WPScaleX

1. Cover Page

Project Title: WPScaleX: WordPress Deployment with Terraform on AWS

Student Name: Sebastian Meisel

Course Name: ber-aws-24-1

Instructor’s Name: Vijay Anandh

Date: 24.02.2025

2. Abstract / Executive Summary

This project focuses on deploying a scalable WordPress environment using Terraform on AWS. The infrastructure includes EC2 instances, an RDS database, an Application Load Balancer, and an Auto Scaling Group. The project aims to automate deployment, ensure high availability, and provide a secure WordPress hosting environment. The document outlines the setup, methodology, implementation, challenges faced, and potential future improvements.

3. Table of Contents

4. Introduction

Background: Deploying WordPress manually can be time-consuming and error-prone. Infrastructure as Code (IaC) using Terraform simplifies the process by automating resource provisioning.

Problem Statement: Manual WordPress deployment lacks scalability and can be inefficient. The goal is to use AWS and Terraform to create a fully automated, scalable, and secure WordPress environment.

Objectives:

- Deploy WordPress using Terraform
- Ensure high availability using an autoscaling group
- Implement security best practices
- Use an RDS instance for database storage

Scope:

- Covers AWS resource provisioning using Terraform
- Excludes WordPress customization and content management

5. Literature Review / Related Work

Various cloud hosting solutions exist for WordPress, such as managed WordPress hosting (e.g., AWS Lightsail, Kinsta). This project leverages AWS native services (EC2, RDS, ALB) to optimize cost and flexibility while maintaining high availability and security.

6. Methodology

Approach and Tools Used:

Terraform is used for provisioning AWS infrastructure. The deployment includes:

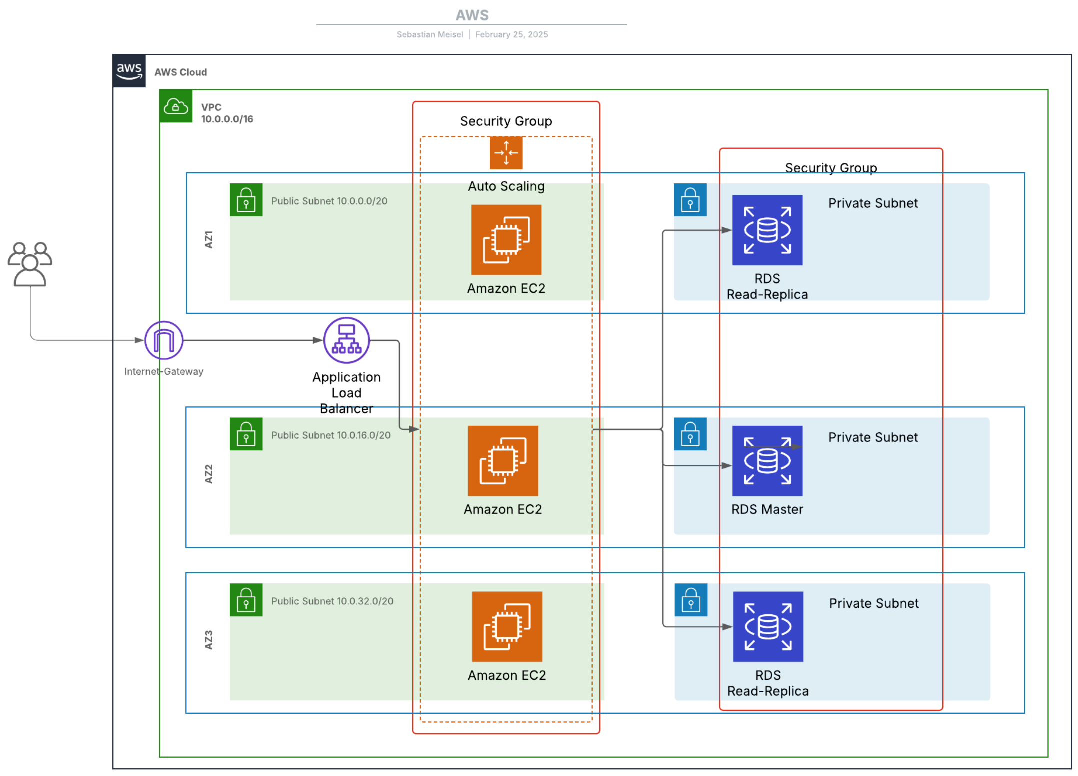
- VPC, subnets, and an internet gateway
- Security groups and IAM roles
- Application Load Balancer and Target Groups
- Auto Scaling Group with EC2 instances
- RDS MySQL Database

