Ein PHP-Wrapper für die CouchDB

Entspannt auf der Couch lümmeln

Dokumentbasierte Datenbanken sind nicht nur ein Hype, sondern eine sehr ernst zu nehmende Alternative für Webapplikationen. CouchDB ist mit Sicherheit eine der am weitesten verbreitete und am besten skalierenden Varianten unter den Projekten. Dass es kein Hexenwerk ist, mit PHP auf eine „Couch" zuzugreifen, werden Sie in diesem Artikel sehen und beim Bauen eines einfachen Wrappers selbst feststellen. Getreu dem Motto: Relax!

von Andreas Wenk

Moment mal! Hat Andy nicht gerade noch in den letzten Ausgaben über PostgreSQL geschrieben? Jetzt CouchDB? Sind das nicht zwei grundverschiedene Dinge? Im Prinzip ja, aber auch nein. Nein, weil CouchDB genauso wie PostgreSQL eine Datenbank ist, um – richtig! – Daten zu speichern. Ja, weil das dahinterstehende Prinzip ein anderes ist. CouchDB [1] ist eine dokumentbasierte Datenbank im Gegensatz zu PostgreSQL (und natürlich anderen „Banken" wie Oracle, DB2, MSSQL oder MySQL), das ein relationales Datenbank-Management-System (RDBMS) ist. Nochmal zu erklären, was ein relationales Datenbanksystem [2] ist, spare ich mir an dieser Stelle, denn das Thema wurde bereits ausführlich besprochen. Allerdings möchte ich an dieser Stelle in der Tat klären, was ein dokumentbasiertes Datenbanksystem ist.

Was ist eine dokumentbasierte Datenbank?

Alleine die Aufzählung der Beispiele und deren Bekanntheitsgrad, der relationale Datenbanken im vorigen Abschnitt zeigt, dass wir, was das Speichern von Daten angeht, hauptsächlich in einer relationalen Welt leben. Wir teilen größere und vermeintlich zusammengehörende Daten in kleinere Datensätze auf und legen diese, beschrieben durch ihr Verhältnis zueinander, in einem RDBMS ab. Wollen wir auf die Daten zugreifen, verknüpfen wir die Datensätze mithilfe der Sprache SQL wieder zu einem Ganzen (Datensatz). Das ist ein gelerntes und erprobtes System, das in vielen Anwendungsfällen nahezu ideal ist – vorausgesetzt, eine solche Datenbank ist gut entworfen.

Dagegen ist die Datenstruktur einer dokumentbasierten Datenbank grundlegend anders. Es gibt keine Relationen, Schemata oder Foreign-Key-Verknüpfungen. Alle zusammengehörenden Daten werden jeweils in Dokumenten gespeichert. Sehen wir uns ein Beispiel an.

Ein Onlineshop bietet Produkte zum Kauf an. In einer relationalen Datenbank gibt es deshalb (als Beispiel) drei Tabellen: *products*, *customer*, *orders*. Wenn ein Kunde nun eine Bestellung durchführt, werden seine persönlichen Daten zuerst in der Relation *customer* gespeichert. Nach Auswahl der Produkte, die aus der Tabelle *products* geholt werden, wird in der Tabelle *orders* seine Bestellung aufgenommen. Dort wird eine Referenz auf die Tabelle *customer* (*customer_id*) und eine Referenz auf die Tabelle *products* (*products_id*) zusätzlich zur Anzahl der Produkte gespeichert. Das Ganze geschieht im Schema *s_orders*. Soweit ist es nachvollziehbar. Ein Produkt könnte noch weitere Eigenschaften haben, die in einer weiteren Tabelle *products_details* gespeichert werden, wobei diese dann mit der Tabelle *products* über eine Foreign-Key-Beziehung verknüpft werden sollte. Diese Aufteilung von Daten nennt man im Übrigen „Normalisierung".

