SIT233 Final Project Report: High Availability WordPress Application

Jifeng Chen
Deakin University, School of Information Technology
Burwood,VIC 3125,Australia
s223313829@deakin.edu.au

***Abstract*—This document forms part of the assessment for SIT233 Cloud Computing. In the past few decades, there's been a growing trend that traditional businesses transit into e-commerce. To ensure this business achieves high availability of cloud services, we need to design and deploy a system architecture capable of fault tolerance, automatic recovery, and continuous service delivery. In this research, I designed a system architecture that includes utilizing multi-region and multi-availability zone deployments, automated load balancing, redundant data storage, and automated deployment. By adopting these strategies, we can significantly enhance the reliability and user experience of our application, reduce downtime, and ensure business continuity.**

***Index Terms*—Cloud Computing, Availability, Data Security**

I. Introduction

In the past few decades, traditional businesses have made a big shift to digitalization and e-commerce. [1] Lots of retailers moved to the online business, and some of them even provide fully virtual services. From the 1970s to the early 2000s, if some organizations or companies wish to provide online services, they must purchase and maintain their own networking facilities. Due to the high cost of the maintenance fee and the scaling of the business, the barrier to entering digitalization is usually high.

Using traditional hosting services, companies and organizations usually need to face the following challenges. Firstly, setting up the server cluster and maintaining the devices could be expensive. Especially for individual businessmen and small companies, the additional cost for these devices might be a waste as they don't have enough digital businesses. As for them, the barrier to entering the digital era is too high to surmount. Although this is not a big issue for large organizations and companies, the second drawback became their nightmares. That is for the fixed hardware facilities, it is hard to adapt to the constantly changing volume of business. It is quite impossible to increase computing capabilities in a short time, facing rapidly increasing businesses, such as Black Friday or Christmas Shipping. But when the business volume is decreased, the redundant hardware will become a kind of waste.

With the idea of "Pay as Needed" and lowering the barrier for small businesses, Amazon started to build cloud computing services, which is named the Amazon Web Services in 2006 [2]. On this platform, the users are allowed to create different element infrastructures like the virtual private cloud, database instances, and the EC2 server instance. The users are also allowed to create customized strategies for scaling in and scaling out their virtual cloud capacity automatically due to different policies. With better version control, unified logs, and many other features, cloud computing is replacing traditional hosting services step by step.

However, cloud computing is not fully reliable. Even big cloud computing providers like AWS will also have outages [3]. It could be a failure in a region or some availability zones. As the cloud's outages are highly influencing the service quality provided to cloud users, there is a growing need to develop a novel cloud system with high availability and automatic scaling policy. No matter whether this is an outage caused by the cloud provider or a hacker attack, the cloud system should be capable of keeping it available to users.

In this paper, it will introduce a highly available system, and the implementation of deploying the system on the Amazon Web Service. The contribution of this paper can be summarized as follows:

- A novel cloud system that has high availability and auto-scaling capability is designed, which is capable of applying to generic cloud application scenarios.
- A deployment of the cloud system on the Amazon Web Service is implemented. As an example, I have deployed a highly available WordPress application.
- Automatical Cloudformation Templates are created, which allow the application to be deployed automatically.
- Each module of the system is tested. The limitation of the system is explained and discussed.

II. Design Diagram

According to the application scenario, the system should have the following features:

- Capable of hosting a WordPress Application.
- Equipped with a MySQL server.
- The system can scaling automatically.
- The servers and database should have replicas in different availability zone.

Driven by the requirements, the Cloud System is designed as the Fig.1. Following our design, the cloud system be composed of the following main modules.

- The Multi Availability Zone VPC and Subnets. The Virtual Private Cloud and the Private/Public Subnets should cover two different availability zones in the same region.

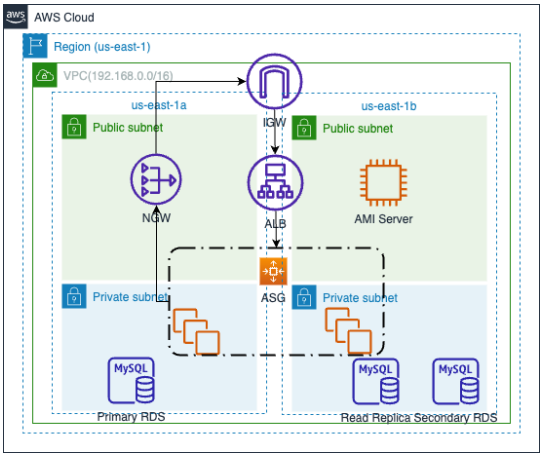

Fig. 1. The Cloud System Design

With this design, the cloud system can be deployed into different availability zone but working as different nodes under the same network environment. Facing with the failure of the availability zone, the existing infrastructures in a different region is capable to keep hosting the application

