Clemens Gull

Know-how ist blau.

Web-Applikationen entwickeln mit NoSQL

> Der einfache Weg: Web-Apps mit CouchDB und CouchApp

> Erhalten Sie eine bessere Performance bei hohen Datenlasten

> Von der Datenbank bis zur fertigen Web-Applikation

Das Buch für Datenbank-Einsteiger und Profis!

FRANZIS

Inhaltsübersicht

Einführung

Worum geht es in diesem Buch?

Dieses Buch ist kein klassisches »Lernbuch«, wie wir es dutzendfach kennen. Aber es ist auch keine Referenz oder ein Kompendium für NoSQL-Datenbanken oder eine bestimmte Programmiersprache. Es ist eine praktische Einführung in das Thema NoSQL im World Wide Web.

Das Buch ist in vier Teile gegliedert. Am Anfang steht eine kurze theoretische Einführung in die Unterschiede der SQL- und NoSQL-Datenbanken. Daraus ergibt sich auch, dass wir die Art der Programmierung von Webseiten überdenken müssen, wenn wir NoSQL-Datenbanken verwenden wollen.

Der zweite Teil befasst sich mit der Installation einer NoSQL-Datenbank auf einem lokalen Computer und der Vorbereitung des Geräts auf den dritten und vierten Teil.

Der dritte Teil ermöglicht Ihnen das Kennenlernen dieser neuen Art von Datenbank und auch die Änderungen in der Programmierung von Web-Applikationen in einfachen Beispielen. Hier legen wir keinen großen Wert auf das Aussehen der Ergebnisse und auch nicht auf die Daten. Sondern es ist uns wichtig, die Funktionalität, die hinter den einzelnen Beispielen steht, zu verstehen.

Im vierten Teil werden wir ein praktisches Beispiel entwickeln. Am besten lernt man einfach aus der Praxis. Alle Dinge, die Sie selbst gemacht haben, bleiben gut in Ihrem Gedächtnis haften. Daher werden wir die meisten Informationen, Techniken und Wissensbestandteile im dritten und vierten Teil kennenlernen.

In den letzten Abschnitten finden Sie ein Glossar, das Ihnen verschiedene Fachbegriffe nochmals einfach erklärt. Im Index sind die wichtigsten Begriffe zusammengefasst, damit Sie sie schnell nachschlagen können.

Was wird für dieses Buch benötigt?

Natürlich sollten Sie einen Computer nutzen können, der ein fehlerfrei installiertes Betriebssystem hat. Ob Sie *Microsoft Windows* , *Mac OS X* oder eine *Unix* -Variante einsetzen, ist für die Arbeit mit diesem Buch nicht von Belang. Aber denken Sie bitte daran: Sie installieren Software, die das System erweitert. Daher wäre es gut, dass keine wichtigen Daten auf dem System gespeichert sind, wenn im Fall der Fälle doch etwas gravierend schiefgehen sollte.

Zusätzlich benötigen Sie eine funktionierende Internetverbindung. Sie wird hauptsächlich für die Installation der einzelnen Softwareteile verwendet. Aber sie kommt auch im vierten Teil dieses Buches zum Einsatz, um die Applikation entwickeln und testen zu können.

Weitere Voraussetzungen

Sie sollten bereits ein gewisses Grundwissen in der Programmierung und im Umgang mit Daten haben. Jahrelange Programmiererfahrung ist zwar nicht notwendig, aber ein paar Grundkenntnisse sind von Vorteil, um den vollen Nutzen aus dem Buch ziehen zu können. Ich werde versuchen, den Rest für Sie so einfach wie möglich zu erklären und Sie bei den entscheidenden Schritten der Programmierung zu begleiten.

Für die Erstellung der praktischen Anwendung im hinteren Teil des Buchs ist es vorteilhaft, wenn Sie HTML und CSS beherrschen. Denn diese beiden Sprachen sind für die

Erstellung einer Web-Applikation zwingend erforderlich. Auch die JavaScript-Bibliothek *jQuery* sollte Ihnen nicht fremd sein. Denn wir werden sehen, dass *CouchDB* sehr eng damit zusammenarbeitet und unsere Applikation ebenfalls Nutzen daraus zieht.

Auch bei den Datenbanken müssen Sie kein SQL-Profi sein. Aber die gängigsten Fachausdrücke sollten Sie beherrschen, obwohl Sie sie auch im Glossar am Ende des Buchs nachschlagen können.

Und Sie sollten keine Scheu vor Neuem haben. Denn NoSQL ist möglicherweise eine der grundlegendsten Änderungen in der Entwicklung von Applikationen für und im World Wide Web.

Die Beispielcodes zum Buch

Besuchen Sie unsere Website unter *http://www.buch.cd*, und geben Sie dort die letzten Ziffern der ISBN dieses Buchs samt Bindestrich ein. Damit können Sie alle Beispielcodes und sonstigen Ressourcen zu diesem Buch herunterladen. Die verfügbaren Dateien werden nach der erfolgreichen Anmeldung angezeigt.

Sie finden dort die Installationsdateien für *CouchDB* und *CouchApp*. Ein Teil kann nur online installiert werden, daher habe ich diese Dateien nicht zur Verfügung gestellt. Sie finden dort auch alle verwendeten Bilder und Icons aus diesem Buch. Zusätzlich ist natürlich der gesamte Quelltext vorhanden. Ich habe ihn gegenüber der im Buch veröffentlichten Fassung ein wenig erweitert und manche Ideen, die im Buch nur kurz vorgestellt werden konnten, verwirklicht. Zusätzlich finden Sie auch alle Daten und

Dokumente im JSON-Format, damit Sie nicht alles von Hand eingeben müssen.

Berichtigungen

Obwohl alle Beteiligten mit größter Sorgfalt vorgehen, um die Richtigkeit der Inhalte sicherzustellen, passieren Fehler. Wenn Sie einen Fehler in diesem Buch entdecken, egal ob im Text oder im Quellcode, bin ich für eine kurze oder auch längere Mitteilung sehr dankbar. So können Sie anderen Lesern Ärger ersparen und mithelfen, die folgende Version des Buchs zu verbessern. Wenn Sie einen Druckfehler finden, senden Sie mir bitte eine E-Mail an *buch@guru-20.info*. Ich werde alle Berichtigungen, Änderungen und Verbesserungen auf meinem Blog *http://www.guru-20.info* veröffentlichen.

Danke!

Herrn Franz Graser, meinem Lektor und Betreuer beim Franzis Verlag: Danke für die kompetente Betreuung bei der Umsetzung der Bücher, die Korrekturen und die Möglichkeit, als Autor zu arbeiten.

Danke auch an das komplette Team im Franzis Verlag. Ohne die vielen im Hintergrund arbeitenden, helfenden Hände wäre dieses Buch nicht erschienen.

Feedback

Ich würde mich über Reaktionen und Anregungen sehr freuen. Sie erreichen mich unter folgender Adresse: *gull@guru-20.info*

Ihr

Clemens Gull

1 Die Theorie hinter NoSQL

In diesem Kapitel lernen Sie die Hintergründe von NoSQL kennen. Falls Sie direkt ins kalte Wasser springen wollen, können Sie dieses Kapitel auch überspringen. Falls Sie sich aber mit Datenbanken und den dahinter stehenden Konzepten noch nicht so gut auskennen sollten, empfehle ich Ihnen, diesen Abschnitt in Ruhe durchzuarbeiten. Denn hier werde ich einige Dinge erklären, die später verwendet werden. Damit Sie aber die einzelnen Fachbegriffe schnell und einfach finden, habe ich die entsprechenden Definitionen am Ende des Buchs in einem Glossar zusammengefasst.

Nach mehr als 15 Jahren einer fast uneingeschränkten Herrschaft der SQL-basierten Datenbanken (SQL steht für Structured Query Language, also zu Deutsch »strukturierte Abfragesprache«) findet nun ein Wechsel in der Landschaft der Web-Applikationen statt. Und genau diese neuen Sichtweisen und Techniken werden Sie hier kennenlernen. Natürlich werden nicht alle SQL-Datenbanken über Nacht verschwinden, aber bei einigen großen Unternehmen vollzieht sich der Wandel dahingehend, dass in den einzelnen Nischen die entsprechenden und idealen Datenbanksysteme (auch parallel zueinander) eingesetzt werden.

1.1 Die Geschichte

Der Begriff NoSQL (eine Abkürzung für *not only SQL*) wurde zum ersten Mal 1998 verwendet. Carlo Strozzi[1] entwickelte zu dieser Zeit eine leichtgewichtige Open-Source-Datenbank, die bewusst keine Zugriffsmöglichkeit unter dem Standard SQL zur Verfügung stellte. Strozzi unterscheidet aber zwischen seiner NoSQL-Datenbank und der NoSQL-Bewegung. Diese verfolgt ein Konzept, das vom relationalen Datenbankmodell abstammt.

Erst mehr als zehn Jahre später wurde der Begriff *NoSQL* bekannter. Denn Johan Oskarsson verwendete ihn Anfang 2009 bei einem Treffen zum Thema strukturierte, verteilte Datenspeicher . Er bezeichnete damit Datenbanken, die nicht relational sind, auf verteilten Systemen liegen und meistens auf ACID-Eigenschaften XE„ACID“(Englisch für *atomicity*, *consistency*, *isolation*, *durability*) verzichten.