In einer dokumentbasierten Datenbank sieht das Ganze etwas anders aus. Um die Produkte abzulegen, wird jeweils ein Dokument erstellt. Der große Unterschied zur Tabelle *products* in der relationalen Datenbank liegt in der Tatsache, dass die Produktdokumente unterschiedlich sein können. Zum Beispiel kann das eine Produktdokument ein Feld *colour* haben und das andere nicht. Das Feld würde nicht den Wert *NULL* anneh-

men, sondern wäre schlichtweg nicht vorhanden. Noch interessanter wird es dann beim Speichern der Bestellung. Auch eine Bestellung ist ein Dokument. Innerhalb dieses Dokuments werden alle Daten als *key-value*-Paare gespeichert: die persönlichen Daten des Kunden, die Produktinformationen und die Mengen der einzelnen Produkte und deren Preise. Es entsteht also ein monolithischer, aber theoretisch veränderbarer Datensatz bzw. besser ausgedrückt, ein Dokument.

Steckbrief CouchDB

CouchDB ist also so eine dokumentbasierte Datenbank. Im Jahr 2005 hat Damien Katz [3] angefangen, CouchDB zu entwickeln. Er hat bereits bei IBM an der zugrunde liegenden Datenbank für Lotus Notes (ebenfalls dokumentbasiert) gearbeitet und wollte seine eigene Vision einer solchen Datenbank umsetzen und der Community als Open-Source-Software zur Verfügung stellen. Zu Anfang hat Damien C als Sprache gewählt, bis ihm Erlang [4] über den Weg gelaufen ist. Erlang und seine extrem große Stärke beim parallelen bzw. gleichzeitigen (concurrent) Ausführen von Anfragen haben ihn so überzeugt, dass er kurzerhand noch einmal alles in Erlang geschrieben hat. CouchDB lässt sich durch einige grundlegende Merkmale beschreiben:

- Dokumentbasiert und schemafrei
- Abfragen (Queries) über Map Reduce per JavaScript
- Alle Daten werden im JSON-Format gespeichert
- RESTful JSON API: *GET*, *POST*, *UPDATE*, *DELETE*, *HEAD*
- Inkrementelle Replikation mit Konfliktlösung (bi-directional Conflict Detection and Resolution)
- Integriert die Eigenschaften von MVCC und ACID
- Open Source unter der Apache-2.0-Lizenz [5]
- Integriertes Webinterface Futon (http://127.0.0.1:5984/_utils/)

Auf die Punkte *Abfragen* und *RESTful JSON-API* werden wir im weiteren Verlauf des Artikels noch näher eingehen. Was MVCC und ACID bedeuten, lesen Sie bitte kurz im Kasten „Erklärung MVCC und ACID“ nach.

An dieser Stelle möchte ich gerne wiederholen, was nicht nur Damien Katz, sondern auch andere Entwickler zum Thema CouchDB vs. RDBMS schon fast predigen. CouchDB und ähnliche Datenbanken sind kein Ersatz für relationale Datenbanken. Es war und ist nie die Intention der Macher dieser Datenbanktypen gewesen, allen anderen Datenbanksystemen den Kampf anzusagen. Leider wird das, auch nicht zuletzt unter dem Begriff „NoSQL“ (Kasten: „NoSQL“), oft falsch verstanden.

Grundlegender Zugriff auf CouchDB

Wie schon im letzten Abschnitt erwähnt, erfolgt der Zugriff auf eine CouchDB-Datenbank über ein RESTful JSON API. Das bedeutet zuerst einmal, dass alle Anfragen und Operationen über HTTP erfolgen. Die Methoden, die verwendet werden können, sind prinzipiell *GET*, um Daten (Dokumente) aus einer Datenbank zu lesen, *PUT*, um Daten in die Datenbank zu schreiben, *POST*, um bestehende Daten in einer Datenbank zu verändern, und *DELETE*, um Daten bzw. Dokumente in der Datenbank zu löschen. Eine nähere Erklärung zum Begriff „RESTful“ lesen Sie im gleichnamigen Kasten. Wichtig ist an dieser Stelle zu verstehen, dass alle Zugriffe auf eine CouchDB per HTTP-Request funktionieren, wobei dabei ein bestimmter URI aufgerufen wird. Als Response wird ein JSON-Objekt geliefert. Sollen Daten in eine Datenbank geschrieben werden, müssen diese ebenfalls als JSON-Objekt vorliegen. Dazu ein paar einfache Beispiele.

