

Highly Available AWS WordPress Application

Gokul Gopakumar

Abstract:

In this project, we explore how to build a highly scalable and fault-tolerant WordPress website using Amazon EC2, RDS, EFS, ASG, ALB and Amazon CloudFormation. We first create the subnets with different availability zones for the web application, storage and database. Next, we will configure a relational database service (RDS) instance to store our WordPress data, EFS storage to store the images and text files and set up auto-scaling to ensure our website can handle high traffic volumes and provide Health checks using ALBs.

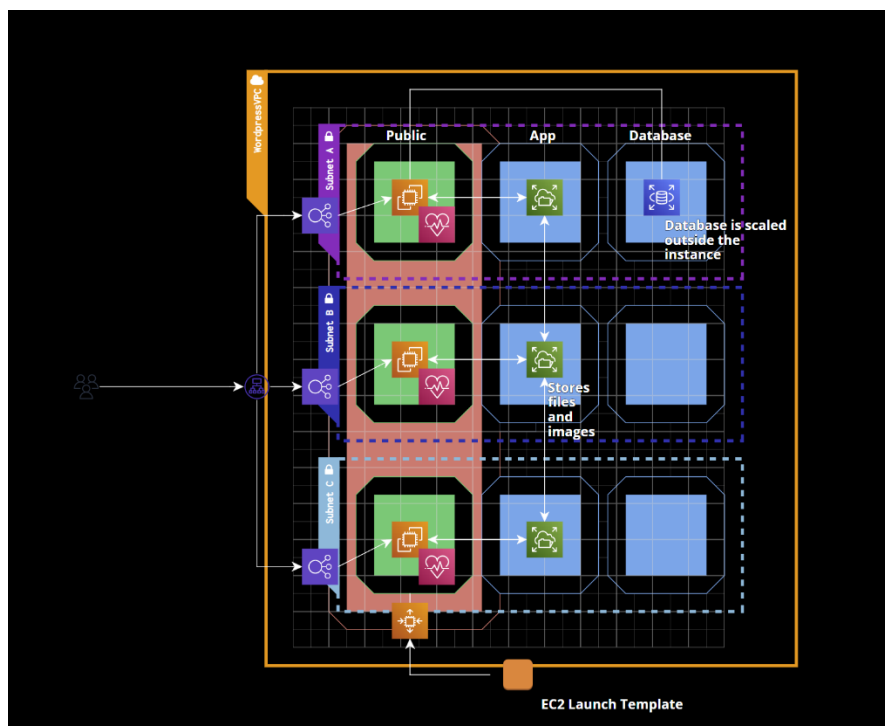
Introduction:

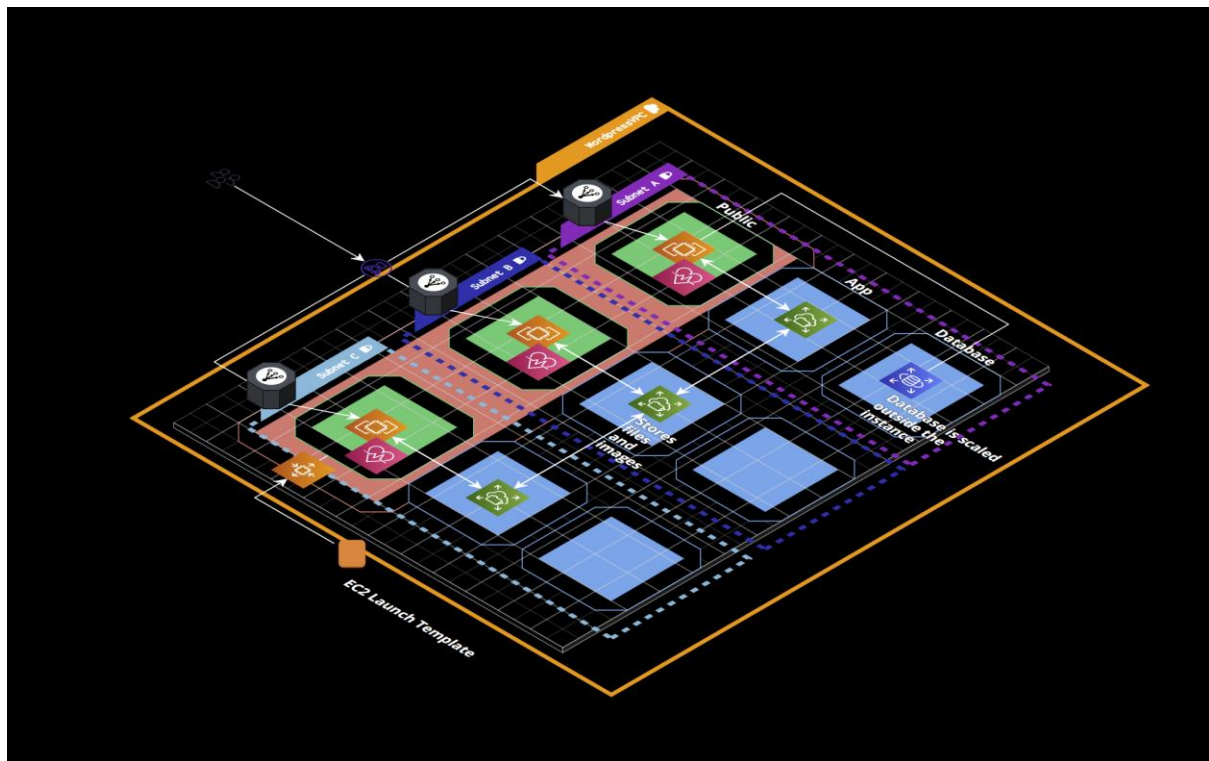
WordPress is one of the most popular Content Management Systems (CMS) in the world, powering more than 40% of all websites on the internet. However, as a website grows in popularity, it becomes increasingly challenging to maintain its performance and availability. Amazon RDS is a managed relational database service that simplifies database administration, while Amazon CloudFront is a global Content Delivery Network that accelerates the delivery of web content. This project will provide a step-by-step guide to building a highly scalable and fault-tolerant WordPress website using these AWS services.

Materials and Methods:

To complete this project, we will need an AWS account with the necessary permissions to create Elastic File System storages, RDS instances and other resources. We will also need to install the AWS Command Line Interface (CLI) to interact with AWS services from our local computer.

The architecture of the program is as follows:





Pic 1 & 2: Architecture of the application as a whole

We start off by creating the subnets with 3 sections which are: Public_web app (which is a public subnet which has connection to the internet) , Application and Database (which are private subnets). We create these subnets in 3 different AZ for each of the sections and start off with the public web app subnets.

Note : We have used the CloudFormation setup for the above from <https://console.aws.amazon.com/cloudformation/home?region=us-east-1#/stacks/quickcreate?templateURL=https://learn-cantrill-labs.s3.amazonaws.com/aws-elastic-wordpress-evolution/A4LVPC.yaml&stackName=A4LVPC> from Adrian Cantrill while learning for the Solutions Architect Associate Exam.

We first create an EC2 instance into the public web subnet and configure it to run a Wordpress file and setup the manual Wordpress build with a configuration of MariaDB as the database and the ephemeral storage for media. We then create a launch template for future automation of this process. (version 1)

Note: All the parameters needed for each of the processes such as DB username, Root passwords and others are stored in the Parameter store for ease of access in the later parts of the code

Now we can't really scale this architecture in the initial phase as the database and the application media are stored inside the instance itself which may lead to data loss.

Hence we start creating the RDS instance for which we select all subnets A,B & C which allows the RDS to decide on the subnet. We then create a MySQL database under the free tier and migrate the MariaDB database to the RDS by taking the backup of MariaDB and restoring it in RDS. (version2)

But the issue still pertains to the storage of media being temporary, so we create an Elastic File System and mount it to the Application section of the VPC. We connect the instance to the File system and copy the data into it. We further update the launch template (version 3).

Now to improve the architecture further, we added an Auto Scaling Group to the Public_Web section of the VPC to add and remove instances based on the load the application is facing.

