Whitepaper: How it works on AWS

Summary

This whitepaper provides a comprehensive guide to the deployment architecture of the website www.bennwokoye.com, hosted on AWS.

The site is built on HTML, CSS, and JavaScript, leveraging AWS S3 for static content hosting, Route 53 for domain management, and CloudFront for content distribution and performance optimization with cache invalidation.

Backend functionality includes AWS API Gateway, Lambda, and SNS, which process user requests and notifications. The entire infrastructure is managed via Terraform, with the state file stored in a remote backend on AWS S3/DynamoDB.

The CI/CD pipeline is powered by AWS CodePipeline integrated into GitHub and is automatically triggered by commits to the main GitHub branch.

Key Components

Static Content Hosting on S3

- **Amazon S3** hosts static content (HTML, CSS, JS, and media files), making the website lightweight, cost-effective, and scalable.
- The S3 bucket is configured to host the website, providing public access to the necessary content. S3's high availability ensures that users can reliably access the website from anywhere in the world.

DNS Management with Route 53

- **AWS Route 53** handles domain registration and DNS resolution for the website.
- The domain www.bennwokoye.com translates human-readable domain names into IP addresses browsers use to access the site.
- Route 53 also supports health checks and failover capabilities to ensure the website remains highly available.

Content Delivery via CloudFront

- **Amazon CloudFront**, a content delivery network (CDN), enhances the website's performance by caching content at edge locations closer to users, thereby reducing latency and load times.
- CloudFront is connected to the S3 bucket as its origin, allowing it to serve static content efficiently.
- **Cache invalidation** is implemented to ensure that website updates (new files, changes to HTML, JS, or CSS) are reflected immediately across all CloudFront edge locations

whenever new commits are made to the GitHub repository. This ensures users always see the latest version of the website.

API Gateway and Lambda for Backend Logic

- **AWS API Gateway** acts as the frontend API for any user interaction that requires server-side logic, in this case, the form submission event handling. API Gateway exposes a REST API to the static website, enabling dynamic interactions with the backend.
- **AWS Lambda** function is triggered by API Gateway to handle business logic in a serverless manner. Lambda is used to process contact form submission requests.
- Using Lambda eliminates the need to manage servers, and the application can scale automatically based on traffic.

Notification Handling with SNS

- **Amazon SNS (Simple Notification Service)** is integrated into the Lambda function to handle notifications. When a user submits a form, an SNS topic is triggered to email the website owner or system administrator.

CI/CD Automation with AWS CodePipeline

- The CI/CD pipeline is powered by **AWS CodePipeline**. Any time a new commit is pushed to the **main branch** of the GitHub repository, the CodePipeline is automatically triggered to initiate the deployment process.
- The pipeline performs the following stages:
    1. **Source**: Fetches the latest code from GitHub.
    2. **Build**: Follow the defined build steps.
    3. **Deploy:** Pushes updated files to the S3 bucket.
    4. **Invalidate CloudFront Cache**: Ensures users are served the latest content from edge locations.
- This automated workflow ensures that updates are deployed quickly, consistently, and without manual intervention.

Infrastructure as Code with Terraform

- **Terraform** defines, manages, and provides the website's infrastructure. Terraform configuration files describe all AWS resources, including S3, Route 53, CloudFront, API Gateway, Lambda, and SNS.
- **Remote State Backend**: Terraform state file is stored in an S3 bucket & DynamoDB, ensuring that the infrastructure's state is consistent and shareable across development environments.
- This approach allows the entire infrastructure to be version-controlled, modular, and reusable. Any changes to the infrastructure are managed through Terraform and can be rolled out systematically.

Benefits of the Architecture

Cost Efficiency:

- Utilizing S3 for static content hosting and a serverless architecture with Lambda makes this deployment highly cost-effective. The serverless approach reduces the need to maintain infrastructure, while CloudFront helps optimize costs related to content delivery.

High Availability:

- S3 offers 99.99% availability, and CloudFront's global network of edge locations ensures fast and reliable delivery to users worldwide.
- Route 53 and CloudFront health checks provide additional layers of reliability to ensure that the website is always available and quickly accessible.

Scalability:

- Lambda functions and CloudFront edge locations scale automatically to handle increases in traffic, meaning there is no need for manual intervention when traffic spikes occur.

Performance Optimization:

- By using CloudFront as a CDN, the website's static assets are served from edge locations closest to users, minimizing latency and improving load times.

Security:

- By using **CloudFront** with **S3**, the website benefits from DDoS protection and AWS Shield, ensuring the content is secure.
- API Gateway offers security features such as throttling and request validation to protect backend resources from malicious activity.

Automation:

- **AWS CodePipeline** and **Terraform** automate deployments and infrastructure changes, reducing the risk of manual errors and allowing for consistent updates.

Conclusion

The deployment of www.bennwokoye.com on AWS showcases a modern, cost-effective, and scalable architecture for a static website with a backend logic. The solution ensures high availability, performance, and security by leveraging services like S3, CloudFront, API Gateway, Lambda, SNS, and Terraform while automating the deployment process with AWS CodePipeline.

This architecture represents a proven solution for creating a fast, secure, scalable web presence. By embracing a serverless, automated deployment pipeline, you reduce operational overhead and ensure your application can easily handle traffic demands. Whether you're building a portfolio site, a content hub, or an e-commerce platform, **this architecture can be tailored to meet your unique needs while maintaining robust performance and security.**

Send a message or connect with me on LinkedIn if you want to learn more or need expert guidance in implementing cloud-based infrastructure for your web applications.