Kapitel 1
Einführung

Dieses Buch richtet sich hauptsächlich an interessierte Laien in der Softwareentwicklung und dem dazugehörigen Projektmanagement, aber ich hoffe, dass Sie auch von ihm profitieren können, wenn Sie schon etwas mehr Erfahrung in der Organisation von Entwicklungsprojekten haben. Sie sollten auf jeden Fall bereits Kenntnisse im Programmieren mitbringen. Optimalerweise kennen Sie sich bereits mit Build-Werkzeugen wie Maven oder Gradle aus.

Auch wenn für viele der beschriebenen Zusammenhänge Beispiele aus der Java-Welt herangezogen wurden, können diese auch auf andere Programmiersprachen und Plattformen übertragen werden. Damit dies möglichst nahtlos gelingt, habe ich viel Wert darauf gelegt, Ihnen etablierte Standards nahezubringen und auch selbst anzuwenden.

1.1 Was Sie schon wissen sollten und was Sie lernen werden

Das Buch beginnt mit einem Theorieteil aus sechs Kapiteln. Danach folgen wir im Praxisteil den logischen Schritten eines Build-Jobs. Die Kapitel sind in sich geschlossen und können nach Interessenlage auch einzeln gelesen werden. Wenn Sie die grundsätzlichen Überlegungen über die Organisation von Softwareentwicklungsprojekten also erst einmal überspringen möchten, können Sie auch gleich mit Kapitel 7 loslegen.

Nach der Lektüre dieses Buches sind Sie hoffentlich nicht nur gut unterhalten, sondern auch in der Lage, einen eigenen Jenkins-Automatisierungsserver für Ihre Projekte in Betrieb zu nehmen. Zudem finden Sie Hintergrundinformationen, mit denen Sie bestehende Build-Infrastrukturen optimieren können. Dabei erhebe ich aber nicht den Anspruch, die absolute Wahrheit zu verkünden, denn viele Praktiken im Umgang mit Automatisierungswerkzeugen sind genauso Modeerscheinungen wie neue Ansätze im Projektmanagement: Hier ändert sich viel, und stetig ist eigentlich nur der Wandel.

Idealerweise basiert das Setup Ihrer Arbeitsumgebung auf einem Linux-Betriebssystem. Die Projekte, die Sie automatisieren möchten, gehören zur Java-Plattform und besitzen bereits einen funktionierenden Maven- oder Gradle-Build. Als Arbeitsum-

gebung verwende ich die freie IDE NetBeans, da diese hervorragende Werkzeuge bereitstellt und neben dem Schreiben des Quelltextes auch ein effizientes Bearbeiten der Build-Logik ermöglicht. Es steht Ihnen aber natürlich frei, die Werkzeuge Ihrer Wahl zu verwenden. Selbst reine Texteditoren wie der *vi* oder Notepad führen zum Ziel, sind allerdings nicht gleichzusetzen mit dem, was eine vollwertige IDE ausmacht. Wenn Sie beabsichtigen, sich ernsthaft mit professioneller Softwareentwicklung zu befassen, kommen Sie an den spezialisierten Werkzeugen wie beispielsweise IDEA IntelliJ, Eclipse, Visual Studio Code und vergleichbaren nicht vorbei.

Dateigröße

Die meisten ASCII-Texteditoren kommen schnell an ihre Grenzen, wenn die zu öffnende Datei mehr als 1 MB an Inhalt enthält. Übliche Beispiele für solche schwergewichtigen Textdateien sind Datenbankexporte. Hier empfehlen sich Sublime Text (*https://www.sublimetext.com*) oder Atom (*https://atom.io*), die sowohl für Linux als auch für Windows frei verfügbar sind und eine gute Mischung aus kompletter Entwicklungsumgebung und leichtgewichtigen Editoren darstellen.

Vielleicht ist dies für Sie selbstverständlich und Sie fragen sich, ob es überhaupt (noch) Leute gibt, die solche modernen Entwicklungswerkzeuge nicht benutzen. Doch, die gibt es, und zwar vor allem im universitäreren Umfeld. Gerade an den Stätten, an denen die Ausbildung künftiger Ingenieure und Entwicklerinnen eine wichtige Rolle spielt, herrscht oftmals der Irrglaube vor, einem guten Programmierer oder einer guten Programmiererin genügten ein Editor und ein Compiler, um gute Software zu erstellen. Das halte ich deswegen für sehr fatal, weil die vielen jungen Talente, die mit ihrem Abschluss die Hochschulen verlassen, sich meist erst in der Arbeitswelt mit den benötigten Werkzeugen intensiv befassen können. Wer sich jedoch bereits während der Ausbildung Schritt für Schritt mit seinem Werkzeugkasten vertraut macht, gelangt schneller zum Erfolg und kann viel Frustration vermeiden.

Ich möchte Ihnen in diesem Buch einige Denkansätze und Beispiele vorstellen, mit denen Sie Entwicklungsprojekte besser organisieren können. Daher werden Sie hier viel Theorie finden, die praktisches Ausprobieren und das Experimentieren mit Code und Builds aber nicht ersetzen kann. Zu diesem Zweck finden Sie Beispiele auf GitHub in meinen persönlichen Repositories, wie etwa TP-CORE, einer kleinen Java-Bibliothek.