Das Konzept NoSQL

Diese Technologie hat prinzipiell ein Ziel: die Nachteile bestehender, eingeführter Datenbanksysteme auszugleichen. Die Technik soll besser skalierbar sein und eine bessere Performance bei hoher Datenlast mit vielen umfangreichen Transaktionen bieten. NoSQL-Datenbanken haben im Gegensatz zu relationalen Datenbanksystemen kein festes Schema zur Speicherung der Daten. Dies ist eine radikale Änderung zu SQL-Datenbanken, die auf eine strenge Struktur (oder ein strenges Schema) der gespeicherten Daten achten.

NoSQL-Datenbanken können folgende Eigenschaften besitzen:

- nicht-relational
- schemafrei
- horizontal skalierbar
- BASE
- einfache Replikation

BASE – Basically Available, Soft State und Eventual Consistent

Da RDBMS (relationale Datenbankmanagementsysteme, also konventionelle Datenbanken) sehr streng auf die Konsistenz der Daten achten, kann es hier zu Problemen mit der Performance und Verfügbarkeit kommen. Dieses Konzept wird bei NoSQL zugunsten der besseren Skalierbarkeit und auch Verfügbarkeit aufgeweicht.

Man akzeptiert die »lose Konsistenz«. Dabei wird die Datenkonsistenz von nachfolgenden Datenoperationen immer wieder neu hergestellt. Die Datenbank wechselt dadurch immer wieder zwischen einem konsistenten und inkonsistenten Zustand. Im Gegensatz dazu müssen sich SQL-Datenbanken dauerhaft in einem konsistenten Zustand befinden. Werden durch verschiedene Datenbankoperationen Duplikate im Datenbestand angelegt, so wird die Datenbank durch eine zeitversetzte Synchronisierung immer wieder in einen konsistenten Teilzustand versetzt. Wobei der Begriff *Eventual Consistent* festlegt, dass alle Clients, die die Daten nutzen, nur in einem bestimmten Zeitfenster einen konsistenten (denselben) Datenbestand sehen.

CAP – Consistency, Availability und Partition Tolerance

Der Informatiker Eric Brewer vermutete im Jahr 2000, dass Systeme zum verteilten Rechnen nicht alle drei genannten Eigenschaften gleichzeitig erfüllen können. Diese Überlegung war und ist besonders wichtig für das Erstellen von Applikationen mit NoSQL-Datenbanken, denn sie sind verteilte Systeme.

- *Consistency – Konsistenz*
  Alle Clients sehen zur selben Zeit dieselben Daten
- *Availability – Verfügbarkeit*
  Ein Ausfall eines Clients (be)hindert die restlichen verfügbaren Clients nicht am Weiterarbeiten.
- *Partition Tolerance – Partitionstoleranz*
  Das verteilte System arbeitet trotz zufälliger Verluste von Nachrichten fehlerfrei weiter.

Im Jahr 2002 lieferten Seth Gilbert und Nancy Lynch einen axiomatischen Beweis, dass Brewers Vermutung richtig ist und nur zwei der drei Eigenschaften gleichzeitig erfüllt werden können. Dieses Theorem wirkt sich, wie wir später sehen werden, entscheidend auf das Design von NoSQL-Datenbanken aus.

1.2 Arten von NoSQL-Datenbanken

Datenbanken lassen sich in verschiedene Klassifikationssysteme einordnen. Am einfachsten gehen wir dabei vor, wenn wir die Art der Datenspeicherung betrachten. Für jede der folgenden Arten gibt es typische Vertreter und auch Anwendungsgebiete. Daher muss man sich vor der Wahl der Datenbank über das Anwendungsgebiet oder auch die zu verwaltenden Daten klar werden.

Dokumentenorientiert

Diese Datenbanken speichern Textdaten von beliebiger Größe in unstrukturierter Form. Der Zugriff auf die Daten erfolgt hier über die Dokumentinhalte. Hierzu gehören *Apache CouchDB* [2], *MongoDB* [3] oder auch *Lotus Notes* [4].

Key-Value-orientiert

Hier werden definierte Schlüssel verwendet, die auf einen bestimmten Wert verweisen. Diese Werte können aus beliebigen Zeichenfolgen bestehen. Die Key-Value-Datenbanken können entweder als *In-Memory-* oder als *On-Disk*-Version implementiert werden. Die Implementierung *In-Memory* eignet sich zum Beispiel sehr gut für Cache-Speichersysteme, da sie speicherresistent ist. Dies ist zum Beispiel das System *memcached* [5]. Die *On-Disk*-Version ist für große Datenmengen vorgesehen und wird beispielsweise von *Google BigTable* [6] oder *Amazon SimpleDB* [7] implementiert. Eine Kombination der beiden Versionen vereint – so gut es geht – die Vorteile beider Implementierungen miteinander. Eine Implementierung ist zum Beispiel *Redis* [8] und wird aktiv von *GitHub* [9] eingesetzt.

Spaltenorientiert

Hier werden die Daten als Schlüssel-Wert-Relation, auch Key/Value oder Tupel genannt, abgelegt. Das Hauptaugenmerk liegt hier auf der Verminderung der Ein-/Ausgabe-Aktivität bei der Berechnung der Datensätze. Ein Vertreter ist beispielsweise die Datenbank *Cassandra* [10] der *Apache Foundation* [11]. Diese Art von Datenbanken ist beispielsweise bei *Facebook* [12], *Digg* [13] oder *Twitter* [14] im Einsatz.

Graphenorientiert

Diese Datenbanken spiegeln die Beziehungen der Daten zueinander wieder. Dazu werden die Daten in einer Art Baumstruktur gespeichert. Sie eignen sich ideal zur Darstellung von Beziehungen in sozialen Netzwerken. Dabei werden die Daten als einzelne Knoten und die Beziehungen als Verbindungen zwischen diesen dargestellt. Beispiele für diese Art der Datenbank sind *FlockDB* [15] (wie sie *Twitter* verwendet), *AllegroGraph* [16] oder *Neo4j* [17].

Übersicht von NoSQL-Datenbanken

In diesem Abschnitt erhalten Sie einen – nicht vollständigen – Überblick über verschiedene NoSQL-Datenbanken, ihre Funktionen, Vor- und Nachteile sowie Einsatzgebiete.

	CouchDB	MongoDB	Redis	Cassandra	Riak
Programmiert in	Erlang	C++	C/C++	Java	Erlang & C
Lizenz	Apache[18]	AGPL[19]/Apache	BSD[20]	Apache	Apache
Hauptvorteil	DB-Konsistenz & einfache Benutzung	Verwendet Abfragen und Indizierung auf SQL-Basis	Geschwindigkeit	Große Tabellen	Fehlertoleranz
Protokoll	HTTP/REST	proprietär	Ähnlich Telnet	Proprietär und Thrift	HTTP/REST
Art	Dokumentorientiert	Dokumentorientiert	In-Memory	spaltenorientiert	spaltenorientiert
Besonderheiten	bidirektionale Replikation	Master-Slave-Replikation	Master-Slave-Replikation	Trade-offs für Verteilung und Replikation	Trade-offs für Verteilung und Replikation
	Konflikterkennung	Abfragen sind JavaScript-Ausdrücke	Einfache Schlüsselwerte	Trade-offs anpassbar	Trade-offs anpassbar
	MVCC blockieren keine Lesezugriffe		Komplexe Datenoperationen		Datenüberprüfung
	Versionierung der Daten	Serverseitiges JavaScript	Verwendet Sets, Lists und Hashes	Abfrage nach Spalten	Datensicherheit

	CouchDB	MongoDB	Redis	Cassandra	Riak
	serverseitige Validierung der Dokumente		Unterstützt Transaktionen		
	Authentifizierung	Verteilte Daten (Sharding)	Ablaufdatum für Daten	Abfrage nach Indexbereichen	Volltextsuche
	Echtzeit-Updates		Sortierte Sets		
	Attachment-Verwaltung	Teilweise Ablage der Daten im Speicher	Überwachen von Datenänderungen	Schreibt schneller als es liest	Commits
	jQuery-Bibliothek				
Nachteile	Wiederholte Komprimierung notwendig	Nach einem Crash müssen die Tabellen repariert werden	Erst Version 2.0 kann auf die Disk auslagern	Schreibt schneller als es liest	Enterprise- und Open-Source-Version
			Datenbankgröße sollte vorhersehbar sein	Übernimmt Komplexität von Java	Verteilte Daten nur mit der Enterprise-Version
Einsatzgebiete	CRM	Dynamische Abfragen	Häufige Schreibzugriffe	Echtzeit-Datenanalyse	Hochverfügbarkeit
	Wenig Schreibzugriffe	Indizierung der Daten	Schnelle Änderung der Daten	Alle Systembestandteile in Java sein sollen	Schnelle Schreibzugriffe
	Häufige Lesezugriffe	Große DB mit Performance	Echtzeitverarbeitung		Seltene Lesezugriffe
	Versionierung	Ähnlich SQL ohne definierte Spalten	Statistiken	Logging	Datenanalyse

In diesem Buch wird die Entscheidung nicht direkt vom Einsatzgebiet bestimmt. Aus didaktischen Gründen, und wegen dem leichteren Handling auf einem lokalen Computersystem entscheiden wir uns für den Einsatz der *CouchDB* als Vertreter der dokumentenorientierten NoSQL-Datenbanken.

