HAW HAMBURG

Salesforce DevOps-Strategien

Continuous Integration, Delivery und Deployment in der Praxis

Bachelor-Thesis zur Erlangung des akademischen Grades B.Sc.

im Studiengang Media Systems

Studierender: Max Kretzschmar

Matrikelnummer: [redacted]

Erstprüfer: Prof. Nils Martini

Zweitprüfer: Philip Czupras

Hamburg, 9. März 2020

Hochschule für Angewandte Wissenschaften Hamburg

Fakultät Design, Medien und Information

Zusammenfassung

Der Begriff DevOps ist in der Softwareentwicklung lange kein Fremdwort mehr. Die Entwicklungsinfrastruktur von Salesforce entwickelte sich hingegen nur sehr langsam in eine Richtung, in der moderne Methoden bei der Entwicklung und Bereitstellung von Code zum Einsatz kommen konnten.

Das Ziel dieser Arbeit ist es herauszufinden, wie die Grundsätze, die der DevOps-Philosophie zugrunde liegen, mit der Salesforce-Entwicklungsumgebung verbunden werden können. Dazu wird zunächst untersucht, was der Begriff DevOps im Kern bedeutet. Die Prinzipien der Kontinuität bei der Bereitstellung von Code, welche mit DevOps einhergehen, werden dazu ausführlich analysiert. Zudem werden neue Methoden der Entwicklung und Bereitstellung von benutzerdefinierten Anwendungen in Salesforce mit Salesforce DX und des Salesforce Command Line Interfaces betrachtet und in den Kontext zu DevOps gebracht. Auf Basis der theoretischen Betrachtungen wird ein neues quellengesteuertes Entwicklungsmodell aufgebaut.

Durch den schrittweisen Aufbau einer Automationskette wurde dieses Modell in einer Salesforce-Partner-Agentur umgesetzt. Die Verbindung neuer Entwicklungs- und Bereitstellungsmethoden mit einer Prozesskette, welche grundlegende Befehle automatisch ausführt, steigert sowohl die Sicherheit der Kommunikation der Systeme untereinander als auch die Effizienz bei der Arbeit. Besonders Entwicklerteams können mit diesem neuen Entwicklungsmodell effektiver an großen Projekten arbeiten und so den gesamten Prozess von der Entwicklung über das Testen bis zur Auslieferung stärker kontrollieren, wodurch Fehler schneller erkannt und behoben werden können.

Abstract

The term DevOps is no longer a foreign word in software development. However, Salesforce's development infrastructure has been slow to evolve in a direction where modern methods could be used to develop and deploy code.

The goal of this paper is to find out how the principles underlying the De-vOps philosophy can be linked to the Salesforce development environment. To do this, we will first examine what the term DevOps essentially means. The principles of continuity in code delivery associated with DevOps are analyzed in detail. In addition, new methods of developing and deploying custom applications in Salesforce with Salesforce DX and the Salesforce Command Line Interface are considered and put into context with DevOps. Based on the theoretical considerations, a new source-driven development model is built.

By building up an automation chain step by step, this model was implemented in a Salesforce partner agency. The combination of new development and deployment methods with a process chain that automatically executes basic commands increases both the security of communication between the systems and the efficiency at work. With this new development model, development teams in particular can work more effectively on large projects and thus have greater control over the entire process from development to testing and delivery, enabling errors to be detected and corrected more quickly.

Inhaltsverzeichnis

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Bachelorarbeit unterstützt und motiviert haben.

Zuerst gebührt mein Dank Philip Czupras, der mich zu diesem Thema ermutigt und mir die Arbeit daran erst ermöglicht hat. Für die jahrelange Unterstützung bei meiner Entwicklung in der Agentur möchte ich mich herzlich bedanken.

Ein besonderer Dank gilt meinen Kollegen bei hanseflow, die mich moralisch unterstützt und motiviert haben.

Außerdem möchte ich Dennis Schöler für das Korrekturlesen meiner Bachelorarbeit danken.

Abschließend möchte ich mich bei meinen Eltern bedanken, die mir mein Studium und die Zeit in Hamburg durch ihre Unterstützung ermöglicht haben und stets ein offenes Ohr für mich hatten.

Max Kretzschmar

Hamburg, 9. März 2020

Abkürzungsverzeichnis

AWS ..... *Amazon Web Services*
CD ..... *Continuous Delivery*
CF ..... *Continuous Feedback*
CI ..... *Continuous Integration*
CLI ..... *Command Line Interface*
CRM ..... *Customer Relationship Management*
DevOps ..... *Developer and Operations*
DX ..... *Developer Experience*
IaaS ..... *Infrastructure-as-a-Service*
IDE ..... *Integrated Development Environment*
JSON ..... *JavaScript Object Notation*
JWT ..... *JSON Web Token*
Org. ..... *Organisation*
PaaS ..... *Platform-as-a-Service*
PM ..... *Projektmanagement*
SaaS ..... *Software-as-a-Service*
SCM ..... *Software Configuration Management*
SOT ..... *Source of Truth*
VCS ..... *Version Control System*
VS Code ..... *Visual Studio Code*
XML ..... *Extensible Markup Language*

1. Einleitung

> *„DevOps ist eher eine Philosophie als eine einfache Vorgehensweise, denn es beschäftigt sich nicht mit einer bestimmten Methode oder Technologie. […] Vorrangiges Ziel ist in diesem Fall vor allem das Zusammenwachsen der Entwickler (Developer) mit dem IT-Betrieb (Operations). Die DevOps-Bewegung soll also sicherstellen, dass Developer nicht an den Administratoren vorbei entwickeln." (Dev-Insider, 2017, S. 3)*

Im Jahr 2009 organisierte Patrick Debois unter dem Namen „DevOpsDays" eine Konferenz in Belgien, welche sich vorrangig auf die Zusammenarbeit zwischen Programmierern, Systemadministratoren und dem IT-Betrieb konzentrieren sollte. Er erkannte, dass es bei der Zusammenarbeit der einzelnen Bereiche zu Diskrepanzen während der Software-Entwicklung kommen kann und machte es sich zur Aufgabe, Strategien zu entwickeln, die diese Probleme lösen. Patrick erschuf den Begriff *DevOps*, der sich aus den englischen Wörtern *Development* und *Operations* – also Entwicklung und Betrieb – zusammensetzt und legte damit den Grundstein der darauffolgenden Bewegung, bei welcher Debois zu einer tragenden Figur wurde (New Relic, kein Datum). Innerhalb der letzten Jahre stieg das Interesse an DevOps deutlich, wie eine Google Trends-Suche zu dem Begriff verdeutlicht (siehe Anhang 1). Damit kann angenommen werden, dass die Ansätze, die unter dem Begriff zusammengefasst werden, an Relevanz gewonnen haben.

„*In practice, DevOps is not something you're ever ‚done with'*." (Davis, 2019, S. 28). DevOps ist kein Prozess oder einheitlicher Standard, vielmehr ist es eine Philosophie bei der Softwareentwicklung. In der Praxis werden unter dem Begriff Ansätze und Strategien zusammengefasst, welche die Zusammenarbeit zwischen Programmierern und dem IT-Betrieb bei der Entwicklung von Software verbessern. Der gesamte Prozess – von der Planung über die Entwicklung bis zur Auslieferung – wird mit dem DevOps-Ansatz mithilfe einer durchgängigen Prozess- und Werkzeugkette realisiert (Dev-Insider, 2017). Durch diese Prozesskette werden Software-Entwicklungszyklen verkürzt und Fehler bei der Implementierung verringert. Die ausgelieferte Software wird fortwährend überwacht.

Entstehen neue Anforderungen, so können Arbeitspakete sehr genau geplant und bearbeitet werden. Besonders die agile Software-Entwicklung, die durch eine enge Zusammenarbeit zwischen der Produktentwicklung und dem Kunden geprägt ist, wird durch DevOps sinnvoll erweitert, da Wünsche und Bedenken des Kunden nach der Auslieferung als Hauptanliegen wieder in neue Entwicklungszyklen einbezogen werden und damit die Basis für weitere Zyklen bilden.

Seit der Begründung der DevOps-Bewegung hat die ihr zugrunde liegende Philosophie viele Menschen erreicht. Immer mehr Unternehmen adaptieren ihre Ansätze und richten eine kontinuierliche Prozesskette ein, um Software effizient, schnell und fehlerarm zu entwickeln. Besonders im Bereich der agilen Software-Entwicklung ist der Ansatz nicht mehr wegzudenken, da er sich gut mit den Grundprinzipien der agilen Methoden verbinden lässt.

1.1. Motivation und Zielsetzung

Mit dieser Arbeit sollen zwei Ziele verfolgt werden. Erstens wird eine Prozess- und Werkzeugkette erarbeitet, um Code in eine Salesforce-Umgebung automatisiert zu implementieren. Dabei werden die einzelnen Schritte, die für eine kontinuierliche Integration und Auslieferung notwendig sind, betrachtet und es werden durch Vergleiche geeignete Werkzeuge ausgewählt. Zweitens wird unter Zuhilfenahme von DevOps-Ansätzen die Prozesskette aus dem ersten Teil in einer Hamburger Agentur etabliert, um benutzerdefinierte Geschäftslogik und Oberflächenelemente in internen Versionskontrollsystemen kontinuierlich zu integrieren und an Produktionsumgebungen kontinuierlich auszuliefern. Dabei sollen durch Skripts automatische Tests und Auslieferungen durchgeführt werden.

Die Hamburger Agentur hanseflow GmbH wurde Ende 2014 gegründet und verfolgt seitdem das Ziel der digitalen Transformation von Unternehmensprozessen. Seit Juni 2017 ist hanseflow Salesforce-Partner und berät Kunden aktiv bei der Umstellung der internen Prozesse. Neben Workshops und Beratungen bietet die Agentur die Implementierung von Prozessen, Migration von Daten, aber auch die Entwicklung neuer und benutzerdefinierter Geschäftslogiken sowie Oberflächen an. Das interne Projektmanagement (PM) hat sich seit der Gründung regelmäßig verändert.

Tools zum Projektmanagement und zur Zeiterfassung wie „MeisterTask“ und „Teamwork“ bildeten für lange Zeit die Grundlage des internen PMs. Vor kurzem wurde das PM auf der Atlassian Cloud neu aufgebaut. Zur Verwaltung von Arbeitspaketen, Planung von Sprints und für das Zeitmanagement wird Jira benutzt. Confluence beinhaltet interne Richtlinien, Wissensdatenbanken und Anforderungsdokumente. Ein wichtiger Schritt für die vollständige Integration der Atlassian-Dienste ist die Anbindung von Bitbucket, dem hauseigenen Versionskontrollsystem. Aktuell beschäftigt hanseflow einen internen Salesforce-Entwickler. Die Entwicklung von benutzerdefinierter Geschäftslogik und Oberflächenelementen findet in Development Sandboxen statt und die Implementierung geschieht mittels Change Set. Eine Versionskontrolle ist zudem nicht vorhanden.

Ein weiteres Ziel dieser Arbeit ist es, den gesamten Prozess der Bereitstellung (engl. *Deployment*) zu überarbeiten, indem neue und fortschrittliche Methoden der Implementierung durch Continuous Delivery (CD), Continuous Integration (CI) und automatisches Testen angewendet werden. Die Integration innerhalb des Unternehmens soll mittels Versionskontrolle als sogenannte *Source of Truth* (SOT) – also der Aufbau einer einzigen aktuellen Datenquelle - realisiert werden. Zudem sollen diese Maßnahmen den Weg für weitere Programmierer ebnen und die Zusammenarbeit erleichtern bzw. erst ermöglichen.

1.2. Vorgehensweise

Zunächst wird mit dem theoretischen Teil dieser Arbeit Grundwissen erarbeitet. Es werden die Grundprinzipien der prozessverbessernden Ansätze, welche unter dem DevOps-Begriff zusammengefasst werden, detailliert beschrieben und im nächsten Schritt auf eine Cloud-Dienstleistung übertragen. Die Herangehensweise von DevOps unterscheidet sich zwischen klassischen On-Premises-Systemen und Cloud Computing-Modellen, weswegen das Salesforce-Kernsystem, dargestellt durch SaaS und PaaS, im weiteren Verlauf ausführlich beschrieben wird. Nachdem fachliche und methodische Grundlagen im Umgang mit Salesforce und DevOps geschaffen wurden, wird der gängige Entwicklungszyklus bei der Arbeit auf der Salesforce-Plattform näher beschrieben.

Dabei liegt der Fokus auf dem Salesforce-Datenmodell sowie auf den Methoden, mit denen Änderungen programmatischer und deklarativer Natur von einer Testumgebung auf einer Produktivumgebung bereitgestellt werden.

Im zweiten Teil der Arbeit werden die im theoretischen Teil gelernten Aussagen in die Praxis übertragen. Die DevOps-Ansätze finden Anwendung in einer Agentur, die als lizensierter Salesforce-Partner Kunden bei der Digitalisierung von Geschäftsprozessen und insbesondere bei der Einführung von Salesforce als CRM-System betreut. Dabei wird der interne Entwicklungs- und Bereitstellungszyklus von benutzerdefiniertem Code und deklarativen Prozessen vollständig überarbeitet. Der bisherige Weg der Bereitstellung wird dabei als Basis für einen Vergleich mit neuen Implementierungsmethoden genutzt. Abschließend fasst diese Arbeit die Ergebnisse der Umstrukturierung des Unternehmens in einem Fazit zusammen und gibt einen Ausblick hinsichtlich der Skalierbarkeit und Produktivität.

2. Theoretische Grundlagen

Im ersten Teil der Arbeit werden die Grundlagen geschaffen, um sie im zweiten Teil in der Praxis anzuwenden. Im Besonderen werden die der DevOps-Philosophie zugrunde liegenden Aspekte sowie die neu eingeführten Ansätze zur Entwicklung und Implementierung von Code in Salesforce mit *Salesforce Developer Experience* (DX) und dem Befehlssatz für die Kommandozeile, Salesforce *Command Line Interface* (CLI) betrachtet.

2.1. Dienstleistung aus der Cloud

Die Salesforce-Kernanwendung ist ein Cloud-Dienst. Um die Unterschiede bei der Umsetzung von prozessverbessernden Ansätzen für verschiedene Systeme zu verstehen, müssen Dienstleistung, die über das Internet bereitgestellt werden, zunächst näher betrachtet werden. Grundsätzlich unterscheidet man bei der Bereitstellung von Ressourcen zwischen Systemen, die vollständig selbstverwaltet werden und solchen, die als Dienstleistung über das Internet genutzt werden. Bei sogenannten *On-Premises*-Systemen muss jedes Teilsystem eines Servers manuell eingerichtet und verwaltet werden. Dazu zählen folgende Dienste: Neben dem Server selbst mit allen notwendigen Netzwerkdiensten und erforderlichem Massenspeicher muss zusätzlich ein Betriebssystem mit Anwendungen, Middleware und Datenbanken installiert und konfiguriert werden. Die Dienste sollten je nach Anwendungsfall einen gewissen Grad an Skalierbarkeit aufweisen, da die Belastung des Systems im Zeitverlauf steigen kann. In der Regel befinden sich Server in dafür umgebauten Räumen direkt im Unternehmen oder sie werden in größeren Serverfarmen zusammengefasst. Die zweite Art der Bereitstellung von Diensten bildet das so genannte *Cloud Computing.* Der Begriff beschreibt die Bereitstellung von Ressourcen wie zum Beispiel Server, Speicher, Datenbanken, Netzwerkkomponenten usw. über das Internet (Microsoft, 2020). Unternehmen kaufen Lizenzen bei einem Cloud-Anbieter und erhalten damit Zugriff auf bestimmte Ressourcen.

