DEPLOY AND SCALE PHP APPS INTO THE CLOUD WITH EASE

www.engineyard.com

DZone Refcardz

Sponsored By: Engine Yard

CONTENTS INCLUDE:

Updated for PHP 5.4

PHP 5.4

By Bradley Holt

ABOUT PHP 5.4

PHP is a scripting language most commonly used for processing and rendering Web pages server-side. Its syntax is simple enough to be easily learned by novices, yet the language is powerful enough to run some of the world's most popular websites. PHP is also the language of choice for many popular web software packages.

DATA TYPES

PHP has scalar data types, compound data types, and special data types. PHP is a weakly typed language. Variables are not declared with a specific data type. Instead, a variable's type is determined based on the context in which it is used. This is often referred to as type juggling. It is possible to use type casting in order to force a variable to be evaluated as a specific type. See http://php.net/types.type-juggling for details on type juggling and type casting. PHP has two sets of comparison operators, those that compare values after type juggling and those that compare both values and types. See http://php.net/types.comparisons for details on how PHP handles type comparisons.

The scalar data types in PHP are boolean, integer, float, and string. By definition, scalar data types can only contain a single value. Boolean values can either be true or false. Integers are whole numbers that also include negative numbers (PHP does not support unsigned integers). The size of an integer is dependent on the platform on which PHP is running. Integer literals can be specified in decimal, octal, hexadecimal, or binary (as of PHP 5.4) format. A float is a number that can contain a fraction. Like integers, the size of a float is dependent on the platform on which PHP is running. Float literals can be specified in decimal format or in scientific notation format. A string in PHP is really just an array of bytes, with each byte typically representing a character. See the section on strings for more information about working with strings.

The compound data types in PHP are array and object. See the respective sections on arrays and objects for details on working with each.

The special data types in PHP are resource, NULL, and callable. A resource is a special type that holds a reference to some external resource. This external resource can be a stream or a database connection, for example. A complete list of resource types can be found at http://php.net/resource. NULL is both a data type and the only value possible for that type. A variable with a NULL value is a variable with no value. A variable can come to be NULL by being assigned the special NULL value, by being declared but not yet set to any value, or by having its value cleared with the unset function. Some of PHP's built-in functions accept user-defined callbacks. These parameters are considered to be of type callable. The callable type can also be used as a type hint within your user-defined functions and methods.

STRINGS

PHP supports several syntaxes for string literals: single quoted, double quoted, heredocs, and (added in PHP 5.3) nowdocs. Single quoted strings is the simplest syntax. See the documentation on strings at http://php.net/types.string for more information about working the various string syntaxes.

Double quoted strings support variable interpolation. This means that you can place a variable within a double quoted string and the variable's value will be expanded into the resulting string. Single quoted strings do not support variable interpolation.

The heredoc syntax works similarly to double quoted string literals. Heredocs make it easy to create multi-line string literals. The heredoc syntax also supports variable interpolation. Nowdocs (added in PHP 5.3) are the single quoted string equivalent of heredocs. Nowdocs do not allow for variable interpolation.

String Functions

PHP has many useful built-in string functions. For example, the implode function produces a string by gluing together array values using a given string. The explode function does the opposite and splits a string by a delimiter, returning an array. You can find the full list of string functions at http://php.net/ref.strings.

Regular Expressions

Regular expressions are a technique for pattern matching against strings. PHP supports Perl-Compatible Regular Expressions (PCRE). You can find out more about PCRE at http://php.net/pcre.

ARRAYS

PHP arrays are an ordered map, meaning that values are mapped to keys. PHP arrays can be used as arrays, lists, hash tables, dictionaries, collections, stacks, and queues. Although PHP arrays support many different data structures, the Standard PHP Library (SPL) provides specialized data structures that are often faster and use less memory (but are less flexible). See http://php.net/spl.datastructures for a list of available SPL data structures.

PHP supports both enumerative and associative arrays. Enumerative arrays are indexed by an integer key, while associative arrays are indexed by a string key. Array values can be any PHP data type, including arrays. When an array contains an array it is referred to as a multidimensional array. PHP has many built-in array related functions. The full list of array functions can be found at http://php.net/ref.array.

Creating Arrays

PHP arrays have a flexible syntax. An example of creating an array using the array() language construct:

```
$colors = array('red', 'green', 'blue');
```

The above is an example of an enumerative array, meaning that it is indexed by a numerical key. Since the keys were not specified, PHP will automatically increment the key starting with 0.

