden Tools, welches für möglichst viele Anwender den besten Kompromiss darstellt. Zunächst wird hierfür eine konzeptionelle Evaluation durchgeführt, in der die Auswahl der Softwarelösungen so verkleinert wird, dass die verbleibenden Werkzeuge in der anschließenden experimentellen Evaluation genauer untersucht werden können. In diesem praktischen Teil wird anhand von Beispiel-Code ein Deployment und die mögliche Verwaltung der Testsoftware im Betrieb untersucht. Beide Evaluationen werden anhand von zuvor aufgestellten Metriken durchgeführt, welche für Unternehmen die größtmögliche Relevanz darstellen.

2 Grundlagen

In diesem Kapitel werden alle für diese Bachelorarbeit relevanten Grundlagen und Grundbegrifflichkeiten geklärt. Sie sind relevant für die anschließenden Evaluationen.

2.1 Open-Source

Essenziell für die Evaluation von Open-Source Werkzeugen ist die Klärung des Begriffs „Open-Source“. Der Quellcode kann hier öffentlich eingesehen werden und von anderen genutzt werden. Oft kann die Software komplett kostenfrei verwendet werden, allerdings können auch nur Teile kostenfrei sein und Zusatzfunktionen kostenpflichtig. (Perens, 1999)

2.1.1 Lizenzen

Besonders im Bereich von Open-Source Projekten sind Lizenzen von wichtiger Bedeutung. Sie sind eine Erlaubnis, ob und für welchen Zweck Software genutzt werden darf. (Jaeger & Metzger, 2006)

Folgende Lizenzen sind in dieser Evaluation von Relevanz:

- Die MIT License gibt kaum Einschränkungen in Verwendung der Software. Mit dieser Lizenz lizenzierte Software darf frei, auch für kommerzielle Nutzung, verwendet werden. Lediglich der Lizenztext und der Urheberrechtsvermerk müssen übernommen und weiter verteilt werden. (vgl. MIT)
- Die Apache License 2.0 ist die am Wenigsten populärste Softwarelizenz. Für sie gelten dieselben Bedingungen wie für die MIT License, jedoch weist sie minimale Mehrbeschränkungen in der Nutzung und Weiterverbreitung der lizenzierten Software auf. (vgl. Apache, 2004)
- Für die GNU General Public License gelten dieselben Einschränkungen, die auch für die Apache License 2.0 gelten. Auch mit dieser Lizenz lizenzierte Software darf für die kommerzielle Nutzung verwendet werden, allerdings gelten im Vergleich zu den

anderen aufgezählten Lizenzen die größten Einschränkungen. (vgl. Picot & Fiedler, 2008)

- Die GNU Affero General Public License basiert auf der GNU General Public License mit der Ausnahme, dass modifizierter Code, der auf Webservern läuft, zum Download bereitgestellt werden muss. (vgl. GNU, 2022)

2.2 Microservices

Von besonderer Bedeutung für die Experimentelle Evaluation sind Microservices. Sie bilden stellen die Architektur der ausgewählten Softwareprojekte in dieser Bachelorarbeit dar und werden daher kurz erläutert.

Microservices, bzw. die Microservice-Architektur dienen der Modularisierung einer Software. Aufgrund von immer komplexer werdender Software wurde ein neuer Ansatz in der Softwareentwicklung benötigt, um Software leichter verändern zu können und die Wartbarkeit sicherzustellen. Microservices bieten einen Ansatz, diese Probleme zu lösen. (vgl. Newman, 2015)

Microservices sind eigenständige Softwarekomponenten, in welcher jeder für sich nur eine Aufgabe im gesamten Softwareprojekt erfüllen soll. Zusammen mit anderen Microservices bilden sie so eine gesamte Softwarelösung, indem sie über Schnittstellen miteinander kommunizieren. (vgl. Newman, 2015)

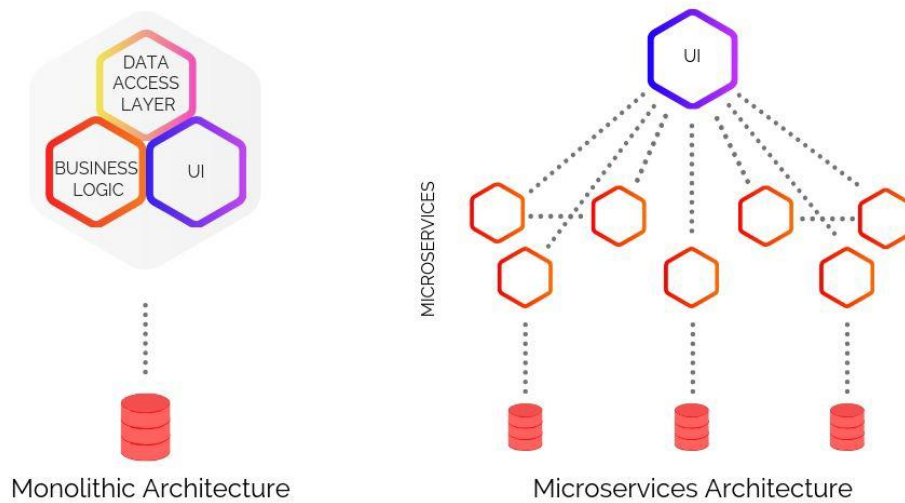


Abbildung 1: Microservice-Architektur vs. Monolith (Schneider, 2020)

In Abbildung 1 wird zur Verdeutlichung eine monolithische Architektur und eine Microservice-Architektur gegenübergestellt. Beide Architektur-Konzepte haben ein User-Interface und eine Datenhaltungsschicht. Allerdings fällt hier bereits auf, dass der Monolith auf weniger Datenbanken als die Microservice-Architektur zugreift. Dieser Unterschied hat Auswirkungen auf den sogenannten Data Access Layer, die Datenzugriffsschicht. Durch die Verwendung mehrerer Datenbanken werden für die Microservice-Architektur auch mehrere Services benötigt, die auf diese Datenbanken zugreifen. Diese Services sind in der Abbildung durch Verbindungen zu den Datenbanken gekennzeichnet. Im Gegensatz zu dem Softwaremonolith, bei welchem die Business Logik lediglich aus einem großen Service besteht, stellen die verbleibenden Sechsecke der Microservice-Architektur die Business Logik dar. Die Verbindungen untereinander, die sogenannten Schnittstellen, stellen die Kommunikation zwischen den Microservices sicher. So liegt der Hauptunterschied in der Architektur in der Aufteilung der Business Logik in mehrere kleinere Services und der Unterschied in der Datenhaltung, bzw. des Datenzugriffs.

Die Vorteile einer solchen Microservice-Architektur werden besonders bei großen Softwareprojekten deutlich. Große Services sind nur schwer änderbar, da diese sehr komplex sind und so viel Zeit in deren Einarbeitung und Verständnis investiert werden muss. Daraus resultiert, dass jede Einarbeitung viel Arbeit bedarf. Microservices können hier Abhilfe schaffen und eine

gute Wartbarkeit sicherstellen. Ein kleinerer Softwarebaustein kann bedeutend schneller und im Kompletten verstanden werden. Außerdem bieten Sie die Möglichkeit, Funktionalität unabhängig von dem großen Rest der Software zu implementieren. So kann ein Microservice einen Anderen im Laufe neuerer Versionen oder einer Weiterentwicklung ersetzen. Hiermit wird das Problem umgangen, dass der Code veraltet ist, da dieser bei Bedarf relativ einfach ersetzt werden kann. Die Entwicklung kann unabhängig von der restlichen Software erfolgen und muss anschließend nur eingebunden, beziehungsweise ausgetauscht werden. Auch benötigte Anpassungen können nach diesem Schema leichter durchgeführt werden, denn es wird nicht die gesamte Software, sondern lediglich der benötigte Microservice betrachtet. Des Weiteren bringt dies den Vorteil mit, dass bei Veränderung der einzelnen Microservices, stets die Architektur bestehen bleibt, was bei einer Veränderung an einem Softwaremonolithen nicht unbedingt gegeben ist. Weiterführend, sind Microservices gut skalierbar. An eine Software werden die verschiedensten Ansprüche gestellt, da kein Anwendungszweck identisch ist. Aus diesem Grund ist es von Vorteil, dass jeder Microservice für sich individuell skaliert werden kann. Sollte ein Service mehr Ressourcen benötigen, so wird dieser dementsprechend angepasst. (vgl. Wolff, 2018)

Allerdings gibt es auch Nachteile der Microservice-Architektur. Besonders der komplexe Aufbau birgt Risiken, die das Testen einer solchen Struktur aufwendiger machen als bei einem gewöhnlichen Monolith, da jeder Service zunächst für sich und außerdem in der Gesamtheit des Systems getestet werden muss. Auch ist die Ausfallwahrscheinlichkeit einzelner Services deutlich höher, da die Ressourcen auf mehrere Funktionsbausteine aufgeteilt werden. Trotzdem sind die Auswirkungen hier in der Regel nicht so schlimm wie bei dem Ausfall eines gesamten Softwaremonolithen, da Microservices gegen den Absturz anderer Services abgesichert sind. Über ein Überwachungstool können Ausfälle festgestellt werden. Somit steigen auch die Anforderungen an solch ein geeignetes Monitoring im Vergleich zu einem Softwaremonolithen. Außerdem ist dies noch bedeutend umfangreicher, da viel mehr Systeme überwacht werden müssen. Doch auch Netzwerk-Latenzen sind bei verteilten Systemen, welche über das Netzwerk in Kontakt stehen von Bedeutung. So stellt eine gute Verbindung sicher, dass die Kommunikation unter den Services verzögerungsfrei funktionieren kann. Störungen in der Netzwerkverbindung hingegen, können die gesamte Schnittstellenkommunikation lahmlegen. Ab-

schließlich kann es auch Probleme im Logging geben. Viele Systeme sind involviert, was entweder viele einzelne, oder ein erst zu entwickelndes, zentrales Logging zur Folge hat, welches mit den Logs eines Softwaremonolithen zu vergleichen ist. (Zhou, et al., 2019)

Aufgrund der Komplexität einer Microservice-Architektur und der Herausforderungen, beziehungsweise der Risiken, die eine solche Struktur mit sich bringt, eignen sich diese gut für den Vergleich von Automatisierungs-Werkzeugen. Eine komplexere Struktur stellt eine größere Herausforderung für ein Automatisierungs-Werkzeug dar, als eine weniger komplexe Struktur wie die, eines einzelnen Softwaremonolithen.

2.3 Automatisiertes Deployment

Der Begriff des automatisierten Deployments lässt sich aufspalten in die Begriffe Automatisierung und Deployment. Im Folgenden werden beide Wortbestandteile zunächst differenziert betrachtet und anschließend wieder zusammengeführt.

2.3.1 Deployment

Das sogenannte Deployment basiert auf einem „Deployment Lifecycle“, bestehend aus mehreren Schritten, welche in der Gesamtheit den Begriff des Deployments bilden. Er wird im Wesentlichen aus dem „Release“, welcher dem Ende der Entwicklungsphase einer Software entspricht, der „Installation“ und der „Activation“ gebildet. Außerdem zählen auch die „Deactivation“, die „Uninstallation“, das „Update“, die „Reconfiguration“, sowie das „Redeploying“ nach der Installation zu dem Deployment. (vgl. Dearle, 2007)

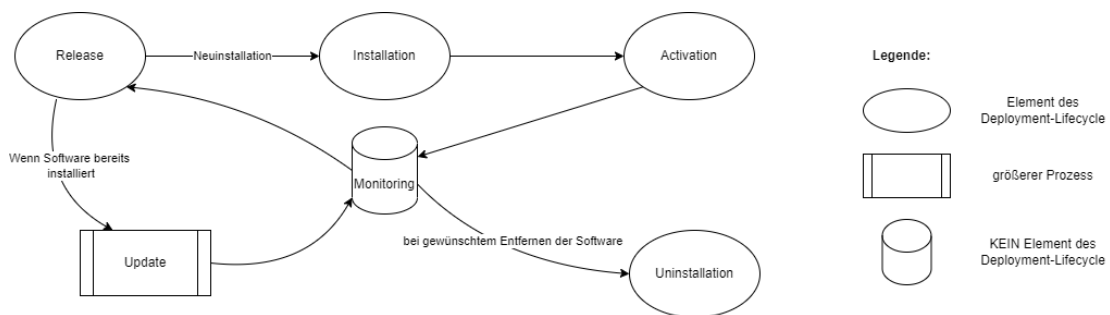


Abbildung 2: Deployment Lifecycle

Der Deployment Lifecycle ist in Abbildung 2 dargestellt. Der Deployment Lifecycle beginnt, wie auf der Abbildung zu sehen, stets mit einem Release. Der Release einer Software findet statt, sobald die Entwickler eine neue Version oder die Erstversion ihrer Software veröffentlichen. Anschließend folgt, wenn die Anwendung noch nicht installiert ist, die Installation auf der Test- oder der Produktivumgebung mit kundenspezifischen Konfigurationen. Durch Konfigurationen besteht die Möglichkeit die Software auf viele Kunden individuell anzupassen und den unterschiedlichsten Anwendungszwecken gerecht zu werden. Die nachfolgende Activation des Anwendungsprogramms beschreibt die Inbetriebnahme, bzw. den initialen Start des nun installierten Programms. Nach der Inbetriebnahme der Software wird während des Betriebes Monitoring betrieben. Das Monitoring, also die Überwachung des Service, stellt keinen Prozess des Deployment Lifecycles dar und wird daher nicht näher betrachtet. Wenn im Laufe der Zeit neue Versionen der Software veröffentlicht werden und der Kunde sich entschließt diese zu installieren, muss die Version geupdated und nicht neu installiert werden, da er die Anwendung bereits auf einer älteren Version betreibt. (vgl. Dearle, 2007)

Je nach Update kann es, so Bauwens (2020), bei laufendem Betrieb, in deaktiviertem Zustand oder automatisch im Hintergrund erfolgen. So ist es hier möglich, dass für ein Update (Over-the-Air-Update) keine anderen oder aber sogar mehrere andere Deployment-Prozesse gestartet werden müssen. Diese Update-Prozesse sind in der folgenden Abbildung 3 dargestellt. Die einfachste Möglichkeit des Over-the-Air-Updates benötigt keine besonderen Zwischenschritte. Das Update erfolgt in der Regel ohne, dass der Benutzer es bemerkt, im Hintergrund. Des Weiteren gibt es Updates, bei denen die Software zunächst deaktiviert werden muss. In Abbildung 3 beschreibt dies der mittlere Pfad. Nach dem Update ist es möglich die Software wieder zu aktivieren und somit in den Betrieb zu nehmen. Die dritte Möglichkeit stellt den umständlichsten Update-Prozess dar. Der dritte und letzte Pfad in der Grafik beschreibt diesen Prozess. Zunächst ist wie bei dem zweiten Pfad die „Deactivation“ der Software durchzuführen. Anschließend kann ebenfalls das Update durchgeführt werden. Allerdings ist nach dem Update eventuell eine Reconfiguration der Software notwendig, um bereits getroffene Einstellungen wiederherzustellen oder neue Softwarekonfigurationen vorzunehmen. Im Anschluss an die Reconfiguration muss dann ein Redeployment durchgeführt werden. (vgl. Dearle, 2007)