For this we first create the load balancer which checks in real-time the state of the instance and depending on that information or the load incoming, it will be able to provide the necessary instructions to the Auto Scaling Group. We change the DNS of the application to the ELB DNS name and update this into the launch template as well. (version 4 – final)

We finally create the Auto Scaling Group providing the updated launch template and integrate it with the Load Balancer. We added certain to the Auto Scaling Group such that when cpu usage is above 40%, an instance is created to manage the load; else it will revert back to the desired number of instances which in this case is 1.

Implementation:

My parameters				
Public parameters				
Settings				
My parameters				
Q Search				
<input type="checkbox"/>	Name	Tier	Type	
<input type="checkbox"/>	/A4L/Wordpress/ALB-DNSname	Standard	String	
<input type="checkbox"/>	/A4L/Wordpress/DBEndpoint	Standard	String	
<input type="checkbox"/>	/A4L/Wordpress/DBName	Standard	String	
<input type="checkbox"/>	/A4L/Wordpress/DBPassword	Standard	SecureString	
<input type="checkbox"/>	/A4L/Wordpress/DBRootPassword	Standard	SecureString	
<input type="checkbox"/>	/A4L/Wordpress/DBUser	Standard	String	
<input type="checkbox"/>	/A4L/Wordpress/EFSfs-id	Standard	String	
<input type="checkbox"/>	CFN-CWAgentConfig-vFLZOpdzdujd	Standard	String	

Pic 3 : All the parameters used throughout the creation of final launch instance (under /A4L)

EC2 > Launch templates > Wordpress-temp

Wordpress-temp (lt-0a4b624ad1ae58208)

Actions Delete template

Launch template details

Launch template ID
lt-0a4b624ad1ae58208

Launch template name
Wordpress-temp

Default version
4

Owner
_Admin_General

Details Versions Template tags

Versions (4)

Actions Delete template version

< 1 > ⌕

	Version	Default version	Description	Creation time		AMI ID
<input type="radio"/>	4	Yes	ALB addition	2023-02-14T07:34:30.000Z		ami-0aa7d40eeae50c9a9
<input type="radio"/>	3	No	App-only-with-EFS from /A4L/Wordpres...	2023-02-13T20:02:38.000Z		ami-0aa7d40eeae50c9a9
<input type="radio"/>	2	No	Single-server-App-only	2023-02-13T19:37:46.000Z		ami-0aa7d40eeae50c9a9
<input type="radio"/>	1	No	Single Server DB and APP	2023-02-11T10:16:13.000Z	arn:aws:iam::329539500818:user/IAM...	ami-0aa7d40eeae50c9a9

Pic 4: All wordpress launch template versions

RDS > Databases > wordpress-blog

wordpress-blog

Modify Actions

Summary

DB identifier
wordpress-blog

CPU
2.78%

Status
Available

Class
db.t3.micro

Role
Instance

Current activity
0 Connections

Engine
MySQL Community

Region & AZ
us-east-1a

Connectivity & security Monitoring Logs & events Configuration Maintenance & backups Tags

Instance

Configuration

DB instance ID
wordpress-blog

Engine version
8.0.28

DB name
a4lwordpressdb

License model
General Public License

Option groups
default:mysql-8-0 In sync

Amazon Resource Name (ARN)
arn:aws:rds:us-east-1:329539500818:db:wordpress-blog

Resource ID
db-G6J2JCMFNFTRHQWR6LUTNRZWSXA

Created time
February 13, 2023, 22:52 (UTC+04:00)

Instance class

Instance class
db.t3.micro

vCPU
2

RAM
1 GB

Availability

Master username
a4lwordpressuser

Master password

IAM DB authentication
Not enabled

Multi-AZ
No

Secondary Zone

Storage

Encryption
Enabled

AWS KMS key
aws/rds

Storage type
General Purpose SSD (gp2)

Storage
20 GiB

Provisioned IOPS
-

Storage throughput
-

Storage autoscaling
Enabled

Maximum storage threshold
1000 GiB

Performance Insights

Performance Insights enabled
Turned off

DB instance password status

© 2023, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences

Pic 5 : RDS Cofiguration

Amazon EFS

BLOG-WORDPRESS-CONTENT (fs-00dba43c45ad6d677)