- The Multi Availability Zone Database Server with Read Replica. The database should has Multi-AZ enabled with read-replica. With this design, the database deployment has higher robustness with different kind of failure. No matter is the server internal error or the availability zone outage, there will always be available database instances.
- EC2 instances, application load balancers, and the auto scaling policies. We wish there are different server instances, increasing the system robustness, which are the EC2 instances hosting the WordPress application. The application load balancer is forwarding the traffic to different EC2 instance to balance the load, and the auto scaling group is scaling in or out automatically according to the system load.

III. Implementation

To implement the design of the highly available WordPress Application, I have deployed the whole WordPress program on the AWS cloud manually. Then I have created two CloudFormation templates to automatically deploy the WordPress system on the AWS.

A. VPC

A customized VPC with private and public subnets and appropriate routing to NAT and IGW is used to host this application.

Firstly, I have created a VPC, which is shown in the Region us-east-1. Then in us-east-1a and us-east-1b, I have separately created the public subnet 1, public subnet 2, private subnet 1, and private subnet 2.To achieve this, I have executed the following operations. The screenshot of the VPC configurations are shown in Fig.2.

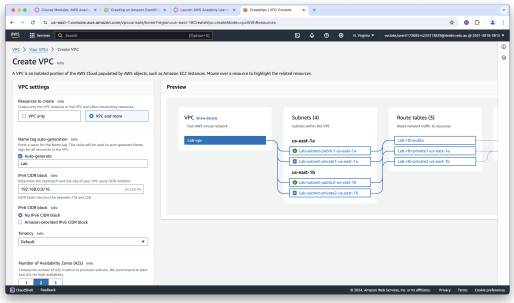

Fig. 2. Create VPC Configurations

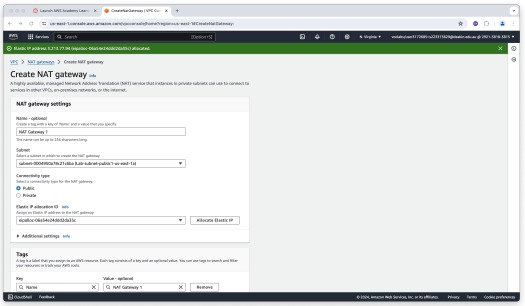

Fig. 3. Create NAT Gateway

- In the AWS console, search and choose the VPC.
- Choose the create VPC and more. By doing this, the VPC and corresponding subnets will be created automatically. For the options, select to choose:
  - 2 Availability Zones (AZs)
  - 2 Public Subnets
  - 2 Private Subnets
  - None NAT Gateways, as we will create later
  - VPC Endpoints: S3 Gateway. To choose this, it is a cheaper choice and make it easier for private network to privately access the S3 bucket.
- Change the IPv4 CIDR according to the Design Diagram. The CIDR should be changed as follows:
  - VPC IPv4 CIDR Block: 192.168.0.0/16
  - Public Subnet 1: 192.168.0.0/24
  - Public Subnet 2: 192.168.1.0/24
  - Private Subnet 1: 192.168.2.0/24
  - Private Subnet 2: 192.168.3.0/24

Secondly, I have to create the NAT gateway in the Public Subnet 1. This NAT gateway will grant the Internet access for the instances in Private Subnet 1 and Private Subnet 2. To achieve this, I have followed the following Instructions. The result is shown in Fig.3.

- Select `NAT Gateways-Create NAT Gateway`
- Name as: `NAT Gateway 1`
- Subnet: `Choose the Public Subnet 1`
- Connectivity Type: `Public`
- Elastic IP allocation ID: `Allocate Elastic IP`

