

---

# Deploying WordPress with AWS Elastic Beanstalk

Implementation Overview for  
Scalable WordPress-powered  
Websites

*Andreas Chatzakis*

*January 2015*



(Please consult <http://aws.amazon.com/whitepapers> for the latest version of this paper.)

# Contents

Contents	2
Abstract	2
Introduction	2
Implementation Walkthrough	3
Preparation	3
Environment Creation	11
Software Installation	21
WordPress Plugin Installation	23
Application Versioning	29
Auto Scaling Configuration	30
Additional considerations	31

## Abstract

WordPress is an open-source blogging tool and content management system (CMS) based on PHP and MySQL that is used to power anything from personal blogs to high-traffic websites. Amazon Web Services (AWS) provides a reliable, scalable, secure, and highly performing infrastructure for the most demanding applications.

A reference architecture that addresses common scalability and high availability requirements has been outlined in the whitepaper, "[WordPress: Best Practices on AWS](#)". To implement that architecture, you can leverage AWS Elastic Beanstalk—a service that reduces complexity by automatically handling the details of capacity provisioning, load balancing, scaling, and application health monitoring. This whitepaper provides system administrators with an overview of the steps involved.

## Introduction

The first version of WordPress was released in 2003, and as such it was not built with modern elastic and scalable cloud-based infrastructures in mind. Through the work of the WordPress community and the release of various WordPress modules, the capabilities of this CMS solution are constantly expanding. Today it is possible to build a WordPress architecture that takes advantage of many of the benefits of the AWS platform.

Amazon Web Services provides a number of application management services for developers and administrators. Provided at no additional charge, AWS Elastic Beanstalk, AWS OpsWorks, and AWS CloudFormation help you manage your AWS cloud applications with added convenience and control. The following example shows how to use AWS Elastic Beanstalk to deploy a WordPress-powered website with a highly available architecture.

AWS Elastic Beanstalk is an application management service that facilitates quick deployment and management of cloud applications. You simply upload your application, and AWS Elastic Beanstalk automatically handles the details of capacity provisioning, load balancing, Auto Scaling, and application health monitoring.

## Implementation Walkthrough

This section presents a walkthrough of an example installation of WordPress with AWS Elastic Beanstalk. In this example AWS Elastic Beanstalk launches an Elastic Load Balancing load balancer and multiple web servers in separate AWS Availability Zones. It also launches an Amazon Relational Database Service (Amazon RDS) database instance running MySQL. In addition to resources managed via AWS Elastic Beanstalk, this walkthrough also sets up an Amazon Simple Storage Service (Amazon S3) Bucket for static assets, an Amazon CloudFront distribution, and an Amazon ElastiCache cluster running the Memcached engine. Integrating WordPress with the above architecture is accomplished via [W3 Total Cache](#), a third-party open source plugin.

With this plugin you can do the following:

- Store static assets (e.g., media library, theme files, etc) on Amazon S3, thus creating a stateless web tier and offloading this workload from your web servers.
- Serve those assets via Amazon CloudFront, Amazon's content delivery network, thus reducing the latency for users around the globe.
- Use ElastiCache's Memcached engine to perform database caching, thus improving performance and reducing the load on the database.
- Implement browser and CloudFront caching using cache-control, future-expire headers and entity tags (ETag), thus further improving the end user experience.

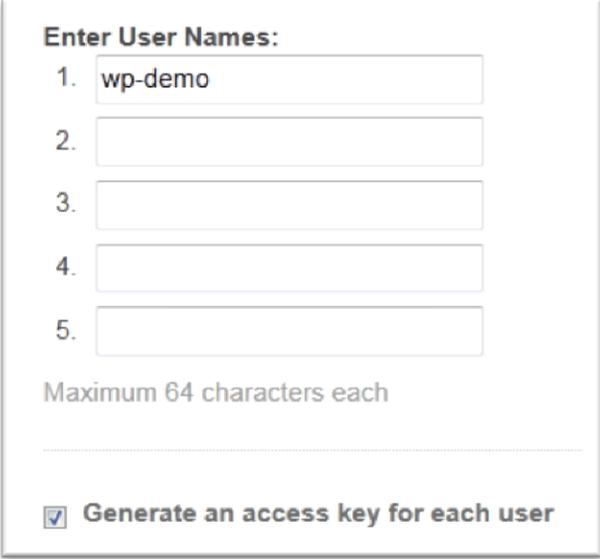
Unless otherwise specified, the links in the steps are to the related AWS service documentation.

### Preparation

1. First, create an AWS Identity and Access Management (IAM) user to be used by the WordPress plugin to access Amazon S3.

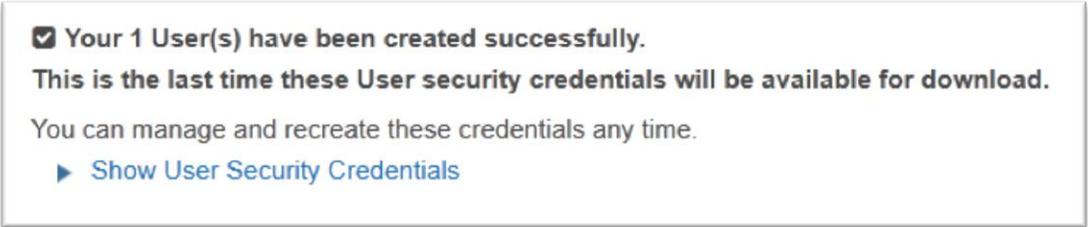
How to: [Creating an IAM User in Your AWS Account](#).<sup>1</sup>

**Note:** IAM roles provide a better way of managing access to AWS resources, but at the time of writing the W3\_Total\_Cache plugin does not support [IAM roles](#).<sup>2</sup>



The screenshot shows a form titled "Enter User Names:" with five numbered input fields. The first field contains the text "wp-demo". Below the fields, it says "Maximum 64 characters each". At the bottom, there is a checked checkbox labeled "Generate an access key for each user".

2. Next, take note of the user security credentials and store them in a secure manner:



The screenshot shows a success message box with a checked checkbox and the text: "Your 1 User(s) have been created successfully. This is the last time these User security credentials will be available for download. You can manage and recreate these credentials any time." Below this is a blue link with a right-pointing triangle icon: "Show User Security Credentials".

3. Now create an Amazon S3 bucket in the region of your choice.

How to: [Creating a Bucket](#).<sup>3</sup>

---

<sup>1</sup> [http://docs.aws.amazon.com/IAM/latest/UserGuide/Using\\_SettingUpUser.html](http://docs.aws.amazon.com/IAM/latest/UserGuide/Using_SettingUpUser.html)

<sup>2</sup> <http://docs.aws.amazon.com/IAM/latest/UserGuide/role-usecase-ec2app.html>

<sup>3</sup> <http://docs.aws.amazon.com/AmazonS3/latest/UG/CreatingaBucket.html>

### Create a Bucket - Select a Bucket Name and Region Cancel

A bucket is a container for objects stored in Amazon S3. When creating a bucket, you can choose a Region to optimize for latency, minimize costs, or address regulatory requirements. For more information regarding bucket naming conventions, please visit the [Amazon S3 documentation](#).