1.3 Grundlagen von CouchDB

Be Relaxed! oder *Entspann Dich!* ist der Leitspruch von *Apache CouchDB,* und genau dieses Motto müssen wir verinnerlichen. Eine der Grundlagen ist die Einfachheit, mit der *CouchDB* erstellt wurde. Die Datenbank versucht sich so weit wie möglich im Hintergrund zu halten und den Administrator nicht zu »belästigen«.

Dazu gehört, dass die interne Architektur sehr fehlertolerant ist. Einzelne Fehler beziehungsweise Ausnahmen treten in einer kontrollierten Umgebung auf und werden dort auch behandelt. Wenn üblicherweise eine Ausnahme auftritt, läuft sie durch das ganze System, wo sie an der Spitze behandelt wird. Bei *CouchDB* betrifft ein Fehler nur eine einzelne Anfrage (Request), wird dort behandelt und beeinflusst den Rest des Systems nicht.

Außerdem ist *CouchDB* auf Lastwechsel ausgerichtet. Wir kennen das Problem, dass es auf einmal viele Requests bei einer Web-Applikation geben kann. *CouchDB* reagiert zwar mit einer höheren Latenzzeit darauf, aber sie vergisst keine Anfrage, sondern beantwortet alle. Ist das hohe Lastaufkommen wieder vorbei, reagiert die Datenbank wieder mit der gewohnten Geschwindigkeit.

Normalerweise wird eine Anwendung so erstellt, dass sie bereits die maximalen Hardwareanforderungen abdeckt. Einer der offensichtlichen Nachteile dieses Ansatzes sind

die hohen Investitionskosten, die am Beginn des Projekts stehen. Weiters ist das Lastaufkommen oft viel zu gering, um die Hardware auszulasten und ihre Vorteile wirklich zu nutzen.

Auch ist die Programmierung viel aufwendiger, da bereits Fälle abgebildet werden müssen, die erst in der Zukunft auftreten werden. *CouchDB* verfolgt hier einen anderen Ansatz: Es besitzt eine eingebaute Skalierbarkeit. Damit diese auch funktioniert, setzt die Datenbank dem Programmierer eindeutige Grenzen und macht nicht alles möglich, was umsetzbar wäre. Dies ist zwar etwas unflexibel, und wir müssen lieb gewonnene Gewohnheiten über Bord werfen. Aber durch das Fehlen mancher Funktionen kann der Programmierer – also Sie und ich – keine Anwendung schreiben, die der Skalierung im Wege steht oder diese unmöglich macht.

Daten müssen modelliert werden!

Dieser Ansatz besteht seit dem Beginn der IT und ist sicherlich richtig. Aber die Art und Weise kann sich je nach Gedankenmodell und Herangehensweise unterscheiden. In der klassischen Anwendung werden Daten im ER-Modell , kurz ERM (Entity-Relationship-Modell), dargestellt. Dies ist zwar eine sehr effektive Art, Daten für Computer aufzubereiten, aber auch eine sehr schwer skalierbare und für Menschen nicht leicht zu verstehende Art.

Im klassischen Modell – nehmen wir als Beispiel eine Rechnung – stellt sich die Datenmodellierung als Referenzmodell schon sehr aufwendig dar. Und dabei sind noch gar nicht alle Möglichkeiten berücksichtigt. Werfen Sie einen Blick auf die folgende Abbildung. Sie ist schon komplex, obwohl es nur ein einfaches, schematisches ERM ist und viele Punkte nicht berücksichtigt sind.

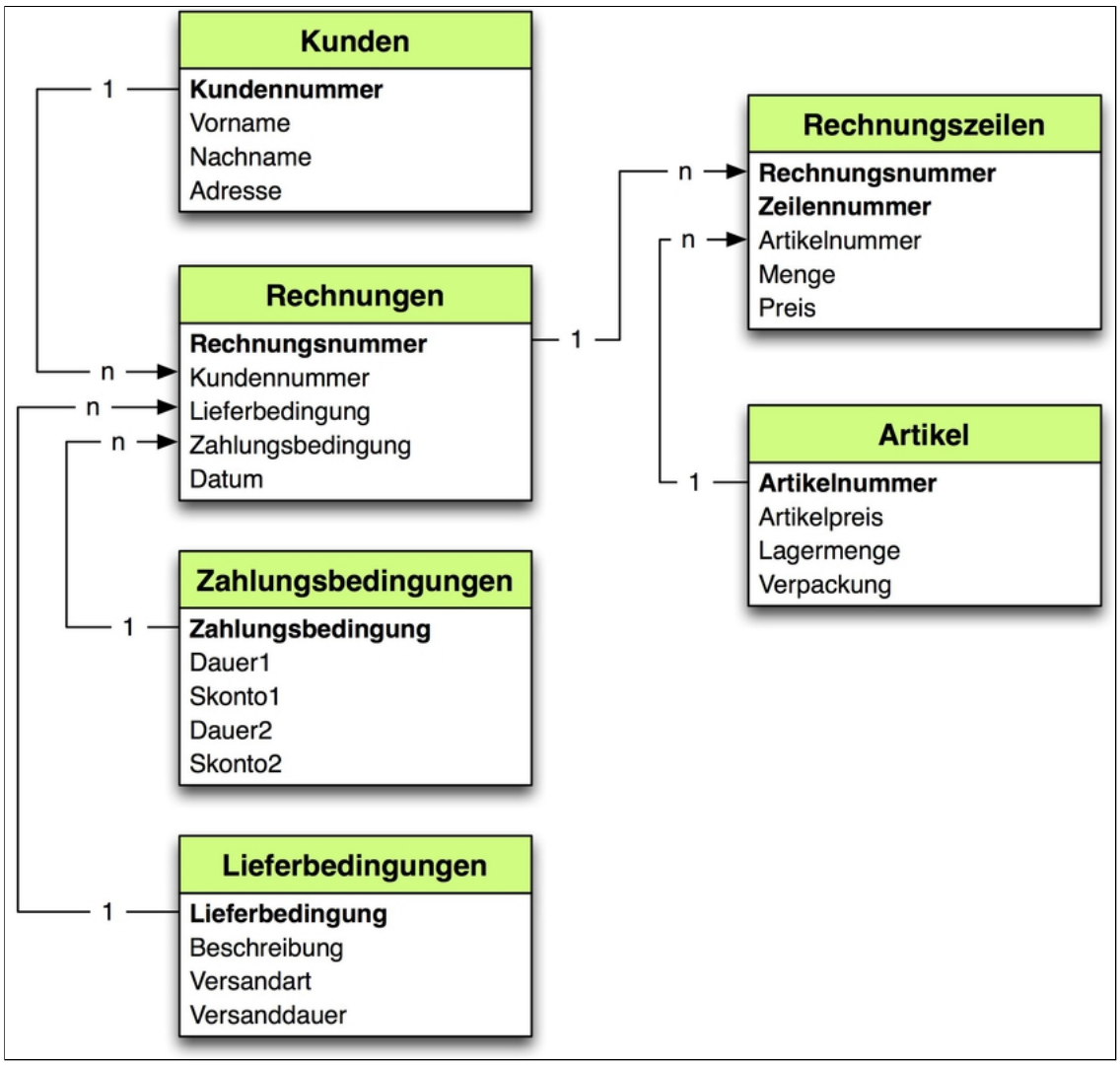

Bild 1.1 Daten in einem Referenzmodell

Im ER-Modell versuchen wir die Daten atomar[21] , eindeutig und nach Typ definiert[22], in einer Datenbank abzulegen. Um dies zu erreichen, müssen wir alle Redundanzen vermeiden und die einzelnen Datensätze in eigene Tabellen ablegen. In unserem Beispiel sind das bereits sechs. Aber dies widerspricht unserem natürlichen Denken. Denn wir hätten gern alle Daten auf einen Blick und nicht in kleine Häppchen aufgeteilt, die wir später wieder zusammensuchen müssen. Außerdem hat dieses Modell noch einen weiteren Haken: Durch das schemaorientierte Modellieren müssen alle möglichen Datenfelder bereits vorhergesehen werden, späteres Hinzufügen (Skalieren) ist oft mit einem beträchtlichen Aufwand verbunden. Dieser Aufwand kann so groß sein, dass eine Anwendung komplett neu entwickelt werden muss, damit sie auf größere Systeme skaliert werden kann. Und was ist, wenn Daten nicht erfasst werden, weil sie entweder nicht vorhanden oder nicht bekannt sind? Auch dafür müssen eigene Mechanismen im Schema vorgesehen werden. Denn leere Datenfelder können eine klassische relationale Datenbank aus dem Tritt bringen.

Dies sind aber keine Gründe, relationale Datenbanken zu verteufeln und nicht mehr einzusetzen. Sie werden sehr häufig verwendet und sind auch für viele Arten der Datenverarbeitung sehr geeignet. Aber manchmal können SQL-Datenbanken die Dokumente der realen Welt nur sehr schwer abbilden. Und hier kommen dokumentenorientierte Datenbanken wie *CouchDB* ins Spiel. Diese sehen die Daten als eine Einheit und speichern und präsentieren sie auch so. Betrachten wir zum Beispiel Visitenkarten – sie kommen oft vor und wir haben alle welche auf dem Schreibtisch liegen:

Bild 1.2 Verschiedene Visitenkarten

Jede Visitenkarte enthält und präsentiert die für uns entscheidenden Informationen auf einen Blick. Es sind also in sich geschlossene Daten, eine der Grundlagen einer dokumentorientierten Datenbank.