Das Ziel ist u. a. die Senkung der Betriebskosten sowie die Steigerung der Effizienz. Cloud-Ressourcen haben den Vorteil, dass sie beliebig hoch skalierbar sind und im Bedarfsfall angepasst werden können, da die Kernanwendungen in der Regel auf mehreren Servern in einem Verbund installiert sind und dadurch über eine hohe Rechenleistung verfügen.

Das Cloud Computing kennt verschiedene Modelle. Das einfachste davon ist *Infrastructure-as-a-Service* (IaaS). Darunter versteht man die Bereitstellung von IT-Infrastruktur wie Server, Speicher, Betriebssysteme oder virtuelle Computer über das Internet, wofür Gebühren entsprechend der Nutzung entrichtet werden müssen. Bei *Platform-as-a-Service* (PaaS) wird im Gegensatz zu IaaS der Schwerpunkt auf die Bereitstellung einer Umgebung für die Entwicklung und Tests sowie die Bereitstellung und Verwaltung von Softwareanwendungen gelegt. Die notwendige Infrastruktur wie bspw. Server und Datenbanken, die für die Entwicklung benötigt wird, ist bereits vorkonfiguriert. *Software-as-a-Service* (SaaS) ist das Modell, welches den geringsten eigenen Konfigurationsaufwand benötigt. Dabei werden bis auf die Daten und die benutzerdefinierten Einstellungen (Prozesse o.ä.) die gesamte notwendige Infrastruktur und Software über das Internet bereitgestellt.

Besonders für kleine oder mittelständische Unternehmen stellen Cloud Computing eine effiziente und kostengünstige Alternative zu On-Premises-Systemen dar. Hierbei können die von dem Unternehmen benötigten Dienste bedarfsgerecht über das Internet in Anspruch genommen werden, ohne dass Kosten und Aufwand für ein eigenverwaltetes System veranschlagt werden müssen. Nichtsdestotrotz ist die effiziente Nutzung dieser Systeme den Unternehmen überlassen. Klassische prozessverbessernde Ansätze wie Continuous Integration und Delivery, automatisierte Tests und Überwachung müssen an diese spezielle Art der Bereitstellung von Ressourcen angepasst werden.

2.2. Die DevOps-Philosophie

DevOps ist ein Kunstwort, welches sich aus den Wörtern Development und Operations – also Entwicklung und Betrieb - zusammensetzt. Ihm liegt eine Philosophie zugrunde, nach welcher die schnelle Auslieferung von Software durch eine Firma an einen Kunden der zentrale Aspekt bei der Schaffung von Werten ist.

Das Kernthema von DevOps ist die kontinuierliche Verbesserung. Die Zusammenarbeit zwischen Entwicklung und Betrieb ist dabei ein Hauptbestandteil der Prozessverbesserung und die Motivation ist eine ständige Verbesserung sowohl bei der Dauer der Auslieferung als auch bei der Qualität und Stabilität der ausgelieferten Software. Erreicht wird das durch das Umsetzen von Continuous Integration, Continuous Delivery, automatisiertes Testen und konstantes Überprüfen (englisch Monitoring).

2.2.1. Prozess- und Werkzeugkette

Am Anfang jedes neuen Entwicklungszyklus steht die **Planung.** Anforderungen werden von dem operativen Geschäft bzw. dem IT-Betrieb zusammengetragen und als Arbeitspakete oder User Stories definiert. Ist der Zyklus nicht der erste, sondern folgt einem oder mehreren bereits abgeschlossenen Zyklen, so fließen die Erfahrungen, Änderungswünsche oder Fehler in die Planung ein. Nach der Planung folgt die eigentliche Entwicklung der Software. Während der **Erstellung** wird der Programmcode von den Entwicklerteams geschrieben. In einem festgelegten Zeitraum werden fertiggestellte Arbeitspakete oder Programmzeilen der Teams kontinuierlich in eine übergeordnete Umgebung integriert. Eine Versionskontrolle bildet hierbei die SOT. Sobald die Arbeit beendet ist, wird der Code automatisch getestet und ausgeliefert. Dieser Schritt fällt unter das **Verifizieren***,* also das automatische Testen der fertiggestellten Softwarepakete vor der Auslieferung an den Kunden. Sind alle Tests erfolgreich, kann der Code in **Paketen** zusammengeschlossen und an den Kunden ausgeliefert werden. Dieser Schritt geschieht genau wie das Testen automatisch. Nach der **Veröffentlichung** steht dem Kunden die Software zur Nutzung zur Verfügung und wird ggf. durch den IT-Betrieb zusätzlich **konfiguriert**. Während des Betriebes wird die neue Software vom Betrieb **überwacht** und es werden Fehler und Probleme dokumentiert. Nun kann durch eine ununterbrochene Prozesskette zwischen dem Betrieb und der Entwicklung auf Fehler schnell reagiert werden. Die durch den Betrieb oder die Administratoren festgestellten Fehler werden in neue Arbeitspakete übersetzt und fließen als Plan in einen neuen Entwicklungszyklus ein.

Abbildung 2 zeigt eine Illustration, wie sie oftmals für die Darstellung der DevOps-Prozesse verwendet wird. Sie verdeutlicht den unendlichen Ablauf der Teilschritte in dem Gesamtkreislauf.

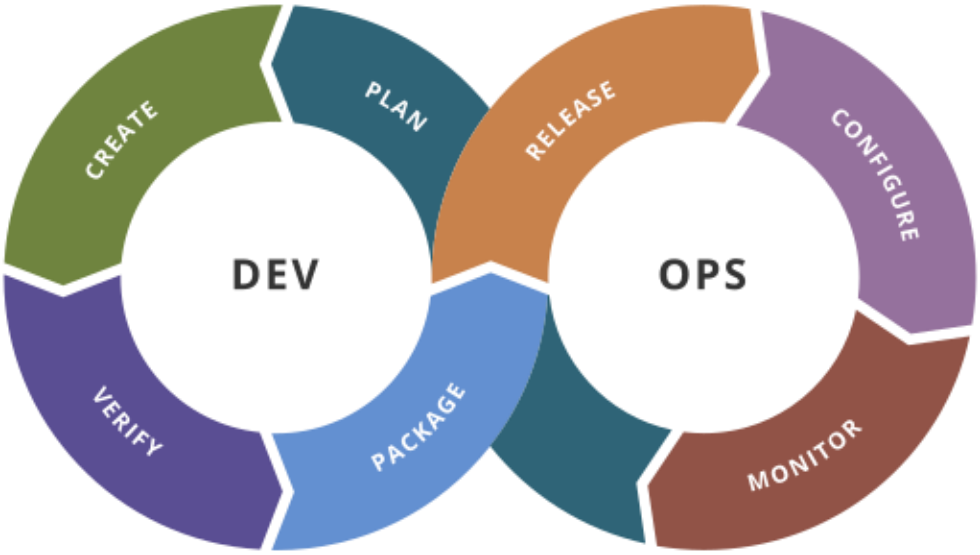


Abbildung 1: Grafische Darstellung der DevOps-Prozesskette (Kharnagy, 2016)

Hochspezialisierte DevOps-Teams können so eine schnelle Bereitstellung von neuem Code erreichen. Laut Kim, Behr und Spafford (2013) betrug die Frequenz der täglichen Bereitstellung der Firma Amazon im Jahr 2013 rund 23.000-mal pro Tag. Im Vergleich wurde unter anderem Google mit 5.500 Bereitstellungen und Facebook mit 1 Bereitstellung pro Tag aufgeführt.

2.2.2. Salesforce und DevOps

Eine Salesforce-Organisation (Org) ist die Umgebung, in der alle Datensätze, Prozesse und benutzerdefinierte Anwendungen gesammelt und visualisiert werden. Durch so genannte Komponenten wird eine individuell angepasste Oberfläche für den Kunden aufgebaut, welche die internen Prozesse optimal abbilden soll. Der Kern des Salesforce-Systems besteht aus der *Sales Cloud,* das Kern-Customer-Relationship-Management-System (CRM), aus der *Service Cloud*, welche den Kundensupport fokussiert, der *Community Cloud,* um benutzerdefinierte Weboberflächen zu erstellen und der *Lightning Plattform* (ehemals Force.com) (Davis, 2019, S. 15).

Sales, Service und Community Cloud bilden den SaaS Teil von Salesforce. Die Lightning-Plattform ist eine PaaS. Beide Systeme werden gemeinsam als große Datei in Datencentern installiert und über das Internet als Cloud-Dienst bereitgestellt. Die Systeme arbeiten in der Praxis nahtlos zusammen. Diese Zusammenarbeit wird durch eine gemeinsame Datenbank realisiert. Es ist wichtig, sich dieses Schema zu merken, da es eine der Stärken der Salesforce-Plattform ist. Sales, Service und Community Cloud sind lediglich unterschiedliche zweckbezogene, visualisierte Oberflächen basierend auf einer einzigen Organisation.

Dadurch, dass Salesforce sowohl PaaS als auch SaaS bereitstellt und damit die Konfigurierbarkeit eines Servers im Vergleich zu On-Premises-Systemen deutlich einschränkt, unterscheidet sich die Umsetzung von DevOps auf cloudbasierten Systemen merklich voneinander. Es gibt eine Reihe von Tools sowohl für das Entwickeln und Bereitstellen durch Programmierer als auch für das Warten und Aktualisieren von Infrastruktur durch Administratoren. Diese Tools sind jedoch kaum auf Salesforce anwendbar. Um diese Feststellung zu verdeutlichen, führt Dawis in seinem Buch *Mastering Salesforce DevOps* von 2019 ein Beispiel auf, in welchem dargestellt ist, wie Teams Infrastruktur mithilfe von Amazon Web Services (AWS) verwalten und eine Anwendung auf Heroku implementieren würden. Heroku ist dabei ein weiteres PaaS-Produkt, welches zu Salesforce gehört und als Host für benutzerdefinierten Code und Anwendungen für verschiedene Programmiersprachen dient und gleichzeitig verschiedene Services bereitstellt, um diese auszuführen. Amazon Web Services stellt die Struktur der AWS Dienste bereit und kann durch Dateien mit dem JavaScript Object Notation-Format (JSON) beschrieben werden. Das Format zeichnet sich dadurch aus, dass es so genannte Key-Value-Paare innerhalb der Datei bildet. Diese Paare beschreiben jeweils eine bestimmte Eigenschaft (*Key*) mit einem ihr zugeordneten Wert (*Value*). JSON kann dadurch beschreiben, welche AWS Services benutzt werden, wie diese konfiguriert sein oder in welchen Datenzentren diese laufen sollen. Werden Änderungen an der JSON-Datei und damit gleichbedeutend an den Einstellungen vorgenommen, können Tools zur kontinuierlichen Integration mithilfe des AWS CLI diese automatisch implementieren. Administratoren steht damit ein Tool zu Seite, mit dem sie Änderungen überwachen und Änderungen automatisieren können.

Ebenso kann Heroku mittels des Heroku CLI genutzt werden, um Code automatisch bei jeder Änderung zu implementieren. Durch sog. *Pipelines* wird dieser automatische Prozess geplant und visualisiert. Damit wird ein Großteil der notwendigen Konfigurierung von Diensten zur Bereitstellung und dem Betrieb von Software von der Plattform selbst übernommen. Entwicklern steht dadurch ein Werkzeug zur Verfügung, um den gesamten Entwicklungszyklus zu verwalten, ohne ein Drittanbieter-Werkzeug benutzen zu müssen. Wie in Kapitel *2.1 Dienstleistungen aus der Cloud* beschrieben, stellt ein PaaS-Modell eine Plattform zur Verfügung, bei der Code ohne notwendige Abhängigkeiten oder zusätzliche Dienste bereitgestellt und betrieben werden kann. Auf dieser Grundlage ist Heroku tatsächlich eine PaaS, da die Dienstleistung eine hohe Flexibilität und damit einen großen Spielraum bei der Art des Codes und der Anwendung einräumt. Die Lightning-Plattform hingegen verhält sich trotz dessen, dass sie auch eine PaaS ist, in der Praxis im Vergleich zu Heroku anders. Auf der Plattform können sowohl server- als auch clientseitige Anwendungen geschrieben werden, allerdings laufen sie nur innerhalb einer Salesforce Org. Des Weiteren sind benutzerdefinierte Felder, Entitäten sowie Beziehungen ebenfalls nur in Salesforce betreibbar. Alle Änderungen, die basierend auf der Lightning-Plattform getätigt werden, können nicht auf andere Datenbanken übertragen werden.

Zusammengefasst bedeutet das, dass Werkzeuge, mit denen andere IaaS und PaaS verwaltet werden, nicht innerhalb des Salesforce-Systems verwendet werden können. Allerdings gibt es mit Salesforce DX die Möglichkeit, Techniken und Prinzipien der Verwaltung und Bereitstellung auf Salesforce zu übertragen (Davis, 2019, S. 18).

2.2.3. Continuous Delivery, Feedback und Improvement

Gene Kim, Autor von „The Phoenix Project“ und „The DevOps Handbook“, erklärt in seinem Artikel „The Three Ways: The Principles Underpinning DevOps“ drei Prinzipien, aus welchen die grundlegenden DevOps-Muster abgeleitet werden können. Wie in Kapitel *2.2.1. Prozess- und Werkzeugkette* bereits beschrieben, besteht DevOps in der Praxis aus einer kontinuierlichen Kette aus ineinandergreifenden Teilprozessen, welche Entwickler und den Betrieb miteinander verbinden.

Die diesen Teilprozessen zugrundeliegenden Prinzipien werden in drei eigenständigen Denkmustern zusammengefasst. Charakteristisch für diese Denkmuster ist ihre Flussrichtung, also der Verlauf der Arbeit in eine bestimmte Richtung. Abbildung 2 illustriert diese Arbeitsabläufe durch farblich gekennzeichnete Pfeile.

Die erste Flussrichtung ist „von links nach rechts“. **Continuous Delivery** konzentriert sich auf die reine Wertschöpfung einer Software, welche von Entwicklern erstellt, der Qualitätssicherung geprüft und durch den Betrieb an den Kunden ausgeliefert wird.