Zuerst müssen Sie natürlich eine CouchDB auf Ihrem System installieren. Standardmäßig läuft CouchDB auf dem Port 5984. Wenn Sie sich auf dem gleichen Rechner befinden, auf dem Sie CouchDB installiert haben, kön-

Erklärung MVCC und ACID

MVCC

Multiversion Concurrency Control ist ein Modell, das es unterschiedlichen Benutzern ermöglicht, gleichzeitig auf CouchDB lesend zuzugreifen. Dabei sieht der Benutzer immer einen aktuellen Snapshot der Datenbank. Erst wenn ein Schreibvorgang erfolgreich beendet wurde, kann ein anderer Benutzer den aktuellsten Stand der Daten sehen. Somit wird sichergestellt, dass ein Benutzer immer nur einen konsistenten Zustand der Daten sieht.

ACID

Atomicity: Gleichzeitig ausgeführte Änderungen sind erfolgreich oder es gibt keine Änderungen.
Consistency: Die Datenbank befindet sich immer in einem konsistenten Zustand.
Isolation: Änderungsvorgänge dürfen sich nicht gegenseitig beeinflussen oder überschreiben.
Durability: Die Zusage, dass erfolgreich ausgeführte Änderungen dauerhaft vorhanden sind.

NoSQL

Ein Buzzword? Nein, aber oft so eingesetzt und noch öfter falsch verstanden. NoSQL ist kein Button wie „Atomkraft, nein danke“. NoSQL beschreibt eine Bewegung, die die Nutzung eines anderen Datenbanktyps propagiert – u. a. auch den Ansatz, Daten als Dokumente zu speichern. Prinzipiell werden die Daten nicht in Schemata organisiert und der Zugriff erfolgt ohne die Verwendung von JOINs. Die Idee entstand schon Ende der 90er Jahre und wurde letztlich durch die Weiterentwicklung des Webs wieder belebt. Oftmals sind die Datenstrukturen, z. B. in einem Blog oder einem Wiki so strukturiert, dass es sich eher anbietet, die Datenspeicherung verteilt (distributed) und in einer dokumentbasierten oder auch *key-value*-Datenbank vorzunehmen. Wikipedia gibt weitere Informationen [6]. Lesen Sie außerdem einen klärenden Artikel zu NoSQL von Jan Lehnardt unter [7].

nen Sie einfach *http://127.0.0.1:5984* in einem Browser eingeben (CouchDB muss natürlich gestartet sein).

Sehen wir uns jetzt ein paar einfache Anfragen an. Sie können zwar Ihren Browser nutzen, allerdings bietet es sich an, das Programm Curl [9] auf der Unix-Kommandozeile zu nutzen, weil es einfacher ist, sich die Header des HTTP-Requests anzusehen. Curl nutzt übrigens standardmäßig die Methode *GET*. Der einfachste Fall ist dieser:

```
$ curl http://127.0.0.1:5984/
{"couchdb":"Welcome","version":"0.11.0"}
```

Um sich die gesendeten Header des HTTP-Requests ansehen zu können, geben Sie Curl die Option *-v* mit. Probieren Sie das aus und werden Sie vertraut mit HTTP.

