WordPress Website on Amazon Web Services (AWS) :
Building a High-Performant and Cost-Efficient Website

Achour Oussama
Department of Computer Science and Mathematics Engineering
INSAT, University of Carthage, Tunisia
***@insat.ucar.tn, ***@insat.ucar.tn, ***@insat.ucar.tn

Abstract

In today's era of digital transformation, businesses across industries are recognizing the need to leverage technology to achieve their goals and stay competitive in the market. The rapid advancement of technology has revolutionized the way businesses operate, interact with customers, and deliver products and services. It has become evident that embracing digital solutions is not merely an option but a necessity for businesses to survive and thrive in the modern business landscape.
As a result, organizations are increasingly turning to digital solutions to streamline their processes, enhance customer engagement, and drive revenue growth. The ever-increasing reliance on technology has compelled organizations to consider digital solutions as a means to enhance their operations, improve customer experiences, and drive revenue growth.

GitHub repository : https://github.com/AchourOussama/wordpress-aws

1 Introduction

Undeniably embarking on digital transformation wouldn't come for free. In fact , pursuing this process comes with inherent challenges and associated costs. For the Organizations which are willing to be self and solely relying on their own resources and expertise throughout these digital initiatives, they would incur an enormous amount of expenses. These costs come in the form of investments in various utilities and resources , including hardware, software, and skilled professionals. Additionally, managing and sustaining these resources can be complex and expensive . Thus, heavy financial duties would be on the back of the business .

That's been said, satisfying these financial constraints are and will never be a guarantee for a successful digital transformation . In fact, businesses are always subject to inadequate or suboptimal solutions which put them far from achieving their objectives .

Therefore, the challenge lies in finding a balance between the need for digital transformation and the associated costs, while ensuring optimal utilization of available resources as well maximizing the chances of having a well fitted solution for the well-defined business problem.

Here we break with the idea of the necessity of having a total acquisition of resources or being solely responsible for designing and building the technical solution. And here it comes the concept of cloud computing where instead of relying solely on in-house resources and infrastructure, organizations are now exploring the concept of leveraging cloud solutions. This shift allows businesses to access scalable and flexible resources on-demand, as well as making use of already built solutions and services and thus reducing the need for extensive upfront investments ,ongoing maintenance costs and development costs. By embracing cloud technologies, businesses can focus more on their core competencies while leveraging the expertise and infrastructure provided by cloud service providers. This approach enables greater agility, cost-efficiency, and the ability to adapt to changing business needs.

In this regard , our primary goal of this project is to showcase the practical application of cloud computing in the creation of a website. By leveraging the power of cloud services, I aim to design and implement a technical solution that excels in both performance and cost-effectiveness.

Moreover, this project serves as an insightful exploration into the process of building on a cloud platform , offering valuable insights into the necessary design principles and best practices. By utilizing Amazon Web Services (AWS) as the chosen cloud services provider and platform for building and deploying solutions, our focus is on meeting performance requirements, including reliability, high availability, and security

2 Background

Before diving into the technical details of the project, it is essential to establish a foundational understanding of the concepts and terminology we will encounter.

Amazon Web Services (AWS)

" Start Building on AWS Today Whether you're looking for compute power, database storage, content delivery, or other functionality, AWS has the services to help you build sophisticated applications with increased flexibility, scalability and reliability " Amazon web services
As its slogan says, Amazon Web Services (AWS) as a subsidiary of Amazon, it offers a wide range of cloud services, including computing power, storage, databases, networking, machine learning, and more. AWS provides businesses and individuals with the ability to build and deploy applications and services in a flexible, scalable, and cost-effective manner, without the need for upfront infrastructure investments. Regarding costs , its main benefit is that it enables users to access computing resources on-demand, pay only for what they use, and easily scale their applications as needed By being the leader of the cloud computing industry , AWS has a global infrastructure that allows users to deploy their applications in various regions around the world, ensuring high availability and low latency.