Wenn wir die obigen Karten betrachten, sehen wir, dass sie alle ähnliche Daten enthalten, aber eben nicht exakt die gleichen. Denn bei Elisa Schmidt fehlt die Webseitenadresse, die Max Muster angeführt hat. Und Herr Meier hat nur eine Telefonnummer und Postanschrift, keine Mailadresse. Die fehlenden Daten werden von uns Menschen einfach ignoriert, denn wir haben das Konzept verinnerlicht. Durch einfaches Weglassen der Mailadresse gibt Herr Meier zu verstehen, dass er keine hat. Er muss nicht *E-Mail: keine vorhanden* auf die Karte schreiben.

Und hier liegt auch der grundlegende Unterschied zwischen SQL- und NoSQL-Datenbanken. Für die SQL-Variante müssen die Daten *vorher* strukturiert in ein Schema gebracht werden, sie sind daher schemaorientiert. Erst dann sind sie zu verarbeiten. Die NoSQL-Variante

hingegen strukturiert die Daten erst *nach dem Speichern*, genauso wie wir Menschen. Diese Datenbanken sind schemafrei.

Grundlegende Irrtümer!

Programmierer gehen oft von falschen Annahmen aus. Dies wurde schon vor fast 20 Jahren erkannt, wird aber heute noch oft und gern ignoriert. Die meisten Programmierer verteilter Systeme gehen von Annahmen aus, die der Hausverstand verneint. Sie wurden bekannt unter der Bezeichnung »*Fallacies of Distributed Computing*«[23].

1. Das Netzwerk ist ausfallsicher.
2. Die Latenzzeit ist gleich null.
3. Der Datendurchsatz ist unendlich.
4. Das Netzwerk ist sicher.
5. Die Netzwerktopologie wird sich nicht ändern.
6. Es gibt immer nur einen Netzwerkadministrator.
7. Die Kosten des Datentransports können mit Null angesetzt werden.
8. Das Netzwerk ist homogen.

Die *CouchDB* versucht nicht, wie andere Werkzeuge diese Punkte zu ignorieren oder zu verstecken. Im Gegensatz dazu sind im grundlegenden Konzept diese acht Punkte enthalten, und das System versucht danach zu arbeiten. Am besten sieht man das bei der Replikation der Datenbanken durch *CouchDB*.

Ein wichtiger Punkt der Skalierbarkeit ist, dass Daten an verschiedenen Orten präsent sind. Dies können verschiedene Server im selben Raum oder auch an geografisch weit auseinander liegenden Orten sein. Damit die Daten von einem Server zum anderen kommen, müssen sie repliziert (kopiert) werden. Bei *CouchDB* passiert dies mit dem im Internet üblichen Protokoll *HTTP* und als inkrementelle Replikation. Das Protokoll ist schließlich vorhanden und funktioniert (meistens). Aber wie der erste Punkt der Irrtümer zeigt, das Netz wird ausfallen! Und das wird nach Murphys Gesetz [24] genau während der Replikation passieren. Dann ist die Datenbank bei *CouchDB* trotzdem verfügbar. Irgendwann ist eine Netzwerkverbindung wieder verfügbar, und dann beginnt das System nicht von vorne mit der gesamten Replikation, sondern setzt genau dort fort, wo die Unterbrechung stattfand.

Die Wartezeit entscheidet!

Dies ist bei allen Anwendungen so. Sobald der Anwender warten muss, nimmt die Unzufriedenheit zu – beobachten Sie sich selbst bei der Arbeit mit lokal installierten Applikationen oder auch beim Surfen im World Wide Web. Nicht umsonst versuchen alle Webseiten, noch das letzte Quäntchen an Geschwindigkeit herauszuholen. Aber das Web hat eben keinen unendlichen Datendurchsatz und auch keine hundertprozentige Verfügbarkeit. Denken wir nur an unsere geliebten Smartphones, Tablets und anderen mobilen Geräte. Wie oft ist die Verbindung zum Anbieter nicht verfügbar? Und was ist, wenn wir im Ausland sind, wollen wir die hohen Kosten für Datenverbindungen wirklich zahlen?

Und hier ist ein anderer Ansatz der *CouchDB*: Wieso versucht man nicht, eine Skalierung nach unten vorzunehmen und eine lokale Kopie der Datenbank zu erzeugen und zu verwenden? Da die Datenbank in der Sprache *Erlang* [25] geschrieben ist, kann sie auch auf sehr einfachen Geräten mit geringer Hardwareausstattung laufen. Und warum nicht eine Datenbank auf einem Smartphone installieren, die sich – sobald eine Datenleitung verfügbar ist – automatisch mit anderen Instanzen synchronisiert? Dieser Ansatz bietet den

Vorteil einer geringen Latenzzeit, denn eine lokal vorhandene Datenbank antwortet in Tausendstelsekunden. Zusätzlich wird das Problem des fehlenden Mobilfunknetzes umgangen. Denn die Daten sind jetzt offline verfügbar.

Mit oder gegen die CouchDB arbeiten?

In der Welt der Programmierung hat sich eine Phrase eingebürgert: »Wir programmieren *gegen* eine Schnittstelle/Datenbank/API.« Und dieser Satz drückt auch die grundlegende Denkweise der Programmierung aus. Wir arbeiten gegen etwas und versuchen es in unsere Anforderung zu zwingen. Die Philosophie der *CouchDB* ist aber, *mit* der Datenbank zu arbeiten und nicht Dinge zu erzwingen. So erhalten wir automatisch einfache, skalierbare und verteilte Systeme.

Bei der Arbeit mit der Datenbank müssen Sie aber einen der grundlegenden Ansätze von SQL über Bord werfen: Daten sind immer konsistent! Bei der *CouchDB* lautet der Ansatz: Die Daten sind am Ende irgendwann konsistent![26] Dies ist ein entscheidender Ansatz beim Hinzufügen von mehreren Datenbankservern zu einem bestehenden System. Denn hier müssen Entscheidungen über die Konsistenz der Daten und auch der Verfügbarkeit getroffen werden. Und hier kommt das erwähnte CAP-Theorem ins Spiel. Dabei legt die *CouchDB* das Hauptaugenmerk auf *Verfügbarkeit* und *Partitionstoleranz* und stellt die *Konsistenz* hintan.

[1]
http://www.strozzi.it

[2]
http://couchdb.apache.org

[3]
http://www.mongodb.org

[4]
http://www-01.ibm.com/software/de/lotus

[5]
http://memcached.org

[6]
http://labs.google.com/papers/bigtable-osdi06.pdf

[7]
http://aws.amazon.com/de/simpledb

[8]
http://redis.io

[9]
https://github.com

[10]
http://cassandra.apache.org

[11]
http://apache.org

[12]
http://www.facebook.com

[13]
http://digg.com

[14]
http://twitter.com

[15]
https://github.com/twitter/flockdb

[16]
http://www.franz.com/agraph/allegrograph

[17]
http://neo4j.org

[18]
http://www.apache.org/licenses

[19]
http://www.gnu.org/licenses/agpl-3.0.html

[20]
http://www.opensource.org/licenses/bsd-license.php

[21]
Nur ein Datum pro Datenfeld. Eine Adresse – z. B. Doktorweg 13 – besteht aus zwei Daten, dem Straßennamen und der Hausnummer. In einem ERM müssten zwei Datenfelder geschaffen werden.

[22]
Es wird also von Anfang an festgelegt, welcher Art die Daten sein müssen. Dies kann ein Datum, ein Text, eine Zahl oder noch etwas anderes sein. Aber hier ergeben sich bereits Probleme, denn welchen Datentyp hat beispielsweise eine Postleitzahl? In Österreich, Deutschland und der Schweiz ist es eine Zahl, aber in Großbritannien ist es eine Zeichenkette aus Buchstaben und Zahlen. Zusätzlich gibt es in Italien und Deutschland Postleitzahlen, die mit einer führenden Null geschrieben werden. Speichern wir diese jetzt als Text oder als Zahl?

[23]
Erstmals wurden vier Trugschlüsse von William Nelson Joy und Tom Lyon veröffentlicht. 1994 wurden sie um drei weitere Punkte von Laurence Peter Deutsch erweitert, und James Gosling ergänzte etwa 1997 den achten Punkt.

[24]
»Whatever can go wrong, will go wrong.« (»Alles, was schiefgehen kann, wird auch schiefgehen.«)

[25]
Erlang wurde ursprünglich im Jahr 1987 für die Programmierung in der Telekommunikation (Vermittlungsstellen von Telefonnetzen) geschaffen. Für weitere Informationen siehe *http://erlang.org*

[26]
Eventual Consistency

2 Installation

In diesem Kapitel bereiten Sie Ihren Computer für den Einsatz einer NoSQL-Datenbank vor, in unserem Fall *Apache CouchDB* . Wählen Sie bitte den für Ihr Betriebssystem notwendigen Abschnitt aus, und arbeiten Sie ihn durch. Als Voraussetzung sind für beide beschriebenen Versionen (Mac OS X und Windows) ein fehlerfrei laufendes Betriebssystem und eine funktionierende Internetverbindung notwendig.