Erstellen wir nun unsere erste CouchDB-Datenbank. Dazu wird die Methode *PUT* verwendet. Um eine andere Methode als *GET* mit Curl verwenden zu können, geben wir die Option *-X* gefolgt von der HTTP-Methode mit. Sagen wir, die Datenbank soll *schubbidu* heißen:

```
$ curl -X PUT http://127.0.0.1:5984/schubbidu
{"ok":true}
```

Gut, das hat geklappt. Wir erhalten als Antwort ein JSON-Objekt, das uns sagt, dass alles gut gegangen ist. Erstellen wir gleich noch eine zweite Datenbank *zickezacke*:

```
$ curl -X PUT http://127.0.0.1:5984/zickezacke
{"ok":true}
```

Und jetzt sehen wir uns an, welche Datenbanken alle existieren. Dafür stellt das CouchDB API eine Methode namens *_all_dbs* zur Verfügung. Der Request sieht folgendermaßen aus:

```
$ curl http://127.0.0.1:5984/_all_dbs
["schubbidu","zickezacke"]
```

Sehr schön, das sind die beiden Datenbanken die wir erstellt haben. Für unsere Anschauungszwecke brauchen wir momentan nur eine Datenbank. Löschen wir also die Datenbank *zickezacke* wieder:

```
$ curl -X DELETE http://127.0.0.1:5984/zickezacke
{"ok":true}
$ curl http://127.0.0.1:5984/_all_dbs
["schubbidu"]
```

Abschließend wollen wir noch ein Dokument erstellen. Dazu müssen Sie wissen, dass jedes Dokument eine eindeutige (unique) *_id* erhalten muss. Da wir gerade keine zur Hand haben, fragen wir einfach CouchDB nach einer:

```
$ curl http://127.0.0.1:5984/_uuids
{"uuids":["2dee255fb520595c45d2c338dc00030d"]}
```

Diese *_id* nutzen wir jetzt, um ein Dokument zu speichern:

```
$ curl -X PUT \
  http://127.0.0.1:5984/schubbidu/2dee255fb520595c45d2c338dc00030d/ \
  -d '{"name":"Andy", "task":"CouchDB Artikel schreiben"}'
{"ok":true,"id":"2dee255fb520595c45d2c338dc00030d", \
   "rev":"1-838f24e606e8b2d77ab6efd6511b26bf"}
```

Die Erklärung zu diesem Request: Mit der *PUT-HTTP*-Methode erstellen wir in der Datenbank *schubbidu* ein Dokument mit der *_id 2dee255fb520595c45d2c338dc00030d*, den Feldern *name* und *task* und füllen diese Felder mit etwas Inhalt. Die Option *-d* sagt dem Programm Curl, dass der nachfolgende String (in unserem Fall das JSON-Objekt) der Body des HTTP-Requests

RESTful

Roy Fielding hat in seiner Dissertation [8] im Jahr 2000 den Begriff „REST“ (Representational State Transfer) eingeführt. RESTful ist davon abgeleitet und beschreibt, wie über die Schnittstelle HTTP auf Ressourcen zugegriffen wird. Dabei gibt es drei grundlegende Charakteristika:

- Zum einen ist der Zugriff auf Ressourcen zustandslos (stateless), was bedeutet, dass ein HTTP-Request auf Ressourcen nach Beendigung abgeschlossen ist und keine Informationen mehr über den Zugriff vorhanden sind.
- Weiterhin wird jede Ressource durch einen URI (Uniform Resource Identifier) beschrieben.
- Der dritte Punkt ist die Tatsache, dass die Information, was mit der Ressource, die angefragt wird, geschehen soll, immer durch die http-Methode beschrieben und nicht in dem URI angegeben wird.

CouchDB installieren

Die Installation der CouchDB ist im einfachsten Fall ziemlich trivial. Auf einem Debian-artigen Linux-System kompilieren Sie CouchDB selbst:

```
$ sudo apt-get build-dep couchdb
$ sudo apt-get install libmozjs-dev libicu-dev libcurl4-gnutls-dev libtool
$ wget http://mirrors.axint.net/apache/couchdb/0.11.0/apache-couchdb- \
  0.11.0.tar.gz
$ tar -zxvf apache-couchdb-0.11.0.tar.gz
$ cd apache-couchdb-0.11.0
$ ./configure
$ make && sudo make install
$ createuser couchdb
$ sudo -u couchdb ./bin/couchdb
```

