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.

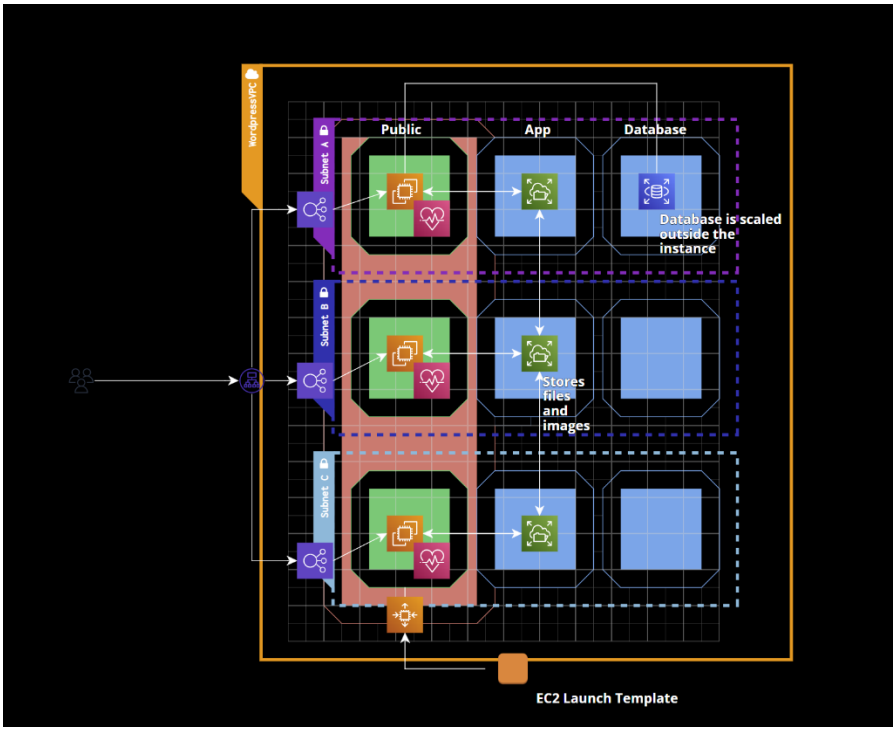
Introduction:

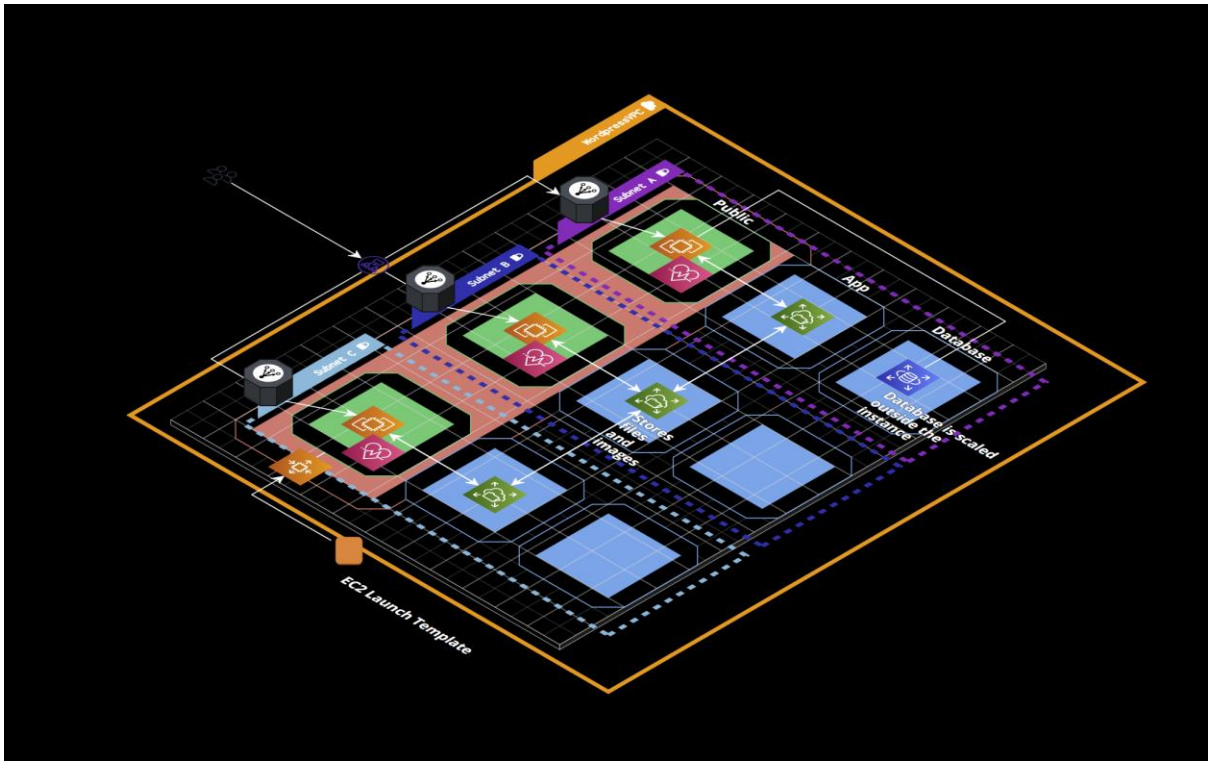
WordPress is one of the most popular Content Management Systems (CMS) in the world, powering more than 40% of all websites on the internet. However, as a website grows in popularity, it becomes increasingly challenging to maintain its performance and availability. Amazon RDS is a managed relational database 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 of 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 of 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)