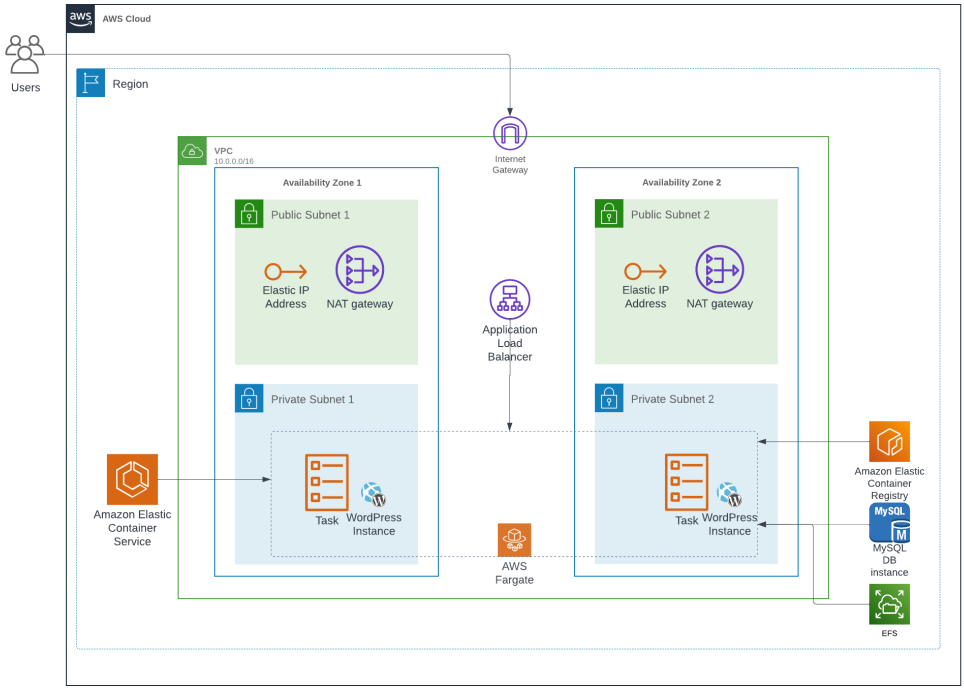
Content management system (CSM) :

CMS stands for Content Management System. It is a software application or platform that allows users to create, manage, and publish digital content on the web (mainly on websites, blogs , etc.) without requiring extensive technical knowledge. A CMS provides a user-friendly interface that enables users to easily add, modify, and organize content, such as text, images, videos, and documents, within a structured framework.

About WordPress :

"Welcome to the world's most popular website builder" wordpress
As WordPress itself is saying , it is considering to be the largest and most popular open-source Content Management System *"43 % of the web is built on WordPress. More bloggers, small businesses, and Fortune 500 companies use WordPress than all other options combined."*wordpress

3 Architecture diagram



Here is the architecture diagram illustrating the infrastructure sector for hosting our WordPress website .
Before getting deep into each element of it ,let's give an overview of all the component as well as a real-life scenario of an access to the WordPress website . Overview about the key components in this architecture diagram

1. User: Represents the end user accessing the website through their web browser.
2. Region: An AWS region is a geographically distinct area that contains multiple Availability Zones.
3. VPC: Virtual Private Cloud, a logically isolated network environment where our AWS resources reside.
4. Availability Zones (AZs): they are isolated data centers within the region, providing redundancy and fault tolerance by allowing us to distribute our resources across multiple AZs.
5. Subnets: they are subdivisions of our VPC network that define IP address ranges within an Availability Zone.
6. Internet Gateway: Acts as a gateway between the VPC and the internet, allowing incoming and outgoing internet traffic.