Der letzte Befehl startet CouchDB als Benutzer *couchdb*. Hat alles geklappt, erhalten Sie folgende Ausgabe:

```
{"couchdb":"Welcome","version":"0.11.0"}
```

ist. Und somit haben wir unser erstes Dokument in einer CouchDB gespeichert. Als Alternative können Sie beim Erstellen eines Dokuments auch die *POST-HTTP*-Methode nutzen. CouchDB erstellt automatisch eine *_id* für Sie. Diese *_id* ist von der Art her die gleiche wie bei der Nutzung von *_uuids*. CouchDB verlangt aber nicht zwangsläufig solch eine Art *_id*. Wenn Sie die *GET-HTTP*-Methode nutzen, können Sie auch selbst eine *_id* wählen: z. B. *produkt_1*, *produkt_2* oder was auch immer. Der Vorteil: Sie haben selbst die Kontrolle über die *_id* und können diese „sprechender" gestalten. Der Nachteil: Sie müssen selbst darauf achten, dass jede *_id* in der Datenbank nur exakt einmal (unique) vorkommt. Natürlich gibt es auch noch die Möglichkeit, Datenbanken und Dokumente zu aktualisieren und zu löschen. Dazu gleich mehr.

Sie haben anhand der Beispiele und der Verwendung des Programms Curl sehr gut sehen können, was RESTful bedeutet. Die HTTP-Methode (*PUT*, *POST* etc.) bestimmt, was mit dem URI und den sich dahinter befindlichen Ressourcen geschehen soll. Weil Sie mit Sicherheit nicht direkt Curl auf der Kommandozeile benutzen wollen, um mit CouchDB zu sprechen, werden wir im Folgenden einen PHP Wrapper für diese Aufgaben erstellen. Für weitere Informationen sehen Sie sich bitte im CouchDB-Wiki das API Cheatsheet an [10].

Ein PHP Wrapper für CouchDB

Kommen wir endlich auf den Punkt. Im vorigen Abschnitt haben Sie die Prinzipien des RESTful CouchDB APIs kennengelernt. Solange Sie nur ein bisschen mit dem API spielen wollen, ist Curl Ihr Freund. Aber in einem PHP-Projekt brauchen wir einen Wrapper, der die Interaktion mit CouchDB so einfach wie möglich macht. Solch einen Wrapper bauen wir jetzt. Die HTTP-Requests werden in diesem Wrapper mit der Curl-Erweiterung von PHP umgesetzt. Das bedeutet, dass die PHP-Erweiterung *php5-curl* installiert sein muss. Till Klampaeckel geht in seinem Projekt Armchair [11] einen etwas anderen Weg und nutzt das PEAR-Paket *HTTP_Request2*, um die HTTP-Requests zu realisieren. Ich habe mich allerdings dazu entschieden, die Anforderungen so gering wie möglich zu halten und gehe davon aus, dass die PHP-Curl-Erweiterung auf den meisten Servern vorhanden sein wird. Lassen Sie uns den Wrapper *CouchDB_PHP* nennen. Der Rahmen der Klasse sieht wie in Listing 1 aus.

Eine Anmerkung vor allen nächsten Schritten: Ich werde nicht den gesamten Code der Klasse in diesem Artikel hinschreiben und Sie finden diesen auch nicht auf der Heft-CD. Denn es gibt ja so etwas Tolles wie Github. Und dort finden Sie unter http://github.com/andywenk/CouchDB_PHP den gesamten Quellcode zum Ansehen, Herunterladen, Forken und zum Klonen: *$ git clone git@github.com:andywenk/CouchDB_PHP.git*. Sollten Sie sich mit Git noch nicht auskennen und es auch noch nicht installiert haben, hätte ich da einen Vorschlag, wie Sie den heutigen Abend verbringen könnten.

