Top Survey AWS migration plan

Thinking about the increase in requests and the application's high availability, a **lift-and-shift** strategy is not recommended.
Constructing a structured infrastructure to profit from the AWS cloud is a better solution.
All the migration solution was designed to focus on AWS as a cloud provider, Gitlab as a CICD tool, and Helm charts to deploy the application in the EKS cluster, but could easily ported to other providers and CICD tools.

To create the AWS infrastructure I decided to use Terraform thinking about a way to re-create all the services in different regions and AWS accounts.

Detailed Steps for Migration

1. Network Design (VPC Setup)

- **Amazon VPC**: Create a Virtual Private Cloud (VPC) with multiple Availability Zones (AZ) for high availability.
    - **Subnets**: Define public subnets for the application tier and private subnets for the database tier.
    - **Security Groups & NACLs**: Implement strict security groups for database and application layers, ensuring secure access.

2. Domain and DNS Migration

- **Amazon Route 53**: Migrate the domain to Route 53 for scalable DNS management. This allows better control and performance routing. Update the DNS to point to AWS resources once the migration is complete.

3. Database Migration

- **Amazon RDS (Managed Database Service)**:
    - Use **Amazon RDS for MySQL/PostgreSQL** (depending on the current DB type) to manage the database with automatic backups, patching, and scaling.
    - **AWS Database Migration Service (DMS)**: Use DMS to migrate the on-premise database to AWS RDS with minimal downtime.
    - Enable **Multi-AZ** for high availability and automatic failover.
    - Configure **Read Replicas** for scaling read-heavy workloads.

4. Application Layer Migration

- **Amazon EKS (Elastic Kubernetes Service)**:

    - Since the application is already containerized, leverage EKS to run containers in AWS.
    - Ensure autoscaling policies are set to handle the millions of users accessing the service.
    - Implement **Auto Scaling** groups to ensure dynamic scaling based on load.

5. Compute Resources

- **EC2**:
    - For EC2, choose an instance type optimized for the application workload (such as compute-optimized for high traffic).

6. Storage and Backup

- **Amazon S3**:
    - Store survey responses and static content (images, assets) in Amazon S3, which provides scalability and durability.
    - Enable **S3 Versioning** and **Lifecycle Policies** for efficient cost management and backup.

6. Load Balancing & CDN

- **Elastic Load Balancer (ELB)**: Use a **Network Load Balancer (NLB)** to distribute incoming traffic to your application running EKS.
- **Amazon CloudFront (CDN)**: Deploy CloudFront to cache static content and improve performance for users globally, reducing latency.

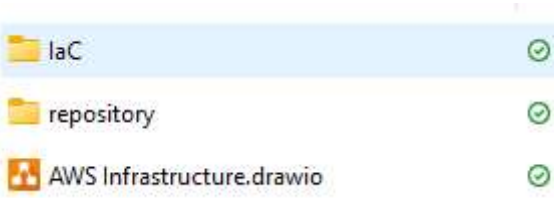
7. Monitoring and Security

- **Amazon CloudWatch**: Set up CloudWatch for real-time monitoring of the application, including CPU, memory, and disk utilization.
- **AWS CloudTrail**: Enable CloudTrail for logging and tracking API calls for auditing purposes.
- **AWS WAF (Web Application Firewall)**: Protect the application from common web exploits (e.g., SQL injection, DDoS).
- **AWS Shield**: Use AWS Shield for DDoS protection to prevent outages due to malicious attacks.

8. Optimization Post-Migration

- Once the application is stable in AWS, re-evaluate the architecture for cost optimization and improved scalability. Consider:
    - Use the Terraform script to create a Developer and Test environment

Resource structure:



IaC → It has all the terraform code to create the AWS resources.
repository → It has the basic project structure including Helm templates, Gitlab CI/CD Yaml file, and the Docker file.