7. NAT Gateway: Provides internet access for resources in private subnets within the VPC, enabling outbound internet connectivity.
8. Elastic IP: Static public IP address associated with the NAT Gateway or other AWS resources.
9. ECS/Fargate Tasks: Containers running our WordPress application, where the actual processing of requests and generation of responses take place.
10. ECR: Elastic Container Registry, a fully managed Docker container registry for storing, managing, and deploying container images
11. Application Load Balancer (ALB): Serves as the entry point for incoming web traffic, distributing requests to multiple ECS/Fargate tasks.
12. RDS MySQL: Managed relational database service providing a MySQL database for our WordPress website, storing data such as posts, pages, and user information.
13. EFS: Elastic File System, a scalable and shared file storage system used for storing WordPress files, themes, plugins, and uploads.

Real-life scenario of a user accessing the WordPress website:

1. The user enters the URL of our website in their web browser and hits Enter.
2. The DNS (Domain Name System) resolves the domain name to the IP address of the Application Load Balancer (ALB) associated with our website.
3. The request from the user reaches the ALB, which acts as the entry point for incoming traffic.
4. The ALB distributes the request to one of the available ECS/Fargate tasks running our WordPress application.
5. The ECS/Fargate task receives the request and processes it.
6. If the request requires data from the database, the WordPress application communicates with the RDS database to retrieve the necessary information.
7. The ECS/Fargate task generates the HTML content for the requested page and sends it back as a response.
8. The response from the ECS/Fargate task is sent back to the ALB.
9. The ALB then forwards the response to the user's web browser.
10. The user's web browser receives the response and renders the HTML content, displaying the requested webpage to the user. Terminology :
11. Service: In AWS, a service refers to a specific cloud-based offering that provides a particular functionality or capability. Examples include EC2, S3, RDS, and Lambda.
12. Resource: In the context of AWS, a resource is a specific entity or component that you can provision and manage within a service. Examples of resources include EC2 instances, S3 buckets, RDS database instances, and IAM roles.

4 Architecture components :

4.1 Architecture core components :

In this section we will talk about the main components taking part of the architecture and in some cases the reasons of choosing them over other ones In each component of the architecture , we will first introduce the services/concepts we made use of and then talk about them in the regard of our project

4.1.1 Network :

Components :

- ***Region and Availability Zones :***
  - regions and Availability Zones are what geographically define the AWS global infrastructure and it is in them where real physical data center facilities reside
  - AWS Regions are large and widely dispersed into separate geographic locations. Availability Zones are distinct locations within an AWS Region that are engineered to be isolated from failures in other Availability Zones. They provide inexpensive, low-latency network connectivity to other Availability Zones in the same AWS Region.
- ***Virtual Private Cloud (VPC) :***
  - A VPC Virtual Private Cloud is logically isolated virtual network that you define on the cloud . In fact, as it names says , it enables you to possess a virtual a portion of the AWS cloud that is private to you .
  - Comparing to a traditional network in a data center , it looks the same ,except that it offers more scalability .
  - It is in our created VPC where you can provision resources(i.e., network , compute resources , etc.) and you will full control over various network environments, resources, connectivity, and security inside it
- ***Subnets :***
  - A subnet is a logical subdivision of the virtual private cloud (VPC) 'the virtual network that you have allocated' which consists on a range of IP addresses
  - A subnet must reside in a single Availability Zone.
  - After you add subnets, you can deploy AWS resources in our VPC.
- ***Security groups :***
  - security group acts as a firewall that controls the traffic allowed to and from the resources in our virtual private cloud (VPC) resources (databases, compute servers , etc.)
    You can choose the ports and protocols to allow for inbound traffic and for outbound traffic.
  - In our case , we have defined security groups for each service,ressouce we have for this solution (ECS cluster , RDS MySQL database , EFS file system , ALB)
    Note: all these elements will be discussed later
- ***Network Access Control List (NACL) :***
  - A network access control list acts as a firewall that allows or denies specific inbound or outbound traffic at the subnet level.
  - As a subnet level firewall, it adds an additional layer of security apar from the security groups (which are instance level)
  - We can use the default network ACL that is created with the VPC , or we can create a customized network ACL .
  - *Resource:* https://docs.AWS.amazon.com/vpc/latest/userguide/vpc-network-acls.html
  - In our case, we made use of the default NACL .
    However, we have made some configuration on the inbound and outbound rules (to see in the implementation documentation)
- ***Network Address Gateway (NAT) Gateway :***
  - Network Address Translation (NAT) gateway permits the instances in a private subnet to connect to services outside our VPC but external services cannot initiate a connection with those instances.

