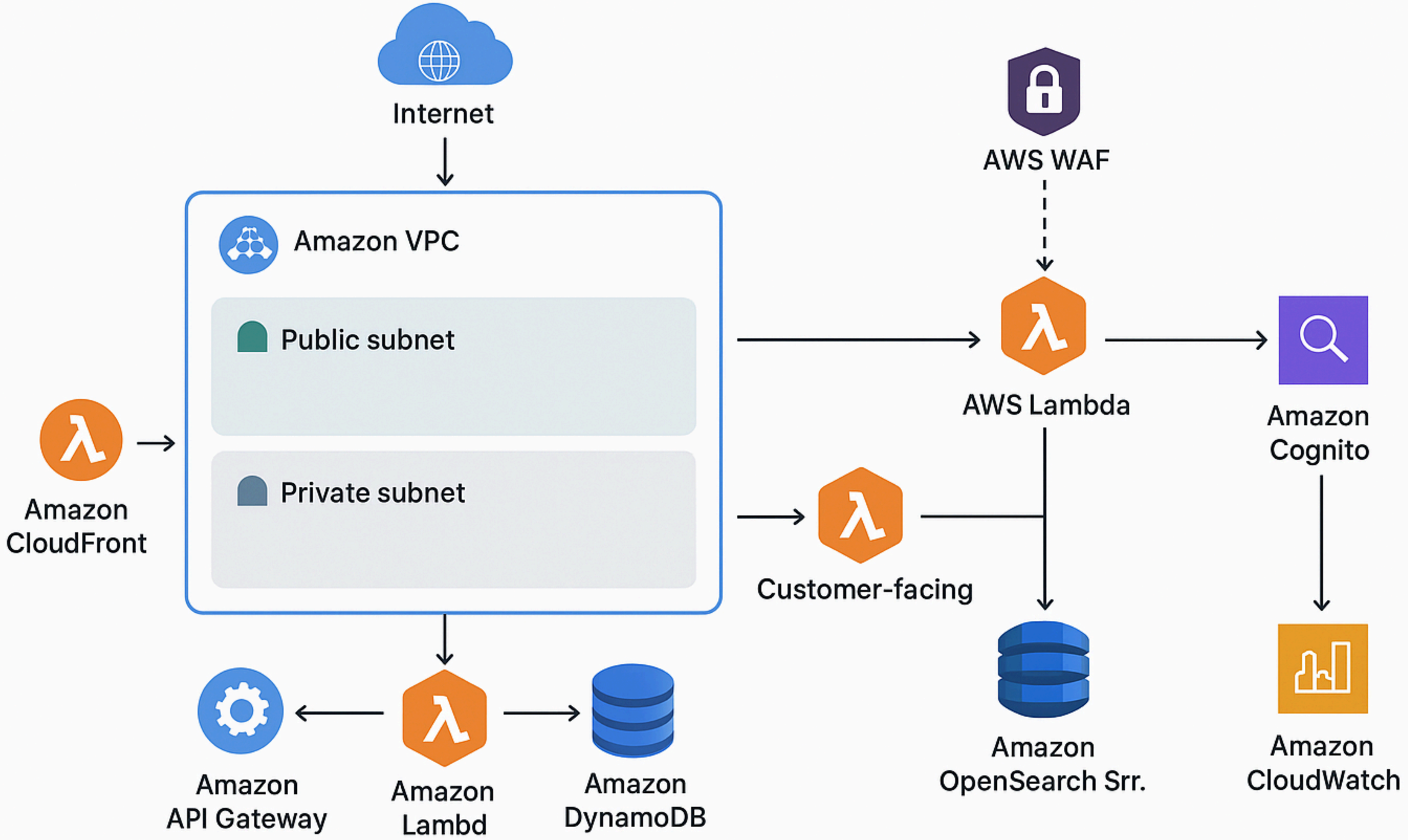
3D Sports Warehouse Platform on

AWS

Andrea Ndzekeni


Internet
AWS WAF
Amazon VPC
Public subnet
Private subnet
Amazon CloudFront
AWS Lambda
Amazon Cognito
Customer-facing
Amazon API Gateway
Amazon Lambd
Amazon DynamoDB
Amazon OpenSearch Srr.
Amazon CloudWatch

3D Sports Warehouse Platform on AWS – Technical Project

1. Project Overview

1.1 Purpose

The 3D Sports Warehouse Platform is designed to provide customers with an immersive, interactive experience for exploring 3D models of sports equipment, apparel, and footwear before purchase. Simultaneously, it supports backend warehouse operations such as inventory management, order fulfillment, and returns processing. The platform leverages AWS's serverless and managed services to ensure scalability, high availability, and cost efficiency.

1.2 Objectives

- Deliver a **highly available**, **secure**, and **scalable** e-commerce platform.
- Enable **real-time 3D visualization** of sports products.
- Integrate **warehouse management** and **inventory tracking** with customer-facing operations.
- Optimize for **low latency**, **global reach**, and **minimal operational overhead**.

1.3 Target Users

- **Customers:** Browse, customize, and purchase sports products.
- **Warehouse Staff:** Manage stock, picking, and returns.
- **Administrators:** Monitor performance, security, and analytics.

2. Technical Architecture

2.1 Architecture Overview

The platform follows a **serverless, event-driven architecture** hosted entirely on AWS. It separates the **presentation layer**, **application logic**, and **data layer** while integrating with AWS-managed services for compute, storage, and analytics.