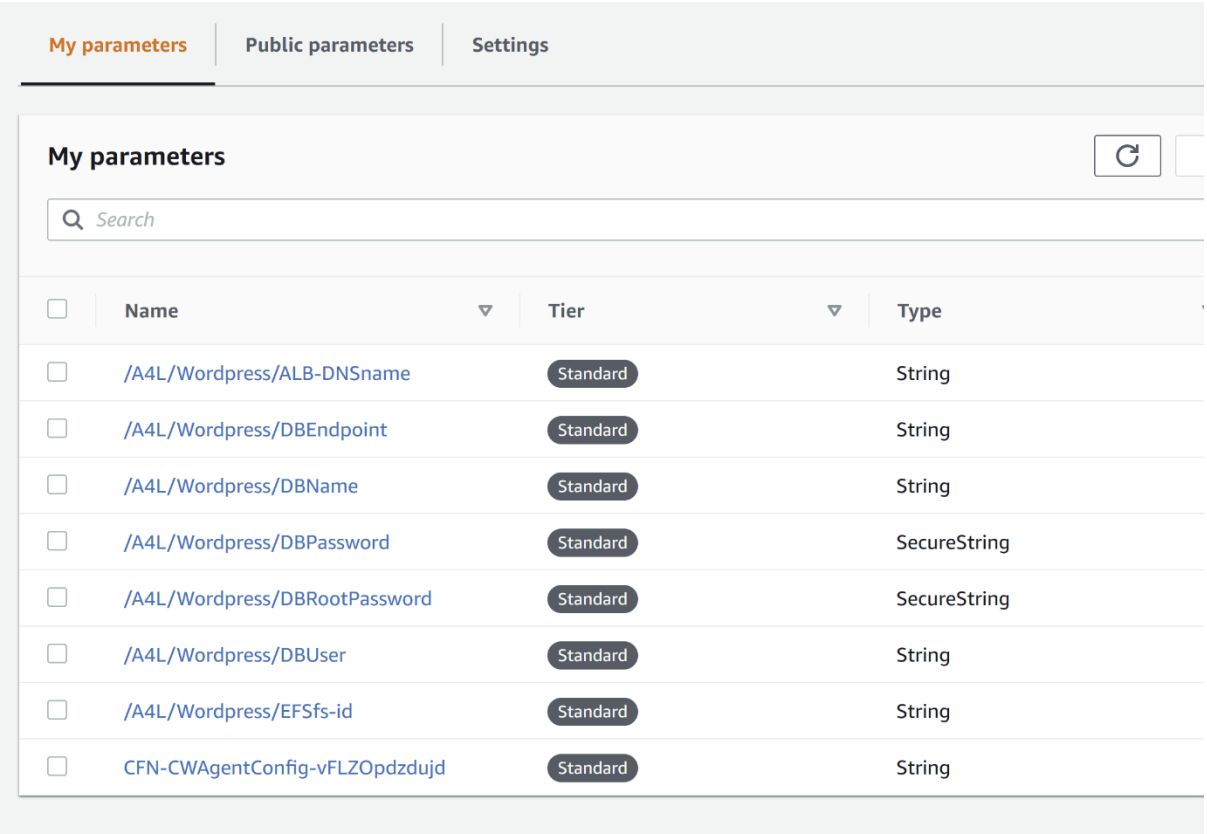
But the issue still pertains to the storage of media being temporary, so we create an Elastic File System and mount it to the Application section of the VPC. We connect the instance to the File system and copy the data into it. We further update the launch template (version 3).