**Bucket Name:**

**Region:**

[Set Up Logging >](#) [Create](#) [Cancel](#)

4. Upload an `index.html` page. Amazon S3 returns this index document when requests are made to the root domain or any of the subfolders.

How to: [Uploading Objects into Amazon S3](#).<sup>4</sup>

5. Enable static website hosting for that bucket.

How to: [Configure a Bucket for Website Hosting](#).<sup>5</sup>

---

<sup>4</sup> <http://docs.aws.amazon.com/AmazonS3/latest/UG/UploadingObjectsintoAmazonS3.html>

<sup>5</sup> <http://docs.aws.amazon.com/AmazonS3/latest/dev/HowDoIWebsiteConfiguration.html>

## Bucket: wp-demo ✕

---

**Bucket:** wp-demo  
**Region:** Ireland  
**Creation Date:** Mon Dec 01 10:11:20 GMT+000 2014  
**Owner:** Me

---

▶ Permissions

---

▼ Static Website Hosting

---

You can [host your static website](#) entirely on Amazon S3. Once you enable your bucket for static website hosting, all your content is accessible to web browsers via the Amazon S3 website endpoint for your bucket.

**Endpoint:** [wp-demo.s3-website-eu-west-1.amazonaws.com](http://wp-demo.s3-website-eu-west-1.amazonaws.com)

Each bucket serves a website namespace (e.g. "www.example.com"). Requests for your host name (e.g. "example.com" or "www.example.com") can be routed to the contents in your bucket. You can also redirect requests to another host name (e.g. redirect "example.com" to "www.example.com"). See our [walkthrough](#) for how to set up an Amazon S3 static website with your host name.

Do not enable website hosting

Enable website hosting

**Index Document:**

**Error Document:**

▶ **Edit Redirection Rules:** You can set custom rules to automatically redirect web page requests for specific content.

6. Attach an IAM policy to the IAM user created previously to allow access to the specific bucket.

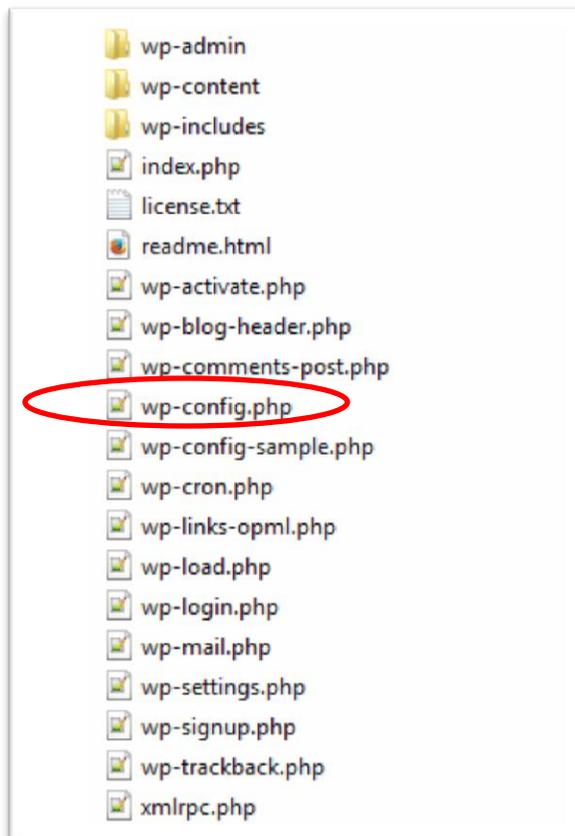
How to: [Managing IAM Policies](#).<sup>6</sup>

```
{
  "Version": "2014-12-10",
  "Statement": [
    {
      "Sid": "Stmt1389783689000",
      "Effect": "Allow",
```

<sup>6</sup> <http://docs.aws.amazon.com/IAM/latest/UserGuide/ManagingPolicies.html>

```
"Action": [  
  "s3:DeleteObject",  
  "s3:GetObject",  
  "s3:GetObjectAcl",  
  "s3:ListBucket",  
  "s3:PutObject",  
  "s3:PutObjectAcl"  
],  
"Resource": [  
  "arn:aws:s3:::wp-demo",  
  "arn:aws:s3:::wp-demo/*"  
]  
}  
]
```

7. Download the WordPress application code to create your initial application code bundle (in this case `wordpress-4.0.1.zip` from <http://wordpress.org/download/>).
8. Follow the included instructions to unzip this to your local file system. Create a copy of `wp-config-sample.php` and rename it `wp-config.php`.



9. Edit the `wp-config.php` file, by configuring database connectivity information with the use of [Amazon RDS environment variables](#)<sup>7</sup> that AWS Elastic Beanstalk maintains automatically for you<sup>8</sup>. In addition you can add other environment variables (e.g., for the authentication unique keys and salts). To do that, you can take advantage of AWS Elastic Beanstalk's ability to customize environment variables as explained in [Customizing and Configuring a PHP Environment](#).<sup>9</sup>

For more details on the WordPress configuration, you can visit [http://codex.wordpress.org/Editing\\_wp-config.php](http://codex.wordpress.org/Editing_wp-config.php) or see the example below.

```
<?php

/** Detect if SSL is used. This is required since we are
terminating SSL either on CloudFront or on ELB */
if (($_SERVER['HTTP_CLOUDFRONT_FORWARDED_PROTO'] ==
'https') OR ($_SERVER['HTTP_X_FORWARDED_PROTO'] ==
'https'))
    {$_SERVER['HTTPS']='on';}

/** The name of the database for WordPress */
define('DB_NAME', $_SERVER["RDS_DB_NAME"]);
/** MySQL database username */
define('DB_USER', $_SERVER["RDS_USERNAME"]);
/** MySQL database password */
define('DB_PASSWORD', $_SERVER["RDS_PASSWORD"]);
/** MySQL hostname */
define('DB_HOST', $_SERVER["RDS_HOSTNAME"]);
/** Database Charset to use in creating database tables. */
define('DB_CHARSET', 'utf8');
/** The Database Collate type. Don't change this if in
doubt. */
define('DB_COLLATE', '');

/**#@+
 * Authentication Unique Keys and Salts.
 * Change these to different unique phrases!
 */
define('AUTH_KEY',          $_SERVER["AUTH_KEY"]);
define('SECURE_AUTH_KEY',   $_SERVER["SECURE_AUTH_KEY"]);
define('LOGGED_IN_KEY',     $_SERVER["LOGGED_IN_KEY"]);
define('NONCE_KEY',         $_SERVER["NONCE_KEY"]);
define('AUTH_SALT',         $_SERVER["AUTH_SALT"]);
define('SECURE_AUTH_SALT',  $_SERVER["SECURE_AUTH_SALT"]);
define('LOGGED_IN_SALT',    $_SERVER["LOGGED_IN_SALT"]);
define('NONCE_SALT',        $_SERVER["NONCE_SALT"]);
```

---

<sup>7</sup> [http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create\\_deploy\\_PHP.rds.html](http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_PHP.rds.html)

<sup>8</sup> This is assuming you create an RDS instance as part of the Elastic Beanstalk Environment. In step 13 we will explain why you might prefer not to do that in a production environment.