- NAT Gateway has 2 connection types : Public and private
  In our case : we used the public type where Instances in private subnets(which are the WordPress containers) can connect to the internet through a public NAT gateway, but cannot receive unsolicited inbound connections from the internet (however , this traffic would be transmitted through a load balancer) . we create a public NAT gateway in a public subnet and must associate an elastic IP address with the NAT gateway at creation. (This step could be chosen to be made automatically when creating the VPC)
- *Resource:* https://docs.AWS.amazon.com/vpc/latest/userguide/vpc-nat-gateway.html

- ***Application Load Balancer (ALB) :***
  - A load balancer serves as the single point of contact for clients. The load balancer distributes incoming application traffic across multiple targets, such as EC2 instances, in multiple Availability Zones.
    In our case , we created an Application Load Balancer that received incoming HTTP/HTTPs traffic and distributes it across the running tasks (resp. containers)
  - A Load balancer requires the creating of a listener and a target group
  - *Resource :* https://docs.AWS.amazon.com/elasticloadatabasealancing/latest/application/introduction.html

For this project :

- ***Region :***

  We chose as a region the *eu-central1* which is in *Europe (Frankfurt)*

  Decision criteria:
  when it comes to choosing what region to use in AWS , different factors should be taken into account including; compliance , latency , cost and services and features .
  For the scope of this project, the only factor that's worth consideration we should take into account is the latency (between the use and the deployed solution) .

  In fact , choosing an AWS Region with close proximity to the user base location can achieve lower network latency and thus increasing communication quality .
  Apart from the end-user , latency could also impact the experience of cloud engineers and administrators when deploying and provisioning infrastructure.

  In order to define what region has the lowest possible latency , I made use of pining tool called cloud.info which measures the latency from the browser(on it we are using the AWS platform) to the different AWS regions.
  Tool's URL: https://www.cloudping.info/

  The average result of these measure always shows that the eu-central1 , Europe (Frankfurt) region is one the best regions to go for .

  Note: that not all regions are accessible . For that reason, we may not be able to select a region even if it has a better latency result .
  The last time measure was 71 ms , which is considered great to experience building and deploying on the AWS cloud

- ***VPC:***
  we have named it *project-vpc* and assigned it a 10.0.0.0/16 network prefix.
  We made it spread across 2 AZs (eu-central-1a and eu-central-1b) where each AZ has 2 subnets

4.1.2 Compute :

Services:

- ***Amazon Elastic Container Service -ECS) :***

  ECS is a highly scalable container orchestration service that lets us run and manage containers (i.e Docker containers). It simplifies the deployment, scaling, and management of containerized applications .

  Terminology:

  - cluster : Amazon ECS cluster is a logical grouping of tasks or services. It allows for shared capacity and common configurations.
  - Task Definition: It is like the blueprint for the application in the way that it describes how containers should be launched and orchestrated. It includes information such as container images, resource requirements, networking settings, etc.
  - Task: An instantiation of a task definition. It represents the running containers and their configuration. A task can have one or more containers that are tightly coupled and share resources.
  - Service: A group of tasks running simultaneously. Services help us manage the lifecycle of tasks by automatically scaling, replacing failed tasks, and distributing tasks across available resources.
  - Container Registry: It is like a repository for storing and managing container images. In the case of ECS, we can use Amazon ECR (Elastic Container Registry) to store our Docker container images securely.
  - Launch type : the way in which containers are launched and managed within the ECS infrastructure. It determines how the underlying compute resources are provisioned and managed for running containers.
  - *Resource:* https://AWS.amazon.com/ECS/

- ***Amazon Fargate :***

  AWS Fargate is a serverless, pay-as-you-go compute engine for the containers that lets us focus on building applications without managing servers.