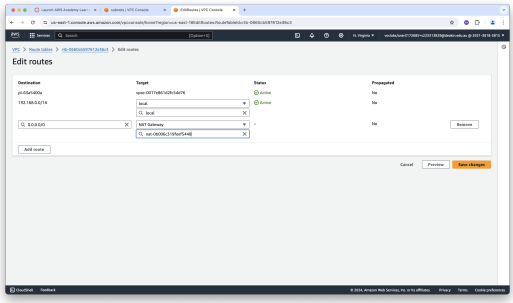

Fig. 4. Modify the Route Tables

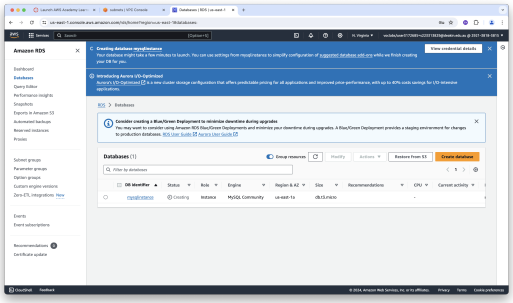

Fig. 5. Create the RDS instance

Finally, I have to modify the route tables. The flow from Private Subnet 1 and Private Subnet 2 should be navigated to the NAT Gateway 1. To change the route table, I have followed the following Instructions. The result is shown in Fig.4.

- Select `VPC-Route Tables`
- For the current route tables, we can see the flows from Public Subnet 1 and Public Subnet 2 are navigated to the Intenet Gateway. While the flow from Private Subnet 1 and Private Subnet 2 is navigated to no where.
- Open the route table for Private Subnet 1 and Private Subnet 2. Add a new route, using the following options:
  - Destination: `0.0.0.0/0`
  - Target: `NAT Gateway`, choose the NAT Gateway just created.
  - Choose `Save Changes`

B. RDS

In this section, I will create a MySQL RDS. To implement this, I have followed the following steps. The result is shown in the Fig.5.

- From the AWS console, search and choose `RDS`
- Choose `Database-Create database`
- Choose `Standard Create-MySQL`
- For the template, choose `Free tier`
- For the credentials management, choose self managed
- For the Public Access, choose `No`
- For the VPC Security Group, choose `Create New`
- The name for the New Security Group is `RDS SG`, and the availability zone is *us-east-1a*

After that, we can see the RDS instance is created. Then we will start from the RDS console, enabling the Multi-AZ and create a read-replica in different AZ.

To enable the Multi-AZ and create a read-replica in a different AZ, I have followed the following steps. The Fig.6 shows the status when converting the database into a Multi-AZ instance, and the Fig. 7 shows read-replica in a different AZ than the primary RDS.

- Choose the database just created, then choose `Action-Convert to Multi-AZ Deployment`
- Then, choose `Create Read Replica` from the action menu.

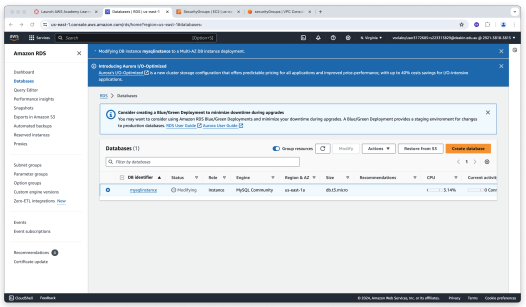

Fig. 6. Enable the Multi-AZ for RDS instance

- When creating the read replica, following the same settings of creating the RDS instance.
- For the availability, choose `Single DB instance`. The Availability Zone should select `us-east-1b`.
- Attention: The availability setting in the read replica is independent from the original database. Introduced by AWS guide [4], creating read replica as a Multi-AZ DB instance is independent of whether the source database is a Multi-AZ Database instance.

C. ELB

I will create a application load balancer and place it in both Public Subnet 1 and Public Subnet 2. To achieve this, I have followed the following instructions. The created load balancer is shown in the Fig.8.