<sup>9</sup> [http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create\\_deploy\\_PHP\\_custom\\_container.html](http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_PHP_custom_container.html)

```

/**#@-*/

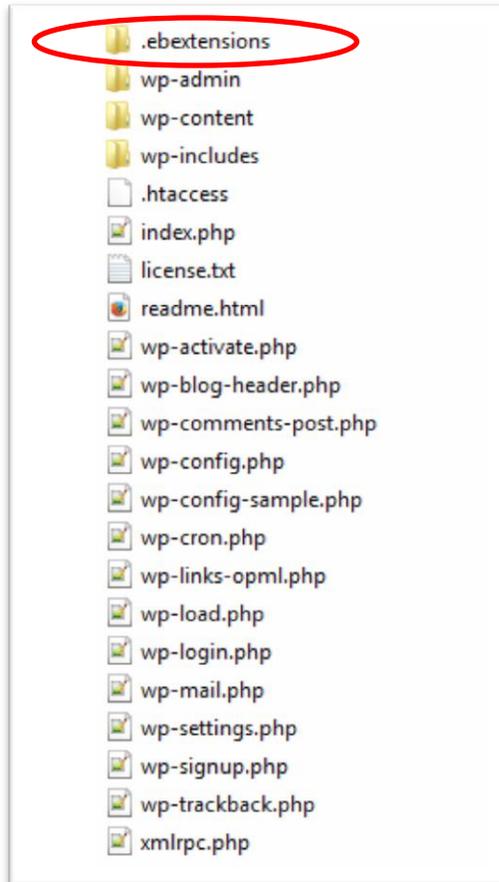
/**
 * WordPress Database Table prefix.
 */
$table_prefix = 'wp_';
/**
 * WordPress Localized Language, defaults to English.
 */
define('WPLANG', '');

/**
 * For developers: WordPress debugging mode.
 *
 * Change this to true to enable the display of notices
 during development.
 */
define('WP_DEBUG', false);
/** Absolute path to the WordPress directory. */
if ( !defined('ABSPATH') )
define('ABSPATH', dirname(__FILE__) . '/');
/** Sets up WordPress vars and included files. */
require_once(ABSPATH . 'wp-settings.php');

```

10. Now set the values for the Authentication Unique Keys and Salts environment variables. You could do this via the AWS console but instead—in our example—we create a folder called `.ebextensions` in the root folder of the local copy of the WordPress application code.

**Tip:** If you are using Windows Explorer, you need to name the folder `.ebextensions.` in order to create a folder with a dot prefix. The last dot will be removed automatically.



Inside the `.ebextensions` folder, add a file called `keys.config` with the following content. Make sure you replace the authentication unique keys and salts with different unique phrases, as this is for illustration only.

```
option_settings:
- option_name: AUTH_KEY
  value: 'AJDSAHCMEWKRSODJFIEWRJSDFMKSDMADS '
- option_name: SECURE_AUTH_KEY
  value: 'SAJDSANCMEWKRSODJFIEWRJSDFMKSDMAD '
- option_name: LOGGED_IN_KEY
  value: 'DSAJDSADCMEWKRSODJFIEWRJSDFMKSDMA '
- option_name: NONCE_KEY
  value: 'ADSAJDSAKMEWKRSODJFIEWRJSDFMKSDM '
- option_name: SECURE_AUTH_SALT
  value: 'MADSAJDSAKMEWKRSODJFIEWRJSDFMKSD '
- option_name: LOGGED_IN_SALT
  value: 'DMADSAJDSAKMEPKRSODJFIEWRJSDFMKSD '
- option_name: NONCE_SALT
  value: 'SDMADSAJDSAKMEWKHSODJFIEWRJSDFMK '
- option_name: AUTH_SALT
```

```
value: 'KSDMADSAJDSAKCMEWKRSODJJIEWRJSDFM'
```

## Environment Creation

11. Next, create an AWS Elastic Beanstalk environment. Specify a **Web Server** tier with the default **PHP** container and of the type **Load balancing, autoscaling**.

How to: [Launching New Environments](#).<sup>10</sup>

The screenshot shows the 'Environment Type' configuration page in the AWS console. It includes a checkbox for 'Launch a new environment running this application' which is checked. Below this are three dropdown menus: 'Environment tier' set to 'Web Server', 'Predefined configuration' set to 'PHP', and 'Environment type' set to 'Load balancing, autoscaling'. Each dropdown has a 'Learn more' link next to it. Text below the dropdowns explains that Elastic Beanstalk will create a Web Server 1.0 environment, an environment running PHP 5.5 on 64bit Amazon Linux 2014.09 v1.0.9, and provides a 'Change Defaults' link.

Zip the application code making sure you include the files and subfolders themselves, rather than zipping the parent folder - e.g. there should be no top-level directory (subdirectories are fine). Then upload the zip file as the source of the initial application version:

The screenshot shows the 'Application Version' configuration page. It asks to 'Select a source for your application version.' There are three radio button options: 'Sample application', 'Upload your own (Learn more)', and 'S3 URL'. The 'Upload your own' option is selected. Below it is a 'Browse...' button followed by the text 'wordpress-4-0-1-deployment-01.zip'. The 'S3 URL' option is unselected, and there is a text input field with a placeholder '(e.g. https://s3.amazonaws.com/s3Bucket/s3Key)'.

12. Type a unique environment name and URL:

---

<sup>10</sup> <http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.environments.html>

### Environment Information

Enter your environment information. [Learn more.](#)

Environment name:

Environment URL:

Description:  Optional: 200 character maximum

13. You can now enable the option to create an Amazon RDS MySQL instance as part of your Elastic Beanstalk environment. You will configure this in a subsequent step. Please note that in a production environment it is instead recommended to create the RDS instance separately – not via AWS Elastic Beanstalk but through the Amazon RDS console, CLI, or API. This would allow you to take advantage of Elastic Beanstalk’s more advanced features such as [zero downtime deployments](#)<sup>11</sup>. For this example we also did not select the ‘Create this environment inside a VPC’ option<sup>12</sup> which is another recommended configuration for production environments.

### Additional Resources

Select additional resources for this environment.