PHP 5.4 introduced a short array syntax:

```
$colors = ['red', 'green', 'blue'];
```

An example of creating an associative array:

```
$colors = array(
    'red'   => 83,
    'green' => 114,
    'blue'  => 124
);
```

The output will be identical to the previous example:

PHP 5.4 introduced a short array syntax, allowing you to replace array() with []. An example:

```
$colors = ['red', 'green', 'blue'];
```

An example of creating an associative array:

```
$colors = array(
    'red'   => 83,
    'green' => 114,
    'blue'  => 124
);
print_r($colors);
```

Accessing Array Values

Individual array values are accessed by their keys. An example of accessing a value within an enumerative array:

```
$colors = array('red', 'green', 'blue');
print_r($colors[0]); // red
```

When accessing a value within an associative array, you would provide a string as the key instead of an integer. PHP 5.4 introduced function array dereferencing, meaning that you can use the result of a function that returns an array directly without first needing to assign the result to a variable. For example:

```
function getColors()
{
    return array(
        'red'   => 83,
        'green' => 114,
        'blue'  => 124
    );
}
print_r(getColors()['red']); // 83
```

Iteration Constructs

The simplest way to iterate over the values of an array is with the foreach language construct. An example:

```
$colors = array(
    'red'   => 83,
    'green' => 114,
    'blue'  => 124
);
foreach ($colors as $value) {
    var_dump($value);
}
```

You can also access the array keys while iterating, for example:

```
$colors = array(
    'red'   => 83,
    'green' => 114,
    'blue'  => 124
);
foreach ($colors as $key => $value) {
    var_dump($key);
    var_dump($value);
}
```

Sorting Arrays

PHP has several built-in functions for sorting arrays in various ways. Some functions sort on array keys, others on values. Some functions maintain key/value associations after sorting, others do not. Functions can sort in ascending, descending, numerical, natural, random, or user-defined order. All of the array sort functions manipulate the source array, rather than returning a new array that is sorted. Read the documentation on sorting arrays at http://php.net/array.sorting for more details.

FUNCTIONS

PHP has well over one thousand internal, or built-in, functions (the number of internal functions depends on which extensions are installed). In addition to its internal functions, PHP supports user-defined functions. Functions can accept arguments and can return values.

Internal Functions

Using an internal function is as simple as calling that function using its name, for example:

```
$id = uniqid();
print_r($id);
```

User-Defined Functions

User-defined functions can be created by providing a name, zero or more arguments, and a return value (optional). For example:

```
function sayHello($to)
{
    return "Hello, $to.";
}
```

Anonymous Functions

PHP 5.3 introduced anonymous functions in the form of lambdas and closures. An example of defining and calling a lambda:

```
$sayHello = function($to)
{
    return "Hello, $to.";
};
$message = $sayHello('Bradley');
print_r($message); // Hello, Bradley.
```

Unlike lambdas, closures use variables from the scope in which the closure was defined. For example:

```
$greeting = 'Hello';
$sayHello = function($to) use($greeting)
{
    return "$greeting, $to.";
};
$message = $sayHello('Bradley');
print_r($message); // Hello, Bradley.
```

CLASSES AND OBJECTS

Object-oriented programming is a commonly used software design paradigm. Objects encapsulate properties as well as methods that operate on those properties. An object is instantiated from a class and multiple object instances can be instantiated from the same class. Please refer to the documentation on classes and objects at http://php.net/oop5 for more details.

DATES AND TIMES

PHP has many built-in functions and classes for handling dates and times. In order to use this functionality, the default timezone should first be set. This can be done with the date.timezone PHP INI setting. The full list of supported timezones can be found at http://php.net/timezones.

The DateTime class can be used to represent a specific date and time, and to perform operations on date and time values. To instantiate a new DateTime instance, provide the date/time in a valid format (an optional timezone can be supplied as the second parameter):

```
$date = new DateTime('2012-05-18');
```

Formatting

The DateTime:: format method allows a date/time to be formatted in pretty much any way conceivable through the use of format characters. The DateTime class includes several predefined constants for common formats. For example, outputing a date/time in ISO 8601 format:

```
$date = new DateTime('2012-05-18');
$formattedDate = $date->format(DateTime::ISO8601);
print_r($formattedDate); // 2012-05-18T00:00:00+0000
```

The list of predefined constants for the DateTime class can be found at http://php.net/class.datetime. The list of format characters can be found at http://php.net/date.