Falls Sie als Betriebssystem eine der vielen Unix-Varianten (wie z. B. Linux) verwenden, können Sie den Abschnitt für Mac OS X durcharbeiten. Da wir beim Mac mit dem Terminalfenster auf Shell-Ebene arbeiten, ist der Unterschied nicht groß.

2.1 Mac OS X vorbereiten

Die Entwicklungsumgebung Xcode installieren

Der erste Schritt für die Installation von *CouchDB* ist das Installieren der Entwicklungsumgebung *Xcode.* Dazu benötigen Sie eventuell die originale Installations-DVD des Systems. Sobald Sie sie eingelegt haben, wählen Sie den Punkt Optionale Installationspakete mit einem Doppelklick aus. Danach aktivieren Sie das Paket *Xcode.mpkg* mit einem Doppelklick. Nun folgen Sie den Installationsanweisungen und installieren *Xcode*.

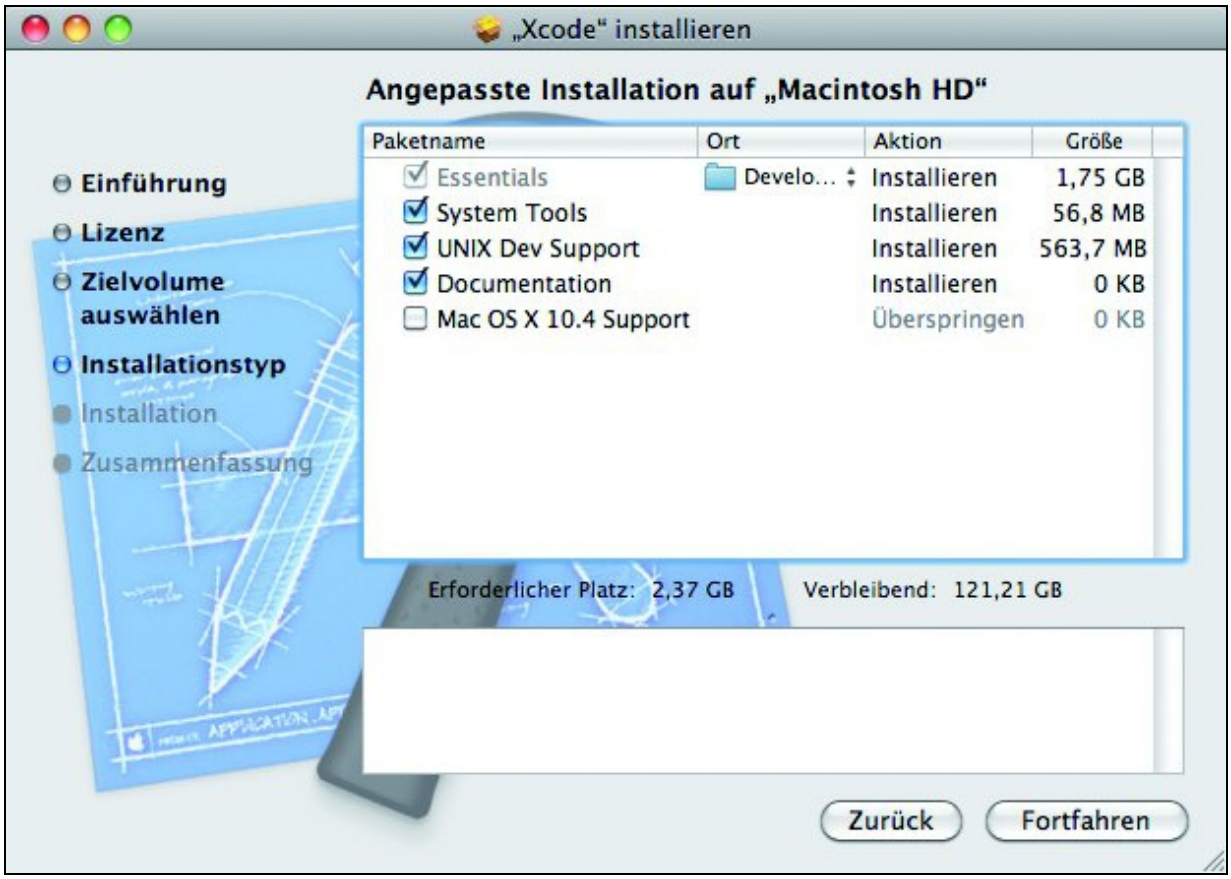

Bild 2.1 Xcode installieren

Alternativ können Sie *Xcode* auch aus dem *AppStore* [1] herunterladen. Falls Sie registrierter Entwickler sind, steht Ihnen der Download kostenlos im Developer-Center[2] zur Verfügung.

MacPorts installieren

Mit dieser freien Software ist es sehr einfach, Open-Source-Software auf ein Mac-OS-X-System zu installieren. Da Sie einige Abhängigkeiten für die *CouchDB* installieren müssen, verwenden wir diese Software zur Vereinfachung der kommenden Aufgaben. Zuerst können Sie sich das Paket *MacPorts* [3] herunterladen und danach mit einem Doppelklick die Installation beginnen. Gehen Sie

schrittweise durch den Assistenten, um die Software in Ihr System zu integrieren. Es sind keine besonderen Einstellungen vorzunehmen, und Sie können beruhigt die Standardvorgaben übernehmen.

Das Terminalfenster

Da die weitere Installation auf Mac OS X im *Terminal* mit dem *root*- Benutzer erfolgen muss, benötigen Sie das Passwort für diesen Benutzer. Haben Sie dieses noch nicht freigeschaltet, folgen Sie bitte der Anleitung in der *Apple Knowledge Base* [4] . Achten Sie bitte darauf, den für Ihre Version (10.5 beziehungsweise 10.6) entsprechenden Abschnitt durchzuarbeiten.

Als Erstes öffnen Sie ein Terminalfenster. Sie finden es in Programme, Dienstprogramme.

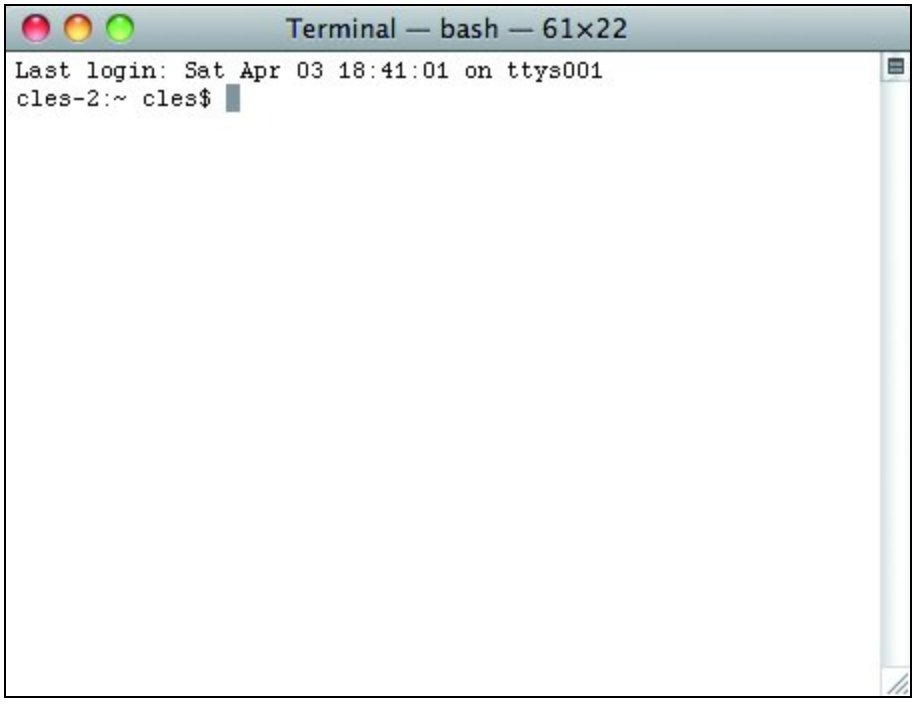

Bild 2.2 Das Terminalfenster

Nun geben Sie den Befehl `su` ein und bestätigen ihn mit [Return]. Es folgt die Eingabe des Passworts, das Sie ebenfalls mit [Return] bestätigen.

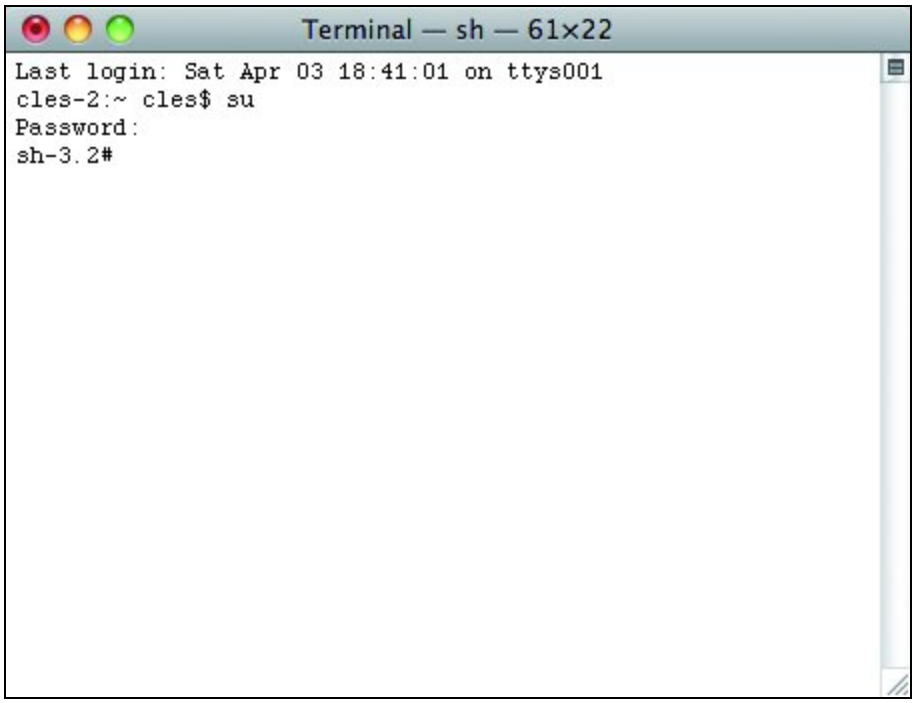

Bild 2.3 Terminal mit angemeldetem *root*-Benutzer

MacPorts überprüfen

Falls Sie *MacPorts* bereits installiert haben, können Sie die aktuelle Installation überprüfen und auch auf die neueste Version aktualisieren. Geben Sie dazu den Befehl `port selfupdate` ein, gefolgt von [Return].