Project Development Steps:

1. Define AWS infrastructure components in Terraform
2. Configure networking and security
3. Deploy WordPress and configure auto-scaling
4. Validate deployment and ensure performance

7. System Design / Architecture

Diagrams:



Explanation:

The VPC includes three public subnets where EC2 instances run WordPress. The RDS instance is hosted in private subnets. The ALB distributes traffic, and the Auto Scaling Group ensures high availability.

8. Implementation

Development Process:

The following Terraform resources are created:

Network:

```
resource "aws_vpc" "webVPC" {
   cidr_block = var.vpc_cidr_block
   enable_dns_support = true
   enable_dns_hostnames = true
   instance_tenancy = "default"
   tags = { Name = "webVPC" }
}
resource "aws_subnet" "webSub1" {
   vpc_id = aws_vpc.webVPC.id
   cidr_block = var.subnet_1_cidr
   map_public_ip_on_launch = true
   availability_zone = var.subnet_1_az
   tags = { Name = "webSub1" }
}
resource "aws_security_group" "webSG" {
   vpc_id = aws_vpc.webVPC.id
   ingress {
      from_port = 80
      to_port = 80
      protocol = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
   }
   ingress {
        from_port = 443
        to_port = 443
        protocol = "tcp"
        cidr_blocks = ["0.0.0.0/0"]
    }
    tags = { Name = "webSG" }
}
```

Load Balancer:

```
resource "aws_lb_target_group" "webTG" {
    name        = "webTG"
    port        = 80
    protocol    = "HTTP"
    vpc_id      = aws_vpc.webVPC.id
}
resource "aws_lb" "webBalancer" {
    name                = "webBalancer"
    internal            = false
    load_balancer_type  = "application"
    security_groups     = [aws_security_group.webSG.id]
    subnets             = [aws_subnet.webSub1.id, aws_subnet.webSub2.id,
aws_subnet.webSub3.id]

    tags = {
        Name = "webBalancer"
    }
}
```

```
resource "aws_lb_listener" "http" {
  load_balancer_arn = aws_lb.webBalancer.arn
  port              = 80
  protocol          = "HTTP"

  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.webTG.arn
  }
}
```