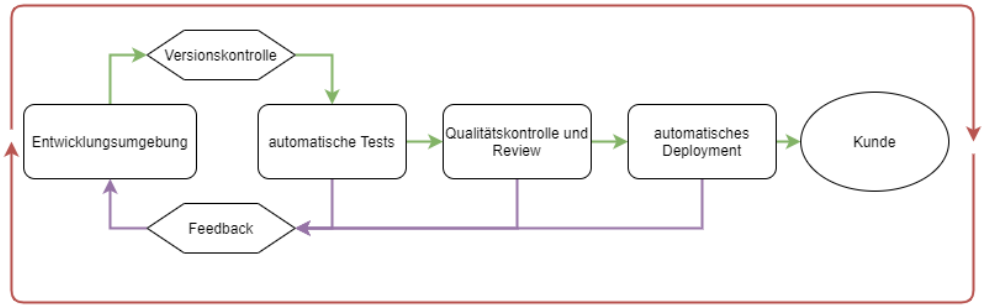


Abbildung 2: Diagramm der 3 Flussrichtungen von DevOps (Eigene Darstellung)

In Abbildung 2 wird dieser Ablauf durch einen grünen Pfeil dargestellt. Deutlich zu erkennen ist hierbei die Nutzung der Versionskontrolle, welche einen von zwei zugrunde liegenden Mechanismen bildet. Fertige Arbeit wird kontinuierlich mit einem Versionskontrollsystem (engl. *Version Control System*, kurz VCS) abgeglichen, wodurch Änderungen an der Software nachverfolgt und Konflikte innerhalb eines bestimmten Abschnittes des Codes abgefangen und gelöst werden können. Ein VCS ist weiterhin essenziell für DevOps, da mithilfe von automatisierten Skripts auf den Code zugegriffen werden kann, was wiederum automatische Tests und automatisches Deployment erst ermöglicht. Der zweite Mechanismus, welcher bei Continuous Delivery zum Einsatz kommt und durch ein VCS realisierbar wird, ist Continuous Integration. Während der Entwicklungsarbeit entsteht durch die Zuarbeit der Entwicklerteams im VCS eine Codebasis, welche nach Beendigung der Arbeiten die Software an sich darstellt. CI bedeutet, dass verschiedene Teams ihre Teilarbeiten kontinuierlich an das VCS übergeben, welches diese automatisch testet.

Dieser Arbeitsablauf stellt sicher, dass Fehlerquellen in Teilcodeabschnitten früh erkannt werden und nicht in die gemeinsame Codebasis einfließen. Ebenso sorgt CI dafür, dass die Codebasis, auf deren Grundlage die Entwicklerteams arbeiten, zwischen allen beteiligten Entwicklern auf einem aktuellen Stand gehalten wird. Realisiert wird das durch CI-Tools wie beispielsweise Jenkins, welches automatische Aktionen auf einer geplanten Basis durchführt oder durch Änderungen am Code ausgelöst wird. Automatische Benachrichtigungen sorgen dafür, dass die Entwickler ständiges Feedback über ihre Arbeit erhalten und schnell reagieren können.

Die Qualität des Codes kann durch die frühestmögliche Entdeckung von Fehlern deutlich gesteigert werden, da Fehler im Verlauf der Entwicklungsarbeiten durch ständige Erweiterung der Codebasis zunehmend schwieriger auffindbar und damit schlechter behebbar sind. Continuous Integration ist seit vielen Jahren ein bewährtes Verfahren in der Softwareentwicklung, da es im Besonderen die Arbeitsabläufe zwischen Entwicklerteams durch Automatisierung vereinfacht. Die Konfigurationen, welche für die Entwicklung von Erweiterungen einer Software benötigt werden, befinden sich vollständig im VCS und werden von dort basierend auf automatischen Abläufen in die lokalen Entwicklungsumgebunden übertragen.

Die Idee, dass Entwicklerteams während der gesamten Entwicklungszeit kontinuierlich Rückmeldung über ihre Arbeit erhalten, wird durch **Continuous Feedback** (CF) beschrieben. Dargestellt durch einen violetten Pfeil in Abbildung 2, ist CF die Flussrichtung der Arbeitsabläufe von „rechts nach links“. Zu jedem Zeitpunkt, von Beginn der Entwicklung bis hin zur Auslieferung an den Kunden, fließt kontinuierliches Feedback zurück an die Entwickler. In der Praxis bedeutet das, dass nach jedem Teilschritt in der Prozesskette ein Feedback mit bspw. Details zum aktuellen Stand der Aus-lieferung, den Testergebnissen und eventuellen Fehlern an die Entwickler übertragen wird. Dass kann mittels Projektverwaltungstools und Plugin realisiert werden. So gibt es z. B. Erweiterungen für Entwicklungsumgebungen, die mit dem VCS verbunden werden, wodurch Entwickler während der Arbeit in ihrer Entwicklungsumgebung Benachrichtigungen erhalten. Ebenso ist eine Integration über E-Mail möglich, bei welcher die Teams durch E-Mail-Benachrichtigungen über den aktuellen Stand informiert werden. Das Ziel des gesamten Prozesses des CF ist die Einarbeitung des Feedbacks in den laufenden Arbeitsschritt.

Das Fehlermanagement wird deutlich verbessert, da Fehler frühzeitig erkannt und behoben werden und so eine nachträgliche Fehlerfortpflanzung in der Codebasis verhindert werden kann. Die Ergebnisse der automatischen Tests können direkt in den aktuellen Arbeitsschritt einfließen und so die Qualität des Codes verbessern. Beide Systeme, sowohl Continuous Delivery und Integration als auch Continuous Feedback, sind Arbeitssysteme. Das bedeutet, dass sie durch ihren Einsatz die Übergabe von Teilcodeabschnitten an den produktiven Betrieb, also an den Kunden, ermöglichen. Diese Mechanismen bilden in Unternehmen mit DevOps-Arbeitsmustern die Grundlage für den gesamten Prozess der Auslieferung von Software.

Continuous Improvement beschreibt als drittes System den Bedarf zur fortlaufenden Verbesserung der Arbeitssysteme. In Abbildung 2 ist es als roter Pfeil dargestellt, welcher den gesamten Ablauf umrandet. Die konstante Anwendung von DevOps-Methoden bedarf einer ständigen Überwachung und Verbesserung dieser. In der Praxis bestehen diese Prozesse aus vielen einzelnen und komplexen Arbeitsabläufen, welche von mehreren Teams ausgeführt werden. Es liegt in der Natur der Sache, dass komplexe Systeme mit fortlaufender Zeit anfällig für Fehler werden. Nachlässigkeit bei der Durchführung von CI oder CF führen zu prozessseitigen Fehlern, welche nur durch kontinuierliche Prüfung der vorhandenen Arbeitsabläufe und gegebenenfalls Verbesserung dieser effektiv verhindert werden können. Continuous Feedback ist durch diese Charakteristika weniger ein werteschaffendes System, vielmehr ist es ein Qualitätssicherungssystem für die prozessverbessernden Arbeitsabläufe.

2.2.4. Versionskontrolle

Bei der Versionskontrolle handelt es sich um einen Mechanismus, durch welchen verschiedene Versionen einer Datei wie bspw. Quelltext verfolgt werden. Durch Versionskontrolle können Änderungen an einer Datei revidiert werden und es ist möglich, Konflikte zu lösen. Besonders bei der Softwareentwicklung, bei der mehrere Entwickler an dem gleichen Abschnitt einer Quelldatei arbeiten, kann es zur mehrfachen Änderung an derselben Codezeile kommen, wodurch Konflikte beim Zusammenführen der Versionen entstehen. Ein Versionskontrollsystem löst diese Konflikte, indem es Versionen zusammenführt oder ggf. zurückstellt.

Ebenso können in neuen Revisionen einer Software Fehler entstehen, welche durch die Nachverfolgung in einem VCS einfach behebbar sind. Die am meisten genutzte Software für die Versionskontrolle ist Git. Im August 2019 betrug der Marktanteil von Git mit über 940.000 angelegten Repositories rund 71 % (siehe Anhang 2). Die Nutzung von Git birgt viele Vorteile. So ist es ein verteiltes System, mit welchem samt Versionshistorie offline gearbeitet werden kann. Neue Repositories lassen sich mithilfe eines Befehls direkt aus einem Verzeichnis heraus erstellen, wodurch die Änderungen an den Dateien direkt erfasst werden.

Entstehen Konflikte innerhalb von Dateien, können diese mithilfe des komplexen Merge-Systems automatisch gelöst werden. Hinzu kommt, dass mit Git, dessen Befehlen und der Versionskontrolle in Branches sehr schnell gearbeitet werden kann. Letztlich ist Git durch seine große Verbreitung oft in Editoren und Entwicklungsumgebungen integriert (Mirco Lang, 2019). Über die Jahre haben sich verschiedene Unternehmen etabliert, die Versionskontrolle mit Git als Dienstleistung anbieten. Git ist Open-Source und damit eine frei benutzbare Technologie, allerdings sind die Plattformen, auf denen Git als Dienstleistung angeboten wird, kommerziell ausgerichtet. Zu nennende Unternehmen sind hierbei GitHub, Bitbucket und GitLab, welche sich den Markt der Bereitstellung von Git-Services teilen. Unabhängig von der Plattform, auf der Git angeboten wird, bietet jedes Unternehmen eine sichere und konfigurierbare Umgebung zur Betreibung von Versionskontrolle an. Die Grundbefehle, mit denen eine Kommunikation mit dem VCS stattfindet, ist ebenfalls unabhängig vom bereitstellenden Unternehmen.

In Git verwalteter Quellcode wird in einem so genannten *Repository* abgelegt. Ein Repository kann sowohl *remote*, also auf einem Server, als auch *local* – lokal auf einem Gerät, angelegt werden. In einem Remote Repository werden alle Quelldateien einer Software gesammelt, getestet und im letzten Schritt ausgeliefert. Entwickler, die an der Software arbeiten, erstellen ein Abbild des Remote Repository auf ihrem lokalen Gerät, ein Local Repository. Werden nun Änderungen oder Erweiterungen an den Quelldateien vorgenommen, werden diese an das Local Repository übergeben, welches dann die Dateien mit dem Remote Repository abgleicht und Änderungen nachverfolgt sowie mögliche Konflikte mit anderen Entwicklern löst.

Dieser Vorgang ist als *add – commit – push* bekannt. Andere Entwickler, die simultan an dem gleichen Abschnitt einer Quelldatei arbeiten, können mithilfe des VCS ein Abbild des aktualisierten Standes in ihr eigenes Local Repository ziehen und Änderungen in ihre lokale Version einfließen lassen. Dieser Vorgang heißt *pull – merge*. Den Vorgang der Kommunikation zwischen Local und Remote mit den einzelnen Schritten visualisiert Abbildung 3. In ihr ist zu erkennen, wie der Arbeitsbereich mithilfe von Git unterteilt wird. Die Arbeit der Entwickler findet ausschließlich in einem Arbeitsverzeichnis über eine Entwicklungsumgebung statt. Sollen Änderungen übernommen werden, wird der Code zunächst in einen Staging Bereich übertragen. Staging bezeichnet dabei den Stand der Codebasis, wie er zwischen der Entwicklung und der Produktion existiert.

Es ist die akkurateste Repräsentation der aktuellen Codebasis und zugleich der Bereich, in welchem das Testen und die Qualitätssicherung stattfindet (Commonplaces Interactive, 2018). Staging kann als eine Art Schleuse zwischen neuem, ungetestetem Code und der funktionierenden Codebasis gesehen werden. Aus dem Staging-Bereich werden alle Änderungen zusammengefasst und an das lokale Repository inklusive einer Nachricht übergeben. Typischerweise beschreiben Entwickler in dieser Nachricht die Änderungen, die vorgenommen wurden oder worum es sich bei dieser neuen Version handelt. Von dort aus wird im letzten Schritt die neue Version an das Remote Repository und die aktuelle Codebasis übergeben. Durch die Vereinigung der neuen Version mit dem aktuellen Stand in der SOT ergibt sich eine neue Codebasis, die von nun an den aktuellen Stand repräsentiert.

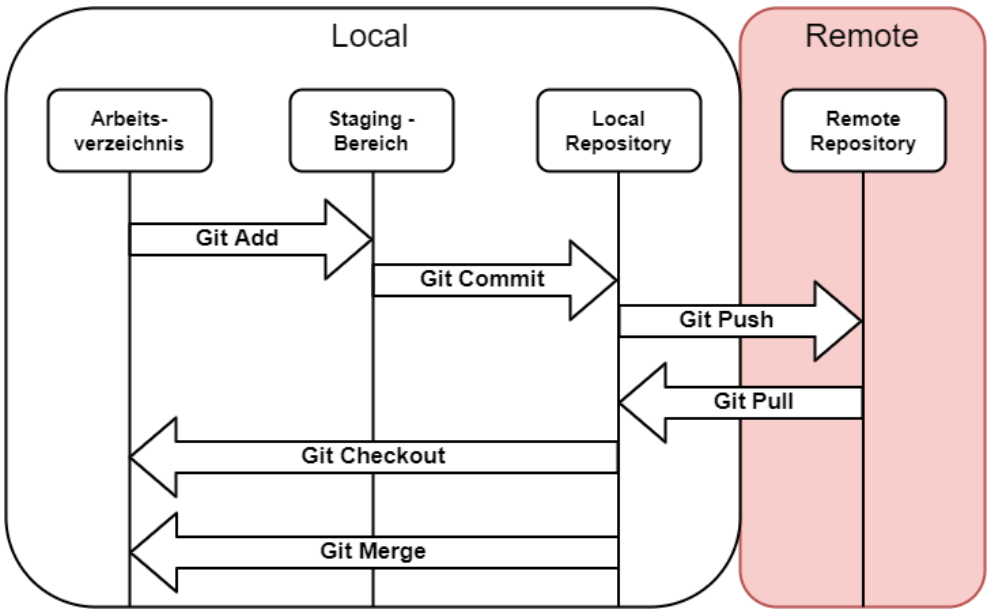


Abbildung 3: Kommunikation zwischen Local und Remote in Git (Davis, 2019, S. 209)

Zur Verdeutlichung der Arbeit mit Git werden nachfolgend die einzelnen Befehle aufgeführt und erklärt.

Um ein lokales Repository zu erstellen, legt man ein Verzeichnis an, in welchem gearbeitet wird. Mit dem *clone*-Befehl wird ein Abbild des Remote Repositories in dem Arbeitsverzeichnis erstellt.

```
git clone <repository>
```

Nachdem das Projekt geklont wurde, kann ein neuer Zweig, ein sog. *Branch* erzeugt werden, mit welchem fortan gearbeitet wird.

Ein Branch ist eine Kopie der Codebasis, die unabhängig von Änderungen an ihr besteht und nach Beendigung der Arbeiten wieder mit dem Kernprojekt, dem sog. *master*, vereinigt wird. Der Befehl zum Erzeugen eines Branches lautet:

```
git checkout -b <branch name>
```

Wurde ein Branch erzeugt, kann die Arbeit an dem Code beginnen. Um Unterschiede zur originalen Version zu sehen, wird der Status geprüft mit:

```
git status
```

Bevor die Änderungen und Neuentwicklungen an das lokale Repository übergeben werden können, müssen sie an den Staging-Bereich übermittelt werden. Der Befehl zur Übermittelung einzelner Dateien lautet:

```
git add <file_name>
```

Wurden die Dateien getestet, werden sie an das lokale Repository übergeben. Dabei ist es wichtig, dass eine entsprechende Nachricht hinzugefügt wird, die den Inhalt der Änderungen oder Merkmale der Neuentwicklung beinhaltet, um von anderen Entwicklern leichter identifiziert werden zu können.

```
git commit -m <commit message>
```

Der letzte Schritt der Bereitstellung des Codes im Remote Repository ist das „Schieben“. Der Push-Vorgang übertragt die Dateien von local zu remote und prüft auf mögliche Konflikte. Treten keine Konflikte auf oder sind die vorhandenen gelöst, bilden die Änderungen zusammen mit der aktuellen Codebasis eine neue Version des Codes.

```
git push
```

Arbeiten andere Entwickler am selben Projekt, ziehen sie sich den neuen aktuellen Stand in ihr lokales Repository, um eine aktualisierte Version der Codebasis als Grundlage für ihre eigenen Arbeiten zu erhalten. Der Befehl lautet:

```
git pull
```

Es gibt weitere Git-Befehle, welche die gemeinsame Arbeit an einem Projekt ermöglichen. Die näher betrachteten Befehle bilden lediglich die grundlegende Kommunikation über ein Git-Versionskontrollsystem. Mit diesem Verfahren wird der Code über die Versionskontrolle aktuell gehalten und jeder Entwickler hat zu jedem Zeitpunkt ein Abbild der aktuellen Version einer Software.