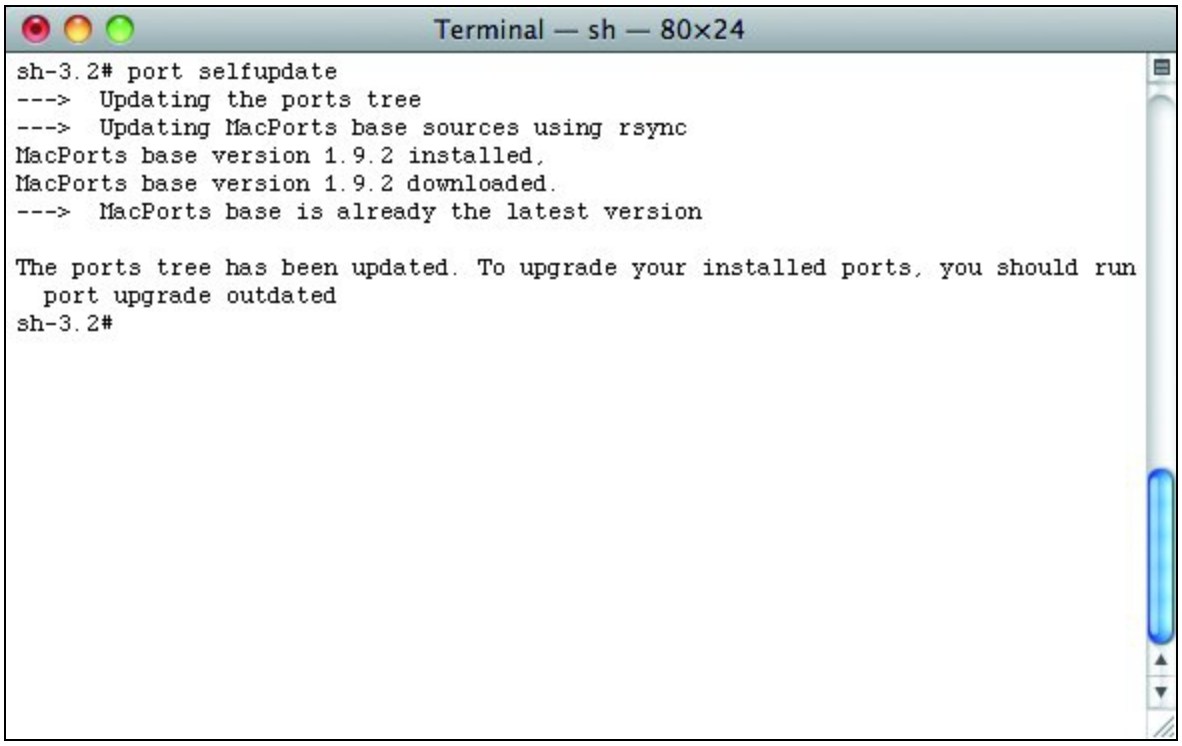

Bild 2.4 Installation von MacPorts überprüfen

CouchDB installieren

Der nächste Schritt ist die Installation der NoSQL-Datenbank *Apache CouchDB* . Wir benutzen dafür *MacPorts* mit dem Befehl `port install couchdb` und einem [Return].

Je nachdem, wie Ihr System installiert ist und/oder wie alt es bereits ist, werden die notwendigen Abhängigkeiten – Dateien, die für das fehlerfreie Funktionieren notwendig sind – vom World Wide Web geladen und installiert. Sie dürfen nicht überrascht sein, denn je nachdem, wie schnell Ihre Internetverbindung, wie schnell Ihr Computer und wie groß die Anzahl der abhängigen Dateien sind, kann dieser Vorgang mehr als zwei Stunden in Anspruch nehmen.

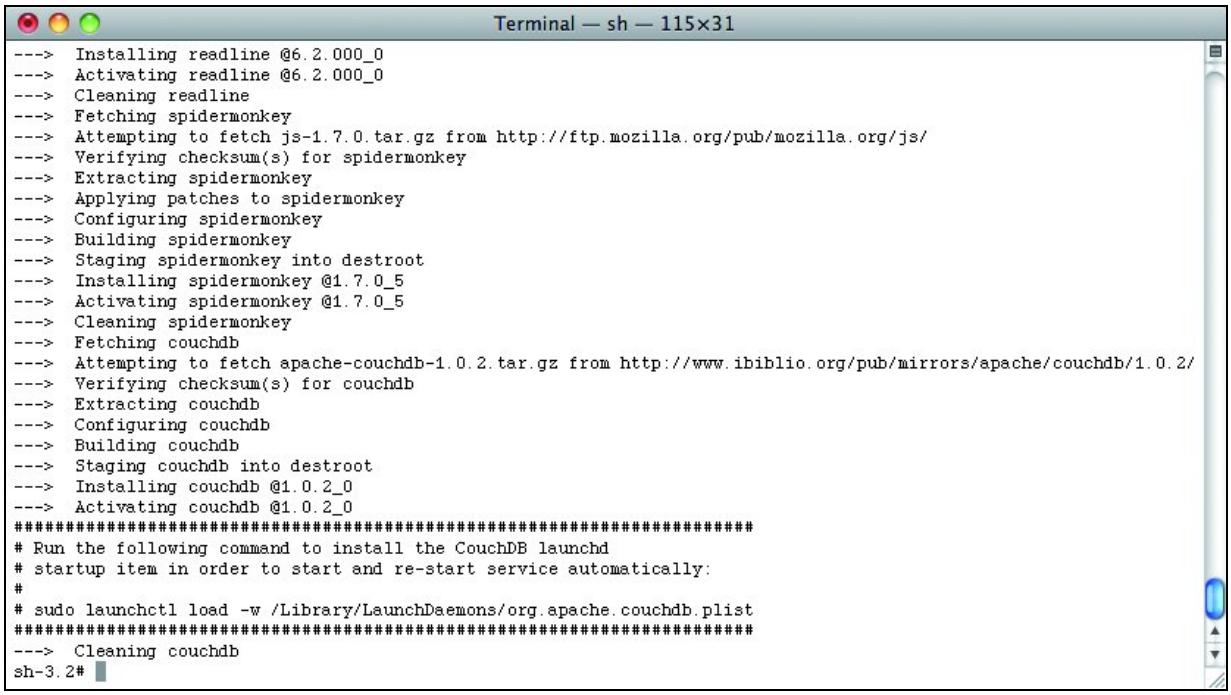

Bild 2.5 Ende der Installation von CouchDB

Nun sind zwar alle Abhängigkeiten installiert, aber leider aktualisiert *MacPorts* vorhandene Dateien nicht. Dies müssen Sie manuell mit dem Befehl `port upgrade couchdb` durchführen, gefolgt von [Return].

Jetzt können Sie *CouchDB* mit dem Befehl `couchdb –b` und [Return] starten. Prüfen Sie mit folgendem Befehl, ob eine Instanz der *CouchDB* läuft. Normalerweise gibt der Befehl `curl http://127.0.0.1:5984` und [Return] eine Willkommensnachricht aus.

Falls Sie eine Fehlermeldung erhalten, liegt das normalerweise an einer falschen Einstellung von *MacPorts*. Durch diesen Fehler wird *Erlang* – eine abhängige Programmiersprache – ohne SSL erstellt. Das lässt sich leicht beheben, indem Sie *Erlang* mit dem Befehl `port install erlang +ssl`, gefolgt von [Return], nochmals neu erstellen und

installieren. Nach der erfolgreichen Installation, sollten Sie Folgendes im Terminalfenster sehen:

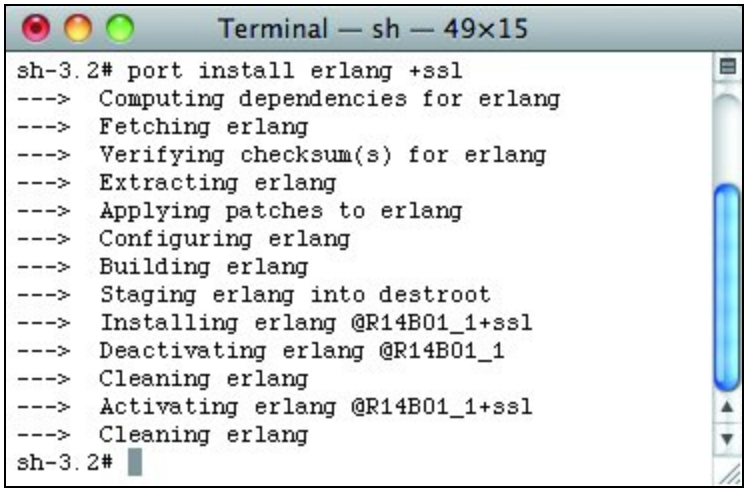

Bild 2.6 Erlang wurde mit SSL installiert

Nun können Sie wie oben beschrieben den Datenbankserver starten und mit dem `curl` -Befehl nochmals die Funktion der *CouchDB* überprüfen. Sie sollten ein ähnliches Bild wie in der folgenden Abbildung dargestellt sehen.