Für dieses Buch habe ich auch ein eigenes Verzeichnis namens *JenkinsCIBuch* im Repository *SamplesAndTrainings* zusammengestellt. Sie finden es unter:

https://github.com/ElmarDott/SamplesAndTrainings

Sie sind herzlich eingeladen, eigene Lösungen und Ideen per Pull-Request in diesem Repository einzureichen.

Sie finden die Dateien zudem auf der Seite des Verlags unter *https://www.rheinwerk-verlag.de/5172*.

1.2 Worüber wir reden: Eine CI-Pipeline für Ihre Softwareprojekte

Jenkins stellt in der Infrastruktur des Softwareentwicklungsprozesses ein besonderes Werkzeug dar, denn es vereint viele unabhängige Tools zu einem großen Ökosystem. Daher spricht man von einer *Continuous Integration Pipeline*. In ihr werden unterschiedliche Arbeitsschritte und die entsprechenden Werkzeuge ausgeführt. Dies geschieht idealerweise kontinuierlich und automatisiert, sodass Sie umgehend über auftretende Probleme bei der Erstellung Ihrer Artefakte im Bilde sind.

Eine solche Linie reicht vom Einchecken des Codes bis zum Ablegen des fertigen Produkts. Ihre Hauptaufgabe ist das schnelle Auffinden von Integrationsproblemen. So wird sichergestellt, dass sich durch die Integration neuer Features oder die Änderung bestehender Codeabschnitte keine Fehler ins Projekt einschleichen. Denn je komplexer (und älter) eine Codebasis ist, desto wahrscheinlicher ist es, dass neue Bestandteile Wechselwirkungen hervorrufen. Vielleicht passen sie einfach nur nicht richtig, oder sie führen schlimmstenfalls zu schwer nachvollziehbaren, subtilen Logikproblemen. Wenn Sie sich erst im letzten Moment darum kümmern, werden Sie nur unter großen Anstrengungen den versprochenen Liefertermin halten können. Deswegen ist das Prinzip *Fail Fast* so essenziell für die Softwareentwicklung. So erarbeiten Sie zeitnahe Lösungen, ohne dass sich Probleme aufstauen und es später zu Katastrophen kommt.

Hinzu kommt, dass über eine CI-Pipeline regelmäßig Analysen zur Bewertung der Prozess- und Produktqualität durchgeführt werden, sodass Sie immer einen Überblick über den tatsächlichen Stand haben. Denn wenn Sie gemeinsam als Team an einem Projekt arbeiten, wollen Sie nicht erst kurz vor dem vereinbarten Meilenstein über Probleme stolpern, die sich schon länger eingeschlichen haben.

Die Tage, in denen man sich als Entwickler oder Entwicklerin in ein stilles Kämmerchen einschließen konnte, um nach getaner Arbeit das Ergebnis dem Rest des Teams »über den Zaun zu werfen«, sind längst Geschichte. Eine gute Kommunikationsfähigkeit und der Wille zur Teamarbeit sind in der modernen Softwareentwicklung absolut zentral. Vielleicht sind sie sogar wichtiger als technische Fähigkeiten, die sich leichter über Schulungen ausbauen lassen. Eine persönliche Einstellung lässt sich nicht so leicht revidieren.

Deswegen führen die ersten Kapitel dieses Buches ein wenig durch die Theorie. Sie vermitteln Ihnen einen Überblick, um ein gemeinsames Verständnis vom Thema zu entwickeln. So erfahren Sie, wie sich wichtige Dienste in die Jenkins-Landschaft integrieren. Auch wenn dies für Sie auf den ersten Blick möglicherweise etwas ermüdend

und langwierig erscheinen mag, halte ich das Wissen um diese Punkte für unverzichtbar, damit Ihnen ein möglichst frustrationsfreies Arbeiten mit Jenkins gelingt. Denn wenn Sie nicht wissen, wie eine Aufgabe »per Hand« umgesetzt wird, kann ein Automatisierungswerkzeug wie Jenkins für mehr Ärger als Nutzen sorgen.

Ich stelle Ihnen daher die einzelnen Arbeitsschritte der kommerziellen Softwareentwicklung vor und zeige, wie sich die wichtigsten Standards und Werkzeuge in diese Schritte einfügen. Anhand von konkreten Fragestellungen und Beispielen lernen Sie den wohl populärsten freien Automationsserver Jenkins in seinen verschiedenen Facetten kennen.

Wortwahl: Build und Deploy

Ganz gleich, ob Sie mit Jenkins als Automatisierungsserver arbeiten oder mit einer der vielen anderen Lösungen agieren: Es haben sich historisch einige Begriffe etabliert, die bei der heutigen Komplexität der Aufgaben dieser Systeme verwirren können. Früher und auch heute wird in der Praxis sprachlich kaum zwischen einem *Build-Job* und einem *Deploy-Job* unterschieden, obwohl es sich um unterschiedliche Aufgabenbereiche handelt. Diese Ungenauigkeit finden Sie auch bei anderen Begriffen wieder, was Sie an Bezeichnungen wie *CI-Server* oder *Build-Server* schnell feststellen können.