2.2.5. Verzweigung

Die Kontrolle über die Versionen in einem Repository wird über die Verzweigung, englisch *Branching*, gelöst. So genannte Branches sind alternative Versionen der Codebasis, welche parallel zur ihr existieren. Bei der Änderung von Dateien in der Codebasis wird aus ihr ein Branch erzeugt, welcher ab diesem Zeitpunkt vom Haupt-Branch, dem *master*, abgeht und parallel fortgeführt wird.

Dabei ist jeder Branch zwar eine exakte Kopie der Codebasis, kann sich jedoch im Verlauf der Entwicklungsarbeit von ihr unterschreiben. Sobald Änderungen durchgeführt sind, wird der Branch über einen *merge*-Befehl mit dem master zusammengeführt und die Codebasis wird aktualisiert. Auf diesem Vorgang basiert die gesamte Versionskontrolle in Git. Abbildung 4 demonstriert die Versionierung mit verschiedenen Branches. Der master ist durch eine blaue Färbung markiert. Der Branch, der beispielsweise für eine neue Funktion erzeugt wird, ist grün dargestellt. Jede Aktualisierung der Dateien ist durch einen Kreis markiert.

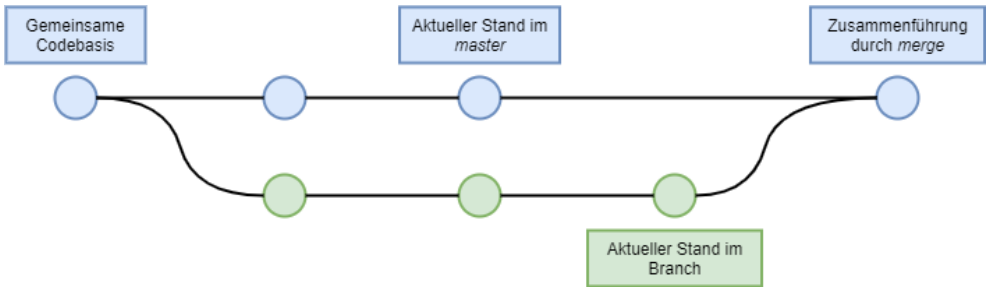


Abbildung 4: Branching in Git (Eigene Darstellung)

Wie in der Praxis mit Branches umgegangen wird, ist von den Entwicklern abhängig. Es gibt sowohl langlebige Branches, welche für gesamte Entwicklungszyklen bestehen bleiben und diverse Änderungen beinhalten, als auch kurzlebige Branches, welche nach abgeschlossenen Arbeitspaketen und kleinen Änderung wieder mit dem master zusammengeführt werden. Neben Branches gibt es in Git eine weitere Methode der Versionierung namens *Fork*. Ein Fork ist, ebenso wie ein Branch, eine Kopie der Codebasis. Jedoch wird bei dem so genannten *Forken* das gesamte Repository kopiert, welches sich ab diesem Zeitpunkt unabhängig von dem originalen Repository verändern kann. Änderungen können auch hier zwischen den Repositories durch eine Verbindung zusammengeführt werden. Ein Grund für einen Fork kann zum Beispiel die unabhängige Weiterentwicklung eines Projektes sein. Ein Projekt kann so in zwei Versionen existieren, welche eine gemeinsame Basis haben, sich allerdings in voneinander unabhängige Richtung entwickeln.

2.3. Salesforce-Infrastruktur

Wie bereits in Kapitel *2.2.2 Salesforce und DevOps* beschrieben, besteht eine Salesforce Org aus einer Kernlandschaft an Systemen. Die Sales, Service und Community Cloud sowie die Lightning-Plattform bilden die Basis einer Organisation, welche durch weitere Dienstleistungen aus dem Salesforce-Ökosystem erweitert wird. Es ist wichtig, dass sowohl Entwickler als auch Administratoren, die innerhalb der Salesforce-Strukturen arbeiten, den Aufbau und das Datenmodell verstehen. Für den Einsatz von DevOps in Salesforce gesteuerten Umgebungen wird zusätzlich ein Verständnis der neuen Mechanismen vorausgesetzt, mit welchen Code von einer lokalen Entwicklungsumgebung in eine Salesforce Org implementiert wird. Im Gegensatz zur Webentwicklung ist es mit Salesforce nicht möglich, Code lokal auszuführen. Dass bedeutet, dass jede Änderung an Metadaten oder Code innerhalb einer Salesforce-Instanz kompiliert und nur dort ausgeführt wird. Umso wichtiger ist die Betrachtung der Methodik, mit welcher die Implementierung von Änderungen nach DevOps-Ansätzen in Salesforce möglich ist.

2.3.1. Metadaten

Um die Struktur einer Salesforce Org zu beschreiben, werden so genannte Metadaten verwendet. Alle Objekte, Felder, Listen, Layouts und benutzerdefinierte Änderungen werden in Dateien abgelegt, welche die Struktur des Datenmodells wiederspiegeln. Diese Dateien können aus einer Salesforce-Instanz in eine lokale Entwicklungsumgebung übertragen, bearbeitet und schließlich wieder in die Salesforce-Instanz eingespielt werden. Metadaten-Dateien sind zum Großteil im XML - kurz für *Extensible Markup Language*-Format gespeichert. Durch die Einführung von Salesforce DX sind moderne Metadaten auch im JSON-Format vorhanden. Durch diese Metadatentypen ist es möglich, Implementierungen und Tests mithilfe der Versionskontrolle automatisch durchzuführen (Davis, 2019, S. 77). Zum Beispiel können zwei Metadaten-Dateien miteinander verglichen werden, um Änderungen festzustellen, woraufhin Prozesse ausgelöst werden können.

2.3.2. Salesforce DX

Im Juni 2017 wurde die Beta von Salesforce DX veröffentlicht und im Winter-Release `18 allgemein zur Verfügung gestellt (Kraft, 2017). Mit Salesforce DX gab es ab diesem Zeitpunkt neue Methoden zur Entwicklung und Implementierung von Code in einer Salesforce Org. Zwei wesentliche Aspekte verändern sich im Vergleich zur vorherigen Methodik.

Für jedes Projekt gibt es eine Versionskontrolle als Source of Truth. In dieser befindet sich die Codebasis, jede Änderung wird an das Versionskontrollsystem übergeben und getestet. Damit findet eine vollständige Versionierung der Software statt und jede Änderung kann verfolgt und ggf. rückgängig gemacht werden. Anpassungen des Codes werden in sog. Scratch Orgs durchgeführt. Das sind temporäre Kopien der Org, die ausschließlich über das VCS erzeugt und nach fertiggestelltem Paket wieder entsorgt werden. Die vorherige Herangehensweise beinhaltet die Entwicklung in dafür vorgesehenen Developer Organisationen, so genannten Sandboxen, die ebenfalls Kopien der Produktions-umgebung sind, jedoch deutlich komplexer aufgebaut sind und nicht aus dem VCS erzeugt werden.

Code sowie Metadaten sollen in Pakete unterteilt werden, wovon jedes eine abgetrennte Funktionalität repräsentiert (Davis, 2019, S. 20). Dadurch wird die Komplexität einer Code-Auslieferung verringert und es kann an mehreren voneinander unabhängigen Paketen parallel gearbeitet werden. Um Code zu entwickeln, werden den Entwicklern neue Werkzeuge an die Hand gegeben. Über das Salesforce CLI kann nun über die Kommandozeile der gesamte Prozess gesteuert werden. Scratch Orgs können in wenigen Sekunden erstellt und konfiguriert, Code mit einem einzigen Befehl implementiert werden. Des Weiteren wird die Integration des Salesforce CLI durch Erweiterungen für die *Visual Studio Code* (VS Code) Entwicklungsumgebung (IDE) ermöglicht. Sowohl Salesforce CLI als auch VS Code ersetzen zwei bisher grundlegende Werkzeuge der Salesforce-Entwicklungslandschaft.

Das *ANT Migration Tool* wurde benutzt, um Metadaten zwischen unterschiedlichen Salesforce-Organisationen zu übertragen. Mit Salesforce DX geschieht dies nun über das Salesforce CLI.

VS Code ersetzt die bisher Salesforce-eigene IDE zur Entwicklung von benutzerdefinierter Logik und benutzerdefinierten Anwendungen *Force.com IDE*, welche als Aufsatz für die bekannte *Eclipse*-Entwicklungsumgebung verfügbar war. Das Ziel bei der Einführung der neuen Methoden war sowohl eine bessere Zusammenarbeit der Entwickler als auch die Steigerung der Effizienz (Salesforce, kein Datum). Mit Salesforce DX haben Entwickler nun mehr Kontrolle über ihre Anwendungen und die notwendigen Schritte zur Implementierung dieser.

2.3.3. Dev Hubs und Scratch Orgs

Das neue Modell für die Entwicklung mit Salesforce DX basiert bei der Bereitstellung von Code auf einem neuen System bestehend aus dem so genannten Dev Hub und die dazugehörige Scratch Orgs. Eine Salesforce-Produktionsumgebung kann in ein Dev Hub umgewandelt werden, damit den Entwicklern die Funktion zur Verfügung steht. Dabei hat diese Einstellung keine nachteilige Auswirkung auf den bisherigen Funktionsumfang oder den laufenden Betrieb. Während der Entwicklungsarbeit können Entwickler eine flüchtige Kopie der Produktionsumgebung erstellen, welche weder Daten noch Metadaten aus der als Dev Hub aktivierten Org beinhalten.

Eine Scratch Org ist die Alternative zu der vormals zur Entwicklung benutzten Developer Sandbox und wird ausschließlich durch die Metadaten und Konfiguration definiert, die sich in dem VCS befinden, aus welchem sie erstellt wurden. Die Idee ist, Scratch Orgs als „Wegwerf-Organisation“ zu betrachten, die allein dem Zweck der Entwicklung eines von anderen Funktionen unabhängigen Teiles einer Software dient. Sind die Arbeiten an dieser Teilsoftware fertiggestellt, wird es mithilfe des Salesforce CLI veröffentlicht und die Scratch Org wird entsorgt. Die standardmäßige Lebenszeit einer Scratch Org beträgt 7 Tage, wobei diese auf 30 Tage erhöht werden kann. Diese neue Art der Organisation wird als *source-driven*, also quellengesteuerte Entwicklung bezeichnet, wohingegen herkömmliche Developer Sandboxen und Teil- bzw. Vollkopien als beständige Testumgebungen für kombinierte Pakete und komplexe Prozesse ihre Daseinsberechtigung behalten.

Der neue Entwicklungsprozess besteht demnach aus der Entwicklung von Code-Paketen und Metadaten in flüchtigen Scratch Orgs, welche mittels CI und CD zunächst in Sandboxen implementiert und dort getestet und anschließend in Produktionsumgebungen bereitgestellt werden (Davis, 2019, S. 21).

2.3.4. Development Lifecycle

Der gesamte Prozess der Entwicklung und Bereitstellung von Code in einer Salesforce-Organisation inklusive der zugehörigen Werkzeuge wird als Salesforce DX Development Lifecycle bezeichnet. Die grundlegenden Elemente sind eine Salesforce Development Org, eine IDE, Versionskontrolle und ein Werkzeug für CI, um eine Automation einzurichten. Die Wahl der Entwicklungsumgebung ist dabei dem Entwickler überlassen, wonach auch ein einfacher Texteditor benutzt werden kann. IDEs wie z. B. VS Code beinhaltet jedoch im Vergleich eine umfassende Bibliothek mit Erweiterungen, welche die Arbeit mit Werkzeugen wie bspw. des Salesforce CLI nahtlos in die Umgebung integrieren und die Entwicklungsarbeit deutlich vereinfachten. Über viele Jahren waren Developer Sandboxen, Teil- und Vollkopien der einzige Weg, um Code zu entwickeln, zu testen und zu implementieren. Mit Salesforce DX und den Scratch Orgs gibt es nun eine moderne, quellengesteuerte Alternative zu diesem Weg. Wie bereits in vergangenen Kapiteln beschrieben, ist die Methodik mit Salesforce DX deutlich zeiteffizienter. Über das Salesforce CLI kann der gesamte Lifecycle mithilfe von wenigen Befehlen über die Kommandozeile gesteuert werden.

Das beinhaltet das Klonen einer Organisation in eine Scratch Org auf Basis von Metadaten, die sich in einem VCS befinden, das Veröffentlichen von Code-Paketen in einer Testumgebung sowie das automatische Testen und Ausliefern von fertiggestellter Software an eine Produktionsumgebung.

3. Integration im Unternehmen

Die im theoretischen Teil behandelten Themen bilden die Grundlage über das Verständnis der Arbeit mit sowohl DevOps als auch Salesforce im Speziellen. Durch DevOps sollen innerbetriebliche Prozesse verbessert und teils automatisiert werden, damit die Zusammenarbeit kontinuierlich und effizient abläuft. Mit Salesforce DX stehen Salesforce-Entwicklern Methoden und Werkzeuge zur Verfügung, welche die DevOps-Ansätze als Grundlage in eine Prozesskette für die Entwicklung und Bereitstellung von Code einfließen lassen. Der zweite Teil der Arbeit behandelt die tatsächliche Umsetzung der gelernten Prinzipien in einem Unternehmen, um die Effizienz und Sicherheit der Entwicklungsarbeit zu steigern.

3.1. Ausgangssituation

Die Agentur hanseflow existiert seit Ende 2014 und hat seit ihrer Gründung einen starken innerbetrieblichen Wandel durchlebt. Ursprünglich im Bereich der Webentwicklung tätig, lag der Fokus der Tätigkeit zunächst auf dem Aufbau von Webseiten und Onlineshops. Im weiteren Verlauf wurde eine Partnerschaft mit *SugarCRM* geschlossen, einem Unternehmen, welches vorranging On-Premises CRM-Systeme anbietet und auf der PHP-Programmiersprache basiert. Mitte 2017 wurde der Fokus durch eine Partnerschaft mit Salesforce stark verschoben, wodurch die Beratung von mittelständischen Unternehmen hinsichtlich des Salesforce CRM in den Vordergrund rückte. Seit Beginn der Arbeit mit Salesforce bietet hanseflow auch die Entwicklung von benutzerdefinierter Geschäftslogik mit der Salesforce-eigenen Programmiersprache APEX sowie im späteren Verlauf auch die Konzeption und Umsetzung von benutzerdefinierten UI-Elementen auf Basis der Lightning-Plattform (ehemals Force.com-Plattform) an. Zwischen Ende 2017 und Anfang 2020 erfuhr die Agentur einen starken personellen Zulauf und die Herausforderungen an die unternehmensinterne Struktur und das Projektmanagement wuchsen. Vor der Umstrukturierung befand sich das Unternehmen hinsichtlich der Entwicklung und Bereitstellung von benutzerdefinierten Anpassungen für große Projekte an einem Wendepunkt.

3.1.1 Softwarelandschaft

Das Unternehmen bediente sich bei der Verwaltung seiner Projekte diverser Clouddienste und Software. Für das Projektmanagement setzte es auf die Dienstleistungen von Atlassian. Um Projektablaufpläne sowie Aufgaben zu erstellen, benutzte hanseflow JIRA. Dieses bietet einen großen Funktionsumfang sowohl für die unternehmensinterne als auch die externe Kommunikation mit dem Kunden. Mit JIRA verbunden ist der *Confluence*-Dienst. Dort wurden Informationen gepflegt und Online-Dokumente für die Projekte angelegt. Sowohl die Anforderungen eines Projektes als auch Ansprechpartner und zusätzliche Informationen befanden sich in Dokumenten, welche nach einer einheitlichen Struktur aufgebaut waren. JIRA und Confluence arbeiten durch gemeinsam genutzte Lizenzen über die Atlassian Cloud effizient zusammen und bildeten damit den Kern des Projektmanagements. Im Hinblick auf weitere Dienstleistungen von Atlassian bot sich die Nutzung von Bitbucket an. Dieser Dienst ist ein Versionskontrollsystem, welches nahtlos in die bereits vorhandene Projektverwaltungslandschaft integriert und dadurch bspw. mit Projekten und Tickets verbunden werden kann. Bitbucket arbeitet weiterhin mit Git, wodurch es für die Nutzung als zentrale Versionskontrolle prädestiniert war. Hinsichtlich der Kommunikation wurde innerhalb des Unternehmens *Slack* benutzt. Slack ist ein Chat-Tool, mit welchem sowohl private Nachrichten ausgetauscht als auch einzelne, abgetrennte Kanäle für Projekte angelegt werden können. Eine Integration in die bestehende Projekteverwaltungslandschaft von Atlassian war durch Erweiterungen möglich. Für die externe Kommunikation fand *Microsoft Outlook* im Unternehmen Verwendung. Outlook kann durch JIRA-, Confluence- und Bitbucket-Plugins erweitert werden, wodurch sowohl dem Betrieb als auch Entwicklern ein möglicher Feedback-Kanal zur Verfügung steht. Besonders im Hinblick auf automatische Tests und Continuous Delivery bot sich die Verwendung von entsprechenden Erweiterungen für Microsoft an und bestätigte die Wahl des E-Mail-Dienstes. Das Entwicklerteam benutzte für die Aufteilung der Arbeitspakete ebenfalls die JIRA-Plattform. Als Entwicklungsumgebung kam VS Code zur Anwendung, welches u.a. durch eine Salesforce CLI-Erweiterung und eine JIRA-Integration konfiguriert wurde.

Dadurch standen Arbeitspakete in Form von vorformulierten Tickets innerhalb der IDE zur Verfügung. Durch die Erweiterung um Salesforce DX und CLI konnten entsprechende Kommandozeilenbefehle innerhalb der Entwicklungsumgebung genutzt werden. Für die Bereitstellung von Code wurde kein Werkzeug benutzt. Dies geschah ausschließlich auf direktem Weg über Kommandozeilenbefehle an die entsprechende Salesforce Organisation.

3.1.2 Projektablauf

Nachdem ein Kunde gewonnen wurde, gab es einen festgelegten Ablauf, der bei allen Projekten identisch war. Zunächst wurden Workshops veranstaltet, in welchen die genauen Anforderungen des Kunden erfasst und ausgearbeitet wurden. Dabei fand eine Analyse der unternehmensinternen Prozesse statt. Es wurde ein Projekttyp – Kanban oder Scrum – festgelegt und das administrative Team teilte die Aufgaben in Pakete ein. Dokumente, Informationen und Details zum Projekt wurden in einem neuen Confluence-Abschnitt eingepflegt, die Arbeitspakete in Form von Tasks in JIRA angelegt. Während die Salesforce-Organisation in einer Developer Sandbox eingerichtet und angepasst wurde, fanden parallel Entwicklungsarbeiten in einer anderen Developer Sandbox statt. In regelmäßigen Abständen wurden diese Arbeiten von dem Kunden überprüft, damit noch zur laufenden Arbeit Feedback eingearbeitet werden konnte. In ebenso regelmäßigen Abständen wurden Anpassungen sowohl deklarativer als auch programmatischer Art in die Produktivumgebung implementiert. Sobald das Projekt abgeschlossen wurde, nahm der Kunde die Arbeiten ab und das System bekam die Freigabe für den Betrieb. Gegebenenfalls fanden im Anschluss Schulungen statt und es wurde ein Servicevertrag für fortlaufende Arbeiten geschlossen. Dieser Projektablauf hatte diverse Nachteile, welche durch die getrennte Arbeit von Entwicklern und Administratoren entstand.

Salesforce-Administratoren arbeiten ausschließlich deklarativ. Das bedeutet, dass sie Objekte, Felder und Prozesse mit „Point-and-Click“-Methoden einrichten. Salesforce hat für die schnelle Einrichtung von Prozessen und Automatismen diverse Hilfsmittel und Werkzeuge in die Oberfläche eingebaut, mit denen eine deklarative Erzeugung dieser Elemente möglich ist. Die Änderungen, die dabei entstehen, erfordern weder Tests noch eine andere Art der Validierung, da sie keine

Auswirkungen auf das System, sondern lediglich auf die Arbeitsabläufe innerhalb der Organisation haben. Entwickler auf der anderen Seite arbeiten fast ausschließlich programmatisch in einer dafür vorgesehen Kopie der Organisation. Der Code, der dabei entsteht, muss über ein Mindestmaß an Testabdeckung verfügen, welches durch einen entsprechenden Testcode erreicht wird. In der Regel fand innerhalb des Unternehmens die Arbeit in einer Developer Sandbox statt, jedoch wurden auch in der Produktivumgebung Änderungen durchgeführt, was dazu führte, dass die Aktualität der parallel genutzten Kopien der Organisation nicht mehr vorhanden war. Entwickler, die in einer solchen Kopie arbeiteten, fertigten Code auf Basis der Felder und Objekte an, die in der Produktionsumgebung vorhanden waren. Dadurch entstanden oft Fehler, die einer langen Suche bedurften, obwohl sie hätten vermieden werden können. Die Diskrepanz zwischen deklarativen Änderungen, die in der Produktivumgebung oder in einer anderen Organisation als der Entwicklungsorganisation durchgeführt werden, und programmatischen Änderungen bestätigt die Notwendigkeit eines Versionskontrollsystems, welches während der gesamten Arbeit die Aktualität der jeweiligen Kopien garantiert.

Ein weiteres Problem, welches sich durch den Projektablauf ergab, war die fehlende Möglichkeit zur Verbesserung des Ablaufes selbst. Alle Projekte mündeten in einer finalen Abnahme, ohne den Prozess der Entwicklung oder Bereitstellung intern zu überarbeiten. Dies ist auf die fehlende Möglichkeit des internen Feedbacks zurückzuführen. Ein Grundprinzip nach DevOps ist die kontinuierliche Verbesserung der Arbeitsabläufe, was durch konstantes Feedback sowohl bei der Entwicklung und beim Testen von Code als auch bei der Bereitstellung von Anpassungen ermöglicht wird. So entsteht ein einheitlicher und sich verbessernder Prozess, welcher nach jedem erfolgreichen Projekt von internem Feedback profitiert und mit den Herausforderungen der Teams innerhalb des Unternehmens wächst.

3.2. Umsetzung

Auf Basis des theoretischen Wissens wurde der Prozess der Entwicklung innerhalb des Unternehmens umgesetzt. Dabei lag der Fokus vorrangig auf der Entwicklung und Bereitstellung von Code mittels eines durch Versionskontrolle und CI-Tools gestützten Prozesses. Dieser Prozess ersetzt den bisherigen Projektablauf im Bereich der Entwicklung vollständig durch neue Mechanismen und Automatisierungen. Nachfolgend wird auf die Einzelheiten bei der Planung und Umsetzung dieser Mechanismen eingegangen und es erfolgt ein Ausblick auf mögliche Erweiterungen und Ausweitungen auf den gesamten internen Projektablauf.

3.2.1. Entwicklungsmodelle im Vergleich

In Kapitel *3.1.2 Projektablauf* wurde auf den allgemeinen Projektablauf eingegangen. Diese Arbeit richtet den Fokus vorranging auf die Umstrukturierung des Entwicklungsprozesses. In Abbildung 5 ist der ursprüngliche Prozess abgebildet, durch welchen benutzerdefinierter Code bereitgestellt wurde.

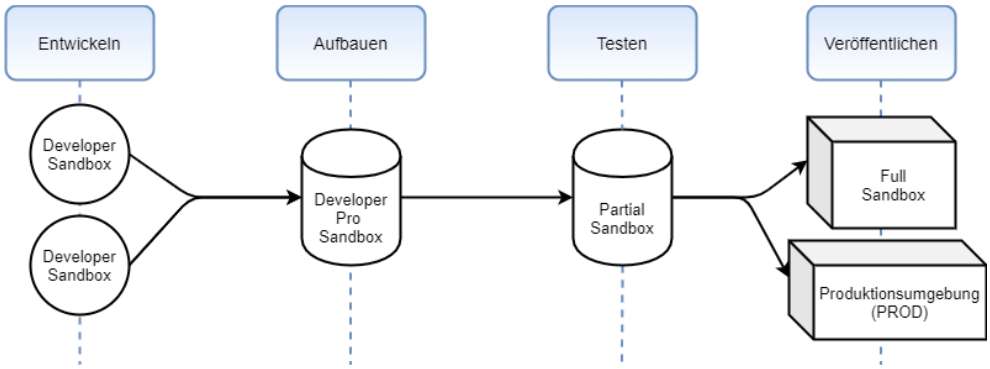


Abbildung 5: Change Set Development (Salesforce Developers, 2019)

Der Bereitstellungsprozess wird in vier Abschnitte unterteilt. Das Entwicklerteam arbeitet in einer oder mehreren Developer Sandboxen an benutzerdefiniertem Code. Sobald Code fertiggestellt ist, wird er in einer Developer Pro Sandbox oder einer Developer Sandbox zusammengefasst und in die deklarative Umgebung der Organisation integriert. Mit diesem Schritt soll sichergestellt werden, dass ursprüngliche Funktionen und Prozesse nicht durch den neu entwickelten Code beeinträchtigt werden.

Nach erfolgreicher Integration wird der Code in eine Partial Sandbox übertragen. Dies ist eine Umgebung, die vollständig der Produktions-umgebung gleicht. Zusätzlich zu den identischen Metadaten wird auch ein Teil der Datensätze in die Partial Sandbox übertragen, um Tests unter realen Umständen durchführen zu können. Waren alle Tests erfolgreich, werden die neuen Funktionen in eine Full Sandbox, also eine Eins-zu-Eins-Kopie der Produktionsumgebung, integriert. Unter Umständen findet auch eine Integration in Produktionsumgebung statt, ohne dass die Funktionen zuvor in einer Full Sandbox überprüft wurden. Im letzten Schritt findet das Ausrollen der neuen Funktionen statt sowie das Schulen der Endnutzer. Dadurch, dass die Bereitstellung der Funktionen in der jeweiligen Umgebung über so genannte *Change Sets* stattfindet, wird diese Art der Entwicklung auch als *Change Set Development* bezeichnet. Change Sets sind Sammlungen diverser Metadaten, welche zusammengefasst zwischen Salesforce-Organisationen transferiert, getestet und anschließend bereitgestellt werden können. Dabei werden Felder, Objekte, Prozesse oder benutzerdefinierter Code über das Salesforce Setup deklarativ einem Change Set hinzugefügt und an eine Zielorganisation geschickt. Durch diesen Prozess sind Entwickler sehr flexibel hinsichtlich der Art der zu übertragenden Daten, haben jedoch einen erhöhten Aufwand durch das manuelle Anweisen von Change Sets bei jeder neuen Funktion.

Der neue Entwicklungsprozess, welcher in dem Unternehmen umgesetzt wurde, basiert im Vergleich zu dem Change Set Development auf der Bereitstellung mittels des CLI. Aus der Entwicklungsumgebung heraus werden über Befehle die Synchronisation mit einer Versionskontrolle und die Auslieferung von Code gesteuert. In Abbildung 6 wird der gesamte Prozess dargestellt.

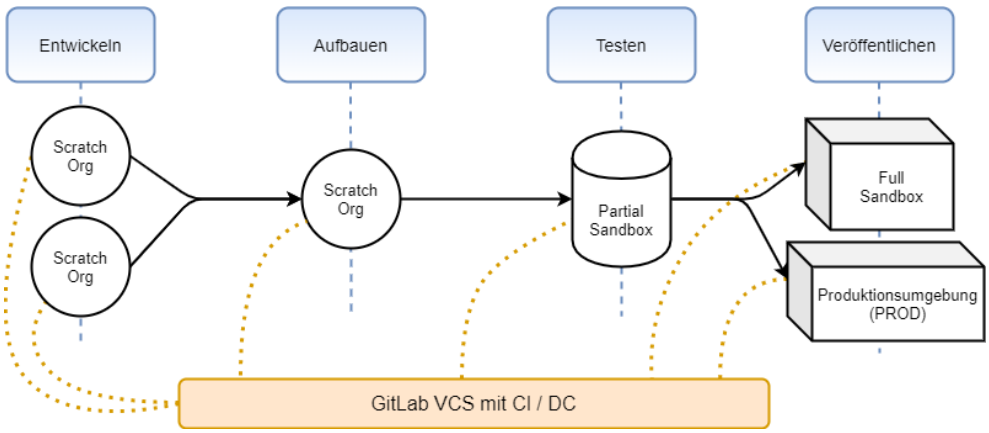


Abbildung 6: Package Based Development (Salesforce Developers, 2019)

Statt in Developer Sandboxen findet die Entwicklung ausschließlich in temporären Scratch Orgs statt. Der Verlauf des Prozesses wird durch ein Versionskontrollsystem gestützt, durch welchen die Versionen in den unterschiedlichen Schritten nachverfolgt werden können. Wie bei dem Change Set Development wird auch bei dem sog. *Package Based Development* Code von einer Organisation in die nächste ausgeliefert. Die Auslieferung findet jedoch in kleinen Paketen über das CLI statt, welche durch ihre Funktion voneinander abgetrennt werden – sog. *Packages.* In der Abbildung sowie in der tatsächlichen Integration kommt *GitLab* zum Einsatz. GitLab ist eine DevOps-Plattform, in welcher sowohl Continuous Integration und Delivery als auch ein Versionskontrollsystem angeboten werden. Dabei bietet das System für alle relevanten Teilprozesse Funktionen aus einer Hand an, durch welche sich der gesamte Entwicklungsprozess steuern lässt. GitLab bietet zudem eine umfangreiche Integration für Salesforce an.

3.2.2. Vorbereitung in Salesforce

Um die Einrichtung von GitLab und die Kommunikation mit Salesforce zu demonstrieren, wurde im Rahmen der Arbeit ein Trailhead Playground erzeugt, welcher als Salesforce-Organisation für die Umsetzung diente. Trailhead ist eine Salesforce-Plattform, welche eine Vielzahl an so genannten *Trails* – Lernpfade – zum Thema Salesforce anbietet. Ein Playground ist eine Übungsorganisation, um neu erlernte Techniken durch Praxisbeispiele zu testen und zu vertiefen. Der Aufbau eines Playgrounds ist identisch zu einer Produktionsorganisation und repräsentiert im Zusammenhang mit dieser Arbeit ein beispielhaftes Unternehmen.

Das neue Entwicklungsmodell, welches im Unternehmen etabliert wurde, basiert auf der Erzeugung von Scratch Orgs. Um Scratch Orgs erzeugen zu können, muss die Organisation des Kunden als Dev Hub aktiviert sein. Die Aktivierung findet über das Setup User Interface statt. Der nächste notwendige Schritt bei der Kommunikation mit GitLab ist die Einrichtung einer sog. *Connected App.* Salesforce bietet über das Setup User Interface die Möglichkeit an, eine externe Anwendung innerhalb des Systems zu registrieren.