Date Math

PHP also includes a DateInterval class that can be used to add or subtract dates, as well as to find the difference between two dates. This date math can done with the DateTime::add, DateTime::sub, and DateTime::diff methods, respectively. Following is an example of subtracting one day from a date:

```
$date = new DateTime('2012-05-18');
$interval = new DateInterval('P1D');
$date->sub($interval);
print_r($date->format(DateTime::ISO8601)); // 2012-05-17T00:00:00+0000
```

DATABASES

PHP supports integration with databases through a variety of database extensions. PHP Data Objects (PDO) is a very commonly used database abstraction layer. Additionally, there are over twenty vendor-specific database extensions available in PHP.

PHP Data Objects (PDO)

The PHP Data Objects (PDO) extension provides a common interface for accessing multiple types of databases in PHP. See the list of database-specific PDO drivers at http://php.net/pdo.drivers for a complete list of supported databases.

Connecting

Creating an instance of the PDO class establishes a database connection. The first parameter to the PDO class' constructor is the data source name (DSN). A DSN can describe properties of the data source such as the type of database, hostname, port number, and database name. The second parameter to the PDO class' constructor is the username and the third is the password. A fourth parameter specifying driver options may also be used. To connect to a MySQL database:

```
try {
    // Assumes $username and $password have been previously set
    $connection = new PDO('mysql:host=localhost;dbname=mydb',
$username, $password);
} catch (PDOException $ex) {
    // Handle exception
}
```

The PDO constructor will throw a PDOException if there is a connection error. An exception is a special type of object that can be thrown and caught in the case of an abnormal condition. The PDOException in this example can be caught and handled in a way that is appropriate for your application.

Executing a Query

To perform a SELECT query, call the PDO::query method. This will return a PDOStatement object which can be treated as an array of database rows over which you can iterate:

```
$connection = new PDO('mysql:host=localhost;dbname=mydb', $username,
$password);
$rows = $connection->query('SELECT firstname, lastname FROM people');
foreach ($rows as $row) {
    echo $row['firstname'] . ' ' . $row['lastname'] . PHP_EOL;
}
```

Prepared Statements

When passing parameters to a query, you should always use prepared statements. There are two major benefits to this approach. One, your queries are not susceptible to SQL injection attacks if all parameters are passed via prepared statements. Two, a query can be parsed (analyzed, compiled, and optimized) once and then executed multiple times with different parameter values.

Hot Tip

Not all of the databases with PDO drivers support prepared statements. If you are using PDO with a database that does not support prepared statements, then PDO will attempt to emulate prepared statements. It is important to be aware that, in this situation, you are not getting all of the benefits of prepared statements. For example, your queries may be vulnerable to SQL injection attack. This is true even if PDO still escapes your data—escaping alone is not enough to mitigate all SQL injection attacks.

Working with prepared statements involves first preparing the statement, then binding the parameters to values, and finally executing the statement:

```
$connection = new PDO('mysql:host=localhost;dbname=mydb', $username,
$password);
// Prepare the statement
$statement = $connection->prepare(
    'UPDATE people SET email = :email WHERE id = :id'
);
// Bind the email parameter's value
$statement->bindValue(':email', 'bob@example.com');
// Bind the id parameter's value
$statement->bindValue(':id', 42);
// Execute the statement
$success = $statement->execute();
// Check for success
var_dump($success); // bool(true)
```

Alternatively, you can bind a variable by reference using the PDOStatement::bindParam method. With this approach, the value of the variable will be evaluated when the statement is executed, not when it is bound.

Transactions

Many of the databases supported by PDO support ACID (Atomicity, Consistency, Isolation, Durability) properties. These properties allow for the use of transactions. A transaction is a unit of work that succeeds or fails as a whole. You can group a series of queries into a transaction through use of the PDO::beginTransaction and PDO::commit methods. A transaction that has been started through PDO::beginTransaction can be rolled back by using the PDO::rollBack method. Read the documentation on transactions and auto-commit at http://php.net/pdo.transactions for more details.

Object-Relational Mapping (ORM)

Object-relational mapping (ORM) is a technique for mapping objects to a relational database. An ORM can be very helpful when using both object-oriented programming and a relational database. Two of the most popular PHP ORMs are Doctrine (http://www.doctrine-project.org/) and Propel (http://www.propelorm.org/).