Zurück zu unserer Klasse. Zu Beginn werden einige Klassenvariablen deklariert. *$host*, *$protocol* und *$port* können direkt verändert werden (weil *public*) und die anderen Variablen müssen über einen Setter und *$db* auch über den Konstruktor gesetzt oder können wie *$last_id* nur über einen Getter aufgerufen werden. Damit ist schon klar, dass ein neues Objekt (oder namentlich Exemplar, aber bitte nicht Instanz) mit *$couch = new CouchDB_PHP();* erzeugt wird, wenn es noch kei-

Listing 1

```
class CouchDB_PHP {
    public $port = 5984;
    public $protocol = 'http';
    public $host = "127.0.0.1";
    protected $last_id;
    protected $db;
    protected $id;
    protected $response_type = 'array';
    protected $request;

    public function __construct($db = null) {
        $this->db = $db;
    }
}
```

Listing 2

```
public function http_request($type = 'GET', $json_data = '') {
  if(!function_exists('curl_init')) {
      return self::error('no_curl');
    }

    $ch = curl_init();

    if($type == 'PUT' || $type == 'POST') {
      if(!empty($json_data)) {
          curl_setopt($ch, CURLOPT_POSTFIELDS, $json_data);
        }
    } elseif ($type == 'GET' || $type == 'DELETE') {
    } else {
      return self::error('invalid_http_method');
    }

    curl_setopt($ch, CURLOPT_URL, self::create_request_url());
    curl_setopt($ch, CURLOPT_HEADER, false);
    curl_setopt($ch, CURLOPT_USERAGENT, 'CouchDB_PHP');
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
     curl_setopt($ch, CURLOPT_CUSTOMREQUEST, $type);

    $response = curl_exec($ch);
    curl_close($ch);

    return self::parse_response($response);
    }
```

ne Datenbank gibt und wir zuerst eine erzeugen wollen, oder mit *$couch = new CouchDB_PHP("hanswurst");*, wenn wir mit einer bestehenden Datenbank sprechen wollen. Wenn Sie während der Arbeit mit CouchDB die Datenbank wechseln wollen, können Sie auch über den Setter *set_db($db)* eine andere Datenbank wählen. In den meisten Fällen werden Sie allerdings ein neues Objekt erzeugen wollen.

Der Kern des Wrappers besteht im Prinzip aus drei Methoden. Die Methode *http_request()* erzeugt die Anfrage an CouchDB unter Verwendung der PHP-Curl-Erweiterung. Den Code sehen Sie in Listing 2.

Es werden zwei Parameter erwartet. Zum einen die HTTP-Request-Methode, wobei der Standard als *GET* festgelegt wird, und außerdem ein Datenobjekt im JSON-Format, wobei dieser Parameter optional ist. Innerhalb der Methode wird nach dem obligatorischen *curl_init()* und dem Test, ob Curl überhaupt vorhanden ist, geprüft, um welche Art Request es sich handelt. Handelt es sich um einen PUT oder POST Request und sind dann auch noch Daten in *$json_data* vorhanden, werden diese in den Body des HTTP-Requests über die Curl-Option *CURLOPT_POSTFIELDS* gepackt und später übermittelt. Die weiteren Curl-Optionen sind relativ trivial. Wichtig ist letztlich noch die Option *CURLOPT_CUSTOMREQUEST*, mit der wir die HTTP-Request-Methode *GET*, *POST*, *PUT* oder *DELETE* angeben. Im Anschluss wird dann der Request durch *curl_exec()* ausgeführt und das Ergebnis, das im Fall von CouchDB immer im JSON-Format vorliegt, in der Variablen *$response* gespeichert.

Die zweite Kernmethode ist *create_request_url()*. Hier wird der komplette Request-URL zusammengebaut und geprüft, ob in *$this->request* überhaupt etwas steht. Ein Request nach dem RESTful-Prinzip muss immer ausreichend Informationen enthalten, damit die angesprochene Applikation irgend etwas tun kann. Im Fall von CouchDB muss dies zumindest der Name einer zu erstellenden neuen Datenbank sein. Diese Methode wird direkt aus *http_request()* aufgerufen. Den Inhalt sehen Sie in Listing 3.

Die dritte Kernmethode ist *parse_response()*. Diese ist ziemlich klein, wird aber bei jedem Aufruf der Methode *http_request* ausgeführt. Den Inhalt sehen Sie in Listing 4.