  Resource: https://aws.amazon.com/fargate/

Choices of technologies:

- ***Terminologies :***

  - Containerization :
    method of running applications in isolated environments called containers. Containers provide a lightweight and portable solution for packaging software, along with all its dependencies, into a standardized unit that can run consistently across different computing environments.

  - Pay-as-you go model :
    a pricing model where, instead of paying an upfront payments or long-term commitments, users only pay for the resources they consume and the duration of their usage. This payment model is adopted by aws.

  - Serverless : by being a serverless service , it means that it abstracts away the underlying infrastructure where we don't need to manage servers or worry about capacity planning.

AWS Fargaet could automatically provision the necessary compute resources to run our containers.

- ***Why choosing containerization :***

  By containerizing WordPress and its dependencies, we achieve portability, enabling consistent performance and agile deployment.
  With containers, you gain the flexibility to scale our WordPress application seamlessly, handling increased traffic and workload efficiently. Containers are resource-efficient, allowing you to optimize resource utilization and reduce costs by running multiple containers on a single host.

- ***Why choosing Elastic Container Service (ECS) :***

  Deploying a containerized application on AWS could be done through different ways :

  - Amazon Elastic Kubernetes Service (EKS): EKS is a managed Kubernetes service provided by AWS. It allows us to run containerized applications using the popular Kubernetes orchestration platform. EKS provides a highly scalable and flexible environment for managing and deploying containers. and flexible environment for managing and deploying containers.
  - EC2 instance with docker agent : as in the on-premises environment

- ***Why choosing Fargate as a launch type:***

  The main reasons behind choosing Fargate over ec2 are :

  - Serverless Experience: Fargate provides a serverless compute engine for containers, allowing you to focus on running our applications without the need to manage the underlying infrastructure.
  - Automated Operations: Fargate automates the container orchestration tasks, such as capacity provisioning, cluster management, and container scaling.

The process: **Note:** This process supposes that we have a well configures vpc (subnets , security groups , Nat gateways , etc.)

1. Create an ECS Cluster: Create an ECS cluster, which acts as a logical grouping of container instances. Here we choose the Fargate as a launch type.
2. Configure Task Definition: Create a task definition that describes how our containers should be run. Specify the container image, CPU and memory requirements, port mappings, and any environment variables. Note : environment variables referred to the provide settings specific to our application, such as database connection details and the and site URL.
3. Create an ECS Service: Create an ECS service to manage the lifecycle of our containers. Configure the service to run multiple instances of the task definition (tasks) across different availability zones for high availability. Also, we could benefit from having an ALB (as discussed earlier) so that it distributes incoming traffic across the containers in our ECS service.
4. Access the WordPress Site: Once the service is running, you can access our WordPress site by accessing the ALB's DNS name or domain name. Complete the initial WordPress setup by providing the necessary information, such as site title, admin credentials, and database connection details.

Note: These are the main and basic steps to pass from creating the container cluster (ECS cluster) to accessing the WordPress website . All technical details of these steps are covered in the

implementation documentation in the GitHub repository
Note: About the container image : In this project, the container image refers to the Docker image that contains the WordPress application and its dependencies. We have two ways to define from where to import the container :

- *Docker hub :*
  - Docker Hub is a cloud-based repository provided by Docker that allows you to store and distribute Docker container images ,as well as it offers existing container images created by other developers or organizations. In our case we could get benefit from the WordPress official docker
  - It is the easiest way since this solution only requires just the entering the path docker hub WordPress docker image in the environment variable in the task definition
- *Elastic Container Registry (ECR) :*
  - Amazon Elastic Container Registry (ECR) is a fully managed container registry service provided by AWS. It allows you to store, manage, and deploy Docker container images.
  - This approach is more complicated that using Docker Hub, since it requires having the Docker platform on our machine, importing the WordPress docker image and uploading it to the ECR .
    That's been said , this solution offers more flexibility when it comes to managing the docker images as well it guarantees a less latency when importing the image comparing to importing it from docker hub

4.1.3 Storage :

These components play a crucial role in storing and managing the data required by our WordPress application.

