# WordPress: Best Practices on AWS

## Reference Architecture for Scalable WordPress-powered Websites

*Andreas Chatzakis*

*December 2014*


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

# Contents



# 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) is designed to provide a reliable, scalable, secure, and highly performing infrastructure built for the most demanding applications.

This whitepaper provides system administrators with specific guidance on how to get started with WordPress on AWS and how to improve both the cost efficiency of the deployment as well as the end user experience. It also outlines a reference architecture that addresses common scalability and high availability requirements.

# 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.

# Getting Started (Single Server)

For low-traffic blogs or websites without strict high availability requirements a simple deployment of a single server might be suitable. Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity so you can launch a virtual server within minutes.[1]

## Installing WordPress

Because you have complete control of your Amazon EC2 instance, you can log in with root access to install and configure all the software components required to run a WordPress website. Once you are done, you can save that configuration as an Amazon Machine Image (AMI), which you can use to launch new instances with all the customizations that you've made.

An easier way to get started is via the AWS Marketplace—an online store that helps customers find, buy, and quickly start using a large variety of software solutions.[2] Customers can use AWS Marketplace's 1-Click deployment to quickly launch an Amazon EC2 instance based on a publicly available AMI. A range of WordPress offerings are available, allowing users to launch a virtual server preinstalled with a database, web server, and the WordPress application code.[3]

## Selecting the Right Instance Type and Size

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases.[4] Instance types comprise varying combinations of CPU, memory, storage, and networking capacity and give you the flexibility to choose the appropriate mix of resources for your applications. Each instance type includes one or more instance sizes, so you can scale your resources to the requirements of your target workload.

---

[1] http://aws.amazon.com/ec2/

[2] https://aws.amazon.com/marketplace/

[3] https://aws.amazon.com/marketplace/search/results/ref=gtw_navgno_search_box?searchTerms=wordpress&search

[4] http://aws.amazon.com/ec2/instance-types/

## Initial Instance Size

If this is a new project, you must initially make some assumptions about the instance type and size you will need. In general T2 instances offer a cost-efficient choice for low-traffic websites that don't use the full CPU often or consistently, but occasionally need to burst. M3 instance types provide more capacity and a balance of compute, memory, and network resources, making them a good choice for more demanding websites. The above two are recommended starting points, but the optimal choice will depend on the traffic patterns, the complexity of the website, the resources required by the installed plugins, and more. Monitoring the performance of the host can help you find the optimal instance type and size for your WordPress website.

## Scaling Vertically (Scale Up)

As traffic grows, you can increase the size of your instance to handle the additional load. When using an Amazon EC2 instance that is backed by Amazon Elastic Block Store (Amazon EBS), you can increase the size of your instance by simply stopping the instance, modifying its instance type, and restarting.[5] Please note that any data stored on the ephemeral instance storage will be lost during that process, as explained in the *Amazon EC2 User Guide for Linux*.[6]

# Recovering from Failure

Recovering from an instance failure is faster than in traditional hosting environments. You can launch a replacement instance in minutes and use a variety of features that help minimize disruption even in the single server scenario.

## Reliable Storage and Backups

To reestablish the availability of a WordPress website, you must be able to recover the following components:

- OS and services installation and configuration (Apache, MySQL, etc.)
- WordPress application code and configuration
- WordPress themes and plugins
- Uploads (e.g., media files for posts)
- Database content (posts, comments, etc.)

As catalogued in the whitepaper, Backup and Recovery Approaches Using Amazon Web Services, AWS provides a variety of methods for backing up and restoring your web application data and assets:[7]

---

[5] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ComponentsAMIs.html#storage-for-the-root-device

[6] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/InstanceStorage.html

[7] http://media.amazonwebservices.com/AWS_Backup_Recovery.pdf

Amazon EBS volumes provide durable block-level storage for use with Amazon EC2 instances (virtual machines). Amazon EBS volume data is replicated across multiple servers to prevent the loss of data from the failure of any single component. Amazon EBS volumes behave like raw, unformatted block devices that you can use as virtual hard drives. You can create a file system on top of an EBS volume and attach it to your Amazon EC2 instance to provide persistent storage for WordPress media and database files. Amazon EBS volumes provide off-instance, network-attached storage (NAS) that persists independently from the running life of a single Amazon EC2 instance. If an instance fails, you can launch a replacement Amazon EC2 instance and attach the Amazon EBS volume to it. This can even be fully automated with the use of Auto Scaling (to replace the instance) and an initialization script (to attach the Amazon EBS volume to the new instance).

As an example, in the scenario of a single-server installation described before, you could configure your system so that user-defined code and assets (e.g., the plugins, themes, and uploads located under the `wp-content` folder) and database content (e.g., `/var/lib/mysql`) are stored on an Amazon EBS volume.

Amazon EBS volumes have a lower annual failure rate (AFR) compared with commodity hard disks, but it is always safest to follow backup best practices. In addition human error is always a possibility (e.g., deletion of important files by a WordPress site administrator). This highlights the importance of defining and testing your backup strategy.