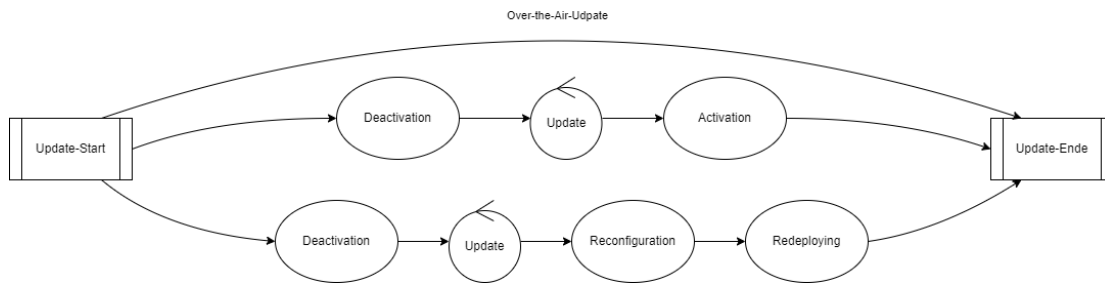


Abbildung 3: Update-Prozess

2.3.2 Automatisierung

Unter dem Begriff der Automatisierung versteht man einen automatisch ablaufenden Prozess, in welchen nach einem Trigger, welcher als Startpunkt gilt, nicht mehr eingegriffen werden muss. Ab diesem Zeitpunkt läuft eine vorher manuell festgelegte Abfolge von Tätigkeiten automatisch ab. (vgl. Heinrich, et al., 2020)

2.3.3 Automatisiertes Deployment

Nach Klärung beider Begrifflichkeiten, müssen diese noch zusammengefügt werden. Automatisiertes Deployment beschreibt somit die automatisierte Abfolge von zuvor festgelegten Schritten, welche die Phasen des Deployment Lifecycle von Beginn bis zum Ende abdecken. Hierbei gibt es in dem automatisierten Deployment von Software zwei verschiedene Trigger. Zunächst der Trigger des initialen Deployments, welcher für das einmalige Deployment beim Kunden oder Testsystem ausgeführt wird. Anschließend befindet sich die Software auf dem entsprechenden Zielsystem und es müssen lediglich Updates und eventuelle Reconfigurations durchgeführt werden. Diese werden durch einen weiteren, zweiten Trigger ausgeführt. Dieser kann entweder manuell oder automatisch, nach einem Release, angetriggert werden. (vgl. Humble & Farley, 2010)

2.4 Container

Die Art der Softwareentwicklung hat sich durch die Verwendung von Containern stark verändert. Insbesondere die Microservice-Architektur hat ein Umdenken erfordert, was die Ressourcenplanung und die vermehrten Deployments betrifft. Container bieten hier eine schlanke Lösung den neuen Anforderungen gerecht werden zu können. (vgl. Mouat, 2015)

Die Funktionsweise von Containern ähnelt der einer Virtual Machine (VM), hat aber entscheidende Vorteile. Die benötigte Container-Engine läuft auf dem Betriebssystem des Hosts. Anders als bei einer VM, benötigen die Container kein Betriebssystem, denn sie verwenden die Ressourcen des Host-Systems. Außerdem können die benötigten Bibliotheken, welche von der Container-Engine verwaltet werden, container-übergreifend verwendet werden – nicht wie bei einer VM. Hier gibt es eine Bibliothek pro virtueller Maschine. Dadurch sind Container leichtgewichtiger als VMs, wodurch sich die Build-Zeit von Containern gegenüber VMs verkürzt. Auf Dauer spart dies Aufwände und somit Kosten ein. Des Weiteren kann hierdurch eine größere Anzahl an Containern auf einem System laufen gelassen werden, was erneut eine höhere Produktivität und Kostenersparnisse zur Folge hat. (vgl. Mouat, 2015)

Durch die Abkapselung in Containern ist die Anwendung auf jedem System ausführbar. Des Weiteren bietet dies den Vorteil, dass die Anwendungen so leicht portierbar sind und sich (in den Containern) schnell und unabhängig vom Host-System starten lassen. (vgl. Kugele, et al., 2018)

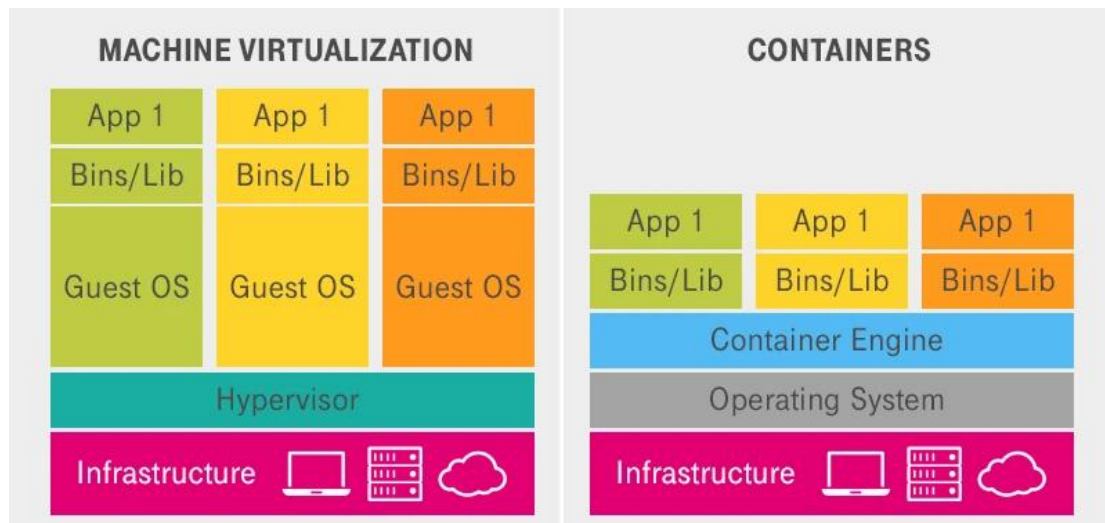


Abbildung 4: Container vs. VM (Redaktion Open Telekom Cloud, 2020)

Nachdem die Funktionsweise der Technologien geklärt ist, wird in Abbildung 4 der Unterschied zwischen dem Einsatz von Containertechnologien und virtuellen Maschinen verdeutlicht. Mehrere Services können unabhängig voneinander als Container auf nur einem System laufen. Im Vergleich dazu, werden trotz weniger laufender Services mehrere virtuelle Maschinen benötigt um die Applikationen in Betrieb zu halten. Des Weiteren ist jede virtuelle Maschine durch das benötigte Betriebssystem großgewichtiger als ein Container.

2.5 Evaluation

Eine Evaluation ist eine Vorgehensweise um Software miteinander zu vergleichen. In einer ähnlichen Form sagten es auch Wottawa und Thierau (1998), denn auch sie sahen „Evaluation (...) als Planungs- und Entscheidungshilfe (...)“. Auch für Sie ist eine „Evaluation (...) ziel- und zweckorientiert“. So besteht das Ziel darin, ein Software-Werkzeug zu finden, welches den unterschiedlichsten Ansprüchen der Kunden am ehesten gerecht wird. Diese hängen von dem Anwendungszweck des Deployment-Werkzeugs ab. Betreiber einer oder mehrerer großen Softwarelösungen haben andere Anforderungen und Ansprüche an ein Produkt als eine Privatperson, welche gelegentlich eine selbstgeschriebene Software in der eigenen Cloud deployen möchte. Auf diese Unterschiede muss in einer Evaluation eingegangen werden, um das richtige Werkzeug für möglichst viele Anwender zu evaluieren. (vgl. Kelter, 2006)

Hierbei folgt eine Evaluation einem festen Schema. In Abbildung 5 sind alle Abläufe, die Teil einer Evaluation sind, von oben nach unten aufgeführt. Im Folgenden wird dieser grundlegende Prozess anhand der Abbildung auf die Evaluation von Software-Werkzeugen für das Deployment von Software übertragen. (vgl. Balzer, et al., 1999)

Wie in der Grafik zu erkennen wird zunächst die Grundlage einer Evaluation benötigt. Sie beschreibt, wieso eine Untersuchung nötig ist und durchgeführt wird. Dies ist hier die Problemstellung der Kunden, welche sich eine neue Softwarelösung wünschen. Der nächste Schritt (hier dargestellt durch einen Folgepfeil) ist das Beschreiben der Zielsetzung. Diese ist das Herausfiltern von einem möglichst optimalen Deployment-Werkzeug für verschiedenste Anwendungszwecke. Die Durchführung, bzw. Planung einer Evaluation bildet einen weiteren fundamentalen Baustein zu einer Lösungsfindung und damit auch den Folgeschritt in dem Evaluationsprozess. In dieser Bachelorarbeit wird zunächst eine konzeptionelle Evaluation durchgeführt. Eine theoretische Untersuchung ist allerdings nicht ausreichend, da sich einige Metriken nur praktisch messen lassen. Aus diesem Grund muss zusätzlich eine experimentelle Evaluation durchgeführt werden. Diese bietet den Vorteil, dass das Versuchsobjekt nicht nur beobachtend oder anhand von Dokumentationen evaluiert, sondern außerdem praktisch vom Benutzer getestet wird. Des Weiteren besteht so die Möglichkeit, Benutzererfahrungen mit in die Evaluation einfließen zu lassen und erhöht somit die Aussagekraft vieler subjektiver Metriken. (vgl. Balzer, et al., 1999)

Der experimentelle Teil basiert auf einer praktischen Untersuchung mit Test (...). In dieser Bachelorarbeit wird das experimentelle Verfahren der Laborprüfung angewendet, bei dem es eine gewisse Anzahl an Probanden gibt, mit welchen die Untersuchung durchgeführt wird. Sie dienen lediglich der Unterstützung für die eigentliche Untersuchung. (vgl. Kühl, 2009)

In Bezug auf Software wird diese experimentelle Evaluation mit Softwareprojekten, basierend auf einer Microservice-Architektur, als Probanden/Testkandidaten durchgeführt. Die Deployment-Werkzeuge werden mit diesen Projekten getestet und daraufhin, anhand von aufgestellten Metriken, miteinander verglichen und bewertet. Anschließend erfolgt eine Auswertung anhand der daraus resultierenden Ergebnisse. Diese tabellenartige Auswertung soll Aufschluss darüber geben, welches Software-Werkzeug am besten geeignet ist. Zuletzt werden die ausgewerteten Daten bewertet und die weitergehende Nutzung diskutiert. Hierzu zählen unter anderem die

Anwendungszwecke und der gewonnene Mehrwert durch die Evaluationen in dieser Bachelorarbeit.

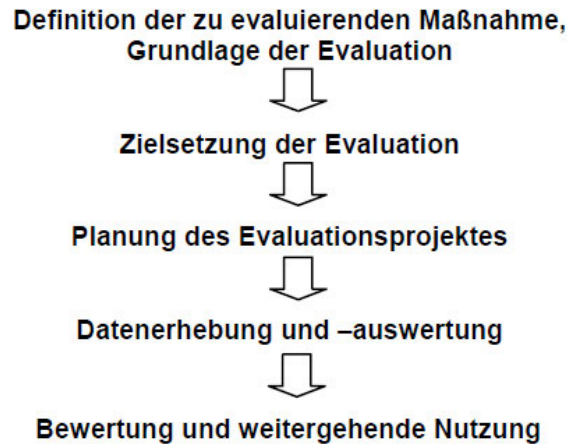


Abbildung 5: Ablauf einer Evaluation (vgl. Balzer, et al., 1999)

2.6 Metriken

Der Vergleich und die damit verbundenen Bewertungen jedes Deployment-Werkzeugs werden mit Metriken durchgeführt.

Metriken bilden in der Softwareentwicklung ein Grundgerüst für die Vergleich- und Bewertbarkeit von Software. Sie sind von Menschen aufgestellte Standards, welche Kriterien messbar machen sollen und bilden so eine gute Evaluationsgrundlage. Es kann zwischen verschiedenen Arten von Metriken unterschieden werden, welche im nächsten Absatz genauer betrachtet werden. Anschließend wird erklärt, wie Metriken aufgestellt werden können und wieso dies in der Regel nicht ohne eine festgelegte Vorgehensweise möglich ist. (vgl. Witte , 2018)

2.6.1 Arten von Metriken

Es kann zwischen drei verschiedenen Arten von Metriken unterschieden werden. In der Regel wird zwischen objektiven, subjektiven und Pseudometriken unterschieden. Die Art der Metrik hat Einfluss auf die Bewertung der Metrik.