ASG:

```
resource "aws_launch_template" "TemplateForAutoScaling" {
 name_prefix   = "TemplateForAutoScaling"
 image_id      = "ami-099da3ad959447ffa" # Amazon Linux 2
 instance_type = "t2.micro"

 network_interfaces {
  security_groups = [aws_security_group.webSG.id]
 }

 user_data = base64encode(<<-EOF
  #!/bin/bash
  dnf update -y
  dnf install -y httpd php php-mysqli mariadb105 wget php-fpm php-json php-devel
php-zip php-xml php-mbstring php-intl php-curl php-bcmath ghostscript

  sed -i "s/AllowOverride None/AllowOverride all/" /etc/httpd/conf/httpd.conf
  sed -i "s/Options Indexes FollowSymLinks/Options all/" /etc/httpd/conf/httpd.conf
  sed -i "s/Options None/Options all/" /etc/httpd/conf/httpd.conf

  systemctl enable httpd
  systemctl start httpd

  usermod -a -G apache ec2-user
  chown -R ec2-user:apache /var/www
  chmod 2775 /var/www
  find /var/www -type d -exec chmod 2775 {} \;
  find /var/www -type f -exec chmod 0664 {} \;

  wget https://github.com/WordPress/WordPress/archive/master.zip
  unzip master -d /tmp/WordPress_Temp
  mkdir -p /var/www/html/wordpress
  cp -paf /tmp/WordPress_Temp/WordPress-master/* /var/www/html/wordpress
  rm -rf /tmp/WordPress_Temp
  rm -f master

  cd /var/www/html/wordpress
  cp wp-config-sample.php wp-config.php
  sed -i "s/database_name_here/${aws_db_instance.wordpressDB.db_name}/"
wp-config.php
  sed -i "s/username_here/admin/" wp-config.php
  sed -i "s/password_here/admin1234/" wp-config.php
  sed -i "s/localhost/${aws_db_instance.wordpressDB.address}/" wp-config.php
```

```
    # BEGIN WordPress

    #echo "# BEGIN WordPress" > /var/www/html/wordpress/.htaccess

    #echo "RewriteEngine On" >> /var/www/html/wordpress/.htaccess

    #echo "RewriteBase /" >> /var/www/html/wordpress/.htaccess

    #echo "RewriteRule ^index\\.php$ - [L]" >> /var/www/html/wordpress/.htaccess

    #echo "RewriteCond %\{REQUEST_FILENAME\} !-f" >>
/var/www/html/wordpress/.htaccess

    #echo "RewriteCond %\{REQUEST_FILENAME\} !-d" >>
/var/www/html/wordpress/.htaccess

    #echo "RewriteRule . /index.php [L]" >> /var/www/html/wordpress/.htaccess

    #echo "# END WordPress" >> /var/www/html/wordpress/.htaccess

    # END WordPress


    chown -R apache:apache /var/www/html

    chmod -R 755 /var/www/html

    chmod -R 755 /var/www/html/wordpress

    chmod -R 775 /var/www/html/wordpress/wp-content/uploads


    systemctl restart httpd

  EOF

  )


  tag_specifications {

    resource_type = "instance"

    tags = {

      Name = "Web-Server"
    }
  }
}


resource "aws_autoscaling_group" "webScale" {
    desired_capacity     = 3
    max_size             = 10
    min_size             = 1
    vpc_zone_identifier = [aws_subnet.webSub1.id, aws_subnet.webSub2.id,
aws_subnet.webSub3.id]


    launch_template {
        id      = aws_launch_template.TemplateForAutoScaling.id
        version = "$Latest"
    }


    target_group_arns = [aws_lb_target_group.webTG.arn]


    tag {
        key                 = "Name"
        value               = "Web-Server"
        propagate_at_launch = true
    }
}


resource "aws_autoscaling_policy" "scale_out" {
    name                   = "scale-out"
    scaling_adjustment     = 1
    adjustment_type        = "ChangeInCapacity"
    cooldown               = 60
    autoscaling_group_name = aws_autoscaling_group.webScale.name
  }
```

RDS:

```
resource "aws_db_subnet_group" "rdsSub" {
  subnet_ids = [aws_subnet.webSub1.id, aws_subnet.webSub2.id, aws_subnet.webSub3.id]

  tags = {
    Name = "rdsSub"
  }
}

resource "aws_db_instance" "wordpressDB" {
  allocated_storage    = 10
  engine               = "mysql"
  instance_class       = "db.t3.micro"
  db_subnet_group_name = aws_db_subnet_group.rdsSub.name
  publicly_accessible = false
  skip_final_snapshot = true
  username             = "admin"
  password             = "admin1234"
  db_name = "wordpressDB"
  vpc_security_group_ids = [aws_security_group.webSG.id]
}
```

Challenges & Solutions:

- Challenge: Ensuring the database is only accessible from EC2 instancesSolution: Use security groups to restrict database access
- Challenge: Maintaining high availabilitySolution: Use Auto Scaling Groups and Load Balancer

9. Results & Analysis

Project Outcomes:

- Successfully deployed WordPress using Terraform
- Achieved automated scaling and security best practices

Testing & Validation:

- Verified ALB routing traffic correctly
- Tested auto-scaling functionality
- Ensured database connectivity from EC2 instances

10. Conclusion & Future Scope

Summary of Key Findings:

Terraform effectively automates the deployment of a highly available WordPress infrastructure.

Future Improvements:

- Implement HTTPS using ACM certificates
- Add CloudFront for caching and performance
- Use S3 for backups

11. References

- Terraform Documentation: https://registry.terraform.io/providers/hashicorp/aws/latest/docs
- AWS EC2: https://aws.amazon.com/ec2/
- WordPress: https://wordpress.org/

12. Appendices

Terraform variables:

```
variable "AWS_REGION" { default = "eu-central-1" }

variable "vpc_cidr_block" { default = "10.0.0.0/16" }

variable "subnet_1_cidr" { default = "10.0.0.0/20" }

variable "subnet_1_az" { default = "eu-central-1a" }
```