DeleteAttach

General

Edit

Performance mode

General Purpose

Throughput mode

Bursting

Lifecycle management

Transition into IA: 30 day(s) since last access

Transition out of IA: None

Availability zone

Standard

Automatic backups

Enabled

Encrypted

No

File system state

Available

DNS name

fs-00dba43c45ad6d677.efs.us-east-1.amazonaws.com

Metered size

Monitoring

Tags

File system policy

Access points

Network

Replication

Metered size

Total size

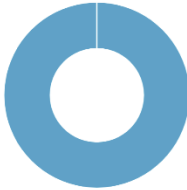
16.25 MiB

Size in Standard / One Zone

16.25 MiB (100%)

Size in Standard-IA / One Zone-IA

0 Bytes (0%)



Size in Standard / One Zone

Size in Standard-IA / One Zone-IA

Pic 6: EFS for the Wordpress media storage

EC2 > Load balancers > BloggerALB

BloggerALB

RefreshActions

Details

Load balancer type

Application

IP address type

IPv4

Date created

February 14, 2023, 11:24 (UTC+04:00)

DNS name

BloggerALB-354436178.us-east-1.elb.amazonaws.com (A Record)

Scheme

Internet-facing

Status

Active

Availability Zones

subnet-06a7b5a5afcd9c66 us-east-1b (use1-az4)
subnet-089f28aa1b2723ade us-east-1a (use1-az2)
subnet-0ef6b59ba80859a0 us-east-1c (use1-az5)

VPC

vpc-07edf1eba3076e5d2

Hosted zone

Z35XDOTRQ7X7K

Listeners

Network mapping

Security

Monitoring

Integrations

Attributes

Tags

Network mapping

Targets in the listed zones and subnets are available for traffic from the load balancer using the IP addresses shown.

VPC

vpc-07edf1eba3076e5d2

IPv4: 10.16.0.0/16

IPv6: 2600:1f10:47ee:ad00::/56

IP address type

IPv4

Mappings

Selecting two or more Availability Zones and corresponding subnets increases the fault tolerance of your applications.

Zone	Subnet	IPv4 address	Private IPv4 address	IPv6 address
us-east-1b (use1-az4)	subnet-06a7b5a5afcd9c66	Assigned by AWS	Assigned from CIDR 10.16.112.0/20	Not applicable
us-east-1a (use1-az2)	subnet-089f28aa1b2723ade	Assigned by AWS	Assigned from CIDR 10.16.48.0/20	Not applicable
us-east-1c (use1-az5)	subnet-0ef6b59ba80859a0	Assigned by AWS	Assigned from CIDR 10.16.176.0/20	Not applicable

Pic 7: Load Balancer configured for the application

EC2 > Target groups > BloggerWordpressALBtg

BloggerWordpressALBtg

Details
 arn:aws:elasticloadbalancing:us-east-1:329539500818:targetgroup/BloggerWordpressALBtg/4c92f357212b3922

Target type Instance	Protocol : Port HTTP: 80	Protocol version HTTP1
IP address type IPv4	Load balancer None associated	

Total targets	Healthy	Unhealthy	Unused
2	0	0	1

Targets | Monitoring | Health checks | Attributes | Tags

Registered targets (2)

Filter resources by property or value

<input type="checkbox"/>	Instance ID	Name	Port	Zone	Health status
<input checked="" type="checkbox"/>	i-0cf2f94445f4d82fe	Wordpress-ASG	80	us-east-1a	draining
<input type="checkbox"/>	i-0127aaf690a7dea9d	Wordpress-ASG	80	us-east-1b	unused

Pic 8: Auto Scaling Group for the wordpress application

EC2 > Auto Scaling groups > WordpressASG

WordpressASG

Details | Activity | **Automatic scaling** | Instance management | Monitoring | Instance refresh

Scaling policies resize your Auto Scaling group to meet changes in demand. With reactive dynamic scaling policies, you can track specific CloudWatch metrics and take action when the CloudWatch alarm threshold is met. Use predictive scaling policies along with dynamic scaling policies in the following situations: when your application demand changes quickly, but with a recurring pattern, or when your EC2 instances require more time to initialize.