Database :

- ***Need for a database:*** WordPress relies on a database to store various types of information, such as posts, pages, user data, comments, settings, and more. Therefore , having a database is essential for a WordPress website as it is responsible for storing and managing these structured data as well as allowing the website to retrieve and display this dynamic content based on user requests.
- ***Choice of technology: Relational DataBase Serivce (RDS) MySQL***
  - Amazon RDS (Relational Database Service) is a managed database service provided by AWS.
  - It offers a convenient and scalable solution for hosting and managing relational databases.
  - It supports different databases engine (MySQL , SQL Server , Oracle , MariaDB , etc.)
  - In our case , we chose MySQL database engine

Storage :

- ***Need for a storage :*** While the database handles the dynamic structured data, a filesystem is responsible for storing static files such as media uploads, themes, plugins, and other assets.
- ***Choice of Technology : Elastic File System(EFS) :*** A scalable and fully managed file storage service provided by AWS. It is designed to provide shared file storage across multiple EC2 instances or containers.

In our case, we created an EFS filesystem which provides a shared file storage to all the WordPress tasks that get into action .

4.2 Recaptulation :

our infrastructure is composed from the following 3 main components:

- Amazon ECS/Fargate:
  This is the compute component of the architecture. Amazon ECS (Elastic Container Service) is a fully managed container orchestration service that allows you to run and scale containerized applications.
  Fargate is the launch type associated within ECS that provides serverless compute for containers, abstracting away the need to manage the underlying infrastructure.
  Therefore ECS/Fargate enables you to deploy and manage our WordPress application containers efficiently and provides automatic scaling capabilities to handle changes in demand.
- Amazon RDS MySQL:
  This is the database component of the architecture. Amazon RDS (Relational Database Service) is a managed database service that simplifies the setup, operation, and scaling of relational databases.
  In our case, you are using the MySQL database engine. Amazon RDS allows you to store and manage the data for our WordPress website, ensuring data durability, high availability, and automated backups.
  It provides features like automatic software patching, monitoring, and scaling to handle database traffic.
- Amazon Virtual Private Cloud (VPC) :
  This is the networking component of the architecture. Amazon VPC provides a logically isolated section of the AWS cloud where we can launch resources in a virtual network. It allows us to define our network topology, set up subnets, configure routing, and control inbound and outbound traffic using security groups and network ACLs (Access Control Lists).
  VPC enables secure communication between our ECS/Fargate instances and RDS database, and allows you to control access to our resources.

Apart from the main 3 conventional components , we have made use of the Amazon EFS, which is a scalable and fully managed file storage service that allows you to share files between containers within the ECS/Fargate cluster.
In our case, you can use Amazon EFS to store and share the WordPress files, such as themes, plugins, and media uploads.
Note: Through this part we tried to cover all aspects of the architecture, and in order to not make the report seem too long and heavy for the reader , we have abstracted different details and instead we focused on the most important aspects of the solution .
All details regarding the technical implementation documentation of that architecture are covered in the following GitHub repository

5 Solution evaluation :

5.1 IT Performance Pillars :

High-Availability : There are any factors in our solution that fosters its high availability . Here we discuss the main ones

- ***Containerization :*** By using the containerization concept, we are leveraging various benefits that contribute to a better availability of our website :