Bild 2.7 Überprüfen von CouchDB mit dem curl-Befehl

CouchDB automatisch starten

Falls der *CouchDB*-Server noch läuft, halten Sie ihn mit `couchdb –d` und [Return] an.

Nun geben Sie `launchctl load -w /opt/local/Library/LaunchDaemons/org.apache.couchdb.plist` ein und bestätigen den Befehl mit [Return]. Ab jetzt wird der *CouchDB*-Server automatisch beim Start des Betriebssystems mit gestartet. Dies ist gerade während der Arbeit mit diesem Buch praktisch. Wollen Sie den automatischen Start wieder entfernen, dann verwenden Sie den folgenden Befehl: `launchctl unload -w /opt/local/Library/LaunchDaemons/org.apache.couchdb.plist`

Jetzt müssen Sie den Server nur einmal starten, indem Sie Ihren Computer neu starten oder den Befehl `launchctl start org.apache.couchdb` eingeben, gefolgt von [Return].

Als Letztes prüfen wir *CouchDB* im Browser. Dazu geben Sie in der Adresszeile *http://127.0.0.1:5984/_utils* ein. In der Folge sehen Sie das Verwaltungsinterface *Futon* im Browserfenster.

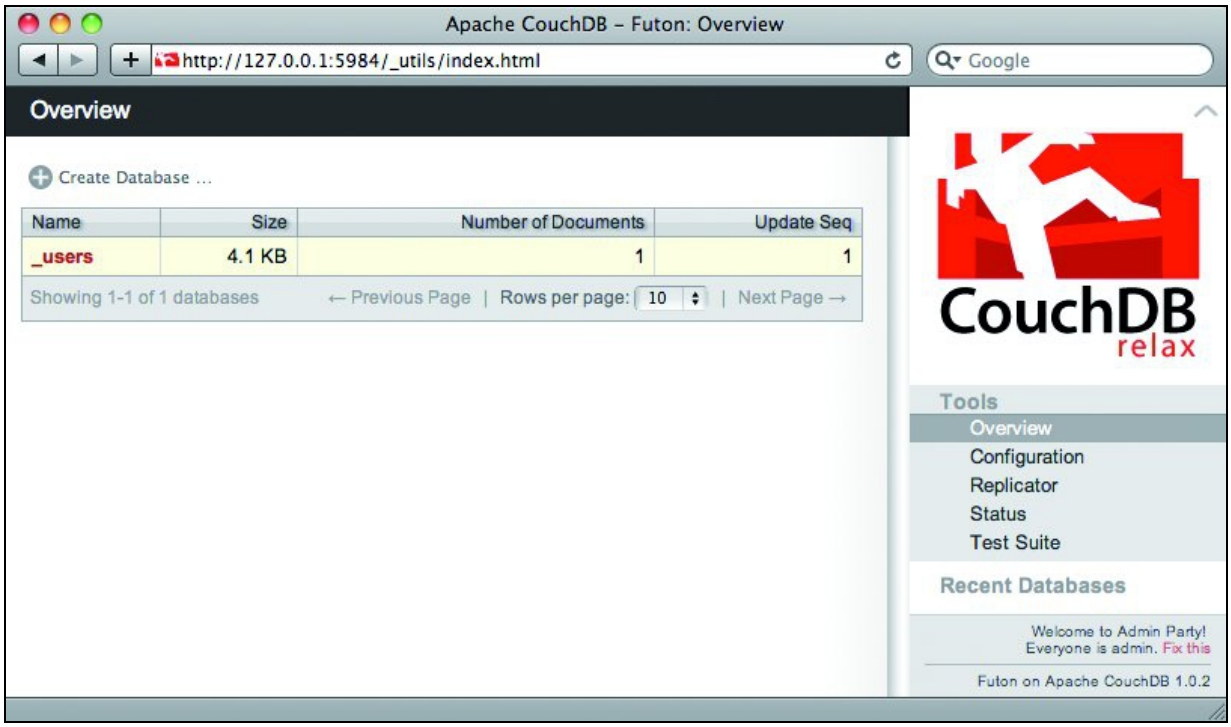

Bild 2.8 Das Verwaltungsinterface von CouchDB im Browser

CouchApp installieren

Nun installieren wir den zweiten notwendigen Teil der *CouchDB*, die *CouchApp*. Dazu prüfen Sie, ob die aktuelle Version von *distributed install* verfügbar ist. Im Terminalfenster geben Sie

```
curl -O http://python-distribute.org/
```

`distribute_setup.py` und [Return] ein.

Sie sollten jetzt die Ausgabe wie in der folgenden Abbildung dargestellt im Terminalfenster sehen.

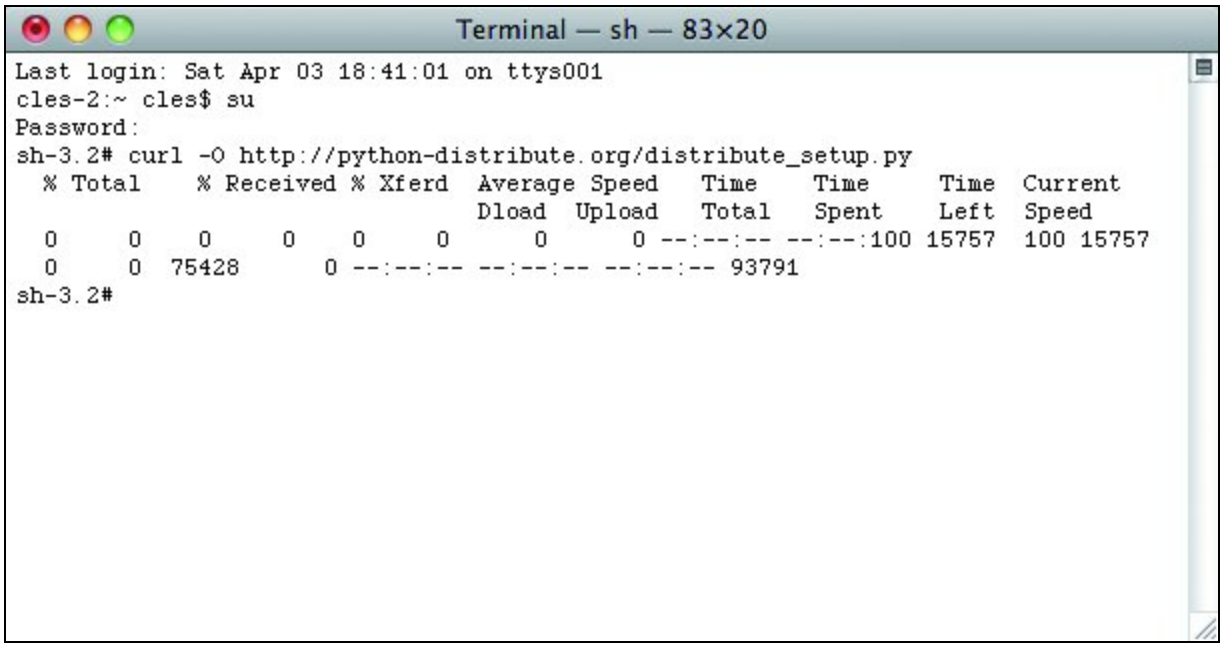

Bild 2.9 Ausgabe des Befehls cURL

Nun können Sie *distributed install* auf das System installieren. Dazu ist folgender Befehl notwendig: `python distribute_setup.py` [Return]. Auch hier erhalten Sie eine komplette Übersicht des Installationsprozesses als Ausgabe im Terminalfenster.