The most suitable AWS storage service for durably storing backups is Amazon Simple Storage Service (Amazon S3).[8] Amazon S3 offers software developers a highly scalable, reliable, and low-latency data storage infrastructure at very low cost and accessible via REST and SOAP web service APIs. Amazon S3 redundantly stores your objects not only on multiple devices but also across multiple facilities in an Amazon S3 region providing even greater durability than Amazon EBS. A variety of WordPress plugins are available for scheduled and manual backups to Amazon S3.[9]

A more cost-efficient backup method for data that is stored on Amazon EBS volumes is to use the snapshot functionality. This feature creates a point-in-time backup copy of an Amazon EBS volume that is then stored in Amazon S3. Amazon EBS snapshots are stored incrementally: Only the blocks that have changed since your last snapshot are saved, and you are billed only for the changed blocks.

Finally it is worth noting that AMIs are also stored on Amazon S3 for durable storage of your server(s) base configuration.

---

[8] http://aws.amazon.com/s3/

[9] Examples include Updraft Plus (https://wordpress.org/plugins/updraftplus/), BackWPup Free (https://wordpress.org/plugins/backwpup/), and BackUpWordPress (https://wordpress.org/plugins/backupwordpress/). Note that neither AWS nor the author has tested nor endorsed these or any other specific plugin.

## IP Addressing and DNS

Elastic IP addresses are static IP addresses designed for dynamic cloud computing with which you can mask instance or Availability Zone (AZ) failures. Instead of waiting for DNS to propagate to all of your customers, Amazon EC2 allows you to engineer around problems with your instance or software by remapping your Elastic IP address to a replacement instance.

In addition, you can use Amazon Route 53, Amazon's highly available DNS hosting solution, to point your domain name to the IP address of the instance that is hosting your website.[10]

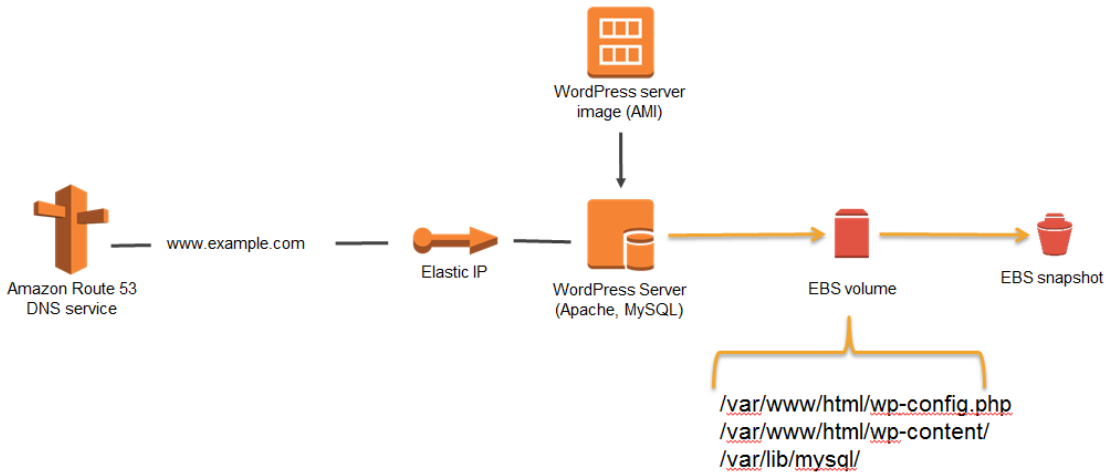


**Figure 1: A Single Instance Deployment**

# Improving Performance and Cost Efficiency

Although the above setup can scale vertically to a certain extent by using a higher spec instance type, you can easily increase performance, reduce cost, and improve the end user experience with a few modifications. A lot of those modifications require the use of one or more WordPress plugins. Although various options are available, W3 Total Cache is a popular choice that combines many of those modifications in a single plugin.[11]

[10] http://aws.amazon.com/route53/

[11] https://wordpress.org/plugins/w3-total-cache/

## Browser and Edge Caching

Any WordPress website needs to deliver a mix of static and dynamic content. Static content includes images, JavaScript files, or style sheets; dynamic content includes anything generated on the server side via the WordPress PHP code—e.g., elements of your site that are loaded from the database or even personalized to each viewer. An important aspect of the end user experience is the network latency involved when delivering the above content to users around the world.

Amazon CloudFront is a web service that gives businesses and web application developers an easy and cost-effective way to distribute their content with low latency and high data transfer speeds through multiple edge locations across the globe.[12] Viewer requests are automatically routed to a suitable Amazon CloudFront edge location in order to lower the latency. If the content can be cached (for a few seconds, minutes, or even days) and is already stored in a particular edge location, CloudFront delivers it immediately. If the content should not be cached, has expired, or is not currently in that edge location, CloudFront retrieves it from the origin configured as the source for the definitive version of the content. This retrieval takes place over optimized network connections, which work to speed up the delivery of content on your website. Apart from improving the end user experience, the above model also reduces the load on your origin servers and has the potential to create significant cost savings.

You can create a CloudFront distribution, map it to your website's domain name, and configure two origins as the sources of the content. You configure rules in the form of CloudFront behaviors that define which origin to use based on specific path patterns.

### *Static Content*

This includes CSS, JavaScript, and image files—either those that are part of your WordPress themes or those media files uploaded by the content administrators. All these files can be stored in Amazon S3 (described previously in Reliable Storage and Backups). Apart from secure storage for private content like backups, you can configure Amazon S3 to serve public content in a scalable and highly available manner.

This has the positive side effect of offloading this workload from your Amazon EC2 web server and letting it focus on the dynamic content generation. This will reduce the load on the server but later on in this document we will also see how this creates a stateless architecture (and why this is a prerequisite before we can implement Auto Scaling).

You can subsequently configure Amazon S3 as an origin for CloudFront to improve delivery of those static assets to users around the world. Although WordPress is not integrated with Amazon S3 and CloudFront out of the box, a variety of plugins add that support (e.g., W3 Total Cache).

---

[12] http://aws.amazon.com/cloudfront/

*Dynamic Content*

This includes the output of server-side WordPress PHP scripts and can also be served via CloudFront by configuring the Amazon EC2 web server as an origin. Since this will include personalized content, you need to configure CloudFront to forward certain HTTP cookies and HTTP headers as part of a request to your custom origin server. CloudFront uses the forwarded cookie values as part of the key that identifies a unique object in its cache. To ensure you maximize the caching efficiency, you should configure CloudFront to only forward those HTTP cookies and HTTP headers that really vary the content (not cookies that are only used on the client side or by third-party applications, e.g., for web analytics).

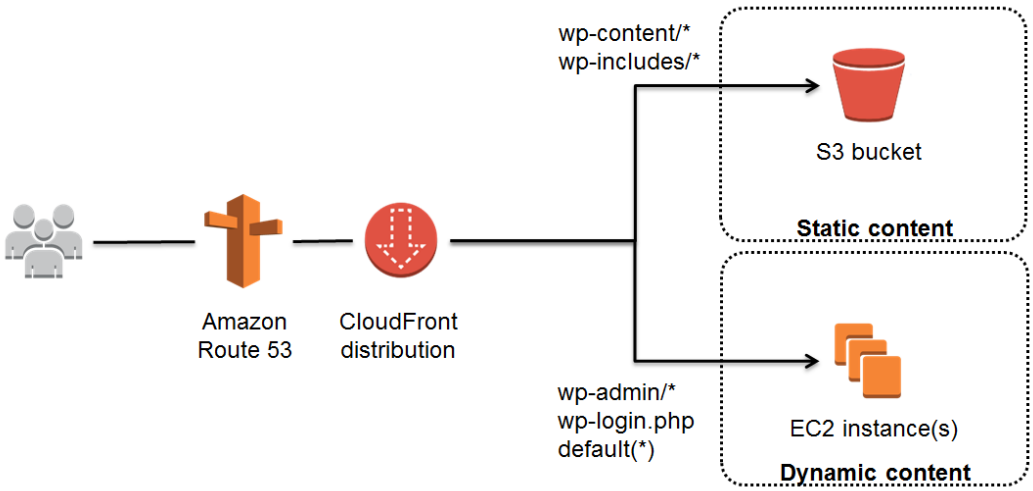


**Figure 2: Whole Website Delivery via CloudFront**

Amazon CloudFront uses standard cache control headers to identify if and for how long it should cache specific HTTP responses. The same cache control headers are also used by web browsers to decide when and for how long to cache content locally for even more optimal end user experience. (For example, a `.css` file that is already downloaded will not be redownloaded every time a returning visitor views a page.) You can configure this on the web server level (e.g., via `.htaccess` files or modifications of the `httpd.conf` file) or install a WordPress plugin (e.g., W3 Total Cache) to dictate how those headers are set for both static and dynamic content.

## Database Caching

Database caching can significantly reduce latency and increase throughput for read-heavy application workloads like WordPress. Application performance is improved by storing frequently accessed pieces of data in memory for low-latency access (e.g., the results of I/O-intensive database queries). When a large percentage of the queries are served from the cache, the number of queries that need to hit the database is reduced, resulting in a lower cost associated with scaling the database.

Although WordPress has limited caching capabilities out of the box, a variety of plugins support integration with Memcached, a widely adopted memory object caching system. The W3 Total Cache plugin is a good example.

You can install Memcached on an Amazon EC2 instance. In the simplest scenarios, you install Memcached on your web server and capture the result as a new AMI. In this case you are responsible for the administrative tasks associated with running a cache.

Another option is to take advantage of Amazon ElastiCache[13] and avoid that operational burden. This is a managed service that makes it easy to deploy, operate, and scale a distributed in-memory cache in the cloud. One of the supported caching engines of ElastiCache is Memcached and as such it can be used with WordPress and a suitable WordPress Memcached plugin without any further customizations.

### Bytecode Caching

Each time a PHP script is executed, it gets parsed and compiled. By utilizing a PHP bytecode cache, the output of the PHP compilation is stored in RAM so that the same script does not have to be compiled again and again. This reduces the overhead related to executing PHP scripts, resulting in better performance and lower CPU requirements.

A bytecode cache can be installed on any EC2 instance that hosts WordPress and can greatly reduce its load. A popular PHP bytecode cache is Alternative PHP Cache or APC,[14] but for PHP 5.5 and later we recommend the use of OPcache that is a bundled extension with that PHP version.[15]

# Scaling Up

Although a single-instance deployment can be sufficient for some cases, any WordPress website that serves a significant business or other purpose requires a highly available and better performing architecture to power it. With AWS, you can launch such an environment in a very short time.

## Separating the Web and Database Tiers

The first step is to separate the web and the database (DB) tiers. By using two distinct servers, we can increase the capacity of the implementation but also gain more control on the configuration characteristics of each layer (web and database).

---

[13] http://aws.amazon.com/elasticache/

[14] http://php.net/apc

[15] http://php.net/manual/en/book.opcache.php

A web server and a DB server typically have different requirements for CPU, memory, storage, and networking capacity. The single-server scenario works well with the general purpose instance types (e.g., T2, M3), but AWS provides a wider variety of Amazon EC2 instance types so you can optimize each workload's server configuration for both performance and cost. Generally speaking, the compute-optimized C3 instance might be a good choice for a WordPress web server, while a memory-optimized R3 could result in higher database performance due to the available RAM and how it can be used, for example, by the InnoDB storage engine for its buffer pool.[16]

When moving the database to its own separate infrastructure, the WordPress configuration file (`wp-config.php`) on the web server would simply need to be reconfigured to point to the host name of the separate database instance. That database instance could be an Amazon EC2 instance that you manage on your own and where you have installed MySQL. In that case you are responsible for the administrative tasks associated with running a database server. A better solution is to use Amazon Relational Database Service (RDS).[17]

Amazon RDS is a managed service that gives you access to the capabilities of a MySQL database engine while managing time-consuming database administration tasks, freeing you up to focus on your applications and business. Amazon RDS automatically patches the database software, backs up your database, stores those backups for a user-defined period, and supports point-in-time recovery. You can also scale your database's compute resources or storage capacity with a single API call.

[16] http://dev.mysql.com/doc/refman/5.6/en/innodb-buffer-pool.html

[17] http://aws.amazon.com/rds/

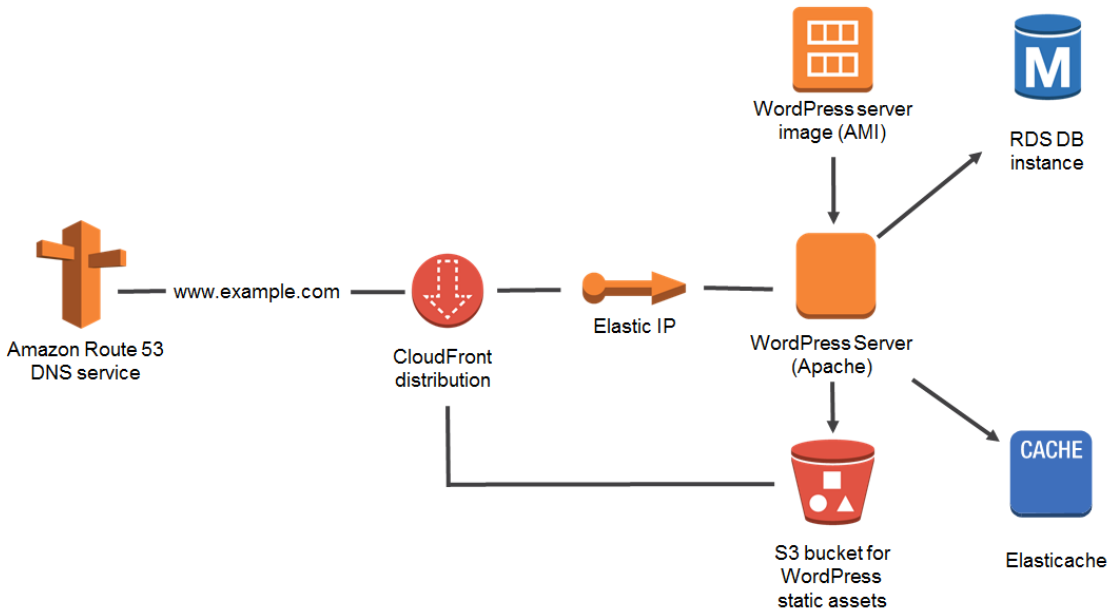


**Figure 3: Separating the Three Workloads (Web, Database, Cache)**

## Availability of the Data Layer

Amazon RDS Multi-AZ deployments provide enhanced availability and durability for database instances, making them a natural fit for production database workloads. When you provision a Multi-AZ DB instance, Amazon RDS automatically creates a primary DB instance and synchronously replicates the data to a standby instance in a different Availability Zone (AZ). AZs are physically distinct locations within a particular AWS region. They provide inexpensive, low-latency network connectivity to other AZs in the same region. Each AZ runs on its own physically distinct, independent infrastructure and is engineered to be highly reliable. In case of an infrastructure failure, Amazon RDS performs an automatic failover to the standby so that you can resume database operations as soon as the failover is complete. Since the endpoint for your DB instance remains the same after a failover, you application can resume database operation without the need for manual administrative intervention.

Regarding the Memcached layer, ElastiCache automatically replaces individual nodes when they fail. To further improve the reliability of the cluster, you can also take advantage of the flexible node placement model for ElastiCache. With that configuration, Memcached cache clusters of two or more nodes can span multiple AZs within a region.

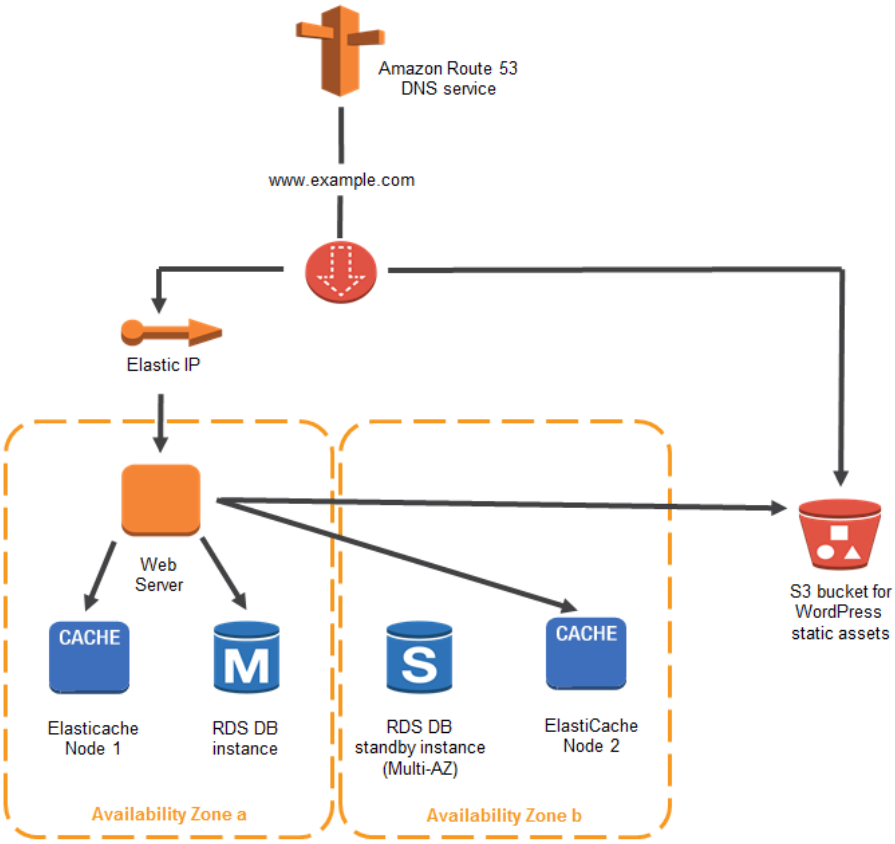


**Figure 4: Highly Available Data Layer**

## A Note on Amazon RDS and WordPress Multisite

WordPress provides a multisite feature with which you can power a network of sites from a single WordPress installation. In this mode, while some of the database tables remain shared, each of those sites also adds its own separate tables in the database. If you plan to power hundreds of websites from a single database instance, be aware that the best practice for Amazon RDS MySQL instances is to not create more than 10,000 tables using Provisioned IOPS or 1,000 tables using standard storage. Large numbers of tables significantly increase database recovery time after a failover or database crash. If you really need to create more tables than recommended, set the `innodb_file_per_table` parameter to 0. For more information, see Working with InnoDB Tablespaces to Improve Crash Recovery Times[18] and Working with DB Parameter Groups[19] in the *Amazon RDS User Guide*.

[18] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Appendix.MySQL.CommonDBATasks.html#Appendix.MySQL.CommonDBATasks.Tables

[19] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_WorkingWithParamGroups.html

# Availability and Scalability of the Web Tier

Once you have built a solid foundation with a highly available data tier, the next concern is the availability and scalability of your web serving capacity.

## High Availability for the Web Tier

To support high availability for the web servers that power a WordPress website you need to deploy more than one node so that your application can withstand an individual web server failure. Similar to what was described in Availability of the Data Layer, you can deploy those instances in separate AZs within a region to increase the reliability of the overall architecture.

To distribute end user requests to multiple web server nodes, you need a load balancing solution. AWS provides this capability through Elastic Load Balancing, a highly available service that distributes traffic to multiple Amazon EC2 instances.[20]

Elastic Load Balancing supports distribution of requests across multiple Availability Zones within an AWS region. You can also configure a health check so that the Elastic Load Balancing load balancer automatically stops sending traffic to individual instances that have failed (e.g., due to a hardware problem or software crash). For more information, see Health Check in the *Elastic Load Balancing Developer Guide*.[21]

## Scalability for the Web Tier

Another key characteristic of the AWS platform is its elasticity. You can launch more compute capacity (e.g., web servers) when you need it and run less when you don't. Because you only pay for what you use and there is no need to overprovision, you can optimize costs. You also minimize the risk of underprovisioning as you don't need to guess the exact required capacity that you need at peak times.

---

[20] http://aws.amazon.com/elasticloadbalancing/

[21] http://docs.aws.amazon.com/ElasticLoadBalancing/latest/DeveloperGuide/TerminologyandKeyConcepts.html#healthcheck

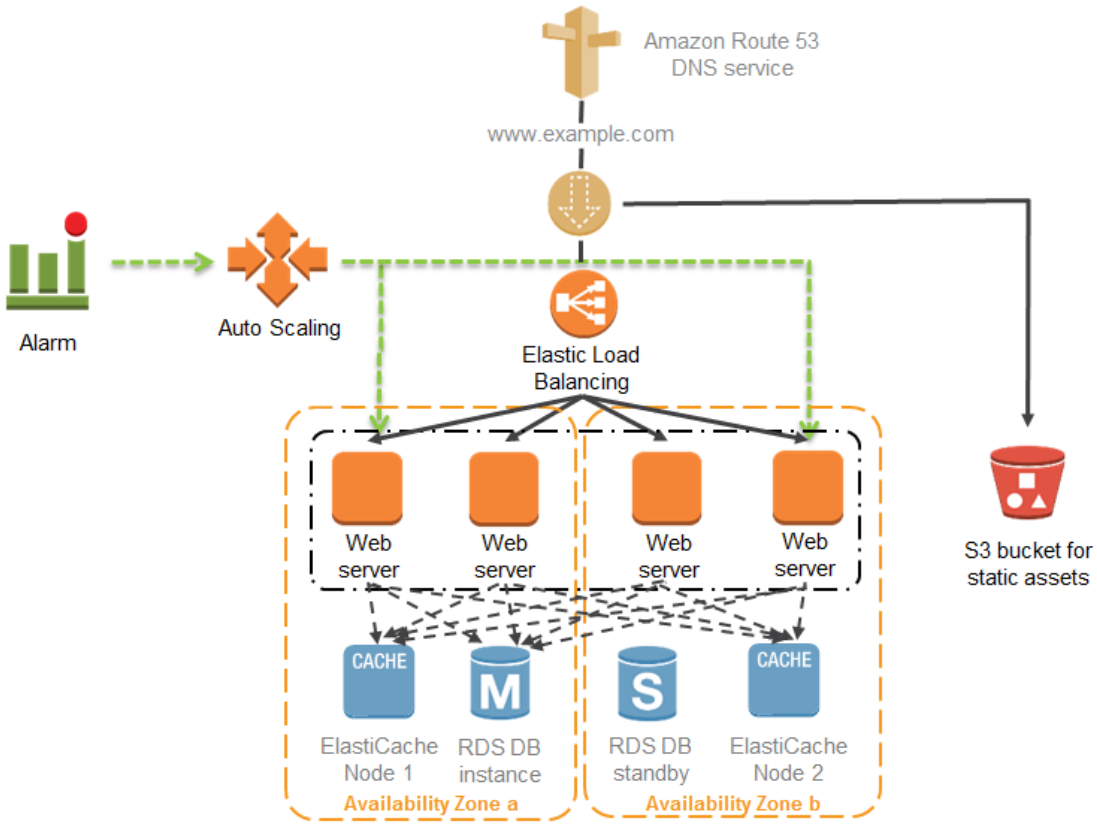


**Figure 5: High Availability and Scalability for the Web Tier**

Auto Scaling is an AWS service that helps you automate this provisioning to scale your Amazon EC2 capacity up or down according to conditions you define with no need for manual intervention.[22] You can configure Auto Scaling so that the number of Amazon EC2 instances you're using increases seamlessly during demand spikes to maintain performance and decreases automatically when traffic diminishes so as to minimize costs.

The Elastic Load Balancing service supports dynamic addition and removal of Amazon EC2 hosts from the load-balancing rotation. The Elastic Load Balancing service itself will also dynamically grow and shrink the load-balancing capacity to adjust to traffic demands with no manual intervention.

## A Stateless Architecture

To take advantage of multiple web servers in an Auto Scaling configuration, your web tier must be stateless. This means that any data that needs to persist for more than a single HTTP request is not stored on the web servers. This is because existing instances are automatically terminated when the Auto Scaling configuration indicates that some of

[22] http://aws.amazon.com/autoscaling/

the provisioned capacity is not needed. When that happens, any data that has not been stored outside of that instance is lost.

By storing any kind of data either at the client side (e.g., cookies) or into some shared durable storage (e.g., a Multi-AZ Amazon RDS database or Amazon S3) instead of the local file system, you have the ability to simply launch more Amazon EC2 instances when you need to scale up and terminate any one of them when you have excess capacity. You also avoid the need to set up any complex mechanism to synchronize data across multiple web servers. In addition a single web server instance failure (e.g., due to a hardware issue) does not impact the integrity of your data as it would not be storing any crucial data locally. For most web applications this breaks down to two areas, user sessions on the one hand and user-generated data and other uploads on the other.

### *User Sessions*

Unlike many other web applications, the WordPress core is completely stateless with respect to session data storage. It instead relies on cookies. For example logged in users are identified via an authentication cookie whose validity is checked against a record stored in the database. As such, session storage is not a concern unless you have installed any custom code (e.g., a WordPress plugin) that instead relies on native PHP sessions. In that case Amazon DynamoDB is the best option for a fast, durable, and scalable store for PHP sessions. For a good discussion of this, see Jeremy Lindblom's blog post on Scalable Session Handling with DynamoDB.[23]

### *User-generated Data and Other Uploads*

By default WordPress stores user uploads on the local file system so this is an area where WordPress does not have a stateless architecture. We already discussed how using a plugin to store those files into Amazon S3 and serving from CloudFront helps reduce server load and improve the end user experience. Another benefit is that it moves this function to a separate tier and makes the web layer stateless.

## Scaling the Data Layer

Even with caching, you may need more database capacity. If so, you can easily switch to a larger Amazon RDS instance type and/or use Provisioned IOPS storage for higher and consistent disk performance. For more information, see DB Instance Class[24] and Provisioned IOPS Storage[25] in the *Amazon Relational Database User Guide*.

WordPress deployments are typically read-heavy workloads, so a more efficient way to scale can be to take advantage of MySQL replication by launching one or more Amazon RDS read replicas. These are read-only copies of your database with which you can

---

[23] http://aws.amazon.com/blogs/aws/scalable-session-handling-in-php-using-amazon-dynamodb/

[24] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.DBInstanceClass.html

[25] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Storage.html#USER_PIOPS

scale out beyond the capacity of a single database deployment. See Working with PostgreSQL and MySQL Read Replicas in the Amazon RDS documentation.[26]

Out of the box WordPress is not designed to take advantage of multiple database instances, so you will need to extend it with a suitable plugin. Such an example is the HyperDB plugin[27] for WordPress, a replacement for the standard WordPress database class, which adds the ability to use multiple database instances.[28]

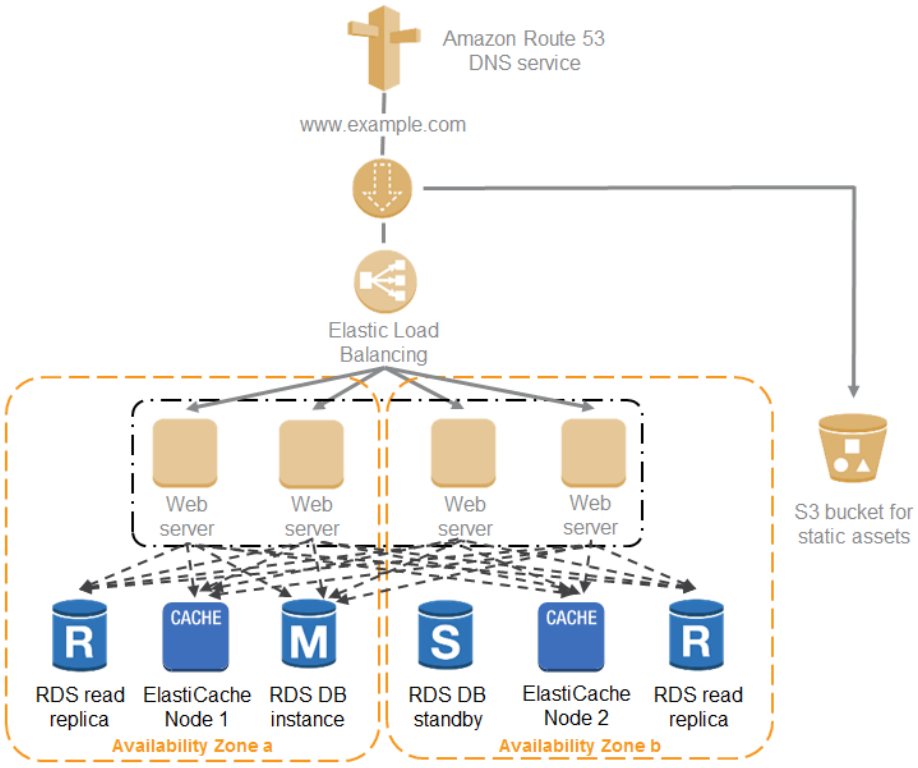


**Figure 6: Scaling the Database with RDS Read Replicas**

[26] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ReadRepl.html

[27] https://wordpress.org/plugins/hyperdb/

[28] At the time of writing, HyperDB and the W3 Total Cache mentioned previously are not compatible with each other out of the box.

# Development and Deployment Considerations

Few websites remain static. In most cases you will be periodically adding publicly available WordPress themes and plugins or upgrading to a newer WordPress version. In other cases you will be developing your own custom themes and plugins from scratch.

## Reducing the Risk of Change

Anytime you are making a structural change to your WordPress installation there is certain risk of introducing unforeseen problems. At the very least you should be taking a backup of your application code, configuration, and database before applying any significant change (e.g., installing a new plugin). For web sites of business or other value, you should certainly be testing those changes on a separate staging environment first. With AWS it is very easy to replicate the configuration of your production environment and run the whole deployment process in a safe manner. Once you are done with your tests, you can simply tear down your test environment and stop paying for those resources. Below we discuss some WordPress specific considerations but for more information on development and test best practices on AWS please refer to the Development and Test on Amazon Web Services whitepaper.[29]

### Deploying New Plugins

Due to the way WordPress plugin installation takes place, some planning is often required. On activation, some WordPress plugins write configuration information to the wp_options database table (or introduce DB schema changes). Some plugins create configuration files in the local file system. In a multiserver environment this can be a challenge because only one of the running instances will have those files created. A simple solution to that problem is to run the plugin installation on a staging server first. Any file system modifications will have to be merged with your application code and introduced as a new version of the application code. You would then use your standard deployment method to make sure these changes are pushed to all your web servers.

### Deploying Theme Changes

When deploying theme changes and if you are using Amazon S3 for the storage of assets (JavaScript, style sheet, and image files), you will need a process to copy those to the right bucket location. Plugins like W3 Total Cache provide a way for you to manually initiate that task. Alternatively you could automate this step as part of a build process.

Because those assets can be cached on CloudFront and at the browser, you need a way to invalidate older versions when you deploy changes. The best way to achieve that is by including some sort of version identifier in your object. This identifier might be a

[29] http://aws.amazon.com/whitepapers/dev-test-on-aws/

query string with a date-time stamp, or a random string. If you use the W3 Total Cache plugin, you can update a media query string that is appended to the URLs of media files.

## Responsive Design

As the use of smart phones and tablets continues to grow, it is important to optimize your website's viewing experience for a variety of devices. You can achieve that in a combination of ways:

- Rely on client side logic (e.g., CSS3 media queries, fluid layout, etc) to adapt the presentation without modifying the HTML content itself.
- Use a plugin to switch the current WordPress theme based on the user's device.
- Use conditional logic within the WordPress theme's templates files

Although you can improve the user experience just with client side logic, you have a lot more control when you can actually modify the HTML output on the server side. You can use the User-Agent HTTP header to identify and discriminate between desktop and mobile devices and to provide content that is suitable for and appropriate to each one. If you are using CloudFront to serve dynamic content, you can instead use its device type detection headers (CloudFront-is-mobile-viewer, CloudFront-is-desktop-viewer, CloudFront-is-tablet-viewer). A good approach is to use those higher level headers that vary much less in their possible values instead of passing the User-Agent back to the origin. This results in a better cache hit-rate and reduces the number of requests that hit your origin server(s).

The following example shows an excerpt from a WordPress template file, which uses CloudFront's headers to conditionally return a differently sized featured image depending on the device type. This reduces unnecessary bandwidth usage and improves page load speed:

```
<?php if ( has_post_thumbnail() && !
post_password_required() && ! is_attachment() ) : ?>
<div class="entry-thumbnail">
<?php if
($_SERVER['HTTP_CLOUDFRONT_IS_DESKTOP_VIEWER']=='true')
                        {the_post_thumbnail('large');}
else if
($_SERVER['HTTP_CLOUDFRONT_IS_TABLET_VIEWER']=='true')
                        {the_post_thumbnail('medium');}
else if
($_SERVER['HTTP_CLOUDFRONT_IS_MOBILE_VIEWER']=='true')
                        {the_post_thumbnail('thumbnail');}
```

```
else
```

```
                            {the_post_thumbnail('large');}?>
</div> <?php endif; ?>
```

# Summary

AWS presents many architecture options for running WordPress. The simplest option is a single server installation for low traffic websites. For more advanced websites, site administrators can add several other options, each one representing an incremental improvement in terms of availability and scalability. Administrators can select the features that most closely match their requirements and their budget. For details on the easiest way to implement such an architecture on AWS, please refer to the "Deploying WordPress with AWS Elastic Beanstalk" whitepaper.