TASK 2

**Pain Point 1: Heavy Reliance Manual Operations and Resultant Inconsistency**

* **Problem with current configuration:** The current workflow is mainly that of manually configuring servers and deploying applications with command-line scripts. This ‘click-ops’ approach is slow, prone to human error, and creates an infrastructure where consistency is hard to establish (but is needed given the testing demands). There is no audit trail to see who changed what, making troubleshooting painfully difficult and scaling impossible. This directly contradicts the primary goal to "automate as much as possible."
* **The Solution - Infrastructure as Code (IaC):**  
  We will solve this by implementing an **Infrastructure as Code (IaC)** workflow using **Terraform** as the tool and **Terraform Cloud** as the collaboration and automation platform. All infrastructure will be defined using HashiCorp Configuration Language (HCL) and stored in a central **GitHub** repository, which will be the single ‘source of truth’. This repository will then be connected to a **Terraform Cloud** workspace, which provides a secure, remote environment for running terraform plan and apply. When an engineer pushes an approved change to the GitHub repository, Terraform Cloud will automatically trigger a run, providing a repeatable, auditable, and reliable automated deployment process.
* **Potential Drawbacks:** The main drawback is the initial investment in training. The engineering team will need to learn the principles of IaC and the syntax for CloudFormation or Terraform. This requires an upfront commitment to education. There may also be the need to hire additional cloud architects
* **Net Positive:** This is a massive net positive and the foundation for the entire project. It directly solves Balerica’s stated goals of automation and reducing click-ops, while making the entire infrastructure more secure, stable, and scalable.

**Pain Point 2: Insecure Desktop Management**

* **The Problem:** The current process of manually reimaging 30 desktops and having proctors launch and fix the secure browser is a major operational bottleneck. It doesn't scale to five countries and is a poor use of the proctors' time. Furthermore, running a secure exam application on a general-purpose desktop that could expose significant security risks.
* **The Solution - Application Streaming with Amazon AppStream 2.0:**  
  Instead of installing the exam application locally, we will use **Amazon AppStream 2.0** to stream it to the desktops. The application is installed just once on a secure, centrally managed "golden image" in AWS. When a student sits for an exam, a secure streaming session is initiated, and the application's user interface appears on the local monitor as if it were running there. However, the application and all its data are actually executing securely on AWS servers. This means to update the application for all 150+ global desktops, an engineer only needs to update the single central image. This completely eliminates the need for local reimaging for application updates and allows administrators to monitor sessions from a central console.
* **Potential Drawbacks:** AmazonAppStream 2.0 streaming to various global testing centers requires a stable internet connection at each testing center. Internet or network outage could interrupt the ability to stream the application to a given center.
* **Net Positive:** This is a transformative improvement. It directly solves the goals of cutting down on reimaging times, enabling remote control capabilities, and improving the security of the exam environment.

**Pain Point 3: Unreliable Application and "Best Effort" Backups**

* **The Problem:** For an educational testing company, the user data, exam questions and student results are critical sets of information. The current "best effort" on-premises backup strategy is extremely risky and could lead to catastrophic data loss. Furthermore, re-configuring the application to work on a global scale will be extremely difficult and time consuming.
* **The Solution - Re-architect with Managed Databases and Serverless Functions:**  
  We will migrate the application's backend to a modern, resilient, and secure serverless architecture. The database of test questions and user data will be moved to **Amazon RDS** (Relational Database Service) a fully managed database service (or DynamoDB should you wish to use a NoSQL database). AWS will handle all the backups automatically, with the option for continuous backups and the ability for point-in-time recover within 35 days. This completely solves the data protection problem. The application's operations logic will be rewritten as a set of **AWS Lambda** functions fronted by an **Amazon API Gateway**. This approach removes the need for hardcoded credentials to access the database and instead utilizes AWS’s suite of highly secure IAM services, enabling a highly scalable backend that Balerica only pays for when it's actually being used.
* **Potential Drawbacks:** This requires a one-time refactoring of the application's backend from its current state to a serverless model (taking into consideration the API and lambda integrations with the database).
* **Net Positive:** This proposal is critically beneficial for the global reach aspirations of Balerica cloud. This implementation transforms their core application from a liability into a highly available, scalable, and secure asset that meets Balerica’s global reach goal.