**Project Documentation: Cloud Infrastructure for Recipe Storage & Distribution**

**Overview**

This project provisions a secure and scalable infrastructure using AWS services for storing, archiving, processing, and distributing recipe files globally. The infrastructure is built with Terraform, ensuring automation, version control, and reproducibility.

**AWS Cloud Services Used & Their Functionality**

1) **Amazon S3 (Simple Storage Service)**

    - **S3 Bucket for Recipe Storage** (automation-recipe-storage)
    - Stores uploaded recipe files.
    - Versioning enabled to keep track of file changes.
    - Server-side encryption (AES-256) to secure data.

- **S3 Bucket for Archives** (automation-recipe-archives)
- Stores processed and archived recipe files.
- Versioning enabled for tracking changes.

**2)** S3 Lifecycle management for both buckets for effective cost optimization
    - Files in S3 bucket **automation-recipe-storage move to Glacier** after 30 days.
    - Files in recipe_archives **expire after 1 year**.

3) **AWS Lambda**

**Used for: Processing recipe files (e.g., zipping and moving them to archives)**

- Lambda function (zip_recipes)
- Triggered when a new file is uploaded to recipe_storage.
- Compresses the file and stores it in recipe_archives.
- Uses an **IAM Role** (lambda_exec) with permissions to read/write S3.

**Lambda Function Role in the Workflow**

- A user uploads a new recipe file → recipe_storage S3 bucket.
- **S3 Event Notification** triggers **Lambda** automatically.
- **Lambda downloads the file**, compresses it, and **uploads the ZIP** to recipe_archives.
- **CloudFront** serves archived files securely to users.

- **S3 Event Notification**
- **S3 triggers Lambda** when a new file is uploaded.
- **Uses S3 bucket** notification to invoke Lambda.

4) **AWS IAM (Identity & Access Management)**
   **Used for: Securing access to AWS resources**

   - **IAM Role for Lambda (lambda_exec)**
   - Allows Lambda to interact with S3 and CloudWatch.
   - IAM Policy for Lambda to access S3
   - Grants Lambda permissions to:
   - Read/write recipe files in both S3 buckets.

   **IAM Policy for Lambda to write logs**

   - Allows Lambda to log execution details in CloudWatch.
   - S3 Bucket Policy
   - Allows CloudFront to securely retrieve archived recipe files.

5) **Amazon CloudFront**

   **Used for: Secure global distribution of archived recipe files**

   - **CloudFront Distribution (recipe_distribution)**
   - Provides fast, secure access to recipe archives.
   - Uses **Origin Access Identity (OAI)** to **restrict direct access** to S3.
   - **HTTPS enforced** for security.

6) **AWS CloudWatch**

**Used for: Monitoring & Logging**

- CloudWatch logs Lambda execution for debugging and monitoring in the log group **/aws/lambda/zip_recipes**

**Key Features & Functional Flow of the Project**

1. **File Upload: Users upload recipe files to recipe_storage S3 bucket.**
2. **File Processing:**
    - **S3 triggers Lambda when a new file is uploaded.**
    - **Lambda compresses the file and moves it to recipe_archives.**
3. **Lifecycle Management:**
    - **Files in recipe_storage move to Glacier after 30 days.**
    - **Files in recipe_archives expire after 1 year.**
4. **Secure Distribution via CloudFront:**
    - **Clients access archived recipes through cloudfront.**
    - **OAI ensures direct S3 access is blocked.**

**Security Considerations**

- **IAM Policies restrict access to only necessary actions.**
- **S3 Encryption ensures data is protected at rest.**
- **CloudFront OAI prevents unauthorized S3 access.**

**Note > "In this project, I opted not to use CloudFront Signed URLs for Access Control and AWS Site-to-Site VPN. While I initially considered implementing Site-to-Site VPN, I found the configuration process to be time-consuming. As a result, I prioritized other AWS services and deferred its implementation through Terraform for the time being."**

**Additionally, we encourage you to provide brief insights into the benefits, potential pitfalls, and important considerations surrounding the following topics:**

1) IaC

2) IoT and Edge computing

3) Cost calculations and optimisations

1. **Infrastructure as Code (IaC)**

- **Benefits:**
- Automation & Consistency – Automates infrastructure deployment, reducing human error.
- Scalability – Easily replicates infrastructure across environments (dev, test, production).
- Version Control – Treats infrastructure like code, enabling rollback and tracking changes.
- Faster Deployments – Speeds up infrastructure provisioning with templates and scripts.

**Pitfalls:**

- Complexity – Learning IaC tools (Terraform, CloudFormation, Pulumi) can be challenging.
- Misconfigurations – A small mistake in code can deploy incorrect resources at scale.
- State Management Issues – Keeping track of infrastructure state across teams requires careful handling.

**Important Considerations:**

- Use GitOps to manage IaC changes efficiently.
- Implement automated testing (e.g., terraform plan, policy checks) before applying changes.
- Follow the principle of least privilege when managing IAM permissions in IaC.

2) **Internet of Things (IoT) and Edge Computing**

**Benefits:**

- Low Latency – Edge computing processes data closer to the source, reducing response times.
- Bandwidth Savings – Less data is sent to centralized cloud services, reducing costs.
- Reliability – Works even with intermittent connectivity by processing data locally.
- Scalability – Supports a vast network of IoT devices in real-time applications.

**Pitfalls:**

- Security Risks – More entry points mean increased attack surfaces for hackers.
- Device Management Complexity – Keeping firmware and software updated across devices is challenging.
- Data Privacy Concerns – Edge devices handling sensitive data may require regulatory compliance (GDPR, HIPAA).

**Important Considerations:**

- Use strong encryption (TLS, AES) for data transmission and storage.
- Implement zero-trust security models for device authentication.
- Optimize data filtering at the edge to reduce cloud processing and storage costs.

**3) Cost Calculations and Optimizations**

**Benefits:**

- Cost Transparency – Helps businesses understand where cloud spend is going.
- Better Budgeting – Enables accurate forecasting of cloud expenses.
- Optimized Resource Usage – Identifies underutilized resources and rightsizes them.
- Cost Savings – Helps reduce unnecessary expenses via reserved instances, auto-scaling, and spot instances.

**Pitfalls:**

- Unexpected Costs – Misconfigured auto-scaling or forgotten resources can lead to bill shock.
- Complex Pricing Models – Cloud providers have intricate pricing structures that are hard to track.
- Over-provisioning – Allocating more resources than needed leads to waste.

**Important Considerations:**

- Use AWS Cost Explorer,AWS Compute Optimizer, Azure Cost Management, or GCP Billing Reports to track spending.
- Implement auto-scaling to match demand and avoid over-provisioning.
- Leverage spot instances and reserved capacity for predictable workloads.
- Regularly review idle resources (unused volumes, old snapshots) and delete unnecessary ones.