Durch die Erstellung einer Connected App werden dieser Anwendung Authentifizierungsdetails zugeteilt, über welche diese Anwendung im Anschluss mit dem Salesforce-System kommunizieren kann. Zwar ist durch die enge Zusammenarbeit von Salesforce und GitLab die An-meldung mit den eigenen Salesforce-Daten in GitLab möglich, jedoch dient dieser Vorgang lediglich der Identifizierung als Person. Um GitLab Zugriff auf eine Organisation zu gewähren, ist die Erstellung einer Connected App zwingend notwendig. Salesforce stellt für die App sog. *Credentials* bereit. Das sind Authentifizierungsschlüssel, bestehend aus einem *Key* und einem *Secret*, die bei der Kommunikation mit der Anwendung während des Authentifizierungsprozesses notwendig sind.

3.2.3. Einrichtung des GitLab SCM

GitLab ist, anders als beispielweise GitHub, deutlich umfangreicher als ein Versionskontrollsystem. Man bezeichnet diese Art von System als Software Configuration-Management-System, kurz SCM. Ein SCM ist ein System, das auf die Konfiguration „in Bezug auf alle Aktivitäten, die das Gebiet der Softwareentwicklung betreffen“ (Augsten, 2017) spezialisiert ist. Durch umfangreiche Konfigurationsmöglichkeiten wie z. B. den Aufbau einer CI/CD Pipeline oder die Integration einer Versionskontrolle ist es ein Software-Versionsverwaltungssystem, welches vorrangig für die Verwaltung von Quellcode genutzt wird.

Um GitLab für eine quellengesteuerte Entwicklung zu nutzen, gibt es zwei Voraussetzungen, die auf dem Rechner des Entwicklers erfüllt sein müssen. Erstens muss das Salesforce CLI installiert sein. Salesforce CLI erlaubt es, Entwicklungs- und Testumgebungen zu erstellen, Quellcode zwischen einer Org und einem Versionskontrollsystem zu synchronisieren und Apex Tests auszuführen. Zweitens muss Git installiert sein, um auf die Git-Befehle von der Kommandozeile aus zugreifen zu können. Mit Git werden alle Befehle installiert, die für die Kommunikation mit einem Repository notwendig sind. Um eine Dev Hub Org zu authentifizieren, damit mit Scratch Orgs gearbeitet werden kann, wird sie über das Salesforce CLI authentifiziert. Der Befehlt lautet:

```
sfdx force:auth:web:login --setalias DevHub --setdefaultdevhubusername
```

Durch die Eingabe des Befehls öffnet sich eine Salesforce Login-Seite, über welche der Benutzername und das Passwort für die Dev Hub Org eingegeben werden. Nach einer erfolgreichen Authentifizierung, welche in Abbildung 7 dargestellt ist, kann die Verbindung mit GitLab aufgebaut werden.

```
PS C:\Users\max> sfdx force:auth:web:login --setalias DevHub --setdefaultdevhubusername
Successfully authorized kretzschmar@brave-goat-he0hri.com with org ID 00D3X000004X6elUAC
```

Abbildung 7: Erfolgreiche Authentifizierung mit der Dev Hub Org (Eigene Darstellung)

Um den nächsten Schritt durchzuführen, ist ein GitLab Account notwendig. In GitLab wird nun ein neues Projekt angelegt. Für das Beispiel des Aufbaus einer Countinuous Integration / Continuous Delivery Pipeline mit Salesforce wurde im Rahmen der Arbeit ein Beispielprojekt von Salesforce namens *Dreamhouse* benutzt. Dreamhouse ist eine vorgefertigte Salesforce-Anwendung, welche in Trailhead-Projekten als Lernanwendung zum Einsatz kommt. Aus Datenschutzgründen konnte im Rahmen dieser Arbeit kein reales Kundenprojekt als Anschauungsbeispiel verwendet werden. Mithilfe der Dreamhouse-App können alle relevanten Schritte zur Einrichtung einer Automation mit GitLab dargestellt werden. Der Quellcode von Dreamhouse befindet sich in einem Remote Repository und wird über die folgende GitHub-URL als Projekt in GitLab importiert.

```
https://github.com/DreamHouseapp/DreamHouse-sfdx.git
```

Bei der Projekterstellung wird unter realen Umständen ein so genanntes *Template* genutzt, um die Metadaten-Grundstruktur, wie sie in einer Salesforce-Organisation vorkommt, in einem Repository abzubilden. Ebenso kann GitLab ausschließlich als CI/CI Automationswerkzeug dienen, indem ein Git-Repository eines anderen Anbieters verwendet wird. Nachdem das Projekt über die Github-URL erzeugt wurde, kann das Projekt in GitLab konfiguriert werden. Abbildung 8 stellt die Projektübersicht in GitLab dar.

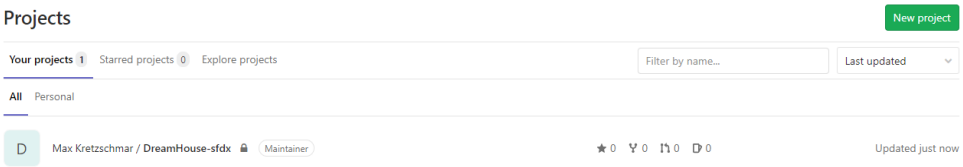


Abbildung 8: Projektübersicht in GitLab (Eigene Darstellung)

Über die Projektansicht der Dreamhouse-App kann im folgenden Schritt über einen clone-Befehl das Remote Repository in ein lokales Repository kopiert werden.

```
git clone https://gitlab.com/ max.kretzschmar/DreamHouse-sfdx.git
```

Git findet das Repository automatisch durch Angabe des Providers (GitLab) und des Benutzernamens (max.kretzschmar). Nach der Eingabe in die Kommando-zeile erscheint eine Erfolgsmeldung wie Abbildung 9 darstellt.

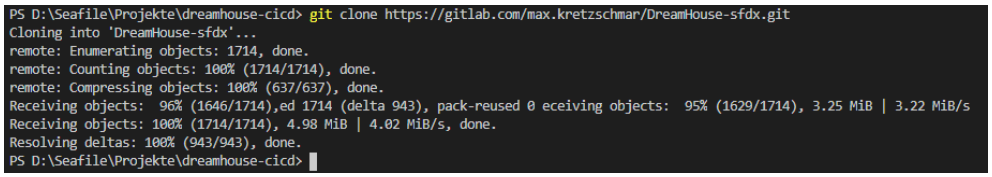


Abbildung 9: Erfolgsmeldung nach git clone (Eigene Darstellung)

Es befindet sich nun ein lokales Repository auf dem Rechner des Entwicklers, welches mit einem Remote Repository in GitLab verbunden ist. Zusätzlich findet nun die gesamte Versions- und Quellenkontrolle über GitLab statt, wodurch im nächsten Schritt eine CI/CD Pipeline eingerichtet werden kann. Abbildung 10 zeigt die Struktur innerhalb des Projektes, wie sie in einem Verzeichnis nach der Erzeugung eines lokalen Repositories vorkommt.

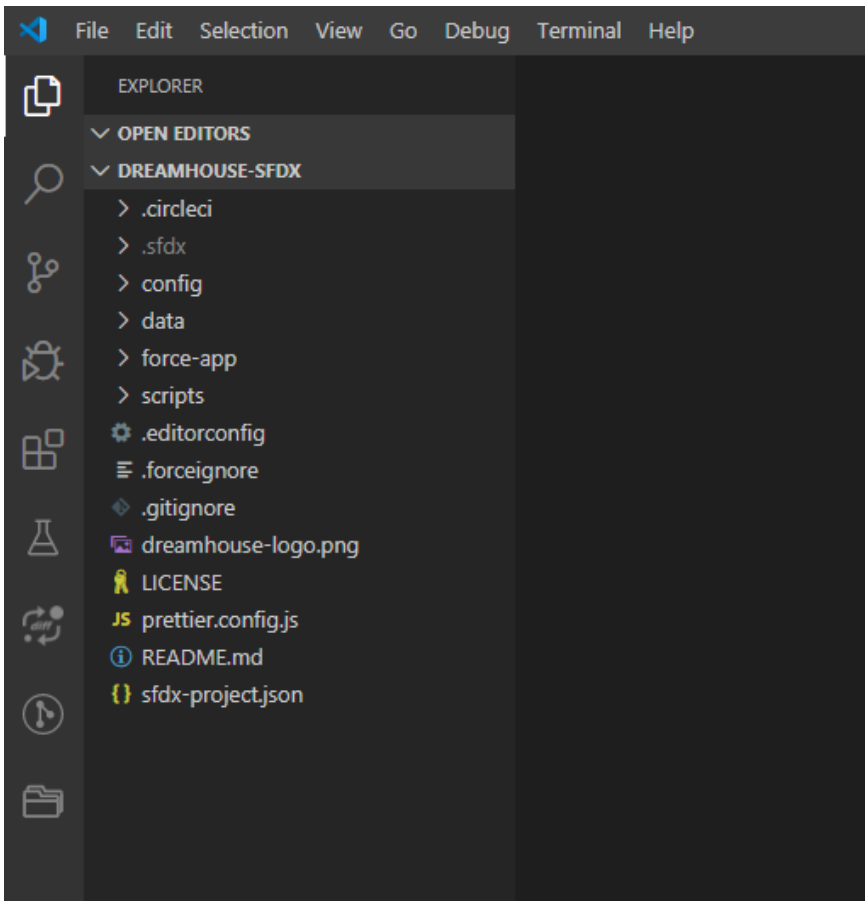


Abbildung 10: Verzeichnisstruktur eines Salesforce Projektes (Eigene Darstellung)

3.2.4. GitLab Integration und Umgebungsvariablen

Nachdem sowohl ein GitLab Projekt als auch ein Repository für die Entwicklung eingerichtet wurde, muss im nächsten Schritt die Kommunikation zwischen der Salesforce Org und GitLab aufgebaut werden. Für die Kommunikation ist eine Authentifizierung erforderlich, welche mit einem Prozess auf Basis von JSON Web Tokens (JWT) stattfindet. JWT ist ein offener Standard, durch welchen Informationen zwischen Systemen als JSON-Objekt ausgetauscht werden. Diese JSON-Objekte sind digital signiert, wodurch es in der Kommunikation als vertrauenswürdiges Verfahren angesehen wird (JWT, kein Datum).

Mit der OpenSSL Bibliothek, ein kryptografisches Werkzeug zur Erzeugung von Zertifikaten (OpenSSL Software Foundation, 2018), wird ein digitales Zertifikat erzeugt, mit welchem die automatische Kommunikation zwischen GitLab und Salesforce authentifiziert wird. Im Verzeichnis, in welchem sich auch der Dreamhouse-Ordner befindet, wird ein neuer Ordner namens *certificates* angelegt.

In diesem Ordner wird ein neuer RSA-Schlüssel mit einer Länge von 4096 bit erzeugt, mit welchem daraufhin eine Schlüsseldatei erstellt werden kann. Abbildung 11 zeigt die Abfolge der Befehle.

```
PS D:\Seafile\Projekte\dreamhouse-cicd> cd certificates
PS D:\Seafile\Projekte\dreamhouse-cicd\certificates> openssl genrsa -des3 -passout pass:MediaSystems2020 -out server.pass.key 4096
Generating RSA private key, 4096 bit long modulus (2 primes)
.....................++++
.....................................................................................................................................++++
e is 65537 (0x010001)
PS D:\Seafile\Projekte\dreamhouse-cicd\certificates> openssl rsa -passin pass:MediaSystems2020 -in server.pass.key -out server.key
writing RSA key
PS D:\Seafile\Projekte\dreamhouse-cicd\certificates> del server.pass.key
```

Abbildung 11: Befehle zur Erzeugung einer RSA Schlüsseldatei (Eigene Darstellung)

Abschließend wird das eigentliche Zertifikat erzeugt. *Server.key* beinhaltet den privaten Schlüssel. In der *server.csr* befindet sich der so genannten *Certificate Signing Request*, welchen wir für das Signieren benötigen. Aus diesen Dateien werden zusätzlich die *server.crt* – das Seitenzertifikat - und *server.key.enc* – eine verschlüsselte Version des privaten Schlüssels - generiert. Die gesamte Abfolge der Erstellung der Zertifikatsdateien und des Verschlüsselns des privaten Schlüssels ist in Abbildung 12 aufgeführt. Nach der Verschlüsselung kann die *server.key* Datei sicher im Git Repository abgelegt werden.

```
PS D:\Seafile\Projekte\dreamhouse-cicd\certificates> openssl req -new -key server.key -out server.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:CA
Locality Name (eg, city) []:San Francisco
Organization Name (eg, company) [Internet Widgits Pty Ltd]:DreamHouse Realty
Organizational Unit Name (eg, section) []:Trailhead
Common Name (e.g. server FQDN or YOUR name) []:deamhoouse.io
Email Address []:kretzschmar@hanseflow.de

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
PS D:\Seafile\Projekte\dreamhouse-cicd\certificates> openssl x509 -req -sha256 -days 365 -in server.csr -signkey server.key -out server.crt
Signature ok
subject=C = US, ST = CA, L = San Francisco, O = DreamHouse Realty, OU = Trailhead, CN = deamhoouse.io, emailAddress = kretzschmar@hanseflow.de
Getting Private key
PS D:\Seafile\Projekte\dreamhouse-cicd\certificates> openssl enc -aes-256-cbc -md sha256 -salt -e -in server.key -out server.key.end -k MediaSystemsSS2020 -pbkdf2
```

Abbildung 12: Befehle zur Erzeugung der Zertifikatsdateien (Eigene Darstellung)

Über *git add*, *commit* und *push* wird das Zertifikat in dem Repository abgelegt und kann nun in Salesforce und GitLab verwendet werden. Über das Salesforce Setup User Interface wird es in der Connected App hinterlegt.

Der letzte Schritt für die Einrichtung der Kommunikation ist die Hinterlegung der durch die Salesforce Connected App erzeugten Credentials in GitLab.

Über die CI/CD-Einstellungen in GitLab für das Dreamhouse-Projekt werden vier sog. *Umgebungsvariablen* hinterlegt. Umgebungsvariablen sind Werte, die für einen Ablauf - wie in diesem Beispiel die Kommunikation - genutzt werden. Sie werden angelegt, um ein direktes Referenzieren in diesem Vorgang und damit eine Verletzung der Datenschutzrichtlinien zu vermeiden. Auf sie kann während der gesamten Kommunikation zugegriffen werden. Sollten sich diese Werte ändern, so müssen sie lediglich an einer Stelle angepasst werden. Abbildung 13 zeigt die Übersicht der Umgebungsvariablen in GitLab

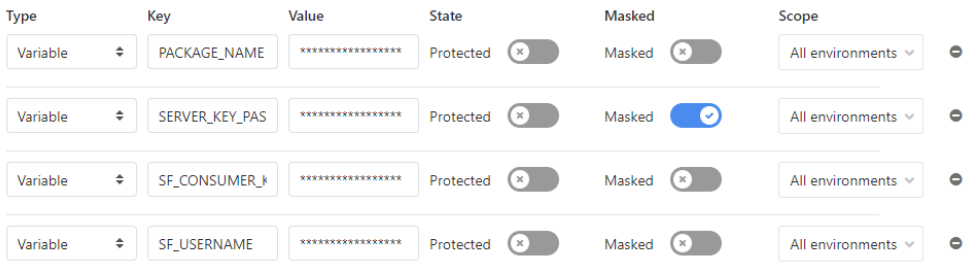


Abbildung 13: Umgebungsvariablen in GitLab (Eigene Darstellung)

PACKAGE_NAME ist der Name der Anwendung. *SERVER_KEY_PASSWORD* ist das Passwort, welches während der Verschlüsselung des privaten Schlüssels verwendet wurde. *SF_CONSUMER_KEY* ist der Wert des Consumer Key Credential, der von Salesforce in der Connected App erstellt wurde. *SF_USERNAME* ist der Nutzername im Trailhead Playground.