Create an RDS DB Instance with this environment [Learn more](#)

Create this environment inside a VPC [Learn more](#)

In the next page, select an instance type for your web servers and specify an existing Amazon Elastic Compute Cloud (Amazon EC2) key pair to be able to login to the instances of the environment via Secure Shell (SSH). If you have no Amazon EC2 key pairs configured, you need to create one, as explained in the AWS [documentation](#).<sup>13</sup> You should also set up a simple health check for the Elastic Load Balancing load balancer by configuring an application health check URL. In our example this will be a simple static file from the root folder of the application

---

<sup>11</sup> <http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.CNAMESwap.html>

<sup>12</sup> In practice if you are launching in a region where your account does not support the EC2-Classical platform and you don't tick the 'Create this environment inside a VPC' option, your resources will be deployed in your Default VPC. You can read more at <http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/default-vpc.html>

<sup>13</sup> <http://docs.aws.amazon.com/gettingstarted/latest/wah/getting-started-prereq.html#create-a-key-pair>

(HTTP:80/readme.html) that will be used to confirm Apache web server is running and responding to HTTP requests.

**Configuration Details**

Modify the following settings or click Next to accept the default configuration. [Learn more.](#)

Instance type:    
 Determines the processing power of the servers in your environment.

EC2 key pair:  [Refresh](#)   
 Optional: Enables remote login to your instances.

Email address:    
 Optional: Get notified about any major changes to your environment.

Application health check URL:    
 Enter the relative URL that ELB continually monitors to ensure your application is available.

Enable rolling updates:  Lets you control how changes to the environment's instances are propagated. [Learn more.](#)

Cross zone load balancing:  Enables load balancing across multiple Availability Zones. [Learn more.](#)

Connection draining:    
 Enables the load balancer to maintain connections to an Amazon EC2 instance to complete in-progress requests while also stopping new requests. [Learn more.](#)

Connection draining timeout:     
 Maximum time that the load balancer maintains connections to an Amazon EC2 instance before forcibly closing connections.

14. When you create a new environment, you can also specify tags to categorize the environment. Tags can help you identify environments in cost allocation reports, especially if you have many to manage. For example, you could create a key-value pair for the website name so that this appears in cost reports:

**Environment Tags**

You can specify tags (key-value pairs) for your Environment. You can add up to 7 unique key-value pairs for each Environment.

	Key (128 characters maximum)	Value (256 characters maximum)
1.	<input type="text" value="website"/>	<input type="text" value="example.com"/> <a href="#">✕</a>
2.	<input type="text"/>	<input type="text"/>

15. Use the next page to configure the Amazon RDS database. Since this is just a test environment, opt for the **Single Availability Zone** deployment. For actual production workloads, the **Multiple Availability Zones** option is recommended.

### RDS Configuration

Specify your RDS settings. [Learn more.](#)

Snapshot:  Refresh

DB engine:  Refresh

Instance class:  Refresh

Allocated storage:  GB  
 You must specify a value from 5 GB to 1024 GB.

Username:

Password:

Retention setting:   
 Terminating your environment can permanently delete your Amazon RDS DB instance and all its data. By default, AWS Elastic Beanstalk saves a snapshot, which preserves your data but may incur backup storage charges. [Learn more.](#)

Availability:

In a few minutes, your AWS Elastic Beanstalk environment is up and running:

### Overview

Refresh



**Health**  
Green

Monitor

**Running Version**  
First Release

Upload and Deploy



**Configuration**  
64bit Amazon Linux 2014.03  
v1.0.4 running PHP 5.5

Edit

Recent Events Show All

Time	Type	Details
2014-07-14 16:43:46 UTC+0100	INFO	Adding instance 'i-88102a2f' to your environment.
2014-07-14 16:43:10 UTC+0100	INFO	Successfully launched environment: <a href="#">wordpress20140714-001</a>
2014-07-14 16:43:09 UTC+0100	INFO	Application available at <a href="#">wordpress20140714-001.elasticbeanstalk.com</a> .
2014-07-14 16:36:23 UTC+0100	INFO	Waiting for EC2 instances to launch. This may take a few minutes.
2014-07-14 16:34:59 UTC+0100	INFO	Created RDS database named: <a href="#">wordpress20140714</a>

Click **Show All** and note the security group name information from the events list: This will be required in a subsequent step.

- The next step involves the creation of an ElastiCache Memcached cluster. Before launching the cluster, we first need to create a security group to control access to the Memcached instances.

### Create Security Group

Security group name ⓘ

Description ⓘ

VPC ⓘ  \*  
\* denotes default VPC

Please note that if you are running your ElastiCache nodes in Amazon VPC (including [Default VPC](#)<sup>14</sup> as in our example), you control access to your cache clusters with Amazon VPC security groups. For more information on using ElastiCache in an Amazon VPC, see [ElastiCache and Amazon Virtual Private Cloud](#)<sup>15</sup> and [Using ElastiCache with Amazon Virtual Private Cloud \(VPC\)](#)<sup>16</sup>. Instead if you are running your ElastiCache nodes in EC2-Classic you will need to create [ElastiCache security groups](#).<sup>17</sup>

17. You can now authorize the security group of your web servers (from step 15) by adding an inbound rule to the security group you created for your ElastiCache cluster. VPC Security groups are explained in the [Amazon Virtual Private Cloud User Guide](#).<sup>18</sup>

### Edit inbound rules

Type ⓘ	Protocol ⓘ	Port Range ⓘ	Source ⓘ
Custom TCP Rule	TCP	11211	Custom IP sg-... <input type="text"/>

18. You are now ready to launch the ElastiCache cluster with the Memcached engine, which in this example consists of two nodes.

How to: [Create a Cache Cluster](#).<sup>19</sup>

<sup>14</sup> <http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/default-vpc.html>

<sup>15</sup> <http://docs.aws.amazon.com/AmazonElastiCache/latest/UserGuide/ElastiCacheAndVPC.html>

<sup>16</sup> <http://docs.aws.amazon.com/AmazonElastiCache/latest/UserGuide/ManagingVPC.html>

<sup>17</sup> <http://docs.aws.amazon.com/AmazonElastiCache/latest/UserGuide/CacheSecurityGroup.html>

<sup>18</sup> [http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC\\_SecurityGroups.html](http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_SecurityGroups.html)

<sup>19</sup> <http://docs.aws.amazon.com/AmazonElastiCache/latest/UserGuide/GettingStarted.CreateCluster.html>

### Specify Cluster Details

---

**Cluster Specifications**

Engine Memcached ⓘ

Engine Version 1.4.14 ⓘ

Port\* 11211 ⓘ

Parameter Group default.memcached1.4 ⓘ

---

**Configuration**

Cluster Name\* wp-demo ⓘ

Node Type cache.m3.medium (2.78 GB ... ⓘ

Number of Nodes\* 2 ⓘ

19. In the second step, specify that you want to spread the nodes across Availability Zones and select the security group you created earlier:

### Configure Advanced Settings

---

**Network & Security**

Cache Subnet Group default ⓘ

Availability Zone(s) Spread Nodes Across Zones ⓘ

VPC Security Group(s)  ⓘ

---

**Maintenance**

Maintenance Window  Select Window  No Preference ⓘ

Topic for SNS Notification\* Disable Notifications ⓘ [Manual ARN input ⓘ](#)