SECURITY

The two most important security practices for PHP programs are filtering input and escaping output. These are the areas from which the majority of security vulnerabilities originate. Some of the most important security exploits to be aware of are SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), and session fixation. Another common problem is securely storing sensitive user data, such as passwords. Note that there is much more to security than what is covered here. Please consider these tips to be a starting point, and not a comprehensive treatment of the subject.

Filter Input

Input can come from many sources such as the end user, a database, an incoming web services API request, and countless other possible sources. At its most basic level, filtering is a form of a "sanity check" on a given input and then sanitizing that input appropriately for the given context. The following is one example of filtering input.The strip_tags function can mitigate XSS attacks by removing HTML and PHP tags from an input string:

```
$unfiltered = '<a onclick="doBadStuff()">Click Me!</a>';
$filtered = strip_tags($unfiltered);
print_r($filtered); // Click Me!
```

Escape Output

Your application may send output to many places including the end user, a database, an outgoing web services API request, and many other possible outlets. Escaping output basically means ensuring that the output is encoded properly for a given destination. Following is one example of escaping output.

The htmlspecialchars function can mitigate XSS attacks by converting special characters to their equivalent HTML entities:

```
$unescaped = '<a onclick="doBadStuff()">Click Me!</a>';
$escaped = htmlspecialchars($unescaped);
print_r($escaped); // &lt;a onclick=&quot;doBadStuff()&quot;&gt;Click
Me!&lt;/a&gt;
```

Password Encryption

Often an application will need to store sensitive user data, such as passwords. Passwords should never be stored in cleartext. Passwords should be encrypted in order to mitigate the impact of compromised passwords. In the case of passwords, the encryption should be one-way. One-way encryption means that, once a value is encrypted, it cannot be decrypted. Two encryption methods that meet these requirements are bcrypt (a cryptographic hash function) and PBKDF2 (a key derivation function). These encryption algorithms are adaptive, meaning that they can be made slower to keep up with Moore's law.

Hot Tip

You will find many resources on hashing passwords—or salting and hashing passwords—using algorithms such as MD5 or SHA1. Hashed passwords are susceptible to rainbow table attacks. A rainbow table is a precomputed list of reversed hashes. Salting helps mitigate rainbow table attacks, but hashed and salted passwords are still vulnerable to brute force attacks. An adaptive hashing algorithm such as bcrypt exponentially increases the cost of launching a brute force attack, as compared to a standard hashing algorithm.

When a user first registers and provides his or her password, encrypt the password as follows:

```
// Password supplied upon registration
$registrationPassword = 'MySuperSecretPassword';
// Create a unique 22 character salt
$salt = substr(str_replace('+', '.', base64_
encode(sha1(microtime(true), true))), 0, 22);
// Choose a two digit cost parameter from 04–31
$cost = '16';
// Create an encrypted hash
$encryptedHash = crypt($registrationPassword, '$2a$'. $cost. '$' .
$salt);
```

The $2a$ string at the beginning of the second parameter instructs the crypt function to use the Blowfish cypher (on which bcrypt is built). The two digit cost (a value between 04 and 31) indicates how much computational power should be used. The higher the cost parameter, the longer it will take for crypt to execute, and the more secure your users' passwords will be. As computers get faster, this value should be increased. Finally, the salt prevents rainbow table attacks. See the crypt function's documentation at http://php.net/crypt for more details.

The encrypted hash ($encryptedHash) can now be stored in your database for reference when the user attempts to login. When the user attempts to login, the original cost, salt, and encrypted password can be extracted from the encrypted hash as follows:

```
// Assumes $encryptedHash has been retrieved from the database
// Extract the original cost
$cost = substr($encryptedHash, 4, 2);
// Extract the original salt
$salt = substr($encryptedHash, 7, 22);
```

Finally, we can generate an encrypted hash using the password from the user attempting to login and compare it to the stored encrypted hash:

```
// Password supplied during a login attempt
$loginAttemptPassword = 'MySuperSecretPassword';
// Create an encrypted hash
$loginAttemptEncryptedHash = crypt($loginAttemptPassword, '$2a$'.
$cost. '$' . $salt);
// Compare the two encrypted hashes
if ($loginAttemptEncryptedHash == $encryptedHash) {
    echo 'Login successful!';
} else {
    echo 'Login failed!';
}
```

As you can see demonstrated above, it is not necessary to store a user's cleartext password in order to later verify that user's password when he or she attempts to login. You may choose to instead use the PBKDF2 key derivation function, but PBKDF2 is not as easy to implement as bcrypt using PHP's built-in functionality.