Objektive Metriken basieren auf Messungen und Standards und können daher mit einfachen Messwerkzeugen ermittelt werden. In der Softwareentwicklung sind dies die zählbaren, also quantitativen Eigenschaften und können daher sehr gut miteinander verglichen werden. Eine bekannte Metrik ist hier zum Beispiel die Anzahl der Codezeilen (Lines of Code, LOC). (vgl. Witte , 2018)

Subjektive Metriken hingegen lassen sich von jedem Individuum selber festlegen und sind in der Bewertung häufig subjektiv. Besonders bei den subjektiven Metriken ist daher die Objektivität des Evaluierenden von besonderer Bedeutung, damit die Ergebnisse tatsächlich miteinander vergleichbar sind. Eine möglichst große Objektivität wird versucht durch Klassifikationen herzustellen. Ein Beispiel für eine subjektive Metrik ist die Benutzerfreundlichkeit einer GUI. Eine mögliche Klassifikation ist die Einteilung in „übersichtlich“ und „unübersichtlich“. (vgl. Specht, 2021)

Pseudometriken sind im Grunde Berechnungen auf Basis von Messungen/Beurteilungen und liefern relevante Aussagen über nicht gleich ersichtliches Material. Sie sind ohne einen entsprechenden Rechenweg meist nicht nachzuvollziehen und müssen daher oft weitergehend erläutert werden. Sie sind in dieser Bachelorarbeit von keiner Relevanz und werden daher nicht näher betrachtet. (vgl. Ludewig & Lichter, 2013)

2.6.2 Aufstellen von Metriken

Der Goal-Question-Metric-Approach (GQM) bietet hier einen systematischen Ansatz Metriken aufzustellen und deren Herleitung darzulegen. Im Wesentlichen besteht die GQM-Methode aus drei Bestandteilen. Sie besteht aus Goals, Questions und den zu erhebenden Metrics. (vgl. Pham & Schneider, 2013) (Diese Methode wurde für die theoretische Evaluation ausgewählt)

In der folgenden Grafik sind die Bestandteile und die Vorgehensweise zu erkennen. Die Schritte werden von oben nach unten abgearbeitet. Zunächst werden die zu erreichenden Ziele definiert. In diesem Fall werden die Ziele beschrieben, welche aus den Deployment-Werkzeugen hervorgehen. Anschließend werden charakteristische Fragen aus diesen Zielen abgeleitet, welche sich auftun, wenn die diese Ziele erreicht werden sollen. Wie aus der Grafik zu entnehmen, sind dies in der Regel mehr Fragen als zuvor definierte Ziele. Zuletzt werden aus diesen Fragen Metriken abgeleitet, welche dazu beitragen, die Antworten auf die zuvor aufgestellten

Fragen zu finden. Eine Metrik kann hier auch mehrere der zuvor aufgestellten Fragen beantworten. Diese aufgestellten Metriken werden anschließend für die konzeptionelle und die experimentelle Evaluation verwendet. (vgl. Basili, et al., 1994)

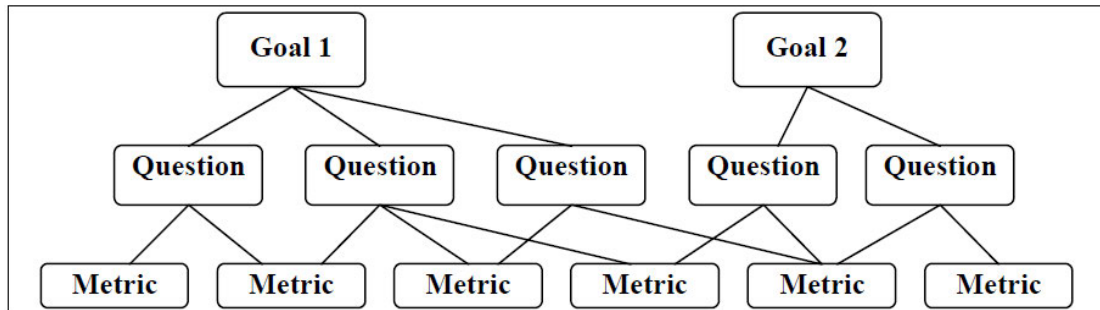


Abbildung 6: GQM-Methode (Basili, et al., 1994)

3 Metriken

Das Aufstellen der Metriken erfolgt nach der Goal-Question-Metrics-Methode. Nach Anwendung der Goal-Question-Metrics-Methode ergeben sich zunächst Ziele, welche ein Deployment-Werkzeug erreichen sollte. Aus diesen zu erreichenden Zielen werden nun Fragen abgeleitet, welche helfen, den Zielen näher zu kommen. Anschließend werden Metriken entwickelt, die Antworten auf die zuvor aufgestellten Fragen geben. Diese Metriken bilden die Grundlagen für die konzeptionelle und die experimentelle Evaluation.

3.1 Ziele

Ein Deployment-Werkzeug sollte einige Funktionen abdecken und erfüllen, welche anhand von Zielen definiert werden.

In dieser Evaluation wurden folgende Ziele für ein gutes Deployment-Werkzeug aufgestellt:

- **Fehler sollen leicht zu erkennen sein**, da diese während eines Deployments auftreten können. Die Software soll hier unterstützen und dem User den Grund des Fehlverhaltens aufzeigen.
- **Aktuellen Deployment-Status erkennen**, um dem User eventuelle Wartezeiten darzustellen oder mögliche Verbesserungen am Deployment einzelner Services aufzuzeigen.
- **Unit- und Integrationstests ausführbar**, die die Möglichkeiten bieten, dass der Code automatisiert getestet wird, ohne ein vollständiges Deployment mit auftretenden Fehlern durchführen zu müssen.
- **Unabhängig vom System sein**, weil eine Software für möglichst viele User zugänglich sein soll, ohne eine Beschränkung auf einzelne Betriebssysteme in Kauf nehmen zu müssen.
- **Hilfestellungen bei Problemen geben**, da es bei der Verwendung von Software stets zu Problemen kommen kann, die man selber nicht lösen kann.
- **Interaktion mit anderen Usern**, die einen Austausch und Kommunikation unter Personen mit einem gleichen Interesse oder Berufsfeld möglich machen.
- **Hilfen im Umgang mit dem Deployment Werkzeug geben**, die dem User das Herantasten an eine für ihn unbekannte Software vereinfachen sollen.
- **Weiterentwicklung der Software und daraus resultierende Anpassungen an neue Technologien**, weil ständig neue Technologien weiterentwickelt werden. Trotz dieser Veränderungen soll es weiterhin möglich sein, dass Deployment Werkzeug mit gewohntem Funktionsumfang nutzen zu können.
- **Keine versteckten Kosten, voller Funktionsumfang**, um sicherzustellen, dass keine Kosten für kommende oder bereits integrierte Funktionen getragen werden müssen.
- **Einsehen der aktuellen verwendeten und noch verfügbaren Ressourcen**, um nicht an die maximale Auslastung des Hostsystems zu kommen.
- **Anwendungen sollen sich ohne Veränderungen am Code deployen lassen**, damit der User nicht den Code des zu deployenden Softwareprojektes abwandeln muss, um es deployen zu können.
- **Es sind keine Vorinstallationen oder andere Infrastruktur notwendig**, um die Komplexität und den Aufwand der Installation möglichst gering zu halten.

- **Die Ressourcen der deployten Anwendung können verändert werden**, damit die Leistungsfähigkeit der Anwendungen nach den gegebenen Anforderungen angepasst werden kann.
- **Die Daten der Anwendungen können, bzw. werden gesichert**, sodass sie bei einem Neustart oder Absturz nicht verloren gehen.
- **Die Daten werden bei Beenden eines Service gespeichert.**
- **Das Deployment Werkzeug ist frei verwendbar und darf auch für gewerbliche Zwecke genutzt werden**, damit das Werkzeug auch für gewerbliche Nutzung genutzt werden kann, ohne mit Lizenzverstößen konfrontiert zu werden.
- **Das Deployment läuft zügig ab und es entstehen keine langen Wartezeiten.**
- **Eine intuitive Benutzeroberfläche**, welche die Benutzerfreundlichkeit und eine logische Bedienreihenfolge ermöglicht.
- **Keine Ausfälle**, welche aufgrund von Softwarefehlern entstanden sind.
- **Eine ständige Erreichbarkeit**, in Bezug auf Software-as-a-Service Anbietern. Die Oberfläche muss immer für den User erreichbar sein.
- **Das Deployment-Werkzeug darf keine Ladezeiten zwischen Benutzeranfragen zeigen**, sodass der User nicht das Gefühl bekommt, die Software sei träge.

3.2 Fragen

Die Fragen ergeben sich aus zuvor festgelegten Zielen. Sie sollen aussagen, ob diese Ziele erreicht wurden und bilden die Grundlage für Metriken. Nachfolgend werden Fragen zu den aufgestellten **Fehler! Verweisquelle konnte nicht gefunden werden.**n abgebildet:

- Können Fehler erkannt werden?
- Wieso treten Fehler auf?
- Tritt der Fehler aufgrund von einem Softwarefehler oder einem Anwendungsfehler auf?
- Wann ist das Deployment beendet? Wie lange ist die Wartezeit?
- Welcher Anwendungsschritt des Deployments wird aktuell ausgeführt?
- Welche Tests können über das Deployment-Werkzeug ausgeführt werden?
- Wo kann das Werkzeug installiert werden, bzw. wo kann es nicht installiert werden?

- Muss die Systemumgebung bestimmte Voraussetzungen erfüllen, die eine Installation möglich machen?
- Benötigt man eine Internetverbindung oder muss eine andere Vorabbedingung für eine erfolgreiche Installation erfüllt sein?
- Kann nach Hilfe bei Problemen geschaut werden?
- Haben andere Anwender bereits ähnliche Probleme gehabt?
- Gibt es andere Anwender und besteht eine Kontaktmöglichkeit für einen Austausch?
- Gibt es ein Troubleshooting oder ähnliche Hilfen bei Problemen?
- Finden weitere Entwicklungen an dem Deployment-Werkzeug statt, sodass auch die Verwendung von neuen Technologien in Zukunft möglich sein wird?
- Sind alle Funktionen uneingeschränkt verwendbar oder muss für zusätzliche Funktionalität gezahlt werden?
- Gibt es eine Übersicht über deployte und laufende Deployments und werden die dafür benötigten Ressourcen grafisch dargestellt?
- Treten Probleme in der Anwendung der Software auf oder ist die Verwendung einfach und selbsterklärend?
- Müssen für das Deployment Anpassungen am Code vorgenommen werden?
- Wird andere Software oder Infrastruktur vor der Installation des Werkzeugs benötigt?
- Können genutzte Ressourcen eingesehen und bei Bedarf erweitert werden?
- Können Daten gesichert werden?
- Werden automatische Daten-Backups generiert?
- Ist eine Ausfallsicherheit gegeben?
- Ist das Deployment-Werkzeug frei verwendbar oder gibt es Einschränkungen, wer das Werkzeug nutzen darf?
- Darf das Werkzeug für gewerbliche Zwecke (auch für Kundensysteme) genutzt werden?
- Wie lange dauert ein Deployment?
- Wie ist die Bedienbarkeit der Software? Ist eine intuitive Bedienung sichergestellt?
- Gibt es Ausfälle oder auftretende Fehler während des Betriebs des Deployment-Werkzeugs?

- Ist die Software immer verfügbar oder startet sie nicht immer?
- Ist die Software träge oder ist sie performant? Werden User Interaktion direkt umgesetzt oder kann es zu Wartezeiten kommen?

Aus diesen aufgestellten Fragestellungen können nun die Metriken für die konzeptionelle und die experimentelle Evaluation abgeleitet werden. Diese werden in der jeweiligen Evaluationsform vorgestellt.

4 Konzeptionelle Evaluation

Die konzeptionelle Evaluation ist die zuerst angewandte Form der Evaluation. Anhand von theoretischen Eigenschaften, die durch Recherche und ohne praktischen Aufwand erarbeitet werden, wird die nachfolgende Evaluation durchgeführt.

4.1 Metriken

Aus den Fragen, die mit Hilfe der GQM-Methode aufgestellt wurden, ergeben sich die Metriken für die Evaluation. Sie sollen die Fragen messbar beantworten und können in objektive, subjektive und Pseudometriken aufgeteilt werden.

Folgende Metriken werden in der konzeptionellen Evaluation behandelt, welche mit Hilfe der Goal-Question-Metrics-Methode aufgestellt wurden:

- Systemumgebung
- Support
- Community
- Dokumentation
- Andauernde Entwicklung
- Weitere Kosten

- Zuvor benötigte Infrastruktur
- Lizenzen

Systemumgebung

Auch bei der Metrik „Systemumgebung“ handelt es sich um eine objektive Metrik, da auch hier aufzählbar ist, welche Möglichkeit das Deployment-Werkzeug bietet, um auf einem System installiert zu werden. Hierbei geht es überwiegend um das Betriebssystem auf welchem die gesamte Software installiert werden kann, oder um den Client, falls das Werkzeug in der Cloud läuft und nur über einen entsprechenden User darauf zugegriffen wird. Über das Ziel der Systemunabhängigkeit folgen die Fragen, welche Systembedingungen erfüllt sein müssen, damit das Werkzeug erfolgreich installiert werden kann.

Support

Eine weitere essenzielle Metrik bildet der „Support“. Hierbei handelt es sich um eine Mischform aus einer objektiven und einer subjektiven Metrik. Das ursprüngliche Ziel, dass der User Hilfe bei Problemen in der Verwendung der Software bekommt, werden durch die Fragen, wo nach Hilfe geschaut werden kann und ob andere User bereits ähnliche Probleme hatten, beantwortet. Beide Fragen können zunächst objektiv bewertet werden, indem die Möglichkeiten aufgezählt und so für jeden nachvollziehbar dargelegt werden. Allerdings müssen die Unterschiede von einem Individuum bewertet werden, um festzustellen, wo die Vor- oder Nachteile des Supports bei den unterschiedlichen Werkzeugen liegen.

Community

Die Metrik „Community“ ähnelt der Metrik des Supports. Auch hier liegt eine Mischform der Art der Metrik vor, da zunächst aufgezeigt werden kann, ob und wo eine Interaktion mit anderen Usern des Deployment-Werkzeugs stattfinden kann. Allerdings liegen die Bewertung und die Abwägung dieser aufgezeigten Möglichkeiten bei einer dritten Person und ist somit subjektiv.

Dokumentation

Eine weitere resultierende Metrik ist die, der „Dokumentation“. Eine Dokumentation einer Software zeigt dem User auf, wie die Software funktioniert und wie sie anzuwenden ist, beziehungsweise was für den Betrieb der Anwendung essenziell ist. Außerdem hat der User in der Dokumentation die Möglichkeit nachzuschlagen, falls Probleme in der Anwendung der Software bestehen. Daher soll die Metrik auch die Frage beantworten, ob die Dokumentation ausreichend ist um sich bei Problemen (unter Umständen auch mit Troubleshootings oder ähnlichem) selber helfen zu können. Diese Bewertung erfolgt durch eine Dritte Person und hängt von ihrer Bewertung ab, weshalb es sich hierbei erneut um eine subjektive Metrik handelt.

Andauernde Entwicklung

Die „Andauernde Entwicklung“ ist eine Metrik, die die Frage beantwortet, ob sich das Deployment-Werkzeug weiterentwickelt und auch für zukünftige Technologien gut aufgestellt ist. Hinter dieser Fragestellung steht das Ziel, ein eingeführtes Deployment-Werkzeug lange nutzen zu können, ohne dass es veraltet oder nicht mehr für moderne Deployments geeignet ist. Hierbei handelt es sich um eine objektive Metrik, die gemessen werden kann und hierdurch eine gute Vergleichbarkeit untereinander gegeben ist.

Weitere Kosten?

Das Ziel keine Folgekosten in der Verwendung einer Open-Source Software zu nutzen ist offensichtlich. Dennoch kann es zu Funktionseinschränkungen kommen, wenn das Deployment-Werkzeug ebenfalls in einer Bezahlversion zu erwerben ist. Daher stellt sich die Frage, ob der volle Funktionsumfang des Werkzeugs zur Verfügung steht, oder nicht. So ergibt sich die objektive Metrik „weitere Kosten?“.

Zuvor benötigte Infrastruktur

Nach dem Ziel, dass der User vor der Installation keine Infrastruktur oder andere Software benötigen soll, ergibt sich die Frage, ob dies vor der Installation, beziehungsweise vor dem Start des Deployments nötig, oder ob es nach der Zielvorgabe auch ohne möglich ist. Aus dieser

Fragstellung heraus ergibt sich die objektive Metrik „zuvor benötigte Infrastruktur“, in der klar abgegrenzt werden kann, ob und wenn ja, welche Vorinstallationen notwendig sind.

Lizenzen

Die letzte Metrik „Lizenzen“ gibt Aufschluss über die Verwendung des Deployment-Werkzeugs. Zielführend ist die freie Verwendbarkeit der Software unabhängig vom Zweck. So soll die Software auch für gewerbliche Zwecke genutzt werden dürfen und auch für Deployments bei Kunden Anwendung finden. Die resultierenden Fragen hieraus sollen Aufschluss darüber geben, ob das Werkzeug frei benutzbar ist oder es Einschränkungen gibt. Außerdem stehen weiterführende Nutzungen für Kundenzwecke zur Debatte, um nicht gegen Lizenzrechte zu verstoßen. Nähere Informationen sind in Kapitel 2.1.1 Lizenzen erläutert. Die Art dieser Metrik ist objektiv.

4.2 Deployment-Werkzeuge

In diesem Kapitel werden Open-Source Deployment-Werkzeuge konzeptionell evaluiert. 19 mögliche Tools werden näher betrachtet und in dieser Evaluation miteinander verglichen. Anschließend werden fünf Deployment-Werkzeuge in der experimentellen Evaluation genauer betrachtet. Die Evaluation basiert auf einer entwickelten Excel-Tabelle, in welcher die Werkzeuge anhand der zuvor aufgestellten Metriken miteinander verglichen werden. Die Ergebnisse dieser zunächst theoretischen Evaluation werden in diesem Kapitel dargestellt.

4.2.1 AWS CodeDeploy

AWS CodeDeploy ist ein Deployment-Werkzeug des bekannten Unternehmens Amazon. Es stellt automatisiert Software auf Clouddiensten oder lokalen Servern zur Verfügung. Es handelt sich bei AWS CodeDeploy also viel mehr um ein Bereitstellungsservice, welche kein Management der deployten Software ermöglicht. Des Weiteren handelt es sich nur um ein kostenfreies Open-Source Tool, wenn bereits ein kostenpflichtiger Amazon-Clouddienst verwendet wird. Aus diesen Gründen wird AWS CodeDeploy in dieser Evaluation nicht näher betrachtet. (vgl. Amazon Web Services, 2022)

4.2.2 GitLab Deployment

Das Deployment von GitLab erfolgt direkt über ein Web Interface aus dem bekannten Versionsverwaltungs-Werkzeug GitLab. Hierbei handelt es sich um Pipelines, welche über ein Dashboard im Web Interface angetriggert werden können. Die Voraussetzung hierfür ist, dass das Projekt auch über GitLab verwaltet wird. Andere Versionsverwaltungssoftware können nicht auf die Funktion zurückgreifen. Des Weiteren ist der Besitz einer Enterprise Edition notwendig damit diese Funktion zur Verfügung steht. In der kostenfreien Community Edition ist nicht die gesamte Funktionalität gegeben. Aus diesem Grund ist GitLab Deployment kein hinreichendes Open-Source Deployment-Werkzeug und wird nicht detaillierter beschrieben. (vgl. GitLab, 2022)

4.2.3 Harness Drone (CI) & Harness CD

Harness Drone und Harness CD sind zwei zunächst unterschiedliche Deployment-Werkzeuge, welche von dem Unternehmen Harness Inc. entwickelt wurden. Harness folgt einem Baukastenprinzip, bei welchem sich der Kunde sein Produkt nach Belieben zusammenstellen kann. Hierzu zählen unter anderem die Softwareprodukte Harness CD und Harness Drone. Des Weiteren enthält jedes einzelne Produkt in der kostenfreien Version einen nur sehr eingeschränkten Funktionsumfang. Es werden Einschränkungen in der Funktionalität, in der Anzahl der Benutzer, sowie in der Anzahl der Deployments oder Builds vorgenommen. Letztlich verbietet die PolyForm Shield License 1.0.0 die Benutzung für kommerzielle Zwecke. Aus diesen Gründen sind die Produkte nicht mit den anderen Tools vergleichbar und werden nicht näher betrachtet. (vgl. Harness Inc., 2022)

4.2.4 JFrog

Das Deployment-Werkzeug JFrog stellt ebenfalls eine Pipeline-Lösung dar, welche den gesamten Deployment-Lifecycle-Prozess abdeckt. Ursprünglich handelte sich um eine selbst gehostete Software, welche beim Endkunden installiert und von diesem gepflegt werden musste. 2019 kaufte JFrog die Cloudplattform Shippable, welche Continuous Integration in der eigenen Cloud ermöglichte. JFrog integrierte Shippable in ihre Software, sodass JFrog die eigene Software auch als Software-as-a-Service-Lösung vermarkten kann. Aktuell (Stand: 08.2022) wird dem Kunden die Wahl zwischen beiden Lösungen angeboten. Es kann zwischen zahlreichen

Bezahlversionen gewählt werden. Jede dieser Optionen grenzt nicht die Funktionalität der Software selber, dafür allerdings die Anzahl der Benutzer, die Größe der Datentransfers, verfügbarer Speicher, Anzahl Projekte, und so weiter stark ein. Es steht nur eine kostenfreie Cloud-Open-Source-Lösung zur Verfügung, bei der die Lizenzbedingungen nicht eindeutig zu finden sind. Die Dokumentation ist umfangreich, jedoch undurchsichtig. Das Suchen nach bestimmten Funktionen gestaltet sich, trotz vorhandener Suchfunktion, oft als schwierig. Die Weiterentwicklung der kostenfreien Version ist nicht sichergestellt, da bereits aktuell für Updates gezahlt werden muss und keine Informationen für einen langfristigen Support dargelegt sind. Der Support wird in der kostenfreien und den günstigeren kostenpflichtigen Versionen durch die Community geleistet. Hier kann auf ein eigenes Confluence zurückgegriffen werden. In anderen Fällen ist eine Suche über öffentliche Foren möglich, da JFrog hier gut vertreten ist. Außerdem werden der Community Live-Events und Online Workshops angeboten. Die Installation der Cloudversion gestaltet sich als problemlos, lediglich ein GitHub Account und eine Docker Installation sind zuvor notwendig. Die Integration des zu deployenden Codes erfolgt über die Weboberfläche im Browser. Andere Versionskontrollen Tools außer GitHub werden nicht unterstützt. Auf Grund der unklaren Verwendungsmöglichkeiten durch fehlende Einsicht in die Lizenzierung und der eingeschränkten Nutzungsmöglichkeiten wird JFrog in der folgenden experimentellen Evaluation nicht näher betrachtet. (vgl. JFrog, 2022)

4.2.5 Docker Compose

Docker Compose ist ein Deployment-Werkzeug, welches dafür bestimmt ist, Multi-Container Applikationen basierend auf Docker, mithilfe von YAML-Dateien zu definieren und anschließend zu deployen, bzw. die Container zu starten. Aus diesem Grund ist Docker Compose mehr ein Container Management Werkzeug, als ein richtiges Deployment-Tool. Eine Verwaltung oder ein Management der deployten Anwendungen ist nicht möglich. Docker Compose ist ein Softwareprodukt des amerikanischen Unternehmen Docker, Inc., welches in seiner ersten Version im Jahr 2015 released und seinem zweiten, aktuellen Stand (Stand: 08.2022), im Jahr 2020 released wurde. Zwischen dem Release von V1 und V2 und auch seit V2 wurden viele Updates veröffentlicht, weshalb von einer anhaltenden Entwicklung auszugehen ist. Docker Compose ist Open-Source. Allerdings wird für das Betreiben der Software die Docker Engine und der Docker Client benötigt. Docker bietet diese benötigte Infrastruktur für macOS und Windows

mit Docker Desktop an. Docker Desktop ist eine einfach zu installierende GUI-Anwendung, welche auf Linux nicht benötigt wird, da die Docker Engine hier direkt lauffähig ist und lediglich installiert werden muss. Auf Windows und macOS wird mit dem Start von Docker Desktop eine virtuelle Linux-Maschine gestartet. Docker Desktop ist ebenfalls eine Open-Source Software. Allerdings gibt es die Einschränkung, dass diese (im Enterprise-Bereich) nur bis zu einer Unternehmensgröße von 250 Angestellten und einem Jahresumsatz von 10 Millionen USD kostenfrei ist. Ansonsten werden Docker Desktop Lizenzen benötigt. Für Linux entfällt diese Einschränkung. Ist Docker Compose erfolgreich installiert können, dank einer Apache License 2.0, auch Enterprise Anwendungen deployed werden. Eine umfangreiche Dokumentation mit Beispielen vereinfachen die Handhabung des Produkts. Sollten dennoch Fragen oder Probleme auftreten, können diese mithilfe eines FAQs und zahlreichen Foreneinträgen gelöst werden. Allerdings bietet Docker keinen eigenen Community-Bereich oder ein eigenes Forum für den Benutzer an. Docker Desktop wird kein Bestandteil der experimentellen Evaluation, da es den Deployment Lebenszyklus nicht ansatzweise abdecken kann, sondern viel mehr ein Tool ist, welches zum geordneten Starten und der Kommunikation zwischen Containern sorgt. (vgl. Docker Inc., 2022)

4.2.6 Juju

Juju ist eine Software des britischen Unternehmens Canonical. Das Deployment Werkzeug ist nur eines von vielen Produkten des Unternehmens und nutzt Charmed Operatoren. Sie sind eine Abwandlung bereits bekannter Kubernetes Operatoren. Charmed Operatoren bilden all das ab, was auch Kubernetes Operatoren abdecken können. Ihr Mehrwert besteht darin, dass Sie nicht nur Unterstützung für Kubernetes Cluster bieten, sondern außerdem für Container, VMs und die Cloud. Des Weiteren bietet ein charmed – gegenüber einem klassischen Operator – Möglichkeiten Applikationen miteinander zu verbinden, anstatt sie nur zu deployen. Der Start eines Deployments erfolgt über verschiedene YAML-Dateien, die Juju selbstständig angelegt. Anschließend sind in diesen Konfigurationsdateien die Schritte zu beschreiben, die von Juju ausgeführt werden sollen. Der Entwicklungsstart von Juju war bereits 2011 und ist, dank einer GNU Affero General Public License, ohne Einschränkungen Open-Source. Ein großer Vorteil ist die sehr umfangreiche und detaillierte Dokumentation. Zu vielen Themen sind Tutorials vorhanden, welche eine Schritt für Schritt-Anleitung beinhalten. Juju bietet seiner Community

einen Blog und Mattermost, sowie ein Discourse Forum. In Verbindung mit videogestützten Tutorials, finden über diese Plattformen auch der Community Support statt. Support von Experten oder Entwicklern gibt es nicht. Juju bietet seinen Benutzern einen kommandozeilenbasierten Client. Dieser ist bevorzugt auf Linux zu installieren, ist aber ebenfalls für macOS und Windows verfügbar. Ein Dashboard ist nur über ein externes Tool verfügbar. Um die Vergleichbarkeit zwischen den Werkzeugen sicherzustellen, wird dieses hier allerdings nicht verwendet. Für die Installation wird keine Software benötigt, die vorher installiert sein muss. Wenn allerdings lokal deployed und getestet (in der „localhost“-Cloud) werden soll, dann wird LXD (der Linux Container Daemon) benötigt. LXD ist, ähnlich wie Docker, ein Management Tool für Container. Anders als bei Docker handelt es sich hier allerdings nicht um Docker – sondern um Linux Betriebssystem-Container. Daher ist LXD auch nur für Linux verfügbar. (vgl. Juju, 2022)

4.2.7 Rundeck

Rundeck ist ein auf Java basierendes Automatisierungs-Werkzeug, welches Teil des 2009 gegründeten amerikanischen Unternehmens PagerDuty ist und unter deren PagerDuty Process Automation-Sparte läuft. Die Entwicklung von Rundeck hat 2010 begonnen. Seit dem ersten Release wurden stetige Verbesserungen an der Software vorgenommen und werden auch mit großer Wahrscheinlichkeit in Zukunft weitergeführt, da das Werkzeug auch als kostenpflichtige (Enterprise) Version angeboten wird. Die kostenfreie Community-Edition ist mit einer Apache License 2.0 lizenziert und steht dementsprechend auch für kommerzielle Zwecke zur freien Verfügung. Im Gegensatz zu vielen anderen Deployment-Werkzeugen, geht im Vergleich zu der Enterprise Version viel Funktionalität verloren, was die Benutzerfreundlichkeit erheblich einschränkt. Zusätzlich bleiben in der Dokumentation Fragen zu der Funktionalität offen, was den Funktionsumfang der Software zunächst unklar lässt. Der Support ist dank einer großen Community gut. In der Community Version wird sich auf den Community Support beschränkt. In der Enterprise-Edition ist zusätzlicher Experten Support enthalten. Weiterer Support ist über How-To's und Tutorials möglich. Die Kommunikation der Community kann über zahlreiche Wege stattfinden. Hierzu zählt der Rundeck Blog, ein Libera.Chat Space, eine Stackoverflow Group, Twitter und noch mehr. Vor der Installation müssen Java und NodeJs installiert sein. Anschließend wird eine YAML-Datei angelegt, in welcher die sogenannten

Nodes beschrieben werden. Anschließend kann das zuvor importierte Softwareprojekt über die Oberfläche deployed und verwaltet werden. Das Deployment-Werkzeug wird aufgrund von einer leichtgewichtigeren Open-Source Variante im Vergleich zu der kostenpflichtigen Variante nicht näher untersucht. Vergleichbare Tools haben gegenüber Rundeck keinen Nachteil und stehen ohne Einschränkungen kostenfrei zur Verfügung, weshalb diese näher evaluiert werden. (vgl. PagerDuty, Inc., 2022)

4.2.8 Spinnaker

Dass unter der Apache License 2.0 lizenzierte Deployment-Werkzeug Spinnaker, ist eine im Jahr 2015 releaste Software, welche mit andauernden neuen Releases aufwartet. Ursprünglich hat Netflix die Entwicklung geleitet, wurde aber anschließend von verschiedenen Unternehmen abgelöst. Aktuell (Stand: 08.2022) wird die Software von dem Unternehmen Armory und OpsMx für den kommerziellen Gebrauch weiterentwickelt. Die Hauptentwicklung war von 2016 bis 2020. Seitdem ist ein starker Trend nach unten zu erkennen, obwohl für 2023 ein neues Release angekündigt ist. Es gibt keine kostenpflichtigen Versionen oder eingeschränkten Funktionsumfang. Spinnaker kann die Deployments in einer Cloud oder auf einem Kubernetes Cluster durchführen. Die Installation von Spinnaker selbst kann nur auf einem Kubernetes Cluster erfolgen. Wenn dies nicht gewünscht ist, kann ein Client installiert werden, über den Spinnaker (zentral gehostet) gesteuert werden kann. Dieser Weg bringt die Einschränkung mit sich, dass der Client (Halyard) heruntergeladen werden und installiert werden muss. Dies ist nur auf Linux, macOS oder in einem Docker Container möglich. Unter Windows ist keine direkte Installation möglich. Die Dokumentation ist ausreichend, allerdings nicht so umfangreich wie bei anderen Deployment-Werkzeugen, weshalb ein guter Support von wichtiger Bedeutung ist. Allerdings kann hier kein Support geboten werden. Support kann lediglich über den Community Space (hier: Slack und Community Blog) erfolgen, was bei Problemen oder Unklarheiten zu Problemen führen kann. Spinnaker wird in dieser Evaluation nicht näher betrachtet, da seit 2020 kaum noch Entwicklungen stattfinden und im Jahr 2022 noch kein Commit stattgefunden hat. Außerdem ist die Dokumentation im Vergleich zu den anderen Tools nicht sehr ausführlich und auch der Support eher mangelhaft. (vgl. Spinnaker, 2022)

4.2.9 AppVeyor

Das Deployment-Werkzeug AppVeyor ist eine große Continuous Integration und Continuous Delivery Software. Der Entwicklungsstart war 2011 und ist seitdem bei großen Unternehmen im Einsatz. Die Deployments erfolgen über Pipelines aus Versionskontrollen Anwendungen wie GitLab oder Bitbucket, zeichnet sich aber durch die Vielfalt an möglicher Versionsverwaltungssoftware-Support aus. Auch AppVeyor unterscheidet zwischen Bezahl- und Open-Source-Versionen. Des Weiteren wird hier zusätzlich zwischen selbst gehosteten und von AppVeyor gehosteten Versionen unterschieden. Beide Versionen bieten eine Open-Source Version. Die drei kostenpflichtigen Software-as-a-Service Lösungen kosten zwischen 29 und 99 USD pro Monat. Die selbst gehostete kostenpflichtige Variante kostet 249 USD pro Monat. Die Versionen untereinander unterscheiden sich nicht im Funktionsumfang der Software selber. Es werden Einschränkungen über Benutzerzahlen, Anzahl Projekte und ausführbare Jobs vorgenommen. Die kostenfreien Varianten sind allerdings nicht für die kommerzielle Nutzung geeignet, da diese aus Lizenzgründen nur für öffentliche Projekte verwendet werden darf. Für die Installation muss keine Infrastruktur vorhanden sein und alle Betriebssysteme werden unterstützt. Bei Bedarf kann die Software auch in einem Docker Container installiert werden. AppVeyor ist eine ausgereifte Software mit vielen User, weshalb auch ständig Weiterentwicklungen und ständige Releases veröffentlicht werden. Auch die Dokumentation ist umfangreich und gut lesbar, allerdings lässt sie Fragen offen und erfordert ein Austesten. 14-Tage Testversionen können heruntergeladen werden. Der Support erfolgt in der Open-Source Variante über das Community Forum. In den kostenpflichtigen Versionen gibt es zusätzlichen technischen Support von AppVeyor. Das Forum ist der einzige Community Bereich, andere Chat-Plattformen oder ähnliches gibt es nicht. Da die kommerzielle Nutzung in den kostenfreien Versionen nicht möglich ist, wird AppVeyor in der experimentellen Evaluation nicht tiefgründiger untersucht. (vgl. AppVeyor, 2022)

4.2.10 CircleCI

Die Pipeline-Lösung CircleCI bietet die Möglichkeit in der Cloud oder selber-gehostet auf Servern betrieben zu werden, sodass je nach Präferenz des Kunden, die passende Lösung ausge-

wählt werden kann. In beiden Version kann die Software nur mit Verknüpfung einer Anwendung für Versionsverwaltung verwendet werden. CircleCI bietet die Verwendung eines GitLab, GitHub oder Bitbucket Repositories an. Lokale Projekte können nicht deployed werden. Außerdem steht nur eine Cloudversion mit Einschränkungen als Open-Source Lösung zu Verfügung. Neben der frei verfügbaren Variante, stehen noch drei weitere Varianten zur Auswahl, welche von 15 bis hin zu mehreren tausenden Dollar kosten können. In dieser Evaluation wird sich ausschließlich auf die freie Cloudversion bezogen. Der Funktionsumfang unterscheidet sich nicht, allerdings werden Einschränkungen in gestellter Infrastruktur vorgenommen. Die Dokumentation ist sehr umfangreich und beantwortet die meisten Fragen. Lediglich Log-Ausgaben sind nicht beschrieben. Auch der Support ist durch eine große Community sichergestellt. CircleCI bietet hier ein eigenes Forum, zahlreiche How-To's und eine freie Trainings Academy mit videogestützten Tutorials an. Da es sich um eine Web-Applikation handelt, ist keine Installation notwendig. Dennoch kann ein Client installiert werden, welcher allerdings nur für macOS und Linux zur Verfügung steht. Ein weiterer entscheidender Nachteil besteht in der Lizenzierung. Es ist unklar, ob die kostenfreie Variante von CircleCI für die kommerzielle Nutzung zugelassen ist. Aus diesem Grund, in Verbindung der daraus resultierenden Einschränkungen, wird CircleCI in der experimentellen Evaluation nicht näher untersucht. (vgl. Circle Internet Services, Inc., 2022)

4.2.11 Jenkins

Jenkins ist eines der größten und bekanntesten, unter der MIT License lizenzierten, Open-Source Automatisierungsserver und ist stark erweiterbar, sodass dieser nicht nur den Prozess des automatisierten Deployments abdecken kann. Die Software ist eine auf Java-basierte Pipeline-Lösung, die keine versteckten Kosten hat, sodass jedem User der volle Funktionsumfang zur Verfügung steht. Die erste Version wurde 2011 veröffentlicht und seitdem stetig weiterentwickelt. Aktuell (Stand: 08.2022) werden wöchentliche Releases veröffentlicht. Jenkins ist sehr umfangreich, detailliert und mit Bildern und Videos dokumentiert, sodass keine Fragen offenbleiben. Außerdem werden alle Betriebssysteme unterstützt und bei Bedarf kann Jenkins auch auf Containern oder auf einem Kubernetes Cluster installiert werden. Jenkins hat sehr viele User, mit welchen über mehrere Plattformen, wie zum Beispiel Chats, Mailinglisten oder Foren und organisierten Community Events/Treffen kommuniziert werden kann. Der Support für

Jenkins erfolgt ausschließlich über die zahlreichen User und der umfangreichen Dokumentation mit Videos und Troubleshooting. (vgl. Jenkins, 2022)

4.2.12 Buildbot

Das Deployment-Werkzeug Buildbot wurde im Jahr 2011 begonnen zu entwickeln und ist unter der GNU General Public License lizenziert. Anders als alle anderen evaluierten Tools, ist Buildbot in Python geschrieben und wird in einem öffentlich einsehbarem Git-Repository von einem Maintainer verwaltet und der Community gepflegt. Auffällig ist die geringe Aktivität der Community in diesem Projekt und die damit verbundenen wenigen Commits. Auch hier steht eine Weboberfläche als GUI zur Verfügung, welche sich nach der Installation und Start der Software lokal über den Browser darstellen lässt. Die Installation der Software erfolgt dank Python über PIP – den Python Package Installer, welcher aber vorher über Python 3 installiert werden muss. Die Dokumentation ist im Vergleich zu den anderen Werkzeugen sehr knapp gehalten und nur für Windows dokumentiert, weshalb ohne das Austesten der Software keine Aussage dazu getroffen werden kann, ob diese mit anderen Betriebssystemen kompatibel ist. Die Community könnte über Mailinglisten und einem Libera.Chat miteinander kommunizieren, allerdings findet keine Kommunikation statt und auch der letzte Blogeintrag wurde 2019 veröffentlicht. (Stand: 08.2022) Des Weiteren gibt es nur Community Support, allerdings werden nur wenige Hilfestellungen oder Fragen beantwortet. Der schwache Support und die wenigen Contributor und die damit verbundene Unsicherheit über Weiterentwicklungen in der Zukunft waren die Gründe für das Ende weitergehender Untersuchungen. (vgl. Buildbot)

4.2.13 GoCD

GoCD ist neben Jenkins ein weiteres populäres Open-Source Deployment-Werkzeug ohne versteckte Kosten. Der Entwicklungsstart, der unter der Apache License 2.0 lizenzierte Software war 2014 und wurde seitdem ständig weiterentwickelt. GoCD wird von der Community weiterentwickelt und zusätzlich von der Thoughtworks Inc. gesponsert. Eine mit der umfangreichsten Dokumentation mit Bildern und Videos lässt kaum eine Frage offen. Die Installation des GoCD Servers ist auf jedem Betriebssystem und auf einem Kubernetes Cluster möglich. Wenn dies gewünscht ist, muss zunächst ein Kubernetes Cluster vorhanden sein, ansonsten ist

keine Infrastruktur zuvor notwendig. Der Support ist dank der großen Community sichergestellt und wird über öffentliche bekannte Foren, sowie eine Google Groups Gruppe und einen Gitter Chat sichergestellt. Außerdem können die neusten Informationen rund um GoCD über den gut gepflegten Blog eingesehen werden. (vgl. GoCD, 2022)

4.2.14 TeamCity (JetBrains)

Ein ebenfalls bekanntes und großes Deployment-Werkzeug ist das von JetBrains entwickelte TeamCity. Allerdings gibt es bei TeamCity Einschränkungen, wenn man die kostenfreie Open-Source Variante verwenden möchte. Grundsätzlich gibt es auch hier zwei Möglichkeiten der Verwendung. Zunächst besteht die Möglichkeit, die Cloudvariante zu wählen, welche als Software-as-a-Service angeboten wird. Dieser Service wird zunächst als zweiwöchige Testphase angeboten und kostet ab diesem Zeitpunkt ab 45 USD pro Monat. Außerdem bietet JetBrains die Software als selbst vom User gehostete Version an. Hier besteht die Möglichkeit zwischen einer kostenfreien und einer kostenpflichtigen Version zu wählen. Die kostenfreie Version verfügt über dieselben Funktionen wie die Enterprise Version, allerdings stehen hier nur 100 Buildkonfigurationen zur Verfügung. Die kostenpflichtige Version kostet circa 167 USD pro Monat und besitzt diese Einschränkungen nicht. Allerdings sind in beiden Konstellationen nur drei Build-Agents enthalten. Jeder weitere Build-Agent muss für weitere 25 USD pro Monat dazugebucht werden. Alle möglichen Konfigurationen benötigen keine vorherige Infrastruktur und sind auf jeder Systemumgebung zu installieren. Das Tool hat viele Kunden, weshalb Weiterentwicklungen auch in Zukunft released werden. Die Dokumentation ist die Beste in dieser Evaluation. Es bleibt keine Frage offen und es gibt zahlreiche Tutorials und grafische Darstellungen. Die Community ist auf vielen Plattformen vertreten, allerdings bietet JetBrains keinen eigenen Chat, Events oder Foren. Die Community ist daher auf bekannten Foren wiederzufinden. Falls die Hilfestellungen und die Tutorials in der Dokumentation nicht ausreichen, wird der Support in der Open-Source Version über die Community in den Foren sichergestellt. Die kostenpflichtigen Versionen bekommen priorisierten technischen Support von TeamCity Experten. Aufgrund der stark Einschränkungen in der Benutzbarkeit in der kostenfreien Open-Source Variante und der teuren Preispolitik, wird TeamCity nicht tiefergehend in einer weiterführenden Evaluation behandelt. Die Einschränkungen machen das Tool für die kommerzielle Nutzung als Open-Source Variante uninteressant. (vgl. JetBrains s.r.o., 2022)

4.2.15 ArgoCD

Das auf einem Kubernetes Cluster laufenden Deployment-Werkzeug ArgoCD, wurde erst 2018 begonnen zu entwickeln. Ein zusätzlicher ArgoCD Client dient zum Ausführen von Befehlen auf der Kommandozeile und kann auf allen Betriebssystemen installiert werden. Zusätzlich wird nach dem Start ein Web-UI für die übersichtliche Bedienung und Darstellungen der Software im Browser geöffnet. Für die Inbetriebnahme wird Vorwissen im Thema Kubernetes benötigt, da zuvor selbstständig ein Kubernetes Cluster aufgesetzt und anschließend mit diesem gearbeitet werden muss. Die ansonsten recht ausführliche Dokumentation beinhaltet diese Schritte nicht. ArgoCD ist ein Community Open-Source Git-Projekt mit vielen aktiven Entwicklern, welches öffentlich einsehbar ist und die zukünftigen Releases sicherstellt. Die neuesten Veränderungen werden in einem Blog festgehalten. Die Software ist auch für kommerzielle Projekte geeignet, da das Tool auf einer Apache License 2.0 lizenziert ist. Nachteile stellen allerdings die Kommunikation unter der Community dar. Hier steht lediglich ein Slack Kanal zur Verfügung. Auch der Support erfolgt nur über ein internes FAQ. Sollte dies nicht genügen, können öffentliche Foren befragt werden. (vgl. Argo CD, 2022)

4.2.16 Codefresh (codefresh.io)

Das Deployment-Werkzeug Codefresh hostet das zuvor beschriebene Tool ArgoCD. Es erweitert die Funktionalität zu einem vollwertigen Automatisierungswerkzeug und wird vollständig von Codefresh gehostet. ArgoCD ist eine Continuous Integration-Lösung. Codefresh ergänzt dies um eine Continuous Delivery-Lösung mit automatisierten Tests, mit der Möglichkeit weitere externe Tools, wie zum Beispiel Jenkins, einzubinden. Im Gegensatz zu ArgoCD gibt es bei Codefresh auch Funktionalität die nicht kostenfrei ist. Lediglich eine Community Edition mit Einschränkung der Benutzeranzahl ist kostenfrei. Die Team-Version, für 49 USD pro Developer und die Enterprise-Version (Preis auf Anfrage) sind kostenpflichtig. Die kommerzielle Nutzung in der kostenfreien Version ist nicht eindeutig gestattet, da keine Lizenzangaben gemacht werden. Allerdings ist Dokumentation ansonsten sehr ausführlich. Neue Weiterentwicklungen sind aufgrund von kostenpflichtiger Versionen und Usern gesichert. Codefresh ist Unabhängig von der Systemumgebung, da es gehostet wird und man lediglich mit seinem User

auf eigenen Code in Repositories zugreift. Außerdem besteht in der kostenpflichtigen Enterprise-Version die Möglichkeit, Codefresh auf einem eigenen Kubernetes Cluster zu hosten. Der Support wird durch die Community sichergestellt, in dem eine umfangreiche Service Plattform zur Verfügung gestellt wird. Alle Fragen und Probleme werden in kurzer Zeit beantwortet. Außerdem werden Webinars, ein eigenes Forum und ein Blog für Mitteilung neuester Ereignisse zur Verfügung gestellt. Trotzdem wird Codefresh nicht weitergehend betrachtet, da die Kapazitäten durch die kostenfreie Open-Source Variante beschränkt sind. Vergleichbare Tools haben diese Einschränkung nicht und werden daher näher evaluiert. (vgl. Codefresh, 2022)

4.2.17 PHP Deployer

Der PHP Deployer ist ein kleines und eher unbekanntes Deployment-Werkzeug, welches dank der MIT License für die kommerzielle Nutzung zugelassen ist und verwendet werden darf. Im Vergleich zu den meisten anderen Tools handelt es sich lediglich um ein schmales Kommandozeilentool, welches 2013 begonnen wurde zu entwickeln. Seit dem Entwicklungsstart bis heute (Stand: 08.2022) ist nur ein Hauptcontributor vorhanden, welcher aber regelmäßig Änderung an dem Tool vornimmt. Die Software ist in PHP geschrieben und daher auf jeder Systemumgebung installierbar. Allerdings müssen für die Installation bereits PHP und der Dependency Manager für PHP, Composer, auf dem System vorhanden sein. Die kurze Dokumentation lässt einige Fragen offen ist nicht sehr detailliert. PHP Kenntnisse sollten vorher vorhanden sein, da die Beschreibungen vereinzelt Wissen voraussetzen. Es sind über GitHub hinaus keine Informationen über eine Community des PHP Deployers zu finden, ebenso gibt es lediglich Support über das GitHub Forum. Allerdings sind hier, nach einer gewissen Wartezeit, Antworten auf den überwiegenden Teil der Fragen zu finden. Das Tool wird nicht näher betrachtet, da es aufgrund seiner Leichtigkeit und der fehlenden Abdeckung des gesamten Deployment-Lebenszyklus, nicht mit den anderen Deployment-Werkzeugen in dieser Evaluation verglichen werden kann. (vgl. Deployer, 2022)

4.2.18 Capistrano

Das ebenfalls schmale Deployment-Werkzeug Capistrano ist ein Kommandozeilentool, welches in Ruby geschrieben ist. Der zu deployende Code wird per SSH aus einem Versionskontrollentool in Capistrano integriert und anschließend über Scripts für das Deployment vorbereitet. Die Entwicklung hat 2013 begonnen und ist seit 2018 stark abgeschwächt. Es gibt kaum Weiterentwicklungen und auch die Google Groups Community, sowie andere Forenbeiträge haben keine hohe Aktivität mehr. Die Installation und Benutzung ist in der Dokumentation gut und ausführlich beschrieben. Eine Vorinstallation von Ruby ist erforderlich, anschließend ist die Software für Linux, Windows und macOS verfügbar. Auch Capistrano ist unter der MIT License lizenziert und daher auch für den kommerziellen Gebrauch geeignet. Das Deployment-Werkzeug wird aufgrund von unzureichender Community-Kommunikation, mangelndem Support und Kleingewichtigkeit im Vergleich zu den übrigen Tools nicht in der weiterführenden Evaluation begutachtet. (vgl. Capistrano, 2022)

4.2.19 Rancher

Bei Rancher handelt es sich um eine Open-Source Container Management Software, die einem ermöglicht Container in Kubernetes zu deployen und zu verwalten. Rancher kann mehrere Kubernetes Cluster in eine zentrale Verwaltung zusammenführen und zeigt daher besonders bei mehreren Cloudumgebungen seine Stärke. Das Unternehmen Rancher Labs, welches hinter dem Deployment-Werkzeug Rancher steht, sichert die zukünftigen Weiterentwicklungen. Die Software ist kostenfrei und unter der Apache 2.0 License lizenziert. Die umfangreiche Software enthält viel Funktionalität, welche die umfangreiche Dokumentation erfordert. Diese ist durch weitere integrierbare Tools stark erweiterbar, sodass zum Beispiel Monitoring- oder Logging-Tools verwendet werden können, um die User Experience zu verbessern. Die Installation ist einfach und betriebssystemunabhängig, da diese auf einem Kubernetes Cluster durchgeführt werden. Anschließend steht dem Benutzer ein Web-UI zur Verfügung, über welches Rancher bedient und die Cluster verwaltet werden. Eine lokale Installation ist nicht möglich. Aus diesem Grund sind eine Docker und eine Kubernetes Installation notwendig und Vorkenntnisse in Kubernetes und Containertechnologien notwendig. Der Deployment-Prozess wird anhand von

YAML-Dateien beschrieben und anschließend über Rancher angetriggert, sodass das Deployment in dem Kubernetes Cluster ausgeführt wird. Die Community von Rancher verfügt über zahlreiche Möglichkeiten mit anderen Usern in Kontakt zu treten oder ein Teil von Rancher zu sein. Hierzu zählen Slack, Live-Events, Webinars, ein eigenes Forum oder die eigene Rancher Community Page. Der Support wird von der Community über die aufgezeigten Community Spaces getragen, jedoch kann bei Bedarf kostenpflichtiger Enterprise-Support hinzugebucht werden. Die Preise hierfür erhält man auf Anfrage nach Ausfüllen eines Kontaktformulars. (vgl. Rancher, 2022)

5 Experimentelle Evaluation

Die experimentelle Evaluation ist die weiterführende Methodik, bei der eine kleinere Auswahl an Deployment-Werkzeugen einer genaueren Untersuchung unterzogen wird. Zunächst wird die Auswahl der Tools anhand der gewonnenen Erkenntnisse aus der konzeptionellen Evaluation auf einige wenige reduziert. Anschließend werden neue Metriken aufgestellt, die nur experimentell erprobt werden können. Mit Hilfe von Open-Source Softwareprojekten werden die Tools auf Funktionalität geprüft und anhand der aufgestellten Metriken bewertet.

5.1 Metriken

- Deployment-Dauer
- Logging
- Testintegration
- Ressourcenmanagement
- Verwendbarkeit/Anpassbarkeit
- Skalierbarkeit
- Backups

- Einfachheit
- Stabilität
- Performance

Deployment-Dauer

Besonders interessant bei der Evaluation von Deployment-Werkzeugen ist die Zeit die für ein Deployment benötigt wird. In dieser Wartezeit kann in der Regel keine andere Tätigkeit verrichtet werden, da der Verlauf des Deployments überwacht werden muss. Kommt es zu unerwarteten Fehler, kann der Benutzer das Fehlverhalten überprüfen. Aus diesem Grund wird die objektive Metrik „Deployment-Dauer“ aufgestellt, welche eindeutig anhand von einer Zeitmessung zu bestimmen ist. Das Ziel jedes Deployment-Werkzeugs sollte daher ein möglichst schnelles Deployment sein.

Logging

Die objektive und subjektive Metrik „Logging“ wurde mit dem Ziel aufgestellt, dass aktuelle Informationen über das Deployment für den User einsehbar sein sollen. Der User möchte über eventuelle Fehler und den Status des Deployments aufmerksam gemacht werden. Hieraus ergibt sich die Frage, ob mit der Hilfe des Deployment-Werkzeugs Fehler erkannt werden und festgestellt werden kann, wieso diese Fehler auftreten. Des Weiteren soll der User darauf aufmerksam gemacht werden, ob es sich bei dem Problem um einen Anwendungsfehler oder einen Softwarefehler handelt. Dies kann mit entsprechenden Log-Ausgaben eingesehen werden. Diese Metrik ist zunächst objektiv, da klar aufgezeigt werden kann, ob das Deployment-Werkzeug das Logging im Allgemeinen unterstützt. Allerdings besteht der subjektive Part der Bewertung in der Angabe über die Qualität der Logs. Jedes Deployment-Werkzeug nutzt ein anderes Logging-Tool oder sie unterscheiden sich in der Art und Weise der Verwendung. Die Bewertung dieser Logs erfolgt durch ein Individuum und ist für Dritte nicht messbar, weshalb es sich ebenfalls um eine subjektive Metrik handelt.

Testintegration

Die Frage, ob Tests über das Deployment-Werkzeug ausführbar sind, ergibt sich aus dem Ziel, dass Unit- und Integrationstests über das Werkzeug ausführbar sein sollen, um frühzeitig auf Fehler aufmerksam gemacht zu werden. Mit der objektiven Metrik „Testintegration“ ist bewertbar, ob das Deployment-Werkzeug diese Funktion bereitstellt oder nicht. Es handelt sich um eine objektive Metrik, da die Feststellung für jeden User gleich und nachvollziehbar bewertet werden kann. Durch diese Metrik kann ermittelt werden, ob das Deployment-Werkzeug die kostensparendere Möglichkeit Fehler zu erkennen, anstatt dies während eines aufwendigeren Deployments oder während der Laufzeit einer Anwendung festzustellen, mitbringt oder nicht.

Ressourcenmanagement

Eine weitere Metrik ist das „Ressourcenmanagement“, welche sich aus dem Ziel ergibt, dass sich der User die benötigten und die verfügbaren Ressourcen anzeigen lassen können soll. Die zu beantwortende Frage ist also, ob der User eine Übersicht über die laufenden Deployments und die laufenden, bereits deployten, Anwendungen hat und die damit verbundenen benötigten, sowie die allgemein verfügbaren Ressourcen einsehen kann. Zunächst gilt also die Frage zu klären, ob das Deployment-Werkzeug eine Möglichkeit bietet dies zu tun. Dies stellt eine objektive Bewertung dar. Die daraus resultierende, zweite Bewertungsgrundlage ergibt sich aus dem Funktionsumfang der gebotenen Darstellung. Diese zweite Bewertung ist subjektiv, sodass es sich erneut um eine Mischform aus einer subjektiven und einer objektiven Metrik handelt.

Verwendbarkeit/Anpassbarkeit

Die Metrik „Verwendbarkeit/Anpassbarkeit“ ergibt sich aus der Fragestellung, ob Codeanpassungen an der eigenen Anwendung notwendig sind, um das Deployment-Werkzeug für diese Anwendung anzuwenden. Je nach Antwort auf diese erste Frage, ergeben sich weitere Folgefragen, die in dieser Metrik mit abgedeckt sind, sodass es sich ebenfalls um eine Mischform aus einer subjektiven und einer objektiven Metrik handelt. Bei keinen nötigen Codeanpassungen handelt es sich um eine rein objektive Metrik. Sollten Codeanpassungen nötig sein ist eine

subjektive Bewertung der benötigten Anpassungen entscheidend, da der Aufwand hier stark variieren kann. Die Vorbereitung für das Deployment über eine Konfigurationsdatei ist zum Beispiel weniger aufwändig als das Umschreiben von Code.

Skalierbarkeit

Diese Metrik entsteht aus dem Ziel, dass ein User die Ressourcen verwalten können soll. Aus dieser Anforderung ergibt sich die Fragestellung, ob ein User die Ressourcen einer deployten Anwendung anpassen/verändern, also skalieren kann. Dies soll zur Laufzeit möglich sein und wird unter der Metrik „Skalierbarkeit“ zusammengefasst. Diese Metrik ist durch eine eindeutige messbare Antwort zu beantworten und wird daher in die Kategorie der messbaren Metriken eingeordnet.

Backups

Die Metrik „Backups“ ergibt sich aus den Zielen, dass die Daten der Anwendungen nicht verloren gehen, sondern gespeichert werden können. Diese Datenspeicherungen sollen im Hintergrund oder auf Wunsch manuell möglich sein und dienen dem Zweck der Datensicherung, falls es zu einem unerwarteten Fehlverhalten der Software oder zu einer Deactivation der deployten Anwendung kommt. Zunächst ergibt sich die Frage, ob eine Datensicherung erstellt werden kann, beziehungsweise ob Daten gespeichert werden können. Bei positiver Beantwortung der ersten Frage kann die weiterführende Frage gestellt werden, welche Datensicherungen möglich sind (automatische Backups, manuelle Backups). Da diese Frage klar und unabhängig vom Antwortgeber zu beantworten ist, ergibt sich die objektive Metrik „Backups“. Diese Metrik zählt trotz der klaren Beantwortung in die experimentelle Evaluation, da nicht jede Dokumentation ausreichend detailliert ist um Unterschiede aufzeigen zu können.

Einfachheit

Eine weitere Metrik ergibt sich aus dem Ziel, dass eine Benutzeroberfläche möglichst intuitiv sein sollte. Daraus ergibt sich anschließend die Frage, wie gut die Benutzeroberfläche bedien-

bar ist. Aus dieser Fragestellung resultiert die subjektive Metrik „Einfachheit“, da sie von jedem Benutzer anders bewertet werden kann. Die Unterschiede werden möglichst objektiv aufgezeigt und anschließend bewertet.

Stabilität

„Stabilität“ beschreibt die ständige Erreichbarkeit eines Systems ohne auftretenden Fehler oder Ausfälle. Die Metrik entsteht aus der Zielsetzung, dass ein System ständig erreichbar sein soll und keine Fehler oder Abstürze auftreten. Es handelt sich um eine Mischform aus subjektiver und objektiver Metrik, da Fehler und Abstürze zählbar sind, allerdings die Entstehung dieser Fehler von einem User angetriggert werden muss. Je nachdem, welche Interaktionen vom Benutzer vorgenommen werden, können unterschiedliche Fehler auftreten.

Performance

Die letzte Metrik für die experimentelle Evaluation ist „Performance“. Sie entsteht aus der zu Grunde liegende Zielstellung, dass keine Wartezeiten während Usereingaben entstehen sollen. Daraus entsteht die Frage, ob das Deployment-Werkzeug träge oder performant ist und wo hier die Unterschiede zwischen den Tools liegen. Die Bewertung erfolgt subjektiv, weil die Unterschiede zwischen den Werkzeugen von der Art der Userinteraktion abhängig sein können. Das Verhalten ist hier ähnlich zu der Bewertung der Stabilität einer Software.

5.2 Softwareprojekte

(basierend auf einer Microservice-Architektur)

Die experimentelle Evaluation wird anhand von einem Open-Source Softwareprojekten durchgeführt. Nach vollständigem Start der Anwendungen wird dieses auf seine Funktionalität überprüft, um die Korrektheit des Deployments festzustellen.

Dennoch müssen auch die Softwareprojekte bestimmte Anforderungen erfüllen. Zunächst muss sichergestellt sein, dass die Lizenzierung eine Verwendung erlaubt. Wenn keine passende Lizenz auf das Projekt zutrifft, kann diese nicht verwendet werden. Bei weiterer Betrachtung ist wichtig, dass der Aufbau des Softwareprojektes auf einer logischen und verständlichen

Struktur basiert. Die Codequalität ist hier besonders für die Wartbarkeit bzw. Veränderbarkeit von großer Bedeutung, da dies bei der experimentellen Evaluation benötigt wird. Anschließend wird ein beliebiges Projekt ausgewählt, welches ein Web User Interface bereitstellt, um ein späteres Austesten im Browser möglich zu machen, damit keine Abhängigkeit zu eventuellen falschen Tests besteht.

Für diese Evaluation wurde ein Microservice-Kubernetes Softwareprojekt (Wolff, et al., 2022) ausgewählt. Es besteht aus lediglich drei Microservices und wird für die Durchführung der experimentellen Evaluation verwendet, da es sich durch die Einbindung von Maven und Kubernetes leicht an das jeweilige Tool anpassen lässt. Die freie Verwendung wird durch die Lizenzierung mit der Apache 2.0 License sichergestellt.

5.3 Deployment-Werkzeuge

Die Deployment-Werkzeuge Juju, ArgoCD, Rancher, Jenkins und GoCD werden in dieser experimentellen Evaluation einer genaueren Betrachtung unterzogen, da sie als kleinere Auswahl aus der konzeptionellen Evaluation hervorgingen.

5.3.1 Juju

Bei dem Juju handelt es sich als einzig verbleibendes, um ein klassisches Kommandozeilen-Tool. Der gesamte Prozess des Deployment-Lifecycles lässt hierüber abdecken. Der Start eines Deployments kann nach der Installation des LXD, dem Lifecycle Manager und des Juju Clients erfolgen. Dieses sind einfach auf allen Systemumgebungen zu installieren, allerdings ist die bevorzugte Variante Linux, da auch die gesamte Dokumentation für Linux geschrieben ist. Wenn die benötigte Software installiert ist, wird mit der Initialisierung des Projektes die benötigte Ordnerstruktur angelegt, die sogenannte Charm-Struktur. Anschließend müssen die Dateien vom Benutzer nach einem bestimmten Schema angepasst werden. Abschließend kann der „Deploy“ Befehl über die Kommandozeile ausgeführt werden und die Anwendung wird, je nach vorheriger Konfiguration, in der Cloud oder als localhost auf einem Cluster laufen. Die

Deployment-Dauer ist aufgrund dessen, dass zuvor kein Kubernetes Cluster laufen muss, sondern dieser erst mit dem Starten des Deployments hochfährt, länger als bei vergleichbaren Tools. Allerdings dauert die Zeit für das manuelle Starten eines Clusters deutlich länger als bei einem Start von Juju. Aus diesem Grund ist die Deployment-Dauer in Summe als sehr gut einzustufen. Allerdings besteht lediglich die Möglichkeit, das Deployment auf einem Cluster durchzuführen. Weniger gut sind die Log-Ausgaben, die lediglich schlecht zu erkennen und zu deuten über die Kommandozeile laufen. Sie sind nicht anpassbar oder ähnliches und für Neulinge nicht zu deuten. Des Weiteren ist kein Ressourcenmanagement möglich, da keine Informationen über verfügbare Ressourcen oder bereits verbrauchte Ressourcen dargestellt werden. Eine Skalierbarkeit ist nur über die Veränderung der Konfigurationsdateien möglich. Hierfür muss die Anwendung zunächst heruntergefahren und anschließend erneut gestartet werden, allerdings gestaltet sich die Skalierung durch fehlende Information der Ressourcen als schwierig. Positiv zu erwähnen ist die vorhandene Testintegration, welche sich ebenfalls über eigene Testdateien anpassen lassen. Die Einfachheit ist bei diesem Deployment-Werkzeug im Vergleich zu den Anderen am Schlechtesten, da es keine intuitive Benutzeroberfläche gibt und bereits vor der Erstnutzung erhebliche Kenntnisse über Kubernetes vorhanden sein müssen. Die Performance der Anwendung ist ansonsten gut und es war stets erreichbar. Positiv anzumerken ist die Verknüpfbarkeit mit anderen Tools, welche zum Beispiel ein Web-UI anbinden lassen.

5.3.2 ArgoCD

Das zweite Deployment-Werkzeug dieser experimentellen Evaluation ist ArgoCD. Anders als die anderen Tools wird dieses nicht auf einem Windows, macOS oder Linux System installiert, sondern direkt auf dem Kubernetes Cluster, auf dem die zu deployende Anwendung laufen soll. Es handelt sich hier nicht um eine Lösung, welche einen Start der Anwendung antriggert, sondern einem System, welches aufpasst, falls eine Änderung passiert. Wenn also ein neuer Commit in einem Git-Projekt stattfindet, beginnt ArgoCD selbstständig mit einem Redeployment, ohne dass der User dieses manuell starten muss. Dies bringt die Einschränkung mit sich, dass die Software nur auf einem Kubernetes Cluster laufen kann und das Projekt in einem Git-Repository liegen muss und vor der Installation bereits ein solcher Cluster auf dem System läuft. Konfiguriert wird auch hier über YAML-Dateien, allerdings werden diese nicht selber von der

Software angelegt, sind dafür aber auch deutlich übersichtlicher und in keine unübersichtliche Ordnerstruktur eingebunden. Die Deployment-Dauer hängt hier von der eingestellten Zeit ab, in die ArgoCD nach Änderungen im Repository schaut. In der Regel sind dies alle drei Minuten, wobei dies auch so umgewandelt werden kann, dass die Software mittels Event-Handler über eine Neuerung des Codes informiert wird. Wenn der Prozess des Deployments gestartet ist, verhält sich die Deployment-Dauer ähnlich wie die anderen Cluster-Lösungen. Diese sind in der Regel schneller deployed als die Pipeline-Varianten, was positiv anzumerken ist. Das Logging ist in der Web-UI konfigurierbar und kann sich zu jedem Service einzeln angeschaut werden, was als sehr gut zu bewerten ist. Das Testen von Software kann über eine einfache Anpassung der YAML-Dateien erfolgen. Hier wird lediglich die Test-Schritte aufgeführt. Anschließend werde diese, wie eine normale Anwendung, im Kubernetes Cluster ausgeführt, allerdings können diese nicht gesondert als Tests aufgeführt werden. Ein Ressourcen-Management, kann über die Veränderung des Kubernetes Cluster oder der Konfigurationsdatei der Anwendung erfolgen. Dies ist abhängig davon, ob der Cluster noch mehr Möglichkeiten hat oder ob diese bereits ausgereizt sind. Diese Daten sind allerdings über die Übersicht von ArgoCD einsehbar. Ähnlich sieht es mit der Skalierbarkeit aus. In die zugehörigen Dateien können Angaben zur maximalen Speicherverfügbarkeit oder der CPU vorgenommen werden, weshalb die Skalierbarkeit, nach einer Veränderung der Dateien und einem Redeployment, gegeben ist. Die Benutzeroberfläche ist intuitiv bedienbar und stellt die meisten wichtigen Informationen zur Verfügung. Zusätzlich wären Informationen über die Auslastung des Clusters von Vorteil. Die Stabilität der Software war zu jeder Zeit gegeben, es gab keine Fehler die aufgetreten sind. Auch die Performance war jederzeit gut, sodass die Seiten der Weboberfläche stets zügig geladen haben. ArgoCD lässt sich aufgrund seiner besonderen Installationsweise gut in andere Software integrieren. Es gibt bereits einige Lösungen (darunter auch Codefresh), die dies nutzen und in ihre integriert haben.

5.3.3 Rancher

Die dritte und letzte Clusterlösung in dieser Evaluation ist Rancher. Es bietet die ausführlichste Web-Oberfläche an, über die die meisten Funktionen gesteuert werden können. Die Software wird auf Docker installiert, ist daher am besten für eine Installation auf Linux geeignet, allerdings stellen auch andere Betriebssysteme kein großes Problem dar, wenn Docker Desktop

vorhanden ist. Nach erfolgreichem initialem Ausführen der Anwendung, kann das Web-Interface aufgerufen werden. Hierüber können neue Kubernetes Cluster erstellt werden, indem einfach die entsprechende Konfigurationsdatei oder über die Schaltflächen ein beliebiges Cluster angeklickt und anschließend selbstständig von Rancher erstellt wird. Die Deployment-Dauer verhält sich ähnlich zu den anderen beiden Cluster-Werkzeugen, da im Endeffekt dieselben Befehle ausgeführt werden und das Gleiche im Hintergrund passiert. Zu erkennen ist dies daran, dass sich die Konfigurationsdatei im Aufbau sehr ähneln, mit dem Unterschied, dass hier keine extra Datei angelegt werden muss. Auch diese wird im Hintergrund automatisch, je nach Eingaben des Benutzers erstellt. Des Weiteren bedarf auch die Skalierbarkeit keiner separaten Lösung, da sich die verbrauchten und verfügbaren für jeden Knoten separat anzeigen lassen und sie durch einfaches Drücken von plus oder minus skaliert werden können, sodass eine optimale Anpassung möglich ist. Allerdings besteht hier der entscheidende Unterschied, dass ein Redeployment manuell angetriggert werden muss. Rancher bietet wie ArgoCD umfangreiche Logging-Ausgaben an, welche auch hier für jeden Service einzeln angepasst und eingesehen werden können, somit ist die Logqualität sehr gut. Das Testen funktioniert, wie bei der zuvor getesteten Software, über das Erstellen einer Extrakonfiguration. Eine Testintegration selber ist nicht vorhanden. Als Einzige, der bislang genauer betrachteten Lösungen, bietet Rancher die Möglichkeit ein Backup der Konfigurationen, sowie der Daten, über das Web-UI anzulegen und dies an einem beliebigen Ort zu speichern, was einen Persistenz Vorteil mit sich bringt. Die Stabilität und die Performance der Software waren permanent gegeben, sodass hier keine Unterschiede im Vergleich zu den anderen Anwendungen aufgefallen sind. Umso mehr stach allerdings die gut bedienbare Benutzeroberfläche heraus. Es sind die mit am Abstand wenigsten Kubernetes oder Docker Kenntnisse nötig, da viele Eingaben direkt über das Webinterface ausgewählt werden können. Das Anlegen von YAML-Dateien ist nicht mehr nötig, sodass selbst Neulingen ein schneller Einstieg gelingt.

5.3.4 Jenkins

Anders als die zuvor getestete Software, handelt es sich bei Jenkins um eine klassische Pipeline-Lösung mit einer umfangreichen Benutzeroberfläche. Zunächst unterscheidet sie sich im Vergleich zu vorher getesteten Variante in der Installation. Es handelt sich bei der Installation um eine klassische Softwareinstallation, welche auf allen gängigen Betriebssystemen möglich

ist. Jenkins startet nach dieser auf einem zuvor ausgewählten Port als Services und kann so über die Einstellungen des Computers gestartet oder gestoppt werden. Es werden keine Fremdsoftware wie Kubernetes oder Docker oder andere Vorkenntnisse benötigt. Jedoch ähnelt der weitere Verlauf der Konfiguration, der Rancher-Lösung. Ähnlich wie dort, kann auch der Source-Code einer beliebigen Versionskontrollen-Software importiert werden. Das Starten dieser Anwendung kann anschließend entweder über das Web-Interface erfolgen, in dem die benötigten Schritte auf den Schaltflächen ausgewählt werden, oder es erfolgt über ein klassisches Jenkinsfile. Ein Jenkinsfile ähnelt dem Aufbau einer YAML-Datei und definiert die Pipeline. Es handelt sich also viel mehr um eine Konfigurationsdatei, welche eine Schritt-für-Schritt-Anleitung für den Jenkins-Server beinhaltet, damit er weiß, wie die Software deployed oder ausgeführt werden soll. Dies ist anders als bei den Kubernetes-Lösungen nicht nur auf einem Cluster, sondern auch klassisch auf einer virtuellen Maschine oder in Docker möglich. Die Deployment-Dauer ist bei Pipelines allgemein länger als bei Kubernetes Deployments. Allerdings ist dies noch immer nicht langsam und zu vernachlässigen, da es sich nur um wenige Sekunden handelt. Ein größerer Unterschied zeigt sich im Logging. Erstmals handelt es sich bei Jenkins um zentrale Ausgaben, die alle Meldungen der Services auf einer Seite darstellen. Ein getrenntes Anschauen der Log-Ausgaben ist hier nicht möglich, weshalb man bei größeren Anwendungen mit deutlich mehr Microservices, trotz unterschiedlicher Farben, schnell den Überblick verlieren kann. Aus diesem Grund ist auch die Log-Qualität nicht so gut zu bewerten, wie es bei zwei der drei Kubernetes Anwendungen möglich war. Das Testen erfolgt ebenfalls einem ähnlichen Prinzip, da hier parallel eine Testpipeline angelegt wird, in der die nötigen Tests ausgeführt werden. Die Unterscheidung zu anderen Pipelines erfolgt nach Namenskonventionen derer Benennung. Ein Ressourcenmanagement ist über eine solche Lösung nicht möglich, da sie lediglich ein Deployment auf dem Zielsystem an triggert, jedoch nicht auf dieses schauen oder Ressourcen abfragen kann. Aus diesem Grund ist auch die Skalierbarkeit nur mit einer Veränderung der Pipeline möglich. Diese muss dann lediglich neugestartet werden und das System wird neu deployed. Auch Jenkins läuft stets stabil und verfügt über die intuitivste Oberfläche. Jeder Schaltfläche befindet sich dort, wo man sie erwartet und die Einstellungsmöglichkeiten sind hier am Größten. Allerdings ist das Webinterface im Vergleich zu den Anderen auch am trägsten, aber keines Weges langsam. Nur im direkten Vergleich fällt der Unterschied zu den übrigen Anwendungen auf.

