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.)
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.
Creating an IAM User in Your AWS Account

**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.

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

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

How to: Creating a Bucket

---

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](http://docs.aws.amazon.com/AmazonS3/latest/UG/UploadingObjectsintoAmazonS3.html).

5. **Enable static website hosting for that bucket.**

   How to: [Configure a Bucket for Website Hosting](http://docs.aws.amazon.com/AmazonS3/latest/dev/HowDoIWebsiteConfiguration.html).

---

4 http://docs.aws.amazon.com/AmazonS3/latest/UG/UploadingObjectsintoAmazonS3.html
5 http://docs.aws.amazon.com/AmazonS3/latest/dev/HowDoIWebsiteConfiguration.html
6. Attach an IAM policy to the IAM user created previously to allow access to the specific bucket.

How to: Managing IAM Policies. 

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

"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 that AWS Elastic Beanstalk maintains automatically for you. 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.

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
<?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']);
*/

```
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: 'AJDSAHCMWKRSDJFIEWRJSDFMKSDMADS'
  - option_name: SECURE_AUTH_KEY
    value: 'SAJDSANCMEWKRSDJFIEWRJSDFMKSDMAD'
  - option_name: LOGGED_IN_KEY
    value: 'DSAJDSADCMEWKRSDJFIEWRJSDFMKSDMA'
  - option_name: NONCE_KEY
    value: 'ADSAJDSAKCMEWKRSDJFIEWRJSDFMKSDM'
  - option_name: SECURE_AUTH_SALT
    value: 'MADSAJDSAKCMEWKRSDJFIEWRJSDFMKSDM'
  - option_name: LOGGED_IN_SALT
    value: 'DMADSAJDSAKCMEWKRSDJFIEWRJSDFMKSDM'
  - option_name: NONCE_SALT
    value: 'SDMADSAJDSAKCMEWKRSDJFIEWRJSDFMK'
  - option_name: AUTH_SALT
```
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.\(^{10}\)

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:

12. Type a unique environment name and URL:

\(^{10}\) \url{http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.environments.html}
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\(^\text{11}\). For this example we also did not select the ‘Create this environment inside a VPC’ option\(^\text{12}\) which is another recommended configuration for production environments.

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.\(^\text{13}\) 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

---


\(^\text{12}\) In practice if you are launching in a region where your account does not support the EC2-Classic 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](http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/default-vpc.html)

\(^\text{13}\) [http://docs.aws.amazon.com/gettingstarted/latest/wah/getting-started-prereq.html#create-a-key-pair](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.

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:

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

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

16. 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.
Please note that if you are running your ElastiCache nodes in Amazon VPC (including Default VPC 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 and Using ElastiCache with Amazon Virtual Private Cloud (VPC). Instead if you are running your ElastiCache nodes in EC2-Classic you will need to create ElastiCache security groups.

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.

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.

19. In the second step, specify that you want to spread the nodes across Availability Zones and select the security group you created earlier:
Please note that this example deploys the cluster into the Default VPC and uses the default Cache Subnet Group\(^{20}\). 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\(^ {21}\)

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:


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](http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.managing.elb.html).

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.
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: wp-admin/
Origin: Elastic Load Balancing (dynamic)
Viewer Protocol Policy: HTTP and HTTPS
Allowed HTTP Methods: GET, HEAD
Cached HTTP Methods: GET, HEAD (Cached by default)
Forward Headers: All
Object Caching: Use Origin Cache Headers
Minimum TTL: 0
Forward Cookies: All
Forward Query Strings: Yes
Object Caching: 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.
Software Installation

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

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

---

23 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:

![Add Plugins Interface]

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

![Installing Plugin Interface]

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

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.
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:

> 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.
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:

![Configuration form]

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:

![TOTAL CACHE]

Plugin configuration successfully updated.

Make sure to export the media library and upload the wp-includes, theme files, custom files, files to the CDN.
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.

Configure both in the same manner:

a. Serve GET HTTP requests only.

b. 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.

c. 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.25

Create Behavior

Cache Behavior Settings

Path Pattern: wp-includes/*

Origin: S3 (static content)

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: None (Improves Caching)

Object Caching:
- Use Origin Cache Headers
- Customize

Minimum TTL: 0

Forward Cookies: None (Improves Caching)

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.
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.

On the same page, configure a scaling policy that will be triggered based on the CPU Utilization metrics:
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:\n
- Backup and Recovery Approaches Using AWS\n- AWS Security Best Practices\n- Operational Checklists for AWS

---

26 [AWS whitepapers](http://d0.awsstatic.com/whitepapers/Backup_Archive_and_Restore_Approaches_Using_AWS.pdf)

27 [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)