3.2.5. CI/CD Pipeline und Automation

Nachdem sowohl das SCM-System inklusive Versionskontrolle eingerichtet und Salesforce für die Kommunikation mit GitLab vorbereitet wurde, können nun Prozesse automatisiert werden. Um die Automation mittels CI/CD zu demonstrieren, werden so genannte *Unlocked Packages* erzeugt und implementiert. Ein Package ist ein Container, welcher mit Metadaten gefüllt wird. Dieser Container dient als Einheit, die in Salesforce Organisationen implementiert werden kann. Es existieren verschiedene Arten von Packages. Charakteristisch für Unlocked Packages ist die Flexibilität nach der Bereitstellung.

Die Metadaten in Unlocked Packages, welche implementiert wurden, können direkt in der Produktionsumgebung verändert werden (Salesforce, 2020). Um die Automationen im weiteren Verlauf zu testen, wird von der Dreamhouse-Anwendung ein Unlocked Package erzeugt.

Damit Bereitstellungs- und Testprozesse automatisiert werden können, wird in GitLab eine *Pipeline* aufgebaut. Pipelines sind Abfolgen von Skriptbefehlen, die durch einen Auslöser – ein sog. *Trigger* – gestartet werden. In dem Repository wird eine yml-Datei erstellt, die alle Befehle enthält, welche automatisch durchgeführt werden sollen. Die Datei wird mit einer Reihe von Anweisungen gefüllt, die zu bestimmten Zeiten ausgeführt werden. Vor der Ausführung des eigentlichen Skripts wird ein *before_script*-Block eingefügt, welcher Werte und Bedingungen für die Pipeline festlegt. In Abbildung 14 ist beispielhaft ein Block zu sehen, der den RSA-Schlüssel entschlüsselt, Bibliotheken installiert, Umgebungsvariablen deklariert und das Salesforce CLI installiert.

```
before_script:
    # Decrypt server key
    - openssl enc -aes-256-cbc -md sha256 -salt -d -in assets/server.key.enc -out assets/server.key -k $SERVER_KEY_PASSWORD -pbkdf2
    # Install jq, a json parsing library
    - apt update && apt -y install jq
    # Setup SFDX environment variables
    # https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_cli_env_variables.htm
    - export SALESFORCE_CLI_URL=https://developer.salesforce.com/media/salesforce-cli/sfdx-linux-amd64.tar.xz
    - export SFDX_AUTOUPDATE_DISABLE=false
    - export SFDX_USE_GENERIC_UNIX_KEYCHAIN=true
    - export SFDX_DOMAIN_RETRY=600
    - export SFDX_LOG_LEVEL=DEBUG
    # Install Salesforce CLI
    - mkdir sfdx
    - wget -qO- $SALESFORCE_CLI_URL | tar xJ -C sfdx --strip-components 1
    - './sfdx/install'
    - export PATH=./sfdx/$(pwd):$PATH
    # Output CLI version and plug-in information
    - sfdx update
    - sfdx --version
    - sfdx plugins --core
```

Abbildung 14: before_script Anweisungsblock (Eigene Darstellung)

Für das Beispiel der Dreamhouse-Anwendung folgt dem Anweisungsblock die Aufteilung der Pipeline in drei Stadien – Codetest, Integrationstest und Bereitstellung der Anwendung. Im ersten Stadium wird eine Scratch Org erstellt, um den Code zu testen. Dabei ruft das Skript eine Reihe von Salesforce CLI-Befehlen auf, welche die Eingabe eines Entwicklers in die Kommandozeile simulieren. Abbildung 15 demonstriert die Befehle des ersten Stadiums.

```
code-testing:
    stage: code-testing
    script:
        # Authenticate to the Dev Hub using the server key
        - sfdx force:auth:jwt:grant --setdefaultdevhubusername --clientid $SF_CONSUMER_KEY --jwtkeyfile assets/server.key --username $SF_USERNAME
        # Create scratch org
        - sfdx force:org:create --setdefaultusername --definitionfile config/project-scratch-def.json --wait 10 --durationdays 7
        - sfdx force:org:display
        # Push source to scratch org (this is with source code, all files, etc)
        - sfdx force:source:push
        # Assign DreamHouse permission set to scratch org default user
        - sfdx force:user:permset:assign --permsetname DreamHouse
        # Add sample data into app
        - sfdx force:data:tree:import --plan data/sample-data-plan.json
        # Unit Testing
        - sfdx force:apex:test:run --wait 10 --resultformat human --codecoverage --testlevel RunLocalTests
        # Delete Scratch Org
        - sfdx force:org:delete --noprompt
```

Abbildung 15: Pipeline-Befehle für den Codetest (Eigene Darstellung)

Die Originaldatei für die Dreamhouse-Anwendung umfasst 137 Zeilen Code, welcher in der Gesamtheit die Pipeline darstellt. Nach dem Codetest-Stadium folgt ein Integrationstest, in welchem eine weitere Scratch Org erstellt und der zuvor getestete Code in einem Package ausgeliefert wird. Nach einer weiteren Reihe an Skriptbefehlen schließt das dritte Stadium die Pipeline ab. Im letzten Schritt wird das Package in einer Sandbox implementiert, um Nutzertests durchführen zu können. Dieser Schritt wird im Dreamhouse-Beispiel manuell gestartet. Eine Pipeline ist nicht zwangsläufig vollautomatisch. Vielmehr stellt sie eine Abfolge sowohl automatischer als auch manueller Schritte dar, die nach dem Eintreten einer bestimmten Situation ausgelöst werden sollen.

Die gesamte Pipeline der Dreamhouse-Anwendung ist im Anhang 3 dokumentiert. Nachdem die yml-Datei dem Repository hinzugefügt wird, erkennt GitLab automatisch eine Pipeline und stellt diese grafisch dar. Abbildung 16 zeigt die Pipeline für die Dreamhouse Anwendung.

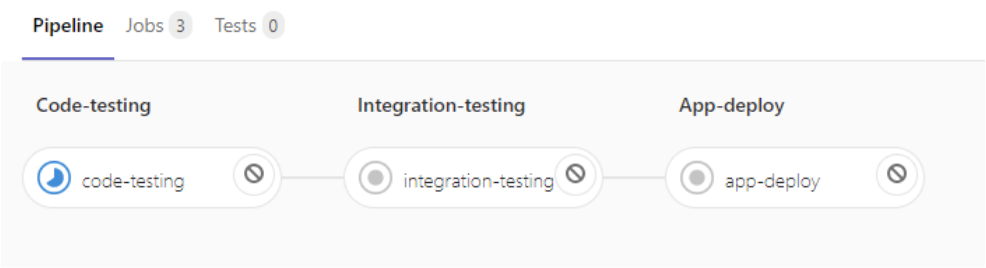


Abbildung 16: Pipeline der Dreamhouse Anwendung (Eigene Darstellung)

Diese Pipeline wird bei jedem commit zum Repository automatisch ausgeführt. In diesem Beispiel wird das *Code*- und *Integration-Testing* automatisch ausgeführt. Lediglich das *App-Deployment*, also die Bereitstellung nach erfolgreichen Funktions- und Integrationstests, ist manuell.

In der Praxis ist es sinnvoll, manuelle Schritte in eine sonst automatische Pipeline einzubauen. Besonders bei der finalen Bereitstellung von Änderungen in der Produktionsumgebung sollte sichergestellt werden, dass alle Tests im Vorfeld erfolgreich waren. Nichtsdestotrotz kann auch dieser Schritt automatisiert werden. Mit dem letzten Schritt der Bereitstellung wird das Unlocked Package der Dreamhouse-Anwendung im DevHub bereitgestellt, womit die Änderungen, die in der lokalen Entwicklungsumgebung durchgeführt wurden, zur Verfügung stehen. Damit ist der gesamte Prozess von der lokalen Entwicklung über die Synchronisation mit dem Versionskontrollsystem inklusive automatischen Tests bis hin zur Bereitstellung in einer Produktionsumgebung aufgebaut. Abbildung 17 demonstriert die Meldung der Pipeline nach erfolgreicher Bereitstellung des Paketes.

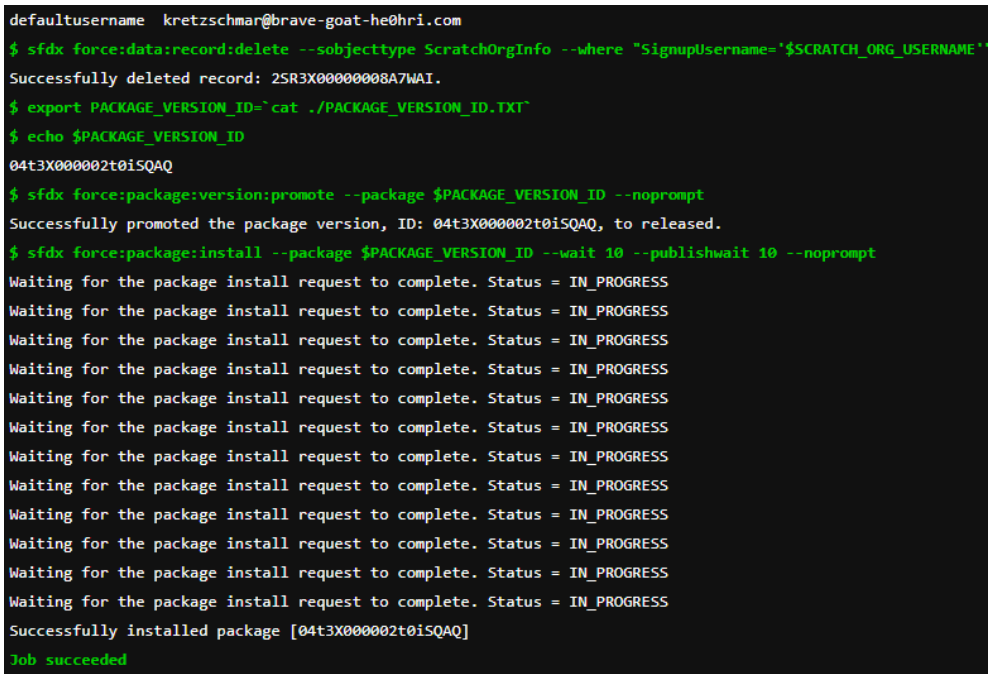


Abbildung 17: Erfolgreiche Bereitstellung des Packages (Eigene Darstellung)

3.3. Ausblick

Das neue Entwicklungsmodell kann nach der erfolgreichen Einrichtung auf alle Kundenprojekte ausgeweitet werden. Die Pipeline der Dreamhouse-Anwendung diente als Beispiel für diese Arbeit, jedoch beinhaltet sie bereits essenzielle Konfigurationen, welche eine Pipeline eines Kundenprojektes unter realen Entwicklungsumständen beinhalten würde. Zusätzlich zur Erweiterung und Anpassung der GitLab Pipeline auf die individuellen Bedürfnisse der Entwickler des Unternehmens, werden Apex-Funktionstests erstellt, um sich wiederholenden Standardcode innerhalb der SalesforceOrganisation effektiv zu testen. Der Entwicklungsprozess dient vorranging der Verbesserung der Arbeitsbedingungen der Entwickler, jedoch können weitere Maßnahmen im Rahmen von DevOps getroffen werden. Da Unlocked Packages Metadaten jeglicher Art, wie z. B. Seitenlayouts und Feldkonfigurationen, beinhalten können, kann der Prozess ebenfalls auf den IT-Betrieb und die Arbeit von Salesforce-Administratoren ausgeweitet werden. Um Feedback über den Stand der Tests zu erhalten, kann die Pipeline dahingehend erweitert werden, die Entwickler regelmäßig über den Status von Tests und Bereitstellungen zu informieren. Grundsätzlich ist die bisher aufgebaute Pipeline die Basis für eine Reihe von notwendigen Erweiterungen, die jedoch individueller Natur sind.

4. Fazit

Das Ziel dieser Arbeit war es, die Vorteile eines quellengesteuerten Entwicklungsprozesses nach DevOps-Prinzipien herauszufinden und in einer Salesforce-Partner-Agentur in die Praxis umzusetzen. Nachdem grundlegende Erkenntnisse zu DevOps und Salesforce erworben wurden, konnten wichtige Vergleiche zwischen dem veralteten organisationsbasierten Salesforce-Entwicklungsmodell und einem neuen quellengesteuerten Modell gezogen werden.

Die ursprüngliche Vermutung, dass ein nach DevOps-Prinzipen ausgerichteter Entwicklungszyklus mit modernen Bereitstellungsmethoden effizienter, sicherer und kontrollierter ist, hat sich durch die Umsetzung in die Praxis bewahrheitet. Durch die Ausführung von grundlegenden Befehlen in einer Pipeline, welche durch Auslöser gestartet wird, wurde ein Großteil des Bereitstellungsaufwandes automatisiert. Ein Versionskontrollsystem, welches den gesamten Entwicklungszyklus überwacht, lässt Fehler schnell erkennen und einfach beheben. Durch eine Kommunikation der Systeme auf Basis eines Sicherheitszertifikates wurde die Sicherheit erhöht. Der Aufbau einer Pipeline in einem Software-Verwaltungssystem ist durch den hohen Grad an Flexibilität bei der Anpassung der Auslöser und Befehle skalierbar und kann auf die individuellen unternehmensinternen Prozesse abgestimmt werden.

Die in dieser Arbeit erlangten Erkenntnisse lassen sich unabhängig von der Größe des Entwicklerteams auf jede Salesforce-Partner-Agentur übertragen. Da es sich bei DevOps mehr um eine Sammlung von Prinzipien handelt als einen fest definierten Ablauf, können die Grundsätze in jede beliebige interne Prozesskette eingearbeitet werden. Sowohl Salesforce DX als auch das Salesforce CLI haben großes Potential durch die Nutzung moderner, etablierter Verfahren. Auf Grundlage dieser Arbeit kann ein modernes Salesforce-Entwicklungsmodell umgesetzt und individuell erweitert werden, um die Entwicklungs- und Bereitstellungsarbeit zukünftig an aktuellen Standards auszurichten.

Literaturverzeichnis

Augsten, S. (2017). *Was ist SCM?* Retrieved März 2, 2020, from Dev-Insider: https://www.dev-insider.de/was-ist-scm-a-623224/

Commonplaces Interactive. (2018). *Web Development: What is Staging?* Retrieved Februar 15, 2020, from commonplaces: https://www.commonplaces.com/blog/web-development-what-is-staging/

Davis, A. (2019). *Mastering Salesforce DevOps.* Apress.

Dev-Insider. (2017). *Die DevOps-Bewegung.* Retrieved from https://www.dev-insider.de/die-devops-bewegung-v-37889-14931/

Gene Kim, K. B. (2013). *The Phoenix Project: A Novel about it, Devops, and Helping Your Business Win.* IT Revolution Press.