5.3.5 GoCD

Das Letzte, in dieser experimentellen Evaluation, zu betrachtende Tool ist GoCD. Es ist Jenkins sehr ähnlich und hat auf den ersten Blick keine erkennbaren Unterschiede, allerdings unterscheiden sich die Lösungen zum Beispiel beim Anlegen einer Pipeline. Hier kann ebenfalls mit dem Anlegen einer Konfiguration oder mit einer Datei gearbeitet werden. Allerdings handelt es sich um Konfiguration mit dem bekannten YAML-Format. Des Weiteren können die Log-Ausgaben hier näher konfiguriert und den eigenen Bedürfnissen angepasst werden. Allerdings erscheinen diese ebenfalls in einem zentralem Logging und können nicht einzeln dargestellt werden. Das Testen erfolgt auch hier über eine parallele Pipeline, welche gesondert angelegt werden muss. Die Einfachheit der Bedienung ähnelt der von Jenkins und ist vom Benutzer abhängig. Beide bieten in etwa denselben Funktionsumfang, sodass je nach optischer Präferenz unterschieden werden muss. Lediglich die etwas schnellere Performance von GoCD fällt im direkten Vergleich zu Jenkins auf. Für die Bedienbarkeit macht dies aber keinen Unterschied.

6 Bewertung

Die Bewertung der Deployment-Werkzeuge muss zunächst in zwei Kategorien aufgeteilt werden. Die erste Kategorie besteht aus den Tools für ein Deployments auf einem Kubernetes Cluster, die zweite aus den Pipeline-Lösungen.

Die Kubernetes Anwendungen unterscheiden sich in ihrer Funktionsweise stark voneinander und können daher gut miteinander verglichen werden. Das schmalgewichtige, aber auch komplizierteste Tool ist Jujy. Es hat für User mit einem großen Kubernetes Know-How den Vorteil, dass es sich schnell und effizient über die Kommandozeile bedienen lässt, wenn man weiß, was man tut. In dieser Evaluation konnte es allerdings nicht mit den anderen Anwendungen mithalten, da die Nachteile, besonders für Nicht-Experten, überwiegen. ArgoCD hingegen bietet eine sehr gute Basis und überzeugt durch seine Methode, ein Deployment durchzuführen

und ist aus diesem Grund auch nicht mit Rancher vergleichbar, da dies mehr wie eine Pipeline, abgestimmt auf Kubernetes Cluster, arbeitet. Beide Tools können gut in kommerziellem Gebrauch eingesetzt werden und sind eine gute Wahl, allerdings kann die Wahl für ein Deployment-Werkzeug hier nur nach dem Anwendungszweck gefällt werden. Allerdings bietet ArgoCD die Möglichkeit, eine Pipeline-Lösung mit dieser zu verknüpfen, was dieses Tool erneut interessant macht.

Die zwei betrachteten Pipeline-Lösungen unterscheiden sich in ihrer Funktionsweise kaum. Beide erfüllen ihren Zweck und stellen eine gute Wahl dar. Die Wahl für eines dieser beiden Anwendungen ist abhängig vom Geschmack des Kunden. Beide Tools können probeweise getestet und anschließend das ansprechendere Werkzeug ausgewählt werden.

Es gibt keinen eindeutigen Sieger in dieser Evaluation. Allerdings ist Rancher die vollkommenste Cluster-Lösung die aktuell auf dem Softwaremarkt zu finden ist. ArgoCD und Juju sind auch gute Tools, die ihren Zweck erfüllen, jedoch auch deutlich mehr Know-How und Verständnis im Umgang mit Kubernetes erfordern und die Funktionen des Deployment-Lifecycles nicht in dem Umfang abdecken, wie es Rancher in der Lage ist zu tun. Die Pipeline-Lösungen Jenkins und GoCD sind gleichermaßen empfehlenswert und können nach persönlicher Präferenz ausgewählt werden.

7 Fazit und Ausblick

Viele Unternehmen stehen mit dem Umbruch der Softwareentwicklung vor der Frage, wie sie ihre Prozesse vereinfachen und so Kosten und Arbeit sparen können. In dieser Bachelorarbeit sollte ein optimales Deployment-Werkzeug für den Großteil von Unternehmen evaluiert werden, welches die Möglichkeit bietet den gesamten Deployment-Lifecycle abzudecken.

Nach Klärung der Begrifflichkeiten wurden mithilfe der Goal-Question-Metric-Methode zunächst Metriken für eine konzeptionelle Evaluation aufgestellt. Diese theoretische Untersuchung sollte die Auswahl der möglichen Deployment-Tools so einschränken, dass eine anschließende experimentelle Evaluation durchgeführt werden kann. Dabei zeigten sich besonders große Unterschiede im Umgang mit der Begrifflichkeit „Open-Source“. Von den vermeintlichen vielen kostenfreien Werkzeugen sind ein Großteil nur zu einem kleinen Teil Open-Source, woraufhin viele Anwendungen aus der Evaluation ausgeschlossen werden konnten. Des Weiteren zeigten sich zwei große Gruppierungen von Deployment-Tools. Hierbei handelte es sich um Pipeline- oder Clusterlösungen. Aus beiden Gruppierungen konnten Werkzeuge in der experimentellen Evaluation genauer belichtet werden.

Das Ergebnis dieser praktischen Untersuchung hat neue große Unterschiede aufzeigen können, die zwischen den Lösungen existieren. Lediglich zwei Pipeline-Anwendungen ähneln sich so stark, dass keine großen Unterschiede herausgearbeitet werden konnten. Die Arbeit mit Clustern ist im Vergleich zu Pipelines verhältnismäßig neu, weshalb noch keine klare Linie in dem Umgang mit dessen Deployment zu erkennen ist. Aus diesem Grund unterscheiden sich diese stärker voneinander, als man es bei Jenkins und GoCD gesehen hat.

Das Ergebnis dieser Evaluationen hat gezeigt, dass es kein bestes Deployment-Werkzeug für jeden Anwendungszweck gibt. Der Anwender muss zunächst selber für sich bestimmen, ob eine Cluster- oder eine Pipeline-Lösung das für seinen Anwendungszweck richtige Tooling darstellt. Daraufhin kann eine Entscheidung zwischen Jenkins und GoCD oder Rancher getroffen werden. Bei tiefgründigem Verständnis wird außerdem die Wahl von ArgoCD empfohlen.

7.1 Ausblick

Diese Arbeit gibt einen kleinen Einblick in die zahlreichen Deployment-Werkzeuge, welche zumindest zu einem Teil Open-Source sind. Insbesondere der konzeptionelle Teil ist recht oberflächlich, da eine Evaluation von so vielen Tools im Rahmen einer Bachelorarbeit nicht tiefgründiger durchgeführt werden kann. Bei weiteren Untersuchung können hier einige Deployment-Werkzeuge genauer belichtet und unter einander verglichen werden. Dies ist dann sinnvoll, wenn besonderes unbekannteren Tool mehr Aufmerksamkeit gegeben werden soll.

Auch die Experimentelle kann in einem größeren Rahmen einer genaueren Untersuchung unterzogen werden, indem zum Beispiel Lasttests oder dauerhafte Einsätze durchgeführt werden.

Literaturverzeichnis

Amazon Web Services, 2022. *AWS CodeDeploy*. [Online]
Available at: <https://aws.amazon.com/de/codedeploy/>
[Zugriff am 19 August 2022].

Apache, 2004. *Apache License, Version 2.0*. [Online]
Available at: <https://www.apache.org/licenses/LICENSE-2.0>
[Zugriff am 19 August 2022].

AppVeyor, 2022. *AppVeyor - CI/CD service for Windows, Linux and macOS*. [Online]
Available at: <https://www.appveyor.com/>
[Zugriff am 20 August 2022].

Argo CD, 2022. *Argo CD - Declarative GitOps CD for Kubernetes*. [Online]
Available at: <https://argo-cd.readthedocs.io/en/stable/>
[Zugriff am 13 August 2022].

Balzer, L., Frey, A. & Nenniger, P., 1999. Was ist und wie funktioniert Evaluation.. *Empirische Pädagogik*, 13(4), pp. 393-413.

Bauwens, J. et al., 2020. Over-the-Air Software Updates in the Internet of Things: An Overview of Key Principles. *IEEE Communications Magazine*, Februar, 58(2), pp. 35-41.

Buildbot, kein Datum *Buildbot*. [Online]
Available at: <https://buildbot.net/>
[Zugriff am 2 August 2022].

Capistrano, 2022. *Capistrano*. [Online]
Available at: <https://capistranorb.com/>
[Zugriff am 13 August 2022].

Circle Internet Services, Inc., 2022. *CircleCi*. [Online]
Available at: <https://circleci.com/>
[Zugriff am 10 August 2022].

Codefresh, 2022. *Codefresh*. [Online]
Available at: <https://codefresh.io/>
[Zugriff am 19 August 2022].

Dearle, A., 2007. Software deployment, past, present and future. *Future of Software Engineering (FOSE'07)*, pp. 269-284.

Deployer, 2022. *Deployer*. [Online]
Available at: <https://deployer.org/>
[Zugriff am 17 August 2022].

Docker Inc., 2022. *Overview of Docker Compose*. [Online]
Available at: <https://docs.docker.com/compose/>
[Zugriff am 14 August 2022].

GitLab, 2022. *About GitLab*. [Online]
Available at: <https://about.gitlab.com/>
[Zugriff am 17 August 2022].

GNU, 2022. *GNU - Why the Affero GPL*. [Online]
Available at: <https://www.gnu.org/licenses/why-affero-gpl.html.en>
[Zugriff am 19 August 2022].

GoCD, 2022. *GoCD - free & open source ci/cd server*. [Online]
Available at: <https://www.gocd.org/>
[Zugriff am 15 August 2022].

Harness Inc., 2022. *The Modern Software Delivery Platform™*. [Online]
Available at: <https://harness.io/>
[Zugriff am 16 August 2022].

Heinrich, B., Linke, P. & Glöckler, M., 2020. Grundlagen zur Automatisierung. In: *Grundlagen Automatisierung*. Wiesbaden: Springer Vieweg, pp. 1-29.

Humble, J. & Farley, D., 2010. *Continuous delivery: reliable software releases through build, test, and deployment automation*. s.l.:Pearson Education.

Jaeger, T. & Metzger, A., 2006. *Open Source Software: Rechtliche Rahmenbedingungen der Freien Software*. 2. Hrsg. s.l.:s.n.

Jenkins, 2022. *Jenkins*. [Online]
Available at: <https://www.jenkins.io/>
[Zugriff am 4 August 2022].

JetBrains s.r.o., 2022. *TeamCity*. [Online]
Available at: <https://www.jetbrains.com/de-de/teamcity/>
[Zugriff am 18 August 2022].

JFrog, 2022. *JFrog*. [Online]
Available at: <https://jfrog.com/>
[Zugriff am 22 August 2022].

Juju, 2022. *Juju*. [Online]
Available at: <https://juju.is/>
[Zugriff am 19 August 2022].

Kelter, U., 2006. Evaluation des Lehrangebots eines Fachbereichs–Einsatzbedingungen und Evaluationskriterien für unterstützende Software..

Kugele, S., Hettler, D. & Peter, J., 2018. Data-Centric Communication and Containerization for Future Automotive Software Architectures. *2018 IEEE International Conference on Software Architecture (ICSA)*, pp. 65-6509.

Kühl, S., 2009. Experiment. In: *Handbuch Methoden der Organisationsforschung*. s.l.:VS Verlag für Sozialwissenschaften, pp. 534-557.

MIT, kein Datum *MIT License.* [Online]
Available at: <https://choosealicense.com/licenses/mit/>
[Zugriff am 19 August 2022].

Mouat, A., 2015. *Using Docker: Developing and Deploying Software with Containers.* s.l.:O'Reilly Media, Inc..

Newman, S., 2015. *Microservices: Konzeption und Design.* s.l.:MITP-Verlags GmbH & Co. KG.

PagerDuty, Inc., 2022. *Rundeck.* [Online]
Available at: <https://www.rundeck.com/>
[Zugriff am 21 August 2022].

Perens, B., 1999. The open source definition. In: *Open sources: voices from the open source revolution.* s.l.:s.n., pp. 171-188.

Picot, A. & Fiedler, M., 2008. Open Source Software und proprietäre Software. In: O. Deppenheuer & K. Pfeifer, Hrsg. *Geistiges Eigentum: Schutzrecht oder Ausbeutungstitel?.* Berlin, Heidelberg: Springer, pp. 165-185.

Rancher, 2022. *Rancher.* [Online]
Available at: <https://www.rancher.com/>
[Zugriff am 4 August 2022].

Redaktion Open Telekom Cloud, 2020. *Container oder VM – was ist für Ihr Projekt besser geeignet?.* [Online]
Available at: <https://open-telekom-cloud.com/Resources/Persistent/b/6/7/2/b6725b1b17fd6b7bbe5f356ecd62e36e606fd43e/open-telekom-cloud-blog-container-vs-vm-799x377.webp>
[Zugriff am 19 August 2022].

Schneider, O., 2020. *Microservices einfach nutzen mit der brudi Platform.* [Online]
Available at: <https://blog.brudi.com/microservices-mit-brudi-platform/>
[Zugriff am 19 Juli 2022].

Spinnaker, 2022. *Spinnaker - Cloud Native Continuous Delivery*. [Online] Available at: <https://spinnaker.io/> [Zugriff am 16 August 2022].

Witte , F., 2018. Definition, Historie und Nutzen von Metriken. In: *Metriken für das Testreporting*. Wiesbaden: Springer Vieweg.

Wolff, E., 2018. *Microservices: Grundlagen flexibler Softwarearchitekturen*. s.l.:dpunkt.verlag.


Wolff, E., Ebbinghaus, M. & B., S., 2022. *Microservice Kubernetes Sample*. s.l.:s.n.

Wottawa, H. & Thierau, H., 1998. *Lehrbuch Evaluation*. s.l.:Verlag Hans Huber.

Zhou, X. et al., 2019. Latent error prediction and fault localization for microservice applications by learning from system trace logs. *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, August, pp. 683-694.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort Datum  Unterschrift im Original