Hier wird entschieden, ob der Wrapper die Daten aus der Antwort von CouchDB im JSON-Format belässt oder als assoziatives Array zurückgibt. Das Standardverhalten ist dabei die Array-Variante. Vorher wird noch die zurückgelieferte ID aus dem JSON Response von CouchDB in eine Klassenvariable gepackt. Damit kann auf diese von außen immer separat zugegriffen werden, egal ob die Daten bei der Rückgabe aus der Klasse im Array- oder JSON-Format vorliegen.

Sehen wir uns jetzt die einzelnen Operationen an, die wir brauchen, um mit der CouchDB zu sprechen. Zuerst einmal sehen wir nach, ob es bereits Datenbanken in unserem Cluster gibt. Über das native JSON HTTP API würden wir das folgendermaßen tun:

request: curl http://127.0.0.1:5984/_all_dbs/
response: ["firlefanz","hanswurst","schubbidu"]

Die Methode *show_all_dbs()* erledigt das für uns:

```
$couch = new CouchDB_PHP();
$res = $couch->show_all_dbs();
Array
(
    [0] => firlefanz
    [1] => hanswurst
    [2] => schubbidu
)
```

Sehr schön. Im nächsten Schritt erstellen wir eine CouchDB-Datenbank:

request: curl -X PUT http://127.0.0.1:5984/php_mag_5_10
response: {"ok":true}

Im Wrapper übernimmt das die Methode *create_db()*:

```
$couch = new CouchDB_PHP();
```

Listing 3

```
protected function create_request_url() {
  if(empty($this->request)) return self::error('create_url_no_request');
     return "{$this->protocol}://{$this->host}:{$this->port}/{$this->request}";
}
```

Listing 4

```
protected function parse_response($json_response) {
  array_response = json_decode($json_response, true);
   $this->last_id = (array_key_exists('id', $array_response))
       ? $array_response['id'] : '';
  return ($this->response_type == 'array') ? $array_response : $json_response;
}
```

Listing 5

```
$couch = new CouchDB_PHP('php_mag_5_10');
$couch->set_id('a733607681750a8aa9f14e5f25000c56');
$data = array("_rev" => "2-43c04835d6a4d4ce2635a8e19bfaacbd",
      "Ausgabe" => "5.2010");
$res = $couch->update_doc($data);
Array
(
    [ok] => 1
    [id] => a733607681750a8aa9f14e5f25000c56
    [rev] => 3-eba35a4de201c9e533fe1965f2d317fb
)
```

```
$res = $couch->create_db('php_mag_5_10');
Array
(
    [ok] => 1
)
```

So weit gut. Der nächste logische Schritt ist es, Dokumente zu erstellen. Das kann, wie bereits oben erwähnt, auf zwei Wegen geschehen. Wenn wir eine individuelle ID nutzen möchten, geben wir diese an und nutzen die HTTP-Methode *PUT*. Im anderen Fall, wenn wir die Generierung der ID CouchDB überlassen wollen, verwenden wir die HTTP-Methode *POST* und geben keine ID an. Denken Sie daran, dass die ID unique sein muss. Sprich, wenn Sie die ID selbst wählen wollen, sollten Sie sich über deren Generierung Gedanken machen und einen entsprechenden Algorithmus verwenden. Hier das Beispiel (ID wird von CouchDB erzeugt):

```
request: curl -X POST http://127.0.0.1:5984/php_mag_5_10
                                                -d '{"ausgabe":"5.10"}'
response: {"ok":true,"id":"a733607681750a8aa9f14e5f25000c56",
 "rev":"1-c81393f0b50296ff143a4284acbc7f72"}
```

Und hier die Variante mit dem Wrapper:

```
$couch = new CouchDB_PHP('php_mag_5_10');
$couch->create_doc(array("ausgabe" => "5.10"));
Array
(
    [ok] => 1
    [id] => a733607681750a8aa9f14e5f250000a3
    [rev] => 1-6bdd9ea372b128c3d7e68d7c89ddf291
)
```