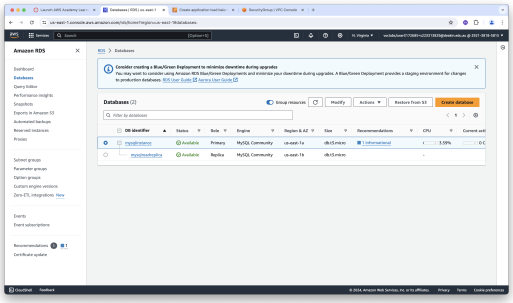

Fig. 7. Enable the Multi-AZ for RDS instance

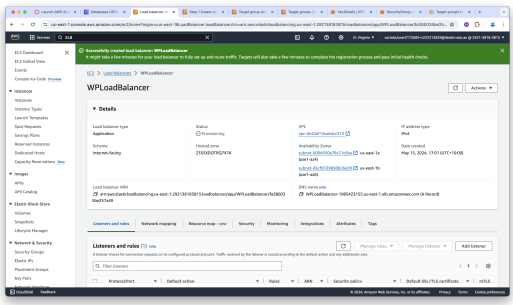

Fig. 8. Create the Application Load Balancer

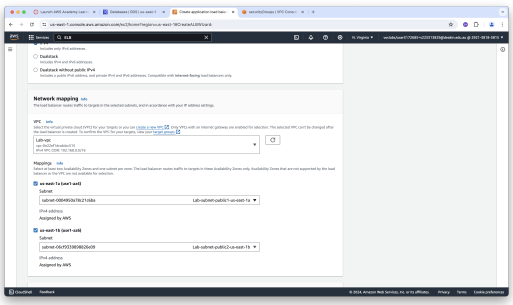

Fig. 9. The Network Setting

- From the AWS console, search and choose `ELB`, which is located in the EC2 panel.
- Then select `Create Load Balancer - Application Load Balancer`.
  For the name, I set it as `WPLoadBalancer`.
  Also, make sure it is `Internet-facing`
- For the network mapping, set VPC to `Lab-vpc`.
- For the subnets, choose Public Subnet 1 and Public Subnet 2. The example is shown in the Fig.9
- For the security group, create a new `ELB SG` security group and choose it, using the security group page.
  - When creating the security group, select the VPC to `Lab-vpc`.
  - For the inbound rule, only add a record allowing the traffic from port 80 for the WordPress Application.
- For the listener, I choose `80` as the listening port with HTTP protocol. Then create a target group.
  - For basic configuration, choose `instances`
  - For the target group name, choose `WP-App`
  - For the protocol, select `HTTP`
  - Edit the health check part. In my setting, I set the unhealthy threshold to 2 and the timeout is 15 seconds.
  - Remain the rest selections unchanged, and proceed to create the target group.
  - The created target group is shown in the Fig.10
- After choosing the target group, add a name tag called `WordPress Listener`. Then select `Create Load Balancer`

D. S3

In this step, I will create a S3 bucket with public access disabled. To do this, I have followed the following instructions. The result is shown in Fig.11.

- From the AWS panel, search and select `S3 - Create Bucket`.
- For the public access, choose block all public access.
- Then choose `create bucket`.

E. EC2 and WordPress

In this step, I will create an Amazon Machine Image and deploy the WordPress application.

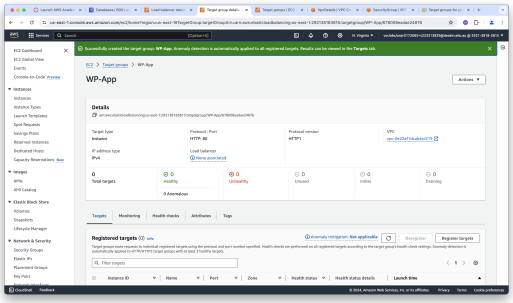

Fig. 10. Create the Target Group

Firstly, I have launched the Amazon Machine Image server with the following steps. The result is shown in Fig.12

- From the AWS panel,
  choose `EC2-launch instance`
- Create a AMI using the following configurations.
  - Name: `WPServer`
  - Amazon Machine Image:
    `Amazon Linux 2 AMI(HVM), SSD Volume Type`
  - Instance type: `t2.micro`
  - Key pair: `Create new key pair`
  - Firewall: `Create Security Group`
    - Name: `Web Server SG`
    - VPC: `Lab-vpc`
    - Inbound rules: `HTTP 80 Anywhere`