- ***Isolation :***
  Containers provide a high level of isolation between applications and their dependencies. Each application runs within its own container, ensuring that failures or issues in one container do not impact others.
  This isolation helps to prevent cascading failures and improves the overall resilience of the system.
- ***Replication and Scaling :***
  One of the elements of a containerization solution is the Container orchestration platforms (in our case ECS) , which enables easy replication and scaling of containers. We just need to define the desired number of replicas for a container, and the orchestration platform will automatically manage their deployment and distribution across the infrastructure.
  This fact when it's combined with the load balancing (we will discuss it later) they assure a fair distribution of the traffic among multiple instances .
  Thus, enhancing availability and handling increased demand.
- ***Auto-recovery :***
  Container orchestration platforms can automatically recover failed containers by restarting them or provisioning new ones to replace the failed instances.
  This automated recovery process helps to minimize downtime and ensures that the application remains available even in the event of container failures.
  If our website was deployed on a server (the traditional deployment method) , this auto-recovery process would take lot of time comparing to our case
- ***Multi Availability Zones Deployment :***
  Our solution is designed to span multiple AZs within an AWS Region. This ensures that our WordPress website is deployed in redundant data centers, providing fault tolerance and minimizing the impact of infrastructure failures (i.e., If one Availability Zone becomes unavailable, the website can continue running in another AZ without disruption.)
  This helps us achieve high availability and improved resilience.
- ***Elastic Load Balancer (ELB)***:
  In our case the type of the ELB we chose was Application Load Balancer (ALB).
  The ALB acts as a central entry point for all the incoming traffic to the WordPress website. It distributes traffic across multiple ECS/Fargate instances (tasks) running our WordPress containers.
  By evenly distributing the load, the ELB prevents any single instance from becoming overloaded . This improves responsiveness, and provides fault tolerance. If any instance becomes unhealthy or fails, the ELB automatically redirects traffic to healthy instances.
- ***EFS File System :*** Elastic File System (EFS) provides to us a shared file storage for the WordPress application across multiple ECS/Fargate instances. EFS is designed to provide high durability and availability, automatically replicating data across multiple AZs. This ensures that our WordPress files and media are accessible from any instance within the ECS/Fargate cluster, promoting high availability and consistent user experience.

Auto scalability : We have already mentioned in different times in this report that ECS / Fargate offers us a great autoscaling feature
So simply we could say, by leveraging the autoscaling service offered by ECS/Fargate, our WordPress application can dynamically adjust the number of running instances based on demand.
Autoscaling helps ensure that the application can handle increased traffic or workload, improving availability during peak periods and scaling down during periods of lower demand. This directly fosters a better availability of the website

Security : Through different practices, we aimed throughout this project to not only care about the performance of the solution but also its security . Thus, we trying to secure it at different layers of the architecture .

Note: we already described some what we will say in the following in the previous section (Architecture Components)

- ***Security Groups :***
  Acting as a virtual firewall, security groups protect all the critical services/resources in our solution (ECS cluster , ALB , RDS MySQL database , EFS file system , etc.) though controlling inbound and outbound traffic and defining what to allow and what to deny the decision is made based on rules which take into account the type of traffic , IP addresses , protocols, and ports, of both the source or the destination .
  This helps us reduce the attack surface and protect our assets.
- ***Network ACLs :***
  Network ACL allow us to define granular rules for inbound and outbound traffic at the subnet level.
  It uses the same concept of rules the same as for the security groups and it creates an additional layer of protection for our VPC, by preventing unauthorized access and controlling traffic flow.

Monitoring and Logging : AWS provides services like CloudWatch and CloudTrail, which allow us to monitor and log activities within the infrastructure.
By enabling these services and configuring appropriate alarms and notifications, we can detect and respond to security events, ensuring the ongoing security of our WordPress website.
Note: CloudWatch and CloudTrail have not been covered in the report neither in the implementation documentation since we had not made lot of use of them

5.2 Other benefits :

The whole idea of this project , as mentioned in the intro and motivation sections , is to provide an easy to do way for the businesses to have they own website with the least effort and resources possible . This gives to the business the space to focus more on its critical business tasks and get rid of all the habitual operations .
Therefore, all the choices of technology have been made in the sake of simplifying the management of the website .
Here is a description of how our technical choices focus more on their core business activities and alleviate the burden of technical management :

Managed Services : In this architecture we have made use of different managed services (i.e., ECS/Fargate, RDS, and EFS) , where we offload the responsibility of infrastructure management to AWS. Therefore , is it AWS who takes care of the underlying infrastructure, including server provisioning, maintenance, and security patches. This eliminates the need for the business owner to handle tedious and time-consuming tasks, allowing them to concentrate on their business operations.
Note : we have described in the previous sections how each service of those offers a simplified way of management and does not require the technical intervention of the owner

