AWS Full-Stack Web Application Deployment Lab

Lab Overview

Deploy a highly available, scalable PHP web application with MySQL database on AWS infrastructure, implementing monitoring, automated backups, and change notifications.

Estimated Time: 4-6 hours
Difficulty Level: Advanced

Scenario

You are a DevOps engineer at **XYZ Solutions**, a growing e-commerce startup. The company has decided to migrate their legacy on-premises inventory management system to AWS. Your task is to design and deploy a cloud-native architecture that ensures high availability, scalability, automatic monitoring, and disaster recovery capabilities.

The application is a PHP-based inventory management system that connects to a MySQL database. The business requirements demand:

- Zero single points of failure
- Automatic scaling during traffic spikes
- Real-time monitoring of application and infrastructure health
- Automated backups with 30-day retention
- Instant notifications when infrastructure changes occur
- Custom domain with proper DNS management
- Estimate the cost as well

Architecture Components Required

1. Database Layer (Amazon RDS - MySQL)

- Create a MySQL RDS instance for the inventory database
- Configure Multi-AZ deployment for high availability
- Set up appropriate security groups
- Create initial database schema for inventory management
- Configure automated backups and maintenance windows

Database Requirements:

- Database name: inventory_db
- Tables needed: products, categories, stock_levels, suppliers
- Sample data should be loaded for testing

---

2. Application Layer (EC2 Instances)

- Launch EC2 instances running Amazon Linux 2 or Ubuntu
- Install and configure Apache/Nginx web server
- Install PHP and required extensions (php-mysqli, php-json, php-curl)
- Deploy the PHP application code
- Configure application to connect to RDS MySQL database
- Create a custom AMI from your configured instance

Application Requirements:

- PHP application should display inventory data from MySQL
- Implement at least 3 pages: Dashboard, Product List, Add Product
- Configure proper file permissions and ownership
- Set up application logs in /var/log/application/

---

3. Load Balancing (Application Load Balancer)

- Create an Application Load Balancer in public subnets
- Configure target groups for EC2 instances
- Set up health checks for application availability
- Configure listener rules (HTTP/HTTPS)
- Implement sticky sessions if needed

Load Balancer Configuration:

- Health check path: /health.php
- Health check interval: 30 seconds
- Healthy threshold: 2 consecutive successes
- Unhealthy threshold: 3 consecutive failures

---

4. Auto Scaling (Auto Scaling Group)

- Create a launch template using your custom AMI
- Configure Auto Scaling Group with min, max, and desired capacity
- Set up scaling policies based on CPU utilization
- Distribute instances across multiple Availability Zones

- Configure lifecycle hooks for instance initialization

Scaling Requirements:

- Minimum instances: 2
- Maximum instances: 6
- Desired capacity: 2
- Scale up when: CPU > 70% for 5 minutes
- Scale down when: CPU < 30% for 5 minutes

---

5. DNS Management (Route 53)

- Register or use an existing domain name
- Create a hosted zone in Route 53
- Configure DNS records to point to your Load Balancer
- Set up appropriate record types (A record or Alias)
- Configure health checks for failover routing (optional advanced task)

DNS Requirements:

- Create A record or Alias record: inventory.yourdomain.com
- TTL: 300 seconds
- Implement proper routing policy

---

6. Monitoring (CloudWatch Dashboard)

- Create a comprehensive CloudWatch Dashboard
- Add metrics for EC2 instances (CPU, Memory, Disk, Network)
- Add RDS metrics (Connections, CPU, Storage, IOPS)
- Add ALB metrics (Request count, Target response time, HTTP errors)
- Set up custom metrics from application logs
- Create CloudWatch Alarms for critical thresholds

Dashboard Requirements: Include widgets for:

- EC2 CPU Utilization (all instances)
- RDS Database Connections
- ALB Request Count and Response Time
- Auto Scaling Group metrics

- Custom application metrics (page views, errors)

Alarms to Create:

- High CPU utilization (>80% for 10 minutes)
- RDS storage space (<20% free)
- ALB unhealthy target count (>0)
- High 5xx error rate from ALB

---

7. Change Notification (Lambda Function)

- Create a Lambda function to detect EC2 state changes
- Configure CloudWatch Events/EventBridge rule to trigger Lambda
- Implement notification logic (SNS, email, or CloudWatch Logs)
- Parse event data to extract relevant information
- Format and send notifications with change details