---

\*Required

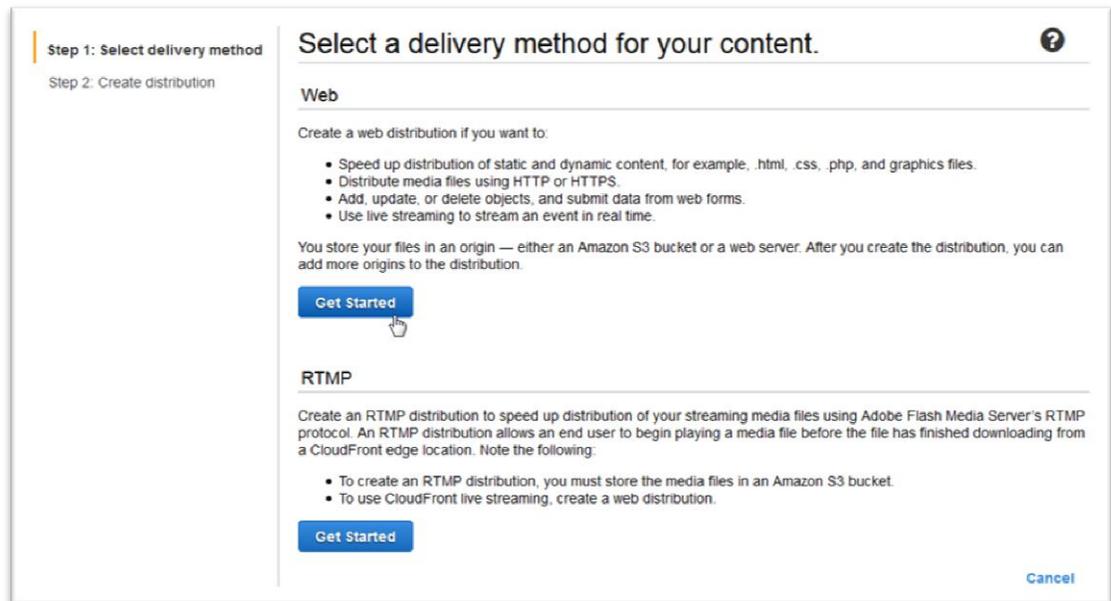
[Cancel](#) [Previous](#) [Next](#)

Please note that this example deploys the cluster into the Default VPC and uses the default [Cache Subnet Group](#)<sup>20</sup>. In addition we did not specify an Amazon Simple Notification Service (Amazon SNS) notification topic. For a production environment, it is best to configure those.

Within a few minutes, the ElastiCache cluster is up and running. Keep a note of the endpoints of the ElastiCache nodes as we are going to need them in a subsequent step.

20. Now create a CloudFront web distribution.

How to: [Task List for Creating a Web Distribution](#).<sup>21</sup>



21. Set up your distribution so that by default it uses the Environment URL of the AWS Elastic Beanstalk environment (from step 12) as its origin:

---

<sup>20</sup> <http://docs.aws.amazon.com/AmazonElastiCache/latest/UserGuide/CacheSubnetGroups.html>

<sup>21</sup> <http://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/distribution-web-creating.html>

## Create Distribution

### Origin Settings

Origin Domain Name	<input type="text" value="wp-demo-env.elasticbeanstalk.com"/>	
Origin Path	<input type="text"/>	
Origin ID	<input type="text" value="Custom-wp-demo-env.elasticbeanstalk.c"/>	
Origin Protocol Policy	<input checked="" type="radio"/> HTTP Only <input type="radio"/> Match Viewer	
HTTP Port	<input type="text" value="80"/>	
HTTPS Port	<input type="text" value="443"/>	

In a production environment you might want to set the Origin Protocol Policy to **Match Viewer** protocol so that if the viewer connects to CloudFront using HTTPS, CloudFront will connect to your origin using HTTPS as well, achieving end-to-end encryption. This requires that you install a trusted SSL certificate on the load balancer as explained in the [AWS Elastic Beanstalk Developer Guide](#).<sup>22</sup>

22. Because the default cache behavior associated with this origin will serve the dynamic content of the front end, use the following configuration:
  - a. Allow all HTTP methods since the dynamic portions of the website require both GET and POST requests (e.g., to support POST for the comment submission forms).
  - b. Forward only the cookies that vary the WordPress output—e.g., `wordpress_*`, `wp-settings-*` and `comment_*`. You will need to extend that list if you have installed any plugins that depend on other cookies not in the list.
  - c. Forward only the HTTP headers that affect the output of WordPress. This example enables the **CloudFront-Forwarded-Proto** header (so that the same page is cached separately if accessed via HTTP or HTTPS) and the three device-detection headers of CloudFront (**CloudFront-Is-Desktop-Viewer**, **CloudFront-Is-Mobile-Viewer**, **CloudFront-Is-Tablet-Viewer**) that you can use to customize the HTML output of your themes based on the end user's device type.
  - d. **Forward Query Strings** as WordPress relies on those.

---

<sup>22</sup> <http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.managing.elb.html>

### Default Cache Behavior Settings

Path Pattern	Default (*)	<a href="#">i</a>
Viewer Protocol Policy	<input checked="" type="radio"/> HTTP and HTTPS <input type="radio"/> Redirect HTTP to HTTPS <input type="radio"/> HTTPS Only	<a href="#">i</a>
Allowed HTTP Methods	<input type="radio"/> GET, HEAD <input type="radio"/> GET, HEAD, OPTIONS <input checked="" type="radio"/> GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE	<a href="#">i</a>
Cached HTTP Methods	<input checked="" type="checkbox"/> GET, HEAD (Cached by default) <input type="checkbox"/> OPTIONS	<a href="#">i</a>
Forward Headers	Whitelist ▾	<a href="#">i</a>
Whitelist Headers	<div style="border: 1px solid #ccc; padding: 5px;"> <input type="text" value="Filter headers or enter a custom header"/> <span style="float: right; border: 1px solid #ccc; padding: 2px 5px;">Add Custom &gt;&gt;</span> </div> <div style="display: flex; justify-content: space-between; align-items: flex-start;"> <div style="border: 1px solid #ccc; padding: 5px; width: 60%;">           Accept            Accept-Charset            Accept-Datetime            Accept-Language            Authorization            CloudFront-Viewer-Country         </div> <div style="width: 35%; text-align: center;"> <span>Add &gt;&gt;</span>  <span>&lt;&lt; Remove</span> </div> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 5px; width: 60%;">           4 header(s) whitelisted            CloudFront-Forwarded-Proto            CloudFront-Is-Desktop-Viewer            CloudFront-Is-Mobile-Viewer            CloudFront-Is-Tablet-Viewer         </div>	<a href="#">i</a>
Object Caching	<input checked="" type="radio"/> Use Origin Cache Headers <input type="radio"/> Customize	<a href="#">i</a>
Minimum TTL	<input type="text" value="0"/>	<a href="#">i</a>
Forward Cookies	Whitelist ▾	<a href="#">i</a>
Whitelist Cookies	<div style="border: 1px solid #ccc; padding: 5px;">           wordpress_.*            wp-settings-.*            comment_.*         </div>	<a href="#">i</a>
Forward Query Strings	<input checked="" type="radio"/> Yes <input type="radio"/> No (Improves Caching)	<a href="#">i</a>

23. Now create two more cache behaviors for dynamic content, one for the login page (path pattern: `wp-login.php`) and one for the admin folder (path pattern: `wp-admin/*`). Those two behaviors have the same exact settings. For example, for both behaviors you enforce the use of HTTPS and whitelist all cookies and HTTP headers. The reason is that this section of the website is highly personalized and typically has just a few users; caching efficiency is not a primary concern here. You can keep configuration simple and ensure maximum compatibility with any installed plugins by just passing all cookies and headers back to the origin.