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.

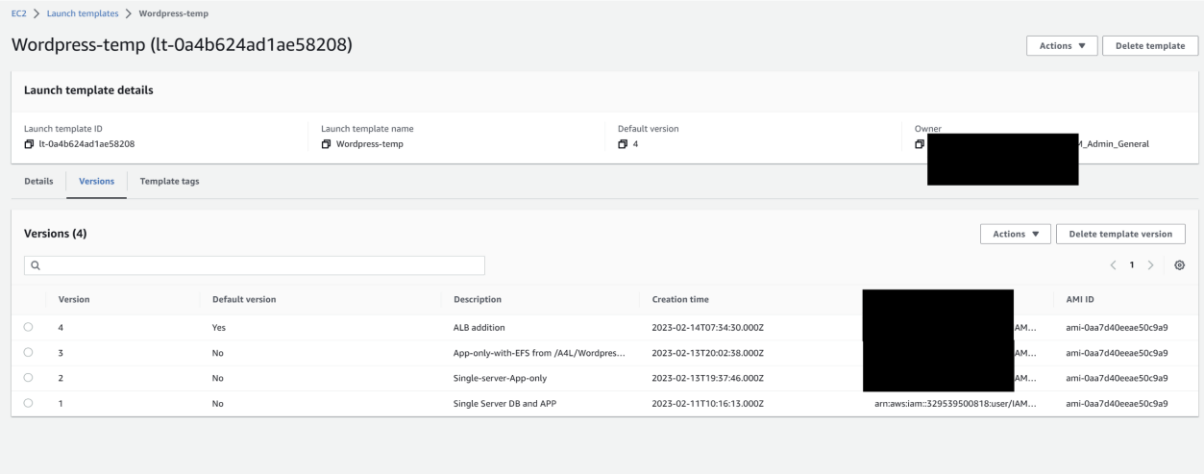
For this we first create the load balancer which checks in real-time the state of the instance and depending on that information or the load incoming, it will be able to provide the necessary instructions to the Auto Scaling Group. We change the DNS of the application to the ELB DNS name and update this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

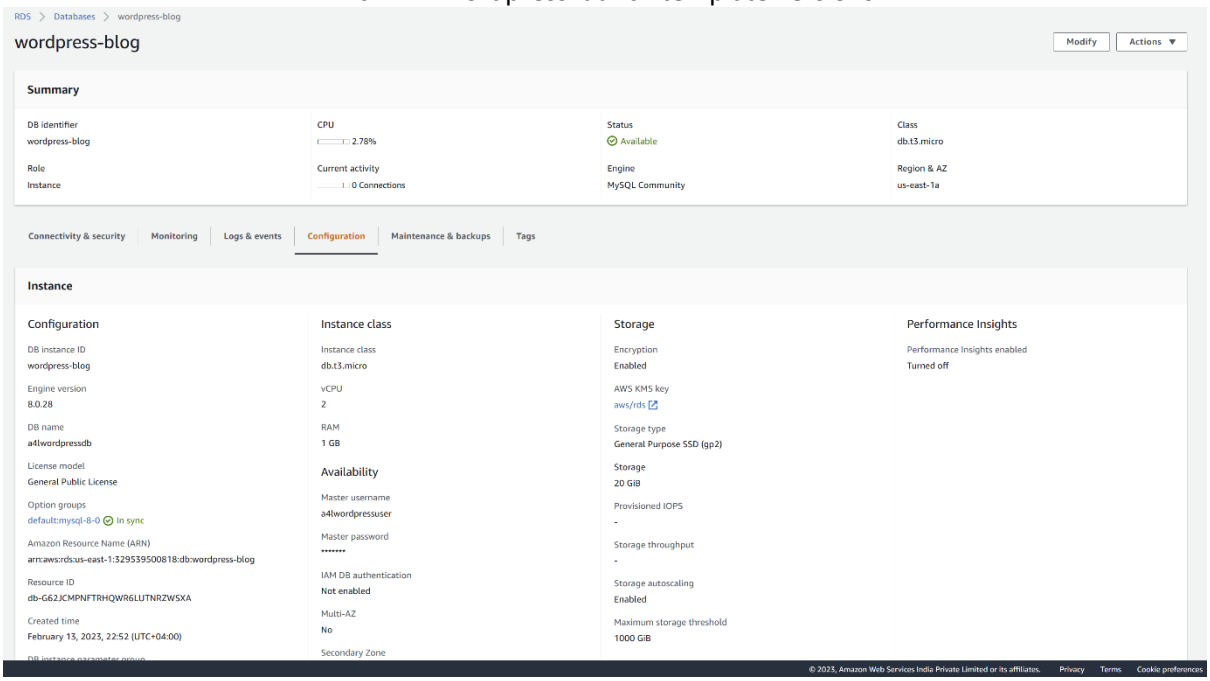
Implementation:



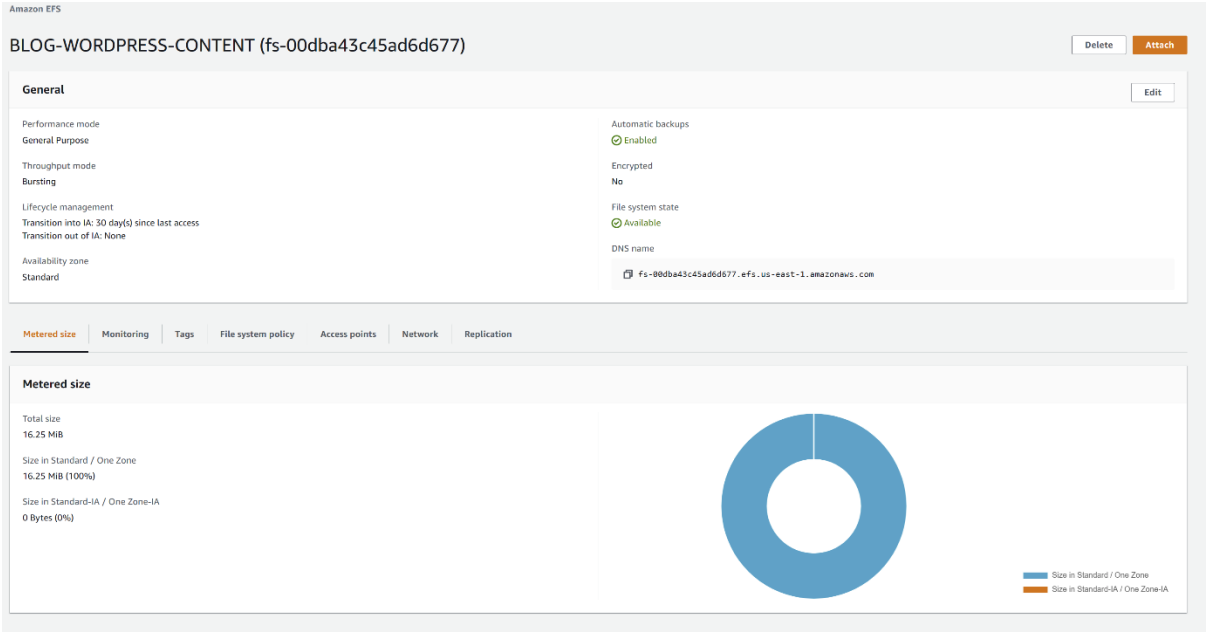
Pic 3 : All the parameters used throughout the creation of final launch instance (under /A4L)