Perfekt! Somit haben Sie das erste Dokument in der CouchDB *php_mag_5_10* erstellt. Als letztes Beispiel möchte ich anhand eines Beispiel zeigen, wie Sie ein Dokument aktualisieren:

```
request: curl -X PUT
  http://127.0.0.1:5984/php_mag_5_10/
                                    a733607681750a8aa9f14e5f25000c56
  -d '{"_rev":"1-c81393f0b50296ff143a4284acbc7f72", "Heft":"5.10"}'
response: {"ok":true,"id":"a733607681750a8aa9f14e5f25000c56",
        "rev":"2-43c04835d6a4d4ce2635a8e19bfaacbd"}
```

Hierbei ist gut zu sehen, dass die *HTTP-PUT*-Methode genutzt wird und zum einen die *_id* im URL geht (RESTful) und zum anderen die Version des Dokuments in das JSON-Objekt, das wir als Body des HTTP-Requests mitsenden. Sehen wir uns das Ganze mit dem Wrapper an (Listing 5).

Wie Sie sehen, habe ich aus dem obigen Request per Curl einfach die *id* und *rev* übernommen und den Inhalt wieder etwas geändert. Als Ergebnis erhalten Sie eine neue Revision. Anhand derer sehen Sie, dass das Dokument bereits zum zweiten Mal geändert wurde und deshalb in der Version 3 vorliegt.

Ich denke, mit diesen Beispielen wird klar, wie der Zugriff auf CouchDB über das RESTful JSON HTTP API funktioniert. Der Wrapper bietet natürlich noch weitere grundlegende Methoden, um Datenbanken und Dokumente zu erstellen, zu aktualisieren und zu löschen. In der Datei *couch_php_examples.php* im Ordner */examples* im Github Repository finden Sie einige weitere Beispiele. Ich ermutige Sie an dieser Stelle mit dem Motto: Learning by Doing.

Fazit

Ich habe Ihnen in diesem Artikel gezeigt, wie Sie mit einem einfachen Wrapper auf eine CouchDB-Datenbank zugreifen können. Die grundlegenden Prinzipien, nach denen CouchDB arbeitet, haben Sie durch die Vorstellung des RESTful JSON APIs kennengelernt. Und dass das alles relativ einfach ist und sich die RESTful-Architektur als äußerst passend zeigt, haben Sie auch gesehen.

Wie geht es weiter? Nun, bislang sind Sie in der Lage, Datenbanken und Dokumente zu erstellen und diese zu aktualisieren oder zu löschen. CouchDB kann aber noch viel mehr. Bis zu diesem Punkt müssen Sie z. B. eine Teilmenge aus den erhaltenen Daten in PHP erstellen. Das kann aber auch CouchDB für Sie erledigen. Das geschieht über so genannte *_design*-Dokumente, die *_views* beinhalten. Ein *_view* enthält wiederrum in JavaScript geschriebene Abfragen. Aber das – Sie ahnen es schon – wäre das Thema eines weiteren Artikels.

Übrigens, CouchDB hat eine extrem gute, aktive und hilfsbereite Community. Schauen Sie doch mal vorbei unter http://couchdb.apache.org/community/.

Andy Wenk ist Software Developer bei SinnerSchrader in Hamburg, hat momentan viel Spaß mit Ruby on Rails und mag Datenbanken wie PostgreSQL und CouchDB. Zusammen mit Till Klampaeckel (@klimpong) aus Berlin schreibt er momentan das erste deutschsprachige CouchDB-Buch. Sie erreichen ihn unter andy@nms.de und @awenkhh.

Links & Literatur

[1] http://couchdb.apache.org
[2] http://goo.gl/IEGR
[3] http://damienkatz.net/
[4] http://erlang.org
[5] http://apache.org/licenses/
[6] http://en.wikipedia.org/wiki/NoSQL
[7] http://goo.gl/ZAYb
[8] http://goo.gl/Q71Y
[9] http://goo.gl/jgx7
[10] http://wiki.apache.org/couchdb/API_Cheatsheet
[11] http://goo.gl/ubmD