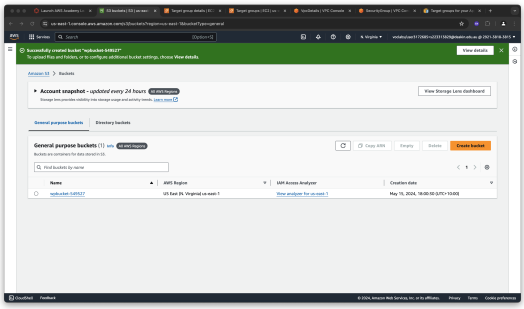

Fig. 11. Create the S3 Bucket

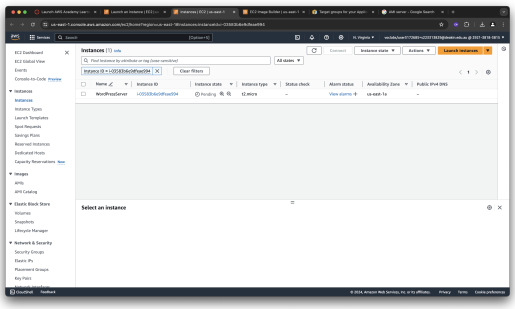

Fig. 12. Create the EC2 instance

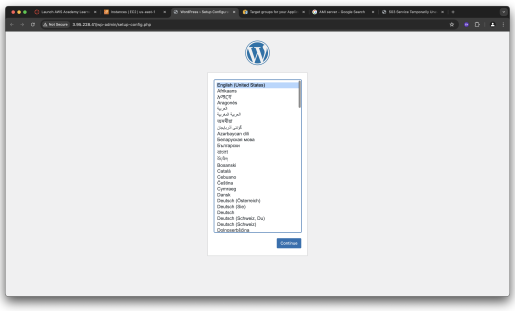

Fig. 13. The WordPress page

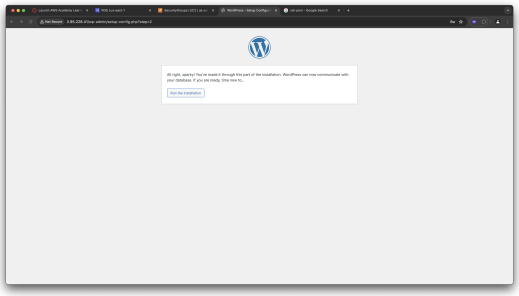

Fig. 14. The Database is Connected

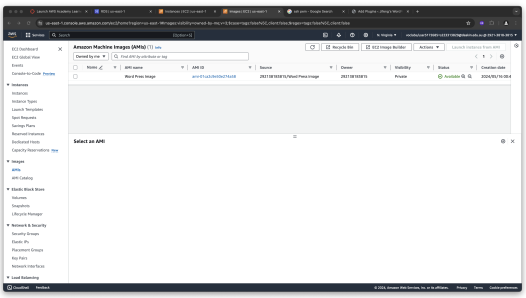

Fig. 15. The Customized AMI is Created

  - User Data: `Command About Deploying the WordPress`.

After these steps, I have an EC2 instance running the WordPress. Entering the public IP of the instance. We can see the configuring page of the WordPress, which is shown in the Fig.13. Now I'm going to set up the RDS connection information in the WordPress, then create an AMI from this server. To set up the WordPress, I have followed the instructions below.

- Select the mysql database we have just created on the RDS page. Find the information on the configuration page, including the master user name and password.
- Using the terminal, I have connected to EC2 and connected to RDS through EC2. Create a database called wordpress in the mysql database.
  I used the command of `mysql -h <endpoint> -P 3306 -u <user> -p` to connect the database,
  and `CREATE DATABASE wordpress` to create the database.
- Then you can see the EC2 instance is successfully connected to the database, which is shown in the screenshot Fig.14.
- After that, follow the steps and install the WordPress Application.

Now, we have the WordPress configured with RDS connection. Next, we will use this server to create an AMI. To achieve this, I have followed the following instructions. In the end, we will see the image is available in the Fig. 15

- Select the instance which we have just created.
- Choose `Actions-Images and Templates-Create image`
- For the name, I entered `Word Press Image`

So far, we will have a configured Amazon Machine Image, which has configured the information to connected with the RDS database.

F. S3 Integration

Based on the EC2 instance configured with the RDS connection we just created, we will configure the Word Press settings to offload the media into the S3 bucket. To achieve this, I have followed the instructions below.

- Go back to the WordPress page held by the EC2 instance we created.
- Select `Plugins - Add New Plugins - Offload Media by Acowebs` and install.
- Input the S3 information in this Plugin. The access key can be found in the `AWS Academy - Cloud Access`. It should be shown as Fig.16.
- However, it doesn't work. I will discuss it in the discussion part.

Before updating the AMI, set the `WordPress URL` in the WordPress setting to the address in the application load balancer. The reason is discussed in the discussion part.

After that, we have the AMI server instance prepared. We are going to use it to update the image and create a launch template for setting up the Auto Scaling Group in later steps. The steps is same to the steps above. Then I have followed the following steps to create a Launch Template:

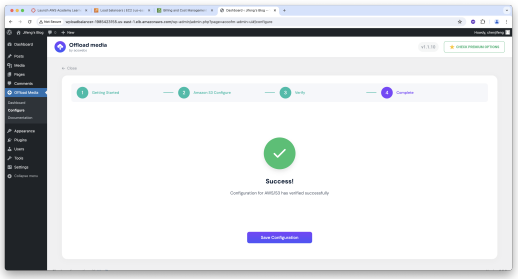

Fig. 16. The WordPress is configured

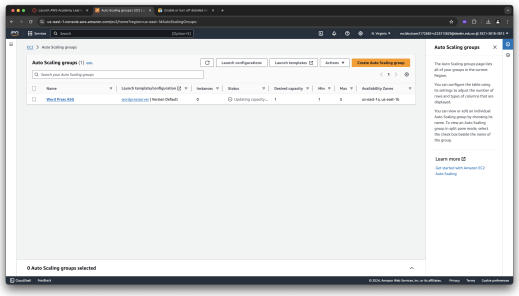

Fig. 17. The ASG is Created

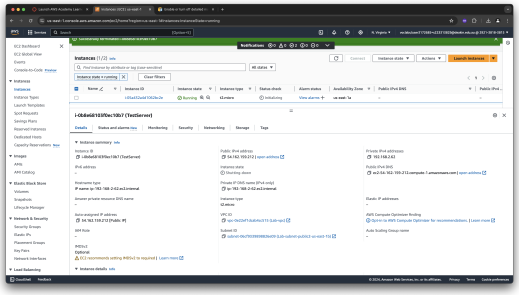

Fig. 18. The EC2 Instance is automatically created

- Select `Launch Template - Create Launch Template`
- For the name, I choose `wordpressserver`
- AMI: The AMI I have just created
- Keypair: The keypair I have just created
- Security Group: For the security group, choose the `ELB SG` which is created by me before.

In the next step, we will create an Auto Scaling Group.

G. Auto Scaling Group

For the Auto Scaling Group (ASG), we are expected to create an ASG following the follow requirements.

- The minimum number of servers is 1
- The maximum number of servers is 3
- Desired capacity is 1
- Scale out when average CPU utilization over 70
- Scale in when average CPU utilization below 25
- The ASG should launch instances into the private subnets.

To achieve this, I have followed the instructions below:

- Select `EC2-Create Auto Scaling Group`
- Choose a suitable name and choose the template we have just created as launch template.
- For the VPC and Subnet, choose the `Lab-vpc` and private subnets
- For the load balancing, choose `Attach to an existing load balancer`.
  Then choose the load balancer we have already created.
- Go to the step 4, choose the desired capacity, min number, and max number according to the requirements of ASG.
- For the Automatic Scaling, choose `Target tracking scaling policy`.
  Metric type is `Average CPU Utilization`. The target value is `25`.

In the end, we can see the Auto Scaling Group is created, which is shown in the Fig.17. We can see the instances are auto created in Fig.18.

H. Create the infrastructure in this phase with CloudFormation

In this section, I will automatically deploy all these AWS deployment modules using the AWS CloudFormation. According to different lifecycles, the infrastructure can be organized into two templates.

- Networking service template, including VPC,NAT,IGW,Route Tables etc.
- Application infrastructure template, including EC2, Auto Scaling Group, RDS etc.

After that, I have followed the following instructions to deploy the cloud environment using the cloud formation template.

- Search and go to the CloudFormation from the Amazon Control Panel
- Create new stack using the network template, which is creating a new VPC, several subnets, and the corresponding route tables.
- After the created stack shows ready, go to create a new stack using the infrastructure template, which is creating the RDS and the application load balancer.

In the end, you will see a new WordPress instance accessible from the new ELB address.

I. Testing

1) Access to WordPress Page: To test whether our WordPress page is fully functioning, I have found the Application Load Balancer on the Load Balancer page of the EC2. I have copied the DNS name. The screenshot is listed as fig.19. From the result, we can see the website is fully functioning.

2) Auto Scaling Group: The Auto Scaling Group (ASG) is managing the running instance of the website and controlling the scale in and scale out policy.

To test the Auto Scaling Group, I have stopped the EC2 instance which is launched by the ASG. Then I can see a new

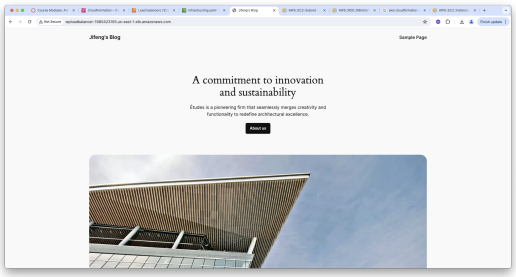

Fig. 19. The WordPress is accessible from the ALB address

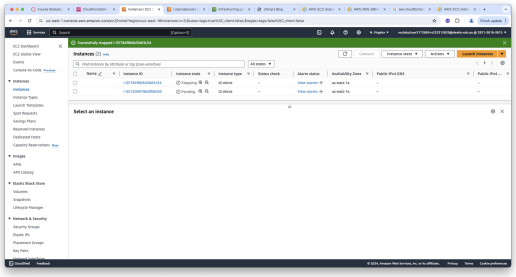

Fig. 20. The EC2 instance will be created by ASG

instance is started by the ASG. The fig.20 shows the ASG policy is starting a new EC2 instance.