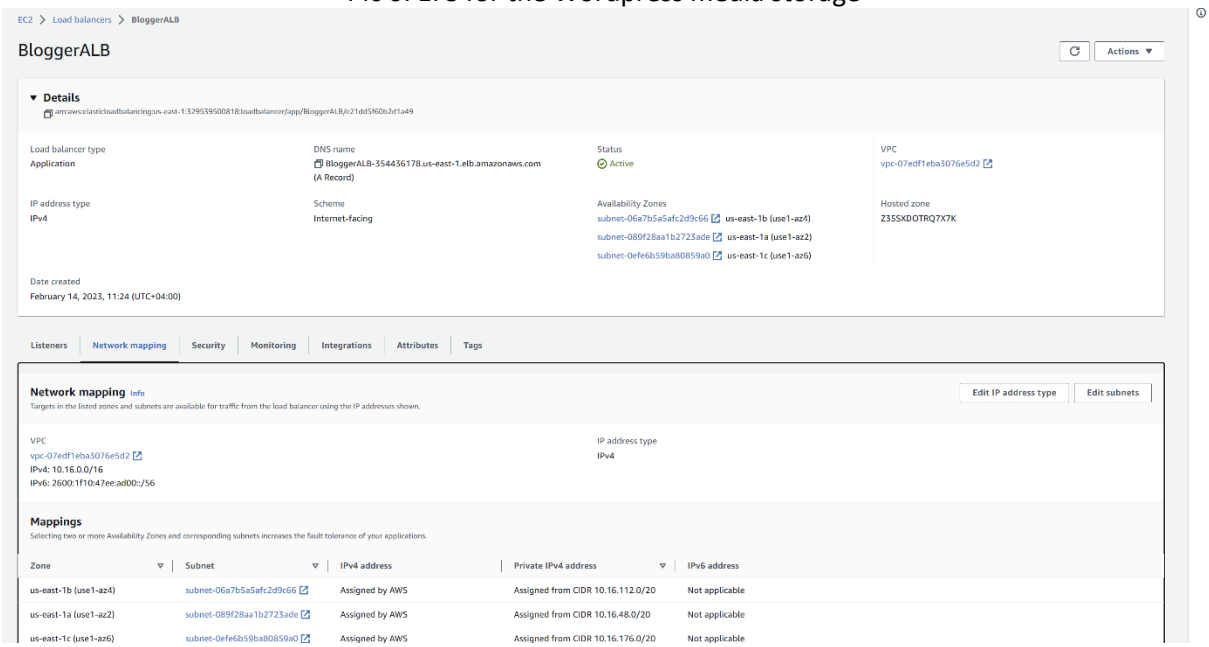
Pic 4: All wordpress launch template versions



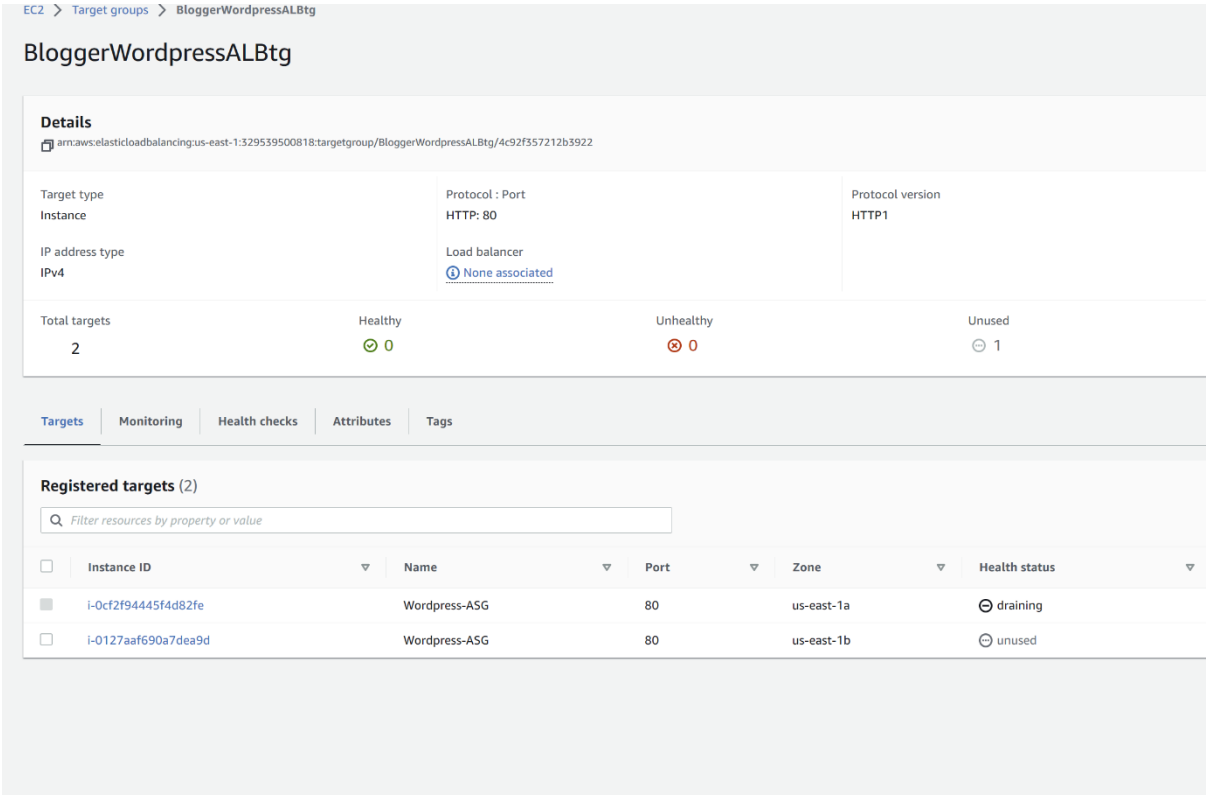
Pic 5 : RDS Cofiguration



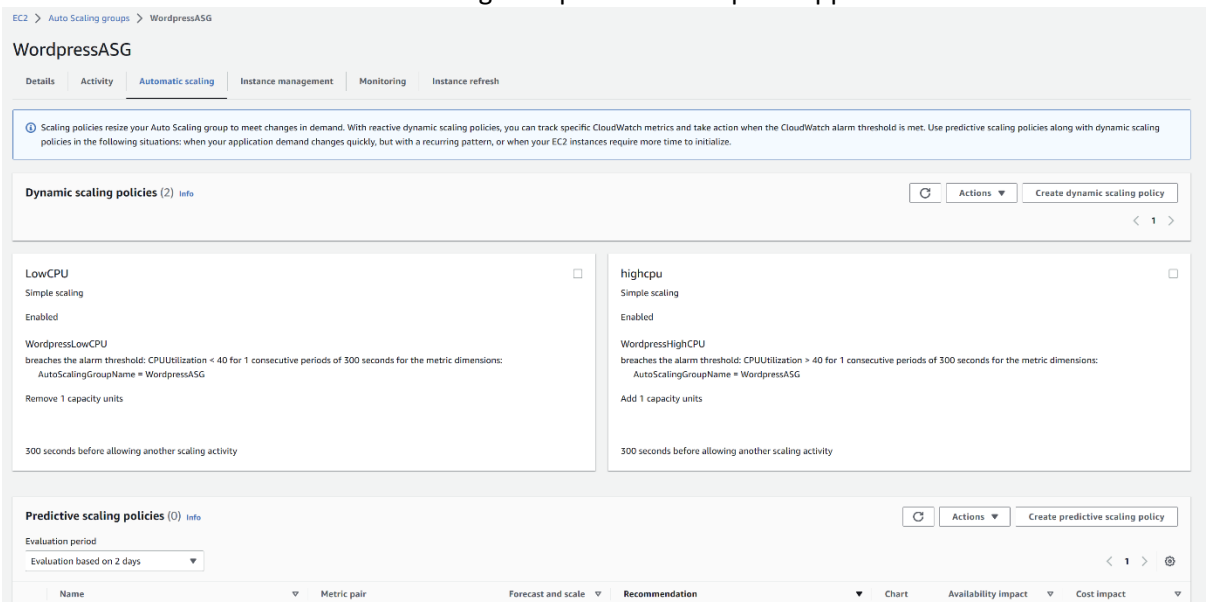
Pic 6: EFS for the Wordpress media storage