2.2 Architecture Components

Referencing the **diagram on page 1 of the original document**, the architecture includes:

- **Amazon CloudFront** for global content delivery of static assets and 3D models.
- **Amazon S3** for hosting static web content and storing 3D assets (.glb/.gltf).
- **Amazon API Gateway** as the unified entry point for RESTful APIs.
- **AWS Lambda** for serverless backend compute functions.
- **Amazon DynamoDB** for low-latency NoSQL data storage.
- **Amazon OpenSearch Service** for catalog search and filtering.
- **Amazon Cognito** for user authentication and authorization.
- **Amazon VPC** for secure, multi-AZ networking.
- **AWS WAF** and **AWS Shield** for web application protection.
- **Amazon CloudWatch** and **AWS X-Ray** for observability and tracing.

3. AWS Services Used and Their Roles

4. Security Considerations

4.1 Identity and Access Management

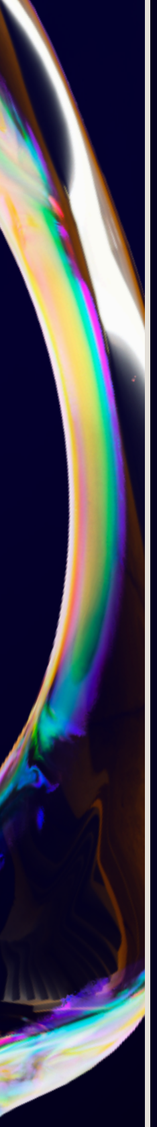
- Implement **least-privilege IAM roles** for Lambda, API Gateway, and DynamoDB.
- Use **Cognito user pools** for authentication and **federated identities** for third-party logins.
- Enforce **MFA** for administrative access.

4.2 Data Protection

- Encrypt all data at rest using AWS KMS (S3, DynamoDB, OpenSearch).
- Encrypt all data in transit using TLS 1.2+ via ACM-managed certificates.
- Apply S3 bucket policies to restrict public access.

4.3 Network Security

- Deploy resources in private subnets for backend services.
- Use NAT Gateways or VPC endpoints for secure outbound access.
- Protect APIs and assets with AWS WAF and AWS Shield Standard.



4.4 Compliance

- Ensure compliance with **GDPR**, **SOC 2**, and **ISO 27001** standards.
- Enable **CloudTrail** for audit logging and governance.

5. Scalability and Performance

5.1 Horizontal and Vertical Scaling

- **Lambda** and **DynamoDB** scale automatically based on demand.
- **OpenSearch** supports horizontal scaling for large catalog queries.
- **API Gateway** throttling prevents overload during flash sales.

5.2 Performance Optimization

- Use **CloudFront edge caching** for 3D assets and static content.
- Enable **Lambda provisioned concurrency** for critical APIs (checkout, cart).
- Optimize 3D models using **Level of Detail (LOD)** and **progressive loading**.

5.3 Load Testing

- Conduct synthetic load tests using **AWS Distributed Load Testing** or **Artillery**.
- Monitor latency and throughput via **CloudWatch metrics**.

6. Cost Optimization

6.1 Serverless Efficiency

- Pay-per-use model for **Lambda**, **API Gateway**, and **DynamoDB** eliminates idle costs.
- Use **DynamoDB Standard Access mode** for balanced cost-performance.
- Apply **S3 lifecycle policies** to transition older assets to **S3 Glacier**.

6.2 Data Transfer Optimization

- Use **VPC endpoints** to reduce NAT Gateway data transfer costs.

- Use DynamoDB Standard Access mode for balanced cost-performance.
- Apply S3 lifecycle policies to transition older assets to S3 Glacier.

6.2 Data Transfer Optimization

- Use VPC endpoints to reduce NAT Gateway data transfer costs.
- Cache frequently accessed data in CloudFront to minimize origin fetches.

6.3 Monitoring and Budgeting

- Set AWS Budgets and Cost Anomaly Detection alerts.
- Use Cost Explorer for continuous cost tracking and optimization.

7. Deployment Steps

7.1 Infrastructure Setup

1. Create VPC with public and private subnets across multiple AZs.
2. Configure Route 53 for domain and DNS routing.
3. Deploy S3 buckets for static content and 3D assets.
4. Set up CloudFront distribution with S3 as the origin.
5. Deploy API Gateway endpoints for customer and warehouse APIs.

7.2 Backend Deployment

1. Implement **Lambda functions** for business logic (orders, inventory, returns).
2. Configure **DynamoDB tables** with partition keys for scalability.
3. Set up **OpenSearch** for catalog indexing and search.
4. Integrate **Cognito** for authentication and user management.

7.3 Security and Monitoring

1. Apply **WAF rules** for API Gateway and CloudFront.
2. Enable **CloudWatch** dashboards and **X-Ray tracing**.
3. Configure **KMS encryption** and **IAM policies**.

7.4 CI/CD Pipeline

- Use **AWS CodePipeline**, **CodeBuild**, and **CodeDeploy** for automated deployments.
- Implement **Infrastructure as Code (IaC)** using **AWS CloudFormation** or **Terraform**.
- **8. Maintenance and Operations**
- **8.1 Monitoring and Alerts**
- Use **CloudWatch Alarms** for Lambda errors, API latency, and DynamoDB throttling.
- Enable **X-Ray** for distributed tracing of API calls.
- **8.2 Data Management**
- Schedule **DynamoDB backups** and **S3 versioning**.
- Periodically reindex **OpenSearch** for optimal performance.
- **8.3 Continuous Improvement**
- Conduct **monthly performance reviews** and **security audits**.
- Update **Lambda runtimes** and dependencies regularly.
- Review **cost reports** and adjust resource configurations as needed.
- **10. References**

- Amazon Web Services (2025). **AWS Cloud Computing Services**. Available at: **aws.amazon.com**
- Lucid Software Inc. (2025). **Lucidchart**. Available at: **lucidchart.com**
- AppMySite (2022). **How to create an eCommerce shopping app like Amazon?** Available at: **appmysite.com**

The End