3) Session Manager: Due to the policy issue, my AWS lab account is not allow to create IAM Role. Due to this, I can't attach the proper IAM Role to the EC2 instance and connect to the web server using the session manager. But with the proper keys, I can connect to the web server using the local terminal or the AWS cloud9 console.

4) S3 Media Content Offloading Test: With the proper setting, the WordPress program should be able to offload the media contents to the AWS S3 bucket. As described in the previous steps, I have use my own S3 bucket.

Now I have uploaded a screenshot into the WordPress. Then we can see the new screenshot appeared in the S3 bucket, shown as fig.21

5) RDS instance test: I have checked my settings that the RDS database is configured properly, and the read replica is

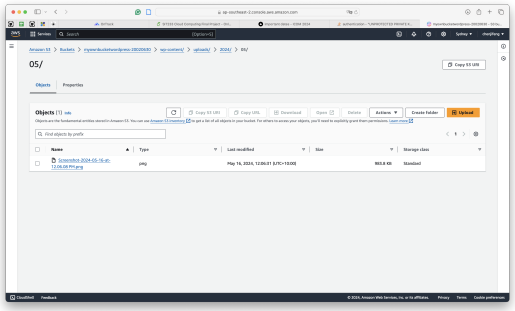

Fig. 21. The Media is Offloaded to the S3 Bucket

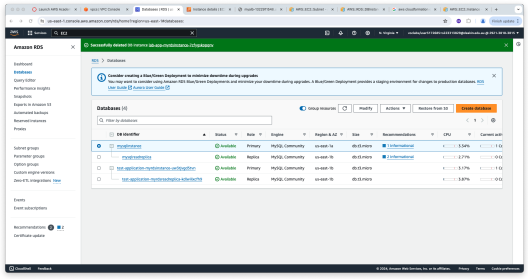

Fig. 22. The RDS instances are created

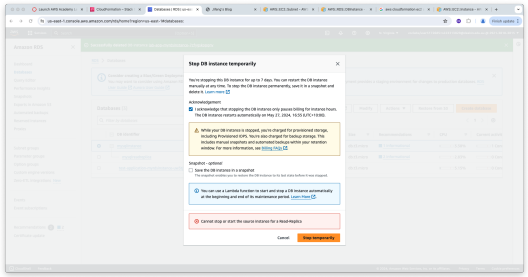

Fig. 23. The RDS instances aren't allowed to stop

running in a different availability zone, which is shown in the fig.22.

However, I didn't manage to stop the running instance because there is a ReadReplica using this RDS instance as the source, which is shown in the fig.23. If I directly create this RDS instance with Multi-AZ cluster, the result might be differently.

Technically, when I stop the Multi-AZ and restart it, the Primary zone and the Secondary Zone will be re-arranged. So when I restart the RDS instance, we can see the primary and secondary zone settings should be changed.

6) Cloudformation Test: As I have described all me deployment using the the cloudformation template. The WordPress should be able to be deployed automatically.

From the fig.24, we can see the stacks are created correctly, which means there are no syntax error in my template and successfully deployed.

Then go to the AWS VPC page, we can see there is an additional testVPC is created. There are also four subnets

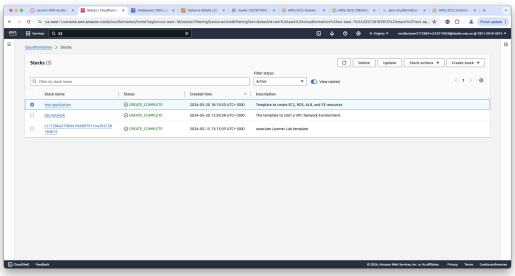

Fig. 24. The Stacks are created automatically

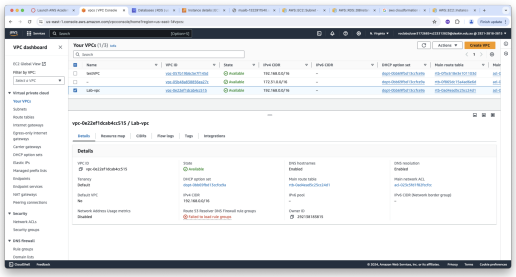

Fig. 25. The Networks are created automatically

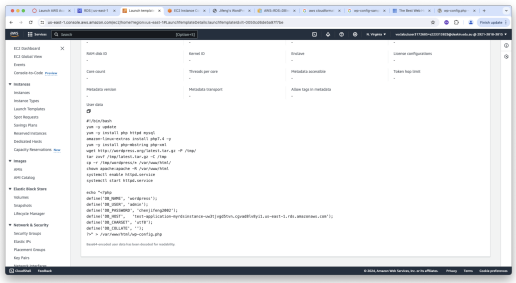

Fig. 26. The launch template are created automatically

in two different availability zone. Besides, we can see there is a NAT gateway created with proper routes. All of these are shown in the fig.25. In a word, the network template is designed correctly.

