Highly Available WordPress on AWS

This project uses Terraform and Ansible to provision and deploy a highly available WordPress environment on Amazon Web Services (AWS). The architecture is designed for redundancy and scalability, distributing resources across multiple Availability Zones to ensure high availability.

Architecture

The architecture consists of the following components:

- **VPC (Virtual Private Cloud):** A dedicated, isolated virtual network in AWS.
- **Subnets:**
  - **Public Subnets:** For components that need to be accessible from the internet, such as the Application Load Balancer and NAT Gateways.
  - **Private Subnets (App):** For the WordPress EC2 instances, which are not directly accessible from the internet.
  - **Private Subnets (Data):** For the database (Amazon RDS) and caching layer (ElastiCache), which are isolated from the application layer.
- **NAT Gateway:** Allows instances in private subnets to initiate outbound traffic to the internet.
- **Application Load Balancer (ALB):** Distributes incoming traffic across the WordPress instances in the private subnets.
- **Auto Scaling Group:** Automatically scales the number of WordPress instances based on traffic demand.
- **WordPress Instances (EC2):** The application servers running WordPress.
- **Amazon RDS:** A managed relational database service for the WordPress database (e.g., MySQL). Configured with a Multi-AZ deployment for high availability.
- **Amazon ElastiCache:** A managed caching service (e.g., Memcached) to improve WordPress performance.
- **Amazon EFS:** A scalable file storage service that provides a shared file system for the WordPress content (plugins, themes, uploads). This ensures all instances in the Auto Scaling Group use the same files.
- **Amazon CloudFront:** A content delivery network (CDN) that caches static content and reduces latency.
- **Amazon S3:** Used to store static assets, such as images and videos, that can be served directly from the S3 bucket to improve performance and reduce the load on the EC2 instances.
- **Amazon Route 53:** A scalable DNS web service that routes traffic to the CloudFront distribution.
- **Bastion Host:** An EC2 instance in a public subnet used for secure access to the instances in the private subnets.

Deployment Steps

1. **Prerequisites:**
   - An AWS account with configured credentials.
   - Terraform installed.
   - Ansible installed.
   - An SSH key pair for the Bastion Host and WordPress instances.
2. **Clone the Repository:**

   ```
   git clone <repository_url>
   cd <repository_directory>
   ```

3. **Configure Terraform Variables:**
   - Edit `variables.tf` to set your desired region, instance types, and other parameters.
   - Ensure the SSH key name matches a key in your AWS account.
4. **Provision Infrastructure with Terraform:**

   ```
   terraform init
   terraform plan
   terraform apply
   ```

   - Terraform will output the public IP of the Bastion Host.
5. **Configure Ansible Inventory:**
   - The `inventory.ini` file will be automatically generated by Terraform. It will contain the private IP addresses of the WordPress instances.
6. **Deploy WordPress with Ansible:**
   - You will need to SSH into the Bastion Host to run the Ansible playbook.
   - Ensure your SSH agent is forwarded so Ansible can connect to the WordPress instances from the Bastion Host.
   - Run the Ansible playbook:

   ```
   # From your local machine
   ssh -A ec2-user@<bastion_host_public_ip>

   # From the bastion host
   ansible-playbook -i inventory.ini playbook.yml
   ```

   - The playbook will install and configure WordPress, set up the EFS mount, and connect to the RDS database.
7. **Finalize Setup:**
   - Update the DNS record in Route 53 to point to the CloudFront distribution.
   - Log in to your WordPress site and complete the setup.

Assumptions

- An AWS account with necessary permissions is available.
- An SSH key pair is already created and imported into AWS.
- The required DNS records are configured in Route 53.
- The CloudFront distribution and S3 bucket for static assets are manually configured after the initial Terraform deployment. (Note: These can also be managed by Terraform but are kept simple for this example.)