Lambda Requirements:

- Runtime: Python 3.x or Node.js
- Trigger: EC2 instance state change events
- Actions to monitor: Running, Stopped, Terminated, Launching
- Output: Send detailed notification including instance ID, state, timestamp, tags
- Log all events to CloudWatch Logs

Event Types to Monitor:

- EC2 instance state changes
- Auto Scaling group scaling activities
- Instance launches and terminations

---

8. Backup Strategy (S3)

- Create S3 bucket for application and configuration backups
- Implement versioning on the S3 bucket
- Configure lifecycle policies for cost optimization
- Set up automated backups of application code and configuration files
- Create backup scripts that run via cron or Lambda

Backup Requirements:

- Daily backup of application code from EC2 instances
- Weekly backup of Apache/Nginx configuration
- 30-day retention with transition to S3 Glacier after 7 days
- Backup database using RDS automated snapshots
- Tag all backups with timestamp and environment

S3 Bucket Configuration:

- Enable versioning
- Enable server-side encryption
- Configure lifecycle rules
- Set up bucket policies for access control

Networking Requirements

VPC Configuration

- Create a VPC with CIDR block 10.0.0.0/16
- Create at least 4 subnets across 2 Availability Zones:
    - 2 public subnets (for ALB)
    - 2 private subnets (for EC2 instances)
    - 2 private subnets (for RDS - if using separate subnet group)
- Configure Internet Gateway for public subnets
- Configure NAT Gateway for private subnets
- Set up appropriate route tables
- Implement Security Groups with least privilege access

Security Group Rules Required

1. **ALB Security Group:** Allow HTTP (80) and HTTPS (443) from 0.0.0.0/0
2. **EC2 Security Group:** Allow HTTP from ALB security group, SSH from your IP
3. **RDS Security Group:** Allow MySQL (3306) from EC2 security group only
4. **Lambda Security Group:** If VPC-enabled, allow necessary outbound connections

---

Deliverables

1. Architecture Diagram

Create a detailed architecture diagram showing:

- All AWS services and their relationships
- Network topology (VPC, subnets, routing)
- Security group configurations
- Data flow between components
- Backup and monitoring workflows

2. Infrastructure Documentation

Document the following:

- Complete list of AWS resources created with their IDs
- Security group rules and their justifications
- IAM roles and policies created
- Configuration files used
- Estimated monthly cost breakdown

3. Testing Evidence

Provide screenshots/evidence of:

- Working application accessible via custom domain
- Load balancer distributing traffic to multiple instances
- Auto Scaling group scaling up and down
- CloudWatch Dashboard showing all metrics
- Lambda function triggering on EC2 state changes
- S3 bucket containing backups with proper lifecycle policies
- RDS instance running and accepting connections

4. Operational Procedures

Document procedures for:

- Deploying application updates
- Performing manual backups
- Restoring from backups
- Responding to CloudWatch alarms
- Adding/removing EC2 instances manually
- Database maintenance operations

Testing Scenarios

Scenario 1: High Traffic Load

Simulate high traffic to trigger auto-scaling:

- Use Apache Bench or similar tool to generate load
- Observe ASG launching new instances
- Verify Lambda notifications are sent
- Confirm CloudWatch metrics reflect the scaling event
- Validate load balancer distributes traffic evenly

Scenario 2: Instance Failure

Simulate EC2 instance failure:

- Terminate one running instance manually
- Observe ASG launching replacement instance
- Verify Lambda notification is triggered
- Confirm application remains available during replacement
- Check CloudWatch alarms activation

Scenario 3: Database Connectivity

Test database resilience:

- Verify application connects to RDS successfully
- Perform CRUD operations through the web interface
- Check connection pooling and handling
- Test application behavior if database is temporarily unreachable

Scenario 4: Backup and Recovery

Test backup and restore procedures:

- Verify automated backups in S3
- Perform manual backup
- Delete application files from one EC2 instance
- Restore from S3 backup
- Verify application functionality after restore

Scenario 5: DNS Resolution

Test DNS configuration:

- Access application using custom domain name
- Verify DNS propagation using dig/nslookup

- Test from multiple locations/networks
- Confirm SSL/TLS if configured