Then go to the RDS page, we can see there is an additional RDS database instance with read replica is created, which is shown in the fig.22.

After that, we can go to the EC2 page. On that page, we can see that in the launch template, the RDS credential is passed into it successfully, which is shown in the fig.26. Then I use the DNS address generated by the application load balancer, we can see the website can be accessed properly, which is shown in the fig.27. In a word, the infrastructure template is designed correctly.

IV. Discussion and Reflections

During the implementation of the system, there are several difference from our original design, and several problems should be pay attention to.

- No Security Group attached to the NAT Gateway. In the AWS, the NAT is not allowed to attach an individual security group to it. To control the inbound and outbound rules of the NAT gateway, the only way is to configure the route tables properly.
- The subnet group can't be created for the RDS instance. In the experiment, I didn't create a a subnet group for the RDS instance. As the experiment is conducted in the AWS virtual environment, the user is not allowed to create the subnet group for RDS instance.
- Can't create IAM roles for EC2. In the design, I should create an IAM role for the EC2 instance. However, I didn't manage to create the IAM role due to the privilege issue. This problem also results in the failure of connect to the webserver using the session manager.
- The S3 offloading is not available with the S3 bucket created by the lab account. This is because as this is a lab environment, the access ID and token is not accessible for the WordPress plugins. To solve this problem, I have created my own AWS S3 bucket, and use it for the media offloading by the AWS plugins.
- In the WordPress setting, the URL setting should be changed to the address of the Application Load Balancer. The URL in the WordPress application is generated with the URL in the setting. For example, if I want to go to `wp-admin.php`, the WordPress will generate the URL as ${User-Set-Url}/wp-admin.php. Thus, it's important to set the URL to the address of the application load balancer to ensure that all the address are routed properly.

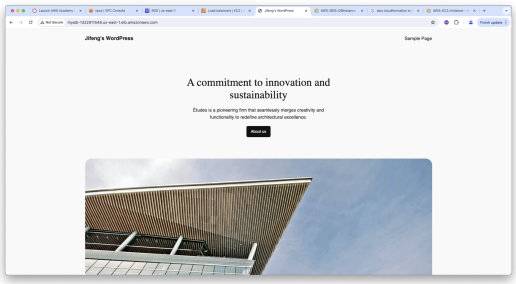

Fig. 27. The automatically created WordPress can be accessed properly

V. Conclusion

In conclusion, in this paper, I have discussed the scenario of developing a highly availability WordPress Application. Then I have used the manual step-by-step methods to create a WordPress deployment with Auto Scaling feature and High robustness. After that, I have created two AWS cloudformation templates, which allow users to create the WordPress deployment autonomously and effectively. Through the validation, I have validated that the main requirements of the system is achieved. However, due to the limitation of the lab account, some of the requirements are implemented in an alternative way. I have discussed the limitation and the reason for the difference from our original design.

VI. Presentation and Source Code

The presentation for this paper is available at the following link. https://youtu.be/2qUaZJC-UEU

The source code is available at the following GitHub Repository containing the cloudformation code and design. https://github.com/Barentatze/SIT233WordPress

References

[1] V. Jain, B. Malviya, and S. Arya, "An overview of electronic commerce (e-commerce)," *Journal of Contemporary Issues in Business and Government*, vol. 27, no. 3, pp. 665–670, 2021.

[2] Wikipedia contributors, “Amazon Web Services,” https://en.wikipedia.org/wiki/Amazon_Web_Services, 2024, accessed on May 21, 2024.
[3] Data Center Knowledge, “History of AWS, Cloud, and Data Center Outages,” https://www.datacenterknowledge.com/uptime/history-aws-cloud-and-data-center-outages, 2024, accessed on May 21, 2024.
[4] A. W. Services. (2024) Working with db instance read replicas - amazon relational database service. Accessed: 2024-05-15. [Online]. Available: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ReadRepl.html