## Cache Behavior Settings

Path Pattern



Origin



Viewer Protocol Policy  HTTP and HTTPS  
 Redirect HTTP to HTTPS  
 HTTPS Only



Allowed HTTP Methods  GET, HEAD  
 GET, HEAD, OPTIONS  
 GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE



Cached HTTP Methods   
 OPTIONS



Forward Headers



Object Caching  Use Origin Cache Headers  
 Customize



When you choose to forward all headers to the origin, CloudFront does not cache your objects. In that configuration, Minimum TTL must be 0 seconds.

Minimum TTL



Forward Cookies



Forward Query Strings  Yes  
 No (Improves Caching)



## Software Installation

24. Once the CloudFront changes are deployed, point your browser to the host name of the CloudFront distribution,<sup>23</sup> and follow the installation process:

### Welcome

---

Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

### Information needed

---

Please provide the following information. Don't worry, you can always change these settings later.

**Site Title**

**Username**   
Usernames can have only alphanumeric characters, spaces, underscores, hyphens, periods and the @ symbol.

**Password, twice**   
A password will be automatically generated for you if you leave this blank.

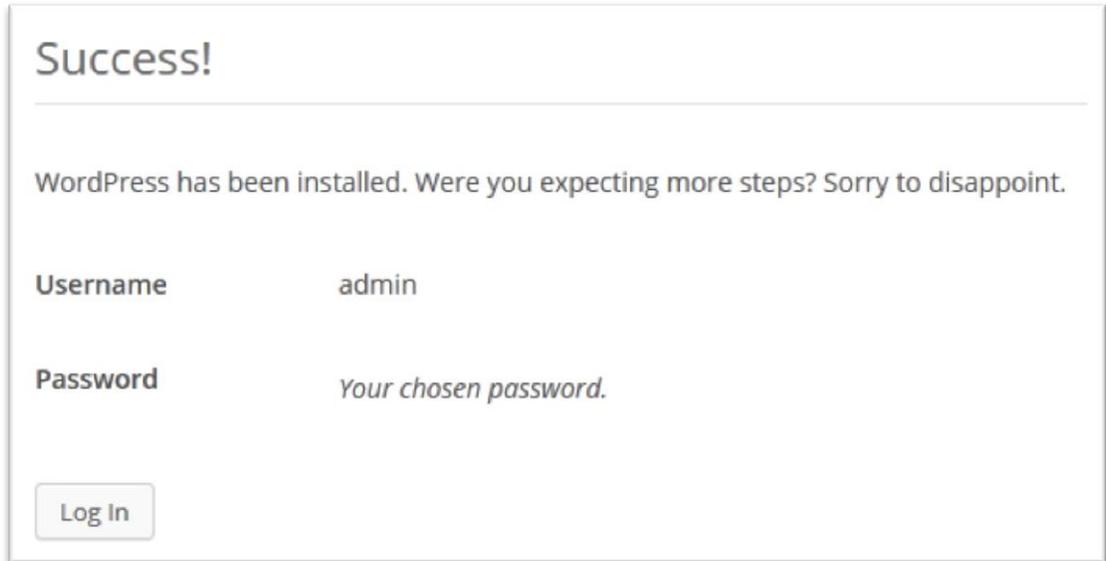
**Strong**

Hint: The password should be at least seven characters long. To make it stronger, use upper and lower case letters, numbers, and symbols like ! " ? \$ % ^ & ).

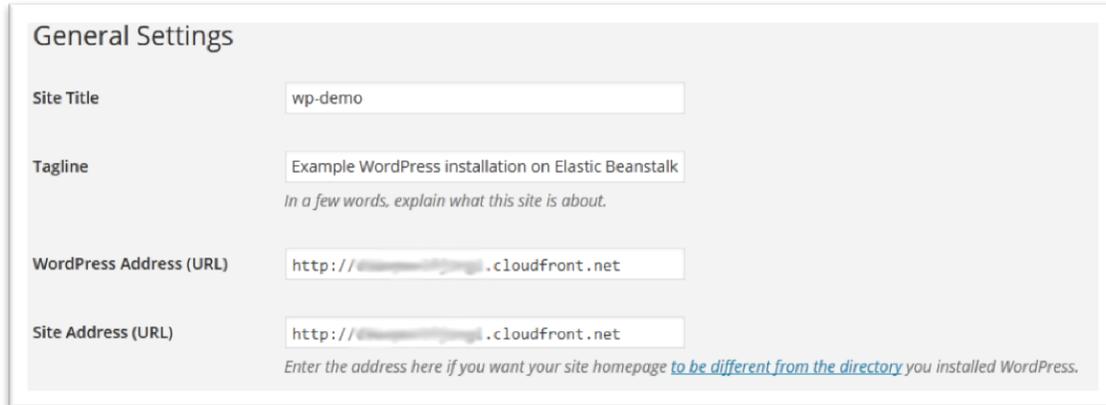
25. Submit this form to initialize the database content. The website is now functional.

---

<sup>23</sup> In a real environment, you would instead use Amazon Route 53 to point to the actual domain name of the website you are about to install.

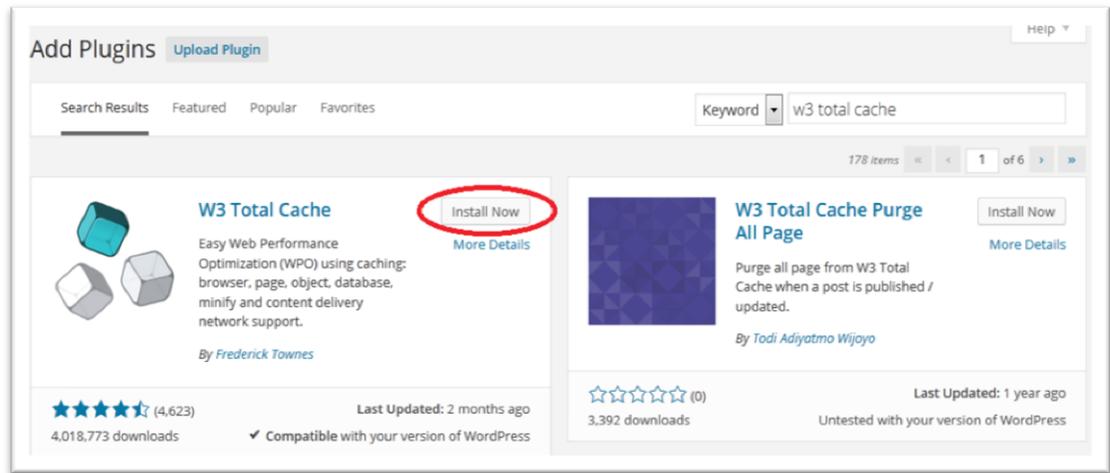


26. The WordPress installation script sets the **WordPress Address (URL)** and **Site Address (URL)** settings to the host name of the Elastic Load Balancing load balancer. To correct this, visit the WordPress **General Settings** tab and set it to the right domain name for your web site (in this example simply the host name of the CloudFront distribution).