FRAMEWORKS

A framework is a tool that aims to provide a set of features that can be reused from project to project, making more efficient user of programmers' time. There are many frameworks available to PHP programmers. Some of the most popular frameworks include Zend Framework, Symfony, CodeIgniter, CakePHP, and Lithium. Microframeworks such as Silex, Slim, and Limonade are also gaining in popularity. Choosing a framework is a decision involving many context-specific factors. A comparison of frameworks is beyond the scope of this reference guide.

CONFIGURATION

PHP's behavior can be configured at a variety of levels:

Global Configuration

The php.ini file is PHP's configuration file, containing more than 200 directives capable of tweaking nearly every aspect of the language's behavior. This file is parsed every time PHP is invoked, which for the server module version occurs only when the web server starts, and every time for the CGI version.

PHP is packaged with two INI files. You can choose to use one of these files as the starting point for your PHP configuration settings. The php.ini-production file contains settings that are recommended for a production environment and the php.ini-development file contains settings that are recommend for a development environment. Likely your operating system or package manager has already picked one of these starting points for you. If so, you can still find references to recommend development and production values in your php.ini file. For example, the recommended values for the display_errors directive from the php.ini file (the semicolon character is a comment, meaning that these lines are for reference only):

```
; display_errors
;   Default Value: On
;   Development Value: On
;   Production Value: Off
```

The actual setting, also in the php.ini file, contains details as to why those settings are recommended:

```
; This directive controls whether or not and where PHP will output errors,
; notices and warnings too. Error output is very useful during development, but
; it could be very dangerous in production environments. Depending on the code
; which is triggering the error, sensitive information could potentially leak
; out of your application such as database usernames and passwords or worse.
; It's recommended that errors be logged on production servers rather than
; having the errors sent to STDOUT.
; Possible Values:
;   Off = Do not display any errors
;   stderr = Display errors to STDERR (affects only CGI/CLI binaries!)
;   On or stdout = Display errors to STDOUT
; Default Value: On
; Development Value: On
; Production Value: Off
; http://www.php.net/manual/en/errorfunc.configuration.php#ini.display-errors
display_errors = On
```

Host- and Directory-specific Configuration

If you lack access to the php.ini file, you may be able to change desired directives within Apache's httpd.conf or .htaccess files. For instance, to force the display of all PHP errors for your development machine, add the following to an .htaccess file:

```
php_value error_reporting -1
php_flag display_errors on
```

Displaying errors is very helpful—and highly recommended—in development. For security reasons, though, you should not display errors in production. Instead, errors can be logged for later review. For more details please read the documentation on error handling and logging at http://php.net/errorfunc.

Hot Tip

Each directive is assigned one of three permission levels (PHP_INI_ALL, PHP_INI_PER_DIR, PHP_INI_SYSTEM) which determines where it can be set. Be sure to consult the PHP documentation before tweaking settings outside of the php.ini file. See http://php.net/ini for a complete list of directives.

Script-specific Configuration

Occasionally you'll want to tweak directives on a per-script basis. For instance to change PHP's maximum allowable execution time for a script tasked with uploading large files, you could call the ini_set() function from within your PHP script like so:

```
ini_set('max_execution_time', 60);
```

BUILT-IN WEB SERVER

PHP 5.4 introduced a built-in web server. This functionality should only be used for development purposes—it should not be used in production. To start the web server, first navigate to your project's document root and then start the web server:

```
$ cd public
$ php -S localhost:8080
```

The web server will now be accessible at http://localhost:8080/.

DEBUGGING

A common debugging tactic in PHP is to make calls to the print_r or var_dump functions within your code. The print_r function accepts an expression to be printed and an optional boolean value indicating whether or not the function should return its output rather than printing it directly (the default value is false). The var_dump function accepts one or more expressions to be dumped and provides more detail about the expression or expressions.

A helpful PHP extension for debugging is Xdebug. When installed, Xdebug replaces PHP's var_dump function with its own. Xdebug's var_dump function outputs HTML (assuming PHP's html_errors configuration directive is set to On and PHP is not being used from the command line) with different colors for different data types.

