DhakaCart E-Commerce Reliability Challenge

Background

DhakaCart is a growing e-commerce company in Dhaka specializing in electronics.
The site attracts about **5,000 daily visitors** and generates roughly **50 lakh BDT per month** in sales. Last week, DhakaCart ran a major online sale with **5 lakh BDT in marketing spend**, expecting **50,000 visitors**. When the sale started, the website **crashed for seven hours**, causing an estimated **15 lakh BDT revenue loss**. Thousands of customers left angry messages and switched to competitors like **Daraz**.

Next Monday, DhakaCart will launch an even larger **Eid Sale**, spending **8 lakh BDT on marketing** and expecting **100,000 visitors**. If the website fails again, management plans to **shut down all online operations**, which would result in **15 job losses**.

Current System Situation

Hardware & Hosting

- The entire website runs on a **single desktop computer (2015)** with **8 GB RAM**, located in an office room with **broken air conditioning**.
- During the last sale, the **CPU overheated to 95 °C** and **shut down automatically**.
- There is **no backup server** — if this computer fails, the entire business stops.
- The system struggles beyond **5,000 concurrent visitors**.

Deployment & Maintenance

- Every code update requires **stopping the entire site for 1–3 hours**.
- The developer manually transfers files using **FileZilla** from his laptop to the production computer.
- Deployments often break because **local and production environments differ**.
- There is **no testing or staging environment** and **no rollback mechanism**.
- The site goes offline **2–3 times per week** for updates, costing customers and revenue.

Monitoring & Logging

- There is **no monitoring system**.
- Downtime is discovered only when **customers call to complain**.
- Once, the site was **down for 5 hours overnight** without notice.
- The developer manually inspects **500 MB log files** to diagnose issues, taking **4+ hours per incident**.

Security & Data Management

- **Database passwords are hard-coded** inside the source code.
- The **database is publicly accessible**, with **no firewall protection**.
- **No HTTPS** — all user and payment data travel in plain text.
- **Weak password encryption** and **no login rate-limiting**.
- Sensitive customer data is stored insecurely, creating potential **legal and privacy liabilities**.

Source Code & Backup Practices

- Code exists only on the developer's **laptop**, the **production computer**, and **occasionally in Gmail attachments**.
- There is **no version control system (e.g., Git)**.
- Six months ago, the developer's laptop got infected with a virus, **losing two weeks of work**.
- Database backups are done manually every Sunday to an **external hard drive**, which recently **failed** — losing multiple backups.
- If either machine fails, **critical code and data could be lost permanently**.

Your Mission

Transform DhakaCart's fragile single-machine setup into a **resilient, scalable, and secure cloud-based e-commerce infrastructure** that can:

- **Handle 100,000+ concurrent visitors** without downtime
- **Survive hardware or software failures** automatically
- **Deploy updates safely and continuously**
- **Protect customer data** through strong security and compliance
- **Enable monitoring, logging, and recovery** for operational transparency

---

Key Goals and Requirements

1. Cloud Infrastructure & Scalability

- Migrate the system to the **cloud** with **redundancy and load balancing**.
- Run **multiple instances** of the website behind a **load balancer**.
- Enable **auto-scaling** to adjust capacity during traffic surges.
- Protect the database with **private subnets** and **firewalls**.
- Define everything using **Infrastructure-as-Code (IaC)** for easy replication and recovery.

---

2. Containerization & Orchestration

- **Containerize** all components — React frontend, Node.js backend, database, and caching layer.
- Use an orchestration system (e.g., **Kubernetes or Docker Swarm**) to:
    - Maintain multiple healthy replicas
    - Perform **health checks and self-healing**
    - Enable **rolling updates** without downtime

---

3. Continuous Integration & Deployment (CI/CD)

- Implement a **fully automated CI/CD pipeline**:
    - On each code commit: run tests → build containers → deploy automatically
    - Use **rolling or blue-green deployments** for zero downtime
    - Add **automatic rollback** if errors occur
    - Send **notifications** for deployment status and failures
- Target: Reduce 3-hour manual updates to **10-minute safe automated deployments**.

---

4. Monitoring & Alerting

- Deploy monitoring tools (e.g., **Prometheus + Grafana**, **CloudWatch**, or similar).
- Create **real-time dashboards** with system health indicators: CPU, memory, latency, requests, and errors.
- Use **color-coded status (green/yellow/red)** for quick checks.
- Configure **alerts via SMS, email, or chat** for anomalies such as high CPU, failed health checks, or low disk space.

---

5. Centralized Logging

- Aggregate logs from all servers using tools like **ELK Stack (Elasticsearch, Logstash, Kibana)** or **Grafana Loki**.
- Support quick searches:
    - *“Errors in the last hour”*
    - *“Requests from a specific customer”*
- Enable visual trend analysis and pattern detection.

---

6. Security & Compliance

- Manage all passwords and API keys using **secrets management** (e.g., Vault, AWS Secrets Manager).
- Enforce **HTTPS (SSL/TLS)** for encrypted traffic.
- Apply **network segmentation** to isolate the database.
- Use **strong password hashing**, **rate-limiting**, and **role-based access control (RBAC)**.
- Add **container image and dependency vulnerability scanning** in CI/CD.

---

7. Database Backup & Disaster Recovery

- Automate **daily backups** stored in **secure, redundant locations**.
- Support **point-in-time recovery** and test restoration regularly.
- Consider **database replication** for automatic failover during outages.

---

8. Infrastructure as Code (IaC)

- Represent all resources (servers, networks, databases, firewalls, etc.) in code using **Terraform** or **Pulumi**.
- Version control all configurations in **Git**.
- Allow quick provisioning, replication, or full recovery from code alone.

---

9. Automation & Operations

- Script server provisioning, software setup, and configuration using **Ansible** or similar tools.
- Automate **routine maintenance** like log rotation, patching, and security updates.
- Simplify new-developer onboarding — setup with just a few commands.

---

10. Documentation & Runbooks

- Create clear, accessible documentation:
    - Architecture diagrams
    - Setup and deployment guides
    - Troubleshooting and recovery runbooks
    - Emergency procedures for outages
- Ensure even junior engineers can understand and operate the system.

Deliverable Scope

Build a **prototype or reference implementation** that demonstrates:

- Cloud-hosted, load-balanced, auto-scaling infrastructure
- Containerized and orchestrated application
- Automated CI/CD pipeline with monitoring, logging, and backups
- Secure network, secrets, and HTTPS
- Fully reproducible IaC definitions

Flexibility & Innovation Welcome

The constraints in this challenge are a helpful baseline—not handcuffs. You may propose any alternative architecture, tools, or workflows (e.g., different provisioning, schedulers, data planes, or scaling controllers) if you can justify them clearly and show that they improve cost efficiency, reliability, or operability. Your design will be fully accepted—even if it diverges from the brief—provided you.

In short: you have near-total freedom to design a more effective, cost-optimized solution—just make it defensible with solid reasoning and evidence.