## WordPress Plugin Installation

27. Now log in to the WordPress admin panel and install the W3 Total Cache plugin via the user interface:



After the plugin is installed you will need to activate it:



28. Next, start enabling the W3 Total Cache plugin features. A detailed description of all settings of the plugin is beyond the scope of this article. Please refer to the W3 Total Cache plugin page at wordpress.org. In this example you first enable the **Database Cache** feature from the Plugin's General Settings.

**Database Cache**

Enable database caching to reduce post, page and feed creation time.

Database Cache:  **Enable**  
*Caching database objects decreases the response time of your site. Best used if object caching is not possible.*

Database Cache Method:

29. Then go to the Database Caching section of the plugin configuration to define the ElastiCache nodes host names (from step 19).

Database caching via memcached is currently **enabled**.

To rebuild the database cache use the  operation.

**General**

**Don't cache queries for logged in users**  
*Enabling this option is recommended to maintain default WordPress behavior.*

**Advanced**

Memcached hostname:port / IP:port:   
  
*Multiple servers may be used and seperated by a comma; e.g. 192.168.1.100:11211, domain.com:22122*

Maximum lifetime of cache objects:  seconds  
*Determines the natural expiration time of unchanged cache items. The higher the value, the larger the cache.*

Garbage collection interval:  seconds  
*If caching to disk, specify how frequently expired cache data is removed. For busy sites, a lower value is best.*

30. Now visit the Browser Cache section of the plugin's configuration and enable the expires, cache control, and entity tag headers. Also activate the **Prevent caching of objects after settings change** option so that a new query string will be generated and appended to objects when the policy changes.

**General**

Specify global browser cache policy.

- Set Last-Modified header**  
*Set the Last-Modified header to enable 304 Not Modified response.*
- Set expires header**  
*Set the expires header to encourage browser caching of files.*
- Set cache control header**  
*Set pragma and cache-control headers to encourage browser caching of files.*
- Set entity tag (eTag)**  
*Set the Etag header to encourage browser caching of files.*
- Set W3 Total Cache header**  
*Set this header to assist in identifying optimized files.*
- Enable HTTP (gzip) compression**  
*Reduce the download time for text-based files.*
- Prevent caching of objects after settings change**  
*Whenever settings are changed, a new query string will be generated and appended to objects allowing the new policy to be applied.*

Next, visit the CDN settings section of the plugin's General Configuration. Enable CDN and select CloudFront as an **Origin Push CDN** configuration (this will have Amazon S3 as its origin). Then visit the CDN section to define the details<sup>24</sup>:

---

<sup>24</sup> Please note that W3 Total Cache has a 'Test S3 upload & CloudFront distribution' button. That test will fail unless you also add the s3:ListAllMyBuckets action access right to the IAM user. Since this privilege is not required for the plugin to function correctly, we have not included it in our example.

Configuration

We recommend that you use [IAM](#) to create a new policy for AWS services that have limited permissions. A helpful tool: [AWS Policy Generator](#)

Access key ID:

Secret key:

Bucket:

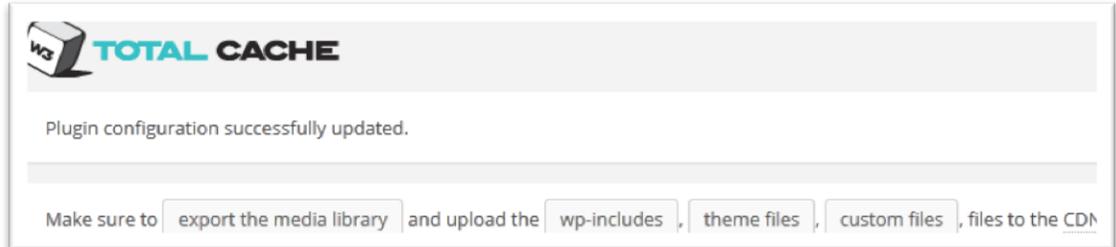
SSL support:    
Some [CDN](#) providers may or may not support [SSL](#), contact your vendor for more information.

Replace site's hostname with: .cloudfront.net or CNAME:

1.

If you have already added a [CNAME](#) to your [DNS](#) Zone, enter it here.

31. You then need to export the media library and upload the wp-includes, theme files, and custom files to Amazon S3 via the plugin's functions:



32. Now that the static files are stored on Amazon S3, go back to the Amazon CloudFront configuration in the CloudFront console and configure Amazon S3 as the origin for static content. To do that, add a second origin pointing to the Amazon S3 bucket used for that purpose:

## Create Origin

---

### Origin Settings

---

Origin Domain Name  ⓘ

Origin ID  ⓘ

Restrict Bucket Access  Yes ⓘ  
 No

33. Then create two more cache behaviors, one for each of the two folders (`wp-content` and `wp-includes`) that should use the Amazon S3 origin rather than the default Elastic Load Balancing origin.

	Precedence	Path Pattern	Origin	Viewer Protocol Policy	Forwarded Query Strings
<input type="checkbox"/>	1	wp-admin/*	Elastic Load Balancing (dynamic)	Redirect HTTP to HTTPS	Yes
<input type="checkbox"/>	2	wp-login.php	Elastic Load Balancing (dynamic)	Redirect HTTP to HTTPS	Yes
<input type="checkbox"/>	3	wp-includes/*	S3 (static content)	HTTP and HTTPS	Yes
<input type="checkbox"/>	4	wp-content/*	S3 (static content)	HTTP and HTTPS	Yes
<input type="checkbox"/>	5	Default (*)	Elastic Load Balancing (dynamic)	HTTP and HTTPS	Yes

Configure both in the same manner:

- Serve GET HTTP requests only.
- Amazon S3 does not vary its output based on cookies or HTTP headers, so you can improve caching efficiency by not forwarding them to the origin via CloudFront.
- Despite the fact that these behaviors serve only static content (which accepts no parameters), you will forward query strings to the origin. This is so that you can use query strings as version identifiers to instantly invalidate, for example, older CSS files when deploying new versions. For more information, see the [Amazon CloudFront Developer Guide](http://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/ReplacingObjects.html).<sup>25</sup>