Pic 7: Load Balancer configured for the application



Pic 8: Auto Scaling Group for the wordpress application

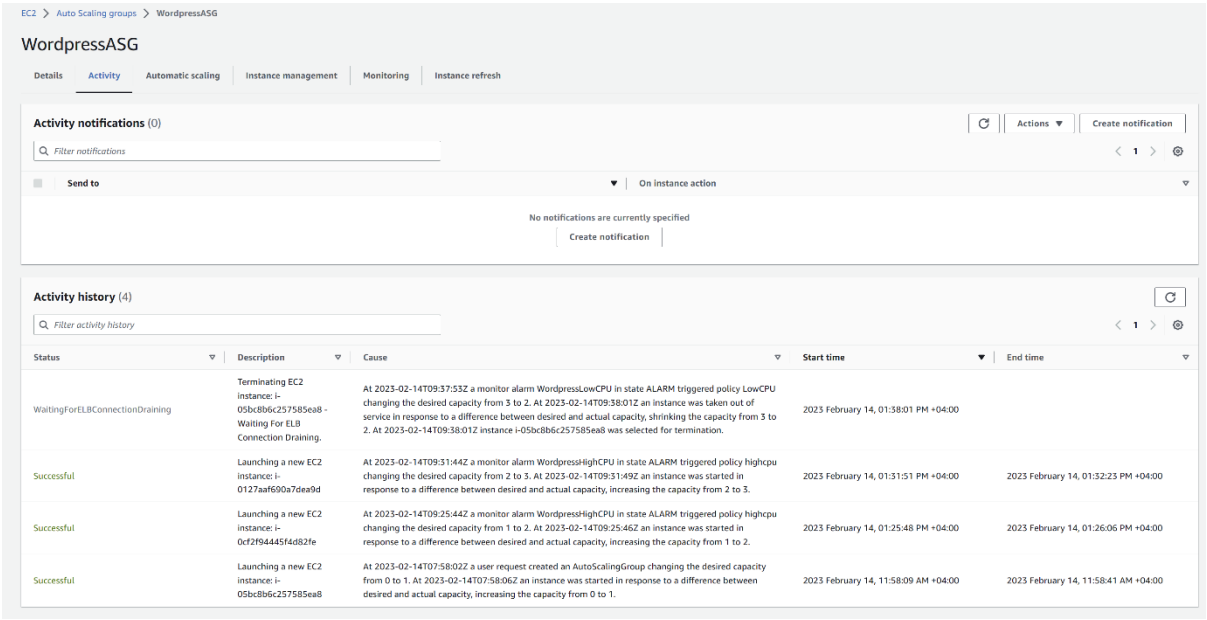


Pic 9: ASG with the dynamic scaling conditions configured.

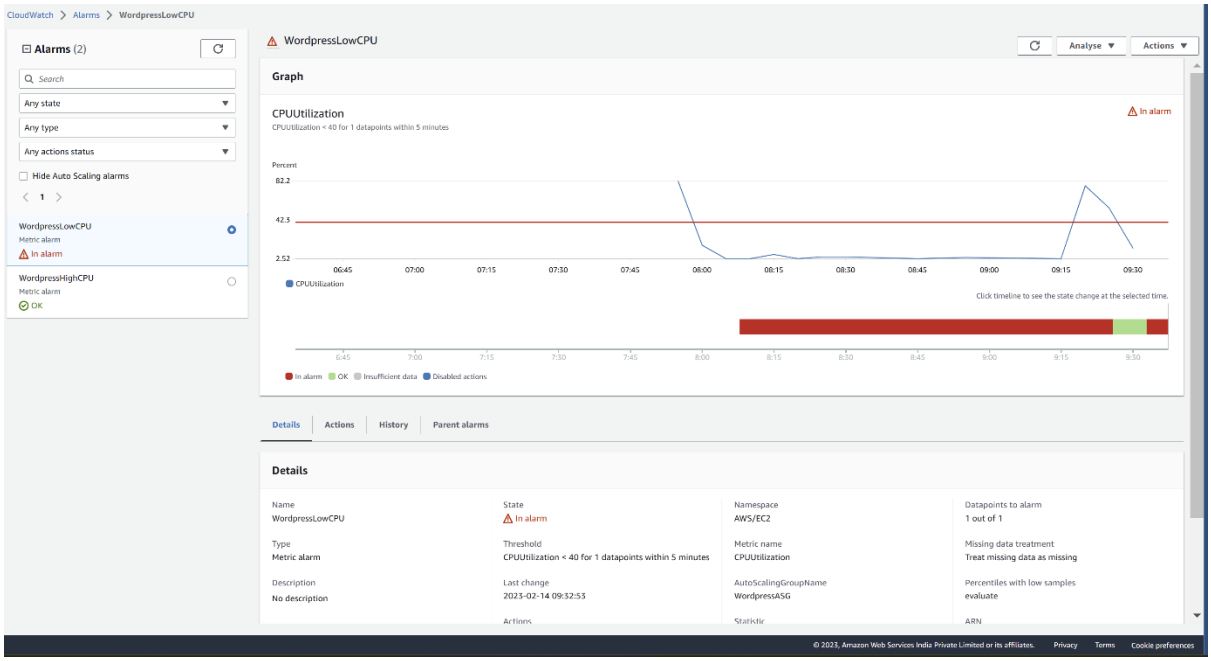
Session ID: IAM_Admin_General-00e729073229ab041 Instance ID: i-05bc8b6c257585ea8

```
[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: dbug: [14061] using backoff sleep of 6000us
stress: dbug: [14061] setting timeout to 3000s
stress: dbug: [14061] --> hogcpu worker 2 [14062] forked
stress: dbug: [14061] using backoff sleep of 3000us
stress: dbug: [14061] setting timeout to 3000s
stress: dbug: [14061] --> hogcpu worker 1 [14063] forked
```