AWS Management Console : The AWS Management Console provides a web-based interface that allows us to manage and configure AWS resources throughout a user-friendly interface with step-by-step wizards . Therefore, it offers us a simplified way of deploying and provisioning infrastructure components (i.e., provisioning and configuring network resources such as VPCs, subnets, and security groups.
The console provides a visual representation of the network topology, making it easy to understand and manage.
Note : the implementation documentation demonstrates all of this

WordPress as a Content Management System (CMS) : Instead of investing significant resources into custom website development and maintenance, we aimed by choosing WordPress to leverage the fact that is a ready-to-use solution offering different significant advantages:

- ***Flexible Customization :***
  WordPress offers a wide range of themes and plugins that enable flexible customization of the website's design and functionality. Moreover , it offers the opportunity of integrate different features such as SEO optimization, and social media integration, without the need for custom development.

- ***Simplified Content Management :*** WordPress provides an intuitive and user-friendly content management interface, allowing the business owner to easily create, edit, and manage website content. This empowers the business owner to take control of their website's content without relying on technical expertise or external resources.

Note about the project :
The primary objective of this project is not solely centered around the development and functionality of the website itself. Rather, the core focus lies in the exploration and implementation of a robust and efficient technical architecture to support the website. The website serves as a tangible manifestation of the underlying architecture, demonstrating the utilization of cloud technologies, best practices, and design principles.

6 Horizons :

6.1 Things that were planned to be done :

- Implement Traffic Testing: Conduct load testing to evaluate how your website handles increasing levels of traffic. This will help identify performance bottlenecks and optimize the infrastructure accordingly.
- Cost calculation : calculating the costs of the solution by leveraging tools like the AWS Pricing Calculator and Cost Explorer. Explore cost-saving strategies, and right-sizing to optimize resource utilization and reduce expenses.

6.2 Future improvements of the project :

- Implementing caching mechanisms for improved website performance.
- Integrating a content delivery network (CDN) to serve content globally with reduced latency.
- Explore advanced security measures such as web application firewalls (WAF) and intrusion detection systems (IDS).
- Conduct regular security audits and vulnerability assessments to ensure ongoing protection.
- Optimize costs by analyzing resource utilization and leveraging AWS cost optimization tools.
- Implement infrastructure as code (IaC) using tools like AWS CloudFormation or AWS CDK.
- Introduce automated scaling policies to handle traffic spikes efficiently.

The primary objective of this project is not solely centered around the development and functionality of the website itself. Rather, the core focus lies in the exploration and implementation of a robust and efficient technical architecture to support the website. The website serves as a tangible manifestation of the underlying architecture, demonstrating the utilization of cloud technologies, best practices, and design principles.

7 Summary :

In conclusion, this project has aimed to demonstrate the utilization of cloud computing, specifically Amazon Web Services (AWS), in building a high-performing and cost-effective WordPress website. Through the implementation of various AWS services and architectural choices, we have achieved key objectives such as high availability, scalability, security, and simplified management.

By leveraging AWS services like Amazon EC2, Amazon ECS (Fargate), Amazon RDS, Amazon EFS, and others, we have created a robust and resilient infrastructure for hosting the WordPress website. The use of containerization with ECS and Fargate has allowed for easy deployment and scaling of the application, while RDS ensures efficient and reliable database management.

Furthermore, the selection of AWS services and the overall architecture has provided numerous benefits. It has promoted high availability by distributing the workload across multiple Availability Zones and implementing auto-scaling mechanisms. It has enhanced security through the use of security groups, network ACLs, and encryption. Additionally, it has offered simplified management through centralized services like AWS Management Console.

Overall, this project serves as a practical example of how organizations can leverage cloud computing to create highly available, scalable, secure, and manageable websites using AWS services. It demonstrates the power and flexibility of cloud technologies in supporting modern web applications and lays the foundation for further enhancements and optimizations.