Hot Tip

Xdebug provides a powerful set of debugging and profiling features. These features include stack traces, function traces, infinite recursion protection, time tracking, improved variable debugging display, code coverage analysis, code profiling, and remote debugging. Xdebug should not be installed on production servers. More information about Xdebug can be found at http://xdebug.org/.

A better approach to debugging than that described previously is to use Xdebug's remote debugging feature. This feature allows a developer to interactively step through and debug PHP code. Xdebug's remote debugging feature is supported by a command line client that is bundled with Xdebug as well as a number of integrated development environments (IDE) and other tools. A list of supported clients can be found at http://xdebug.org/docs/remote.

UNIT TESTING

Unit testing is a method of verifying the correctness of code components through automated tests. PHPUnit is by far the most popular unit testing framework for PHP applications. Using PHPUnit, you can write tests that exercise your code and perform assertions to check that your code behaves correctly. More information about PHPUnit can be found at http://www.phpunit.de/.

PACKAGE AND DEPENDENCY MANAGEMENT

PHP Extension Application Repository (PEAR)

The PHP Extension Application Repository (PEAR) is a package management tool and component distribution system. Packages are available from the main repository at http://pear.php.net/ as well as from independently maintained repositories. Once PEAR is installed on your system, installing a PEAR package is as simple as using the pear install command, for example:

```
$ pear install HTTP_Request2
```

A list of available PEAR packages can be found on the PEAR website. There are many PEAR packages available from other sources as well. For example, the PHPUnit testing framework is available from the pear.phpunit.de PEAR channel. In order to install a package from a source other than pear.php.net you must first discover its channel, for example:

```
$ pear channel-discover pear.phpunit.de
```

Once the channel has been discovered, you can install a package from that channel using the install command, for example:

```
$ pear install phpunit/PHPUnit
```

If you're using PHP 5.3 or later, then you can instead use PEAR2 and Pyrus to install PEAR packages. More information about Pyrus can be found at http://pear2.php.net/.

Composer
While PEAR is a tool for managing system-wide packages, Composer is a tool for managing dependencies on a per-project basis. Declaring dependencies is as simple as adding a JavaScript Object Notation (JSON) formatted file named composer.json to the root of your project. For example, you can indicate that your project requires PHPUnit version 3.7 with the following composer.json file:

```
{
    "require": {
        "phpunit/phpunit": "3.7.*"
    }
}
```

Next, you need to install Composer by downloading it in to the root of your project:

```
$ curl -s https://getcomposer.org/installer | php
```

Once Composer is installed in your project, you can then install dependencies with the install command:

```
$ php composer.phar install
```

You can learn more about Composer by visiting http://getcomposer.org/.

USEFUL ONLINE RESOURCES

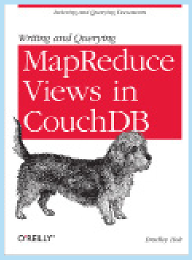
Table 1. Useful Online Resources

Name	URL
The PHP Website	http://php.net/
PHP Zone	http://php.dzone.com/
Zend Developer Zone	http://devzone.zend.com/
php|architect	http://www.phparch.com/
Planet PHP	http://www.planet-php.net/
PHPDeveloper.org	http://phpdeveloper.org/
O'Reilly Media PHP Development and Resources	http://oreilly.com/php/
Highest Voted 'PHP Questions on Stack Overflow	http://stackoverflow.com/questions/tagged/php

ABOUT THE AUTHOR

Bradley Holt is Co-Founder & Technical Director of Found Line (http://foundline.com), a creative studio with capabilities in web development, web design, and print design. He organizes the Burlington, Vermont PHP Users Group and is a co-organizer of Vermont Code Camp (http://vtcodecamp.org) as well as the Northeast PHP Conference (http://nephp.org). He has spoken at SXSW Interactive, OSCON, the jQuery Conference, ZendCon, and CouchConf. He can be found on Twitter as @BradleyHolt.

RECOMMENDED BOOK



If you want to use CouchDB to support real-world applications, you'll need to create MapReduce views that let you query this document-oriented database for meaningful data. With this short and concise ebook, you'll learn how to create a variety of MapReduce views to help you query and aggregate data in CouchDB's large, distributed datasets.

DZone, Inc.
150 Preston Executive Dr.
Suite 201
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome

refcardz@dzone.com

Sponsorship Opportunities

sales@dzone.com

ISBN-13: 978-1-936502-60-8
ISBN-10: 1-936502-60-7
50795
9 781936 502608
$7.95

Version 1.0