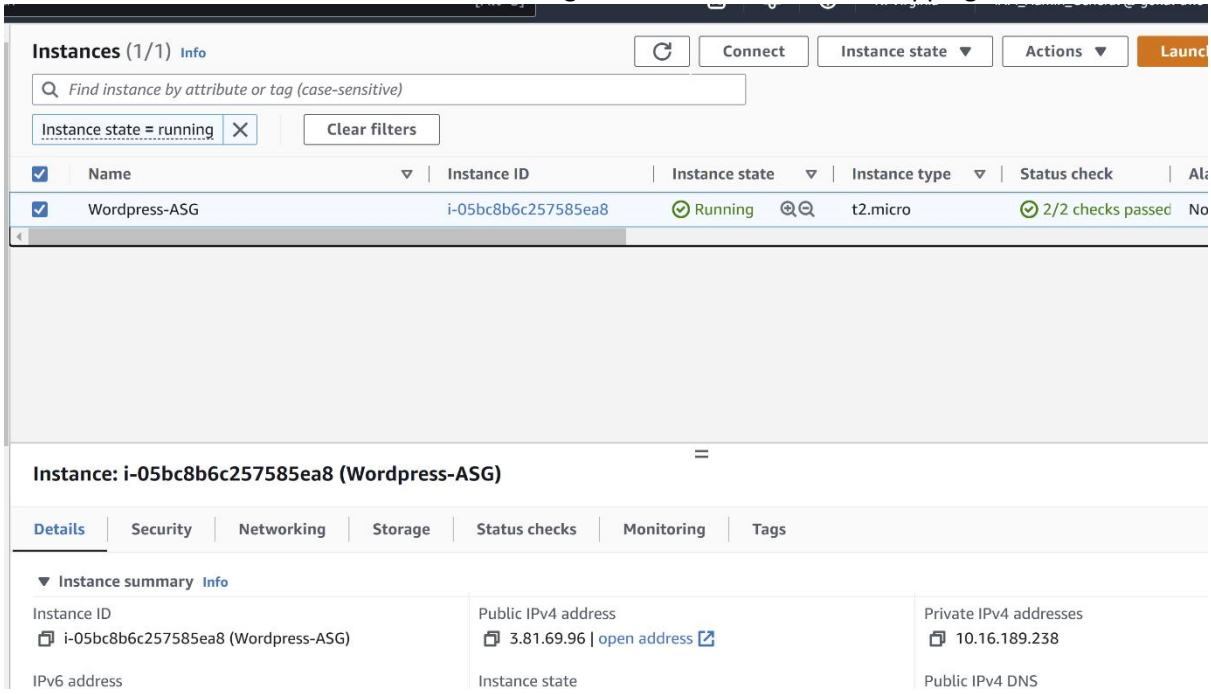
Pic 10: Stress test the wordpress instance



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



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.