Dynamic scaling policies (2) Info

Actions Create dynamic scaling policy

LowCPU

Simple scaling

Enabled

WordpressLowCPU

breaches the alarm threshold: CPUUtilization < 40 for 1 consecutive periods of 300 seconds for the metric dimensions:
AutoScalingGroupName = WordpressASG

Remove 1 capacity units

300 seconds before allowing another scaling activity

highcpu

Simple scaling

Enabled

WordpressHighCPU

breaches the alarm threshold: CPUUtilization > 40 for 1 consecutive periods of 300 seconds for the metric dimensions:
AutoScalingGroupName = WordpressASG

Add 1 capacity units

300 seconds before allowing another scaling activity

Predictive scaling policies (0) Info

Evaluation period
Evaluation based on 2 days

Actions Create predictive scaling policy

Name	Metric pair	Forecast and scale	Recommendation	Chart	Availability impact	Cost impact
------	-------------	--------------------	----------------	-------	---------------------	-------------

Pic 9: ASG with the dynamic scaling conditions configured.

Session ID: IAM_Admin_General-00e729073229ab041

Instance ID: i-05bc8b6c257585ea8

```
[root@ip-10-16-189-238 ~]# stress -c 2 -v -t 3000
stress: info: [14061] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
stress: debug: [14061] using backoff sleep of 6000us
stress: debug: [14061] setting timeout to 3000s
stress: debug: [14061] --> hogcpu worker 2 [14062] forked
stress: debug: [14061] using backoff sleep of 3000us
stress: debug: [14061] setting timeout to 3000s
stress: debug: [14061] --> hogcpu worker 1 [14063] forked
```

Pic 10: Stress test the wordpress instance

EC2 > Auto Scaling groups > WordpressASG

WordpressASG

Details Activity Automatic scaling Instance management Monitoring Instance refresh

Activity notifications (0)