JWT. (n.d.). *What is JSON Web Token?* Retrieved März 2, 2020, from jwt.io: https://jwt.io/introduction/

Kharnagy. (2016). *A visual representation of the DevOps workflow.* Retrieved Januar 23, 2020, from Wikipedia: https://commons.wikimedia.org/wiki/File:Devops-toolchain.svg

Kim, G. (2012). *The Three Ways: The Principles Underpinning DevOps*. Retrieved Februar 4, 2020, from IT Revolution: http://itrevolution.com/the-three-ways-principles-underpinning-devops/

Kraft, K. (2017). *Salesforce DX allgemein verfügbar*. Retrieved Februar 21, 2020, from heise.de: https://www.heise.de/developer/meldung/Salesforce-DX-allgemein-verfuegbar-3864511.html

Microsoft. (2020). *Was ist Cloud Computing?* Retrieved Januar 29, 2020, from Microsoft Azure: https://azure.microsoft.com/de-de/overview/what-is-cloud-computing/

Mirco Lang, S. A. (2019). *5 freie Versionskontrollsysteme im Überblick*. Retrieved Februar 13, 2020, from Dev-Insider: https://www.dev-insider.de/5-freie-versionskontrollsysteme-im-ueberblick-a-835250/

Nägele, V. (n.d.). *Was ist die AWS Cloud?* Retrieved Januar 29, 2020, from cloudmag: https://www.cloud-mag.com/was-ist-aws-cloud/

New Relic. (n.d.). *What Is DevOps*. Retrieved Januar 17, 2020, from https://newrelic.com/devops/what-is-devops

OpenSSL Software Foundation. (2018). *OpenSSL*. Retrieved März 2, 2020, from OpenSSL, Cryptography and SSL/TLS Toolkit: https://www.openssl.org/

Salesforce. (2019). *Build an Automated CI/CD Pipeline with GitLab*. Retrieved Februar 29, 2020, from Trailhead: https://trailhead.salesforce.com/en/content/learn/projects/automate-cicd-with-gitlab

Salesforce. (2020). *Unlocked Packages for Customers*. Retrieved März 2, 2020, from Trailhead: https://trailhead.salesforce.com/en/content/learn/modules/unlocked-packages-for-customers/break-up-your-metadata

Salesforce. (n.d.). *Continuous Delivery – schnellere, kontinuierliche Bereitstellung Ihrer Salesforce Anwendungen*. Retrieved Februar 21, 2020, from Salesforce.com: https://www.salesforce.com/de/products/platform/products/salesforce-dx/

Salesforce Developers. (2019). *Accelerate DevOps with GitLab and Salesforce*. Retrieved Februar 28, 2020, from YouTube: https://www.youtube.com/watch?v=tylPp9QlLu4

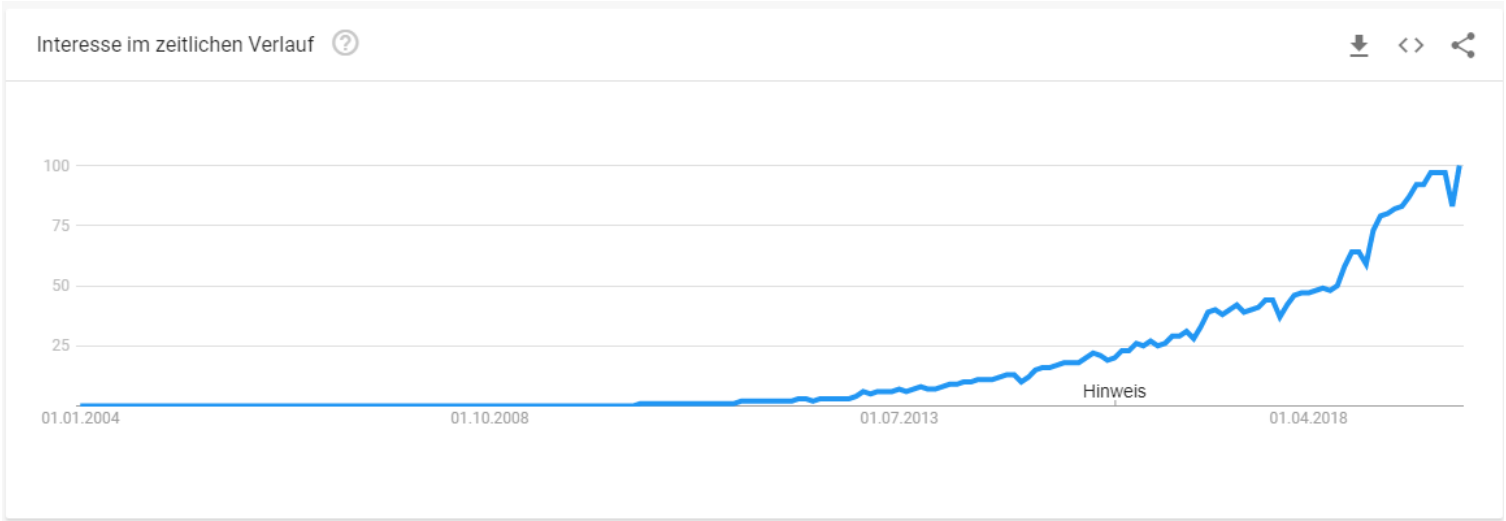
Weins, K. (2016). *New DevOps Trends: 2016 State of the Cloud Survey*. Retrieved Januar 21, 2020, from https://www.flexera.com/blog/cloud/2016/05/new-devops-trends-2016-state-of-the-cloud-survey/

Abbildungsverzeichnis

Anhangsverzeichnis

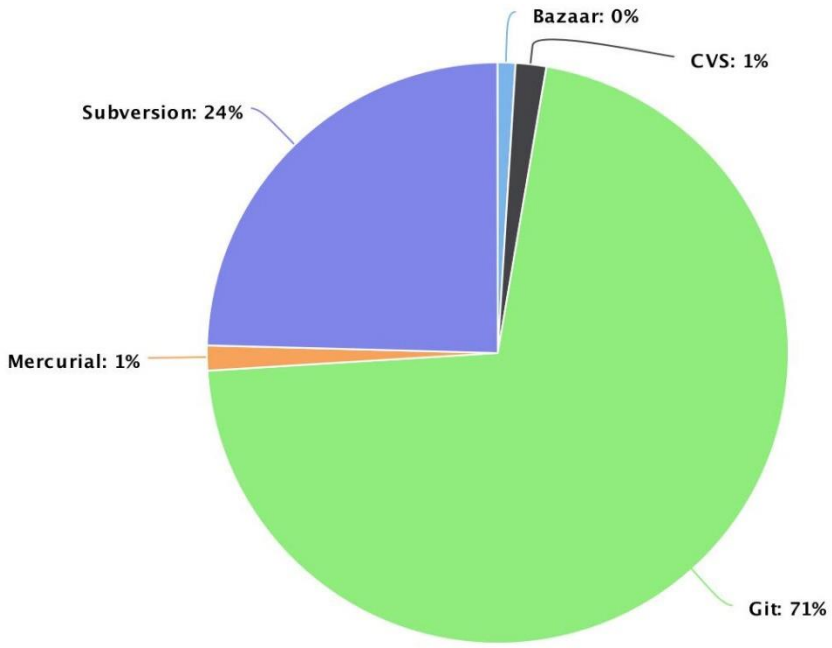
Anhang 1:

Google Trends Suche zu dem Begriff „DevOps“



Anhang 2:

OpenHub Chart: Repositories im Vergleich



Anhang 3:

Inhalt der YML-Datei des Beispiels „Dreamhouse“

```
before_script:
    # Decrypt server key
    - openssl enc -aes-256-cbc -md sha256 -salt -d -in assets/server.key.enc -out assets/server.key -
k $SERVER_KEY_PASSWORD -pbkdf2
    # Install jq, a json parsing library
    - apt update && apt -y install jq
    # Setup SFDX environment variables
    # https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_cli_env_variables.htm
    - export SALESFORCE_CLI_URL=https://developer.salesforce.com/media/salesforce-cli/sfdx-linux-amd64.tar.xz
    - export SFDX_AUTOUPDATE_DISABLE=false
    - export SFDX_USE_GENERIC_UNIX_KEYCHAIN=true
    - export SFDX_DOMAIN_RETRY=600
    - export SFDX_LOG_LEVEL=DEBUG
    # Install Salesforce CLI
    - mkdir sfdx
    - wget -qO- $SALESFORCE_CLI_URL | tar xJ -C sfdx --strip-components 1
    - './sfdx/install'
    - export PATH=./sfdx/$(pwd):$PATH
    # Output CLI version and plug-in information
    - sfdx update
    - sfdx --version
    - sfdx plugins --core
#
# Define the stages of our pipeline
#
stages:
    - code-testing
    - integration-testing
    - app-deploy
#
# Stage 1 -- Create a scratch org for code testing
#
code-testing:
    stage: code-testing
    script:
        # Authenticate to the Dev Hub using the server key
        - sfdx force:auth:jwt:grant --setdefaultdevhubusername --clientid $SF_CONSUMER_KEY --
jwtkeyfile assets/server.key --username $SF_USERNAME
        # Create scratch org
        - sfdx force:org:create --setdefaultusername --definitionfile config/project-scratch-def.json --
wait 10 --durationdays 7
        - sfdx force:org:display
        # Push source to scratch org (this is with source code, all files, etc)
        - sfdx force:source:push
        # Assign DreamHouse permission set to scratch org default user
        - sfdx force:user:permset:assign --permsetname DreamHouse
        # Add sample data into app
        - sfdx force:data:tree:import --plan data/sample-data-plan.json
        # Unit Testing
        - sfdx force:apex:test:run --wait 10 --resultformat human --codecoverage --testlevel RunLocalTests
        # Delete Scratch Org
        - sfdx force:org:delete --noprompt
#
# Stage 2 -- Create a scratch org, create a package version, and push into org for testing
#
integration-testing:
    # Specify file paths that we want available
    # in downstream build stages for this pipeline execution.
    # This is a way to pass dynamic values from one stage to another.
    artifacts:
        paths:
            - PACKAGE_VERSION_ID.TXT
            - SCRATCH_ORG_USERNAME.TXT
    stage: integration-testing
    script:
        # Authenticate to the Dev Hub using the server key
        - sfdx force:auth:jwt:grant --setdefaultdevhubusername --clientid $SF_CONSUMER_KEY --
jwtkeyfile assets/server.key --username $SF_USERNAME
        # Create scratch org
        - sfdx force:org:create --setdefaultusername --definitionfile config/project-scratch-def.json --
wait 10 --durationdays 7
        - sfdx force:org:display
        # Increment package version number
        - echo $PACKAGE_NAME
        - PACKAGE_VERSION_JSON="$(eval sfdx force:package:version:list --concise --released --
packages $PACKAGE_NAME --json | jq '.result | sort_by(-.MajorVersion, -.MinorVersion, -.PatchVersion, -
.BuildNumber) | .[0] // ""')"
        - echo $PACKAGE_VERSION_JSON
        - IS_RELEASED=$(jq -r '.IsReleased?' <<< $PACKAGE_VERSION_JSON)
        - MAJOR_VERSION=$(jq -r '.MajorVersion?' <<< $PACKAGE_VERSION_JSON)
        - MINOR_VERSION=$(jq -r '.MinorVersion?' <<< $PACKAGE_VERSION_JSON)
        - PATCH_VERSION=$(jq -r '.PatchVersion?' <<< $PACKAGE_VERSION_JSON)
        - BUILD_VERSION="NEXT"
        - if [ -z $MAJOR_VERSION ]; then MAJOR_VERSION=1; fi;
        - if [ -z $MINOR_VERSION ]; then MINOR_VERSION=0; fi;
        - if [ -z $PATCH_VERSION ]; then PATCH_VERSION=0; fi;
        - if [ "$IS_RELEASED" == "true" ]; then MINOR_VERSION=$(($MINOR_VERSION+1)); fi;
```

```
        - VERSION_NUMBER="$MAJOR_VERSION.$MINOR_VERSION.$PATCH_VERSION.$BUILD_VERSION"
        - echo $VERSION_NUMBER
        # Create packaged version
        - export PACKAGE_VERSION_ID="$(eval sfdx force:package:version:create --package $PACKAGE_NAME --
versionnumber $VERSION_NUMBER --installationkeybypass --wait 10 --json | jq -
r '.result.SubscriberPackageVersionId')"
        # Save your PACKAGE_VERSION_ID to a file for later use during deploy so you know what version to deploy
        - echo "$PACKAGE_VERSION_ID" > PACKAGE_VERSION_ID.TXT
        - echo $PACKAGE_VERSION_ID
        # Install package in DevHub org (this is a compiled library of the app)
        - sfdx force:package:list
        - sfdx force:package:install --package $PACKAGE_VERSION_ID --wait 10 --publishwait 10 --noprompt
        # Assign DreamHouse permission set to scratch org default user
        - sfdx force:user:permset:assign --permsetname DreamHouse
        # Add sample data into app
        - sfdx force:data:tree:import --plan data/sample-data-plan.json
        # Run unit tests in scratch org
        - sfdx force:apex:test:run --wait 10 --resultformat human --codecoverage --testlevel RunLocalTests
        # Get the username for the scratch org
        - export SCRATCH_ORG_USERNAME="$(eval sfdx force:user:display --json | jq -r '.result.username')"
        - echo "$SCRATCH_ORG_USERNAME" > ./SCRATCH_ORG_USERNAME.TXT
        # Generate a new password for the scrach org
        - sfdx force:user:password:generate
        - echo -e "\n\n\n\n"
        # Display username, password, and instance URL for login
        # Be careful not to do this in a publicly accessible pipeline as it exposes the credentials of your scr
atch org
        - sfdx force:user:display
#
# Stage 3 -- Promote the package to downstream environment for UAT for example
#
app-deploy:
    stage: app-deploy
    # This stage must be started manually as an example of
    # conditional stages that need to wait for an approval,
    # such as waiting for QA signoff from the previous stage.
    when: manual
    script:
        # Read the scratch org username from file created in prior stage
        - export SCRATCH_ORG_USERNAME=`cat ./SCRATCH_ORG_USERNAME.TXT`
        - echo $SCRATCH_ORG_USERNAME
        # Authenticate with your playground or sandbox environment
        - sfdx force:auth:jwt:grant --setdefaultdevhubusername --clientid $SF_CONSUMER_KEY --
jwtkeyfile assets/server.key --username $SF_USERNAME
        - sfdx force:config:set defaultusername=$SF_USERNAME
        # Delete Scratch Org that you were inspecting from your browser
        - sfdx force:data:record:delete --sobjecttype ScratchOrgInfo --
where "SignupUsername='$SCRATCH_ORG_USERNAME'"
        # Read the package version id from file created in prior stage
        - export PACKAGE_VERSION_ID=`cat ./PACKAGE_VERSION_ID.TXT`
        - echo $PACKAGE_VERSION_ID
        # Promote the package version
        - sfdx force:package:version:promote --package $PACKAGE_VERSION_ID --noprompt
        # Install the package version
        - sfdx force:package:install --package $PACKAGE_VERSION_ID --wait 10 --publishwait 10 --noprompt
```

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelor-Thesis mit dem Titel:

Salesforce DevOps-Strategien

Continuous Integration, Delivery und Deployment in der Praxis

selbständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.