Um ein wenig Klarheit zu schaffen, verwende ich besonders in Kontext der (Build-) Jobs für die allgemeine Variante vornehmlich die Bezeichnung *Jenkins-Job*.

Damit es nicht zu Missverständnissen kommt: Ich bespreche das Deployment aus der Perspektive der Softwareentwicklung. Es geht darum, wie möglichst einfach Artefakte erstellt und abgelegt werden. Wie diese in die Produktion gebracht werden, ist nur am Rande Thema dieses Buchs. Dies benötigt in den allermeisten Umgebungen fundiertes Wissen zur Infrastruktur, das den Fachabteilungen vorbehalten ist. Denn auch wenn eine gelebte DevOps-Kultur dafür sorgt, dass die Mauern zwischen den Abteilungen kleiner werden, züchtet sie doch keine eierlegenden Wollmilchschweine. Das finale Deployment sollte daher von Spezialisten organisiert werden, die eher der Administration zuzurechnen sind. Ich stelle Ihnen aber in Abschnitt 5.3 einige Strategien vor, mit denen Sie ein solches Deployment gut vorbereiten können.

Wenn Sie schnell ein Projekt mit Jenkins bauen wollen, steigen Sie am besten gleich mit Kapitel 2, »Jenkins für Eilige«, ein. Dort finden Sie auch eine Übersicht über die wichtigsten Begriffe, auf Sie in der modernen Softwareentwicklung stoßen werden.

Kapitel 3, »Das große B(u)ild der Automatisierung«, gibt Ihnen einen ersten Überblick über die unterschiedlichen Rollen und Aufgabenschritte beim Bauen und Deployen von Software. Danach beschäftigt uns in Kapitel 4, »Software testen – aber wie?«, das Thema Testen, das aber auf die notwendigen Aspekte reduziert wird. Auch ein kurzer Exkurs zum Release-Management und dazu, wie dieses den verschiedenen Programmierparadigmen zuzuordnen ist, ist in Kapitel 5, »Release-Management in

einer agilen Welt«, eine Station unserer Reise. Die letzte Etappe in der Theorie führt uns in Kapitel 6 zum Source-Control-Management.

Es gibt also viele Themen, die sich rund um Jenkins auftun und den Erfolg einer reibungslosen CI-Pipeline beeinflussen. Oft sind es die winzigen Details aus Entscheidungen, die bereits früh im Projektverlauf getroffen werden, die später starke Auswirkungen haben. Meist lassen sich solche vermeintlichen Kleinigkeiten nur mit erheblicher Mühe wieder korrigieren. Ich werde daher im ersten Teil dieses Buches recht ausführlich darauf eingehen.

Den praktischen Einstieg zu Jenkins beginnen wir in Kapitel 7, »Einen CI-Server mit Jenkins einrichten und betreiben«. Neben den verschiedenen Variationen der Installation besprechen wir auch wichtige Punkte der Konfiguration. Sie erfahren, wie Sie mit der Benutzerverwaltung den Weg zu ausgewählten Aktivitäten einer eingeschränkten Personengruppe zugänglich machen. Aber auch Fragen der Lastverteilung mithilfe einer dezentralen Infrastruktur werden besprochen.

In Kapitel 8, »Build-Management«, widmen wir uns den verschiedenen Charakteristika von Software-Projekten. Das schärft den Blick dafür, wie effiziente Jenkins-Jobs zur Automatisierung formuliert werden können. Denn wie Sie Build-Jobs für Jenkins schreiben, ist die wohl wichtigste Frage dieses Buchs. Darum geht es in Kapitel 9, »Jenkins-Build-Jobs«.

Natürlich dürfen auch Überlegungen zur Überwachung und Verbesserung der Codequalität nicht fehlen. Diese finden Sie in Kapitel 10, »Qualitätskontrolle«. Mit SonarQube stelle ich ein Werkzeug vor, das verschiedene Programmiersprachen beherrscht und sich hervorragend in Jenkins integrieren lässt.

Schließlich sind auch weiterführende Betrachtungen darüber, wie Sie die erzeugten Artefakte für eine erfolgreiche Kollaboration ablegen, Bestandteil dieses Buches. Hierzu stelle ich Ihnen in Kapitel 11, »Sonatype Nexus: Der Repository-Manager«, den Repository-Manager Nexus vor, in dem nicht nur Java-Artefakte abgelegt werden können. Als besonderes Bonbon erfahren Sie, wie Sie ein Java-Projekt auf Maven Central für die Allgemeinheit veröffentlichen können.

Am Ziel unseres Ausfluges haben Sie ein umfassendes Bild über die Schritte, Werkzeuge und Ergebnisse einer vollständigen CI-Pipeline gewonnen, um selbst Hand anzulegen. Also, worauf warten Sie noch? Bitte steigen Sie ein, die Türen schließen selbsttätig.