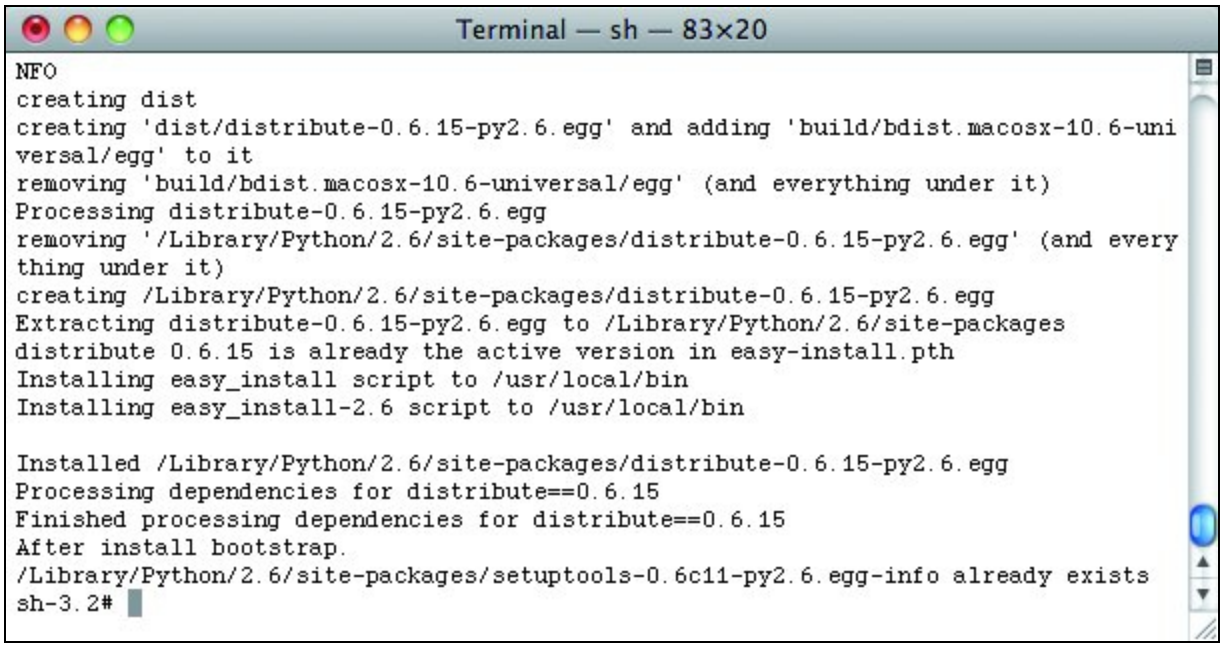

Bild 2.10 Ausgabe der Installation von *distributed install*

Als Nächstes müssen Sie den *Easy-Installer* auf Ihr System installieren. Dazu geben Sie `easy_install pip` und [Return] ein. Auch dieser Befehl zeigt durch die Ausgabe der Log-Einträge den vollständigen Installationsvorgang im Terminalfenster an.

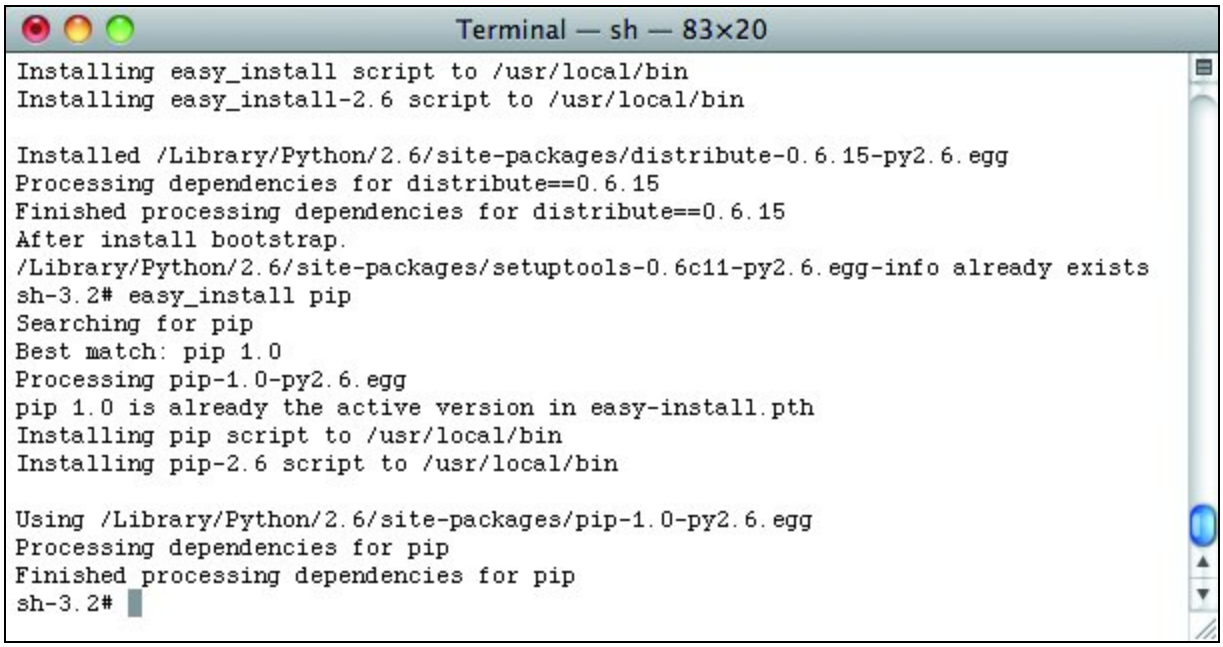

Bild 2.11 Ausgabe der Installation des *Easy-Installers*

Nun ist das System für die Installation der *CouchApp* vorbereitet. Sie wird mit `pip install couchapp` und [Return] gestartet. Die Liste der Log-Einträge, die im Terminalfenster ausgegeben werden, ist hier sehr lang. Aber für uns sind nur die letzten beiden Zeilen interessant. Diese sehen Sie in der folgenden Abbildung und hoffentlich auch in Ihrem Terminalfenster.

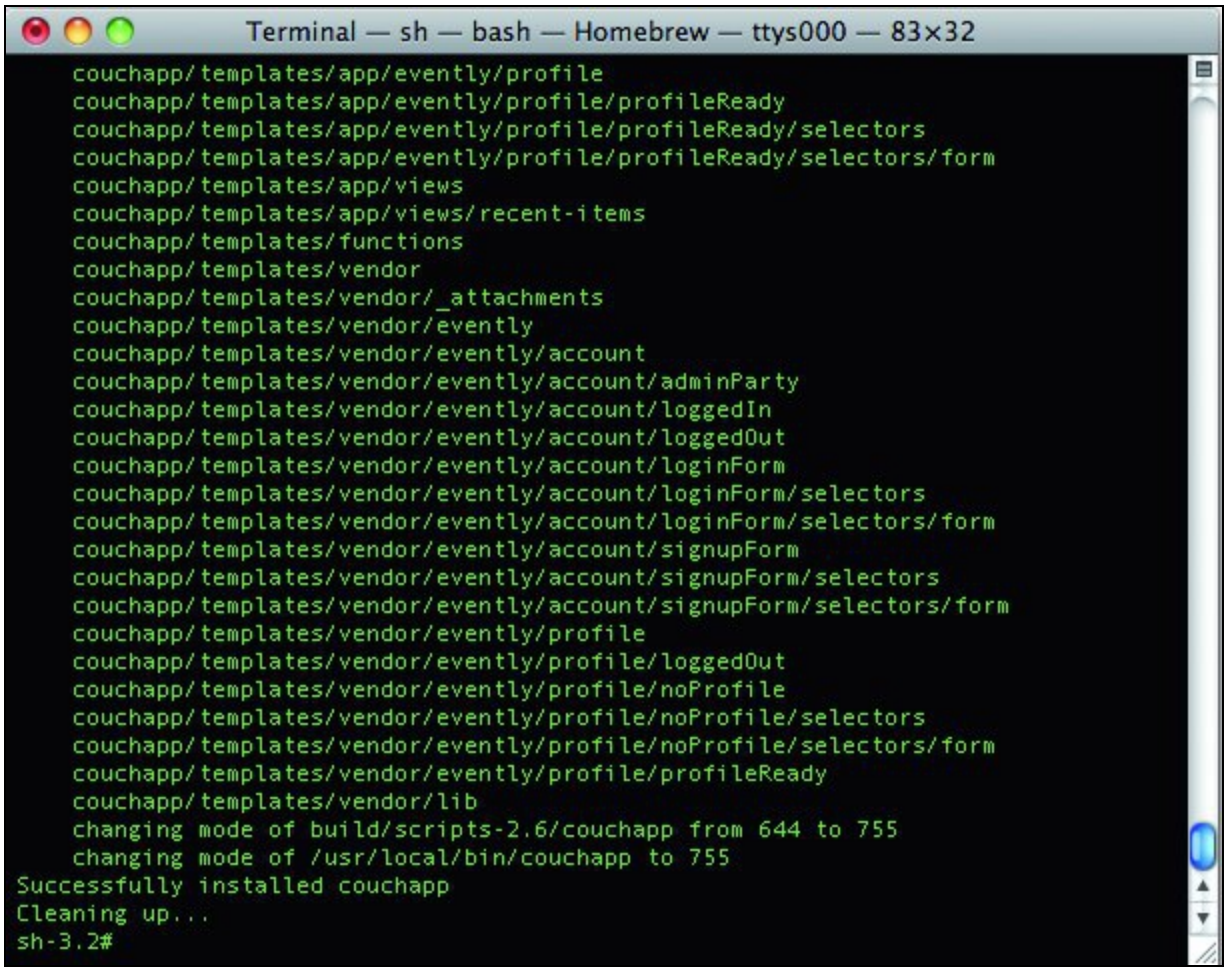

Bild 2.12 Ausgabe der erfolgreichen Installation der *CouchApp*

Überprüfen der Installation von CouchApp

Um zu überprüfen, ob der Vorgang abgeschlossen wurde, rufen Sie die Versionsnummer der installierten *CouchApp* ab. Dazu geben Sie `couchapp version` und [Return] ein. Als Antwort erhalten Sie folgende Ausgabe im Terminalfenster: