WordPress: Best Practices on AWS

Reference Architecture for Scalable WordPress-Powered Websites

February 2018

Notices

Contents

Abstract

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

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.

The first version of WordPress was released in 2003, and 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 Cloud.

Simple Deployment

For low-traffic blogs or websites without strict high availability requirements a simple deployment of a single server might be suitable. This deployment isn't the most resilient or scalable architecture, but it is the quickest and most economical way to get your website up and running.

Considerations

We will start with a single web server deployment. There may be occasions when you outgrow it, for example:

- The virtual machine that your WordPress website is deployed on will be a single point of failure. A problem with this instance will cause a loss of service for your website.
- Scaling resources to improve performance can only be achieved by "vertical scaling," that is, by increasing the size of the virtual machine running your WordPress website.

Available Approaches

AWS has a number of different options for provisioning virtual machines. There are three main ways to host your own WordPress website on AWS:

- Amazon Lightsail

- Amazon Elastic Compute Cloud (Amazon EC2)
- AWS Marketplace

Amazon Lightsail is a service that allows you to quickly launch a virtual private server (a Lightsail instance) to host a WordPress website.[1] Lightsail is the easiest way to get started if you don't need highly configurable instance types or access to advanced networking features.

Amazon EC2 is a web service that provides resizable compute capacity so you can launch a virtual server within minutes.[2] Amazon EC2 provides more configuration and management options than Lightsail, which is desirable in more advanced architectures. You have administrative access to your EC2 instances and can install any software packages you choose, including WordPress.

AWS Marketplace is an online store where you can find, buy, and quickly deploy software that runs on AWS. [3] You can use 1-Click deployment to launch preconfigured WordPress images directly to Amazon EC2 in your own AWS account in just a few minutes. There are a number of Marketplace vendors offering ready-to-run WordPress instances.

We will cover the Lightsail option as the recommended implementation for a single server WordPress website.

Amazon Lightsail

Lightsail is the easiest way to get started on AWS for developers, small businesses, students, and other users who need a simple virtual private server (VPS) solution.

The service abstracts many of the more complex elements of infrastructure management away from the user. It is, therefore, an ideal starting point if you have less infrastructure experience, or when you need to focus on running your website and a simplified product is sufficient for your needs.

With Amazon Lightsail you can choose Windows or Linux/Unix operating systems and popular web applications, including WordPress, and deploy these with a single click from preconfigured templates.

As your needs grow, you will have the ability to smoothly step outside of the initial boundaries and connect to additional AWS database, object storage, caching, and content distribution services.

Selecting an Amazon Lightsail Pricing Plan

A Lightsail Plan defines the monthly cost of the Lightsail resources you will use to host your WordPress website.[4] There are a number of plans available to cover a variety of use cases, with varying levels of CPU resource, memory, solid-state drive (SSD) storage, and data transfer. If your website is complex you may need a larger instance with more resources. You can achieve this by migrating your server to a larger plan using the web console[5] or as described in the Amazon Lightsail CLI documentation.[6]

Installing WordPress

Lightsail provides templates for commonly used applications such as WordPress. This template is a great starting point for running your own WordPress website as it comes pre-installed with most of the software you will need. You can install additional software or customize the software configuration by using the in-browser terminal or your own SSH client, or via the WordPress administration web interface. For more information about managing WordPress on Lightsail, refer to the Getting started using WordPress from your Amazon Lightsail instance documentation.[7] Once you are finished customizing your WordPress website, we recommend taking a snapshot of your instance.

A snapshot is a way to create a backup image of your Lightsail instance.[8] It is a copy of the system disk and also stores the original machine configuration (that is, memory, CPU, disk size, and data transfer rate). Snapshots can be used to revert to a known good configuration after a bad deployment or upgrade.

This snapshot will allow you to recover your server if needed, but also to launch new instances with the same customizations.

Recovering from Failure

A single web server is a single point of failure, so you must ensure that your website data is backed up. The snapshot mechanism described earlier can also be used for this purpose. To recover from failure, you can restore a new instance from your most recent snapshot. To reduce the amount of data that could be lost during a restore, your snapshots must be as recent as possible.

To minimize the potential for data loss, ensure that snapshots are being taken on a regular basis. This can be done by automating snapshots using the Lightsail Auto Snapshots solution.[9]

We recommend that you use a static IP—a fixed, public IP address that is dedicated to your Lightsail account. If you need to replace your instance with another one, you can reassign the static IP to the new instance. In this way, you don't have to reconfigure any external systems (such as DNS records) to point to a new IP address every time you want to replace your instance.

Improving Performance and Cost Efficiency

You may eventually outgrow your single-server deployment. You will need to consider options for improving your website's performance. Before migrating to a multi-server, scalable deployment – as we discuss later in this paper – there are a number of performance and cost efficiencies you can apply. These are good practices that you should follow anyway, even if you do move to a multi-server architecture.

The following sections introduce a number of options that can improve aspects of your WordPress website's performance and scalability. Some can be applied to a single-server deployment, while many take advantage of the scalability of multiple servers. 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.[10]

Accelerating Content Delivery

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 using the WordPress PHP code, for example, elements of your site that are generated from the database or personalized to each viewer. An important aspect of the end-user experience is the network latency involved when delivering the previous content to users around the world. Accelerating the delivery of the previous content will improve the end-user experience, especially users geographically spread across the globe. This can be achieved with a Content Delivery Network (CDN) such as Amazon CloudFront.

Amazon CloudFront is a web service that provides an easy and cost-effective way to distribute content with low latency and high data transfer speeds through multiple edge locations across the globe.[11] Viewer requests are automatically routed to a suitable CloudFront edge location in order to lower the latency.[12] 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 isn't currently in that edge location, CloudFront retrieves content from one or more sources of truth, referred to as the origin(s) (in this case, the Lightsail instance) in the CloudFront configuration. 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 model we have discussed also reduces the load on your origin servers and has the potential to create significant cost savings.

Static Content Offload

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 Simple Storage Service (Amazon S3) using a plugin such as W3 Total Cache and served to users in a scalable and highly available manner. Amazon S3 offers a highly scalable, reliable, and low-latency data storage infrastructure at very low cost, which is accessible via REST APIs.[13] Amazon S3 redundantly stores your objects, not only on multiple devices, but also across multiple facilities in an Amazon S3 Region, thus providing exceptionally high levels of durability.

This has the positive side effect of offloading this workload from your Lightsail instance and letting it focus on dynamic content generation. This reduces the load on the server. Later in this document we will also see that this is an important step towards creating 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 isn't integrated with Amazon S3 and CloudFront out of the box, a variety of plugins add support for these services (for example, W3 Total Cache).

Dynamic Content

Dynamic content includes the output of server-side WordPress PHP scripts. It can also be served via CloudFront by configuring the WordPress website 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 that 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, for example, for web analytics).

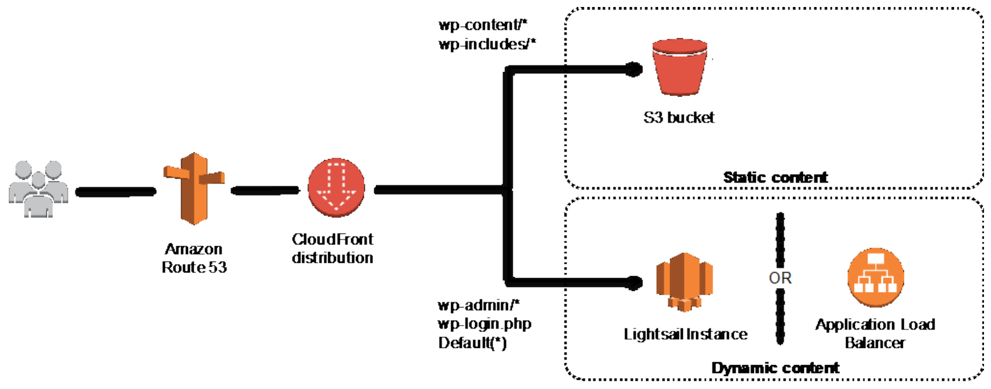

Figure 1: Whole website delivery via CloudFront

In Figure 1 you can see that we now have two origins: one for static content and another for dynamic content. For implementation details, see [Appendix A](#).

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 a more optimal end-user experience (for example, a `.css` file that is already downloaded will not be re-downloaded every time a returning visitor views a page). You can configure cache control headers on the web server level (for example, via `.htaccess` files or modifications of the `httpd.conf` file) or install a WordPress plugin (for example, 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 (for example, 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 running 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.[14]

In the simplest scenarios, you install Memcached on your web server and capture the result as a new snapshot. In this case, you are responsible for the administrative tasks associated with running a cache.

Another option is to take advantage of a managed service such as Amazon ElastiCache[15] and avoid that operational burden. ElastiCache makes it easy to deploy, operate, and scale a distributed in-memory cache in the cloud. You can find information about how to connect to your ElastiCache cluster nodes in the Amazon ElastiCache documentation.[16]

If you are using Lightsail and wish to access an ElastiCache cluster in your AWS account privately, you can do so by using VPC peering. You can find instructions to enable VPC peering in the Lightsail documentation.[17]

Bytecode Caching

Each time a PHP script is executed, it gets parsed and compiled. By using a PHP bytecode cache, the output of the PHP compilation is stored in RAM so that the same script doesn't 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 Lightsail instance that hosts WordPress and can greatly reduce its load. For PHP 5.5 and later we recommend the use of OPcache, a bundled extension with that PHP version.[18]

Note that OPcache is enabled by default in the Bitnami WordPress Lightsail template so no further action is required.

Elastic Deployment

There are many scenarios where a single-server deployment may not be sufficient for your website. In these situations, you will need a multi-server, scalable architecture.

Reference Architecture

There is a reference architecture available on GitHub that outlines best practices for deploying WordPress on AWS and includes a set of CloudFormation templates to get you up and running quickly.[19] The following architecture is based on that reference architecture. The rest of this section will review the reasons behind the architectural choices.

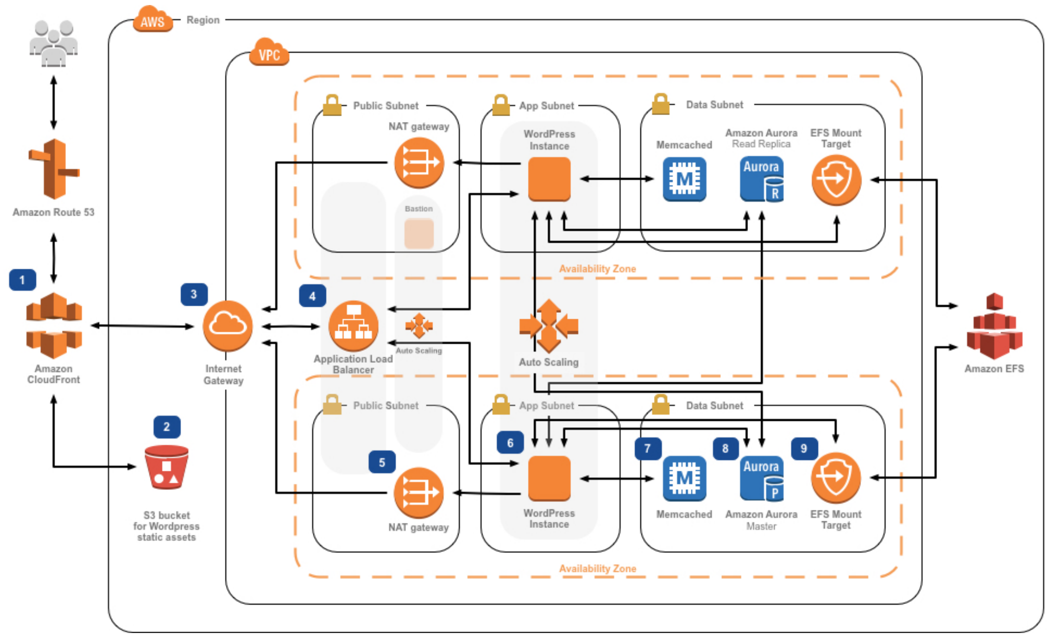

Figure 2: Reference architecture for hosting WordPress on AWS

Architecture Components

The reference architecture in Figure 2 illustrates a complete best practice deployment for a WordPress website on AWS. It starts with edge caching in **Amazon CloudFront** (1) to cache content close to end users for faster delivery. CloudFront pulls static content from an **S3 bucket** (2) and dynamic content from an **Application Load Balancer** (4) in front of the web instances. The web instances run in an **Auto Scaling group** of **Amazon EC2 instances** (6). An **ElastiCache** cluster (7) caches frequently queried data to speed up responses. An **Amazon Aurora** MySQL instance (8) hosts the WordPress database. The WordPress EC2 instances access shared WordPress data on an **Amazon EFS** file system via an **EFS Mount Target** (9) in each Availability Zone. An **Internet Gateway** (3) allows communication between resources in your VPC and the internet. **NAT Gateways** (5) in each Availability Zone enable EC2 instances in private subnets (App and Data) to access the internet.

Within the **Amazon VPC** there exist two types of subnets: public (**Public Subnet**) and private (**App Subnet** and **Data Subnet**). Resources deployed into the public subnets will receive a public IP address and will be publically visible on the internet. The **Application Load Balancer** (4) and a Bastion host for administration are deployed here. Resources deployed into the private subnets receive only a private IP address and hence are not publically visible on the internet, improving the security of those resources. The **WordPress web server instances** (6), **ElastiCache cluster instances** (7), **Aurora MySQL database instances** (8), and **EFS Mount Targets** (9) are all deployed in private subnets.

The remainder of this section deals with each of these considerations in more detail.

Scaling the Web Tier

To evolve your single-server architecture into a multi-server, scalable one there are five key components you will need to use: EC2 instances, Amazon Machine Images (AMIs), load balancers, Auto Scaling, and health checks.

AWS provides a wide variety of EC2 instance types so you can choose the best server configuration for both performance and cost. Generally speaking, the compute-optimized (for example, C4) instance type might be a good choice for a WordPress web server. You can deploy your instances across multiple Availability Zones within a Region to increase the reliability of the overall architecture.

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

To distribute end-user requests to multiple web server nodes, you need a load balancing solution. AWS provides this capability through Elastic Load Balancing (ELB), a highly available service that distributes traffic to multiple EC2 instances.[20] Because your website will be serving content to your users via HTTP or HTTPS, we recommend that you make use of the Application Load Balancer, an application-layer load balancer with content routing and the ability to run multiple WordPress websites on different domains if required.

ELB supports distribution of requests across multiple Availability Zones within an AWS Region. You can also configure a health check so that the Application Load Balancer automatically stops sending traffic to individual instances that have failed (for example, due to a hardware problem or software crash). We recommend using the WordPress admin login page (`/wp-login.php`) for the health check because this page will confirm both that the web server is running and that the web server is configured to serve PHP files correctly. You may choose to build a custom health check page that checks other dependent resources, such as database and cache resources. For more information, see Health Checks for Your Target Groups in the *Application Load Balancer Guide.*[21]

Elasticity is a key characteristic of the AWS Cloud. You can launch more compute capacity (for example, web servers) when you need it and run less when you don't. 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 EC2 instances you're using increases seamlessly during demand spikes to maintain performance and decreases automatically when traffic diminishes, so as to minimize costs.

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

Stateless Web Tier

To take advantage of multiple web servers in an Auto Scaling configuration, your web tier must be stateless. A stateless application is one that needs no knowledge of previous interactions and stores no session information. In the case of WordPress, this means that all end users receive the same response, regardless of which web server processed their request. A stateless application can scale horizontally since any request can be serviced by any of the available compute resources (that is, web server instances). When that capacity is no longer required, any individual resource can be safely terminated (after running tasks have been drained). Those resources do not need to be aware of the presence of their peers – all that is required is a way to distribute the workload to them.

When it comes to user session data storage, the WordPress core is completely stateless because it relies on cookies that are stored in the client's web browser. Session storage isn't a concern unless you have installed any custom code (for example, a WordPress plugin) that instead relies on native PHP sessions.

However, WordPress was originally designed to run on a single server. As a result, it stores some data on the server's local file system. When running WordPress in a multi-server configuration, this creates a problem because there is inconsistency across web servers. For example, if a user uploads a new image, it is only stored on one of the servers.

This demonstrates why we need to improve the default WordPress running configuration to move important data to shared storage. The best practice architecture will, therefore, have a database as a separate layer outside the web server and will make use of shared storage to store user uploads, themes, and plugins.

Shared Storage (Amazon S3 and Amazon EFS)

By default, WordPress stores user uploads on the local file system and so isn't stateless. Therefore, we need to move the WordPress installation and all user customizations (such as configuration, plugins, themes, and user-generated uploads) into a shared data platform to help reduce load on the web servers and to make the web tier stateless.

Amazon Elastic File System (Amazon EFS) provides scalable network file systems for use with EC2 instances.[23] EFS file systems are distributed across an unconstrained number of storage servers, enabling file systems to grow elastically and allowing massively parallel access from EC2 instances. The distributed design of Amazon EFS avoids the bottlenecks and constraints inherent to traditional file servers.

By moving the entire WordPress installation directory onto an EFS file system and mounting it into each of your EC2 instances when they boot, your WordPress site and all its data will automatically be stored on a distributed file system that isn't dependent on any one EC2 instance, making your web tier completely stateless. The benefit of this architecture is that you don't need to install plugins and themes on each new instance launch, and you can significantly speed up the installation and recovery of WordPress instances. It is also easier to deploy changes to plugins and themes in WordPress, as outlined in the Deployment Considerations section of this document.

To ensure optimal performance of your website when running from an EFS file system, check the recommended configuration settings for Amazon EFS and OPcache on the AWS Reference Architecture for WordPress.[24]

You also have the option to offload all static assets, such as image, CSS, and JavaScript files, to an S3 bucket with CloudFront caching in front. The mechanism for doing this in a multi-server architecture is exactly the same as for a single-server architecture, as discussed in the Static Content section of this whitepaper. The benefits are the same as in the single-server architecture—you can offload the work associated with serving your static assets to Amazon S3 and CloudFront, thereby allowing your web servers to focus on generating dynamic content only and serve more user requests per web server.

Data Tier (Amazon Aurora and Amazon ElastiCache)

With the WordPress installation stored on a distributed, scalable, shared network file system, and static assets being served from Amazon S3, we can now focus our attention on the remaining stateful component: the database. As with the storage tier, the database should not be reliant on any single server, so we can't host it on one of the web servers. Instead, we will host the WordPress database on Amazon Aurora.

Amazon Aurora is a MySQL and PostgreSQL compatible relational database built for the cloud that combines the performance and availability of high-end commercial databases with the simplicity and cost-effectiveness of open source databases. Aurora MySQL increases MySQL performance and availability by tightly integrating the database engine with a purpose-built distributed storage system, backed by SSD. It is fault-tolerant and self-healing, replicates six copies of your data across three Availability Zones, is designed for greater than 99.99% availability, and continuously backs up your data in Amazon S3. Amazon Aurora is designed to automatically detect database crashes and restart without the need for crash recovery or to rebuild the database cache.

Amazon Aurora provides a number of instances types to suit different application profiles, including memory-optimized and burstable instances.[25] To improve the performance of your database you can select a large instance type to provide more CPU and memory resources.

Amazon Aurora automatically handles failover between the primary instance and Aurora Replicas so that your applications can resume database operations as quickly as possible without manual administrative intervention. Failover typically takes less than 30 seconds.

After you have created at least one Aurora Replica, connect to your primary instance using the cluster endpoint to allow your application to automatically fail over in the event the primary instance fails. You can create up to 15 low-latency read replicas across three Availability Zones.

As your database scales, your database cache will also need to scale. As discussed previously in the Database Caching section, ElastiCache has features to scale the cache across multiple nodes in an ElastiCache cluster, and across multiple Availability Zones in a Region for improved availability. As you scale your ElastiCache cluster, you should ensure that you configure your caching plugin to connect using the configuration endpoint so that WordPress can use new cluster nodes as they are added and stop using old cluster nodes as they are removed. You will also need to set up your web servers to use the ElastiCache Cluster Client for PHP and update your AMI to store this change.[26]

Conclusion

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.

Contributors

The following individuals and organizations contributed to this document:

- Paul Lewis, Solutions Architect, Amazon Web Services
- Ronan Guilfoyle, Solutions Architect, Amazon Web Services
- Andreas Chatzakis, Solutions Architect Manager, Amazon Web Services

Document Revisions

Date	Description
February 2018	Updated to clarify Amazon Aurora product messaging.
December 2017	Updated to include AWS services launched since first publication.
December 2014	First publication.

Appendix A: CloudFront Configuration

To get optimal performance and efficiency when using Amazon CloudFront with your WordPress website, it's important to configure the website correctly for the different types of content being served.

Origins and Behaviors

An origin is a location where CloudFront sends requests for content that it distributes through the edge locations.[27] You can point CloudFront to the location where you are storing your static content (in the reference architecture above this is Amazon S3) using an Amazon S3 origin. You can point CloudFront to your dynamic content (in the single-server deployment above this is a Lightsail instance, or in the reference architecture above this is the Application Load Balancer) using a custom origin. When you use Amazon S3 as an origin for your distribution, you need to use a bucket policy to make the content publically accessible.[28]

Behaviors allow you to set rules that govern how CloudFront caches your content, and, in turn, determine how effective the cache will be.[29] Behaviors allow you to control the protocol and HTTP methods your website is accessible by. They also allow you to control whether to pass HTTP headers, cookies, or query strings to your backend (and, if so, which ones). Behaviors can apply to specific URL path patterns.

IAM User Creation

You will need to create an AWS Identity and Access Management (IAM) user for the WordPress plugin to store static assets in Amazon S3. You can follow this guide for Creating an IAM User in Your AWS Account.[30]

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.[31]

Take a note of the user security credentials and store them in a secure manner – you will need them later.

S3 Bucket Creation

You will also need to create an Amazon S3 bucket in the Region of your choice. You can follow this guide for Creating an Amazon S3 Bucket.[32] Enable static website hosting for the bucket by following the guide for Configuring a Bucket for Website Hosting.[33]

Create an IAM policy to provide the IAM user created previously to access the specified S3 bucket, and attach the policy to the IAM user. You can follow this guide for Managing IAM Policies to create the following policy:[34]

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "Stmt1389783689000",
   "Effect": "Allow",
   "Principal": "*",
   "Action": [
    "s3:DeleteObject",
    "s3:GetObject",
    "s3:GetObjectAcl",
    "s3:ListBucket",
    "s3:PutObject",
    "s3:PutObjectAcl"
   ],
   "Resource": [
    "arn:aws:s3:::wp-demo",
    "arn:aws:s3:::wp-demo/*"
   ]
  }
 ]
}
```

CloudFront Distribution Creation

Next you will need to create a CloudFront web distribution by following the Task List for Creating a Web Distribution guide.[35] When you create a new CloudFront distribution you will automatically create a default origin and behavior, which you will use for dynamic content. We will also create four additional behaviors to further customize the way both static and dynamic requests are treated. The table below summarizes the configuration properties for the five behaviors.

	Static	Dynamic (admin)	Dynamic (front end)
Paths	wp-content/* wp-includes/*	wp-admin/* wp-login.php	default (*)
Protocols	HTTP and HTTPS	Redirect to HTTPS	HTTP and HTTPS
HTTP methods	GET, HEAD	ALL	ALL
HTTP headers	NONE	ALL	Host CloudFront-Forwarded-Proto CloudFront-Is-Desktop-Viewer

	Static	Dynamic (admin)	Dynamic (front end)
			CloudFront-Is-Mobile-Viewer CloudFront-Is-Tablet-Viewer
Cookies	NONE	ALL	comment_* wordpress_* wp-settings-*
Query strings	YES (invalidation)	YES	YES

Table 1: Summary of configuration property for CloudFront behaviors

For the default behavior we recommend the following configuration:

- Allow the Origin Protocol Policy to **Match Viewer**, so that if viewers connect to CloudFront using HTTPS, CloudFront will connect to your origin using HTTPS as well, achieving end-to-end encryption. Note that this requires you install a trusted SSL certificate on the load balancer as explained in the Amazon CloudFront Developer Guide.[36]
- Allow all HTTP methods since the dynamic portions of the website require both GET and POST requests (for example, to support POST for the comment submission forms).
- Forward only the cookies that vary the WordPress output, for example, `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.
- Forward only the HTTP headers that affect the output of WordPress, for example, **Host**, **CloudFront-Forwarded-Proto**, **CloudFront-is-Desktop-Viewer**, **CloudFront-is-Mobile-Viewer**, and **CloudFront-is-Tablet-Viewer**. The **Host** header allows multiple WordPress websites to be hosted on the same origin; the **CloudFront-Forwarded-Proto** header allows different versions of pages to be cached depending on whether they are accessed via HTTP or HTTPS; and the **CloudFront-is-Desktop-Viewer**, **CloudFront-is-Mobile-Viewer**, **CloudFront-is-Tablet-Viewer** headers allow you to customize the output of your themes based on the end user's device type.
- Forward and cache based on all query strings because WordPress relies on these, and they can be used to invalidate cached objects.

If you wish you serve your website under a custom domain name (that is, not *.cloudfront.net), then you should enter the appropriate URIs under **Alternate Domain Names** in the Distribution Settings. In this case you will also need an SSL certificate for your custom domain name. SSL certificates can be requested for free via the AWS Certificate Manager and configured against a CloudFront distribution.[37]

You will now need to create two more cache behaviors for dynamic content: one for the login page (path pattern: `wp-login.php`) and one for the admin dashboard (path pattern: `wp-admin/*`). These two behaviors have the exact same settings, as follows:

- Enforce a Viewer Protocol Policy of HTTPS Only.
- Allow all HTTP methods.
- Cache based on all HTTP headers.
- Forward all cookies.
- Forward and cache based on all query strings.

The reason behind this configuration is that this section of the website is highly personalized and typically has just a few users, so caching efficiency isn't a primary concern here. The aim is to keep the configuration simple to ensure maximum compatibility with any installed plugins by passing all cookies and headers to the origin.

At this point, the CloudFront configuration for dynamic content is complete. However, before adding the configuration for static content the WordPress website must be configured appropriately, which is covered next.

WordPress Plugin Installation and Configuration

In this example, the W3 Total Cache (W3TC) plugin is used to store static assets on Amazon S3. However, there are other plugins available with similar capabilities. If you wish to use an alternative you can adapt the steps below accordingly. The steps below only refer to features or settings relevant to this example. A detailed description of all settings is beyond the scope of this document. Please refer to the W3 Total Cache plugin page at wordpress.org for more information.

Install and activate the W3TC plugin from the WordPress admin panel. Browse to the General Settings section of the plugin's configuration, and ensure that both **Browser Cache** and **CDN** are enabled. From the drop-down list in the CDN configuration, select **Origin Push: Amazon CloudFront** (this will have Amazon S3 as its origin).

Browse to the Browser Cache section of the plugin's configuration and enable the **expires**, **cache control**, and **entity tag (ETag)** 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 whenever any settings are changed.

Browse to the CDN section of the plugin's configuration and enter the security credentials of the IAM user you created earlier, as well as the name of the S3 bucket. If you will be serving your website via the CloudFront URL, enter the distribution domain name in the relevant box. Otherwise, enter one or more CNAMEs for your custom domain name(s).

Finally, you must export the media library and upload the wp-includes, theme files, and custom files to Amazon S3 using the W3TC plugin. These upload functions are available in the **General** section of the **CDN** configuration page.

Static Origin Creation

Now that the static files are stored on Amazon S3, go back to the 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 S3 bucket you created for that purpose. Then create two more cache behaviors, one for each of the two folders (`wp-content` and `wp-includes`) that should use the S3 origin rather than the default origin for dynamic content. Configure both in the same manner:

- Serve HTTP GET 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.[38]

Note: After adding the static origin behaviors to your CloudFront distribution, check the order to ensure the behaviors for `wp-admin/*` and `wp-login.php` have higher precedence than the behaviors for static content. Otherwise, you may see strange behavior when accessing your admin panel.

Appendix B: Backup and Recovery

Recovering from failure in AWS is faster and easier to do compared to traditional hosting environments. For example, you can launch a replacement instance in minutes in response to a hardware failure, or you can make use of automated failover in many of our managed services to negate the impact of a reboot due to routine maintenance.

However, you still need to ensure you are backing up the right data in order to successfully recover it. In order to re-establish the availability of a WordPress website, you must be able to recover the following components:

- Operating system (OS) and services installation and configuration (Apache, MySQL, etc.)
- WordPress application code and configuration
- WordPress themes and plugins
- Uploads (for example, 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.[39]

We have previously discussed making use of Lightsail snapshots to protect all data stored on the instance's local storage. If your WordPress website runs off the Lightsail instance only, regular Lightsail snapshots should be sufficient for you to recover your WordPress website in its entirety. However, you will still lose any changes applied to your website since the last snapshot was taken if you do restore from a snapshot.

In a multi-server deployment you will need to back up each of the components discussed earlier using different mechanisms. Each component may have a different requirement for backup frequency, for example, the OS and WordPress installation and configuration will change much less frequently than user-generated content and, therefore, can be backed up less frequently without losing data in the event of a recovery.

To back up the OS and services installation and configuration, and the WordPress application code and configuration, you can create an AMI of a properly configured EC2 instance. AMIs can serve two purposes: to act as a backup of instance state, and to act as a template when launching new instances.

To back up the WordPress application code and configuration, you will need to make use of AMIs and also Aurora backups (more to follow).

To back up the WordPress themes and plugins installed on your website you need to back up the Amazon S3 bucket or the Amazon EFS file system they are stored on.

- For themes and plugins stored in an S3 bucket, you can enable Cross-Region Replication so that all objects uploaded to your primary bucket are automatically replicated to your backup bucket in another AWS Region.[40] Cross-Region Replication requires that Versioning is enabled on both your source and destination buckets, which provides you with an additional layer of protection and allows you to revert to a previous version of any given object in your bucket.[41]
- For themes and plugins stored on an EFS file system, you can create an AWS Data Pipeline to copy data from your production EFS file system to another EFS file system, as outlined in the documentation page Back Up an EFS File System.[42] You can also back up an EFS file system using any backup application you are already familiar with.
- To back up user uploads you should follow the steps outlined earlier for backing up the WordPress themes and plugins.

- To back up database content you need to make use of Aurora backup.[43] Aurora backs up your cluster volume automatically and retains restore data for the length of the *backup retention period*. Aurora backups are continuous and incremental so you can quickly restore to any point within the backup retention period. No performance impact or interruption of database service occurs as backup data is being written. You can specify a backup retention period from 1 to 35 days. You can also create manual database snapshots, which will persist until you delete them. Manual database snapshots are useful for long-term backups and archiving.[44]

Appendix C: Deploying New Plugins and Themes

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

Any time you are making a structural change to your WordPress installation there is a certain risk of introducing unforeseen problems. At the very least you should take a backup of your application code, configuration, and database before applying any significant change (such as installing a new plugin). For websites of business or other value, you should certainly be testing those changes in 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. After you are done with your tests, you can simply tear down your test environment and stop paying for those resources. Later we discuss some WordPress-specific considerations. For more information on development and test best practices on AWS, see the Development and Test on Amazon Web Services whitepaper.[45]

Some plugins will write configuration information to the `wp_options` database table (or introduce database schema changes), while others will create configuration files in the WordPress installation directory. Because we have moved the database and storage to shared platforms, these changes will be immediately available to all your running instances without any further effort on your part.

When deploying new themes in WordPress, a little more effort may be required. If you are only making use of Amazon EFS to store all your WordPress installation files, then new themes will be immediately available to all running instances. However, if you are offloading static content to Amazon S3, you will need to process a copy of these 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 theme 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 this is by including some sort of version identifier in your object. This identifier might be a 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.

Notes

[1] https://amazonlightsail.com/

[2] https://aws.amazon.com/ec2/

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

[4] https://amazonlightsail.com/pricing/

[5] https://lightsail.aws.amazon.com/ls/docs/how-to/article/lightsail-how-to-create-instance-from-snapshot

[6] https://lightsail.aws.amazon.com/ls/docs/how-to/article/lightsail-how-to-create-larger-instance-from-snapshot-using-aws-cli

[7] https://lightsail.aws.amazon.com/ls/docs/getting-started/article/getting-started-with-wordpress-and-lightsail

[8] https://lightsail.aws.amazon.com/ls/docs/overview/article/understanding-instance-snapshots-in-amazon-lightsail

[9] https://github.com/awslabs/lightsail-auto-snapshots

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

[11] https://aws.amazon.com/cloudfront/

[12] https://aws.amazon.com/cloudfront/details/#edge-locations

[13] https://aws.amazon.com/s3/

[14] https://memcached.org/

[15] https://aws.amazon.com/elasticache/

[16] https://docs.aws.amazon.com/AmazonElastiCache/latest/UserGuide/GettingStarted.ConnectToCacheNode.html

[17] https://lightsail.aws.amazon.com/ls/docs/how-to/article/lightsail-how-to-set-up-vpc-peering-with-aws-resources

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

[19] https://github.com/awslabs/aws-refarch-wordpress

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

[21] https://docs.aws.amazon.com/elasticloadbalancing/latest/application/target-group-health-checks.html

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

[23] https://aws.amazon.com/efs/details/

[24] https://github.com/awslabs/aws-refarch-wordpress#opcache

[25] https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.DBInstanceClass.html

[26] https://docs.aws.amazon.com/AmazonElastiCache/latest/UserGuide/Appendix.PHPAutoDiscoverySetup.html

[27] https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/DownloadDistS3AndCustomOrigins.html

[28] https://docs.aws.amazon.com/AmazonS3/latest/dev/WebsiteAccessPermissionsReqd.html

[29] https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/RequestAndResponseBehavior.html

[30] https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users_create.html

[31] http://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-ec2.html

[32] https://docs.aws.amazon.com/AmazonS3/latest/user-guide/create-bucket.html

[33] https://docs.aws.amazon.com/AmazonS3/latest/dev/HowDoIWebsiteConfiguration.html

[34] https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_manage.html

[35] https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/distribution-web-creating.html

[36] https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/using-https-cloudfront-to-custom-origin.html

[37] https://docs.aws.amazon.com/acm/latest/userguide/gs-acm-request.html

[38] https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/ReplacingObjects.html

[39] https://d0.awsstatic.com/whitepapers/Storage/Backup_and_Recovery_Approaches_Using_AWS.pdf

[40] https://docs.aws.amazon.com/AmazonS3/latest/dev/crr.html

[41] https://docs.aws.amazon.com/AmazonS3/latest/dev/Versioning.html

[42] https://docs.aws.amazon.com/efs/latest/ug/efs-backup.html

[43] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Managing.html#Aurora.Managing.Backups

[44] https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CreateSnapshot.html

[45] https://d0.awsstatic.com/whitepapers/aws-development-test-environments.pdf