Filter notifications

Send to On Instance action

No notifications are currently specified

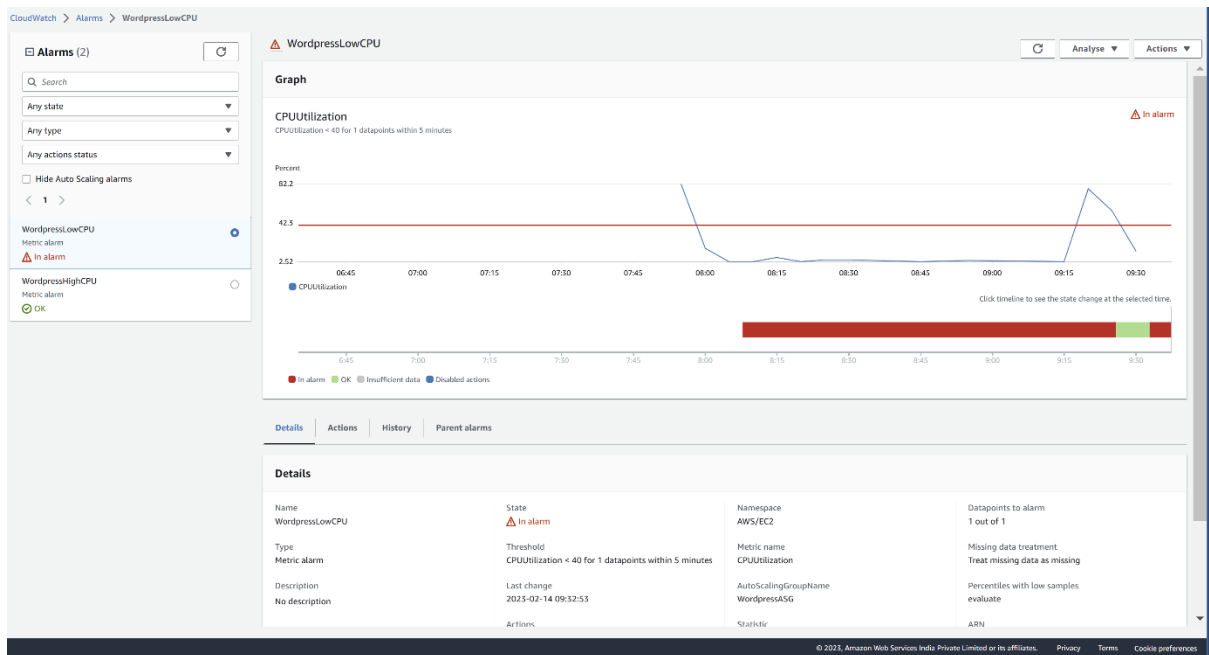
Create notification

Activity history (4)

Filter activity history

Status	Description	Cause	Start time	End time
WaitingForELBConnectionDraining	Terminating EC2 instance: i-05bc8b6c257585ea8 - Waiting For ELB Connection Draining.	At 2023-02-14T09:37:53Z a monitor alarm WordpressLowCPU in state ALARM triggered policy LowCPU changing the desired capacity from 3 to 2. At 2023-02-14T09:38:01Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 3 to 2. At 2023-02-14T09:38:01Z instance i-05bc8b6c257585ea8 was selected for termination.	2023 February 14, 01:38:01 PM +04:00	
Successful	Launching a new EC2 instance: i-0127aaf690a70ea9d	At 2023-02-14T09:31:44Z a monitor alarm WordpressHighCPU in state ALARM triggered policy highcpu changing the desired capacity from 2 to 3. At 2023-02-14T09:31:49Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 2 to 3.	2023 February 14, 01:31:51 PM +04:00	2023 February 14, 01:32:23 PM +04:00
Successful	Launching a new EC2 instance: i-0cf2f94445f4d82fe	At 2023-02-14T09:25:44Z a monitor alarm WordpressHighCPU in state ALARM triggered policy highcpu changing the desired capacity from 1 to 2. At 2023-02-14T09:25:46Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 1 to 2.	2023 February 14, 01:25:48 PM +04:00	2023 February 14, 01:26:06 PM +04:00
Successful	Launching a new EC2 instance: i-05bc8b6c257585ea8	At 2023-03-14T07:58:02Z a user request created an AutoScalingGroup changing the desired capacity from 0 to 1. At 2023-02-14T07:58:06Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 0 to 1.	2023 February 14, 11:58:09 AM +04:00	2023 February 14, 11:58:41 AM +04:00

Pic 11: The ASG activity page showing all the times the dynamic policies are called.



Pic 12: CPU utilization during the stress test and after stopping

The screenshot shows the AWS Management Console 'Instances' page. The instance 'Wordpress-ASG' is listed with a state of 'Running'. Below the instance list, the 'Instance: i-05bc8b6c257585ea8 (Wordpress-ASG)' details are shown, including the instance ID, public IPv4 address (3.81.69.96), private IPv4 addresses (10.16.189.238), and instance state.

Pic 13: Wordpress instance final result.

Results:

By following these steps, we will have successfully created a highly scalable and fault-tolerant WordPress website using AWS EFS, Amazon RDS, Auto Scaling Groups and Load Balancers. Our website will be able to handle high traffic volumes, and our data will be stored securely in an RDS instance. The health of the instance is also taken into account with the ASG which keeps the application active.

Some of the limitations found during the implementation of this architecture is:

- ⊗ The application only allows the tcp port 80 to pass through the resource policies and security groups which limits the feature of editing the wordpress, changing and downloading the template of the wordpress application using FTP server.
- ⊗ Since SSH server is not configured, there is another issue for wordpress to not connect and use various features available to wordpress.
- ⊗ The connection occurs through the HTTP port which is not a secure connection.

Conclusion:

In conclusion, AWS EFS, Amazon RDS, Auto Scaling Groups, Load Balancers and CloudFormation provide a powerful suite of services that can be used to build highly scalable and fault-tolerant WordPress websites. By following the steps outlined in this project, you will have the necessary skills to build your own WordPress website on AWS.