---

<sup>25</sup> <http://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/ReplacingObjects.html>

# Create Behavior

## Cache Behavior Settings

**Path Pattern**  

**Origin**  

**Viewer Protocol Policy**  HTTP and HTTPS   
 Redirect HTTP to HTTPS  
 HTTPS Only

**Allowed HTTP Methods**  GET, HEAD   
 GET, HEAD, OPTIONS  
 GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE

**Cached HTTP Methods** GET, HEAD (Cached by default) 

**Forward Headers**  

**Object Caching**  Use Origin Cache Headers   
 Customize

**Minimum TTL**  

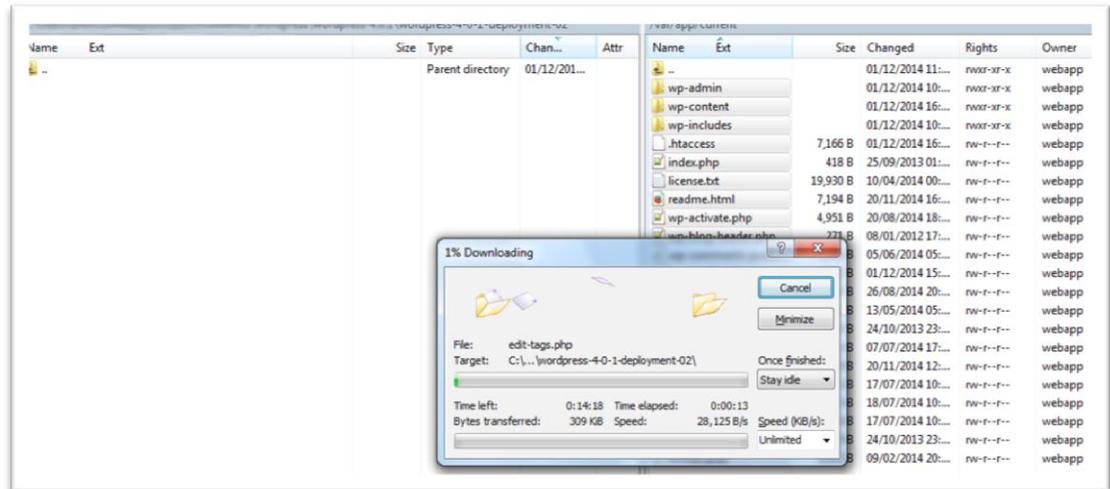
**Forward Cookies**  

**Forward Query Strings**  Yes   
 No (Improves Caching)

## Application Versioning

34. During installation and configuration of the W3 Total Cache plugin, certain changes have been performed on the file system. For example the plugin code has been downloaded and certain configuration files have been created. Those changes (that you implement after your initial AWS Elastic Beanstalk deployment) will be lost if the instance is terminated. They will also not be applied to any new instances added later (e.g., Auto Scaling).

To avoid this, as a first step you can use secure copy (SCP) to download the web folder from the staging environment (folder `/var/app/current/`) locally.



Then create a new `.zip` file (remember to also include the `.ebextensions` folder) and deploy it as a new application version. Please note that this now includes hard-coded values (e.g., the host names of the ElastiCache nodes in the W3 Total Cache configuration) and as such is environment specific.

Upload and Deploy
✕

**i** To deploy a previous version, go to the [Application Versions page](#).

Upload application: Browse... wordpress-4-0-1-deployment-02.zip

Version label: wordpress-4-0-1-deployment-02

### Deployment Limits

Elastic Beanstalk will deploy to **30%** of instances in your autoscaling group at a time.  
Current number of instances: **2**

Batch size:  Percentage

▲▼
%
of instances at a time

Fixed

▲▼
instances at a time (max: 10)

Cancel
Deploy

## Auto Scaling Configuration

35. As the last step you now need to set up the Auto Scaling options. In the Auto Scaling configuration page for your environment in the AWS Elastic Beanstalk console, set up a minimum of 2 and a maximum of 10 nodes across Availability Zones.

### Auto Scaling

Use the following settings to control auto scaling behavior. [Learn more](#).

Minimum instance count:  Minimum number of instances to run.

Maximum instance count:  Maximum number of instances to run. Must be less than 10000.

Availability Zones: Any ▼ Number of Availability Zones to run in.

Custom Availability Zones: eu-west-1a  
eu-west-1b  
eu-west-1c ▲▼ Specific Availability Zones to launch instances in.

Scaling cooldown (seconds):  The amount of time after a scaling activity before any further trigger-related scaling activities can occur.

On the same page, configure a scaling policy that will be triggered based on the CPU Utilization metrics:

**Scaling Trigger**

Trigger measurement: CPUUtilization  The measure name associated with the metric the trigger uses.

Trigger statistic: Average  The statistic that the trigger uses when fetching metrics statistics to examine.

Unit of measurement: Percent  The standard unit that the trigger uses when fetching metric statistics to examine.

Measurement period (minutes): 1  The period between metric evaluations.

Breach duration (minutes): 5  The amount of time used to determine the existence of a breach. The service looks at data between the current time and the number of minutes specified to see if a breach has occurred.

Upper threshold: 60  The upper limit for the metric. If the data points in the last breach duration exceed the threshold, the trigger is activated.

Upper breach scale increment: 1  The incremental amount to use when performing scaling activities when the upper threshold has been breached. Must be an integer, optionally followed by a % sign.

Lower threshold: 15  The lower limit for the metric. If all the data exceeded this threshold during a breach duration, the trigger is activated.

Lower breach scale increment: -1  The incremental amount to use when performing scaling activities when the lower threshold has been breached. Must be an integer optionally followed by a % sign.

## Additional considerations

The above example is a starting point implementation. For a production environment there are additional considerations around security, backups, operations etc. For further reading we recommend going through the following material in the [AWS whitepapers](#)<sup>26</sup> section:

- [Backup and Recovery Approaches Using AWS](#)<sup>27</sup>
- [AWS Security Best Practices](#)<sup>28</sup>
- [Operational Checklists for AWS](#)<sup>29</sup>

---

<sup>26</sup> [AWS whitepapers](#)

<sup>27</sup> [http://d0.awsstatic.com/whitepapers/Backup\\_Archive\\_and\\_Restore\\_Approaches\\_Using\\_AWS.pdf](http://d0.awsstatic.com/whitepapers/Backup_Archive_and_Restore_Approaches_Using_AWS.pdf)

<sup>28</sup> [http://media.amazonwebservices.com/AWS\\_Security\\_Best\\_Practices.pdf](http://media.amazonwebservices.com/AWS_Security_Best_Practices.pdf)

<sup>29</sup> [http://media.amazonwebservices.com/AWS\\_Operational\\_Checklists.pdf](http://media.amazonwebservices.com/AWS_Operational_Checklists.pdf)